

The Impossibility of Strategyproof Rank Aggregation

Manuel Eberl and Patrick Lederer

February 10, 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	7
1.3	The lexicographic order on lists	10
1.4	Maximal and minimal elements	11
2	Social welfare functions	14
2.1	Preference profiles	15
2.2	Definition and desirable properties of SWFs	17
2.3	Majority consistency	18
2.4	Concrete classes of SWFs	19
2.4.1	Dictatorships	19
2.4.2	Fixed-result SWFs	19
2.5	Anonymised preference profiles	20
2.6	Social Welfare Functions with explicit lists of agents and alternatives	24
2.7	Lowering constructions for SWFs	28
2.7.1	Decreasing the number of alternatives	28
2.7.2	Decreasing the number of agents by a factor	33
2.7.3	Decreasing the number of agents by an even number	35
3	Impossibility results	37
3.1	Infrastructure for SAT import and export	37
3.2	Automation for computing topological sortings	37
3.3	Automation for strategyproofness	39
3.4	Automation for majority consistency	39
3.5	For 5 alternatives and 2 agents	41
3.6	For 4 alternatives and 4 agents	42

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*: $wfp_on\ A\ R \longleftrightarrow wfp\ (\lambda x\ y.\ R\ x\ y \wedge x \in A \wedge y \in A)$
<proof>

lemma *permutations-of-set-conv-mset*:
 $finite\ A \implies permutations_of_set\ A = \{xs.\ mset\ xs = mset_set\ A\}$
<proof>

lemma *Set-filter-insert-if*:
 $Set.filter\ P\ (insert\ x\ A) = (if\ P\ x\ then\ insert\ x\ (Set.filter\ P\ A)\ else\ Set.filter\ P\ A)$
<proof>

lemma *Set-filter-insert*:
 $P\ x \implies Set.filter\ P\ (insert\ x\ A) = insert\ x\ (Set.filter\ P\ A)$
 $\neg P\ x \implies Set.filter\ P\ (insert\ x\ A) = Set.filter\ P\ A$
<proof>

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

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

lemma *image-mset-repeat-mset*: $image_mset\ f\ (repeat_mset\ n\ A) = repeat_mset\ n\ (image_mset\ f\ A)$
<proof>

lemma *filter-mset-repeat-mset*: $filter_mset\ P\ (repeat_mset\ n\ A) = repeat_mset\ n\ (filter_mset\ P\ A)$
<proof>

lemma *mset-eq-mset-set-iff*:
assumes $finite\ A$
shows $mset\ xs = mset_set\ A \longleftrightarrow xs \in permutations_of_set\ A$
<proof>

lemma *size-Diff-mset-same-size*:
fixes $A\ B :: 'a\ multiset$
assumes $size\ (A - B) = n\ size\ A = size\ B$
shows $size\ (B - A) = n$

$\langle proof \rangle$

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

fixes $f :: 'a \Rightarrow 'b$

assumes *inj-on* f (*set-mset* ($A+B$))

shows *image-mset* f ($A - B$) = *image-mset* f A - *image-mset* f B

$\langle proof \rangle$

context *preorder-on*

begin

sublocale *dual: preorder-on carrier* $\lambda x y. le\ y\ x$

$\langle proof \rangle$

end

context *order-on*

begin

sublocale *dual: order-on carrier* $\lambda x y. le\ y\ x$

$\langle proof \rangle$

end

context *total-preorder-on*

begin

sublocale *dual: total-preorder-on carrier* $\lambda x y. le\ y\ x$

$\langle proof \rangle$

end

context *linorder-on*

begin

sublocale *dual: linorder-on carrier* $\lambda x y. le\ y\ x$

$\langle proof \rangle$

end

context *finite-linorder-on*

begin

sublocale *dual*: *finite-linorder-on carrier* $\lambda x y. le\ y\ x$
 $\langle proof \rangle$

