

The Impossibility of Strategyproof Rank Aggregation

Manuel Eberl and Patrick Lederer

February 6, 2026

In Social Choice Theory, a *social welfare function* (SWF) is a function that takes a collection of individual preferences on some set of alternatives and returns an aggregated preference relation.

More formally: Consider finite sets of agents $N = \{1, \dots, n\}$ and alternatives $A = \{x_1, \dots, x_m\}$. The input of an SWF is an n -tuple of rankings (i.e. linear orders) of A , and its output is a ranking of A as well.

Various desirable properties on SWFs can be defined:

- Anonymity: The SWF is invariant under permutation of the agents.
- Unanimity: If all voters prefer x over y , then x is preferred over y in the output ranking as well.
- Majority consistency: If there exists a ranking x_1, \dots, x_m such that for every $i < j$, the alternative x_i is preferred over x_j by more than half of the agents, that ranking must be returned.
- Kemeny strategyproofness: Strategic voting is not possible for a single agent, i.e. no agent can achieve a result more aligned with their own preferences by lying about them.

This entry contains two impossibility results for SWFs with m alternatives and n agents:

- There exists no anonymous, unanimous, and Kemeny-strategyproof SWF for $m \geq 5$ and n even or for $m = 4$ and n a multiple of 4.
- There exists no majority-consistent and Kemeny-strategyproof SWF for $m = 4$ and $n \geq 3$ or $m \geq 4$ and $n \in \{9, 11, 13, 15\} \cup \{17, \dots\}$

For some of the base cases, SAT solving is used by letting specialised automation prove a large number of clauses, translating to the DIMACS format, and importing a proof pre-generated by an external SAT solver using Lammich's GRAT format.

Contents

1	Auxiliary Material	3
1.1	Miscellaneous	3
1.2	The Majority Relation	10
1.3	The lexicographic order on lists	17
1.4	Maximal and minimal elements	21
2	Social welfare functions	25
2.1	Preference profiles	25
2.2	Definition and desirable properties of SWFs	29
2.3	Majority consistency	32
2.4	Concrete classes of SWFs	33
2.4.1	Dictatorships	33
2.4.2	Fixed-result SWFs	33
2.5	Anonymised preference profiles	34
2.6	Social Welfare Functions with explicit lists of agents and alternatives	47
2.7	Lowering constructions for SWFs	54
2.7.1	Decreasing the number of alternatives	55
2.7.2	Decreasing the number of agents by a factor	68
2.7.3	Decreasing the number of agents by an even number	76
3	Impossibility results	79
3.1	Infrastructure for SAT import and export	79
3.2	Automation for computing topological sortings	79
3.3	Automation for strategyproofness	83
3.4	Automation for majority consistency	84
3.5	For 5 alternatives and 2 agents	88
3.6	For 4 alternatives and 4 agents	93

1 Auxiliary Material

1.1 Miscellaneous

theory *SWF-Impossibility-Library*

imports

Randomised-Social-Choice.Preference-Profiles

HOL-Combinatorics.Multiset-Permutations

begin

lemma *wfp-on-iff-wfp*: $wfp\text{-on } A \ R \longleftrightarrow wfp (\lambda x \ y. \ R \ x \ y \wedge x \in A \wedge y \in A)$

proof –

have $wfp\text{-on } A \ R \longleftrightarrow wf\text{-on } A \ \{(x,y). \ R \ x \ y\}$

by (*simp add: wfp-on-def wf-on-def*)

also have $\dots = wf \ \{(x,y). \ R \ x \ y \wedge x \in A \wedge y \in A\}$

by (*subst wf-on-iff-wf simp-all*)

also have $\dots \longleftrightarrow wfp (\lambda x \ y. \ R \ x \ y \wedge x \in A \wedge y \in A)$

by (*simp add: wfp-def*)

finally show *?thesis* .

qed

lemma *permutations-of-set-conv-mset*:

$finite \ A \implies permutations\text{-of-set } A = \{xs. \ mset \ xs = mset\text{-set } A\}$

by (*metis permutations-of-multiset-def permutations-of-set-altdef*)

lemma *Set-filter-insert-if*:

$Set.filter \ P \ (insert \ x \ A) = (if \ P \ x \ then \ insert \ x \ (Set.filter \ P \ A) \ else \ Set.filter \ P \ A)$

by *auto*

lemma *Set-filter-insert*:

$P \ x \implies Set.filter \ P \ (insert \ x \ A) = insert \ x \ (Set.filter \ P \ A)$

$\neg P \ x \implies Set.filter \ P \ (insert \ x \ A) = Set.filter \ P \ A$

by *auto*

lemma *Set-filter-empty [simp]*: $Set.filter \ P \ \{\} = \{\}$

by *auto*

lemma *filter-mset-empty-conv*: $filter\text{-mset } P \ A = \{\#\} \longleftrightarrow (\forall x \in \#A. \ \neg P \ x)$

by (*induction A auto*)

lemma *image-mset-repeat-mset*: $image\text{-mset } f \ (repeat\text{-mset } n \ A) = repeat\text{-mset } n \ (image\text{-mset } f \ A)$

by (*induction A auto*)

lemma *filter-mset-repeat-mset*: $filter\text{-mset } P \ (repeat\text{-mset } n \ A) = repeat\text{-mset } n \ (filter\text{-mset } P \ A)$

by (*induction n auto*)

lemma *mset-eq-mset-set-iff*:

assumes *finite A*
shows $mset\ xs = mset\text{-}set\ A \longleftrightarrow xs \in permutations\text{-}of\text{-}set\ A$
using *assms unfolding permutations-of-set-def mem-Collect-eq*
by (*metis mset-set-set permutations-of-multisetI permutations-of-setD(1,2) permutations-of-set-altdef*)

lemma *size-Diff-mset-same-size:*

fixes $A\ B :: 'a\ multiset$
assumes $size\ (A - B) = n\ size\ A = size\ B$
shows $size\ (B - A) = n$
proof –
define E **where** $E = A - B$
define C **where** $C = B \cap\# A$
define D **where** $D = B - A$
have $B = C + D$
unfolding *C-def D-def* **by** (*simp add: inter-mset-def*)
have $A - B + B = E + B$
by (*simp add: E-def*)
also have $A - B + B = A + D$ **unfolding** *D-def*
by (*metis add.commute subset-mset.inf.commute union-diff-inter-eq-sup union-mset-def*)
finally have $size\ (A + D) = size\ (E + B)$
by (*rule arg-cong*)
hence $size\ A + size\ D = size\ B + size\ E$
by *simp*
also have $size\ A = size\ B$
by *fact*
finally have $size\ D = size\ E$
by *simp*
thus *?thesis* **using** *assms*
by (*simp add: D-def E-def*)

qed

lemma *image-mset-diff-if-inj-on:*

fixes $f :: 'a \Rightarrow 'b$
assumes *inj-on f (set-mset (A+B))*
shows $image\text{-}mset\ f\ (A - B) = image\text{-}mset\ f\ A - image\text{-}mset\ f\ B$
using *assms*
proof (*induction B arbitrary: A*)
case (*add x B A*)
show *?case*
proof (*cases x ∈# A*)
case *False*
hence $f\ x \notin\# image\text{-}mset\ f\ A$
using *add.prem* **by** *auto*
have $image\text{-}mset\ f\ (A - add\text{-}mset\ x\ B) = image\text{-}mset\ f\ (A - B)$
using *False* **by** *simp*
also have $\dots = image\text{-}mset\ f\ A - image\text{-}mset\ f\ B$
by (*rule add.IH*) (*use add.prem in auto*)
also have $\dots = image\text{-}mset\ f\ A - image\text{-}mset\ f\ (add\text{-}mset\ x\ B)$
using $\langle f\ x \notin\# image\text{-}mset\ f\ A \rangle$ **by** *simp*

```

    finally show ?thesis .
next
case True
define A' where A' = A - {#x#}
have A-eq: A = A' + {#x#}
  using True by (simp add: A'-def)
have image-mset f (A - add-mset x B) = image-mset f (A' - B)
  by (simp add: A-eq)
also have ... = image-mset f A' - image-mset f B
  by (rule add.IH) (use add.prem1 in ⟨auto simp: A-eq⟩)
also have ... = image-mset f A - image-mset f (add-mset x B)
  by (simp add: A-eq)
finally show ?thesis .
qed
qed auto

```

```

context preorder-on
begin

sublocale dual: preorder-on carrier  $\lambda x y. le y x$ 
  by standard (use not-outside refl in ⟨auto intro: trans⟩)

end

```

```

context order-on
begin

sublocale dual: order-on carrier  $\lambda x y. le y x$ 
  by standard (use antisymmetric in auto)

end

```

```

context total-preorder-on
begin

sublocale dual: total-preorder-on carrier  $\lambda x y. le y x$ 
  by standard (use total in auto)

end

```

```

context linorder-on
begin

```

sublocale *dual*: *linorder-on carrier* $\lambda x y. le\ y\ x$
 by *standard* (use **total in auto**)

end

context *finite-linorder-on*
begin

sublocale *dual*: *finite-linorder-on carrier* $\lambda x y. le\ y\ x$
 by *standard auto*

end

locale *linorder-family* = *preorder-family dom carrier R* **for** *dom carrier R* +
assumes *linorder-in-dom* [*simp*]: $i \in dom \implies linorder-on\ carrier\ (R\ i)$

lemma *preorder-familyI* [*intro?*]:
fixes *dom*
assumes $dom \neq \{\}$
assumes $\bigwedge i. i \in dom \implies preorder-on\ carrier\ (R\ i)$
assumes $\bigwedge i\ x\ y. i \notin dom \implies \neg R\ i\ x\ y$
shows *preorder-family dom carrier R*
using *assms* **unfolding** *preorder-family-def* **by** *auto*

lemma *linorder-familyI* [*intro?*]:
fixes *dom*
assumes $dom \neq \{\}$
assumes $\bigwedge i. i \in dom \implies linorder-on\ carrier\ (R\ i)$
assumes $\bigwedge i\ x\ y. i \notin dom \implies \neg R\ i\ x\ y$
shows *linorder-family dom carrier R*
proof –
have *preorder-family dom carrier R*
by *rule* (use *assms in* $\langle auto\ simp: linorder-on-def\ total-preorder-on-def \rangle$)
thus *?thesis*
unfolding *linorder-family-def linorder-family-axioms-def*
using *assms* **by** *auto*
qed

context *order-on*
begin

lemma *order-on-restrict*:
order-on (*carrier* $\cap A$) (*restrict-relation A le*)
proof –
interpret *restrict*: *preorder-on carrier* $\cap A$ *restrict-relation A le*
by (*rule preorder-on-restrict*)

```

show ?thesis
  by standard (use antisymmetric in ‹auto simp: restrict-relation-def›)
qed

```

```

lemma order-on-restrict-subset:
   $A \subseteq \text{carrier} \implies \text{order-on } A \text{ (restrict-relation } A \text{ le)}$ 
  using order-on-restrict[of A] by (simp add: Int-absorb1)

```

end

```

context linorder-on
begin

```

```

lemma linorder-on-restrict:
  linorder-on (carrier  $\cap$  A) (restrict-relation A le)
proof –
  interpret restrict: order-on carrier  $\cap$  A restrict-relation A le
    by (rule order-on-restrict)
  show ?thesis
    by standard (use total in ‹auto simp: restrict-relation-def›)
qed

```

```

lemma linorder-on-restrict-subset:
   $A \subseteq \text{carrier} \implies \text{linorder-on } A \text{ (restrict-relation } A \text{ le)}$ 
  using linorder-on-restrict[of A] by (simp add: Int-absorb1)

```

end

```

lemma linorder-on-concat:
  assumes linorder-on A R linorder-on B S  $A \cap B = \{\}$ 
  shows linorder-on (A  $\cup$  B) ( $\lambda x y. \text{if } x \in A \text{ then } R \ x \ y \vee y \in B \text{ else } S \ x \ y$ )
proof –
  interpret R: linorder-on A R
    by fact
  interpret S: linorder-on B S
    by fact
  show ?thesis
  proof (unfold-locales, goal-cases)
    case (1 x y)
    thus ?case
      using R.not-outside S.not-outside by (auto split: if-splits)
  next
    case (2 x y)
    thus ?case
      using R.not-outside S.not-outside by (auto split: if-splits)
  next
    case (3 x)
    thus ?case

```

```

    by (auto simp: R.refl S.refl)
next
case (4 x y z)
thus ?case using assms(3) R.not-outside S.not-outside
  by (auto split: if-splits intro: R.trans S.trans)
next
case (5 x y)
thus ?case using assms(3) R.not-outside S.not-outside
  by (auto split: if-splits intro: R.antisymmetric S.antisymmetric)
next
case (6 x y)
thus ?case using R.total S.total
  by auto
qed
qed

lemma linorder-on-prepend:
  assumes linorder-on A R z  $\notin$  A
  shows linorder-on (insert z A) ( $\lambda x y.$  if  $x = z$  then  $y \in \text{insert } z A$  else  $R x y$ )
proof -
  have *: linorder-on {z} ( $\lambda x y.$   $x = z \wedge y = z$ )
  by standard auto
  have linorder-on ({z}  $\cup$  A) ( $\lambda x y.$  if  $x \in \{z\}$  then  $x = z \wedge y = z \vee y \in A$  else  $R x y$ )
  by (rule linorder-on-concat) (use assms * in auto)
  also have ... = ( $\lambda x y.$  if  $x = z$  then  $y \in \text{insert } z A$  else  $R x y$ )
  by auto
  finally show ?thesis
  by simp
qed

lemma finite-linorder-on-exists:
  assumes finite A
  shows  $\exists R.$  linorder-on A R
  using assms
proof (induction rule: finite-induct)
  case empty
  have linorder-on ({ } :: 'a set) ( $\lambda - .$  False)
  by standard auto
  thus ?case by blast
next
  case (insert x A)
  from insert.IH obtain R where R: linorder-on A R
  by blast
  have linorder-on (insert x A) ( $\lambda y z.$  if  $y = x$  then  $z \in \text{insert } x A$  else  $R y z$ )
  by (rule linorder-on-prepend) fact+
  thus ?case
  by blast
qed

```

context *order-on*

begin

lemma *order-on-map*:

assumes *bij-betw f A carrier*

shows *order-on A (restrict-relation A (map-relation f le))*

proof –

have *preorder-on (f -‘ carrier) (map-relation f le)*

by (*rule preorder-on-map*)

hence *preorder-on (f -‘ carrier \cap A) (restrict-relation A (map-relation f le))*

by (*rule preorder-on.preorder-on-restrict*)

also have *f -‘ carrier \cap A = A*

using *assms* **by** (*auto simp: bij-betw-def*)

finally interpret *f: preorder-on A restrict-relation A (map-relation f le)* .

show *?thesis*

proof

fix *x y*

assume *restrict-relation A (map-relation f le) x y restrict-relation A (map-relation f le) y x*

hence *f x = f y x \in A y \in A* **using** *antisymmetric*

by (*auto simp: restrict-relation-def map-relation-def*)

thus *x = y*

using *assms* **by** (*auto simp: bij-betw-def inj-on-def*)

qed

qed

end

context *linorder-on*

begin

lemma *linorder-on-map*:

assumes *bij-betw f A carrier*

shows *linorder-on A (restrict-relation A (map-relation f le))*

proof –

interpret *order-on A restrict-relation A (map-relation f le)*

by (*rule order-on-map*) *fact*

have *total-preorder-on (f -‘ carrier) (map-relation f le)*

by (*rule total-preorder-on-map*)

hence *total-preorder-on (f -‘ carrier \cap A) (restrict-relation A (map-relation f le))*

by (*rule total-preorder-on.total-preorder-on-restrict*)

also have *f -‘ carrier \cap A = A*

using *assms* **by** (*auto simp: bij-betw-def*)

finally interpret *f: total-preorder-on A restrict-relation A (map-relation f le)* .

show ?thesis ..
qed

end

context *finite-linorder-on*
begin

lemma *finite-linorder-on-map*:
assumes *bij-betw f A carrier*
shows *finite-linorder-on A (restrict-relation A (map-relation f le))*
proof –
interpret *linorder-on A restrict-relation A (map-relation f le)*
by (*rule linorder-on-map*) *fact*
have [*simp*]: *finite A*
using *finite-carrier bij-betw-finite[OF assms]* **by** *simp*
show ?thesis
by *standard auto*
qed

end

1.2 The Majority Relation

Given a family of preorders, the majority relation induced by it is the one where x and y are related iff $x \preceq y$ holds in at least half of the relations in the family.

Note that the majority relation is in general neither antisymmetric (due to the possibility of ties) nor transitive (due to Condorcet cycles).

definition *majority* :: ('a \Rightarrow 'b relation) \Rightarrow 'b relation **where**
majority $R\ x\ y \longleftrightarrow (\exists i. R\ i\ x\ x) \wedge (\exists i. R\ i\ y\ y) \wedge \text{card } \{i. R\ i\ x\ y\} \geq \text{card } \{i. R\ i\ y\ x\}$

The same notion can easily be defined for multisets of relations as well.

definition *majority-mset* :: 'a relation multiset \Rightarrow 'a relation **where**
majority-mset $R\ s\ x\ y \longleftrightarrow$
 $(\exists R \in \#Rs. R\ x\ x) \wedge (\exists R \in \#Rs. R\ y\ y) \wedge$
 $\text{size } (\text{filter-mset } (\lambda R. R\ x\ y) Rs) \geq \text{size } (\text{filter-mset } (\lambda R. R\ y\ x) Rs)$

lemma *majority-mset-not-outside*:
assumes *majority-mset Rs x y* $\wedge R. R \in \# Rs \implies \text{preorder-on } A\ R$
shows $x \in A\ y \in A$
proof –
from *assms(1)* **obtain** $R1\ R2$ **where** $R1 \in \# Rs\ R2 \in \# Rs\ R1\ x\ x\ R2\ y\ y$
unfolding *majority-mset-def* **by** *blast*
thus $x \in A\ y \in A$
using *assms(2)* **by** (*meson preorder-on.not-outside(1)*)
qed

lemma *majority-mset-refl-iff'*: $\text{majority-mset } R s x x \longleftrightarrow (\exists R \in \# R s. R x x)$
unfolding *majority-mset-def* **by** *simp*

lemma *majority-mset-refl-iff*:
assumes $\bigwedge R. R \in \# R s \implies \text{preorder-on } A R R s \neq \{\#\}$
shows $\text{majority-mset } R s x x \longleftrightarrow x \in A$
unfolding *majority-mset-refl-iff'* **using** *assms*
by (*metis multiset-nonemptyE preorder-on.not-outside(1) preorder-on.refl*)

lemma *majority-mset-refl*:
assumes $\bigwedge R. R \in \# R s \implies \text{preorder-on } A R R s \neq \{\#\} x \in A$
shows $\text{majority-mset } R s x x$
using *majority-mset-refl-iff*[*OF assms(1,2)*] *assms(3)* **by** *simp*

lemma *majority-mset-iff'*:
assumes $\bigwedge R. R \in \# R s \implies \text{preorder-on } A R R s \neq \{\#\}$
shows $\text{majority-mset } R s x y \longleftrightarrow$
 $x \in A \wedge y \in A \wedge$
 $\text{size } (\text{filter-mset } (\lambda R. R x y) R s) \geq \text{size } (\text{filter-mset } (\lambda R. R y x) R s)$

proof –

obtain *R* **where** $R: R \in \# R s$
using $\langle R s \neq \{\#\} \rangle$ **by** *auto*
interpret *R*: *preorder-on* *A* *R*
using *assms(1)* *R* **by** *auto*
have $*$: $R x x \longleftrightarrow x \in A$ **if** $R \in \# R s$ **for** *R* *x*
using *assms(1)*[*OF that*] *preorder-on.refl* *preorder-on.not-outside(1)* **by** *metis*
show *?thesis* **using** *R*
unfolding *majority-mset-def* **by** (*auto simp: **)

qed

lemma *majority-mset-iff*:
assumes $\bigwedge R. R \in \# R s \implies \text{preorder-on } A R R s \neq \{\#\} x \in A y \in A$
shows $\text{majority-mset } R s x y \longleftrightarrow$
 $\text{size } (\text{filter-mset } (\lambda R. R x y) R s) \geq \text{size } (\text{filter-mset } (\lambda R. R y x) R s)$
by (*subst majority-mset-iff'[of Rs A]*) (*use assms in auto*)

lemma *majority-mset-iff-ge*:
assumes $\bigwedge R. R \in \# R s \implies \text{linorder-on } A R R s \neq \{\#\} x \in A y \in A$
shows $\text{majority-mset } R s x y \longleftrightarrow$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R x y) R s) \geq \text{size } R s$

proof (*cases* $x = y$)

case *True*
have [*simp*]: $R y y$ **if** $R \in \# R s$ **for** *R*
using *assms(1)* $\langle y \in A \rangle$ **by** (*metis linorder-on-def that total-preorder-on.total*)
have $\text{majority-mset } R s y y$
using *assms* **by** (*metis linorder-on-def majority-mset-refl order-on-def*)
moreover **have** $\{\#R \in \# R s. R y y\# \} = \text{filter-mset } (\lambda -. \text{True}) R s$
by (*intro filter-mset-cong*) *auto*
ultimately **show** *?thesis* **using** *True*

by *simp*
 next
 case *False*
 have $R_s = \text{filter-mset } (\lambda R. R x y) R_s + \text{filter-mset } (\lambda R. \neg R x y) R_s$
 by (*rule multiset-partition*)
 also have $\text{size } \dots = \text{size } (\text{filter-mset } (\lambda R. R x y) R_s) + \text{size } (\text{filter-mset } (\lambda R. \neg R x y) R_s)$
 by (*rule size-union*)
 also have $\text{filter-mset } (\lambda R. \neg R x y) R_s = \text{filter-mset } (\lambda R. R y x) R_s$
 proof (*rule filter-mset-cong*)
 fix R assume $R \in\# R_s$
 then interpret R : *linorder-on A R* using *assms(1)* by *auto*
 show $\neg R x y \longleftrightarrow R y x$
 using $\langle x \neq y \rangle R.\text{antisymmetric } R.\text{total}$ *assms(3-4)* by *blast*
 qed *auto*
 finally have *eq*: $\text{size } R_s = \text{size } \{\#R \in\# R_s. R x y\# \} + \text{size } \{\#R \in\# R_s. R y x\# \} .$
 show *?thesis*
 proof (*subst majority-mset-iff[of R_s A]*)
 fix R assume $R \in\# R_s$
 then interpret *linorder-on A R* using *assms(1)* by *blast*
 show *preorder-on A R ..*
 qed (*use assms eq in auto*)
 qed

lemma *majority-mset-iff-le*:

assumes $\bigwedge R. R \in\# R_s \implies \text{linorder-on } A R R_s \neq \{\#\} x \in A y \in A x \neq y$
 shows $\text{majority-mset } R_s x y \longleftrightarrow$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R y x) R_s) \leq \text{size } R_s$

proof –

have $R_s = \text{filter-mset } (\lambda R. R x y) R_s + \text{filter-mset } (\lambda R. \neg R x y) R_s$
 by (*rule multiset-partition*)
 also have $\text{size } \dots = \text{size } (\text{filter-mset } (\lambda R. R x y) R_s) + \text{size } (\text{filter-mset } (\lambda R. \neg R x y) R_s)$
 by (*rule size-union*)
 also have $\text{filter-mset } (\lambda R. \neg R x y) R_s = \text{filter-mset } (\lambda R. R y x) R_s$
 proof (*rule filter-mset-cong*)
 fix R assume $R \in\# R_s$
 then interpret R : *linorder-on A R* using *assms(1)* by *auto*
 show $\neg R x y \longleftrightarrow R y x$
 using $\langle x \neq y \rangle R.\text{antisymmetric } R.\text{total}$ *assms(3-4)* by *blast*
 qed *auto*
 finally have *eq*: $\text{size } R_s = \text{size } \{\#R \in\# R_s. R x y\# \} + \text{size } \{\#R \in\# R_s. R y x\# \} .$
 show *?thesis*
 proof (*subst majority-mset-iff[of R_s A]*)
 fix R assume $R \in\# R_s$
 then interpret *linorder-on A R* using *assms(1)* by *blast*
 show *preorder-on A R ..*
 qed (*use assms eq in auto*)
 qed

lemma *strongly-preferred-majority-mset-iff-gt*:

```

assumes  $\bigwedge R. R \in \# Rs \implies \text{linorder-on } A R Rs \neq \{\#\} x \in A y \in A$ 
shows  $x \prec_{[\text{majority-mset } Rs]} y \longleftrightarrow x \neq y \wedge$ 
 $2 * \text{size } (\text{filter-mset } (\lambda R. R x y) Rs) > \text{size } Rs$ 
proof (cases  $x = y$ )
  case True
    show ?thesis using True
    by (auto simp: strongly-preferred-def)
  next
    case False
    have  $Rs = \text{filter-mset } (\lambda R. R x y) Rs + \text{filter-mset } (\lambda R. \neg R x y) Rs$ 
    by (rule multiset-partition)
    also have  $\text{size } \dots = \text{size } (\text{filter-mset } (\lambda R. R x y) Rs) + \text{size } (\text{filter-mset } (\lambda R. \neg R x y) Rs)$ 
    by (rule size-union)
    also have  $\text{filter-mset } (\lambda R. \neg R x y) Rs = \text{filter-mset } (\lambda R. R y x) Rs$ 
    proof (rule filter-mset-cong)
    fix  $R$  assume  $R \in \# Rs$ 
    then interpret  $R$ : linorder-on  $A R$  using assms(1) by auto
    show  $\neg R x y \longleftrightarrow R y x$ 
    using  $\langle x \neq y \rangle R.\text{antisymmetric } R.\text{total}$  assms(3-4) by blast
    qed auto
    finally have  $\text{eq: } \text{size } Rs = \text{size } \{\#R \in \# Rs. R x y\# \} + \text{size } \{\#R \in \# Rs. R y x\# \} .$ 
    show ?thesis unfolding strongly-preferred-def
    proof (subst (1 2) majority-mset-iff[of Rs A])
    fix  $R$  assume  $R \in \# Rs$ 
    then interpret linorder-on  $A R$  using assms(1) by blast
    show preorder-on  $A R ..$ 
    qed (use assms eq in auto)
  qed

```

```

lemma strongly-preferred-majority-mset-iff-lt:
  assumes  $\bigwedge R. R \in \# Rs \implies \text{linorder-on } A R Rs \neq \{\#\} x \in A y \in A$ 
  shows  $x \prec_{[\text{majority-mset } Rs]} y \longleftrightarrow$ 
 $2 * \text{size } (\text{filter-mset } (\lambda R. R y x) Rs) < \text{size } Rs$ 
proof (cases  $x = y$ )
  case True
    have [simp]:  $R y y$  if  $R \in \# Rs$  for  $R$ 
    using assms(1)  $\langle y \in A \rangle$  by (metis linorder-on-def that total-preorder-on.total)
    have  $\{\#R \in \# Rs. R y y\# \} = \text{filter-mset } (\lambda -. \text{True}) Rs$ 
    by (intro filter-mset-cong) auto
    thus ?thesis using True
    by (auto simp: strongly-preferred-def)
  next
    case False
    have  $*$ :  $\bigwedge R. R \in \# Rs \implies \text{preorder-on } A R$ 
    using assms(1) by (simp add: linorder-on-def order-on-def)
    have  $x \prec_{[\text{majority-mset } Rs]} y \longleftrightarrow \neg(y \preceq_{[\text{majority-mset } Rs]} x)$ 
    using False majority-mset-iff[OF * assms(2)] assms(3,4)
    by (auto simp: strongly-preferred-def)
    also have  $\dots \longleftrightarrow \text{size } Rs > 2 * \text{size } \{\#R \in \# Rs. R y x\# \}$ 

```

by (subst majority-mset-iff-ge[of Rs A]) (use assms in auto)
 finally show ?thesis .
 qed

context preorder-family
 begin

lemma majority-iff':

majority $R x y \longleftrightarrow x \in \text{carrier} \wedge y \in \text{carrier} \wedge \text{card } \{i \in \text{dom}. R i x y\} \geq \text{card } \{i \in \text{dom}. R i y x\}$

proof –

have *: $\{i. R i x y\} = \{i \in \text{dom}. R i x y\}$ for $x y$

using not-in-dom by blast

from nonempty-dom obtain i where $i \in \text{dom}$

by blast

then interpret Ri : preorder-on carrier $R i$

by simp

show ?thesis

using Ri .refl unfolding majority-def *

by (meson in-dom not-in-dom preorder-on.not-outside(1))

qed

lemma majority-iff:

assumes $x \in \text{carrier} y \in \text{carrier}$

shows majority $R x y \longleftrightarrow \text{card } \{i \in \text{dom}. R i x y\} \geq \text{card } \{i \in \text{dom}. R i y x\}$

using assms by (simp add: majority-iff')

lemma majority-refl [simp]: $x \in \text{carrier} \implies \text{majority } R x x$

by (auto simp: majority-iff)

lemma majority-refl-iff: majority $R x x \longleftrightarrow x \in \text{carrier}$

by (auto simp: majority-iff')

lemma majority-total: $x \in \text{carrier} \implies y \in \text{carrier} \implies \text{majority } R x y \vee \text{majority } R y x$

by (auto simp: majority-iff)

lemma strongly-preferred-majority-iff:

assumes $x \in \text{carrier} y \in \text{carrier}$

shows $x \prec[\text{majority } R] y \longleftrightarrow \text{card } \{i \in \text{dom}. R i x y\} > \text{card } \{i \in \text{dom}. R i y x\}$

unfolding strongly-preferred-def by (auto simp: majority-iff assms)

lemma majority-not-outside:

assumes majority $R x y$

shows $x \in \text{carrier} y \in \text{carrier}$

using assms in-dom not-in-dom preorder-on.not-outside unfolding majority-def by meson+

The majority relation chains with the unanimity relation.

lemma majority-Pareto1:

assumes Pareto $R x y$ majority $R y z$ finite dom

shows *majority R x z*
proof –
have *xyz: x ∈ carrier y ∈ carrier z ∈ carrier*
using *Pareto.not-outside assms majority-not-outside* **by** *auto*
have *card {i ∈ dom. R i z x} ≤ card {i ∈ dom. R i z y}*
by *(rule card-mono)*
(use assms(1,3) in-dom in ⟨auto simp: Pareto-iff intro: preorder-on.trans[OF in-dom]⟩)
also have *card {i ∈ dom. R i z y} ≤ card {i ∈ dom. R i y z}*
using *assms(2) xyz* **by** *(simp add: majority-iff)*
also have *... ≤ card {i ∈ dom. R i x z}*
by *(rule card-mono)*
(use assms(1,3) in-dom in ⟨auto simp: Pareto-iff intro: preorder-on.trans[OF in-dom]⟩)
finally show *?thesis*
using *assms(2) xyz* **by** *(simp add: majority-iff)*
qed

lemma *majority-Pareto2:*

assumes *majority R x y Pareto R y z finite dom*
shows *majority R x z*
proof –
have *xyz: x ∈ carrier y ∈ carrier z ∈ carrier*
using *Pareto.not-outside assms majority-not-outside* **by** *auto*
have *card {i ∈ dom. R i z x} ≤ card {i ∈ dom. R i y x}*
by *(rule card-mono)*
(use assms in-dom in ⟨auto simp: Pareto-iff intro: preorder-on.trans[OF in-dom]⟩)
also have *card {i ∈ dom. R i y x} ≤ card {i ∈ dom. R i x y}*
using *assms(1) xyz* **by** *(simp add: majority-iff)*
also have *... ≤ card {i ∈ dom. R i x z}*
by *(rule card-mono)*
(use assms in-dom in ⟨auto simp: Pareto-iff intro: preorder-on.trans[OF in-dom]⟩)
finally show *?thesis*
using *assms(2) xyz* **by** *(simp add: majority-iff)*
qed

lemma *majority-conv-majority-mset:*

assumes *finite dom*
shows *majority R = majority-mset (image-mset R (mset-set dom)) (is ?lhs = ?rhs)*
proof *(intro ext)*
fix *x y*
show *?lhs x y ↔ ?rhs x y*
unfolding *majority-iff'*
by *(subst majority-mset-iff'[where A = carrier])*
(use in-dom nonempty-dom
in *⟨auto simp del: in-dom simp: assms mset-set-empty-iff filter-mset-image-mset⟩)*
qed

end

context *linorder-family*
begin

lemma *majority-iff-ge-half*:

assumes $x \in \text{carrier } y \in \text{carrier } \text{finite dom}$
shows $\text{majority } R \ x \ y \longleftrightarrow 2 * \text{card } \{i \in \text{dom}. R \ i \ x \ y\} \geq \text{card dom}$

proof (*cases* $x = y$)

case [*simp*]: *True*

have $\{i \in \text{dom}. R \ i \ x \ y\} = \text{dom}$

using *assms preorder-on.refl[OF in-dom]* **by** *auto*

with *assms* **show** *?thesis*

by (*simp add: majority-iff*)

next

case *False*

have $\text{dom} = \{i \in \text{dom}. R \ i \ x \ y\} \cup \{i \in \text{dom}. \neg R \ i \ x \ y\}$

by *auto*

also have $\text{card } \dots = \text{card } \{i \in \text{dom}. R \ i \ x \ y\} + \text{card } \{i \in \text{dom}. \neg R \ i \ x \ y\}$

by (*rule card-Un-disjoint*) (*use* $\langle \text{finite dom} \rangle$ **in** *auto*)

also have $\{i \in \text{dom}. \neg R \ i \ x \ y\} = \{i \in \text{dom}. R \ i \ y \ x\}$

proof (*rule set-eqI, unfold mem-Collect-eq, intro conj-cong refl*)

fix i **assume** $i: i \in \text{dom}$

interpret Ri : *linorder-on carrier R i*

using i **by** *simp*

show $\neg R \ i \ x \ y \longleftrightarrow R \ i \ y \ x$

using $Ri.\text{total}[of \ x \ y]$ $Ri.\text{antisymmetric}[of \ x \ y]$ *assms* $\langle x \neq y \rangle$ **by** *blast*

qed

finally show *?thesis*

using *assms* **by** (*auto simp: majority-iff*)

qed

lemma *majority-iff-le-half*:

assumes $x \in \text{carrier } y \in \text{carrier } x \neq y \text{ finite dom}$

shows $\text{majority } R \ x \ y \longleftrightarrow 2 * \text{card } \{i \in \text{dom}. R \ i \ y \ x\} \leq \text{card dom}$

proof –

have $\text{dom} = \{i \in \text{dom}. R \ i \ x \ y\} \cup \{i \in \text{dom}. \neg R \ i \ x \ y\}$

by *auto*

also have $\text{card } \dots = \text{card } \{i \in \text{dom}. R \ i \ x \ y\} + \text{card } \{i \in \text{dom}. \neg R \ i \ x \ y\}$

by (*rule card-Un-disjoint*) (*use* $\langle \text{finite dom} \rangle$ **in** *auto*)

also have $\{i \in \text{dom}. \neg R \ i \ x \ y\} = \{i \in \text{dom}. R \ i \ y \ x\}$

proof (*rule set-eqI, unfold mem-Collect-eq, intro conj-cong refl*)

fix i **assume** $i: i \in \text{dom}$

interpret Ri : *linorder-on carrier R i*

using i **by** *simp*

show $\neg R \ i \ x \ y \longleftrightarrow R \ i \ y \ x$

using $Ri.\text{total}[of \ x \ y]$ $Ri.\text{antisymmetric}[of \ x \ y]$ *assms* $\langle x \neq y \rangle$ **by** *blast*

qed

finally show *?thesis*

using *assms* **by** (*auto simp: majority-iff*)

qed

For families of odd cardinality, the majority rule is always antisymmetric.

```

lemma odd-imp-majority-antisymmetric:
  assumes odd (card dom) majority R x y majority R y x
  shows  $x = y$ 
proof (rule ccontr)
  assume  $x \neq y$ 
  have [simp]: finite dom
    using assms(1) card-ge-0-finite odd-pos by blast
  have  $xy: x \in \text{carrier } y \in \text{carrier } \text{card } \{i \in \text{dom}. R i y x\} = \text{card } \{i \in \text{dom}. R i x y\}$ 
    using assms unfolding majority-iff' by auto
  have  $\text{dom} = \{i \in \text{dom}. R i x y\} \cup \{i \in \text{dom}. \neg R i x y\}$ 
    by auto
  also have  $\text{card } \dots = \text{card } \{i \in \text{dom}. R i x y\} + \text{card } \{i \in \text{dom}. \neg R i x y\}$ 
    by (rule card-Un-disjoint) auto
  also have  $\{i \in \text{dom}. \neg R i x y\} = \{i \in \text{dom}. R i y x\}$ 
proof (rule set-eqI, unfold mem-Collect-eq, intro conj-cong refl)
  fix  $i$  assume  $i: i \in \text{dom}$ 
  interpret  $Ri$ : linorder-on carrier R i
    using  $i$  by simp
  show  $\neg R i x y \longleftrightarrow R i y x$ 
    using  $Ri.total[of x y] Ri.antisymmetric[of x y] xy(1,2) \langle x \neq y \rangle$  by blast
qed
  also have  $\text{card } \dots = \text{card } \{i \in \text{dom}. R i x y\}$ 
    using  $xy(3)$  by simp
  finally have even (card dom)
    by simp
  with  $\langle \text{odd (card dom)} \rangle$  show False
    by simp
qed
end

```

1.3 The lexicographic order on lists

```

fun lexprod-list-aux :: 'a relation  $\Rightarrow$  'a list relation where
  lexprod-list-aux R [] ys  $\longleftrightarrow$  True
| lexprod-list-aux R (x # xs) []  $\longleftrightarrow$  False
| lexprod-list-aux R (x # xs) (y # ys)  $\longleftrightarrow$   $x \preceq[R] y \wedge (x \prec[R] y \vee \text{lexprod-list-aux } R \text{ xs } ys)$ 

```

```

lemma lexprod-list-aux-Nil-right-iff [simp]: lexprod-list-aux R xs []  $\longleftrightarrow$   $xs = []$ 
  by (cases xs) auto

```

```

lemma lexprod-list-aux-refl:  $(\forall x \in \text{set } xs. R x x) \Longrightarrow \text{lexprod-list-aux } R \text{ xs } xs$ 
  by (induction xs) auto

```

```

definition lexprod-list :: 'a relation  $\Rightarrow$  'a list relation where
  lexprod-list R = restrict-relation \{xs. \forall x \in \text{set } xs. R x x\} (lexprod-list-aux R)

```

```

definition lexprod-length-list :: nat  $\Rightarrow$  'a relation  $\Rightarrow$  'a list relation where

```

$lexprod\text{-}length\text{-}list\ n\ R = restrict\text{-}relation\ \{xs.\ length\ xs = n\}\ (lexprod\text{-}list\ R)$

context *preorder-on*
begin

lemma *lexprod-list-aux-trans*:

assumes *lexprod-list-aux le xs ys lexprod-list-aux le ys zs*

shows *lexprod-list-aux le xs zs*

using *assms*

proof (*induction xs arbitrary: ys zs*)

case (*Cons x xs ys zs*)

obtain *y ys'* **where** [*simp*]: *ys = y # ys'*

using *Cons.prem*s **by** (*cases ys*) *auto*

obtain *z zs'* **where** [*simp*]: *zs = z # zs'*

using *Cons.prem*s **by** (*cases zs*) *auto*

show *?case*

using *Cons.prem*s *Cons.IH*[*of ys' zs'*] *trans* **by** (*auto simp: strongly-preferred-def*)

qed *auto*

lemma *preorder-lexprod-list: preorder-on (lists carrier) (lexprod-list le)*

proof

show *lexprod-list le xs xs* **if** *xs ∈ lists carrier* **for** *xs*

proof –

have *lexprod-list-aux le xs xs*

using *that* **by** (*induction xs*) (*auto intro: refl*)

thus *?thesis*

using *that* **by** (*auto simp: lexprod-list-def restrict-relation-def refl*)

qed

next

show *lexprod-list le xs zs* **if** *lexprod-list le xs ys lexprod-list le ys zs* **for** *xs ys zs*

using *lexprod-list-aux-trans*[*of xs ys zs*] *that*

by (*auto simp: lexprod-list-def restrict-relation-def*)

next

show *xs ∈ lists carrier ys ∈ lists carrier*

if *lexprod-list le xs ys* **for** *xs ys*

proof –

have $\{xs, ys\} \subseteq \{xs.\ \forall x \in set\ xs.\ le\ x\ x\}$

using *that* **by** (*auto simp: lexprod-list-def restrict-relation-def*)

also have $\dots \subseteq lists\ carrier$

using *not-outside* **by** *blast*

finally show *xs ∈ lists carrier ys ∈ lists carrier*

by *blast+*

qed

qed

lemma *preorder-lexprod-length-list*:

preorder-on $\{xs.\ set\ xs \subseteq carrier \wedge length\ xs = n\}$ (*lexprod-length-list n le*)

```

proof –
  interpret lex: preorder-on lists carrier lexprod-list le
    by (rule preorder-lexprod-list)
  have preorder-on (lists carrier ∩ {xs. length xs = n}) (lexprod-length-list n le)
    unfolding lexprod-length-list-def by (rule lex.preorder-on-restrict)
  also have lists carrier ∩ {xs. length xs = n} = {xs. set xs ⊆ carrier ∧ length xs = n}
    by auto
  finally show ?thesis .
qed

end

```

```

context total-preorder-on
begin

```

```

lemma total-preorder-lexprod-list: total-preorder-on (lists carrier) (lexprod-list le)

```

```

proof –
  interpret lex: preorder-on lists carrier lexprod-list le
    by (rule preorder-lexprod-list)
  show ?thesis
  proof
    show lexprod-list le xs ys ∨ lexprod-list le ys xs
      if xs ∈ lists carrier ys ∈ lists carrier for xs ys
      proof –
        have lexprod-list-aux le xs ys ∨ lexprod-list-aux le ys xs using that total
          by (induction le xs ys rule: lexprod-list-aux.induct)
          (auto simp: strongly-preferred-def)
        thus ?thesis
          using that not-outside refl unfolding lexprod-list-def restrict-relation-def
          by blast
      qed
    qed
  qed

```

```

lemma total-preorder-lexprod-length-list:

```

```

  total-preorder-on {xs. set xs ⊆ carrier ∧ length xs = n} (lexprod-length-list n le)
proof –
  interpret lex: total-preorder-on lists carrier lexprod-list le
    by (rule total-preorder-lexprod-list)
  have total-preorder-on (lists carrier ∩ {xs. length xs = n}) (lexprod-length-list n le)
    unfolding lexprod-length-list-def by (rule lex.total-preorder-on-restrict)
  also have lists carrier ∩ {xs. length xs = n} = {xs. set xs ⊆ carrier ∧ length xs = n}
    by auto
  finally show ?thesis .
qed

end

```

context *order-on*

begin

lemma *order-lexprod-list: order-on (lists carrier) (lexprod-list le)*

proof –

interpret *lex: preorder-on lists carrier lexprod-list le*

by (*rule preorder-lexprod-list*)

show *?thesis*

proof

show $xs = ys$ **if** *lexprod-list le xs ys lexprod-list le ys xs* **for** $xs\ ys$

proof –

have *lexprod-list-aux le xs ys lexprod-list-aux le ys xs*

set xs \subseteq carrier set ys \subseteq carrier

using *that not-outside* **by** (*auto simp: lexprod-list-def restrict-relation-def*)

thus $xs = ys$ **using** *antisymmetric*

by (*induction le xs ys rule: lexprod-list-aux.induct*)

(*auto simp: strongly-preferred-def*)

qed

qed

qed

lemma *order-lexprod-length-list:*

order-on {xs. set xs \subseteq carrier \wedge length xs = n} (lexprod-length-list n le)

proof –

interpret *lex: order-on lists carrier lexprod-list le*

by (*rule order-lexprod-list*)

have *order-on (lists carrier \cap {xs. length xs = n}) (lexprod-length-list n le)*

unfolding *lexprod-length-list-def* **by** (*rule lex.order-on-restrict*)

also have *lists carrier \cap {xs. length xs = n} = {xs. set xs \subseteq carrier \wedge length xs = n}*

by *auto*

finally show *?thesis .*

qed

end

context *linorder-on*

begin

lemma *order-lexprod-list: linorder-on (lists carrier) (lexprod-list le)*

proof –

interpret *lex: order-on lists carrier lexprod-list le*

by (*rule order-lexprod-list*)

interpret *lex: total-preorder-on lists carrier lexprod-list le*

by (*rule total-preorder-lexprod-list*)

show *?thesis ..*

qed

lemma *linorder-lexprod-length-list*:

linorder-on $\{xs. \text{set } xs \subseteq \text{carrier} \wedge \text{length } xs = n\}$ (*lexprod-length-list* n *le*)

proof –

interpret *lex*: *linorder-on lists carrier lexprod-list le*

by (*rule order-lexprod-list*)

have *linorder-on* (*lists carrier* $\cap \{xs. \text{length } xs = n\}$) (*lexprod-length-list* n *le*)

unfolding *lexprod-length-list-def* **by** (*rule lex.linorder-on-restrict*)

also have *lists carrier* $\cap \{xs. \text{length } xs = n\} = \{xs. \text{set } xs \subseteq \text{carrier} \wedge \text{length } xs = n\}$

by *auto*

finally show *?thesis* .

qed

end

1.4 Maximal and minimal elements

definition *Min-wrt-among* :: 'a relation \Rightarrow 'a set \Rightarrow 'a set **where**

Min-wrt-among R $A = \{x \in A. R \ x \ x \wedge (\forall y \in A. R \ y \ x \longrightarrow R \ x \ y)\}$

lemma *Min-wrt-among-cong*:

assumes *restrict-relation* $A \ R = \text{restrict-relation } A \ R'$

shows *Min-wrt-among* $R \ A = \text{Min-wrt-among } R' \ A$

proof –

from *assms* **have** $R \ x \ y \longleftrightarrow R' \ x \ y$ **if** $x \in A \ y \in A$ **for** $x \ y$

using *that* **by** (*auto simp: restrict-relation-def fun-eq-iff*)

thus *?thesis* **unfolding** *Min-wrt-among-def* **by** *blast*

qed

definition *Min-wrt* :: 'a relation \Rightarrow 'a set **where**

Min-wrt $R = \text{Min-wrt-among } R \ \text{UNIV}$

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

unfolding *Min-wrt-def* *Min-wrt-among-def* **by** *simp*

lemma *Min-wrt-among-conv-Max-wrt-among*: *Min-wrt-among* $R \ A = \text{Max-wrt-among } (\lambda x \ y. R \ y \ x) \ A$

by (*simp add: Min-wrt-among-def Max-wrt-among-def*)

context *preorder-on*

begin

lemma *Min-wrt-among-preorder*:

Min-wrt-among $le \ A = \{x \in \text{carrier} \cap A. \forall y \in \text{carrier} \cap A. le \ y \ x \longrightarrow le \ x \ y\}$

unfolding *Min-wrt-among-def* **using** *not-outside refl* **by** *blast*

lemma *Min-wrt-preorder*:

$Min\text{-}wrt\ le = \{x \in carrier. \forall y \in carrier. le\ y\ x \longrightarrow le\ x\ y\}$
unfolding *Min-wrt-altdef* **using** *not-outside refl* **by** *blast*

lemma *Min-wrt-among-subset*:

Min-wrt-among $le\ A \subseteq carrier\ Min\text{-}wrt\ among\ le\ A \subseteq A$
unfolding *Min-wrt-among-preorder* **by** *auto*

lemma *Min-wrt-subset*:

Min-wrt $le \subseteq carrier$
unfolding *Min-wrt-preorder* **by** *auto*

lemma *Min-wrt-among-nonempty*:

assumes $B \cap carrier \neq \{\}$ *finite* $(B \cap carrier)$
shows *Min-wrt-among* $le\ B \neq \{\}$
by (*simp add: Min-wrt-among-conv-Max-wrt-among* *assms(1,2)* *dual.Max-wrt-among-nonempty*)

lemma *Min-wrt-nonempty*:

$carrier \neq \{\} \implies finite\ carrier \implies Min\text{-}wrt\ le \neq \{\}$
using *Min-wrt-among-nonempty[of UNIV]* **by** (*simp add: Min-wrt-def*)

lemma *Min-wrt-among-map-relation-vimage*:

$f - ' Min\text{-}wrt\ among\ le\ A \subseteq Min\text{-}wrt\ among\ (map\text{-}relation\ f\ le)\ (f - ' A)$
by (*auto simp: Min-wrt-among-def map-relation-def*)

lemma *Min-wrt-map-relation-vimage*:

$f - ' Min\text{-}wrt\ le \subseteq Min\text{-}wrt\ (map\text{-}relation\ f\ le)$
by (*auto simp: Min-wrt-altdef map-relation-def*)

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

assumes *bij* $(f :: 'a \Rightarrow 'b)$
shows $f - ' Min\text{-}wrt\ among\ le\ A \subseteq$
 $Min\text{-}wrt\ among\ (map\text{-}relation\ (inv\ f)\ le)\ (f - ' A)$
using *assms Min-wrt-among-map-relation-vimage[of inv f A]*
by (*simp add: bij-imp-bij-inv inv-inv-eq bij-vimage-eq-inv-image*)

lemma *Min-wrt-among-map-relation-bij*:

assumes *bij* f
shows $f - ' Min\text{-}wrt\ among\ le\ A = Min\text{-}wrt\ among\ (map\text{-}relation\ (inv\ f)\ le)\ (f - ' A)$

proof (*intro equalityI Min-wrt-among-map-relation-bij-subset* *assms*)

interpret R : *preorder-on* $f - ' carrier\ map\text{-}relation\ (inv\ f)\ le$

using *preorder-on-map[of inv f]* *assms*

by (*simp add: bij-imp-bij-inv bij-vimage-eq-inv-image inv-inv-eq*)

show $Min\text{-}wrt\ among\ (map\text{-}relation\ (inv\ f)\ le)\ (f - ' A) \subseteq f - ' Min\text{-}wrt\ among\ le\ A$

unfolding *Min-wrt-among-preorder* $R.Min\text{-}wrt\ among\ preorder$

using *assms bij-is-inj[OF assms]*

by (*auto simp: map-relation-def inv-f-f image-Int [symmetric]*)

qed

lemma *Min-wrt-map-relation-bij*:

$bij\ f \implies f \text{ ' } Min\text{-}wrt\ le = Min\text{-}wrt\ (map\text{-}relation\ (inv\ f)\ le)$
proof –
assume bij : $bij\ f$
interpret R : $preorder\text{-}on\ f \text{ ' } carrier\ map\text{-}relation\ (inv\ f)\ le$
using $preorder\text{-}on\ map[of\ inv\ f]\ bij$
by ($simp\ add$: $bij\text{-}imp\text{-}bij\text{-}inv\ bij\text{-}vimage\text{-}eq\text{-}inv\text{-}image\ inv\text{-}inv\text{-}eq$)
from bij **show** $?thesis$
unfolding $R.Min\text{-}wrt\text{-}preorder\ Min\text{-}wrt\text{-}preorder$
by ($auto\ simp$: $map\text{-}relation\text{-}def\ inv\text{-}f\text{-}f\ bij\text{-}is\text{-}inj$)
qed

lemma $Min\text{-}wrt\text{-}among\text{-}mono$:
 $le\ y\ x \implies x \in Min\text{-}wrt\text{-}among\ le\ A \implies y \in A \implies y \in Min\text{-}wrt\text{-}among\ le\ A$
using $not\text{-}outside$ **by** ($auto\ simp$: $Min\text{-}wrt\text{-}among\text{-}preorder\ intro$: $trans$)

lemma $Min\text{-}wrt\text{-}mono$:
 $le\ y\ x \implies x \in Min\text{-}wrt\ le \implies y \in Min\text{-}wrt\ le$
unfolding $Min\text{-}wrt\text{-}def$ **using** $Min\text{-}wrt\text{-}among\text{-}mono[of\ y\ x\ UNIV]$ **by** $blast$

end

context $total\text{-}preorder\text{-}on$
begin

lemma $Min\text{-}wrt\text{-}among\text{-}total\text{-}preorder$:
 $Min\text{-}wrt\text{-}among\ le\ A = \{x \in carrier \cap A. \forall y \in carrier \cap A. le\ x\ y\}$
unfolding $Min\text{-}wrt\text{-}among\text{-}preorder$ **using** $total$ **by** $blast$

lemma $Min\text{-}wrt\text{-}total\text{-}preorder$:
 $Min\text{-}wrt\ le = \{x \in carrier. \forall y \in carrier. le\ x\ y\}$
unfolding $Min\text{-}wrt\text{-}preorder$ **using** $total$ **by** $blast$

lemma $decompose\text{-}Min$:
assumes A : $A \subseteq carrier$
defines $M \equiv Min\text{-}wrt\text{-}among\ le\ A$
shows $restrict\text{-}relation\ A\ le = (\lambda x\ y. x \in M \wedge y \in A \vee (y \notin M \wedge restrict\text{-}relation\ (A - M)\ le\ x\ y))$
using A **by** ($intro\ ext$) ($auto\ simp$: $M\text{-}def\ Min\text{-}wrt\text{-}among\text{-}total\text{-}preorder\ restrict\text{-}relation\text{-}def\ Int\text{-}absorb1\ intro$: $trans$)

end

definition $min\text{-}wrt\text{-}among$:: $'a\ relation \Rightarrow 'a\ set \Rightarrow 'a$ **where**
 $min\text{-}wrt\text{-}among\ R\ A = the\text{-}elem\ (Min\text{-}wrt\text{-}among\ R\ A)$

definition $min\text{-}wrt$:: $'a\ relation \Rightarrow 'a$ **where**

$min-wrt R = min-wrt-among R UNIV$

definition $max-wrt-among :: 'a relation \Rightarrow 'a set \Rightarrow 'a$ **where**
 $max-wrt-among R A = the-elem (Max-wrt-among R A)$

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

context $finite-linorder-on$
begin

lemma $Max-wrt-among-singleton:$

assumes $A \neq \{\}$ $A \subseteq carrier$

shows $is-singleton (Max-wrt-among le A)$

proof –

have $x = y$ **if** $x \in Max-wrt-among le A$ $y \in Max-wrt-among le A$ **for** $x y$

using $antisymmetric[of x y]$ $total[of x y]$ **that** $assms$

unfolding $Max-wrt-among-def$ **by** $blast$

moreover **have** $Max-wrt-among le A \neq \{\}$

by $(rule Max-wrt-among-nonempty)$ $(use assms in auto)$

ultimately show $?thesis$

unfolding $is-singleton-def$ **by** $blast$

qed

lemma $max-wrt-among-inside:$

assumes $A \neq \{\}$ $A \subseteq carrier$

shows $max-wrt-among le A \in A$

proof –

have $max-wrt-among le A \in Max-wrt-among le A$

using $Max-wrt-among-singleton[OF assms]$

unfolding $is-singleton-def max-wrt-among-def$ **by** $force$

also have $\dots \subseteq A$

by $(auto simp: Max-wrt-among-def)$

finally show $?thesis$.

qed

lemma $le-max-wrt-among:$

assumes $y \in A$ $A \subseteq carrier$

shows $le y (max-wrt-among le A)$

proof –

have $A \neq \{\}$

using $assms$ **by** $auto$

have $max-wrt-among le A \in Max-wrt-among le A$

using $Max-wrt-among-singleton[OF \langle A \neq \{\} \rangle assms(2)]$

unfolding $is-singleton-def max-wrt-among-def$ **by** $force$

thus $?thesis$ **using** $\langle y \in A \rangle$

by $(metis assms(2) decompose-Max restrict-relation-def)$

qed

end

context *finite-linorder-on*
begin

lemma *Min-wrt-among-singleton*:
 assumes $A \neq \{\}$ $A \subseteq \text{carrier}$
 shows *is-singleton* (*Min-wrt-among* le A)
 using *assms* by (*metis* *Min-wrt-among-conv-Max-wrt-among* *dual.Max-wrt-among-singleton*)

lemma *min-wrt-among-inside*:
 assumes $A \neq \{\}$ $A \subseteq \text{carrier}$
 shows *min-wrt-among* le $A \in A$
 using *dual.max-wrt-among-inside*[*OF* *assms*]
 by (*simp* *add: max-wrt-among-def min-wrt-among-def* *Min-wrt-among-conv-Max-wrt-among*)

lemma *le-min-wrt-among*:
 assumes $y \in A$ $A \subseteq \text{carrier}$
 shows le (*min-wrt-among* le A) y
 using *dual.le-max-wrt-among*[*OF* *assms*]
 by (*simp* *add: max-wrt-among-def min-wrt-among-def* *Min-wrt-among-conv-Max-wrt-among*)

end

end

2 Social welfare functions

theory *Social-Welfare-Functions*
imports
 Swap-Distance.Swap-Distance
 Rankings.Topological-Sortings-Rankings
 Randomised-Social-Choice.Preference-Profiles
 SWF-Impossibility-Library
begin

2.1 Preference profiles

In the context of social welfare functions, a preference profile consists of a linear ordering (a *ranking*) of alternatives for each agent.

locale *pref-profile-linorder-wf* =
 fixes *agents* :: 'agent set and *alts* :: 'alt set and $R :: ('agent, 'alt)$ *pref-profile*
 assumes *nonempty-agents* [*simp*]: $\text{agents} \neq \{\}$ and *nonempty-alts* [*simp*]: $\text{alts} \neq \{\}$
 assumes *prefs-wf* [*simp*]: $i \in \text{agents} \implies \text{finite-linorder-on } \text{alts } (R \ i)$
 assumes *prefs-undefined* [*simp*]: $i \notin \text{agents} \implies \neg R \ i \ x \ y$
begin

lemma *finite-alt*s [simp]: *finite alt*s

proof –

from *nonempty-agents* **obtain** *i* **where** $i \in \text{agents}$ **by** *blast*

then interpret *finite-linorder-on alt*s *R i* **by** *simp*

show *?thesis* **by** (*rule finite-carrier*)

qed

lemma *prefs-wf'* [simp]:

$i \in \text{agents} \implies \text{linorder-on alt$ s (*R i*)

using *prefs-wf*[*of i*] **by** (*simp-all add: finite-linorder-on-def del: prefs-wf*)

lemma *not-outside*:

assumes $x \preceq[R\ i]\ y$

shows $i \in \text{agents}\ x \in \text{alt$ s $y \in \text{alt$ s

proof –

from *assms* **show** $i \in \text{agents}$ **by** (*cases i \in agents*) *auto*

then interpret *linorder-on alt*s *R i* **by** *simp*

from *assms* **show** $x \in \text{alt$ s $y \in \text{alt$ s **by** (*simp-all add: not-outside*)

qed

sublocale *linorder-family agents alt*s *R*

proof

fix *i* **assume** $i \in \text{agents}$

thus *linorder-on alt*s (*R i*)

by *simp*

qed *auto*

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

lemma *wf-update*:

assumes $i \in \text{agents}\ \text{linorder-on alt$ s *Ri'*

shows *pref-profile-linorder-wf agents alt*s (*R(i := Ri')*)

proof –

interpret *linorder-on alt*s *Ri'* **by** *fact*

from *finite-alt*s **have** *finite-linorder-on alt*s *Ri'* **by** *unfold-locales*

with *assms* **show** *?thesis*

by (*auto intro!: pref-profile-linorder-wf.intro split: if-splits*)

qed

lemma *wf-permute-agents*:

assumes σ *permutes agents*

shows *pref-profile-linorder-wf agents alt*s (*R \circ \sigma*)

unfolding *o-def* **using** *permutes-in-image[OF assms(1)]*

by (*intro pref-profile-linorder-wf.intro prefs-wf*) *simp-all*

lemma (*in* –) *pref-profile-eqI*:

assumes *pref-profile-linorder-wf agents alt*s *R1* *pref-profile-linorder-wf agents alt*s *R2*

assumes $\bigwedge x. x \in \text{agents} \implies R1\ x = R2\ x$

shows $R1 = R2$
proof
interpret $R1$: *pref-profile-linorder-wf agents alts* $R1$ **by fact**
interpret $R2$: *pref-profile-linorder-wf agents alts* $R2$ **by fact**
fix x **show** $R1\ x = R2\ x$
by (*cases* $x \in \text{agents}$; *intro ext*) (*simp-all add: assms(3)*)
qed

An obvious fact: if the number of agents is at most 2 and there are no ties then the majority relation coincides with the unanimity relation.

lemma *card-agents-le-2-imp-majority-eq-unanimity*:
assumes $\text{card agents} \leq 2$ **and** [*simp*]: *finite agents*
assumes *linorder-on alts* (*majority R*)
shows *majority R = Pareto R*
proof (*intro ext*)
fix $x\ y$
interpret *maj*: *linorder-on alts majority R* **by fact**
show *majority R* $x\ y = \text{Pareto R}\ x\ y$
proof (*cases* $x \in \text{alts} \wedge y \in \text{alts}$)
case xy : *True*
define d **where** $d = \text{card } \{i \in \text{agents}. R\ i\ x\ y\}$
have *neq*: $x \neq y$ **if** $d \neq \text{card agents}$
proof
assume $x = y$
hence $\{i \in \text{agents}. R\ i\ x\ y\} = \text{agents}$
using *preorder-on.refl[OF in-dom]* xy **by auto**
thus *False*
using that by (*simp add: d-def*)
qed

have $d = 0 \vee d = \text{card agents}$
proof (*rule ccontr*)
assume $\neg(d = 0 \vee d = \text{card agents})$
moreover have $d \leq \text{card agents}$
unfolding *d-def* **by** (*rule card-mono*) *auto*
ultimately have $d > 0\ d < \text{card agents}$
by *simp-all*
hence $d = 1\ \text{card agents} = 2$
using $\langle \text{card agents} \leq 2 \rangle$ **by** *linarith+*
have $x \neq y$
by (*rule neq*) (*use* $\langle d < \text{card agents} \rangle$ **in auto**)
have *majority R* $x\ y \wedge \text{majority R}\ y\ x$
using $\langle d = 1 \rangle\ \langle \text{card agents} = 2 \rangle\ \langle x \neq y \rangle$ *xy majority-iff-ge-half[of x y]*
majority-iff-le-half[of y x]
by (*simp add: d-def*)
thus *False*
using *maj.antisymmetric xy* $\langle x \neq y \rangle$ **by blast**
qed

```

thus ?thesis
proof
  assume  $d = 0$ 
  have  $\text{majority } R \ x \ y \longleftrightarrow 2 * d \geq \text{card agents}$ 
    unfolding  $d\text{-def}$  using  $xy$  by ( $\text{auto simp: majority-iff-ge-half}$ )
  with  $\langle d = 0 \rangle$  have  $\neg \text{majority } R \ x \ y$ 
    by  $\text{simp}$ 
  moreover from  $\langle d = 0 \rangle$  have  $\forall i \in \text{agents}. \neg R \ i \ x \ y$ 
    unfolding  $d\text{-def}$  by  $\text{simp}$ 
  hence  $\neg \text{Pareto } R \ x \ y$ 
    by ( $\text{auto simp: Pareto-iff}$ )
  ultimately show ?thesis
    by  $\text{simp}$ 
next
  assume  $d = \text{card agents}$ 
  have  $\text{majority } R \ x \ y \longleftrightarrow 2 * d \geq \text{card agents}$ 
    unfolding  $d\text{-def}$  using  $xy$  by ( $\text{subst majority-iff-ge-half}$ )  $\text{auto}$ 
  with  $\langle d = \text{card agents} \rangle$  have  $\text{majority } R \ x \ y$ 
    by  $\text{simp}$ 
  moreover have  $\{i \in \text{agents}. R \ i \ x \ y\} = \text{agents}$ 
    by ( $\text{rule card-subset-eq}$ ) ( $\text{use } \langle d = \text{card agents} \rangle$  in  $\langle \text{simp-all add: } d\text{-def} \rangle$ )
  hence  $\text{Pareto } R \ x \ y$ 
    by ( $\text{auto simp: Pareto-iff}$ )
  ultimately show ?thesis
    by  $\text{simp}$ 
qed
qed ( $\text{use Pareto.not-outside in } \langle \text{auto simp: majority-iff}' \rangle$ )
qed
end

```

An *election*, in our terminology, consists of a finite set of agents and a finite non-empty set of alternatives. It is this context in which we then consider all the set of possible preference profiles and SWFs.

```

locale  $\text{linorder-election} =$ 
  fixes  $\text{agents} :: \text{'agent set}$  and  $\text{alts} :: \text{'alt set}$ 
  assumes  $\text{finite-agents}$  [ $\text{simp}$ ,  $\text{intro}$ ]:  $\text{finite agents}$ 
  assumes  $\text{finite-alts}$  [ $\text{simp}$ ,  $\text{intro}$ ]:  $\text{finite alts}$ 
  assumes  $\text{nonempty-agents}$  [ $\text{simp}$ ]:  $\text{agents} \neq \{\}$ 
  assumes  $\text{nonempty-alts}$  [ $\text{simp}$ ]:  $\text{alts} \neq \{\}$ 
begin

abbreviation  $\text{is-pref-profile} \equiv \text{pref-profile-linorder-wf agents alts}$ 

lemma  $\text{finite-linorder-on-iff}'$  [ $\text{simp}$ ]:
   $\text{finite-linorder-on alts } R \longleftrightarrow \text{linorder-on alts } R$ 
  by ( $\text{simp add: finite-linorder-on-def finite-linorder-on-axioms-def}$ )

lemma  $\text{finite-pref-profiles}$  [ $\text{intro}$ ]:  $\text{finite } \{R. \text{is-pref-profile } R\}$ 

```

and card-pref-profiles: $\text{card } \{R. \text{ is-pref-profile } R\} = \text{fact } (\text{card alts}) \wedge \text{card agents}$
proof –
define $f :: ('agent \Rightarrow 'alt \text{ relation}) \Rightarrow 'agent \Rightarrow 'alt \text{ relation}$
where $f = (\lambda R i. \text{ if } i \in \text{agents} \text{ then } R \ i \ \text{else } (\lambda - . \text{ False}))$
define $g :: ('agent \Rightarrow 'alt \text{ relation}) \Rightarrow 'agent \Rightarrow 'alt \text{ relation}$
where $g = (\lambda R. \text{ restrict } R \ \text{agents})$
have $\text{bij}: \text{bij-betw } f \ (\text{agents} \rightarrow_E \{R. \text{ linorder-on alts } R\}) \ \{R. \text{ is-pref-profile } R\}$
by $(\text{rule } \text{bij-betwI}[\text{of } - \ - \ g])$
 $(\text{auto simp: } f\text{-def } g\text{-def } \text{pref-profile-linorder-wf-def } \text{fun-eq-iff})$
have $\text{finite } (\text{agents} \rightarrow_E \{R. \text{ linorder-on alts } R\})$
by $(\text{intro } \text{finite-PiE } \text{finite-linorders-on}) \ \text{auto}$
thus $\text{finite } \{R. \text{ is-pref-profile } R\}$
using $\text{bij-betw-finite}[\text{OF } \text{bij}]$ **by** simp
show $\text{card } \{R. \text{ is-pref-profile } R\} = \text{fact } (\text{card alts}) \wedge \text{card agents}$
using $\text{bij-betw-same-card}[\text{OF } \text{bij}]$ **by** $(\text{simp add: } \text{card-PiE } \text{card-linorders-on})$
qed

lemma $\text{pref-profile-exists}: \exists R. \text{ is-pref-profile } R$
proof –
have $\text{card } \{R. \text{ is-pref-profile } R\} > 0$
by $(\text{subst } \text{card-pref-profiles}) \ \text{auto}$
thus $?thesis$
by $(\text{simp add: } \text{card-gt-0-iff})$
qed

lemma $\text{pref-profile-wfI}'$ [*intro?*]:
 $(\bigwedge i. i \in \text{agents} \implies \text{linorder-on alts } (R \ i)) \implies$
 $(\bigwedge i. i \notin \text{agents} \implies R \ i = (\lambda - . \text{ False})) \implies \text{is-pref-profile } R$
by $(\text{simp add: } \text{pref-profile-linorder-wf-def } \text{finite-linorder-on-def } \text{finite-linorder-on-axioms-def})$

lemma $\text{is-pref-profile-update}$ [*simp,intro*]:
assumes $\text{is-pref-profile } R \ \text{linorder-on alts } R \ i' \ i \in \text{agents}$
shows $\text{is-pref-profile } (R(i := R i'))$
using assms **by** $(\text{auto intro!: } \text{pref-profile-linorder-wf.wf-update})$

lemma election [*simp,intro*]: $\text{linorder-election agents alts}$
by $(\text{rule } \text{linorder-election-axioms})$

end

2.2 Definition and desirable properties of SWFs

locale $\text{social-welfare-function} = \text{linorder-election agents alts}$
for $\text{agents} :: 'agent \ \text{set}$ **and** $\text{alts} :: 'alt \ \text{set} +$
fixes $\text{swf} :: ('agent, 'alt) \ \text{pref-profile} \Rightarrow 'alt \ \text{relation}$
assumes $\text{swf-wf}: \text{is-pref-profile } R \implies \text{linorder-on alts } (\text{swf } R)$
begin

lemma $\text{swf-wf}'$:

```

assumes is-pref-profile R
shows finite-linorder-on alts (swf R)
proof –
  interpret linorder-on alts swf R
    by (rule swf-wf) fact
  show ?thesis
    by standard auto
qed

end

```

```

lemma (in linorder-election) social-welfare-functionI [intro]:
  ( $\bigwedge R. \text{is-pref-profile } R \implies \text{linorder-on alts (swf R)} \implies \text{social-welfare-function agents alts swf}$ )
  unfolding social-welfare-function-def social-welfare-function-axioms-def
  using linorder-election-axioms
  by blast

```

Anonymity: the identities of the agents do not matter, i.e. the SWF is stable under renaming of the authors.

```

locale anonymous-swf = social-welfare-function agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +
  assumes anonymous:  $\pi$  permutes agents  $\implies$  is-pref-profile R  $\implies$  swf (R  $\circ$   $\pi$ ) = swf R

```

An obvious fact: if there is only one agent, any SWF is anonymous.

```

lemma (in social-welfare-function) one-agent-imp-anonymous:
  assumes card agents = 1
  shows anonymous-swf agents alts swf
proof
  fix  $\pi$  R assume  $\pi: \pi$  permutes agents and R: is-pref-profile R
  from  $\pi$  have  $\pi = id$ 
    by (metis assms card-1-singletonE permutes-sing)
  thus swf (R  $\circ$   $\pi$ ) = swf R
    by simp
qed

```

Neutrality: the identities of the alternatives does not matter, i.e. the SWF commutes with renaming the alternatives.

This is not a particularly interesting property since it clashes with anonymity whenever tie-breaking is required.

```

locale neutral-swf = social-welfare-function agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +
  assumes neutral:  $\sigma$  permutes alts  $\implies$  is-pref-profile R  $\implies$  swf (map-relation  $\sigma \circ R$ ) = map-relation  $\sigma$  (swf R)

```

Unanimity: any ordering of two alternatives that all agents agree on is also present in the result ranking.

```

locale unanimous-swf = social-welfare-function agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +

```

assumes *unanimous*: *is-pref-profile* $R \implies \forall i \in \text{agents}. x \succ [R\ i] y \implies x \succ [\text{swf } R] y$
begin

lemma *unanimous'*:

assumes *is-pref-profile* $R \forall i \in \text{agents}. x \succeq [R\ i] y$

shows $x \succeq [\text{swf } R] y$

using *assms*

by (*metis linorder-on-def order-on.antisymmetric order-on-def
 pref-profile-linorder-wf.not-outside(2) pref-profile-linorder-wf.prefs-wf'
 preorder-on.refl strongly-preferred-def swf-wf unanimous*)

A more convenient form of unanimity for computation: the SWF must return a ranking that is a topological sorting of the Pareto dominance relation.

In other words: we define the relation P as the intersection of all the preference relations of the agents. This relation is a partial order that captures everything the agents agree on. Due to unanimity, the result returned by the SWF must be a linear ordering that extends P , i.e. a topological sorting of P .

These topological sortings can be computed relatively easily using the standard algorithm, i.e. repeatedly picking a maximal element nondeterministically and putting it as the next element of the result ranking.

If the number of possible rankings is relatively small, this is more efficient than listing all $n!$ possible rankings and then weeding out the ones ruled out by unanimity.

lemma *unanimous-topo-sort-Pareto*:

assumes R : *is-pref-profile* R

shows $\text{swf } R \in \text{of-ranking 'topo-sorts alts (Pareto}(R))$

proof –

interpret R : *pref-profile-linorder-wf agents alts* R

by *fact*

have $\text{Pareto}(R) \leq \text{swf } R$

using R **unfolding** *le-fun-def R.Pareto-iff le-bool-def* **by** (*auto intro: unanimous'*)

moreover have *finite-linorder-on alts (swf R)*

using R **by** (*rule swf-wf'*)

ultimately have $\text{swf } R \in \{R'. \text{finite-linorder-on alts } R' \wedge \text{Pareto}(R) \leq R'\}$

by *simp*

also have *bij-betw of-ranking (topo-sorts alts (Pareto}(R))) \{R'. finite-linorder-on alts R' \wedge Pareto}(R) \leq R'\}*

by (*rule bij-betw-topo-sorts-linorders-on*) (*use R.Pareto.not-outside in auto*)

hence $\{R'. \text{finite-linorder-on alts } R' \wedge \text{Pareto}(R) \leq R'\} = \text{of-ranking 'topo-sorts alts (Pareto}(R))$

by (*simp add: bij-betw-def*)

finally show *?thesis .*

qed

end

Kemeny strategyproofness: no agent can achieve a better outcome for themselves by unilaterally submitting a preference ranking different from their true one. Here, “better” is defined by the swap distance (also known as the Kendall tau distance).

locale *kemeny-strategyproof-swf* = *social-welfare-function agents alts swf*
for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf* +
assumes *kemeny-strategyproof*:
is-pref-profile *R* \implies $i \in \text{agents} \implies \text{linorder-on alts } R' \implies$
swap-dist-relation (*R* *i*) (*swf* *R*) \leq *swap-dist-relation* (*R* *i*) (*swf* (*R*(*i* := *R*[^])))

2.3 Majority consistency

locale *majority-consistent-swf* = *social-welfare-function agents alts swf*
for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf* +
assumes *majority-consistent*:
is-pref-profile *R* \implies *linorder-on alts* (*majority* *R*) \implies *swf* *R* = *majority* *R*

locale *majcons-kstratproof-swf* =
majority-consistent-swf *agents alts swf* +
kemeny-strategyproof-swf *agents alts swf*
for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf*

A unanimous SWF with at most 2 agents is always majority-consistent (since the only way for a preference relation to have no ties is for it to be unanimous).

lemma (*in unanimous-swf*)
assumes *card agents* \leq 2
shows *majority-consistent-swf* *agents alts swf*

proof

fix *R* **assume** *R*: *is-pref-profile* *R*
assume *maj*: *linorder-on alts* (*majority* *R*)
interpret *R*: *pref-profile-linorder-wf* *agents alts R* **by** *fact*
interpret *maj*: *linorder-on alts majority R* **by** *fact*
interpret *S*: *linorder-on alts swf R* **by** (*rule swf-wf*) *fact*

have *eq*: *majority* *R* = *Pareto* *R*
by (*rule R.card-agents-le-2-imp-majority-eq-unanimity*) (*use* *assms maj in simp-all*)
have *Pareto-imp-swf*: *swf* *R* *x y* **if** *Pareto* *R* *x y* **for** *x y*
using *unanimous*'[*of R x y*] *R* **that** **by** (*auto simp: R.Pareto-iff*)

show *swf* *R* = *majority* *R*

proof (*intro ext*)

fix *x y*
show *swf* *R* *x y* = *majority* *R* *x y*
proof (*cases x* \in *alts* \wedge *y* \in *alts*)
case *True*
show *swf* *R* *x y* = *majority* *R* *x y*
using *eq unanimous*'[*OF R*] *Pareto-imp-swf maj.total S.antisymmetric True* **by** *metis*
qed (*use S.not-outside maj.not-outside in auto*)

qed
qed

For a non-unanimous SWF, Kemeny strategyproofness does not survive the addition of dummy alternatives. However, a weaker notion does, namely Kemeny strategyproofness

where only manipulations to profiles with a linear majority relation are forbidden.

```

locale majority-consistent-weak-kstratproof-swf =
  majority-consistent-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +
  assumes majority-consistent-kemeny-strategyproof:
    is-pref-profile R  $\implies$   $i \in$  agents  $\implies$  linorder-on alts S  $\implies$ 
    linorder-on alts (majority (R(i := S)))  $\implies$ 
    swap-dist-relation (R i) (swf R)  $\leq$  swap-dist-relation (R i) (majority (R(i := S)))

```

2.4 Concrete classes of SWFs

2.4.1 Dictatorships

A dictatorship rule simply returns the ranking of one fixed agent (the dictator). It is clearly neutral, anonymous, and strategyproof, but neither anonymous (unless $n = 1$) nor majority-consistent (unless $n \leq 2$).

```

locale dictatorship-swf = linorder-election agents alts
  for agents :: 'agent set and alts :: 'alt set +
  fixes dictator :: 'agent
  assumes dictator-in-agents: dictator  $\in$  agents
begin

  sublocale social-welfare-function agents alts  $\lambda R$ . R dictator
  proof
    fix R assume R: is-pref-profile R
    thus linorder-on alts (R dictator)
    by (simp add: dictator-in-agents pref-profile-linorder-wf.prefs-wf')
  qed

  sublocale neutral-swf agents alts  $\lambda R$ . R dictator
  by unfold-locales auto

  sublocale unanimous-swf agents alts  $\lambda R$ . R dictator
  by unfold-locales (use dictator-in-agents in auto)

  sublocale kemeny-strategyproof-swf agents alts  $\lambda R$ . R dictator
  by unfold-locales auto

end

```

2.4.2 Fixed-result SWFs

Another degenerate case is an SWF that always returns the same ranking, completely ignoring the preferences of the agents. Such an SWF is clearly anonymous and strategyproof, but not unanimous (except for the degenerate case where $m = 1$).

```

locale fixed-swf = linorder-election agents alts
  for agents :: 'agent set and alts :: 'alt set +

```

```

fixes ranking :: 'alt relation
assumes ranking: linorder-on alts ranking
begin

sublocale social-welfare-function agents alts λ-. ranking
  by standard (use ranking in auto)

sublocale anonymous-swf agents alts λ-. ranking
  by unfold-locales auto

sublocale kemeny-strategyproof-swf agents alts λ-. ranking
  by unfold-locales auto

end

end

```

2.5 Anonymised preference profiles

```

theory SWF-Anonymous
  imports Social-Welfare-Functions
begin

context anonymous-swf
begin

lemma anonymous':
  assumes R: is-pref-profile R and R': is-pref-profile R'
  assumes image-mset R (mset-set agents) = image-mset R' (mset-set agents)
  shows swf R = swf R'
proof –
  interpret R: pref-profile-linorder-wf agents alts R by fact
  interpret R': pref-profile-linorder-wf agents alts R' by fact
  obtain π where π: π permutes agents  $\forall x \in \text{agents}. R\ x = R' (\pi\ x)$ 
    by (rule image-mset-eq-implies-permutes[OF - assms(3)]) (use finite-agents in auto)
  from π(1) R' have swf (R' ◦ π) = swf R'
    by (rule anonymous)
  also have R' ◦ π = R
    using R'.not-outside(1) R.not-outside(1) π(2) permutes-in-image[OF π(1)]
    unfolding fun-eq-iff o-def by blast
  finally show ?thesis .
qed

```

For convenience we define a simpler view on SWFs where the input is not a regular preference profile but an “anonymised” profile. Formally, this is simply the multiset of the agents’ rankings without any information on the identities of the agents.

definition *is-apref-profile* :: 'alt relation multiset \Rightarrow bool **where**
is-apref-profile Rs \longleftrightarrow size Rs = card agents \wedge ($\forall R \in \#Rs. \text{linorder-on alts } R$)

The following is the corresponding version of the SWF that takes an anonymised profile:

definition *aswf* :: 'alt relation multiset \Rightarrow 'alt relation
where *aswf* *Rs* = *swf* (SOME *R*. *is-pref-profile* *R* \wedge *Rs* = *image-mset* *R* (*mset-set* *agents*))

Every valid anonymised profile also has at least one corresponding "non-anonymised" version.

lemma *deanonymised-profile-exists*:

assumes *is-apref-profile* *Rs*

obtains *R* **where** *is-pref-profile* *R* *Rs* = *image-mset* *R* (*mset-set* *agents*)

proof –

have *size-eq*: *size* (*mset-set* *agents*) = *size* *Rs*

using *assms* **by** (*simp* *add*: *is-apref-profile-def*)

obtain *agents'* **where** *agents'*: *distinct* *agents'* *set* *agents'* = *agents*

using *finite-distinct-list* **by** *blast*

obtain *Rs'* **where** *Rs'*: *mset* *Rs'* = *Rs*

using *ex-mset* **by** *blast*

have *length-Rs'*: *length* *Rs'* = *card* *agents*

using *Rs'* *size-eq* **by** *auto*

have *length-agents'*: *length* *agents'* = *card* *agents*

using *agents'*(1,2) *distinct-card* **by** *fastforce*

have *index-less*: *index* *agents'* *i* < *card* *agents* **if** *i* \in *agents* **for** *i*

using *that* **by** (*simp* *add*: *agents'*(2) *length-agents'*)

define *R* **where** *R* = (λi . *if* *i* \in *agents* *then* *Rs'* ! *index* *agents'* *i* *else* (λ - . *False*))

show *?thesis*

proof (*rule* *that*[*of* *R*])

show *is-pref-profile* *R*

proof

fix *i* **assume** *i*: *i* \in *agents*

hence *R* *i* = *Rs'* ! *index* *agents'* *i*

by (*auto* *simp*: *R-def*)

also **have** ... \in *set* *Rs'*

using *index-less*[*of* *i*] *i* *length-Rs'* **by** *auto*

also **have** ... = *set-mset* *Rs*

by (*simp* *flip*: *Rs'*)

finally **show** *linorder-on* *alts* (*R* *i*)

using *assms* **by** (*auto* *simp*: *is-apref-profile-def*)

qed (*auto* *simp*: *R-def*)

next

have *image-mset* *R* (*mset-set* *agents*) = *image-mset* (λi . *Rs'* ! *index* *agents'* *i*) (*mset-set* *agents*)

by (*intro* *image-mset-cong*) (*auto* *simp*: *R-def*)

also **have** *mset-set* *agents* = *mset* *agents'*

using *agents'* *mset-set-set* **by** *blast*

also **have** *image-mset* (λi . *Rs'* ! *index* *agents'* *i*) (*mset* *agents'*) =

mset (*map* (λi . *Rs'* ! *index* *agents'* *i*) *agents'*)

by *simp*

also **have** *map* (λi . *Rs'* ! *index* *agents'* *i*) *agents'* =

map ((λi . *Rs'* ! *index* *agents'* *i*) \circ (λi . *agents'* ! *i*)) [*0*..*length* *agents'*]

unfolding *map-map* [*symmetric*] **by** (*subst* *map-nth*) *simp-all*

```

also have ... = map ( $\lambda i. Rs' ! i$ ) [0.. $length\ Rs'$ ]
  using agents' by (intro map-cong) (auto simp: index-nth-id length-agents' length-Rs')
also have ... = Rs'
  by (rule map-nth)
also have mset agents' = mset-set agents
  using agents' mset-set-set by metis
also have mset Rs' = Rs
  using Rs' by simp
finally show Rs = image-mset R (mset-set agents) ..
qed
qed

```

The anonymous version of the SWF is well-defined w.r.t. the regular version of the SWF, i.e. plugging in the anonymised version of a profile gives the same result as plugging the original profile into the original SWF.

lemma *aswf-welldefined*:

```

assumes is-pref-profile R
defines Rs  $\equiv$  image-mset R (mset-set agents)
shows aswf Rs = swf R
proof –
interpret R: pref-profile-linorder-wf agents alts R
  by fact

```

```

define R' where R' = (SOME R. is-pref-profile R  $\wedge$  Rs = image-mset R (mset-set agents))
have *:  $\exists R. is-pref-profile R \wedge Rs = image-mset R (mset-set agents)$ 
  using assms by blast
have R': is-pref-profile R' Rs = image-mset R' (mset-set agents)
  using someI-ex[OF *] unfolding R'-def by blast+
interpret R': pref-profile-linorder-wf agents alts R'
  by fact

```

```

have aswf Rs = swf R'
  by (simp add: aswf-def R'-def)
also have swf R' = swf R
  by (rule anonymous') (use assms R' in simp-all)
finally show ?thesis .
qed

```

The anonymous version of the SWF always returns a valid ranking if the input is a valid anonymised profile.

lemma *aswf-wf*:

```

assumes is-apref-profile Rs
shows linorder-on alts (aswf Rs)
using assms by (metis aswf-welldefined deanonymised-profile-exists swf-wf)

```

lemma *aswf-wf'*:

```

assumes is-apref-profile Rs
shows finite-linorder-on alts (aswf Rs)
proof –

```

```

interpret linorder-on alts aswf Rs
  by (rule aswf-wf) fact
show ?thesis
  by standard auto
qed

```

For extra notational convenience, we define yet another version of our SWF that directly takes multisets of lists as inputs rather than multisets of preference relations.

```

definition aswf' :: 'alt list multiset  $\Rightarrow$  'alt list
  where aswf' Rs = ranking (aswf (image-mset of-ranking Rs))

```

```

definition is-apref-profile' :: 'alt list multiset  $\Rightarrow$  bool where
  is-apref-profile' Rs  $\longleftrightarrow$  size Rs = card agents  $\wedge$  ( $\forall R \in \#Rs. R \in$  permutations-of-set alts)

```

```

lemma is-apref-profile'-imp-is-apref-profile:
  assumes is-apref-profile' Rs
  shows is-apref-profile (image-mset of-ranking Rs)
  unfolding is-apref-profile-def
proof (intro ballI conjI)
  fix R assume R  $\in$  # image-mset of-ranking Rs
  then obtain xs where xs: xs  $\in$  # Rs R = of-ranking xs
  by auto
  hence xs': xs  $\in$  permutations-of-set alts
  using assms xs(1) by (auto simp: is-apref-profile'-def)
  show linorder-on alts R
  unfolding xs(2) by (rule linorder-of-ranking) (use xs' in <auto simp: permutations-of-set-def>)
qed (use assms in <auto simp: is-apref-profile'-def>)

```

```

lemma aswf'-wf:
  assumes is-apref-profile' Rs
  shows aswf' Rs  $\in$  permutations-of-set alts
proof –
  interpret linorder-on alts aswf (image-mset of-ranking Rs)
  by (rule aswf-wf) (use assms in <auto intro: is-apref-profile'-imp-is-apref-profile>)
  interpret finite-linorder-on alts aswf (image-mset of-ranking Rs)
  by unfold-locales auto
  show ?thesis
  unfolding aswf'-def permutations-of-set-def using distinct-ranking set-ranking by force
qed

```

end

```

locale anonymous-unanimous-swf =
  anonymous-swf agents alts swf +
  unanimous-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf
begin

```

lemma *unanimous-aswf*:
assumes *is-apref-profile* $R_s \forall R \in \#Rs. x \succ [R] y$
shows $x \succ [aswf\ R_s] y$
using *assms*
by (*metis aswf-welldefined deanonymised-profile-exists finite-agents finite-set-mset-mset-set image-eqI multiset.set-map unanimous*)

lemma *unanimous-aswf'*:
assumes *is-apref-profile* $R_s \forall R \in \#Rs. x \succeq [R] y$
shows $x \succeq [aswf\ R_s] y$
using *assms*
by (*metis aswf-welldefined deanonymised-profile-exists finite-agents finite-set-mset-mset-set image-eqI multiset.set-map unanimous*)

lemma *is-apref-profile-unanimous-not-outside*:
assumes *is-apref-profile* $R_s \forall R \in \#Rs. R\ x\ y$
shows $x \in alts \wedge y \in alts$

proof –
from *assms* **have** $R_s \neq \{\#\}$
by (*auto simp: is-apref-profile-def*)
then obtain R **where** $R: R \in \#Rs$
by *auto*
with *assms* **interpret** $R: linorder-on\ alts\ R$
by (*auto simp: is-apref-profile-def*)
from *assms* **have** $R\ x\ y$
using R **by** *auto*
with $R.not-outside$ **show** *?thesis*
by *blast*

qed

lemma *unanimous-topo-sorts-Pareto-aswf*:
assumes $R_s: is-apref-profile\ R_s$
shows $aswf\ R_s \in of-ranking\ 'topo-sorts\ alts\ (\lambda x\ y. \forall R \in \#Rs. R\ x\ y)$

proof –
obtain R **where** $R: is-pref-profile\ R\ R_s = image-mset\ R\ (mset-set\ agents)$
using $R_s\ deanonymised-profile-exists$ **by** *blast*
interpret $R: pref-profile-linorder-wf\ agents\ alts\ R$
by *fact*

have $aswf\ R_s = swf\ R$
using $R\ aswf-welldefined$ **by** *blast*
also have $swf\ R \in of-ranking\ 'topo-sorts\ alts\ (Pareto(R))$
by (*rule unanimous-topo-sort-Pareto*) *fact+*
also have $Pareto(R) = (\lambda x\ y. \forall R \in \#Rs. R\ x\ y)$
unfolding $R(2)$ **by** (*auto simp: R.Pareto-iff fun-eq-iff*)
finally show *?thesis* .

qed

end

locale *anonymous-kemeny-strategyproof-swf* =
 anonymous-swf agents alts swf +
 kemeny-strategyproof-swf agents alts swf
 for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf*
begin

lemma *kemeny-strategyproof-aswf*:

assumes *is-apref-profile R1 is-apref-profile R2*

assumes *size (R1 - R2) = 1*

assumes $\exists R \in \#(R1 - R2). \text{swap-dist-relation } R \ S1 > \text{swap-dist-relation } R \ S2$

shows *aswf R1 \neq S1 \vee aswf R2 \neq S2*

proof (rule *ccontr*)

assume $\neg(\text{aswf } R1 \neq S1 \vee \text{aswf } R2 \neq S2)$

hence *S12: aswf R1 = S1 aswf R2 = S2*

by *auto*

from *assms(1)* **obtain** *R1'* **where** *R1': is-pref-profile R1' R1 = image-mset R1' (mset-set agents)*

using *deanononymised-profile-exists* **by** *blast*

from *assms(2)* **obtain** *R2'* **where** *R2': is-pref-profile R2' R2 = image-mset R2' (mset-set agents)*

using *deanononymised-profile-exists* **by** *blast*

from $\langle \text{size } (R1 - R2) = 1 \rangle$ **obtain** *R* **where** *R: R1 - R2 = {#R#}*

using *size-1-singleton-mset[of R1 - R2]* **by** *auto*

have $R \in \# R1$

using *R* **by** (*metis in-diffD multi-member-last*)

obtain *i* **where** *i: i \in agents R = R1' i*

using *R* **unfolding** *R1'(2) R2'(2)*

by (*metis (no-types, lifting) Multiset.diff-right-commute add-mset-diff-bothsides diff-single-trivial finite-agents finite-set-mset-mset-set imageE multi-self-add-other-not-self multiset.set-map zero-diff*)

obtain *S* **where** *S: R2 - R1 = {#S#}*

proof -

have *size (R2 - R1) = 1*

by (rule *size-Diff-mset-same-size*)

 (*use assms in \langle auto simp: is-apref-profile-def \rangle*)

thus *?thesis*

using *that size-1-singleton-mset* **by** *blast*

qed

have $S \in \# R2$

using *S* **by** (*metis in-diffD multi-member-last*)

have *swap-dist-relation R S1 \leq swap-dist-relation R S2*

proof -

```

have swap-dist-relation (R1' i) (swf R1') ≤ swap-dist-relation (R1' i) (swf (R1'(i := S)))
proof (rule kemeny-strategyproof)
  from ⟨S ∈# R2⟩ and assms show linorder-on alts S
    by (auto simp: is-apref-profile-def)
qed fact+
also have swf R1' = aswf R1
  unfolding R1'(2) by (rule aswf-welldefined [symmetric]) fact
also have swf (R1'(i := S)) = aswf R2
proof -
  from ⟨S ∈# R2⟩ and assms have linorder-on alts S
    by (auto simp: is-apref-profile-def)
  hence is-pref-profile (R1'(i := S))
    by (intro is-pref-profile-update) (use R1'(1) i in auto)
  hence aswf (image-mset (R1'(i := S)) (mset-set agents)) = swf (R1'(i := S))
    by (rule aswf-welldefined)
  also have mset-set agents = add-mset i (mset-set agents - {#i#})
    using i by simp
  also have image-mset (R1'(i := S)) ... =
    {#S#} + image-mset (R1'(i := S)) (mset-set agents - {#i#})
    by simp
  also have mset-set agents - {#i#} = mset-set (agents - {i})
    by (subst mset-set-Diff) (use i in auto)
  also have image-mset (R1'(i := S)) (mset-set (agents - {i})) =
    image-mset R1' (mset-set (agents - {i}))
    by (intro image-mset-cong) auto
  also have image-mset R1' (mset-set (agents - {i})) = image-mset R1' (mset-set agents
- {#i#})
    by (subst mset-set-Diff) (use i in auto)
  also have ... = R1 - {#R#}
    by (subst image-mset-Diff) (use i in ⟨auto simp: R1'(2)⟩)
  also have {#S#} + (R1 - {#R#}) = R2
proof (rule multiset-eqI)
  fix T :: 'alt relation
  have count (R1 - R2) T = count {#R#} T
    by (subst R) auto
  moreover have count (R2 - R1) T = count {#S#} T
    by (subst S) auto
  ultimately show count ({#S#} + (R1 - {#R#})) T = count R2 T
    by auto
qed
finally show ?thesis ..
qed
also have aswf R1 = S1
  by fact
also have aswf R2 = S2
  by fact
also have R1' i = R
  using i by simp
finally show ?thesis .

```

qed

moreover have $\text{swap-dist-relation } R \ S1 > \text{swap-dist-relation } R \ S2$
using $\langle \exists R \in \#(R1 - R2). \text{swap-dist-relation } R \ S1 > \text{swap-dist-relation } R \ S2 \rangle$ unfolding R
by *simp*
ultimately show *False*
by *linarith*
qed

lemma *kemeny-strategyproof-aswf-strong*:

assumes *is-apref-profile* $R1$ *is-apref-profile* $R2$
assumes $\text{size } (R1 - R2) = 1$
assumes $(\exists R \in \#R1 - R2. \text{swap-dist-relation } R \ S1 > \text{swap-dist-relation } R \ S2) \vee$
 $(\exists R \in \#R2 - R1. \text{swap-dist-relation } R \ S2 > \text{swap-dist-relation } R \ S1)$
shows $\text{aswf } R1 \neq S1 \vee \text{aswf } R2 \neq S2$

proof –

have $sz: \text{size } (R2 - R1) = 1$
by (*rule size-Diff-mset-same-size*)
(*use assms in <auto simp: is-apref-profile-def>*)
show *?thesis*
using *kemeny-strategyproof-aswf*[*OF* *assms*(1–3), *of* $S2 \ S1$]
kemeny-strategyproof-aswf[*OF* *assms*(2,1) sz , *of* $S1 \ S2$] *assms*(4)
by *blast*

qed

lemma *kemeny-strategyproof-aswf'*:

assumes *is-apref-profile'* $R1$ *is-apref-profile'* $R2$
assumes $\text{size } (R1 - R2) = 1$
assumes $\exists R \in \#(R1 - R2). \text{swap-dist } R \ S1 > \text{swap-dist } R \ S2$
shows $\text{aswf}' \ R1 \neq S1 \vee \text{aswf}' \ R2 \neq S2$

proof (*rule ccontr*)

assume $\neg (\text{aswf}' \ R1 \neq S1 \vee \text{aswf}' \ R2 \neq S2)$
hence $S12: \text{aswf}' \ R1 = S1 \ \text{aswf}' \ R2 = S2$
by *blast+*
have $S12\text{-wf}: S1 \in \text{permutations-of-set alts } S2 \in \text{permutations-of-set alts}$
using $S12 \ \text{aswf}'\text{-wf } \text{assms}(1,2)$ by *blast+*
have *inj-on of-ranking* (*permutations-of-set alts*)
by (*metis inj-on-inverseI permutations-of-setD*(2) *ranking-of-ranking*)
hence *inj: inj-on of-ranking* (*set-mset* ($R1 + R2$))
by (*rule inj-on-subset*) (*use assms*(1,2) *in <auto simp: is-apref-profile'-def is-apref-profile-def>*)

have $\text{aswf} \ (\text{image-mset of-ranking } R1) \neq \text{of-ranking } S1 \vee \text{aswf} \ (\text{image-mset of-ranking } R2) \neq \text{of-ranking } S2$

proof (*rule kemeny-strategyproof-aswf*)

have $\text{image-mset of-ranking } R1 - \text{image-mset of-ranking } R2 = \text{image-mset of-ranking } (R1 - R2)$

using *inj* by (*rule image-mset-diff-if-inj-on* [*symmetric*])

also have $\text{size } \dots = 1$

using *assms* by *simp*

finally show $size (image\text{-}mset\ of\text{-}ranking\ R1 - image\text{-}mset\ of\text{-}ranking\ R2) = 1$.
next
from $assms(4)$ **obtain** R **where** $R: R \in\# R1 - R2$ $swap\text{-}dist\ R\ S1 > swap\text{-}dist\ R\ S2$
by *blast*
have $of\text{-}ranking\ R \in\# image\text{-}mset\ of\text{-}ranking\ (R1 - R2)$
using $R(1)$ **by** *simp*
also have $image\text{-}mset\ of\text{-}ranking\ (R1 - R2) = image\text{-}mset\ of\text{-}ranking\ R1 - image\text{-}mset\ of\text{-}ranking\ R2$
using *inj* **by** (rule *image-mset-diff-if-inj-on*)
finally have $R': of\text{-}ranking\ R \in\# image\text{-}mset\ of\text{-}ranking\ R1 - image\text{-}mset\ of\text{-}ranking\ R2$.

have $R \in\# R1$
using R **by** (*meson in-diffD*)
hence $R \in permutations\text{-}of\text{-}set\ alts$
using $R(1)$ $assms(1)$ **by** (*auto simp: is-apref-profile'-def*)
hence $swap\text{-}dist\text{-}relation\ (of\text{-}ranking\ R)\ (of\text{-}ranking\ S1) > swap\text{-}dist\text{-}relation\ (of\text{-}ranking\ R)\ (of\text{-}ranking\ S2)$
using $S12\text{-}wf\ R(2)$ **by** (*simp add: swap-dist-def permutations-of-set-def*)
with R' **show** $\exists R \in\# image\text{-}mset\ of\text{-}ranking\ R1 - image\text{-}mset\ of\text{-}ranking\ R2.$
 $swap\text{-}dist\text{-}relation\ R\ (of\text{-}ranking\ S2) < swap\text{-}dist\text{-}relation\ R\ (of\text{-}ranking\ S1)$
by *blast*
qed (*use assms in <auto intro!: is-apref-profile'-imp-is-apref-profile>*)

hence $aswf'\ R1 \neq S1 \vee aswf'\ R2 \neq S2$
unfolding $aswf'\text{-}def$
by (*metis aswf-wf' assms(1,2) finite-linorder-on.of-ranking-ranking is-apref-profile'-imp-is-apref-profile*)
with $S12$ **show** *False*
by *blast*
qed

lemma *kemeny-strategyproof-aswf'-strong:*
assumes *is-apref-profile' R1 is-apref-profile' R2*
assumes $size\ (R1 - R2) = 1$
assumes $(\exists R \in\#(R1 - R2). swap\text{-}dist\ R\ S1 > swap\text{-}dist\ R\ S2) \vee (\exists R \in\#(R2 - R1). swap\text{-}dist\ R\ S2 > swap\text{-}dist\ R\ S1)$
shows $aswf'\ R1 \neq S1 \vee aswf'\ R2 \neq S2$
proof –
have $sz: size\ (R2 - R1) = 1$
by (*rule size-Diff-mset-same-size*)
(*use assms in <auto simp: is-apref-profile'-def>*)
show *?thesis*
using *kemeny-strategyproof-aswf'[OF assms(1-3), of S2 S1]*
 $kemeny\text{-}strategyproof\text{-}aswf'\ [OF\ assms(2,1)\ sz,\ of\ S1\ S2]$ $assms(4)$
by *blast*
qed

A consequence of strategyproofness: if a profile contains clones (i.e. it contains the same ranking A multiple times) then simultaneous deviations by the clones may not result in

a better outcome w.r.t. A .

This is simply proven using a chain of n successive single-agent deviations, each replacing one copy of A with another ranking.

lemma *kemeny-strategyproof-aswf'-clones-aux*:
assumes *is-apref-profile' R1 is-apref-profile' R2*
assumes $R1 - R2 = \text{replicate-mset } n \ A$
shows $\text{swap-dist } A \ (\text{aswf}' \ R1) \leq \text{swap-dist } A \ (\text{aswf}' \ R2)$
using *assms*
proof (*induction n arbitrary: R1*)
case 0
hence $R1 - R2 = \{\#\}$
by *auto*
moreover have $\text{size } R1 = \text{size } R2$
using 0 **by** (*auto simp: is-apref-profile'-def*)
ultimately have $R1 = R2$
by (*metis Diff-eq-empty-iff-mset cancel-comm-monoid-add-class.diff-cancel nonempty-has-size size-Diff-submset subset-mset.add-diff-inverse*)
thus ?case using 0 **by** *auto*
next
case (*Suc n R1*)
have $A \in\# \ R1$
using *Suc.premis(3)* **by** (*metis in-diffD in-replicate-mset zero-less-Suc*)

define X **where** $X = R2 - R1$
have $\text{size } R1 = \text{size } R2$
using *Suc.premis* **by** (*auto simp: is-apref-profile'-def*)
have $\text{eq: } R1 + X = R2 + \text{replicate-mset } (\text{Suc } n) \ A$
using *Suc.premis(3)* **unfolding** $X\text{-def}$
by (*metis add.commute diff-intersect-left-idem diff-subset-eq-self inter-mset-def subset-mset.diff-add-assoc2 union-diff-inter-eq-sup union-mset-def*)

define $R0$ **where** $R0 = R1 - \text{replicate-mset } (\text{Suc } n) \ A$
have $R1\text{-eq: } R1 = R0 + \text{replicate-mset } (\text{Suc } n) \ A$
using *Suc.premis(3)* **unfolding** $R0\text{-def}$ **by** (*metis diff-subset-eq-self subset-mset.diff-add*)
have $R2\text{-eq: } R2 = R0 + X$
using eq **unfolding** $R1\text{-eq}$ **by** *simp*

have $\text{size } X = \text{Suc } n$
by (*metis <size R1 = size R2> add-diff-cancel-left' eq size-replicate-mset size-union*)
hence $X \neq \{\#\}$
by *auto*
then obtain B **where** $B: B \in\# \ X$
by *blast*
have $B': B \in\# \ R2$
using B **by** (*auto dest: in-diffD simp: X-def*)
define $R1'$ **where** $R1' = R1 - \{\#A\# \} + \{\#B\# \}$
have $R1': \text{is-apref-profile}' \ R1'$
using *Suc.premis(1,2)* $B \ \langle A \in\# \ R1 \rangle \ B'$

by (auto simp: is-apref-profile'-def R1'-def size-Suc-Diff1 dest: in-diffD)
 have $A \neq B$ using $B \text{ Suc.prem}(3)$
 unfolding $X\text{-def}$ by (metis in-diff-count in-replicate-mset not-less-iff-gr-or-eq zero-less-Suc)

have $\text{swap-dist } A \text{ (aswf' } R1) \leq \text{swap-dist } A \text{ (aswf' } R1')$
 proof –
 have $\text{diff-eq: } R1 - R1' = \{\#A\# \}$
 using $B \langle A \in\# R1 \rangle \langle A \neq B \rangle$ unfolding $R1'\text{-def}$
 by (metis Multiset.diff-add add-diff-cancel-left' diff-union-swap insert-DiffM2 zero-diff)
 show ?thesis
 by (cases $\text{swap-dist } A \text{ (aswf' } R1) \leq \text{swap-dist } A \text{ (aswf' } R1')$)
 (use kemeny-strategyproof-aswf'[of $R1 R1' \text{ aswf' } R1' \text{ aswf' } R1$] $\text{Suc.prem}(1) R1'$)
 in $\langle \text{simp-all add: diff-eq not-le} \rangle$

qed
 also have $\dots \leq \text{swap-dist } A \text{ (aswf' } R2)$
 proof (rule Suc.IH)
 have $R1' - R2 = \text{add-mset } B \text{ (replicate-mset } n \text{ } A) - X$
 by (simp add: $R1'\text{-def } R2\text{-eq } R1\text{-eq}$)
 also have $\dots = \text{replicate-mset } n \text{ } A - X$
 using $\langle A \neq B \rangle \langle B \in\# X \rangle$
 by (metis add-mset-diff-bothsides in-replicate-mset insert-DiffM minus-add-mset-if-not-in-lhs)
 also have $A \notin\# X$
 by (metis $\text{Suc.prem}(3) X\text{-def in-diff-count not-less-iff-gr-or-eq replicate-mset-Suc union-single-eq-member}$)
 hence $\text{replicate-mset } n \text{ } A - X = \text{replicate-mset } n \text{ } A$
 by (induction n) auto
 finally show $R1' - R2 = \text{replicate-mset } n \text{ } A$.
 qed fact+
 finally show ?case .

qed

lemma *kemeny-strategyproof-aswf'-clones:*
 assumes $\text{is-apref-profile' } R1 \text{ is-apref-profile' } R2$
 assumes $R1 - R2 = \text{replicate-mset } n \text{ } A$
 assumes $\text{swap-dist } A \text{ } S1 > \text{swap-dist } A \text{ } S2$
 shows $\text{aswf' } R1 \neq S1 \vee \text{aswf' } R2 \neq S2$
 using *kemeny-strategyproof-aswf'-clones-aux*[OF $\text{assms}(1-3)$] $\text{assms}(4)$ by auto

Another consequence of Kemeny strategyproofness: if an agent gets a non-optimal result (i.e. the result ranking is not the ranking of the agent), no deviation of the agent can yield the optimal result either.

lemma *kemeny-strategyproof-aswf'-no-obtain-optimal:*
 assumes $\text{is-apref-profile' } R \text{ is-apref-profile' } R' \text{ add-mset } S \text{ } R' = \text{add-mset } S' \text{ } R$
 shows $\text{aswf' } R = S \vee \text{aswf' } R' \neq S$
 proof (rule *ccontr*)
 assume $\neg(\text{aswf' } R = S \vee \text{aswf' } R' \neq S)$
 hence *: $\text{aswf' } R \neq S \text{ aswf' } R' = S$
 by auto
 have $S \neq S'$

```

using * assms( $\beta$ ) by auto

have aswf'  $R \neq \text{aswf}' R \vee \text{aswf}' R' \neq S$ 
proof (rule kemeny-strategyproof-aswf')
  show  $\text{size}(R - R') = 1$ 
    using assms( $\beta$ )  $\langle S \neq S' \rangle$  count-add-mset[of  $S R' S$ ] in-diff-count[of  $S R R'$ ]
      size-Suc-Diff1[of  $S R - R'$ ] by auto
next
  have  $S \in \# R - R'$ 
    using * assms( $\beta$ ) by (metis add-eq-conv-ex count-add-mset in-diff-count lessI)
  hence  $S \in \# R$ 
    by (meson in-diffD)
  hence  $S \in \text{permutations-of-set alts}$ 
    using assms(1) by (auto simp: is-apref-profile'-def)
  hence swap-dist  $S$  (aswf'  $R$ )  $> 0$ 
    by (subst swap-dist-pos-iff)
      (use * aswf'-wf[OF assms(1)] in  $\langle \text{auto simp: permutations-of-set-def} \rangle$ )
  with  $\langle S \in \# R - R' \rangle$  show  $\exists T \in \# R - R'. \text{swap-dist } T S < \text{swap-dist } T (\text{aswf}' R)$ 
    by (intro bexI[of -  $S$ ]) auto
qed fact+
with * show False
  by auto
qed
end

```

The following relation says that the given anonymised set of preferences R_s has a majority relation that is a linear order, and this linear order is exactly the one described by the ranking S .

```

definition majority-rel-mset :: 'a list multiset  $\Rightarrow$  'a list  $\Rightarrow$  bool where
  majority-rel-mset  $R_s S \iff$ 
    majority-mset (image-mset of-ranking  $R_s$ ) = of-ranking  $S \wedge \text{distinct } S$ 

```

```

locale anonymous-majority-consistent-swf =
  anonymous-swf agents alts swf +
  majority-consistent-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf
begin

```

```

lemma majority-consistent-aswf:
  assumes is-apref-profile  $R_s$  linorder-on alts (majority-mset  $R_s$ )
  shows aswf  $R_s = \text{majority-mset } R_s$ 
proof -
  obtain  $R$  where  $R$ : is-pref-profile  $R R_s = \text{image-mset } R$  (mset-set agents)
    using assms(1) deanonymised-profile-exists by blast
  interpret  $R$ : pref-profile-linorder-wf agents alts R by fact
  have maj-eq: majority  $R = \text{majority-mset } R_s$ 
    by (subst R.majority-conv-majority-mset) (use  $R$  in simp-all)

```

have $aswf\ Rs = swf\ R$
using $R\ aswf\ welldefined$ **by** $blast$
also have $\dots = majority\ R$
by $(rule\ majority\ consistent)\ (use\ assms(2)\ R(1)\ maj\ eq\ in\ simp\ all)$
also have $\dots = majority\ mset\ Rs$
by $fact$
finally show $aswf\ Rs = majority\ mset\ Rs$.
qed

lemma $majority\ consistent\ aswf'$:
assumes $is\ apref\ profile'\ Rs\ majority\ rel\ mset\ Rs\ S$
shows $aswf'\ Rs = S$
proof –
define $R s'$ **where** $R s' = image\ mset\ of\ ranking\ Rs$
define S' **where** $S' = of\ ranking\ S$
have $is\ apref\ profile\ Rs'$
using $assms(1)\ unfolding\ Rs'\ def$ **by** $(simp\ add:\ is\ apref\ profile'\ imp\ is\ apref\ profile)$
have $S' = majority\ mset\ Rs'$
using $assms(2)\ unfolding\ majority\ rel\ mset\ def$ **by** $(auto\ simp:\ Rs'\ def\ S'\ def)$
have $distinct\ S$
using $assms(2)$ **by** $(auto\ simp:\ majority\ rel\ mset\ def)$
have $linorder\ on\ alts\ S'$
proof –
have $R s'\ wf:\ \bigwedge R.\ R \in \# Rs' \implies preorder\ on\ alts\ R\ Rs' \neq \{\#\}$
using $\langle is\ apref\ profile\ Rs' \rangle\ unfolding\ is\ apref\ profile\ def$
using $linorder\ on\ def\ order\ on\ def$ **by** $fastforce+$
have $set\ S = alts$
proof $(rule\ set\ eqI)$
fix x
have $x \in set\ S \longleftrightarrow of\ ranking\ S\ x\ x$
by $(metis\ of\ ranking\ imp\ in\ set(2)\ of\ ranking\ refl)$
also have $\dots \longleftrightarrow majority\ mset\ Rs'\ x\ x$
using $assms(2)$ **by** $(simp\ add:\ majority\ rel\ mset\ def\ Rs'\ def)$
also have $\dots \longleftrightarrow x \in alts$
by $(rule\ majority\ mset\ refl\ iff)\ (use\ Rs'\ wf\ in\ auto)$
finally show $x \in set\ S \longleftrightarrow x \in alts$.
qed
thus $?thesis$
unfolding $S'\ def$ **using** $\langle distinct\ S \rangle$ **by** $(intro\ linorder\ of\ ranking)$
qed

have $aswf'\ Rs = ranking\ (aswf\ Rs')$
by $(simp\ add:\ aswf'\ def\ Rs'\ def)$
also have $aswf\ Rs' = majority\ mset\ Rs'$
by $(rule\ majority\ consistent\ aswf)$
 $(use\ \langle is\ apref\ profile\ Rs' \rangle\ \langle linorder\ on\ alts\ S' \rangle\ \langle S' = majority\ mset\ Rs' \rangle\ in\ simp\ all)$
also have $\dots = S'$
by $(rule\ sym)\ fact$

```

    also have ranking  $S' = S$ 
      using <distinct  $S$ > by (simp add:  $S'$ -def ranking-of-ranking)
    finally show ?thesis .
qed
end
end

```

2.6 Social Welfare Functions with explicit lists of agents and alternatives

```

theory SWF-Explicit
  imports SWF-Anonymous
begin

locale linorder-election-explicit =
  linorder-election agents alts
  for agents :: 'agent set and alts :: 'alt set +
  fixes agents-list :: 'agent list and alts-list :: 'alt list
  assumes agents-list: mset agents-list = mset-set agents
  assumes alts-list: mset alts-list = mset-set alts
begin

lemma distinct-alts-list: distinct alts-list
  using alts-list by (metis finite-alts mset-eq-mset-set-imp-distinct)

lemma alts-conv-alts-list: alts = set alts-list
  using alts-list by (metis finite-alts finite-set-mset-mset-set set-mset-mset)

lemma card-alts [simp]: card alts = length alts-list
  using alts-list by (metis size-mset size-mset-set)

lemma distinct-agents-list: distinct agents-list
  using agents-list by (metis finite-agents mset-eq-mset-set-imp-distinct)

lemma agents-conv-agents-list: agents = set agents-list
  using agents-list by (metis finite-agents finite-set-mset-mset-set set-mset-mset)

lemma card-agents: card agents = length agents-list
  using agents-list by (metis size-mset size-mset-set)

lemma mset-eq-alts-list-iff: mset  $xs$  = mset alts-list  $\longleftrightarrow$  distinct  $xs$   $\wedge$  set  $xs$  = alts
  by (metis alts-conv-alts-list card-alts card-distinct
    mset-set-set set-mset-mset size-mset)

lemma mset-eq-agents-list-iff: mset  $xs$  = mset agents-list  $\longleftrightarrow$  distinct  $xs$   $\wedge$  set  $xs$  = agents
  by (metis agents-conv-agents-list card-agents card-distinct
    mset-set-set set-mset-mset size-mset)

```

definition *prefs-from-rankings*

$:: 'alt\ list\ list \Rightarrow ('agent \Rightarrow 'alt\ relation)$ **where**

prefs-from-rankings *rs* =

$(\lambda i. \text{if } i \in \text{agents} \text{ then } \text{of-ranking } (rs\ !\ \text{index}\ \text{agents-list}\ i) \text{ else } (\lambda - .\ \text{False}))$

definition *prefs-from-rankings-wf* $:: 'alt\ list\ list \Rightarrow \text{bool}$ **where**

prefs-from-rankings-wf *rs* \longleftrightarrow

$\text{length } rs = \text{card } \text{agents} \wedge \text{list-all } (\lambda r. \text{mset } r = \text{mset } \text{alts-list})\ rs$

lemma *prefs-from-rankings-wf-imp-is-pref-profile* [intro]:

assumes *prefs-from-rankings-wf* *rs*

shows *is-pref-profile* (*prefs-from-rankings* *rs*)

proof

fix *i* **assume** *i*: $i \in \text{agents}$

hence $rs\ !\ \text{index}\ \text{agents-list}\ i \in \text{set } rs$

by (*intro nth-mem*)

(*use* *assms* **in** $\langle \text{auto simp: } \text{prefs-from-rankings-wf-def}\ \text{card-agents}\ \text{index-less-size-conv}\ \text{simp flip: } \text{agents-conv-agents-list} \rangle$)

hence $\text{distinct } (rs\ !\ \text{index}\ \text{agents-list}\ i) \wedge \text{set } (rs\ !\ \text{index}\ \text{agents-list}\ i) = \text{alts}$

using *assms* **unfolding** *prefs-from-rankings-wf-def* *list.pred-set* *mset-eq-alts-list-iff* **by** *blast*

thus *linorder-on* *alts* (*prefs-from-rankings* *rs* *i*)

using *assms* *i* **by** (*auto simp: prefs-from-rankings-def intro!: linorder-of-ranking*)

qed (*use* *assms* **in** $\langle \text{auto simp: } \text{prefs-from-rankings-def} \rangle$)

lemma *prefs-from-rankings-nth*:

assumes *prefs-from-rankings-wf* *R1* $i < \text{card } \text{agents}$

shows *prefs-from-rankings* *R1* (*agents-list* ! *i*) = *of-ranking* (*R1* ! *i*)

using *assms* *card-agents* *agents-conv-agents-list* *distinct-agents-list*

unfolding *prefs-from-rankings-def* **by** (*simp add: index-nth-id*)

lemma *prefs-from-rankings-outside*:

assumes $i \notin \text{agents}$

shows *prefs-from-rankings* *R1* *i* = $(\lambda - .\ \text{False})$

using *assms* **by** (*auto simp: prefs-from-rankings-def*)

lemma *prefs-from-rankings-update*:

assumes *prefs-from-rankings-wf* *R1* $i < \text{card } \text{agents}$ $\text{mset } xs = \text{mset } \text{alts-list}$

shows *prefs-from-rankings* (*R1*[*i* := *xs*]) =

$(\text{prefs-from-rankings } R1)(\text{agents-list}\ !\ i := \text{of-ranking } xs)$

using *assms* *distinct-agents-list* *card-agents* *agents-conv-agents-list* *index-less-size-conv*[*of* *agents-list*]

unfolding *prefs-from-rankings-def* *prefs-from-rankings-wf-def*

by (*auto simp: fun-eq-iff index-nth-id nth-list-update*)

lemma *prefs-from-rankings-wf-update*:

assumes *prefs-from-rankings-wf* *R1* $i < \text{card } \text{agents}$ $\text{mset } xs = \text{mset } \text{alts-list}$

shows *prefs-from-rankings-wf* (*R1*[*i* := *xs*])

using *assms* *set-update-subset-insert*[*of* *R1* *i* *xs*] **unfolding** *prefs-from-rankings-wf-def*

by (*auto simp: list.pred-set set-update-distinct*)

lemma *majority-prefs-from-rankings*:
assumes *prefs-from-rankings-wf* R
shows $\text{majority} (\text{prefs-from-rankings } R) = \text{majority-mset} (\text{mset} (\text{map of-ranking } R))$
proof –
interpret R : *pref-profile-linorder-wf agents alts prefs-from-rankings* R
using *assms* **by** *blast*
have $\text{majority} (\text{prefs-from-rankings } R) =$
 $\text{majority-mset} (\text{image-mset} (\text{prefs-from-rankings } R) (\text{mset-set agents}))$
by (*rule* R .*majority-conv-majority-mset*) *auto*
also have $\text{image-mset} (\text{prefs-from-rankings } R) (\text{mset-set agents}) =$
 $\text{image-mset} (\text{of-ranking} \circ (\lambda i. R ! i) \circ \text{index agents-list}) (\text{mset-set agents})$
by (*intro image-mset-cong*)
(use assms in ‹auto simp: prefs-from-rankings-wf-def prefs-from-rankings-def›)
also have $\dots = \text{image-mset of-ranking} (\text{image-mset} (\lambda i. R ! i) (\text{image-mset} (\text{index agents-list})$
 $(\text{mset agents-list})))$
by (*simp add: image-mset.compositionality o-def agents-list*)
also have $\text{image-mset} (\lambda i. R ! i) (\text{image-mset} (\text{index agents-list}) (\text{mset agents-list})) =$
 $\text{mset} (\text{map} (\lambda i. R ! i) (\text{map} (\text{index agents-list}) \text{agents-list}))$
unfolding *mset-map* **by** *simp*
also have $\text{map} (\text{index agents-list}) \text{agents-list} = [0..<\text{length } R]$
by (*subst map-index-self*)
(use distinct-agents-list card-agents assms in ‹simp-all add: prefs-from-rankings-wf-def›)
also have $\text{map} (\lambda i. R ! i) \dots = R$
by (*rule map-nth*)
finally show *?thesis* **by** *simp*
qed

lemma *majority-prefs-from-rankings-eq-of-ranking*:
assumes *prefs-from-rankings-wf* R *majority-rel-mset* ($\text{mset } R$) ys
shows $\text{majority} (\text{prefs-from-rankings } R) = \text{of-ranking } ys$
proof –
have $\text{of-ranking } ys = \text{majority-mset} (\text{image-mset of-ranking} (\text{mset } R))$
using *assms*(2) **by** (*auto simp: majority-rel-mset-def*)
also have $\dots = \text{majority} (\text{prefs-from-rankings } R)$
by (*subst majority-prefs-from-rankings*) (*use assms in simp-all*)
finally show *?thesis* ..
qed

lemma *majority-rel-mset-imp-mset*:
assumes *prefs-from-rankings-wf* R *majority-rel-mset* ($\text{mset } R$) xs
shows $\text{mset } xs = \text{mset alts-list}$
proof –
interpret R : *pref-profile-linorder-wf agents alts prefs-from-rankings* R
by (*rule* *prefs-from-rankings-wf-imp-is-pref-profile*) *fact*
have $\text{majority} (\text{prefs-from-rankings } R) = \text{of-ranking } xs$
by (*rule* *majority-prefs-from-rankings-eq-of-ranking*) *fact+*
thus *?thesis*
by (*metis* R .*majority-not-outside*(2) R .*majority-refl* *assms*(2) *majority-rel-mset-def*)

```

    mset-eq-alts-list-iff of-ranking-imp-in-set(2) of-ranking-refl
    order-antisym-conv subset-iff)
qed
end

locale social-welfare-function-explicit =
  social-welfare-function agents alts swf +
  linorder-election-explicit agents alts agents-list alts-list
  for agents :: 'agent set and alts :: 'alt set and swf agents-list alts-list
begin

definition swf' :: 'alt list list  $\Rightarrow$  'alt list where
  swf' R = ranking (swf (prefs-from-rankings R))

lemma swf'-wf: prefs-from-rankings-wf R  $\implies$  mset (swf' R) = mset-set alts
  unfolding swf'-def
  using finite-linorder-on.distinct-ranking finite-linorder-on.set-ranking alts-list finite-alts
  prefs-from-rankings-wf-imp-is-pref-profile mset-eq-alts-list-iff swf-wf' by metis

end

locale majority-consistent-swf-explicit =
  social-welfare-function-explicit agents alts swf agents-list alts-list +
  majority-consistent-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf agents-list alts-list
begin

lemma majority-consistent-swf':
  assumes prefs-from-rankings-wf R majority-rel-mset (mset R) ys
  shows swf' R = ys
  using assms
  by (metis linorder-of-ranking majority-consistent majority-prefs-from-rankings-eq-of-ranking
  majority-rel-mset-imp-mset mset-eq-alts-list-iff
  prefs-from-rankings-wf-imp-is-pref-profile ranking-of-ranking swf'-def)

end

locale majcons-kstratproof-swf-explicit =
  social-welfare-function-explicit agents alts swf agents-list alts-list +
  majcons-kstratproof-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf agents-list alts-list
begin

sublocale majority-consistent-swf-explicit ..

sublocale majority-consistent-weak-kstratproof-swf

```

by *unfold-locales*
 (*metis kemeny-strategyproof majority-consistent pref-profile-linorder-wf.wf-update*)

lemma *distinct-alts-list-aux: distinct alts-list*
 using *alts-list* by (*metis finite-alts mset-eq-mset-set-imp-distinct*)

lemma *distinct-agents-list-aux: distinct agents-list*
 using *agents-list* by (*metis finite-agents mset-eq-mset-set-imp-distinct*)

lemma *prefs-from-rankings-wf-iff:*
prefs-from-rankings-wf xss \longleftrightarrow
length xss = length agents-list \wedge list-all ($\lambda y s. mset ys = mset alts-list$) xss
 unfolding *prefs-from-rankings-wf-def* using *card-agents* by *simp*

lemma *swf'-in-all-rankings:*
 assumes *prefs-from-rankings-wf xss permutations-of-set-list alts-list = yss*
 shows *list-ex ($\lambda y s. swf' xss = ys$) yss*

proof –
 have *mset (swf' xss) = mset-set alts*
 by (*rule swf'-wf*) *fact*
 hence *swf' xss \in permutations-of-set alts*
 unfolding *permutations-of-set-def* using *alts-list mset-eq-alts-list-iff* by *force*
 also have *permutations-of-set alts = set yss*
 by (*metis alts-conv-alts-list distinct-alts-list assms(2)*
permutations-of-list remdups-id-iff-distinct)
 finally show *?thesis*
 unfolding *list-ex-iff* by *blast*

qed

lemma *kemeny-strategyproof-swf':*
 assumes *prefs-from-rankings-wf R1 i < card agents*
 assumes *mset zs = mset alts-list*
 assumes *xs = R1 ! i R2 = R1 [i := zs]*
 shows *swap-dist xs (swf' R1) \leq swap-dist xs (swf' R2)*

proof –
 define *R1'* where *R1' = prefs-from-rankings R1*
 define *j* where *j = agents-list ! i*
 have *j: j \in agents*
 unfolding *j-def* using *assms agents-conv-agents-list card-agents* by *force*
 have *zs: linorder-on alts (of-ranking zs)*
 using *assms* by (*intro linorder-of-ranking*) (*auto simp: mset-eq-alts-list-iff*)
 have *xs \in set R1*
 using *assms card-agents* by (*auto simp: prefs-from-rankings-wf-def*)
 hence *xs: mset xs = mset alts-list*
 using *assms* by (*auto simp: prefs-from-rankings-wf-def list.pred-set*)
 have *R2: prefs-from-rankings-wf R2*
 using *assms prefs-from-rankings-wf-update* by *blast*

have *swap-dist-relation (R1' j) (swf (R1'(j := of-ranking zs))) \geq swap-dist-relation (R1' j)*

(swf R1')
 by (rule kemeny-strategyproof) (use assms j zs in ⟨auto simp: R1'-def⟩)
 also have R1' j = of-ranking xs
 using assms prefs-from-rankings-nth unfolding R1'-def j-def by metis
 also have R1'(j := of-ranking zs) = prefs-from-rankings R2
 using assms unfolding R1'-def j-def using prefs-from-rankings-update by metis
 also have swf R1' = of-ranking (swf' R1)
 unfolding swf'-def R1'-def
 by (metis assms(1) finite-linorder-on.of-ranking-ranking
 prefs-from-rankings-wf-imp-is-pref-profile swf-wf')
 also have swf (prefs-from-rankings R2) = of-ranking (swf' R2)
 unfolding swf'-def
 by (metis assms(1,2,3,5) finite-linorder-on.of-ranking-ranking prefs-from-rankings-wf-update
 prefs-from-rankings-wf-imp-is-pref-profile swf-wf')
 also have swap-dist-relation (of-ranking xs) (of-ranking (swf' R1)) =
 swap-dist xs (swf' R1)
 using swf'-wf[of R1] alts-list assms(1) mset-eq-alts-list-iff xs
 unfolding swap-dist-def by auto
 also have swap-dist-relation (of-ranking xs) (of-ranking (swf' R2)) =
 swap-dist xs (swf' R2)
 using xs swf'-wf[of R2] alts-list R2 mset-eq-alts-list-iff
 unfolding swap-dist-def by auto
 finally show ?thesis .
 qed

lemma kemeny-strategyproof-swf'-aux:

assumes prefs-from-rankings-wf xss prefs-from-rankings-wf yss
 assumes map (index ys) S1 = S1' map (index ys) S2 = S2'
 assumes inversion-number S1' = d1 inversion-number S2' = d2
 assumes d1 > d2 ∧ i < length agents-list ∧ ys = xss ! i ∧ yss = xss[i := zs]
 shows swf' xss ≠ S1 ∨ swf' yss ≠ S2
 proof (rule ccontr)
 assume *: ¬(swf' xss ≠ S1 ∨ swf' yss ≠ S2)
 with assms(1,2) have S12: S1 ∈ permutations-of-set alts S2 ∈ permutations-of-set alts
 using swf'-wf by (auto simp: permutations-of-set-conv-mset)
 have ys ∈ set xss
 using assms card-agents by (auto simp: prefs-from-rankings-wf-def)
 hence ys: distinct ys set ys = alts
 using assms by (auto simp: prefs-from-rankings-wf-def list.pred-set mset-eq-alts-list-iff)
 have d12: swap-dist ys S1 = d1 ∧ swap-dist ys S2 = d2
 using assms(3-6) S12 ys
 by (subst (1 2) swap-dist-conv-inversion-number) (simp-all add: permutations-of-set-def)

 have zs ∈ set yss
 using assms card-agents unfolding prefs-from-rankings-wf-def by (metis set-update-memI)
 hence zs: mset zs = mset-set alts
 using assms(2) by (auto simp: prefs-from-rankings-wf-def list.pred-set alts-list)
 have swap-dist ys (swf' xss) ≤ swap-dist ys (swf' yss)
 by (rule kemeny-strategyproof-swf'[where i = i]) (use zs assms card-agents alts-list in auto)

```

with * d12 assms show False
  by simp
qed
end

locale majcons-weak-kstratproof-swf-explicit =
  social-welfare-function-explicit agents alts swf agents-list alts-list +
  majority-consistent-weak-kstratproof-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf agents-list alts-list
begin

sublocale majority-consistent-swf-explicit agents alts swf agents-list alts-list ..

lemma majority-consistent-kemeny-strategyproof-swf':
  assumes prefs-from-rankings-wf R1 i < card agents mset zs = mset alts-list
  assumes xs = R1 ! i majority-rel-mset (mset (R1[i := zs])) ys
  shows swap-dist xs (swf' R1) ≤ swap-dist xs ys
proof -
  define R2 where R2 = R1[i := zs]
  interpret res: finite-linorder-on alts swf (prefs-from-rankings R1)
  by (intro swf-wf' prefs-from-rankings-wf-imp-is-pref-profile assms)
  have R2-eq: prefs-from-rankings R2 = (prefs-from-rankings R1)(agents-list ! i := of-ranking
zs)
  unfolding <R2 = -> by (rule prefs-from-rankings-update) (use assms in auto)
  have R2-wf: prefs-from-rankings-wf R2
  unfolding <R2 = -> by (rule prefs-from-rankings-wf-update) (use assms in auto)
  interpret R2: pref-profile-linorder-wf agents alts prefs-from-rankings R2
  by (rule prefs-from-rankings-wf-imp-is-pref-profile) fact
  interpret res': finite-linorder-on alts swf (prefs-from-rankings R2)
  by (intro swf-wf' prefs-from-rankings-wf-imp-is-pref-profile R2-wf)

  have xs ∈ set R1
  using assms(1) <xs = R1 ! i> <i < ->
  unfolding prefs-from-rankings-wf-def by auto
  hence xs: distinct xs set xs = alts
  using assms(1) by (auto simp: prefs-from-rankings-wf-def list.pred-set mset-eq-alts-list-iff)
  have swf'-R1: mset (swf' R1) = mset alts-list
  using assms(1) by (simp add: swf'-wf alts-list)
  have swf'-R2: mset (swf' R2) = mset alts-list
  using R2-wf by (simp add: swf'-wf alts-list)

  have ys-eq: majority (prefs-from-rankings R2) = of-ranking ys
  by (rule majority-prefs-from-rankings-eq-of-ranking) (use assms R2-wf in <auto simp: R2-def>)
  have mset ys = mset alts-list
  by (rule majority-rel-mset-imp-mset) (use R2-wf <majority-rel-mset - -> in <auto simp:
R2-def>)
  have linorder-ys: linorder-on alts (of-ranking ys)

```

```

by (intro linorder-of-ranking) (use ⟨mset ys = -⟩ in ⟨auto simp: mset-eq-alts-list-iff⟩)

have swap-dist-relation (prefs-from-rankings R1 (agents-list ! i)) (swf (prefs-from-rankings
R1)) ≤
  swap-dist-relation (prefs-from-rankings R1 (agents-list ! i)) (majority (prefs-from-rankings
R2))
  unfolding R2-eq
proof (rule majority-consistent-kemeny-strategyproof)
  show is-pref-profile (prefs-from-rankings R1)
    using assms(1) by auto
  show agents-list ! i ∈ agents
    using ⟨i < -⟩ card-agents by (metis agents-list finite-agents finite-set-mset-mset-set
nth-mem-mset)
  show linorder-on alts (of-ranking zs)
    using ⟨mset zs = -⟩ alts-list finite-alts
  by (metis finite-set-mset-mset-set linorder-of-ranking mset-eq-mset-set-imp-distinct set-mset-mset)
  show linorder-on alts (majority ((prefs-from-rankings R1) (agents-list ! i := of-ranking zs)))
    unfolding R2-eq [symmetric] ys-eq by (rule linorder-ys)
qed
also have prefs-from-rankings R1 (agents-list ! i) = of-ranking (R1 ! i)
  by (rule prefs-from-rankings-nth) (use assms in auto)
also have R1 ! i = xs
  using assms by simp
also have swf (prefs-from-rankings R1) = of-ranking (ranking (swf (prefs-from-rankings R1)))
  by (simp add: res.of-ranking-ranking)
also have ... = of-ranking (swf' R1)
  by (simp add: swf'-def prefs-from-rankings-def)
also have swap-dist-relation (of-ranking xs) (of-ranking (swf' R1)) = swap-dist xs (swf' R1)
  unfolding swap-dist-def using xs swf'-R1 by (auto simp: mset-eq-alts-list-iff)
also have majority (prefs-from-rankings R2) = of-ranking ys
  by (rule ys-eq)
also have swap-dist-relation (of-ranking xs) ... = swap-dist xs ys
  unfolding swap-dist-def using xs swf'-R2 ⟨mset ys = -⟩ by (auto simp: mset-eq-alts-list-iff)
finally show ?thesis
  using assms by simp
qed

end

end

```

2.7 Lowering constructions for SWFs

```

theory SWF-Lowering
  imports SWF-Explicit
begin

```

In this section, we will give constructions that turn an SWF for some number of alternatives into an SWF for fewer alternatives and agents.

Concretely:

- We can create an SWF for fewer alternatives by simply adding the missing alternatives at the very end of all the agents' rankings in some fixed orders. However, this only works if the SWF is unanimous, so that the dummy alternatives are guaranteed to be at the very end of the output ranking.
- If the number of agents is $n = kn'$ for some $k > 0$, we can create an SWF for n' agents by simply cloning every agent in the input profile k times.

These constructions preserve anonymity, unanimity, and Kemeny-strategyproofness.

2.7.1 Decreasing the number of alternatives

locale *swf-restrict-alts* = *social-welfare-function agents alts swf*
for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf* +
fixes *dummy-alts alts'*
assumes *alts'-nonempty*: *alts' ≠ {}* **and** *finite-alts'*: *finite alts'*
assumes *dummy-alts-alts'*: *mset-set alts = mset dummy-alts + mset-set alts'*
begin

lemma *alts'*: *alts' ⊆ alts alts' ≠ {}*

proof –

show *alts' ⊆ alts* **using** *dummy-alts-alts'*

by (*metis finite-alts finite-alts' mset-subset-eq-add-right msubset-mset-set-iff*)

show *alts' ≠ {}*

by (*rule alts'-nonempty*)

qed

sublocale *new*: *linorder-election agents alts'*

by *standard* (*use alts' finite-subset[OF - finite-alts] in auto*)

lemma *dummy-alts*: *distinct dummy-alts set dummy-alts = alts – alts'*

proof –

show *distinct dummy-alts* **using** *dummy-alts-alts'*

by (*metis add-diff-cancel-right' alts'(1) finite-Diff*

finite-alts mset-eq-mset-set-iff mset-set-Diff permutations-of-setD(2))

show *set dummy-alts = alts – alts'*

by (*metis add-diff-cancel-right' alts'(1) dummy-alts-alts' finite-Diff2 finite-alts finite-alts'*

finite-set-mset-mset-set mset-set-Diff set-mset-mset)

qed

The following lifts a ranking on the smaller set of alternatives to the full set, by adding the dummy alternatives at the end in the order we fixed.

definition *extend-ranking* :: 'alt relation \Rightarrow 'alt relation **where**

extend-ranking $R =$

$(\lambda x y. R x y \vee \text{of-ranking dummy-alts } x y \vee x \in \text{alts} - \text{alts}' \wedge y \in \text{alts}')$

lemma *linorder-on-extend-ranking*:

assumes *linorder-on alts' R*

shows *linorder-on alts (extend-ranking R)*
proof –
interpret *R: linorder-on alts' R*
by *fact*
have *linorder-on ((alts – alts') ∪ alts')*
(λx y. if x ∈ alts – alts' then of-ranking dummy-alts x y ∨ y ∈ alts' else R x y)
proof (*rule linorder-on-concat*)
show *linorder-on (alts – alts') (of-ranking dummy-alts)*
by (*rule linorder-of-ranking*) (*use dummy-alts in auto*)
qed (*use assms in auto*)
also have *... = extend-ranking R*
using *R.not-outside of-ranking-imp-in-set[of dummy-alts] dummy-alts*
by (*auto simp: extend-ranking-def fun-eq-iff*)
also have *(alts – alts') ∪ alts' = alts*
using *alts' by auto*
finally show *?thesis .*
qed

lemma *restrict-extend-ranking:*
assumes *linorder-on alts' R*
shows *restrict-relation alts' (extend-ranking R) = R*
proof –
interpret *R: linorder-on alts' R*
by *fact*
show *?thesis*
using *alts' R.not-outside of-ranking-imp-in-set[of dummy-alts] dummy-alts*
unfolding *restrict-relation-def extend-ranking-def fun-eq-iff* **by** *auto*
qed

lemma *swap-dist-extend-ranking:*
assumes *linorder-on alts' R linorder-on alts' S*
shows *swap-dist-relation (extend-ranking R) (extend-ranking S) = swap-dist-relation R S*
proof –
interpret *R: linorder-on alts' R by fact*
have *swap-dist-relation-aux (extend-ranking R) (extend-ranking S) = swap-dist-relation-aux R S*
unfolding *swap-dist-relation-aux-def extend-ranking-def*
using *R.not-outside of-ranking-imp-in-set[of dummy-alts] dummy-alts by fast*
thus *?thesis*
by (*simp add: swap-dist-relation-def*)
qed

lemma *extend-ranking-eq-iff:*
assumes $\bigwedge x y. R x y \implies x \in \text{alts}' \wedge y \in \text{alts}' \wedge x y \implies x \in \text{alts}' \wedge y \in \text{alts}'$
shows *extend-ranking R = extend-ranking S \longleftrightarrow R = S*
using *of-ranking-imp-in-set[of dummy-alts] dummy-alts alts' assms*
unfolding *extend-ranking-def fun-eq-iff* **by** *blast*

We extend a profile to the full set of alternatives by extending each ranking.

definition *extend-profile* :: ('agent \Rightarrow 'alt relation) \Rightarrow 'agent \Rightarrow 'alt relation **where**
extend-profile $R\ i = (\lambda x\ y.\ i \in \text{agents} \wedge \text{extend-ranking}\ (R\ i)\ x\ y)$

lemma *is-pref-profile-extend* [intro]:

assumes *new.is-pref-profile* R

shows *is-pref-profile* (*extend-profile* R)

proof

fix i **assume** $i: i \in \text{agents}$

interpret R : *pref-profile-linorder-wf* $\text{agents}\ \text{alts}'\ R$

by *fact*

have *linorder-on* $\text{alts}\ (\text{extend-ranking}\ (R\ i))$

using i **by** (*simp* *add*: *linorder-on-extend-ranking*)

thus *linorder-on* $\text{alts}\ (\text{extend-profile}\ R\ i)$

using i **by** (*simp* *add*: *extend-profile-def*)

qed (*auto* *simp*: *extend-profile-def*)

lemma *count-extend-ranking-multiset*:

assumes $\bigwedge R.\ R \in \#Rs \implies \text{linorder-on}\ \text{alts}'\ R$ **and** $xy: x \in \text{alts}\ y \in \text{alts}$

shows $\text{size}\ \{\#R \in \#Rs.\ \text{extend-ranking}\ R\ x\ y\ \# \} =$

(*if* $x \in \text{alts}' \wedge y \in \text{alts}'$ *then* $\text{size}\ \{\#R \in \#Rs.\ R\ x\ y\ \# \}$

else if $x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking}\ \text{dummy-alts}\ x\ y)$ *then* $\text{size}\ Rs$ *else* 0)

proof –

have $*$: $x \in \text{alts}' \wedge y \in \text{alts}'$ **if** $R\ x\ y\ R \in \#Rs$ **for** R

proof –

interpret *linorder-on* $\text{alts}'\ R$

using *assms*(1) **that** **by** *blast*

show *?thesis*

using *not-outside*[*OF* *that*(1)] **by** *auto*

qed

have $**$: $x \in \text{alts} - \text{alts}' \wedge y \in \text{alts} - \text{alts}'$ **if** *of-ranking* *dummy-alts* $x\ y$

using *of-ranking-imp-in-set*[*OF* *that*] *dummy-alts* **by** *simp*

have $\{\#R \in \#Rs.\ \text{extend-ranking}\ R\ x\ y\ \# \} =$

(*if* $x \in \text{alts}' \wedge y \in \text{alts}'$ *then*

$\{\#R \in \#Rs.\ R\ x\ y\ \# \}$

else if $x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking}\ \text{dummy-alts}\ x\ y)$ *then* $\text{size}\ Rs$ *else* $\{\# \}$)

unfolding *extend-ranking-def*

using $xy\ \text{alts}'$ **by** (*auto* *intro*!: *filter-mset-cong* *simp*: *filter-mset-empty-conv* *dest*: $*$ $**$)

also have $\text{size}\ \dots =$ (*if* $x \in \text{alts}' \wedge y \in \text{alts}'$ *then* $\text{size}\ \{\#R \in \#Rs.\ R\ x\ y\ \# \}$

else if $x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking}\ \text{dummy-alts}\ x\ y)$ *then* $\text{size}\ Rs$ *else* 0)

by *simp*

finally show *?thesis* .

qed

lemma *count-extend-profile*:

assumes *new.is-pref-profile* R **and** $xy: x \in \text{alts}\ y \in \text{alts}$

shows $\text{card}\ \{i \in \text{agents}.\ \text{extend-profile}\ R\ i\ x\ y\} =$

(*if* $x \in \text{alts}' \wedge y \in \text{alts}'$ *then* $\text{card}\ \{i \in \text{agents}.\ R\ i\ x\ y\}$

else if $x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking}\ \text{dummy-alts}\ x\ y)$ *then* $\text{card}\ \text{agents}$ *else* 0)

proof –

interpret R : *pref-profile-linorder-wf agents alts' R* **by fact**
have $\text{card } \{i \in \text{agents}. \text{extend-profile } R \ i \ x \ y\} =$
 $(\text{card } \{i \in \text{agents}. \text{extend-ranking } (R \ i) \ x \ y\})$
 using xy **by** (*simp add: extend-profile-def extend-ranking-def*)
also have $\{i \in \text{agents}. \text{extend-ranking } (R \ i) \ x \ y\} =$
 $(\text{if } x \in \text{alts}' \wedge y \in \text{alts}' \text{ then}$
 $\{i \in \text{agents}. R \ i \ x \ y\}$
 $\text{else if } x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking dummy-alts } x \ y) \text{ then agents else } \{\})$
 unfolding *extend-ranking-def*
 using xy *alts' dummy-alts of-ranking-imp-in-set[of dummy-alts x y] R.not-outside(2,3)[of -*
 $x \ y]$
 by force
 also have $\text{card } \dots = (\text{if } x \in \text{alts}' \wedge y \in \text{alts}' \text{ then card } \{i \in \text{agents}. R \ i \ x \ y\}$
 $\text{else if } x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking dummy-alts } x \ y) \text{ then card agents}$
 $\text{else } 0)$
 by simp
 finally show *?thesis* .
qed

lemma *majority-extend-profile:*

assumes *new.is-pref-profile R*

shows $\text{majority } (\text{extend-profile } R) = \text{extend-ranking } (\text{majority } R)$

proof (*intro ext*)

fix $x \ y$

interpret R : *pref-profile-linorder-wf agents alts' R* **by fact**

interpret R' : *pref-profile-linorder-wf agents alts extend-profile R*

using *assms(1)* **by auto**

show $\text{majority } (\text{extend-profile } R) \ x \ y = \text{extend-ranking } (\text{majority } R) \ x \ y$

proof (*cases* $x \in \text{alts} \wedge y \in \text{alts}$)

case xy : *True*

show *?thesis*

using xy *assms(1) dummy-alts of-ranking-imp-in-set[of dummy-alts x y] R.not-outside(2,3)[of -*
 $x \ y]$

of-ranking-imp-in-set[of dummy-alts y x] of-ranking-total[of x dummy-alts y]

by (*auto simp: R'.majority-iff' R.majority-iff' count-extend-profile card-gt-0-iff extend-ranking-def*)

next

case *False*

thus *?thesis* **using** *alts' dummy-alts of-ranking-imp-in-set[of dummy-alts x y]*

by (*auto simp: R'.majority-iff' extend-ranking-def R.majority-iff'*)

qed

qed

lemma *majority-mset-extend-profile:*

assumes $\bigwedge R. R \in \# \text{Rs} \implies \text{linorder-on alts}' R \ \text{Rs} \neq \{\#\}$

shows $\text{majority-mset } (\text{image-mset } \text{extend-ranking } \text{Rs}) = \text{extend-ranking } (\text{majority-mset } \text{Rs})$

proof (*intro ext*)

fix $x \ y$

have *linorder: linorder-on alts (extend-ranking R) if $R \in \# \text{Rs}$ for R*

```

using assms(1)[OF that] by (rule linorder-on-extend-ranking)
have *:  $x \in \text{alts}' \wedge y \in \text{alts}'$  if majority-mset Rs x y
using assms by (meson linorder-on-def majority-mset-not-outside order-on-def that)
have **:  $x \in \text{alts} - \text{alts}' \wedge y \in \text{alts} - \text{alts}'$  if of-ranking dummy-alt s x y
using of-ranking-imp-in-set[OF that] dummy-alt s by simp

show majority-mset (image-mset extend-ranking Rs) x y = extend-ranking (majority-mset Rs)
x y
proof (cases x ∈ alts ∧ y ∈ alts)
  case xy: True
    have majority-mset (image-mset extend-ranking Rs) x y  $\longleftrightarrow$ 
       $2 * \text{size } \{\#R \in \# \text{Rs. } \text{extend-ranking } R \ x \ y\# \} \geq \text{size } \text{Rs}$ 
    by (subst majority-mset-iff-ge[of - alts] *)
      (use linorder ⟨Rs ≠ {#}⟩ xy in ⟨auto simp: linorder-on-def order-on-def filter-mset-image-mset⟩)
    also have  $\dots = \text{extend-ranking (majority-mset Rs) x y$ 
    by (subst count-extend-ranking-multiset)
      (use assms xy in ⟨auto simp: extend-ranking-def majority-mset-iff-ge[of - alts'] dest: * **⟩)
  **)
  finally show ?thesis .
next
  case xy: False
    have  $\neg \text{majority-mset (image-mset extend-ranking Rs) x y$ 
    using majority-mset-not-outside[of image-mset extend-ranking Rs x y alts] xy linorder
    using linorder-on-def total-preorder-on.axioms(1) by fastforce
    moreover have  $\neg \text{extend-ranking (majority-mset Rs) x y$ 
    using xy alts' by (auto simp: extend-ranking-def dest: * **)
    ultimately show ?thesis
    by simp
  qed
qed

```

We define our new SWF on the full set of alternatives by extending the input profile and removing the extra alternatives from the output ranking.

definition *swf-low* :: (*'agent* \Rightarrow *'alt relation*) \Rightarrow *'alt relation* **where**
swf-low R = restrict-relation alts' (swf (extend-profile R))

sublocale *new: social-welfare-function agents alts' swf-low*

```

proof
  fix R assume new.is-pref-profile R
  then interpret swf: linorder-on alts swf (extend-profile R)
    using is-pref-profile-extend swf-wf by blast
  show linorder-on alts' (swf-low R)
    unfolding swf-low-def by (rule swf.linorder-on-restrict-subset) (fact alts')
  qed

```

Our construction preserves anonymity, unanimity, and Kemeny-strategyproofness.

lemma *anonymous-restrict*:

```

assumes anonymous-swf agents alts swf
shows anonymous-swf agents alts' swf-low

```

proof
interpret *anonymous-swf agents alts swf*
 by fact
fix πR
assume π : π *permutes agents* **and** R : *new.is-pref-profile* R
have *swf-low* $(R \circ \pi) = \text{restrict-relation alts}' (\text{swf } (\text{extend-profile } (R \circ \pi)))$
 by (*simp add: swf-low-def*)
also have *extend-profile* $(R \circ \pi) = \text{extend-profile } R \circ \pi$
 using *permutes-in-image[OF π]* **by** (*simp add: extend-profile-def fun-eq-iff*)
also have *swf ... = swf (extend-profile R)*
 by (*rule anonymous*) (*use πR in auto*)
also have *restrict-relation alts' ... = swf-low R*
 by (*simp add: swf-low-def*)
finally show *swf-low* $(R \circ \pi) = \text{swf-low } R$.
qed

lemma *unanimous-restrict*:
 assumes *unanimous-swf agents alts swf*
 shows *unanimous-swf agents alts' swf-low*
proof
interpret *unanimous-swf agents alts swf*
 by fact
fix $R x y$
assume R : *new.is-pref-profile* R **and** xy : $\forall i \in \text{agents. } y \prec [R i] x$
from $R xy$ **have** xy' : $x \in \text{alts}' \wedge y \in \text{alts}'$
 by (*metis equalsOI nonempty-agents pref-profile-linorder-wf.not-outside(2,3)*
 strongly-preferred-def)
have $y \prec [\text{swf } (\text{extend-profile } R)] x$
 by (*rule unanimous*)
 (*use $R xy xy'$ of-ranking-imp-in-set[of dummy-alts] dummy-alts*
 in *auto simp: extend-profile-def extend-ranking-def strongly-preferred-def*)
thus $y \prec [\text{swf-low } R] x$
 unfolding *swf-low-def* **using** xy' **by** (*auto simp: restrict-relation-def strongly-preferred-def*)
qed

lemma *majority-consistent-restrict*:
 assumes *majority-consistent-swf agents alts swf*
 shows *majority-consistent-swf agents alts' swf-low*
proof
fix R **assume** R : *new.is-pref-profile* R *linorder-on alts'* (*majority* R)
interpret *majority-consistent-swf agents alts swf* **by fact**
have *swf-low* $R = \text{restrict-relation alts}' (\text{swf } (\text{extend-profile } R))$
 by (*simp add: swf-low-def*)
also have *swf (extend-profile R) = majority (extend-profile R)*
 by (*rule majority-consistent*)
 (*use R in auto simp: majority-extend-profile linorder-on-extend-ranking*)
also have $\dots = \text{extend-ranking (majority } R)$
 by (*rule majority-extend-profile*) *fact*

also have *restrict-relation* *alts'* (*extend-ranking* (*majority R*)) = *majority R*
by (*rule restrict-extend-ranking*) *fact*
finally show *swf-low R* = *majority R* .
qed
end

locale *unanimous-swf-restrict-alts* =
swf-restrict-alts agents alts swf dummy-alts alts' +
unanimous-swf agents alts swf
for *agents* :: '*agent set* **and** *alts* :: '*alt set* **and** *swf dummy-alts alts'*
begin

sublocale *new: unanimous-swf agents alts' swf-low*
by (*rule unanimous-restrict*) *unfold-locales*

lemma *swf-dummy-alts-least-preferred*:
assumes *new.is-pref-profile R x ∈ alts' y ∈ alts – alts'*
shows $x \succ_{[swf (extend-profile R)]} y$
proof (*rule unanimous*)
interpret *R: pref-profile-linorder-wf agents alts' R*
by *fact*
show $\forall i \in agents. x \succ_{[extend-profile R i]} y$
using *assms(2,3) alts' R.not-outside(3) of-ranking-imp-in-set[of dummy-alts] dummy-alts*
by (*auto simp: extend-profile-def extend-ranking-def strongly-preferred-def*)
qed (*use assms in auto*)

lemma *swf-strongly-preferred-dummy-alts*:
assumes *new.is-pref-profile R x ∈ alts – alts' y ∈ alts – alts'*
assumes $x \succ_{[of-ranking dummy-alts]} y$
shows $x \succ_{[swf (extend-profile R)]} y$
proof (*rule unanimous*)
interpret *R: pref-profile-linorder-wf agents alts'*
by *fact*
show $\forall i \in agents. y \prec_{[extend-profile R i]} x$
using *assms(2–) R.not-outside(3)*
by (*auto simp: strongly-preferred-def extend-profile-def extend-ranking-def*)
qed (*use assms(1) in auto*)

lemma *swf-preferred-dummy-alts-iff*:
assumes *new.is-pref-profile R x ∈ alts – alts' y ∈ alts – alts'*
shows $x \succeq_{[of-ranking dummy-alts]} y \longleftrightarrow x \succeq_{[swf (extend-profile R)]} y$
proof –
interpret *dummy-alts: linorder-on alts – alts' of-ranking dummy-alts*
by (*rule linorder-of-ranking*) (*use dummy-alts in auto*)
interpret *res: linorder-on alts swf (extend-profile R)*
by (*rule swf-wf*) (*use assms(1) in auto*)

```

show ?thesis
  using swf-strongly-preferred-dummy-alts[OF assms(1-3)]
    swf-strongly-preferred-dummy-alts[OF assms(1,3,2)]
    dummy-alts.total res.total dummy-alts.antisymmetric res.antisymmetric assms(2,3)
  unfolding strongly-preferred-def by blast
qed

lemma swf-strongly-preferred-dummy-alts-iff:
  assumes new.is-pref-profile R  $x \in \text{alts} - \text{alts}'$   $y \in \text{alts} - \text{alts}'$ 
  shows  $x \succ_{\text{swf}} (\text{extend-profile } R) y \iff x \succ_{\text{of-ranking dummy-alts}} y$ 
proof -
  interpret dummy-alts: linorder-on alts - alts' of-ranking dummy-alts
    by (rule linorder-of-ranking) (use dummy-alts in auto)
  interpret res: linorder-on alts swf (extend-profile R)
    by (rule swf-wf) (use assms(1) in auto)
  show ?thesis
    using swf-strongly-preferred-dummy-alts[OF assms(1-3)]
      swf-strongly-preferred-dummy-alts[OF assms(1,3,2)]
      dummy-alts.total res.total dummy-alts.antisymmetric res.antisymmetric assms(2,3)
    unfolding strongly-preferred-def by blast
qed

lemma extend-ranking-swf-low:
  assumes new.is-pref-profile R
  shows extend-ranking (swf-low R) = swf (extend-profile R)
proof -
  interpret lhs: linorder-on alts extend-ranking (swf-low R)
    using assms by (intro linorder-on-extend-ranking new.swf-wf)
  interpret rhs: linorder-on alts swf (extend-profile R)
    by (rule swf-wf) (use assms in auto)

  have extend-ranking (swf-low R)  $x y \iff \text{swf} (\text{extend-profile } R) x y$ 
  if  $x \in \text{alts}$   $y \in \text{alts}$  for  $x y$ 
  proof (cases  $x \in \text{alts}'$ ; cases  $y \in \text{alts}'$ )
    assume  $x \in \text{alts}'$   $y \in \text{alts}'$ 
    thus ?thesis using of-ranking-imp-in-set[of dummy-alts  $x y$ ] dummy-alts
      by (auto simp: swf-low-def restrict-relation-def extend-ranking-def)
  next
    assume  $x \in \text{alts}'$   $y \notin \text{alts}'$ 
    thus ?thesis
      using that of-ranking-imp-in-set[of dummy-alts  $x y$ ] dummy-alts
        swf-dummy-alts-least-preferred[of R  $x y$ ] assms
      by (auto simp: swf-low-def restrict-relation-def extend-ranking-def strongly-preferred-def)
  next
    assume  $x \notin \text{alts}'$   $y \in \text{alts}'$ 
    thus ?thesis
      using that swf-dummy-alts-least-preferred[of R  $y x$ ] assms
      by (auto simp: swf-low-def restrict-relation-def extend-ranking-def strongly-preferred-def)
  next

```

```

assume  $x \notin \text{alts}' y \notin \text{alts}'$ 
thus ?thesis using that swf-preferred-dummy-alts-iff[OF assms]
  by (auto simp: swf-low-def restrict-relation-def extend-ranking-def)
qed
thus ?thesis
  using lhs.not-outside rhs.not-outside unfolding fun-eq-iff by blast
qed

```

lemma *kemeny-strategyproof-restrict:*

```

assumes kemeny-strategyproof-swf agents alts swf
shows kemeny-strategyproof-swf agents alts' swf-low
proof

```

```

interpret kemeny-strategyproof-swf agents alts swf
  by fact
fix  $R i S$ 

```

```

assume  $R$ : new.is-pref-profile R and  $i$ :  $i \in \text{agents}$  and  $S$ : linorder-on alts' S
define  $R'$  where  $R' = \text{extend-profile } R$ 
define  $S'$  where  $S' = \text{extend-ranking } S$ 

```

```

interpret  $R$ : pref-profile-linorder-wf agents alts' R
  by fact
interpret  $Ri$ : linorder-on alts' R i
  using  $i$  by simp

```

```

have swap-dist-relation (R i) (swf-low R) = swap-dist-relation (R' i) (swf R')
proof –

```

```

  have swap-dist-relation (R i) (swf-low R) =
    swap-dist-relation (extend-ranking (R i)) (extend-ranking (swf-low R))
    by (rule swap-dist-extend-ranking [symmetric])
    (use R Ri.linorder-on-axioms in <auto intro: new.swf-wf>)
  also have extend-ranking (R i) = R' i
    using  $i$  by (simp add: R'-def extend-profile-def)
  also have extend-ranking (swf-low R) = swf R'
    by (subst extend-ranking-swf-low) (use R in <simp-all add: R'-def>)
  finally show swap-dist-relation (R i) (swf-low R) = swap-dist-relation (R' i) (swf R') .
qed

```

```

also have swap-dist-relation (R' i) (swf R') ≤ swap-dist-relation (R' i) (swf (R'(i := S')))
  by (rule kemeny-strategyproof)
  (use R S i in <auto simp: R'-def S'-def linorder-on-extend-ranking>)

```

```

also have swap-dist-relation (R' i) (swf (R'(i := S'))) =
  swap-dist-relation (R i) (swf-low (R(i := S)))

```

```

proof –
  have swap-dist-relation (R i) (swf-low (R(i := S))) =
    swap-dist-relation (extend-ranking (R i)) (extend-ranking (swf-low (R(i := S))))
    by (rule swap-dist-extend-ranking [symmetric])
    (use R S Ri.linorder-on-axioms i in <auto intro: new.swf-wf>)
  also have extend-ranking (R i) = R' i
    using  $i$  by (simp add: R'-def extend-profile-def)

```

```

also have extend-ranking (swf-low ( $R(i := S)$ )) = swf (extend-profile ( $R(i := S)$ ))
  by (subst extend-ranking-swf-low) (use R S i in auto)
also have extend-profile ( $R(i := S)$ ) =  $R'(i := S')$ 
  using i unfolding R'-def S'-def extend-profile-def by (auto simp: fun-eq-iff)
finally show swap-dist-relation ( $R' i$ ) (swf ( $R'(i := S')$ )) =
  swap-dist-relation ( $R i$ ) (swf-low ( $R(i := S)$ )) ..

qed
finally show swap-dist-relation ( $R i$ ) (swf-low  $R$ ) ≤ swap-dist-relation ( $R i$ ) (swf-low ( $R(i := S)$ )) .
qed

end

```

```

locale majority-consistent-weak-kstratproof-swf-restrict-alts =
  majority-consistent-weak-kstratproof-swf agents alts swf +
  swf-restrict-alts agents alts swf dummy-alts alts'
  for agents :: 'agent set and alts :: 'alt set and swf dummy-alts alts'
begin

```

```

sublocale new: majority-consistent-swf agents alts' swf-low
  by (rule majority-consistent-restrict) unfold-locales

```

```

sublocale new: majority-consistent-weak-kstratproof-swf agents alts' swf-low
proof

```

```

  fix  $R i S$ 
  assume  $R$ : new.is-pref-profile  $R$  and  $i$ :  $i \in \text{agents}$  and  $S$ : linorder-on alts' S
  assume  $maj$ : linorder-on alts' (majority (R(i := S)))
  define  $R'$  where  $R' = \text{extend-profile } R$ 
  define  $S'$  where  $S' = \text{extend-ranking } S$ 
  interpret  $R$ : pref-profile-linorder-wf agents alts' R by fact
  interpret  $Ri$ : finite-linorder-on alts' R i
  using  $i$  by simp

```

```

  have  $maj'$ : linorder-on alts (majority (R'(i := S')))

```

```

proof –

```

```

  have linorder-on alts (extend-ranking (majority (R(i := S))))
    by (intro linorder-on-extend-ranking maj)
  also have extend-ranking (majority (R(i := S))) = majority (extend-profile (R(i := S)))
    by (rule majority-extend-profile [symmetric]) (use R i S in auto)
  also have extend-profile (R(i := S)) = R'(i := S')
    unfolding  $R'$ -def  $S'$ -def using  $i$  by (auto simp: extend-profile-def fun-eq-iff)
  finally show linorder-on alts (majority (R'(i := S'))) .

```

```

qed

```

```

have swap-dist-relation ( $R i$ ) (swf-low  $R$ ) =
  swap-dist-relation ( $R i$ ) (restrict-relation alts' (swf R'))
  by (simp add: swf-low-def R'-def)
also have ... = swap-dist-relation (restrict-relation alts' (R' i)) (restrict-relation alts' (swf

```

$R')$
using i *linorder-on-axioms*
by (*simp add: R'-def extend-profile-def restrict-extend-ranking*)
also have $\dots \leq \text{swap-dist-relation } (R' i) (\text{swf } R')$
proof (*rule swap-dist-relation-restrict*)
show *linorder-on alts* $(R' i)$
by (*metis R R'-def i is-pref-profile-extend pref-profile-linorder-wf.prefs-wf'*)
qed (*use R in <auto intro!: swf-wf simp: R'-def>*)
also have $\dots \leq \text{swap-dist-relation } (R' i) (\text{majority } (R'(i := S')))$
by (*rule majority-consistent-kemeny-strategyproof*)
(use R i S maj' in <auto simp: R'-def S'-def linorder-on-extend-ranking>)
also have $R'(i := S') = \text{extend-profile } (R(i := S))$
using i **by** (*auto simp: S'-def R'-def extend-profile-def*)
also have *majority* $\dots = \text{extend-ranking } (\text{majority } (R(i := S)))$
by (*rule majority-extend-profile*) (*use R i S in auto*)
also have $R' i = \text{extend-ranking } (R i)$
using i **by** (*auto simp: R'-def extend-profile-def fun-eq-iff*)
also have *swap-dist-relation* (*extend-ranking* $(R i)$) (*extend-ranking* (*majority* $(R(i := S))$))
 $=$
swap-dist-relation $(R i) (\text{majority } (R(i := S)))$
by (*rule swap-dist-extend-ranking*) (*use R i S maj in auto*)
finally show *swap-dist-relation* $(R i) (\text{swf-low } R) \leq$
swap-dist-relation $(R i) (\text{majority } (R(i := S)))$.
qed
end

locale *swf-restrict-alts-explicit* =
swf-restrict-alts agents alts swf dummy-alts alts' +
social-welfare-function-explicit agents alts swf agents-list alts-list
for *agents* :: 'agent set **and** *alts* :: 'alt set
and *swf dummy-alts alts' agents-list alts-list alts-list' +*
assumes *alts-list-expand: alts-list = alts-list' @ dummy-alts*
begin

lemma *mset-alts-list: mset alts-list = mset alts-list' + mset dummy-alts*
by (*simp add: alts-list-expand*)

sublocale *new: social-welfare-function-explicit agents alts' swf-low agents-list alts-list'*
proof

show *mset alts-list' = mset-set alts'*
using *alts-list dummy-alts-alts' mset-alts-list* **by** *auto*
qed (*fact agents-list*)

definition *extend* :: 'alt list \Rightarrow 'alt list **where** *extend* = $(\lambda xs. xs @ \text{dummy-alts})$

lemma *distinct-alts-list': distinct alts-list'*
and *alts-list'-not-in-dummy-alts: set alts-list' \cap set dummy-alts = {}*

using *distinct-alts-list* **unfolding** *alts-list-expand* **by** *auto*

lemma *wf-extend*:

assumes *new.prefs-from-rankings-wf R*

shows *prefs-from-rankings-wf (map extend R)*

using *assms* **unfolding** *new.prefs-from-rankings-wf-def prefs-from-rankings-wf-def extend-def*

by (*auto simp: list.pred-set mset-alts-list*)

lemma *of-ranking-extend*:

assumes *mset xs = mset alts-list'*

shows *of-ranking (extend xs) = extend-ranking (of-ranking xs)*

unfolding *extend-def of-ranking-append extend-ranking-def fun-eq-iff*

unfolding *alts-conv-alts-list alts-list-expand*

using *alts-list'-not-in-dummy-alts new.mset-eq-alts-list-iff[of xs] assms new.alts-conv-alts-list*

by *auto*

lemma *swap-dist-extend*:

assumes *mset xs = mset alts-list' mset ys = mset alts-list'*

shows *swap-dist (extend xs) (extend ys) = swap-dist xs ys*

proof –

have ***: *distinct xs ∧ set xs = alts' ∧ distinct ys ∧ set ys = alts'*

using *assms* **by** (*metis new.mset-eq-alts-list-iff*)

show *?thesis* **unfolding** *extend-def*

by (*rule swap-dist-append-right*) (*use * dummy-alts alts-list'-not-in-dummy-alts in auto*)

qed

lemma *prefs-from-rankings-extend*:

assumes *R: new.prefs-from-rankings-wf R*

shows *prefs-from-rankings (map extend R) = extend-profile (new.prefs-from-rankings R)*

(*is ?lhs = ?rhs*)

proof

fix *i*

note *R' = wf-extend[OF R]*

show *?lhs i = ?rhs i*

proof (*cases i ∈ agents*)

case *True*

then obtain *j* **where** *j: j < card agents i = agents-list ! j*

by (*metis agents-conv-agents-list card-agents index-less-size-conv nth-index*)

show *?thesis*

using *j(1) new.prefs-from-rankings-nth[OF R, of j] prefs-from-rankings-nth[OF R', of j] R*

True

unfolding *j(2)*

by (*simp add: extend-profile-def new.prefs-from-rankings-wf-def of-ranking-extend list.pred-set*)

qed (*auto simp: extend-profile-def prefs-from-rankings-outside*)

qed

lemma *majority-rel-mset-extend*:

assumes *R: new.prefs-from-rankings-wf R* **and** *S: mset S = mset alts-list'*

shows *majority-rel-mset (mset (map extend R)) (extend S) ↔ majority-rel-mset (mset R) S*

proof –
have S' : $\text{distinct } S \wedge \text{set } S = \text{alts}'$ **using** S **unfolding** extend-def
by ($\text{metis new.mset-eq-alts-list-iff}$)
have $\text{majority-rel-mset } (\text{mset } (\text{map } \text{extend } R)) (\text{extend } S) \longleftrightarrow$
 $(\text{majority-mset } (\text{image-mset } (\text{of-ranking } \circ \text{extend}) (\text{mset } R)) = \text{of-ranking } (\text{extend } S) \wedge$
 $\text{distinct } (\text{extend } S))$
by ($\text{simp add: majority-rel-mset-def image-mset.compositionality}$)
also have $\text{distinct } (\text{extend } S) \longleftrightarrow \text{distinct } S$
using S' $\text{alts-list'-not-in-dummy-alts dummy-alts}$ **by** ($\text{auto simp: extend-def}$)
also have $\text{of-ranking } (\text{extend } S) = \text{extend-ranking } (\text{of-ranking } S)$
by ($\text{rule of-ranking-extend}$) ($\text{use } S$ **in** simp-all)
also have $\text{image-mset } (\text{of-ranking } \circ \text{extend}) (\text{mset } R) =$
 $\text{image-mset } (\text{extend-ranking } \circ \text{of-ranking}) (\text{mset } R)$ **unfolding** o-def
by ($\text{intro image-mset-cong of-ranking-extend}$)
 $(\text{use } R$ **in** $\langle \text{auto simp: new.prefs-from-rankings-wf-def list.pred-set} \rangle)$
also have $\dots = \text{image-mset } \text{extend-ranking } (\text{image-mset } \text{of-ranking } (\text{mset } R))$
by ($\text{simp add: image-mset.compositionality o-def}$)
also have $\text{majority-mset } \dots = \text{extend-ranking } (\text{majority-mset } (\text{image-mset } \text{of-ranking } (\text{mset } R)))$
proof –
have [simp]: $R \neq []$
using R **by** ($\text{auto simp: new.prefs-from-rankings-wf-def}$)
have $\text{linorder-on alts}' (\text{of-ranking } rs)$ **if** $rs \in \text{set } R$ **for** rs
using $\text{that } R$ $\text{new.mset-eq-alts-list-iff}[of rs]$
by ($\text{intro linorder-of-ranking}$) ($\text{auto simp: new.prefs-from-rankings-wf-def list.pred-set}$)
thus $?thesis$
by ($\text{intro majority-mset-extend-profile}$) auto
qed
also have $\text{extend-ranking } (\text{majority-mset } (\text{image-mset } \text{of-ranking } (\text{mset } R))) =$
 $\text{extend-ranking } (\text{of-ranking } S) \longleftrightarrow$
 $\text{majority-mset } (\text{image-mset } \text{of-ranking } (\text{mset } R)) = \text{of-ranking } S$
proof ($\text{rule extend-ranking-eq-iff}$)
have $*$: $\text{preorder-on alts}' (\text{of-ranking } rs)$ **if** $rs \in \text{set } R$ **for** rs
proof –
have $\text{distinct } rs \wedge \text{set } rs = \text{alts}'$
using $\text{that } R$ $\text{new.mset-eq-alts-list-iff}[of rs]$
by ($\text{auto simp: new.prefs-from-rankings-wf-def list.pred-set}$)
then interpret $\text{linorder-on alts}' \text{of-ranking } rs$
by ($\text{intro linorder-of-ranking}$) auto
show $?thesis ..$
qed
show $x \in \text{alts}' \wedge y \in \text{alts}'$
if $\text{majority-mset } (\text{image-mset } \text{of-ranking } (\text{mset } R)) x y$ **for** $x y$
using $\text{majority-mset-not-outside}[OF \text{that}, \text{of alts}] *$ **by** auto
next
show $x \in \text{alts}' \wedge y \in \text{alts}'$ **if** $\text{of-ranking } S x y$ **for** $x y$
using S' $\text{of-ranking-imp-in-set}[OF \text{that}]$ **by** auto
qed
also have $\dots \wedge \text{distinct } S \longleftrightarrow \text{majority-rel-mset } (\text{mset } R) S$

unfolding *majority-rel-mset-def ..*
finally show *?thesis .*
qed

lemma *new-swf'-eq:*

assumes *R: new.prefs-from-rankings-wf R*

shows *new.swf' R = filter (λx. x ∈ alts') (swf' (map extend R))*

proof –

have *mset (swf' (map extend R)) = mset-set alts*

by *(intro swf'-wf wf-extend R)*

hence *distinct (swf' (map extend R))*

using *distinct-alts-list mset-eq-imp-distinct-iff alts-list* **by** *metis*

have *new.swf' R = ranking (swf-low (new.prefs-from-rankings R))*

by *(simp add: new.swf'-def new.swf'-def new.prefs-from-rankings-def)*

also have *... = ranking (restrict-relation alts' (swf (prefs-from-rankings (map extend R))))*

using *R* **by** *(simp add: swf-low-def prefs-from-rankings-extend)*

also have *swf (prefs-from-rankings (map extend R)) =*

of-ranking (ranking (swf (prefs-from-rankings (map extend R))))

by *(rule finite-linorder-on.of-ranking-ranking [OF swf-wf', symmetric])*

(use R in ⟨auto intro: wf-extend⟩)

also have *... = of-ranking (swf' (map extend R))*

unfolding *swf'-def* **by** *(simp add: new.prefs-from-rankings-def swf'-def)*

also have *restrict-relation alts' ... =*

of-ranking (filter (λx. x ∈ alts') (swf' (map extend R)))

unfolding *of-ranking-filter Collect-mem-eq ..*

also have *ranking ... = filter (λx. x ∈ alts') (swf' (map extend R))*

by *(intro ranking-of-ranking distinct-filter)*

(use ⟨distinct (swf' (map extend R))⟩ in auto)

finally show *?thesis .*

qed

end

2.7.2 Decreasing the number of agents by a factor

The nicest way to formalise the cloning construction would be using the view where a profile is a multiset of rankings. However, this requires anonymity. For full generality, we show that the construction also works in the absence of anonymity.

To this end, we first define the notion of a *cloning*. Let $A \subseteq B$. The idea is that $B \setminus A$ consists of clones of elements of A , and each element of A is cloned equally often. We model this via a function called “unclone” which maps each element of A to itself and every element of $B \setminus A$ to the original element in B that it was cloned from.

locale *cloning =*

fixes *A B unclone*

assumes *subset: A ⊆ B*

assumes *finite: finite B*

assumes *unclone: ∧x. x ∈ B ⇒ unclone x ∈ A*

assumes *unclone-ident: ∧x. x ∈ A ⇒ unclone x = x*

assumes *card-unclone*:
 $x \in A \implies y \in A \implies \text{card} (\text{unclone } -' \{x\} \cap B) = \text{card} (\text{unclone } -' \{y\} \cap B)$
begin

definition *clones* :: 'a \Rightarrow 'a set
where *clones* *i* = *unclone* -' {*i*} \cap *B*

definition *factor* :: nat
where *factor* = *card* *B* *div* *card* *A*

lemma *finite-clones*: *finite* (*clones* *i*)
by (*rule* *finite-subset*[*OF* - *finite*]) (*auto simp: clones-def*)

lemma *clones-outside*: $i \notin A \implies \text{clones } i = \{\}$
unfolding *clones-def* **using** *unclone* **by** *auto*

lemma *card-clones'*:
assumes $i \in A$
shows $\text{card} (\text{clones } i) * \text{card } A = \text{card } B$
proof –
have $B = (\bigcup_{i \in A} \text{clones } i)$
using *unclone* **unfolding** *clones-def* **by** *blast*
also from *subset* **have** $\dots = (\sum_{j \in A} \text{card} (\text{clones } j))$
by (*subst* *card-UN-disjoint*) (*auto simp: clones-def intro: finite-subset*[*OF* - *finite*])
also have $\dots = (\sum_{j \in A} \text{card} (\text{clones } i))$
unfolding *clones-def* **by** (*intro sum.cong card-unclone assms refl*)
also have $\dots = \text{card } A * \text{card} (\text{clones } i)$
by *simp*
finally show *?thesis* **by** (*simp add: mult-ac*)
qed

lemma *card-clones*:
assumes $i \in A$
shows $\text{card} (\text{clones } i) = \text{factor}$
proof (*cases* $B = \{\}$)
case *True*
thus *?thesis*
unfolding *clones-def factor-def* **by** *simp*
next
case *False*
hence $A \neq \{\}$
using *unclone* **by** *auto*
have $\text{factor} = \text{card} (\text{clones } i) * \text{card } A \text{ div } \text{card } A$
unfolding *factor-def* **using** *card-clones'*[*OF* *assms*] **by** *simp*
also have $\dots = \text{card} (\text{clones } i)$
by (*rule nonzero-mult-div-cancel-right*)
(use finite-subset[*OF* *subset finite*] $\langle A \neq \{\} \rangle$ **in** *auto*)
finally show *?thesis* ..
qed

lemma *image-mset-unclone*:
 $image\text{-}mset\ unclone\ (mset\text{-}set\ B) = repeat\text{-}mset\ factor\ (mset\text{-}set\ A)$
(is ?lhs = ?rhs)
proof (*rule multiset-eqI*)
fix $i :: 'a$
have $count\ (image\text{-}mset\ unclone\ (mset\text{-}set\ B))\ i =$
 $sum\ (count\ (mset\text{-}set\ B))\ (clones\ i)$
by (*subst count-image-mset*) (*simp-all add: clones-def finite*)
also have $\dots = sum\ (\lambda\cdot.\ 1)\ (clones\ i)$
by (*rule sum.cong*) (*auto simp: clones-def finite*)
also have $\dots = card\ (clones\ i)$
by *simp*
also have $\dots = (if\ i \in A\ then\ factor\ else\ 0)$
using *card-clones* **by** (*auto simp: clones-outside*)
also have $\dots = count\ (repeat\text{-}mset\ factor\ (mset\text{-}set\ A))\ i$
using *finite-subset[OF subset finite]* **by** (*simp add: count-mset-set'*)
finally show $count\ ?lhs\ i = count\ ?rhs\ i$.
qed

lemma *factor-pos*: $B \neq \{\}$ $\implies factor > 0$
using *card-clones card-clones' local.finite unclone* **by** *fastforce*

end

It is easy to see (but somewhat tedious to show) that a cloning exists whenever $|B|$ is a multiple of $|A|$

lemma *cloning-exists*:
assumes $A \subseteq B$ *finite* $B \neq \{\}$ $card\ A\ dvd\ card\ B$
shows $\exists\ unclone.\ cloning\ A\ B\ unclone$
using *assms(2,1,3,4)*
proof (*induction rule: finite-psubset-induct*)
case (*psubset B*)
show *?case*
proof (*cases A = B*)
case *True*
have *cloning A B id*
by *standard (use True psubset.hyps in auto)*
thus *?thesis*
by *blast*
next
case *False*
have $card\ B \geq card\ A$
using *False psubset.prem card-mono psubset.hyps* **by** *blast*
hence $card\ (B - A) = card\ B - card\ A$
by (*subst card-Diff-subset*) (*use psubset.hyps psubset.prem in auto*)
also have $\dots \geq card\ A$
using *False psubset.prem psubset.hyps*
by (*meson antisym-conv1 dvd-imp-le dvd-minus-self psubset-card-mono zero-less-diff*)

finally obtain X where $X: X \subseteq B - A$ $\text{card } X = \text{card } A$
by (*meson obtain-subset-with-card-n*)
obtain f where $f: \text{bij-betw } f \ X \ A$
using $X(2)$ *psubset.prem*s *psubset.hyps*
by (*metis card-gt-0-iff finite-same-card-bij finite-subset*)

have $\exists \text{unclone. cloning } A \ (B - X) \ \text{unclone}$
proof (*rule psubset.IH*)
have $X \neq \{\}$
using X *psubset.hyps* *psubset.prem*s **by** *force*
thus $B - X \subset B$
using $X(1)$ **by** *blast*
have $\text{card } A \ \text{dvd} \ \text{card } B - \text{card } X$
using X $\langle \text{card } B \geq \text{card } A \rangle$ *dvd-minus-self* *psubset.prem*s(3) **by** *metis*
also have $\text{card } B - \text{card } X = \text{card } (B - X)$
by (*subst card-Diff-subset*) (*use X finite-subset[OF X(1)] psubset.hyps in auto*)
finally show $\text{card } A \ \text{dvd} \ \text{card } (B - X)$.
qed (*use X psubset.hyps psubset.prem in auto*)
then obtain unclone where $\text{cloning } A \ (B - X) \ \text{unclone}$..
interpret $\text{cloning } A \ B - X \ \text{unclone}$ by *fact*

define $\text{unclone}'$ where $\text{unclone}' = (\lambda x. \text{if } x \in X \text{ then } f \ x \ \text{else } \text{unclone } \ x)$
have $\text{cloning } A \ B \ \text{unclone}'$
proof
show $A \subseteq B$ *finite B*
by *fact+*
next
show $\text{unclone}' \ x \in A$ **if $x \in B$ **for** x**
using *unclone f* **that** **by** (*auto simp: unclone'-def bij-betw-def*)
next
show $\text{unclone}' \ x = x$ **if $x \in A$ **for** x**
using *that X unclone-ident* **by** (*auto simp: unclone'-def*)
next
have $*$: $\text{card } (\text{unclone}' - \{x\} \cap B) = \text{factor} + 1$ **if $x \in A$ **for** x**
proof -
have $\text{unclone}' - \{x\} \cap B = (\text{unclone}' - \{x\} \cap (B - X)) \cup (\text{unclone}' - \{x\} \cap X)$
using X **by** *blast*
also have $\text{card } \dots = \text{card } (\text{unclone}' - \{x\} \cap (B - X)) + \text{card } (\text{unclone}' - \{x\} \cap X)$
by (*rule card-Un-disjoint*) (*use X psubset.hyps in <auto intro: finite-subset>*)
also have $\text{unclone}' - \{x\} \cap (B - X) = \text{clones } x$
by (*auto simp: unclone'-def clones-def*)
also have $\text{card } (\text{clones } x) = \text{factor}$
by (*rule card-clones*) *fact*
also have $\text{unclone}' - \{x\} \cap X = f - \{x\} \cap X$
by (*auto simp: unclone'-def*)
also have $f - \{x\} \cap X = \{\text{inv-into } X \ f \ x\}$
using f *bij-betw-inv-into-left[OF f]* *bij-betw-inv-into-right[OF f]* **that**
by (*auto intro!: inv-into-into simp: bij-betw-def inj-on-def*)
finally show ?thesis

```

    by simp
  qed
  show card (unclone' - {x} ∩ B) = card (unclone' - {y} ∩ B) if x ∈ A y ∈ A for x y
    using that[THEN *] by simp
  qed
  thus ?thesis
    by blast
  qed
qed

```

We are now ready to give the actual construction.

```

locale swf-split-agents =
  social-welfare-function agents alts swf +
  clone: cloning agents' agents unclone
  for agents :: 'agent set and alts :: 'alt set and swf and agents' unclone
begin

```

```

lemmas agents' = clone.subset

```

```

lemma nonempty-agents': agents' ≠ {}
  using clone.unclone nonempty-agents by blast

```

```

sublocale new: linorder-election agents' alts
  by standard (use finite-subset[OF agents'] nonempty-agents' in auto)

```

The profiles are extended in the obvious way: the ranking declared by a clone is the same as the ranking of its original.

```

definition extend-profile :: ('agent ⇒ 'alt relation) ⇒ 'agent ⇒ 'alt relation where
  extend-profile R i = (if i ∈ agents then R (unclone i) else (λ-. False))

```

```

lemma is-pref-profile-extend-profile [intro]:
  assumes new.is-pref-profile R
  shows is-pref-profile (extend-profile R)
proof
  fix i assume i: i ∈ agents
  interpret R: pref-profile-linorder-wf agents' alts R
    by fact
  show linorder-on alts (extend-profile R i)
    using i clone.unclone unfolding extend-profile-def by auto
qed (auto simp: extend-profile-def)

```

```

lemma count-extend-profile:
  card {i∈agents. extend-profile R i x y} = clone.factor * card {i∈agents'. R i x y}
proof -
  have card {i ∈ agents. extend-profile R i x y} =
    size (filter-mset (λi. extend-profile R i x y) (mset-set agents))
    by simp
  also have filter-mset (λi. extend-profile R i x y) (mset-set agents) =
    filter-mset (λi. R (unclone i) x y) (mset-set agents)

```

```

unfolding extend-profile-def by (intro filter-mset-cong) auto
also have size ... = size (filter-mset ( $\lambda i. R i x y$ ) (image-mset unclone (mset-set agents)))
by (simp add: filter-mset-image-mset)
also have image-mset unclone (mset-set agents) = repeat-mset clone.factor (mset-set agents')
by (simp add: clone.image-mset-unclone)
also have size (filter-mset ( $\lambda i. R i x y$ ) ...) =
    clone.factor * card {i  $\in$  agents'. R i x y}
by (simp add: filter-mset-repeat-mset)
finally show ?thesis .
qed

```

```

lemma majority-extend-profile:
  assumes new.is-pref-profile R
  shows majority (extend-profile R) = majority R
proof (intro ext)
  fix x y :: 'alt
  interpret R: pref-profile-linorder-wf agents' alts R by fact
  interpret R': pref-profile-linorder-wf agents alts extend-profile R
  using assms by auto
  show majority (extend-profile R) x y = majority R x y
  using clone.factor-pos by (simp add: R.majority-iff' R'.majority-iff' count-extend-profile)
qed

```

Correspondingly, we define our new SWF by feeding the cloned profiles to the old one.

```

definition swf-low :: ('agent  $\Rightarrow$  'alt relation)  $\Rightarrow$  'alt relation
  where swf-low R = swf (extend-profile R)

```

```

sublocale new: social-welfare-function agents' alts swf-low
proof
  fix R assume R: new.is-pref-profile R
  thus linorder-on alts (swf-low R)
  unfolding swf-low-def by (intro swf-wf) auto
qed

```

It is easy to see that cloning commutes with a permutation of the agents, so the resulting SWF is still anonymous if the original one was.

```

lemma anonymous-clone:
  assumes anonymous-swf agents alts swf
  shows anonymous-swf agents' alts swf-low
proof
  interpret anonymous-swf agents alts swf by fact
  fix  $\pi$  R assume  $\pi$ :  $\pi$  permutes agents' and R: new.is-pref-profile R
  interpret R: pref-profile-linorder-wf agents' alts R
  by fact
  have  $\pi'$ :  $\pi$  permutes agents
  using  $\pi$  agents' by (rule permutes-subset)
  show swf-low (R  $\circ$   $\pi$ ) = swf-low R
  unfolding swf-low-def
  proof (rule anonymous')

```

```

have image-mset (extend-profile (R ∘ π)) (mset-set agents) =
  image-mset (R ∘ π ∘ unclone) (mset-set agents)
  by (intro image-mset-cong) (auto simp: extend-profile-def)
also have ... = image-mset (R ∘ π) (image-mset unclone (mset-set agents))
  by (simp add: multiset.map-comp)
also have ... = repeat-mset clone.factor (image-mset (R ∘ π) (mset-set agents'))
  by (simp add: clone.image-mset-unclone image-mset-repeat-mset)
also have image-mset (R ∘ π) (mset-set agents') = image-mset R (mset-set agents')
  using  $\pi$  by (simp add: permutes-image-mset flip: multiset.map-comp)
also have repeat-mset clone.factor ... = image-mset (R ∘ unclone) (mset-set agents)
  by (simp add: image-mset-repeat-mset clone.image-mset-unclone flip: multiset.map-comp)
also have ... = image-mset (extend-profile R) (mset-set agents)
  by (rule image-mset-cong) (auto simp: extend-profile-def)
finally show image-mset (extend-profile (R ∘ π)) (mset-set agents) =
  image-mset (extend-profile R) (mset-set agents) .
qed (use R R.wf-permute-agents[OF π] in auto)
qed

```

Unanimity is obviously preserved as well.

lemma *unanimous-clone:*

assumes *unanimous-swf agents alts swf*
shows *unanimous-swf agents' alts swf-low*

proof

interpret *unanimous-swf agents alts swf*
by *fact*

fix $R\ x\ y$ **assume** R : *new.is-pref-profile R* **and** xy : $\forall i \in \text{agents}'.\ y \prec_{[R\ i]} x$

show $y \prec_{[swf\text{-low}\ R]} x$

unfolding *swf-low-def*

proof (*rule unanimous*)

show $\forall i \in \text{agents}.\ y \prec_{[extend\text{-profile}\ R\ i]} x$

using xy *clone.unclone* **by** (*auto simp: strongly-preferred-def extend-profile-def*)

qed (*use R in auto*)

qed

Strategyproofness is slightly more involved. A manipulation by a single agent in an original profile corresponds to a simultaneous manipulation of them and all their clones. However, it can be shown that the normal notion of Kemeny strategyproofness (where only one agent is allowed to manipulate) also implies that no set of clones can obtain a better result by manipulating simultaneously. This works by simply considering a chain of single-agent manipulations.

This shows that strategyproofness is also preserved.

lemma *kemeny-strategyproof-clone:*

assumes *kemeny-strategyproof-swf agents alts swf*
shows *kemeny-strategyproof-swf agents' alts swf-low*

proof

fix $R\ i\ S$ **assume** R : *new.is-pref-profile R* **and** i : $i \in \text{agents}'$ **and** S : *linorder-on alts S*

interpret *kemeny-strategyproof-swf agents alts swf* **by** *fact*

interpret R : *pref-profile-linorder-wf agents' alts R* **by** *fact*

```

define  $C$  where  $C = \text{unclone } - \{i\} \cap \text{agents}$ 
define  $R'$  where  $R' = (\lambda X j. \text{if } j \in X \text{ then } S \text{ else } \text{extend-profile } R j)$ 

have  $\text{step: swap-dist-relation } (R i) (\text{swf } (R' X)) \leq \text{swap-dist-relation } (R i) (\text{swf } (R' (\text{insert } x X)))$ 
  if  $x: \text{insert } x X \subseteq C \ x \notin X$  for  $x X$ 
  proof –
    have  $\text{swap-dist-relation } (R' X x) (\text{swf } (R' X)) \leq \text{swap-dist-relation } (R' X x) (\text{swf } ((R' X)(x := S)))$ 
    proof (rule kemeny-strategyproof)
      show  $x \in \text{agents}$ 
      using  $x$  by (auto simp: C-def)
    next
      show  $\text{is-pref-profile } (R' X)$ 
      proof
        fix  $j$  assume  $j \in \text{agents}$ 
        thus  $\text{linorder-on alts } (R' X j)$ 
        unfolding  $R'\text{-def}$  using  $S R \text{ clone.unclone}$  by (auto simp: extend-profile-def)
        qed (use x in ‹auto simp: R'-def C-def extend-profile-def›)
      qed fact+
    also have  $R' X x = R i$ 
    using  $x$  by (auto simp: R'-def extend-profile-def C-def)
    also have  $(R' X)(x := S) = R' (\text{insert } x X)$ 
    using  $x$  by (auto simp: R'-def)
    finally show ?thesis .
  qed

show  $\text{swap-dist-relation } (R i) (\text{swf-low } R) \leq \text{swap-dist-relation } (R i) (\text{swf-low } (R(i := S)))$ 
proof –
  define  $X$  where  $X = C$ 
  have  $\text{finite } C$  unfolding  $C\text{-def}$ 
  by (rule finite-subset[OF - finite-agents]) auto
  moreover have  $x \in \text{agents}$   $\text{unclone } x = i$  if  $x \in C$  for  $x$ 
  using that unfolding  $C\text{-def}$  by blast+
  ultimately have  $\text{swap-dist-relation } (R i) (\text{swf-low } R) \leq \text{swap-dist-relation } (R i) (\text{swf } (R' C))$ 
  proof (induction rule: finite-induct)
    case ( $\text{insert } x X$ )
    have  $\text{swap-dist-relation } (R i) (\text{swf-low } R) \leq \text{swap-dist-relation } (R i) (\text{swf } (R' X))$ 
    by (rule insert.IH) (use insert.prem in auto)
    also have  $\text{swap-dist-relation } (R i) (\text{swf } (R' X)) \leq \text{swap-dist-relation } (R i) (\text{swf } (R' (\text{insert } x X)))$ 
    by (rule step) (use insert.hyps insert.prem in ‹auto simp: C-def›)
    finally show ?case .
  qed (simp-all add: swf-low-def R'-def)
  also have  $R' C = \text{extend-profile } (R(i := S))$ 
  unfolding  $\text{extend-profile-def } R'\text{-def } C\text{-def}$   $\text{fun-eq-iff}$  by auto
  finally show ?thesis

```

```

    by (simp add: swf-low-def)
  qed
qed

lemma majority-consistent-clone:
  assumes majority-consistent-swf agents alts swf
  shows majority-consistent-swf agents' alts swf-low
proof
  fix R assume R: new.is-pref-profile R linorder-on alts (majority R)
  interpret majority-consistent-swf agents alts swf by fact
  interpret R: pref-profile-linorder-wf agents' alts R by fact
  have swf-low R = swf (extend-profile R)
    unfolding swf-low-def ..
  also have ... = majority (extend-profile R)
    by (rule majority-consistent) (use R in <auto simp: majority-extend-profile>)
  also have ... = majority R
    by (rule majority-extend-profile) fact+
  finally show swf-low R = majority R .
qed

end

```

2.7.3 Decreasing the number of agents by an even number

Given an SWF for m alternatives and n agents, we can construct an SWF for m alternatives and $n - 2k$ agents by fixing some arbitrary ranking of alternatives and adding k clones of it to the input profile as well as k reversed clones.

This construction clearly violates anonymity and unanimity. It does however preserve strategyproofness (by a similar argument as for the cloning, but simpler) and majority consistency since the majority relation is preserved by our changes to the profile.

```

locale swf-reduce-agents-even =
  social-welfare-function agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +
  fixes agents1 agents2 :: 'agent set and dummy-ord :: 'alt relation
  assumes agents12:
    agents1  $\cup$  agents2  $\subset$  agents agents1  $\cap$  agents2 = {} card agents1 = card agents2
  assumes dummy-ord: linorder-on alts dummy-ord
begin

sublocale new: linorder-election agents - agents1 - agents2 alts
  by standard (use agents12 in auto)

definition extend-profile :: ('agent  $\Rightarrow$  'alt relation)  $\Rightarrow$  'agent  $\Rightarrow$  'alt relation where
  extend-profile R =
    ( $\lambda i$ . if  $i \in$  agents1 then dummy-ord else if  $i \in$  agents2 then  $\lambda x y$ . dummy-ord  $y x$  else R  $i$ )

lemma dummy-ord': linorder-on alts ( $\lambda x y$ . dummy-ord  $y x$ )
proof -

```

interpret *linorder-on alts dummy-ord*
 by (fact *dummy-ord*)
 show *?thesis ..*
qed

lemma *is-pref-profile-extend-profile [intro]:*

assumes *new.is-pref-profile R*
 shows *is-pref-profile (extend-profile R)*

proof

interpret *R: pref-profile-linorder-wf agents – agents1 – agents2 alts R*
 by fact

show *linorder-on alts (extend-profile R i) if i: i ∈ agents for i*

using *i agents12 dummy-ord dummy-ord' R.in-dom* by (auto simp: *extend-profile-def*)

show *extend-profile R i = (λ-. False) if i: i ∉ agents for i*

using *i R.not-in-dom agents12* by (auto simp: *extend-profile-def fun-eq-iff*)

qed

lemma *count-extend-profile:*

assumes *new.is-pref-profile R x ∈ alts y ∈ alts*

shows $\text{card } \{i \in \text{agents}. \text{extend-profile } R \ i \ x \ y\} =$
 $\text{card } \{i \in \text{agents} - \text{agents1} - \text{agents2}. R \ i \ x \ y\} +$
 $(\text{if } x = y \text{ then } 2 \text{ else } 1) * \text{card } \text{agents1}$

proof –

have *fin: finite agents1 finite agents2*

by (rule *finite-subset[OF - finite-agents]*; use *agents12 in blast*)

interpret *dummy-ord: linorder-on alts dummy-ord* by (fact *dummy-ord*)

have $\{i \in \text{agents}. \text{extend-profile } R \ i \ x \ y\} =$

$\{i \in \text{agents} - \text{agents1} - \text{agents2}. \text{extend-profile } R \ i \ x \ y\} \cup$

$\{i \in \text{agents1}. \text{extend-profile } R \ i \ x \ y\} \cup \{i \in \text{agents2}. \text{extend-profile } R \ i \ x \ y\}$

using *agents12* by *blast*

also have $\dots = \{i \in \text{agents} - \text{agents1} - \text{agents2}. R \ i \ x \ y\} \cup$

$(\{i \in \text{agents1}. \text{dummy-ord } x \ y\} \cup \{i \in \text{agents2}. \text{dummy-ord } y \ x\})$

using *agents12* by (auto simp: *extend-profile-def*)

also have $\text{card } \dots = \text{card } \{i \in \text{agents} - \text{agents1} - \text{agents2}. R \ i \ x \ y\} +$

$\text{card } (\{i \in \text{agents1}. \text{dummy-ord } x \ y\} \cup \{i \in \text{agents2}. \text{dummy-ord } y \ x\})$

by (rule *card-Un-disjoint*) (use *agents12 fin in auto*)

also have $\text{card } (\{i \in \text{agents1}. \text{dummy-ord } x \ y\} \cup \{i \in \text{agents2}. \text{dummy-ord } y \ x\}) =$

$(\text{if } \text{dummy-ord } x \ y \text{ then } \text{card } \text{agents1} \text{ else } 0) +$

$(\text{if } \text{dummy-ord } y \ x \text{ then } \text{card } \text{agents1} \text{ else } 0)$

by (subst *card-Un-disjoint*) (use *agents12 fin in auto*)

also have $\dots = (\text{if } x = y \text{ then } 2 \text{ else } 1) * \text{card } \text{agents1}$

using *dummy-ord.total[of x y] dummy-ord.antisymmetric[of x y] assms(2,3)* by auto

finally show *?thesis .*

qed

lemma *majority-extend-profile:*

assumes *new.is-pref-profile R*

shows *majority (extend-profile R) = majority R*

proof (*intro ext*)

```

fix  $x y :: 'alt$ 
interpret  $R$ : pref-profile-linorder-wf  $agents - agents1 - agents2$  alts  $R$  by fact
interpret  $R'$ : pref-profile-linorder-wf  $agents$  alts extend-profile  $R$ 
  using assms by auto
show majority (extend-profile  $R$ )  $x y = majority R x y$ 
proof (cases  $x \in alts \wedge y \in alts \wedge x \neq y$ )
  case  $xy$ : True
  show ?thesis
    using  $xy$  assms by (auto simp: R.majority-iff R'.majority-iff count-extend-profile)
  qed (auto simp: R.majority-iff' R'.majority-iff')
qed

```

definition *swf-low* :: (*'agent* \Rightarrow *'alt relation*) \Rightarrow *'alt relation* **where**
swf-low $R = swf$ (*extend-profile* R)

sublocale *new: social-welfare-function* $agents - agents1 - agents2$ *alts* *swf-low*
by standard (*auto simp: swf-low-def intro: swf-wf*)

lemma *kemeny-strategyproof-reduce*:

```

assumes kemeny-strategyproof-swf  $agents$  alts swf
shows kemeny-strategyproof-swf ( $agents - agents1 - agents2$ ) alts swf-low
proof
  fix  $R i S$ 
  assume  $R$ : new.is-pref-profile  $R$ 
  assume  $i$ :  $i \in agents - agents1 - agents2$ 
  assume  $S$ : linorder-on alts  $S$ 
  interpret kemeny-strategyproof-swf  $agents$  alts swf by fact
  interpret  $R$ : pref-profile-linorder-wf  $agents - agents1 - agents2$  alts  $R$  by fact

```

```

  have swap-dist-relation ( $R i$ ) (swf-low  $R$ ) =
    swap-dist-relation (extend-profile  $R i$ ) (swf (extend-profile  $R$ ))
    using  $i$  agents12 by (simp add: swf-low-def extend-profile-def)
  also have  $\dots \leq$  swap-dist-relation (extend-profile  $R i$ ) (swf ((extend-profile  $R$ )( $i := S$ )))
    by (rule kemeny-strategyproof) (use i agents12 S R in auto)
  also have (extend-profile  $R$ )( $i := S$ ) = extend-profile ( $R(i := S)$ )
    using  $i$  agents12 by (auto simp: extend-profile-def)
  also have swap-dist-relation (extend-profile  $R i$ ) (swf  $\dots$ ) =
    swap-dist-relation ( $R i$ ) (swf-low ( $R(i := S)$ ))
    using  $i$  agents12 by (simp add: swf-low-def extend-profile-def)
  finally show swap-dist-relation ( $R i$ ) (swf-low  $R$ )  $\leq$  swap-dist-relation ( $R i$ ) (swf-low ( $R(i := S)$ )) .
qed

```

lemma *majority-consistent-reduce*:

```

assumes majority-consistent-swf  $agents$  alts swf
shows majority-consistent-swf ( $agents - agents1 - agents2$ ) alts swf-low
proof
  fix  $R$  assume  $R$ : new.is-pref-profile  $R$  linorder-on alts (majority  $R$ )

```

```

interpret majority-consistent-swf agents alts swf by fact
interpret R: pref-profile-linorder-wf agents - agents1 - agents2 alts R by fact
have swf-low R = swf (extend-profile R)
  unfolding swf-low-def ..
also have ... = majority (extend-profile R)
  by (rule majority-consistent) (use R in ⟨auto simp: majority-extend-profile⟩)
also have ... = majority R
  by (rule majority-extend-profile) fact+
finally show swf-low R = majority R .
qed

```

end

end

3 Impossibility results

3.1 Infrastructure for SAT import and export

```

theory SWF-Impossibility-Automation
  imports SWF-Lowering SWF-Anonymous PAPP-Impossibility.SAT-Replay
begin

```

3.2 Automation for computing topological sortings

```

definition topo-sorts-aux-step :: ('a × 'a set) list ⇒ ('a × 'b set) list ⇒ 'a list list where
  topo-sorts-aux-step rel rel' =
    List.bind (map fst (filter (λ(-,ys). ys = {}) rel'))
      (λx. map ((#) x) (topo-sorts-aux (map (λ(y,ys). (y, Set.filter (λz. z ≠ x) ys))
        (filter (λ(y,-). y ≠ x) rel))))

```

lemma topo-sorts-aux-step-simps:

```

  topo-sorts-aux-step rel [] = []
  topo-sorts-aux-step rel ((x, insert y ys) # rel') = topo-sorts-aux-step rel rel'
  topo-sorts-aux-step rel ((x, {}) # rel') =
    map ((#) x) (topo-sorts-aux (map (λ(y,ys). (y, Set.filter (λz. z ≠ x) ys)) (filter (λ(y,-). y
    ≠ x) rel))) @
    topo-sorts-aux-step rel rel'
  by (simp-all add: topo-sorts-aux-step-def)

```

lemma topo-sorts-aux-Cons':

```

  fixes x xs defines rel ≡ x # xs
  shows topo-sorts-aux rel = topo-sorts-aux-step rel rel
  unfolding topo-sorts-aux-step-def assms
  by (subst topo-sorts-aux-Cons; unfold map-prod-def id-def) (rule refl)

```

context

begin

```

qualified fun dom-set :: 'a ⇒ 'a list ⇒ 'a set where
  dom-set x [] = {}
| dom-set x (y # ys) = (if x = y then {} else insert y (dom-set x ys))

qualified lemma dom-set-altdef:
  assumes distinct r x ∈ set r
  shows dom-set x r = {y. y >[of-ranking r] x}
  using assms
  by (induction r)
    (force simp: strongly-preferred-def of-ranking-Cons of-ranking-imp-in-set)+

qualified definition unanimity :: 'a list ⇒ 'a list multiset ⇒ ('a × 'a set) list where
  unanimity xs R = map (λx. (x, ⋂ r∈set-mset R. SWF-Impossibility-Automation.dom-set x r))
xs

end

locale anonymous-unanimous-kemenysp-swf =
  anonymous-swf agents alts swf +
  unanimous-swf agents alts swf +
  kemeny-strategyproof-swf agents alts swf
  for agents :: 'agent set and alts :: 'alt set and swf
begin

sublocale anonymous-unanimous-swf agents alts swf ..

sublocale anonymous-kemeny-strategyproof-swf agents alts swf ..

end

locale anonymous-unanimous-kemenysp-swf-explicit = anonymous-unanimous-kemenysp-swf agents
alts swf
  for agents :: 'agent set and alts :: 'alt set and swf +
  fixes agent-card :: nat and alts-list :: 'alt list
  assumes card-agents: card agents = agent-card
  assumes alts-list: mset alts-list = mset-set alts
begin

lemma distinct-alts-list: distinct alts-list
  using alts-list by (metis finite-alts mset-eq-mset-set-imp-distinct)

lemma alts-conv-alts-list: alts = set alts-list
  using alts-list by (metis finite-alts finite-set-mset-mset-set set-mset-mset)

lemma card-alts [simp]: card alts = length alts-list
  using alts-list by (metis size-mset size-mset-set)

```

fun (in $-$) *expand-ranking* :: 'a list \Rightarrow ('a \times 'a) list **where**
expand-ranking [] = []
| *expand-ranking* (x # xs) = map ($\lambda y. (y, x)$) xs @ *expand-ranking* xs

lemma (in $-$) *set-expand-ranking*:
distinct xs \implies set (*expand-ranking* xs) = $\{(x,y). x \neq y \wedge \text{of-ranking } xs \ x \ y\}$
by (*induction* xs) (*auto simp: of-ranking-Cons*)

definition *allowed-results* :: 'alt list multiset \Rightarrow 'alt list set **where**
allowed-results Rs = set (*topo-sorts-aux* (*SWF-Impossibility-Automation.unanimity* alts-list Rs))

lemmas *eval-allowed-results* =
allowed-results-def *topo-sorts-aux-Cons'* *Set-filter-insert-if* *SWF-Impossibility-Automation.dom-set.simps*
SWF-Impossibility-Automation.unanimity-def *disj-ac* *topo-sorts-aux-Nil* *topo-sorts-aux-step-simps*

lemma *aswf'-in-all-rankings*:
assumes *is-apref-profile'* R
defines A \equiv set (*topo-sorts-aux* (map ($\lambda x. (x, \{\})$) alts-list))
shows *aswf'* R \in A

proof –
have set (*topo-sorts-aux* (map ($\lambda x. (x, \{\})$) alts-list)) = *topo-sorts* alts ($\lambda x \ y. \text{False}$)
proof (*subst set-topo-sorts-aux, goal-cases*)
 case 3
 show ?*case*
 by (*rule arg-cong2*[*of - - - topo-sorts*]) (*auto simp: alts-conv-alts-list*)
qed (*use distinct-alts-list in* \langle *auto simp: o-def* \rangle)
also have ... = *permutations-of-set* alts
 by (*subst topo-sorts-correct*) (*auto simp: le-fun-def*)
finally have A = *permutations-of-set* alts
 unfolding A-def .
with *aswf'-wf*[*OF assms*(1)] **show** ?*thesis*
 by *simp*
qed

lemma *aswf'-in-allowed-results*:
assumes *is-apref-profile'* Rs
shows *aswf'* Rs \in *allowed-results* Rs
proof –
have Rs \neq {#}
 using *assms* **unfolding** *is-apref-profile'-def* **by** *force*
then obtain R **where** R: R \in # Rs
 by *auto*
then interpret R: *linorder-on* alts *of-ranking* R
 using *assms* **by** (*auto intro!*: *linorder-of-ranking simp: is-apref-profile'-def permutations-of-set-def*)

note *wf* = *is-apref-profile'-imp-is-apref-profile*[*OF assms*]

```

have aswf'  $R_s \in \text{ranking}$  ‘ of-ranking ‘ topo-sorts alts  $(\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. R x y)$ 
using unanimous-topo-sorts-Pareto-aswf[OF wf] unfolding aswf'-def by blast
also have ... =  $(\lambda x s. xs)$  ‘ topo-sorts alts  $(\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. R x y)$ 
unfolding image-image
proof (intro image-cong refl)
fix xs assume  $xs \in \text{topo-sorts alts} (\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. R x y)$ 
also have ... =  $\{xs \in \text{permutations-of-set alts.} (\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. R x y) \leq \text{of-ranking } xs\}$ 
by (subst topo-sorts-correct) (use is-apref-profile-unanimous-not-outside[OF wf] in auto)
finally show ranking (of-ranking xs) = xs
by (intro ranking-of-ranking) (auto simp: permutations-of-set-def)
qed
also have topo-sorts alts  $(\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. R x y) =$ 
topo-sorts alts  $(\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. x \neq y \wedge R x y)$ 
by (rule topo-sorts-cong) auto
also have ... = topo-sorts (set alts-list)  $(\lambda x y. \forall R \in \# \text{image-mset of-ranking } R_s. x \neq y \wedge R x y)$ 
by (subst alts-conv-alts-list) simp-all
also have ... = set (topo-sorts-aux (SWF-Impossibility-Automation.unanimity alts-list Rs))
proof (subst set-topo-sorts-aux, goal-cases)
case 1
thus ?case using distinct-alts-list
by (simp add: SWF-Impossibility-Automation.unanimity-def o-def)
next
case (2 x ys)
thus ?case using assms R.not-outside R.antisymmetric R unfolding is-apref-profile'-def
by (fastforce simp: SWF-Impossibility-Automation.unanimity-def SWF-Impossibility-Automation.dom-set-altdef
permutations-of-set-def alts-conv-alts-list strongly-preferred-def)
next
case 3
show ?case
proof (intro arg-cong2[of - - - topo-sorts] ext, goal-cases)
case (2 x y)
have  $(\forall R \in \# \text{image-mset of-ranking } R_s. x \neq y \wedge R x y) \longleftrightarrow$ 
 $(\forall R \in \# R_s. x \in \text{alts} \wedge y \in \text{SWF-Impossibility-Automation.dom-set } x R)$ 
unfolding set-image-mset ball-simps
proof (intro ball-cong refl)
fix S assume  $S: S \in \# R_s$ 
interpret S: linorder-on alts of-ranking S using assms S
by (auto simp: is-apref-profile'-def permutations-of-set-def intro!: linorder-of-ranking)
have distinct S set S = alts
using S assms by (auto simp: is-apref-profile'-def permutations-of-set-def)
thus  $(x \neq y \wedge \text{of-ranking } S x y) = (x \in \text{alts} \wedge y \in \text{SWF-Impossibility-Automation.dom-set } x S)$ 
using S.not-outside[of x y] S.antisymmetric[of x y]
by (auto simp: SWF-Impossibility-Automation.dom-set-altdef strongly-preferred-def)
qed
also have ...  $\longleftrightarrow (\exists ys. (x, ys) \in \text{set } (\text{SWF-Impossibility-Automation.unanimity alts-list}$ 

```

```

Rs)  $\wedge$   $y \in ys$ )
  unfolding SWF-Impossibility-Automation.unanimity-def using R
  by (auto simp: image-iff simp flip: alts-conv-alts-list)
  finally show ?case .
qed (auto simp: SWF-Impossibility-Automation.unanimity-def)
qed
also have ... = allowed-results Rs
  unfolding allowed-results-def ..
  finally show ?thesis by simp
qed

lemma is-apref-profile'-iff:
  is-apref-profile' Rs  $\longleftrightarrow$  (size Rs = agent-card  $\wedge$  ( $\forall R \in \#Rs$ . mset R = mset alts-list))
  unfolding is-apref-profile'-def card-agents alts-list
  by (subst mset-eq-mset-set-iff) simp-all

end

```

3.3 Automation for strategyproofness

```

lemma (in anonymous-unanimous-kemenysp-swf-explicit) kemeny-strategyproof-aswf'-aux:
  assumes is-apref-profile' R1 is-apref-profile' R2
  assumes inversion-number S1' = d1 inversion-number S2' = d2
  assumes map (index T) S1 = S1' map (index T) S2 = S2'
  assumes R12: add-mset T' R1  $\equiv$  add-mset T R2
  assumes d2 < d1
  shows aswf' R1  $\neq$  S1  $\vee$  aswf' R2  $\neq$  S2
proof (rule ccontr)
  assume *:  $\neg$ (aswf' R1  $\neq$  S1  $\vee$  aswf' R2  $\neq$  S2)
  hence S12: S1  $\in$  permutations-of-set alts S2  $\in$  permutations-of-set alts
    using assms(1,2) aswf'-wf by blast+
  have T  $\neq$  T'
    using assms * by fastforce
  hence T  $\in \#$  R1 using R12
    by (metis insert-noteq-member)
  have T'  $\in \#$  R2
    using R12 by (metis  $\langle T \neq T' \rangle$  insert-noteq-member)
  have R1 - R2 = {#T#}
    using R12  $\langle T \in \# R1 \rangle$   $\langle T \neq T' \rangle$   $\langle T' \in \# R2 \rangle$ 
      add-diff-cancel-left add-mset-remove-trivial[of T R2]
      add-mset-remove-trivial[of T' R1 - {#T#}]
      diff-union-swap insert-DiffM2[of T R1] insert-DiffM2[of T' R2] zero-diff
    by metis

  have T: T  $\in$  permutations-of-set alts
    using assms(1)  $\langle T \in \# R1 \rangle$  by (auto simp: is-apref-profile'-def)
  have swap-dist T S1 = d1 swap-dist T S2 = d2
    by (subst swap-dist-conv-inversion-number;
      use S12 T assms in  $\langle$ simp add: permutations-of-set-def $\rangle$ ; fail)+

```

with *assms* * **show** *False*
using *kemeny-strategyproof-aswf*[*of R1 R2 S2 S1*] $\langle R1 - R2 = \{\#T\# \rangle$ **by** *simp*
qed

lemma (**in** *anonymous-unanimous-kemenysp-swf-explicit*) *kemeny-strategyproof-aswf'-no-obtain-optimal*:
assumes *is-apref-profile' R is-apref-profile' R' add-mset S R' \equiv add-mset S' R*
shows *aswf' R = S \vee aswf' R' \neq S*
using *kemeny-strategyproof-aswf'-no-obtain-optimal*[*of R R' S S'*] *assms* **by** *simp*

3.4 Automation for majority consistency

fun *majority-rel-mset-aux* :: '*a list multiset* \Rightarrow '*a list* \Rightarrow *bool* **where**
majority-rel-mset-aux *Rs* [] \longleftrightarrow *True*
| *majority-rel-mset-aux* *Rs* (*x # xs*) \longleftrightarrow
 $(\forall y \in \text{set } xs. 2 * \text{size} (\text{filter-mset} (\lambda R. \text{of-ranking } R \ y \ x) \ Rs) > \text{size } Rs) \wedge$
majority-rel-mset-aux *Rs* *xs*

fun *majority-rel-list-aux* :: '*a list list* \Rightarrow '*a list* \Rightarrow *bool* **where**
majority-rel-list-aux *Rs* [] \longleftrightarrow *True*
| *majority-rel-list-aux* *Rs* (*x # xs*) \longleftrightarrow
 $\text{list-all} (\lambda y. 2 * \text{length} (\text{filter} (\lambda R. \text{of-ranking } R \ y \ x) \ Rs) > \text{length } Rs) \ xs \wedge$
majority-rel-list-aux *Rs* *xs*

lemma *majority-rel-mset-aux-mset*:
majority-rel-mset-aux (*mset* *Rs*) *ys* \longleftrightarrow *majority-rel-list-aux* *Rs* *ys*
by (*induction* *ys*) (*simp-all* *add: list.pred-set flip: mset-filter*)

lemma *majority-rel-mset-aux-correct*:
assumes $\bigwedge R. R \in \# \ Rs \implies \text{distinct } R \wedge \text{set } R = A \ Rs \neq \{\#\} \text{ distinct } zs \text{ set } zs \subseteq A$
defines *Rs'* \equiv *image-mset of-ranking* *Rs*
defines *M* \equiv *majority-mset* *Rs'*
shows *majority-rel-mset-aux* *Rs* *zs* \longleftrightarrow
 $(\forall x \in \text{set } zs. \forall y \in \text{set } zs. x \prec[M] y \longleftrightarrow x \prec[\text{of-ranking } zs] y)$
(is - \longleftrightarrow ?rhs *zs*)
using *assms*(3,4)
proof (*induction* *zs*)
case (*Cons* *z* *zs*)
define *P* **where** *P* = $(\lambda zs \ x \ y. x \prec[M] y \longleftrightarrow x \prec[\text{of-ranking } zs] y)$
have *R*: *linorder-on* *A* *R* **if** *R* $\in \#$ *Rs'* **for** *R*
using *assms*(1) *that linorder-of-ranking unfolding Rs'-def by fastforce*
have *R'*: *preorder-on* *A* *R* **if** *R* $\in \#$ *Rs'* **for** *R*
using *R[OF that] linorder-on-def order-on-def by blast*
have *: *P* (*z* $\#$ *zs*) *z* *z*
using *Cons.prem*s *majority-mset-refl*[*of Rs' A z*] *R'* *assms*(2)
unfolding *P-def* *strongly-preferred-def* *M-def* **by** (*auto simp: of-ranking-Cons*)
have *less-iff*: $x \prec[M] y \longleftrightarrow \text{size } Rs' < 2 * \text{size } \{\#R \in \# Rs'. R \ x \ y \#\}$
if *x* \in *A* *y* \in *A* *x* \neq *y* **for** *x* *y*
using *strongly-preferred-majority-mset-iff-gt*[*of Rs' A, OF R*] *that assms*(2)
by (*simp* *add: Rs'-def M-def strongly-preferred-def filter-mset-image-mset not-le*)

have *majority-rel-mset-aux* Rs $(z \# zs) \longleftrightarrow$
 $(\forall y \in set\ zs. size\ Rs' < 2 * size\ \{\#R \in \# Rs'. R\ y\ z\ \}) \wedge$
 $majority-rel-mset-aux\ Rs\ zs$
by (*simp add: Rs'-def filter-mset-image-mset*)
also have $(\forall y \in set\ zs. size\ Rs' < 2 * size\ \{\#R \in \# Rs'. R\ y\ z\ \}) \longleftrightarrow$
 $(\forall y \in set\ zs. P\ (z \# zs)\ y\ z)$
by (*intro ball-cong refl*)
(use less-iff Cons.prem in <auto simp: P-def of-ranking-strongly-preferred-Cons-iff>)
also have *majority-rel-mset-aux* $Rs\ zs \longleftrightarrow$
 $(\forall x \in set\ zs. \forall y \in set\ zs. x \prec[M]\ y = x \prec[of-ranking\ zs]\ y)$
by (*rule Cons.IH*) (*use Cons.prem in auto*)
also have $\dots \longleftrightarrow (\forall x \in set\ zs. \forall y \in set\ zs. P\ zs\ x\ y)$
unfolding *P-def ..*
also have $\dots \longleftrightarrow (\forall x \in set\ zs. \forall y \in set\ zs. P\ (z \# zs)\ x\ y)$
using *Cons.prem*
by (*intro ball-cong refl*) (*auto simp: P-def of-ranking-strongly-preferred-Cons-iff*)
also have $(\forall y \in set\ zs. P\ (z \# zs)\ y\ z) \longleftrightarrow (\forall x \in set\ zs. P\ (z \# zs)\ x\ z) \wedge (\forall x \in set\ zs. P\ (z$
 $\# zs)\ z\ x)$
proof (*intro iffI conjI*)
assume $*$: $\forall y \in set\ zs. P\ (z \# zs)\ y\ z$
show $\forall y \in set\ zs. P\ (z \# zs)\ z\ y$
proof
fix y **assume** $y: y \in set\ zs$
have [*simp*]: $y \neq z\ z \neq y$
using *Cons.prem y* **by** *auto*
have $P\ (z \# zs)\ y\ z$
using $y *$ **by** *blast*
moreover have $y \prec[of-ranking\ (z \# zs)]\ z$
using *Cons.prem y of-ranking-imp-in-set[of zs z y]*
by (*auto simp: strongly-preferred-def of-ranking-Cons*)
ultimately have $y \prec[M]\ z$
by (*auto simp: P-def*)
hence $\neg z \prec[M]\ y$
by (*auto simp: strongly-preferred-def*)
moreover have $\neg z \prec[of-ranking\ (z \# zs)]\ y$
using *Cons.prem y of-ranking-imp-in-set[of zs z y]*
by (*auto simp: strongly-preferred-def of-ranking-Cons*)
ultimately show $P\ (z \# zs)\ z\ y$
by (*auto simp: P-def*)
qed
qed blast+
also have $\dots \wedge (\forall x \in set\ zs. \forall y \in set\ zs. P\ (z \# zs)\ x\ y) \longleftrightarrow$
 $(\forall x \in set\ (z\ \# zs). \forall y \in set\ (z\ \# zs). P\ (z \# zs)\ x\ y)$
unfolding *list.set* **using** $*$ **by** *blast*
finally show *?case unfolding P-def .*
qed simp-all

lemma *majority-rel-mset-aux-correct'*:

assumes $\bigwedge R. R \in \# Rs \implies \text{distinct } R \wedge \text{set } R = A \text{ } Rs \neq \{\#\}$
assumes $\text{set } S = A \text{ } \text{distinct } S$
assumes *majority-rel-mset-aux* $Rs \ S$
shows *majority-rel-mset* $Rs \ S$
proof –
define Rs' **where** $Rs' = \text{image-mset of-ranking } Rs$
define M **where** $M = \text{majority-mset } Rs'$
have Rs' : *linorder-on* $A \ R$ **if** $R \in \# Rs'$ **for** R
using *that assms unfolding* Rs' -*def* **by** (*auto intro: linorder-of-ranking*)
have Rs'' : *preorder-on* $A \ R$ **if** $R \in \# Rs'$ **for** R
using Rs' [*OF that*] *linorder-on-def order-on-def* **by** *blast*
have $x \in A \wedge y \in A$ **if** $M \ x \ y$ **for** $x \ y$
using *majority-mset-not-outside*[*of* $Rs' \ x \ y \ A$] *that* Rs'' **unfolding** M -*def* **by** *blast*
moreover have $x \in A \wedge y \in A$ **if** *of-ranking* $S \ x \ y$ **for** $x \ y$
using *of-ranking-imp-in-set*[*OF that*] *assms* **by** *auto*
moreover have $\forall x \in A. \forall y \in A. x \prec[M] \ y \longleftrightarrow x \prec[\text{of-ranking } S] \ y$
using *majority-rel-mset-aux-correct*[*OF assms*(1,2,4)] *assms*(3,5) **unfolding** M -*def* Rs' -*def*
by *blast*
hence $\forall x \in A. \forall y \in A. x \preceq[M] \ y \longleftrightarrow x \preceq[\text{of-ranking } S] \ y$
unfolding *strongly-preferred-def*
by (*metis* M -*def* Rs'' Rs' -*def* *assms*(2,3,4) *image-mset-is-empty-iff majority-mset-refl nle-le*
nth-index of-ranking-altdef)
ultimately have $M = \text{of-ranking } S$
by *blast*
thus *?thesis*
unfolding *majority-rel-mset-def* **using** $\langle \text{distinct } S \rangle$ **by** (*simp add: M-def Rs'-def*)
qed

context *social-welfare-function-explicit*
begin

lemma *majority-rel-list-aux-imp-majority-rel-mset*:

assumes *prefs-from-rankings-wf* R *majority-rel-list-aux* $R \ ys$ $\text{mset } ys = \text{mset } \text{alts-list}$

shows *majority-rel-mset* $(\text{mset } R) \ ys$

proof –

have *distinct* ys

using $\langle \text{mset } ys = \rightarrow \rangle$ **by** (*metis* *alts-list finite-altts mset-eq-mset-set-imp-distinct*)

have $\text{set } ys = \text{alts}$

using $\langle \text{mset } ys = \rightarrow \rangle$ **by** (*metis* *alts-list finite-altts finite-set-mset-mset-set set-mset-mset*)

note $ys = \langle \text{distinct } ys \rangle \langle \text{set } ys = \text{alts} \rangle$

show *?thesis*

proof (*rule* *majority-rel-mset-aux-correct*'[**where** $A = \text{alts}$])

show $\text{mset } R \neq \{\#\}$

using *assms*(1) *agents-conv-agents-list* **unfolding** *prefs-from-rankings-wf-def* **by** *force*

show $\text{distinct } S \wedge \text{set } S = \text{alts}$ **if** $S \in \# \text{mset } R$ **for** S

using *that assms*(1) **by** (*auto simp: prefs-from-rankings-wf-def list.pred-set mset-eq-altts-list-iff*)

qed (*use* ys *assms* **in** $\langle \text{simp-all add: majority-rel-mset-aux-mset} \rangle$)

qed

lemma *majority-prefs-from-rankings-eq-of-ranking-aux:*

assumes *prefs-from-rankings-wf* R *majority-rel-list-aux* R ys $mset\ ys = mset\ alts\ list$

shows *majority* (*prefs-from-rankings* R) = *of-ranking* ys

using *majority-rel-list-aux-imp-majority-rel-mset* *majority-prefs-from-rankings-eq-of-ranking*
assms

by *metis*

end

lemma (in *majcons-kstratproof-swf-explicit*) *majority-consistent-swf'-aux:*

assumes *prefs-from-rankings-wf* xss $mset\ ys = mset\ alts\ list$

assumes *majority-rel-list-aux* xss ys

shows $swf'\ xss = ys$

proof (rule *majority-consistent-swf'*)

show *majority-rel-mset* ($mset\ xss$) ys

proof (rule *majority-rel-mset-aux-correct'*)

show $distinct\ R \wedge set\ R = alts$ **if** $R \in \#$ $mset\ xss$ **for** R

using *assms(1)* *that*

by (*auto simp: prefs-from-rankings-wf-def* *mset-eq-alts-list-iff* *list.pred-set*)

next

show *majority-rel-mset-aux* ($mset\ xss$) ys

using *assms(3)* **by** (*subst* *majority-rel-mset-aux-mset*)

next

show $mset\ xss \neq \{\#\}$

using *assms(1)* **unfolding** *prefs-from-rankings-wf-def* **by** *auto*

next

show $set\ ys = alts$

using *assms(2)* *alts-conv-alts-list* *mset-eq-setD* **by** *blast*

next

show $distinct\ ys$

using *assms(2)* *distinct-alts-list* *mset-eq-imp-distinct-iff* **by** *blast*

qed

qed *fact+*

lemma (in *majcons-weak-kstratproof-swf-explicit*) *majority-consistent-kemeny-strategyproof-swf'-aux:*

assumes *prefs-from-rankings-wf* $R1$ $i < card\ agents$

assumes $mset\ zs = mset\ alts\ list$ $mset\ ys = mset\ alts\ list$

assumes $xs = R1\ !\ i$ *majority-rel-list-aux* ($R1[i := zs]$) ys

shows $swap\ dist\ xs\ (swf'\ R1) \leq swap\ dist\ xs\ ys$

using *majority-consistent-kemeny-strategyproof-swf'* *assms*

using *majority-rel-list-aux-imp-majority-rel-mset* *prefs-from-rankings-wf-update* **by** *presburger*

lemma *permutations-of-set-aux-list-Nil:* *permutations-of-set-aux-list* $acc\ [] = [acc]$

by (*subst* *permutations-of-set-aux-list.simps*) *simp-all*

```

lemma permutations-of-set-aux-list-Cons:
  permutations-of-set-aux-list acc (x#xs) =
    permutations-of-set-aux-list (x # acc) xs @ List.bind xs
    (λxa. permutations-of-set-aux-list (xa # acc) (if xa = x then xs else x # remove1 xa xs))
  by (subst permutations-of-set-aux-list.simps) simp-all

```

```

ML-file <sat-problem.ML>
ML-file <swf-util.ML>
ML-file <anon-unan-stratproof-impossibility.ML>
ML-file <majcons-stratproof-impossibility.ML>

```

```

end
theory Anon-Unan-Stratproof-Impossibility
  imports SWF-Impossibility-Automation
begin

```

3.5 For 5 alternatives and 2 agents

We prove the impossibility for $m = 5$ and $n = 2$ via the SAT encoding using a fixed list of 198 profiles. For symmetry breaking, we assume that the profile $(abcde, acbed)$ is mapped to the ranking $abcde$. This assumption will be justified later on by picking the values of a, b, c, d, e accordingly.

```

external-file sat-data/kemeny-profiles-5-2.xz
external-file sat-data/kemeny-5-2.grat.xz

```

```

locale anonymous-unanimous-kemenysp-swf-explicit-5-2 =
  anonymous-unanimous-kemenysp-swf-explicit agents alts swf 2 [a,b,c,d,e]
  for agents :: 'agent set and alts :: 'alt set and swf and a b c d e +
  assumes symmetry-breaking: aswf' {# [a,b,c,d,e], [a,c,b,e,d] #} = [a,b,c,d,e]
begin

```

```

local-setup <fn lthy =>
  let
    val params = {
      name = kemeny-5-2,
      locale-thm = @{thm anonymous-unanimous-kemenysp-swf-explicit-axioms},
      profile-file = SOME path <sat-data/kemeny-profiles-5-2.xz>,
      sp-file = NONE,
      grat-file = path <sat-data/kemeny-5-2.grat.xz>,
      extra-clauses = @{thms symmetry-breaking}
    }
  val thm =
    Goal.prove-future lthy [] [] prop <False>
    (fn {context, ...} =>
      HEADGOAL (resolve-tac context [Anon-Unan-Stratproof-Impossibility.derive-false context

```

```

params]))
in
  Local-Theory.note ((binding ⟨contradiction⟩, []), [thm]) lthy |> snd
end
›

```

end

We now get rid of the symmetry-breaking assumption by choosing an appropriate permutation of the five alternatives.

locale *anonymous-unanimous-kemenysp-swf-5-2* = *anonymous-unanimous-kemenysp-swf agents alts swf*

```

  for agents :: 'agent set and alts :: 'alt set and swf +
  assumes card-agents: card agents = 2
  assumes card-alts: card alts = 5

```

begin

sublocale *anonymous-unanimous-swf agents alts swf* ..

sublocale *anonymous-kemeny-strategyproof-swf agents alts swf* ..

lemma *symmetry-breaking-aux1*:

```

  assumes distinct: distinct [a,b,c,d,e] and alts-eq: alts = {a,b,c,d,e}

```

```

  defines R ≡ {# [a,b,c,d,e], [a,c,b,e,d] #}

```

```

  assumes R: aswf' R = [a,c,b,d,e]

```

```

  shows aswf' {# [a,b,c,e,d], [a,c,b,d,e] #} ∈ {[a,b,c,e,d], [a,c,b,d,e]}

```

proof –

```

  have alts-eq': alts = set [a,b,c,d,e]

```

```

  by (simp add: alts-eq)

```

```

  have [simp]: a ≠ b a ≠ c a ≠ d a ≠ e b ≠ a b ≠ c b ≠ d b ≠ e

```

```

             c ≠ a c ≠ b c ≠ d c ≠ e d ≠ a d ≠ b d ≠ c d ≠ e

```

```

             e ≠ a e ≠ b e ≠ c e ≠ d

```

```

  using distinct by (simp-all add: eq-commute)

```

```

  interpret anonymous-unanimous-kemenysp-swf-explicit agents alts swf 2 [a,b,c,d,e]

```

```

  by standard (simp-all add: alts-eq card-agents)

```

```

  have R-wf: is-apref-profile' R

```

```

  unfolding is-apref-profile'-def by (auto simp: card-agents permutations-of-set-def alts-eq
R-def)

```

```

  define R2 where R2 = {# [a,c,b,d,e], [a,c,b,e,d] #}

```

```

  define R3 where R3 = {# [a,b,c,d,e], [a,c,b,d,e] #}

```

```

  define R4 where R4 = {# [a,b,c,e,d], [a,c,b,d,e] #}

```

```

  note R-defs = R-def R2-def R3-def R4-def

```

```

  have wf: is-apref-profile' R is-apref-profile' R2 is-apref-profile' R3 is-apref-profile' R4

```

```

  by (simp-all add: is-apref-profile'-iff R-defs add-mset-commute)

```

```

  have R2: aswf' R2 = [a,c,b,d,e]

```

proof –

have $aswf' R2 = [a,c,b,d,e] \vee aswf' R2 = [a,c,b,e,d]$
using $aswf'-in-allowed-results[OF wf(2)]$ **unfolding** $R-defs$
by ($simp$ $add: eval-allowed-results$ $del: Set.filter-eq$)
moreover have $aswf' R2 \neq [a,c,b,e,d] \vee aswf' R \neq [a,c,b,d,e]$
by ($intro$ $kemeny-strategyproof-aswf'-strong$ wf)
($simp-all$ $add: R-defs$ $insert-commute$ $swap-dist-code'$ $inversion-number-Cons$)
ultimately show $?thesis$ **using** R
by $blast$
qed

have $R3: aswf' R3 = [a,c,b,d,e]$
proof –
have $aswf' R3 = [a,b,c,d,e] \vee aswf' R3 = [a,c,b,d,e]$
using $aswf'-in-allowed-results[OF wf(3)]$ **unfolding** $R-defs$
by ($simp$ $add: eval-allowed-results$ $del: Set.filter-eq$)
moreover have $aswf' R3 \neq [a,b,c,d,e] \vee aswf' R \neq [a,c,b,d,e]$
by ($intro$ $kemeny-strategyproof-aswf'-strong$ wf)
($simp-all$ $add: R-defs$ $insert-commute$ $swap-dist-code'$ $inversion-number-Cons$)
ultimately show $?thesis$ **using** R
by $blast$
qed

show $?thesis$
proof –
have $aswf' R4 \in \{[a,b,c,d,e], [a,c,b,e,d], [a,b,c,e,d], [a,c,b,d,e]\}$
using $aswf'-in-allowed-results[OF wf(4)]$ **unfolding** $R-defs$
by ($simp$ $add: eval-allowed-results$ $del: Set.filter-eq$)
moreover have $aswf' R4 \neq [a,b,c,d,e] \vee aswf' R3 \neq [a,c,b,d,e]$
by ($intro$ $kemeny-strategyproof-aswf'-strong$ wf)
($simp-all$ $add: R-defs$ $insert-commute$ $swap-dist-code'$ $inversion-number-Cons$)
moreover have $aswf' R4 \neq [a,c,b,e,d] \vee aswf' R2 \neq [a,c,b,d,e]$
by ($intro$ $kemeny-strategyproof-aswf'-strong$ wf)
($simp-all$ $add: R-defs$ $insert-commute$ $swap-dist-code'$ $inversion-number-Cons$)
ultimately show $?thesis$ **using** $R2$ $R3$ **unfolding** $R4-def$
by $blast$
qed
qed

lemma $symmetry-breaking-aux2$:
obtains $abcde$ **where**
 $distinct$ $abcde$ $alts = set$ $abcde$ $length$ $abcde = 5$
 $case$ $abcde$ of $[a,b,c,d,e] \Rightarrow aswf' \{\#$ $[a,b,c,d,e], [a,c,b,e,d] \# \} = [a,b,c,d,e]$
proof –
obtain $abcde$ **where** $abcde: distinct$ $abcde$ set $abcde = alts$
using $finite-distinct-list$ **by** $blast$
have $length$ $abcde = 5$
using $card-alts$ $abcde$ $distinct-card$ **by** $metis$
obtain a b c d e **where** $abcde-expand: abcde = [a,b,c,d,e]$
using $\langle length$ $abcde = 5 \rangle$ **by** ($force$ $simp: eval-nat-numeral$ $length-Suc-conv$)

```

have [simp]: a ≠ b a ≠ c a ≠ d a ≠ e b ≠ a b ≠ c b ≠ d b ≠ e
             c ≠ a c ≠ b c ≠ d c ≠ e d ≠ a d ≠ b d ≠ c d ≠ e
             e ≠ a e ≠ b e ≠ c e ≠ d
  using abcde(1) unfolding abcde-expand by (simp-all add: eq-commute)
have alts-eq: alts = {a,b,c,d,e}
  unfolding abcde(2) [symmetric] abcde-expand by simp
interpret anonymous-unanimous-kemenysp-swf-explicit agents alts swf 2 [a,b,c,d,e]
  by standard (simp-all add: alts-eq card-agents)

define R where R = {# [a,b,c,d,e], [a,c,b,e,d] #}
have R-wf: is-apref-profile' R
  unfolding R-def is-apref-profile'-def
  by (auto simp: card-agents abcde-expand simp flip: abcde(2) intro!: linorder-of-ranking)

have aswf' R ∈ {[a,b,c,d,e], [a,c,b,e,d]} ∨ aswf' R ∈ {[a,b,c,e,d], [a,c,b,d,e]}
  using aswf'-in-allowed-results[OF R-wf] unfolding R-def
  by (simp add: eval-allowed-results del: Set.filter-eq)

thus ?thesis
proof
  assume *: aswf' R ∈ {[a,b,c,d,e], [a,c,b,e,d]}
  show ?thesis
    by (rule that[of aswf' R])
      (use * in ⟨unfold R-def, auto simp add: alts-eq insert-commute add-mset-commute⟩)
next
  assume aswf' R ∈ {[a,b,c,e,d], [a,c,b,d,e]}
  hence aswf' R = [a,b,c,e,d] ∨ aswf' R = [a,c,b,d,e]
    by blast
  thus ?thesis
  proof
    assume *: aswf' R = [a,c,b,d,e]
    have **: aswf' {#[a,b,c,e,d], [a,c,b,d,e]#} ∈ {[a,b,c,e,d], [a,c,b,d,e]}
      by (rule symmetry-breaking-aux1 [of a b c d e])
      (use * in ⟨simp-all add: R-def add-mset-commute alts-eq⟩)
    show ?thesis
      by (rule that [of aswf' {#[a,b,c,e,d], [a,c,b,d,e]#}])
      (use ** in ⟨auto simp: alts-eq add-mset-commute insert-commute⟩)
  next
    assume *: aswf' R = [a,b,c,e,d]
    have **: aswf' {#[a,c,b,d,e], [a,b,c,e,d]#} ∈ {[a,c,b,d,e], [a,b,c,e,d]}
      by (rule symmetry-breaking-aux1 [of a c b e d])
      (use * in ⟨simp-all add: R-def add-mset-commute alts-eq insert-commute⟩)
    show ?thesis
      by (rule that [of aswf' {#[a,b,c,e,d], [a,c,b,d,e]#}])
      (use ** in ⟨auto simp: alts-eq add-mset-commute insert-commute⟩)
  qed
qed
qed
qed

```

lemma *contradiction: False*

proof –

obtain *abcde* **where** *abcde*:

distinct abcde alts = set abcde length abcde = 5

case abcde of [a,b,c,d,e] ⇒ aswf' {# [a,b,c,d,e], [a,c,b,e,d] #} = [a,b,c,d,e]

using *symmetry-breaking-aux2* .

from $\langle \text{length } abcde = 5 \rangle$ **obtain** *a b c d e* **where** *abcde-expand: abcde = [a,b,c,d,e]*

by (*auto simp: length-Suc-conv eval-nat-numeral*)

interpret *anonymous-unanimous-kemenyssp-swf-explicit-5-2* *agents alts swf a b c d e*

proof

show *aswf' {#[a, b, c, d, e], [a, c, b, e, d]#} = [a, b, c, d, e]*

using *abcde unfolding abcde-expand* **by** *simp*

qed (*use abcde(1) in <simp-all add: card-agents abcde abcde-expand eq-commute>*)

show *?thesis*

by (*fact contradiction*)

qed

end

Finally, we employ the usual construction of padding with dummy alternatives and cloning voters to extend the impossibility to any setting with $m \geq 5$ and n even.

theorem (*in anonymous-unanimous-kemenyssp-swf*) *impossibility:*

assumes *even (card agents) and card alts ≥ 5*

shows *False*

proof –

have *card agents > 0*

using *assms(1) finite-agents nonempty-agents* **by** *fastforce*

with *assms(1)* **have** *card agents ≥ 2*

by *presburger*

obtain *agents'* **where** *agents' ⊆ agents card agents' = 2*

using $\langle \text{card agents} \geq 2 \rangle$ **by** (*meson obtain-subset-with-card-n*)

have [*simp*]: *agents' ≠ {}*

using *agents'* **by** *auto*

obtain *alts'* **where** *alts' ⊆ alts card alts' = 5*

using $\langle \text{card alts} \geq 5 \rangle$ **by** (*meson obtain-subset-with-card-n*)

obtain *dummy-alts* **where** *alts-list': mset dummy-alts = mset-set (alts – alts')*

using *ex-mset* **by** *blast*

obtain *unclone* **where** *cloning agents' agents unclone*

using *cloning-exists[of agents' agents] agents' <even (card agents)> finite-agents* **by** *auto*

interpret *cloning agents' agents unclone* **by** *fact*

interpret *split: swf-split-agents agents alts swf agents' unclone ..*

interpret *new1: anonymous-swf agents' alts split.swf-low*

by (*rule split.anonymous-clone*) *unfold-locales*

interpret *new1: unanimous-swf agents' alts split.swf-low*

by (*rule split.unanimous-clone*) *unfold-locales*

interpret *new1: kemeny-strategyproof-swf agents' alts split.swf-low*

by (*rule split.kemeny-strategyproof-clone*) *unfold-locales*

```

interpret restrict: unanimous-swf-restrict-alts agents' alts split.swf-low dummy-alts alts'
proof
  show finite alts'
    using alts'(1) finite-subset by blast
  show mset-set alts = mset dummy-alts + mset-set alts'
    by (simp add: ⟨finite alts'⟩ alts'(1) alts-list' mset-set-Diff)
  show alts' ≠ {}
    using alts'(2) by fastforce
qed
interpret new2: anonymous-swf agents' alts' restrict.swf-low
  by (rule restrict.anonymous-restrict) unfold-locales
interpret new2: unanimous-swf agents' alts' restrict.swf-low ..
interpret new2: kemeny-strategyproof-swf agents' alts' restrict.swf-low
  by (rule restrict.kemeny-strategyproof-restrict) unfold-locales

interpret new2: anonymous-unanimous-kemenysp-swf-5-2 agents' alts' restrict.swf-low
  by standard fact+
show False
  by (fact new2.contradiction)
qed

```

3.6 For 4 alternatives and 4 agents

We now similarly show the impossibility for $m = n = 4$. The main difference now is that the number of profiles involved is much larger, namely 9900, so the approach of simply generating all strategyproofness clauses that arise between these profiles is no longer feasible.

Instead we work with an explicit list of the required 254269 strategyproofness clauses that was extracted from an unsatisfiable core found with MUSer2.

The symmetry-breaking assumption we use this time is that the profile where two agents report $abcd$ and the other two report $badc$ is mapped to $abcd$.

```

external-file sat-data/kemeny-sp-4-4.xz
external-file sat-data/kemeny-4-4.grat.xz

```

```

locale anonymous-unanimous-kemenysp-swf-explicit-4-4 =
  anonymous-unanimous-kemenysp-swf-explicit agents alts swf 4 [a,b,c,d]
  for agents :: 'agent set and alts :: 'alt set and swf and a b c d +
  assumes symmetry-breaking: aswf' {# [a,b,c,d], [a,b,c,d], [b,a,d,c], [b,a,d,c] #} = [a,b,c,d]
begin

```

```

local-setup ⟨fn lthy =>
  let
    val params = {
      name = kemeny-4-4,
      locale-thm = @{thm anonymous-unanimous-kemenysp-swf-explicit-axioms},

```

```

    profile-file = NONE,
    sp-file = SOME path ⟨sat-data/kemeny-sp-4-4.xz⟩,
    grat-file = path ⟨sat-data/kemeny-4-4.grat.xz⟩,
    extra-clauses = @{thms symmetry-breaking}
  }
  val thm =
    Goal.prove-future lthy [] [] prop ⟨False⟩
      (fn {context, ...} =>
        HEADGOAL (resolve-tac context [Anon-Unan-Stratproof-Impossibility.derive-false context
params]))
  in
    Local-Theory.note ((binding ⟨contradiction⟩, []), [thm]) lthy |> snd
  end
end

```

We again get rid of the symmetry-breaking assumption. The argument is almost exactly the same one as before, except that we remove the alternative a and all agents get cloned. Consequently, the arguments involving strategyproofness have to use the stronger notion of strategyproofness considering simultaneous deviations by clones.

```

locale anonymous-unanimous-kemeny-sp-swf-4-4 = anonymous-unanimous-kemeny-sp-swf agents
alts swf

```

```

  for agents :: 'agent set and alts :: 'alt set and swf +
  assumes card-agents: card agents = 4
  assumes card-alts: card alts = 4

```

```

begin

```

```

sublocale anonymous-unanimous-swf agents alts swf ..

```

```

sublocale anonymous-kemeny-strategyproof-swf agents alts swf ..

```

```

lemma symmetry-breaking-aux1:

```

```

  assumes distinct: distinct [a,b,c,d] and alts-eq: alts = {a,b,c,d}

```

```

  defines R ≡ repeat-mset 2 {# [a,b,c,d], [b,a,d,c] #}

```

```

  assumes R: aswf' R = [b,a,c,d]

```

```

  shows aswf' (repeat-mset 2 {# [a,b,d,c], [b,a,c,d] #}) ∈ {[a,b,d,c], [b,a,c,d]}

```

```

proof –

```

```

  have alts-eq': alts = set [a,b,c,d]

```

```

    by (simp add: alts-eq)

```

```

  have [simp]: a ≠ b a ≠ c a ≠ d b ≠ a b ≠ c b ≠ d

```

```

    c ≠ a c ≠ b c ≠ d d ≠ a d ≠ b d ≠ c

```

```

    using distinct by (simp-all add: eq-commute)

```

```

  interpret anonymous-unanimous-kemeny-sp-swf-explicit agents alts swf 4 [a,b,c,d]

```

```

    by standard (simp-all add: alts-eq card-agents)

```

```

  have R-wf: is-apref-profile' R

```

```

    unfolding is-apref-profile'-def by (auto simp: card-agents permutations-of-set-def alts-eq
R-def)

```

define $R2$ **where** $R2 = \text{repeat-mset } 2 \{ \# [b,a,c,d], [b,a,d,c] \# \}$
define $R3$ **where** $R3 = \text{repeat-mset } 2 \{ \# [a,b,c,d], [b,a,c,d] \# \}$
define $R4$ **where** $R4 = \text{repeat-mset } 2 \{ \# [a,b,d,c], [b,a,c,d] \# \}$
note $R\text{-defs} = R\text{-def } R2\text{-def } R3\text{-def } R4\text{-def}$

have $wf: \text{is-apref-profile}' R \text{ is-apref-profile}' R2 \text{ is-apref-profile}' R3 \text{ is-apref-profile}' R4$
by ($\text{simp-all add: is-apref-profile}'\text{-iff } R\text{-defs add-mset-commute}$)

have $R2: \text{aswf}' R2 = [b,a,c,d]$

proof –

have $\text{aswf}' R2 = [b,a,c,d] \vee \text{aswf}' R2 = [b,a,d,c]$
using $\text{aswf}'\text{-in-allowed-results}[OF wf(2)]$ **unfolding** $R\text{-defs}$
by ($\text{simp add: eval-allowed-results del: Set.filter-eq}$)
moreover have $\text{aswf}' R2 \neq [b,a,d,c] \vee \text{aswf}' R \neq [b,a,c,d]$
by ($\text{intro kemeny-strategyproof-aswf}'\text{-clones}[\text{where } A = [b,a,c,d] \text{ and } n = 2] wf$)
($\text{simp-all add: } R\text{-defs insert-commute swap-dist-code}' \text{ inversion-number-Cons eval-nat-numeral}$)
ultimately show $?thesis$ **using** R
by blast

qed

have $R3: \text{aswf}' R3 = [b,a,c,d]$

proof –

have $\text{aswf}' R3 = [a,b,c,d] \vee \text{aswf}' R3 = [b,a,c,d]$
using $\text{aswf}'\text{-in-allowed-results}[OF wf(3)]$ **unfolding** $R\text{-defs}$
by ($\text{simp add: eval-allowed-results del: Set.filter-eq}$)
moreover have $\text{aswf}' R3 \neq [a,b,c,d] \vee \text{aswf}' R \neq [b,a,c,d]$
by ($\text{intro kemeny-strategyproof-aswf}'\text{-clones}[\text{where } A = [b,a,c,d] \text{ and } n = 2] wf$)
($\text{simp-all add: } R\text{-defs insert-commute swap-dist-code}' \text{ inversion-number-Cons eval-nat-numeral}$)
ultimately show $?thesis$ **using** R
by blast

qed

show $?thesis$

proof –

have $\text{aswf}' R4 \in \{ [a,b,c,d], [b,a,d,c], [a,b,d,c], [b,a,c,d] \}$
using $\text{aswf}'\text{-in-allowed-results}[OF wf(4)]$ **unfolding** $R\text{-defs}$
by ($\text{simp add: eval-allowed-results del: Set.filter-eq}$)
moreover have $\text{aswf}' R3 \neq [b,a,c,d] \vee \text{aswf}' R4 \neq [a,b,c,d]$
by ($\text{intro kemeny-strategyproof-aswf}'\text{-clones}[\text{where } A = [a,b,c,d] \text{ and } n = 2] wf$)
($\text{simp-all add: } R\text{-defs insert-commute swap-dist-code}' \text{ inversion-number-Cons eval-nat-numeral}$)
moreover have $\text{aswf}' R2 \neq [b,a,c,d] \vee \text{aswf}' R4 \neq [b,a,d,c]$
by ($\text{intro kemeny-strategyproof-aswf}'\text{-clones}[\text{where } A = [b,a,d,c] \text{ and } n = 2] wf$)
($\text{simp-all add: } R\text{-defs insert-commute swap-dist-code}' \text{ inversion-number-Cons eval-nat-numeral}$)
ultimately show $?thesis$ **using** $R2 R3$ **unfolding** $R4\text{-def}$
by blast

qed

qed

```

lemma symmetry-breaking-aux2:
  obtains abcd where
    distinct abcd alts = set abcd length abcd = 4
    case abcd of [a,b,c,d] ⇒ aswf' (repeat-mset 2 {# [a,b,c,d], [b,a,d,c] #}) = [a,b,c,d]
proof –
  obtain abcd where abcd: distinct abcd set abcd = alts
    using finite-distinct-list by blast
  have length abcd = 4
    using card-alts abcd distinct-card by metis
  obtain a b c d where abcd-expand: abcd = [a,b,c,d]
    using  $\langle \text{length } abcd = 4 \rangle$  by (force simp: eval-nat-numeral length-Suc-conv)
  have [simp]:  $a \neq b \ a \neq c \ a \neq d \ b \neq a \ b \neq c \ b \neq d$ 
     $c \neq a \ c \neq b \ c \neq d \ d \neq a \ d \neq b \ d \neq c$ 
    using abcd(1) unfolding abcd-expand by (simp-all add: eq-commute)
  have alts-eq: alts = {a,b,c,d}
    unfolding abcd(2) [symmetric] abcd-expand by simp
  interpret anonymous-unanimous-kemeny-sp-swf-explicit agents alts swf 4 [a,b,c,d]
    by standard (simp-all add: alts-eq card-agents)

  define R where  $R = \text{repeat-mset } 2 \ \{ \# \ [a,b,c,d], [b,a,d,c] \ \# \}$ 
  have R-wf: is-apref-profile' R
    unfolding R-def is-apref-profile'-def
    by (auto simp: card-agents abcd-expand simp flip: abcd(2) intro!: linorder-of-ranking)

  have  $aswf' R \in \{[a,b,c,d], [b,a,d,c]\} \vee aswf' R \in \{[a,b,d,c], [b,a,c,d]\}$ 
    using aswf'-in-allowed-results[OF R-wf] unfolding R-def
    by (simp add: eval-allowed-results del: Set.filter-eq)

  thus ?thesis
proof
  assume *:  $aswf' R \in \{[a,b,c,d], [b,a,d,c]\}$ 
  show ?thesis
    by (rule that[of aswf' R])
      (use * in <unfold R-def, auto simp add: alts-eq insert-commute add-mset-commute eval-nat-numeral>)
  next
  assume  $aswf' R \in \{[a,b,d,c], [b,a,c,d]\}$ 
  hence  $aswf' R = [a,b,d,c] \vee aswf' R = [b,a,c,d]$ 
    by blast
  thus ?thesis
proof
  assume *:  $aswf' R = [b,a,c,d]$ 
  have **:  $aswf' (\text{repeat-mset } 2 \ \{ \# \ [a,b,d,c], [b,a,c,d] \ \# \}) \in \{[a,b,d,c], [b,a,c,d]\}$ 
    by (rule symmetry-breaking-aux1 [of a b c d])
      (use * in <simp-all add: R-def add-mset-commute alts-eq>)
  show ?thesis
    by (rule that[of aswf' (repeat-mset 2 {#[a,b,d,c], [b,a,c,d]#})])
      (use ** in <auto simp: alts-eq add-mset-commute insert-commute add.commute>)
  next

```

```

assume *: aswf'  $R = [a,b,d,c]$ 
have **: aswf' (repeat-mset 2 {#[b,a,c,d], [a,b,d,c]#}) ∈ {[b,a,c,d], [a,b,d,c]}
  by (rule symmetry-breaking-aux1 [of b a d c])
    (use * in ⟨simp-all add: R-def add-mset-commute alts-eq insert-commute⟩)
show ?thesis
  by (rule that[of aswf' (repeat-mset 2 {#[a,b,d,c], [b,a,c,d]#})])
    (use ** in ⟨auto simp: alts-eq add-mset-commute insert-commute add.commute⟩)
qed
qed
qed

lemma contradiction: False
proof –
  obtain abcd where abcd:
    distinct abcd alts = set abcd length abcd = 4
    case abcd of [a,b,c,d] ⇒ aswf' (repeat-mset 2 {# [a,b,c,d], [b,a,d,c] #}) = [a,b,c,d]
    using symmetry-breaking-aux2 .
  from ⟨length abcd = 4⟩ obtain a b c d where abcd-expand: abcd = [a,b,c,d]
    by (auto simp: length-Suc-conv eval-nat-numeral)
  interpret anonymous-unanimous-kemenysp-swf-explicit-4-4 agents alts swf a b c d
  proof
    show aswf' {#[a, b, c, d], [a, b, c, d], [b, a, d, c], [b, a, d, c]#} = [a, b, c, d]
      using abcd unfolding abcd-expand by (simp add: eval-nat-numeral add-mset-commute)
    qed (use abcd(1) in ⟨simp-all add: card-agents abcd abcd-expand eq-commute⟩)
    show ?thesis
      by (fact contradiction)
  qed
end

```

The final result: extending the impossibility to $m \geq 2$ and n a multiple of 4.

```

theorem (in anonymous-unanimous-kemenysp-swf) impossibility':
  assumes 4 dvd card agents and card alts ≥ 4
  shows False
proof –
  have card agents ≥ 4
    using assms(1) nonempty-agents finite-agents by (meson card-0-eq dvd-imp-le not-gr0)
  obtain agents' where agents': agents' ⊆ agents card agents' = 4
    using ⟨card agents ≥ 4⟩ by (meson obtain-subset-with-card-n)
  have [simp]: agents' ≠ {}
    using agents' by auto
  obtain alts' where alts': alts' ⊆ alts card alts' = 4
    using ⟨card alts ≥ 4⟩ by (meson obtain-subset-with-card-n)
  obtain dummy-alts where alts-list': mset dummy-alts = mset-set (alts - alts')
    using ex-mset by blast
  obtain unclone where cloning agents' agents unclone
    using cloning-exists[of agents' agents] agents' ⟨4 dvd card agents⟩ finite-agents by auto
  interpret cloning agents' agents unclone by fact

```

```

interpret split: swf-split-agents agents alts swf agents' unclone ..
interpret new1: anonymous-swf agents' alts split.swf-low
  by (rule split.anonymous-clone) unfold-locales
interpret new1: unanimous-swf agents' alts split.swf-low
  by (rule split.unanimous-clone) unfold-locales
interpret new1: kemeny-strategyproof-swf agents' alts split.swf-low
  by (rule split.kemeny-strategyproof-clone) unfold-locales

interpret restrict: unanimous-swf-restrict-alts agents' alts split.swf-low dummy-alts alts'
proof
  show finite alts'
    using alts'(1) finite-subset by blast
  show mset-set alts = mset dummy-alts + mset-set alts'
    by (simp add: <finite alts'> alts'(1) alts-list' mset-set-Diff)
  show alts' ≠ {}
    using alts'(2) by fastforce
qed
interpret new2: anonymous-swf agents' alts' restrict.swf-low
  by (rule restrict.anonymous-restrict) unfold-locales
interpret new2: unanimous-swf agents' alts' restrict.swf-low ..
interpret new2: kemeny-strategyproof-swf agents' alts' restrict.swf-low
  by (rule restrict.kemeny-strategyproof-restrict) unfold-locales

interpret new2: anonymous-unanimous-kemenysp-swf-4-4 agents' alts' restrict.swf-low
  by standard fact+
show False
  by (fact new2.contradiction)
qed

```

The following collects thw two impossibility results in one theorem.

```

theorem anonymous-unanimous-kemenysp-impossibility:
  assumes (card alts = 4 ∧ 4 dvd card agents) ∨ (card alts ≥ 5 ∧ even (card agents))
  assumes anonymous-swf agents alts swf
  assumes unanimous-swf agents alts swf
  assumes kemeny-strategyproof-swf agents alts swf
  shows False
proof –
  interpret anonymous-swf agents alts swf by fact
  interpret unanimous-swf agents alts swf by fact
  interpret kemeny-strategyproof-swf agents alts swf by fact
  interpret anonymous-unanimous-kemenysp-swf agents alts swf ..
  show False using assms(1) impossibility impossibility' by linarith
qed

end
theory Majcons-Stratproof-Impossibility
  imports SWF-Impossibility-Automation
begin

```

A somewhat technical lemma: If the swap distance of two rankings restricted to some

subset A is the same as the swap distance of the full rankings and additionally the elements of A are all ranked above the elements not in A in one of the rankings, then the second ranking must also have all elements not in A ranked below those in A and in the same order.

lemma *swap-dist-append-eq-swap-dist-filter-imp-eq*:

fixes $xs\ ys\ zs$

defines $zs' \equiv (filter\ (\lambda x. x \in set\ xs)\ zs)$

assumes $swap-dist\ (xs\ @\ ys)\ zs \leq swap-dist\ xs\ zs'$

assumes $wf: distinct\ (xs\ @\ ys)\ distinct\ zs\ set\ (xs\ @\ ys) = set\ zs$

shows $zs = zs' @ ys$

proof –

have $linorder-on\ (set\ zs)\ (of-ranking\ (xs\ @\ ys))$

by $(rule\ linorder-of-ranking)\ (use\ assms\ in\ auto)$

moreover have $linorder-on\ (set\ zs)\ (of-ranking\ zs)$

by $(rule\ linorder-of-ranking)\ (use\ assms\ in\ auto)$

ultimately have $*$: $of-ranking\ (xs\ @\ ys)\ a\ b = of-ranking\ zs\ a\ b$

if $a \notin set\ xs \vee b \notin set\ xs$ **for** $a\ b$

proof $(rule\ swap-dist-relation-restrict-eq-imp-eq)$

note $\langle swap-dist\ (xs\ @\ ys)\ zs \leq swap-dist\ xs\ zs' \rangle$

also have $swap-dist\ (xs\ @\ ys)\ zs = swap-dist-relation\ (of-ranking\ (xs\ @\ ys))\ (of-ranking\ zs)$

unfolding $swap-dist-def$ **using** $assms$ **by** $auto$

also have $swap-dist\ xs\ zs' = swap-dist-relation\ (of-ranking\ xs)\ (of-ranking\ zs')$

unfolding $swap-dist-def$ **using** $assms$ **by** $auto$

also have $filter\ (\lambda x. x \in set\ xs)\ ys = []$

unfolding $filter-empty-conv$ **using** $assms$ **by** $auto$

hence $of-ranking\ xs = of-ranking\ (filter\ (\lambda x. x \in set\ xs)\ (xs\ @\ ys))$

by $simp$

finally show $swap-dist-relation\ (restrict-relation\ (set\ xs)\ (of-ranking\ (xs\ @\ ys)))$

$(restrict-relation\ (set\ xs)\ (of-ranking\ zs)) \geq$

$swap-dist-relation\ (of-ranking\ (xs\ @\ ys))\ (of-ranking\ zs)$

unfolding $zs'-def\ of-ranking-filter$ **by** $simp$

qed $(use\ that\ in\ auto)$

have $of-ranking\ zs\ a\ b = of-ranking\ (zs' @ ys)\ a\ b$ **for** $a\ b$

proof $(cases\ a \in set\ xs \wedge b \in set\ xs)$

case $True$

hence $of-ranking\ zs\ a\ b \longleftrightarrow of-ranking\ zs'\ a\ b$

by $(auto\ simp: zs'-def\ of-ranking-filter\ restrict-relation-def)$

also have $\dots \longleftrightarrow of-ranking\ (zs' @ ys)\ a\ b$

using $wf\ of-ranking-imp-in-set[of\ ys\ a\ b]$ $True$

by $(auto\ simp: of-ranking-append\ zs'-def)$

finally show $?thesis$.

next

case $False$

hence $of-ranking\ zs\ a\ b \longleftrightarrow of-ranking\ (xs\ @\ ys)\ a\ b$

by $(intro\ * [symmetric])\ auto$

also have $\dots \longleftrightarrow of-ranking\ (zs' @ ys)\ a\ b$

using $wf\ False\ of-ranking-imp-in-set[of\ xs\ a\ b]\ of-ranking-imp-in-set[of\ zs'\ a\ b]$

by $(auto\ simp: of-ranking-append\ zs'-def)$

finally show *?thesis* .
qed
hence *of-ranking* $zs = \text{of-ranking } (zs' @ ys)$
by *blast*
hence *ranking* (*of-ranking* zs) = *ranking* (*of-ranking* ($zs' @ ys$))
by (*rule arg-cong*)
also have *ranking* (*of-ranking* zs) = zs
by (*rule ranking-of-ranking*) (*use wf in auto*)
also have *ranking* (*of-ranking* ($zs' @ ys$)) = $zs' @ ys$
by (*rule ranking-of-ranking*) (*use wf in <auto simp: zs'-def>*)
finally show *?thesis* .
qed

We now turn to a setting where we have exactly 9 agents and 4 alternatives and an SWF that is majority consistent and satisfies our weak form of Kemeny strategyproofness where the only manipulated profiles that have a linear majority relation are considered. We will, in particular, consider two specific profiles and show that there is only one admissible result ranking for them.

When strengthening the strategyproofness assumption to full strategyproofness, these two results also turn out to be incompatible, yielding a contradiction.

locale *majcons-weak-kstratproof-swf-explicit-4-9* =
majcons-weak-kstratproof-swf-explicit *agents* *alts* *swf* *agents-list* [a,b,c,d]
for *agents* :: '*agent set* **and** *alts* :: '*alt set* **and** *swf*
and *agents-list* **and** $a\ b\ c\ d$ +
assumes *card-agents-9* [*simp*]: *card* *agents* = 9
begin

lemma *distinct-abcd* [*simp*]:
 $a \neq b\ a \neq c\ a \neq d\ b \neq a\ b \neq c\ b \neq d$
 $c \neq a\ c \neq b\ c \neq d\ d \neq a\ d \neq b\ d \neq c$
using *distinct-alts-list* **by** *auto*

We consider the following profile R . This profile does not have a linear majority relation, but many manipulations of it do.

definition R :: '*alt list list* **where**
 $R = [[c,d,b,a],[b,a,d,c],[d,b,a,c],[c,b,a,d],$
 $[a,d,c,b],[c,a,d,b],[d,c,b,a],[d,a,b,c],[a,b,c,d]]$

lemma R -*wf* [*simp*]: *prefs-from-rankings-wf* R
by (*simp add: prefs-from-rankings-wf-def R-def*)

We perform five independent manipulations of R , all of which result in profiles with a transitive majority relation. This gives us five upper bounds about the swap distance between $f(R)$ and one other ranking each. It turns out that there is only one ranking that satisfies all of these constraints, and that ranking is $adcb$.

Note also that the first four inequalities are all sharp.

lemma swf' - R : $swf' R = [a,d,c,b]$

proof –

note $SP = \text{majority-consistent-kemeny-strategyproof-swf'-aux}$

have $\text{swap-dist } [c,d,b,a] (\text{swf}' R) \leq \text{swap-dist } [c,d,b,a] [a,d,c,b]$

by (*rule* $SP[\text{where } i = 0 \text{ and } zs = [c,d,a,b]]$)

(*simp-all add: R-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 1: $\text{swap-dist } [c,d,b,a] (\text{swf}' R) \leq 4$

by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $\text{swap-dist } [b,a,d,c] (\text{swf}' R) \leq \text{swap-dist } [b,a,d,c] [a,d,c,b]$

by (*rule* $SP[\text{where } i = 1 \text{ and } zs = [a,b,d,c]]$)

(*simp-all add: R-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 2: $\text{swap-dist } [b,a,d,c] (\text{swf}' R) \leq 3$

by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $\text{swap-dist } [d,b,a,c] (\text{swf}' R) \leq \text{swap-dist } [d,b,a,c] [a,d,c,b]$

by (*rule* $SP[\text{where } i = 2 \text{ and } zs = [d,a,b,c]]$)

(*simp-all add: R-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 3: $\text{swap-dist } [d,b,a,c] (\text{swf}' R) \leq 3$

by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $\text{swap-dist } [c,b,a,d] (\text{swf}' R) \leq \text{swap-dist } [c,b,a,d] [a,d,c,b]$

by (*rule* $SP[\text{where } i = 3 \text{ and } zs = [c,a,b,d]]$)

(*simp-all add: R-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 4: $\text{swap-dist } [c,b,a,d] (\text{swf}' R) \leq 4$

by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $\text{swap-dist } [a,d,c,b] (\text{swf}' R) \leq \text{swap-dist } [a,d,c,b] [d,b,a,c]$

by (*rule* $SP[\text{where } i = 4 \text{ and } zs = [d,a,b,c]]$)

(*simp-all add: R-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 5: $\text{swap-dist } [a,d,c,b] (\text{swf}' R) \leq 3$

by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

define $\text{constraints} :: ('alt \text{ list} \times \text{nat}) \text{ list where}$

$\text{constraints} = [([c,d,b,a],4), ([b,a,d,c],3), ([d,b,a,c],3), ([c,b,a,d],4), ([a,d,c,b],3)]$

define $P \text{ where } P = (\lambda ys. \text{list-all } (\lambda(xs,d). \text{swap-dist } xs \ ys \leq d) \text{ constraints})$

have $\text{swf}' R \in \text{permutations-of-set alts}$

using $\text{swf}'\text{-wf}[of R] \text{ mset-eq-alts-list-iff}[of \text{swf}' R] \text{ alts-conv-alts-list}$

by (*simp add: permutations-of-set-def*)

moreover have $P (\text{swf}' R)$

unfolding $P\text{-def using } 1 \ 2 \ 3 \ 4 \ 5 \text{ by } (simp \text{ add: constraints-def})$

ultimately have $\text{swf}' R \in \text{Set.filter } P (\text{permutations-of-set alts})$

by *simp*

also have $\text{permutations-of-set alts} = \text{set } (\text{permutations-of-set-list } [a,b,c,d])$

unfolding $\text{alts-conv-alts-list by } (subst \text{ permutations-of-list}) \text{ simp-all}$

also have $\text{Set.filter } P \dots = \{[a,d,c,b]\}$

by (*simp add: P-def constraints-def permutations-of-set-list-def insert-commute permutations-of-set-aux-list-Nil permutations-of-set-aux-list-Cons*)

Set-filter-insert swap-dist-conv-inversion-number inversion-number-Cons
del: Set.filter-eq)

finally show *?thesis*
 by *simp*
qed

We now consider a second profile, which differs from R only by a manipulation of the third agent.

definition $S :: 'alt\ list\ list\ \mathbf{where}$
 $S = [[c,d,b,a],[b,a,d,c],[d,b,c,a],[c,b,a,d],[a,d,c,b],$
 $[c,a,d,b],[d,c,b,a],[d,a,b,c],[a,b,c,d]]$

lemma $S\text{-wf}$ [*simp*]: *prefs-from-rankings-wf S*
 by (*simp add: prefs-from-rankings-wf-def S-def*)

We similarly show that $f(S) = dcba$.

lemma $swf'\text{-}S$: $swf' S = [d,c,b,a]$

proof –

note $SP = majority-consistent-kemeny-strategyproof-swf'\text{-}aux$

have $swap-dist [c,b,a,d] (swf' S) \leq swap-dist [c,b,a,d] [d,c,b,a]$
 by (*rule SP[where i = 3 and zs = [c,b,d,a]]*)
 (*simp-all add: S-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 1: $swap-dist [c,b,a,d] (swf' S) \leq 3$
 by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $swap-dist [a,d,c,b] (swf' S) \leq swap-dist [a,d,c,b] [d,c,b,a]$
 by (*rule SP[where i = 4 and zs = [d,a,c,b]]*)
 (*simp-all add: S-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 2: $swap-dist [a,d,c,b] (swf' S) \leq 3$
 by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $swap-dist [c,a,d,b] (swf' S) \leq swap-dist [c,a,d,b] [d,c,b,a]$
 by (*rule SP[where i = 5 and zs = [c,d,a,b]]*)
 (*simp-all add: S-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 3: $swap-dist [c,a,d,b] (swf' S) \leq 3$
 by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $swap-dist [b,a,d,c] (swf' S) \leq swap-dist [b,a,d,c] [d,c,b,a]$
 by (*rule SP[where i = 1 and zs = [b,d,a,c]]*)
 (*simp-all add: S-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 4: $swap-dist [b,a,d,c] (swf' S) \leq 4$
 by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

have $swap-dist [d,c,b,a] (swf' S) \leq swap-dist [d,c,b,a] [c,a,d,b]$
 by (*rule SP[where i = 6 and zs = [c,d,a,b]]*)
 (*simp-all add: S-def prefs-from-rankings-wf-def of-ranking-Cons*)

hence 5: $swap-dist [d,c,b,a] (swf' S) \leq 3$
 by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)

```

define constraints :: ('alt list × nat) list where
  constraints = [([c,b,a,d],3), ([a,d,c,b],3), ([c,a,d,b],3), ([b,a,d,c],4), ([d,c,b,a],3)]
define P where P = (λys. list-all (λ(xs,d). swap-dist xs ys ≤ d) constraints)

have swf' S ∈ permutations-of-set alts
  using swf'-wf[of S] mset-eq-alts-list-iff[of swf' S] alts-conv-alts-list
  by (simp add: permutations-of-set-def)
moreover have P (swf' S)
  unfolding P-def using 1 2 3 4 5 by (simp add: constraints-def)
ultimately have swf' S ∈ Set.filter P (permutations-of-set alts)
  by simp
also have permutations-of-set alts = set (permutations-of-set-list [a,b,c,d])
  unfolding alts-conv-alts-list by (subst permutations-of-list) simp-all
also have Set.filter P ... = {[d,c,b,a]}
  by (simp add: P-def constraints-def permutations-of-set-list-def insert-commute
      permutations-of-set-aux-list-Nil permutations-of-set-aux-list-Cons
      Set-filter-insert swap-dist-conv-inversion-number inversion-number-Cons
      del: Set.filter-eq)
finally show ?thesis
  by simp
qed

end

```

We use the argument outlined in the paper to derive the impossibility for 9 agents and ≥ 4 alternatives. We call the first four alternatives a, b, c, d and treat the remaining ones as “dummy alternatives” in some fixed order. Agents will always list them as their least preferred alternatives in exactly that fixed order.

The complication is that, since we do not have unanimity, the ranking returned by the SWF does not have to respect this order or put them as less preferred than the ‘real’ alternatives. However, we can show that for the profiles we consider, the SWF indeed has to respect the order.

```

context majcons-kstratproof-swf-explicit
begin

```

```

sublocale majcons-weak-kstratproof-swf-explicit agents alts swf agents-list alts-list ..

```

```

lemma contradiction-eq9-ge4-aux:

```

```

  assumes card agents = 9 card alts ≥ 4

```

```

  shows False

```

```

proof –

```

```

  define a b c d where

```

```

    a = alts-list ! 0 and b = alts-list ! 1 and c = alts-list ! 2 and d = alts-list ! 3

```

```

  define dummy-alts where dummy-alts = drop 4 alts-list

```

```

have alts-list-expand: alts-list = a # b # c # d # dummy-alts

```

```

  by (rule nth-equalityI)

```

```

  (use ‹card alts ≥ 4›
    in ‹auto simp: a-def b-def c-def d-def dummy-alts-def nth-Cons eval-nat-numeral
        split: nat.splits›)
have mset-alts-list [simp]: mset alts-list = {#a,b,c,d#} + mset dummy-alts
  by (simp add: alts-list-expand)
have distinct-abcd: distinct [a,b,c,d]
and abcd-not-in-dummy-alts: {a,b,c,d} ∩ set dummy-alts = {}
and distinct-dummy-alts
using distinct-alts-list unfolding alts-list-expand by auto

interpret majority-consistent-weak-kstratproof-swf-restrict-alts
  agents alts swf dummy-alts {a,b,c,d}
by unfold-locales
  (use distinct-abcd in ‹simp-all add: alts-conv-alts-list mset-set-set distinct-alts-list›)
interpret swf-restrict-alts-explicit agents alts swf dummy-alts {a,b,c,d}
  agents-list alts-list [a,b,c,d]
by standard (simp-all add: alts-list-expand)
interpret new: majcons-weak-kstratproof-swf-explicit-4-9
  agents {a,b,c,d} swf-low agents-list a b c d
by standard (use distinct-abcd ‹card agents = 9› in ‹simp-all add: agents-list›)

define R S where R = map extend new.R and S = map extend new.S
have R-wf: prefs-from-rankings-wf R and S-wf: prefs-from-rankings-wf S
by (simp-all add: prefs-from-rankings-wf-def R-def new.R-def S-def new.S-def extend-def)

```

The swap distance inequalities we derived through strategyproofness before still hold after adding the dummy alternatives. We only need one of them for each of R and S .

```

have swap-dist-swf'-R: swap-dist (extend [c,d,b,a]) (swf' R) ≤ 4
proof –
  have swap-dist (extend [c,d,b,a]) (swf' R) ≤ swap-dist (extend [c,d,b,a]) (extend [a,d,c,b])
  proof (rule majority-consistent-kemeny-strategyproof-swf'
    [OF R-wf, where i = 0 and zs = extend [c,d,a,b]])
  have majority-rel-mset (mset (new.R[0 := [c,d,a,b]]) [a,d,c,b])
  by (rule new.majority-rel-list-aux-imp-majority-rel-mset)
    (use distinct-abcd in ‹simp-all add: new.R-def new.prefs-from-rankings-wf-def
      add-mset-commute of-ranking-Cons›)
  hence majority-rel-mset (mset (map extend (new.R[0 := [c,d,a,b]])) (extend [a,d,c,b]))
  by (subst majority-rel-mset-extend) (simp-all add: new.R-def new.prefs-from-rankings-wf-def)
  also have map extend (new.R[0 := [c,d,a,b]]) = R[0 := extend [c,d,a,b]]
  by (simp add: R-def new.R-def)
  finally show majority-rel-mset (mset (R[0 := extend [c,d,a,b]]) (extend [a,d,c,b])) .
qed (simp-all add: R-def new.R-def extend-def)
also have ... = swap-dist [c,d,b,a] [a,d,c,b]
  by (subst swap-dist-extend) auto
also have ... = 4
  using distinct-abcd
  by (simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons)
finally show ?thesis .
qed

```

```

have swap-dist-swf'-S: swap-dist (extend [c,b,a,d]) (swf' S) ≤ 3
proof –
  have swap-dist (extend [c,b,a,d]) (swf' S) ≤ swap-dist (extend [c,b,a,d]) (extend [d,c,b,a])
  proof (rule majority-consistent-kemeny-strategyproof-swf'
    [OF S-wf, where i = 3 and zs = extend [c,b,d,a]])
  have majority-rel-mset (mset (new.S[3 := [c,b,d,a]]) [d,c,b,a])
  by (rule new.majority-rel-list-aux-imp-majority-rel-mset
    (use distinct-abcd in ⟨simp-all add: new.S-def new.prefs-from-rankings-wf-def
      add-mset-commute of-ranking-Cons⟩)
  hence majority-rel-mset (mset (map extend (new.S[3 := [c,b,d,a]])) (extend [d,c,b,a])
  by (subst majority-rel-mset-extend) (simp-all add: new.S-def new.prefs-from-rankings-wf-def)
  also have map extend (new.S[3 := [c,b,d,a]]) = S[3 := extend [c, b, d, a]]
  by (simp add: S-def new.S-def)
  finally show majority-rel-mset (mset (S[3 := extend [c, b, d, a]]) (extend [d, c, b, a]) .
qed (simp-all add: S-def new.S-def extend-def)
also have ... = swap-dist [c,b,a,d] [d,c,b,a]
  by (subst swap-dist-extend) auto
also have ... = 3
  using distinct-abcd
  by (simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons)
finally show ?thesis .
qed

```

Since we know that R returns the ranking $adcb$ when restricted to four alternatives and this already has a swap distance of 3 to $cbad$, that means that all the dummy alternatives in the ranking returned for R must be in the desired position since they would only introduce additional swap distance, i.e. $f(R) = \uparrow adcb$.

```

have swf'-R: swf' R = extend [a, d, c, b]
proof –
  have new.swf' new.R = [a, d, c, b]
  by (rule new.swf'-R)
  also have new.swf' new.R = filter (λx. x ∈ {a,b,c,d}) (swf' R)
  using new-swf'-eq[OF new.R-wf] by (simp add: R-def)
  finally have restrict-swf'-R: filter (λx. x ∈ {a, b, c, d}) (swf' R) = [a, d, c, b] .

have swf' R = filter (λx. x ∈ set [c,d,b,a]) (swf' R) @ dummy-alts
proof (rule swap-dist-append-eq-swap-dist-filter-imp-eq)
  have mset ([c,d,b,a] @ dummy-alts) = mset alts-list
  by simp
  with distinct-alts-list show distinct ([c,d,b,a] @ dummy-alts)
  using mset-eq-imp-distinct-iff by blast
next
  have *: mset (swf' R) = mset-set alts
  by (rule swf'-wf) (fact R-wf)
  from * show distinct (swf' R)
  by (metis alts-list mset-eq-alts-list-iff)
  have mset ([c,d,b,a] @ dummy-alts) = mset alts-list
  by simp

```

```

with * show  $set ([c,d,b,a] @ dummy\text{-}alts) = set (swf' R)$ 
  by (metis alts-list mset-eq-setD)
next
have  $filter (\lambda x. x \in set [c,d,b,a]) (swf' R) = [a,d,c,b]$ 
  using restrict-swf'-R by (simp add: disj-ac)
hence  $swap\text{-}dist [c,d,b,a] (filter (\lambda x. x \in set [c,d,b,a]) (swf' R)) =$ 
   $swap\text{-}dist [c,d,b,a] [a,d,c,b]$ 
  by (rule arg-cong)
also have  $\dots = 4$ 
  using distinct-abcd
  by (simp add: swap-dist-conv-inversion-number insert-commute eq-commute inversion-number-Cons)
also have  $4 \geq swap\text{-}dist ([c,d,b,a] @ dummy\text{-}alts) (swf' R)$ 
  using swap-dist-swf'-R by (simp add: extend-def)
finally show  $swap\text{-}dist ([c,d,b,a] @ dummy\text{-}alts) (swf' R) \leq$ 
   $swap\text{-}dist [c,d,b,a] (filter (\lambda x. x \in set [c,d,b,a]) (swf' R)) .$ 
qed
also have  $filter (\lambda x. x \in set [c,d,b,a]) (swf' R) = [a,d,c,b]$ 
  using restrict-swf'-R by (simp add: disj-ac)
finally show  $swf' R = extend [a, d, c, b]$  by (simp add: extend-def)
qed

```

And similarly for S .

```

have  $swf'\text{-}S: swf' S = extend [d, c, b, a]$ 
proof –
  have  $new.swf' new.S = [d, c, b, a]$ 
    by (rule new.swf'-S)
  also have  $new.swf' new.S = filter (\lambda x. x \in \{a,b,c,d\}) (swf' S)$ 
    using new-swf'-eq[OF new.S-wf] by (simp add: S-def)
  finally have  $restrict\text{-}swf'\text{-}S: filter (\lambda x. x \in \{a, b, c, d\}) (swf' S) = [d, c, b, a] .$ 

have  $swf' S = filter (\lambda x. x \in set [c,b,a,d]) (swf' S) @ dummy\text{-}alts$ 
proof (rule swap-dist-append-eq-swap-dist-filter-imp-eq)
  have  $mset ([c,b,a,d] @ dummy\text{-}alts) = mset alts\text{-}list$ 
    by simp
  with distinct-alts-list show  $distinct ([c,b,a,d] @ dummy\text{-}alts)$ 
    using mset-eq-imp-distinct-iff by blast
next
have *:  $mset (swf' S) = mset\text{-}set alts$ 
  by (rule swf'-wf) (fact S-wf)
from * show  $distinct (swf' S)$ 
  by (metis alts-list mset-eq-alts-list-iff)
have  $mset ([c,b,a,d] @ dummy\text{-}alts) = mset alts\text{-}list$ 
  by simp
with * show  $set ([c,b,a,d] @ dummy\text{-}alts) = set (swf' S)$ 
  by (metis alts-list mset-eq-setD)
next
have  $filter (\lambda x. x \in set [c,b,a,d]) (swf' S) = [d,c,b,a]$ 
  using restrict-swf'-S by (simp add: disj-ac)

```

hence $\text{swap-dist } [c,b,a,d] (\text{filter } (\lambda x. x \in \text{set } [c,b,a,d]) (\text{swf}' S)) =$
 $\text{swap-dist } [c,b,a,d] [d,c,b,a]$
by (*rule arg-cong*)
also have $\dots = 3$
using *distinct-abcd*
by (*simp add: swap-dist-conv-inversion-number insert-commute eq-commute inversion-number-Cons*)
also have $3 \geq \text{swap-dist } ([c,b,a,d] @ \text{dummy-alts}) (\text{swf}' S)$
using *swap-dist-swf'-S* **by** (*simp add: extend-def*)
finally show $\text{swap-dist } ([c,b,a,d] @ \text{dummy-alts}) (\text{swf}' S) \leq$
 $\text{swap-dist } [c,b,a,d] (\text{filter } (\lambda x. x \in \text{set } [c,b,a,d]) (\text{swf}' S)) .$
qed
also have $\text{filter } (\lambda x. x \in \text{set } [c,b,a,d]) (\text{swf}' S) = [d,c,b,a]$
using *restrict-swf'-S* **by** (*simp add: disj-ac*)
finally show $\text{swf}' S = \text{extend } [d, c, b, a]$
by (*simp add: extend-def*)
qed

Finally, strategyproofness tells us that $\Delta(f(R), \uparrow dbac) \leq \Delta(f(S), \uparrow dbac)$. However, we also know that $f(R) = \uparrow adcb$ and $f(S) = \uparrow dcba$, leading to $3 = \Delta(f(R), \uparrow dbac) \leq 2 = \Delta(f(S), \uparrow dbac)$.

have $\text{swap-dist } (\text{extend } [d,b,a,c]) (\text{swf}' R) \leq \text{swap-dist } (\text{extend } [d,b,a,c]) (\text{swf}' S)$
by (*rule kemeny-strategyproof-swf'* [**where** $i = 2$ **and** $zs = \text{extend } [d,b,c,a]$])
(use R-wf in <simp-all add: R-def S-def new.R-def new.S-def extend-def>)
also have $\text{swf}' R = \text{extend } [a,d,c,b]$
by fact
also have $\text{swap-dist } (\text{extend } [d,b,a,c]) (\text{extend } [a,d,c,b]) = \text{swap-dist } [d,b,a,c] [a,d,c,b]$
by (*rule swap-dist-extend*) *simp-all*
also have $\dots = 3$
using *distinct-abcd*
by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)
also have $\text{swf}' S = \text{extend } [d,c,b,a]$
by fact
also have $\text{swap-dist } (\text{extend } [d,b,a,c]) (\text{extend } [d,c,b,a]) = \text{swap-dist } [d,b,a,c] [d,c,b,a]$
by (*rule swap-dist-extend*) *simp-all*
also have $\dots = 2$
using *distinct-abcd*
by (*simp add: swap-dist-conv-inversion-number insert-commute inversion-number-Cons*)
finally show *False*
by simp
qed

Using agent cloning, we can lift the impossibility to any multiple of 9 agents. In particular, we can derive it for 18 agents.

lemma *contradiction-eg18-ge4-aux*:
assumes $\text{card agents} = 18$ $\text{card alts} \geq 4$
shows *False*
proof –
have $\text{card agents} \geq 9$

using *assms* **by** *simp*
then obtain *agents'* **where** *agents'*: $agents' \subseteq agents$ *card agents' = 9*
using *obtain-subset-with-card-n* **by** *metis*
obtain *unclone* **where** *cloning agents' agents unclone*
using *cloning-exists[of agents' agents]* *agents' assms* **by** *force*
interpret *unclone: cloning agents' agents unclone* **by** *fact*
interpret *swf-split-agents agents alts swf agents' unclone ..*

interpret *new: majority-consistent-swf agents' alts swf-low*
by (*rule majority-consistent-clone*) *unfold-locales*
interpret *new: kemeny-strategyproof-swf agents' alts swf-low*
by (*rule kemeny-strategyproof-clone*) *unfold-locales*
have *finite agents'*
by (*rule finite-subset[OF - finite-agents]*) (*use agents' in auto*)
then obtain *agents-list'*
where *agents-list'*: $mset\ agents-list' = mset-set\ agents'$
using *ex-mset* **by** *blast*
interpret *new: majcons-kstratproof-swf-explicit agents' alts swf-low agents-list' alts-list*
by *unfold-locales (use agents-list' alts-list in auto)*

show *False*
by (*rule new.contradiction-eg9-ge4-aux*) (*use <card agents' = 9> assms in auto*)
qed

By adding k agents together with k 'anti-clones' of these agents, we can lift the impossibility to $9 + 2k$ or $18 + 2k$ agents. This covers every $n \geq 9$ except for $n \in \{10, 12, 14, 16\}$.

lemma *contradiction-geq9-ge4-aux*:

assumes $card\ agents \in \{9, 11, 13, 15\} \cup \{17..\}$ *card alts ≥ 4*
shows *False*

proof –

define k **where** $k = (card\ agents - (if\ even\ (card\ agents)\ then\ 18\ else\ 9))\ div\ 2$

from *assms* **have** $card\ agents \geq (if\ even\ (card\ agents)\ then\ 18\ else\ 9)$

by *auto*

hence k : $card\ agents = (if\ even\ (card\ agents)\ then\ 18\ else\ 9) + 2 * k$

unfolding $k-def$ **by** *auto*

have $k \leq card\ agents$

using k **by** *auto*

then obtain *agents1* **where** *agents1*: $agents1 \subseteq agents$ *card agents1 = k*

using k *obtain-subset-with-card-n* **by** *metis*

have $k \leq card\ (agents - agents1)$

by (*subst card-Diff-subset*) (*use agents1 finite-subset[OF agents1(1)] k in auto*)

then obtain *agents2* **where** *agents2*: $agents2 \subseteq agents - agents1$ *card agents2 = k*

using k *obtain-subset-with-card-n* **by** *metis*

have [*simp*]: *finite agents1 finite agents2*

by (*rule finite-subset[of - agents]; use agents1 agents2 in force*)+

interpret *dummy-ord: linorder-on alts of-ranking alts-list*

by (*rule linorder-of-ranking*) (*use alts-conv-alts-list distinct-alts-list in auto*)

```

interpret swf-reduce-agents-even agents alts swf agents1 agents2 of-ranking alts-list
proof
  have card (agents1 ∪ agents2) < card agents
    by (subst card-Un-disjoint) (use agents1 agents2 k in <auto split: if-splits>)
  moreover have agents1 ∪ agents2 ⊆ agents
    using agents1 agents2 by blast
  ultimately show agents1 ∪ agents2 ⊂ agents
    by blast
qed (use agents1 agents2 in auto)

interpret new: majority-consistent-swf agents - agents1 - agents2 alts swf-low
  by (rule majority-consistent-reduce) unfold-locales
interpret new: kemeny-strategyproof-swf agents - agents1 - agents2 alts swf-low
  by (rule kemeny-strategyproof-reduce) unfold-locales
have finite (agents - agents1 - agents2)
  by (rule finite-subset[OF - finite-agents]) auto
then obtain agents-list'
  where agents-list': mset agents-list' = mset-set (agents - agents1 - agents2)
  using ex-mset by blast
interpret new: majcons-kstratproof-swf-explicit
  agents - agents1 - agents2 alts swf-low agents-list' alts-list
  by unfold-locales (use agents-list' alts-list in auto)

have card (agents - agents1 - agents2) ∈ {9, 18}
  using agents1 agents2 k by (simp add: card-Diff-subset split: if-splits)
thus False
  using new.contradiction-eq9-ge4-aux new.contradiction-eq18-ge4-aux assms by auto
qed

end

```

We get rid of the explicit lists of agents and alternatives.

```

context majcons-kstratproof-swf
begin

lemma contradiction-geq9-ge4:
  assumes card agents ∈ {9, 11, 13, 15} ∪ {17..} card alts ≥ 4
  shows False
proof -
  obtain agents-list where mset agents-list = mset-set agents
    using ex-mset by blast
  obtain alts-list where mset alts-list = mset-set alts
    using ex-mset by blast
  interpret majcons-kstratproof-swf-explicit agents alts swf agents-list alts-list
    by standard fact+
  show ?thesis
    using contradiction-geq9-ge4-aux assms by simp
qed

```

end

We use an imported SAT proof to show the case of $m = 4$, $n = 3$.

external-file *sat-data/maj-profiles-4-3.xz*

external-file *sat-data/maj-4-3.grat.xz*

external-file *sat-data/maj-sp-4-4.xz*

external-file *sat-data/maj-4-4.grat.xz*

locale *majcons-kstratproof-swf-explicit-4-3* =

majcons-kstratproof-swf-explicit *agents* *alts* *swf* [*A1*,*A2*,*A3*] [*a*,*b*,*c*,*d*]

for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf* **and** *A1 A2 A3* **and** *a b c d*

begin

local-setup <fn *lthy* =>

let

val *params* = {

name = *maj-4-3*,

locale-thm = @{*thm* *majcons-kstratproof-swf-explicit-axioms*},

profile-file = *SOME path* <*sat-data/maj-profiles-4-3.xz*>,

sp-file = *NONE*,

grat-file = *path* <*sat-data/maj-4-3.grat.xz*>

}

val *thm* =

Goal.prove-future *lthy* [] [] *prop* <*False*>

(*fn* {*context*, ...} =>

HEADGOAL (*resolve-tac* *context* [*Majcons-Stratproof-Impossibility.derive-false* *context* *params*]))

in

Local-Theory.note ((**binding** <*contradiction*>, []), [*thm*]) *lthy* |> *snd*

end

>

end

locale *majcons-kstratproof-swf-explicit-4-4* =

majcons-kstratproof-swf-explicit *agents* *alts* *swf* [*A1*,*A2*,*A3*,*A4*] [*a*,*b*,*c*,*d*]

for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf* **and** *A1 A2 A3 A4* **and** *a b c d*

begin

local-setup <fn *lthy* =>

let

val *params* = {

name = *maj-4-4*,

locale-thm = @{*thm* *majcons-kstratproof-swf-explicit-axioms*},

profile-file = *NONE*,

sp-file = *SOME path* <*sat-data/maj-sp-4-4.xz*>,

grat-file = *path* <*sat-data/maj-4-4.grat.xz*>

}

```

val thm =
  Goal.prove-future lthy [] [] prop <False>
    (fn {context, ...} =>
      HEADGOAL (resolve-tac context [Majcons-Stratproof-Impossibility.derive-false context
params]))
  in
    Local-Theory.note ((binding <contradiction>, []), [thm]) lthy |> snd
  end
>

```

end

context *majcons-kstratproof-swf-explicit*
begin

lemma *contradiction-ge3-eq4*:

assumes *card agents* ≥ 3 *card alts* = 4
shows *False*

proof –

from *assms* **have** *length alts-list* = 4
by *simp*

then obtain *a b c d* **where** *alts-list-eq*: *alts-list* = [*a, b, c, d*]
by (*auto simp: eval-nat-numeral length-Suc-conv*)

define *k* **where** *k* = (*card agents* – (if even (*card agents*) then 4 else 3)) div 2
from *assms* **have** *card agents* \geq (if even (*card agents*) then 4 else 3)

by *auto*
hence *k*: *card agents* = (if even (*card agents*) then 4 else 3) + 2 * *k*
unfolding *k-def* **by** *auto*

have *k* \leq *card agents*
using *k* **by** *auto*

then obtain *agents1* **where** *agents1*: *agents1* \subseteq *agents* *card agents1* = *k*
using *k* *obtain-subset-with-card-n* **by** *metis*

have *k* \leq *card (agents – agents1)*
by (*subst card-Diff-subset*) (*use agents1 finite-subset[OF agents1(1)] k in auto*)

then obtain *agents2* **where** *agents2*: *agents2* \subseteq *agents – agents1* *card agents2* = *k*
using *k* *obtain-subset-with-card-n* **by** *metis*

have [*simp*]: *finite agents1* *finite agents2*
by (*rule finite-subset[of - agents]*; *use agents1 agents2 in force*)+

interpret *dummy-ord*: *linorder-on alts of-ranking alts-list*
by (*rule linorder-of-ranking*) (*use alts-conv-alts-list distinct-alts-list in auto*)

interpret *swf-reduce-agents-even agents alts swf agents1 agents2 of-ranking alts-list*

proof

have *card (agents1 \cup agents2)* < *card agents*
by (*subst card-Un-disjoint*) (*use agents1 agents2 k in <auto split: if-splits>*)

moreover **have** *agents1 \cup agents2* \subseteq *agents*

```

    using agents1 agents2 by blast
  ultimately show agents1  $\cup$  agents2  $\subset$  agents
    by blast
qed (use agents1 agents2 in auto)

interpret new: majority-consistent-swf agents - agents1 - agents2 alts swf-low
  by (rule majority-consistent-reduce) unfold-locales
interpret new: kemeny-strategyproof-swf agents - agents1 - agents2 alts swf-low
  by (rule kemeny-strategyproof-reduce) unfold-locales
have finite (agents - agents1 - agents2)
  by (rule finite-subset[OF - finite-agents]) auto
define agents' where agents' = agents - agents1 - agents2
then obtain agents-list'
  where agents-list': mset agents-list' = mset-set agents'
  using ex-mset by blast
interpret new: majcons-kstratproof-swf-explicit
  agents - agents1 - agents2 alts swf-low agents-list' alts-list
  by unfold-locales (use agents-list' alts-list in  $\langle$ auto simp: agents'-def $\rangle$ )

have card agents' = 3  $\vee$  card agents' = 4
  using agents1 agents2 k by (simp add: card-Diff-subset agents'-def split: if-splits)
hence length agents-list' = 3  $\vee$  length agents-list' = 4
  using agents-list' by (metis size-mset size-mset-set)
thus False
proof
  assume length agents-list' = 3
  then obtain A1 A2 A3 where agents-list-eq: agents-list' = [A1, A2, A3]
    by (auto simp: eval-nat-numeral length-Suc-conv)
  interpret new2: majcons-kstratproof-swf-explicit-4-3 agents - agents1 - agents2
    alts swf-low A1 A2 A3 a b c d
  proof
    show mset [A1, A2, A3] = mset-set (agents - agents1 - agents2)
      using agents-list-eq new.agents-list by force
    qed (simp-all flip: agents-list alts-list add: agents-list-eq alts-list-eq)
    show False
      using new2.contradiction assms(1,2) by simp
  next
  assume length agents-list' = 4
  then obtain A1 A2 A3 A4 where agents-list-eq: agents-list' = [A1, A2, A3, A4]
    by (auto simp: eval-nat-numeral length-Suc-conv)
  interpret new2: majcons-kstratproof-swf-explicit-4-4 agents - agents1 - agents2
    alts swf-low A1 A2 A3 A4 a b c d
  proof
    show mset [A1, A2, A3, A4] = mset-set (agents - agents1 - agents2)
      using agents-list-eq new.agents-list by force
    qed (simp-all flip: agents-list alts-list add: agents-list-eq alts-list-eq)
    show False
      using new2.contradiction assms(1,2) by simp
  qed

```

qed

end

We now have everything to put together the final impossibility theorem.

theorem *majcons-kstratproof-impossibility*:

assumes $(card\ alts = 4 \wedge card\ agents \geq 3) \vee$
 $(card\ alts \geq 4 \wedge card\ agents \in \{9, 11, 13, 15\} \cup \{17..\})$

assumes *majority-consistent-swf agents alts swf*

assumes *kemeny-strategyproof-swf agents alts swf*

shows *False*

using *assms(1)*

proof

assume *: $card\ alts \geq 4 \wedge card\ agents \in \{9, 11, 13, 15\} \cup \{17..\}$

interpret *majority-consistent-swf agents alts swf* **by** *fact*

interpret *kemeny-strategyproof-swf agents alts swf* **by** *fact*

interpret *majcons-kstratproof-swf agents alts swf* **..**

show *False*

using *contradiction-geq9-ge4* * **by** *simp*

next

assume *: $card\ alts = 4 \wedge card\ agents \geq 3$

interpret *majority-consistent-swf agents alts swf* **by** *fact*

interpret *kemeny-strategyproof-swf agents alts swf* **by** *fact*

interpret *majcons-kstratproof-swf agents alts swf* **..**

obtain *alts-list* **where** *alts-list*: $mset\ alts\ list = mset\ set\ alts$

using *ex-mset* **by** *blast*

obtain *agents-list* **where** *agents-list*: $mset\ agents\ list = mset\ set\ agents$

using *ex-mset* **by** *blast*

interpret *majcons-kstratproof-swf-explicit agents alts swf agents-list alts-list*

by *unfold-locales (simp-all add: agents-list alts-list)*

show *False*

using * *contradiction-ge3-eq4* **by** *auto*

qed

end

References

- [1] A. Belov and J. Marques-Silva. Muser2: An efficient MUS extractor. *J. Satisf. Boolean Model. Comput.*, 8(3/4):123–128, 2012.
- [2] A. Biere, M. Fleury, and M. Heisinger. CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021. In T. Balyo, N. Froylyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, editors, *Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Report Series B*, pages 10–13. University of Helsinki, 2021.
- [3] P. Lammich. The GRAT tool chain – efficient (UN)SAT certificate checking with

formal correctness guarantees. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 457–463. Springer, 2017.

- [4] N. Wetzler, M. Heule, and W. A. H. Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014, Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.