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*
 $\langle proof \rangle$

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*
 $\langle proof \rangle$

context *order-on*
begin

lemma *order-on-restrict*:
 $order-on\ (carrier \cap A)\ (restrict-relation\ A\ le)$
 $\langle proof \rangle$

lemma *order-on-restrict-subset*:
 $A \subseteq carrier \implies order-on\ A\ (restrict-relation\ A\ le)$
 $\langle proof \rangle$

end

context *linorder-on*
begin

lemma *linorder-on-restrict*:
 $linorder-on\ (carrier \cap A)\ (restrict-relation\ A\ le)$
 $\langle proof \rangle$

lemma *linorder-on-restrict-subset*:

$A \subseteq \text{carrier} \implies \text{linorder-on } A \text{ (restrict-relation } A \text{ le)}$
 $\langle \text{proof} \rangle$

end

lemma *linorder-on-concat*:

assumes *linorder-on* A R *linorder-on* B S $A \cap B = \{\}$

shows *linorder-on* $(A \cup B)$ $(\lambda x y. \text{if } x \in A \text{ then } R \ x \ y \ \vee \ y \in B \text{ else } S \ x \ y)$

$\langle \text{proof} \rangle$

lemma *linorder-on-prepend*:

assumes *linorder-on* A R $z \notin A$

shows *linorder-on* $(\text{insert } z \ A)$ $(\lambda x y. \text{if } x = z \text{ then } y \in \text{insert } z \ A \text{ else } R \ x \ y)$

$\langle \text{proof} \rangle$

lemma *finite-linorder-on-exists*:

assumes *finite* A

shows $\exists R. \text{linorder-on } A \ R$

$\langle \text{proof} \rangle$

context *order-on*

begin

lemma *order-on-map*:

assumes *bij-betw* f A *carrier*

shows *order-on* A $(\text{restrict-relation } A \text{ (map-relation } f \text{ le)})$

$\langle \text{proof} \rangle$

end

context *linorder-on*

begin

lemma *linorder-on-map*:

assumes *bij-betw* f A *carrier*

shows *linorder-on* A $(\text{restrict-relation } A \text{ (map-relation } f \text{ le)})$

$\langle \text{proof} \rangle$

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>
 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* $R\ s\ x\ y \wedge R. R \in \#Rs \implies \text{preorder-on } A\ R$
shows $x \in A\ y \in A$
 <proof>

lemma *majority-mset-refl-iff'*: *majority-mset* $R\ s\ x\ x \longleftrightarrow (\exists R \in \#Rs. R\ x\ x)$
 <proof>

lemma *majority-mset-refl-iff*:
assumes $\bigwedge R. R \in \#Rs \implies \text{preorder-on } A\ R\ Rs \neq \{\#\}$
shows *majority-mset* $R\ s\ x\ x \longleftrightarrow x \in A$
 <proof>

lemma *majority-mset-refl*:
assumes $\bigwedge R. R \in \#Rs \implies \text{preorder-on } A\ R\ Rs \neq \{\#\}$ $x \in A$
shows *majority-mset* $R\ s\ x\ x$
 <proof>

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

lemma *majority-mset-iff*:
assumes $\bigwedge R. R \in \#Rs \implies \text{preorder-on } A\ R\ Rs \neq \{\#\}$ $x \in A\ y \in A$

shows $\text{majority-mset } R s \ x \ y \longleftrightarrow$
 $\text{size } (\text{filter-mset } (\lambda R. R \ x \ y) \ R s) \geq \text{size } (\text{filter-mset } (\lambda R. R \ y \ x) \ R s)$
 $\langle \text{proof} \rangle$

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 $\text{majority-mset } R s \ x \ y \longleftrightarrow$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R \ x \ y) \ R s) \geq \text{size } R s$
 $\langle \text{proof} \rangle$

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 $\text{majority-mset } R s \ x \ y \longleftrightarrow$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R \ y \ x) \ R s) \leq \text{size } R s$
 $\langle \text{proof} \rangle$

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

assumes $\bigwedge R. R \in \# \ R s \implies \text{linorder-on } A \ R \ R s \neq \{\#\} \ x \in A \ y \in A$
shows $x \prec [\text{majority-mset } R s] \ y \longleftrightarrow x \neq y \wedge$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R \ x \ y) \ R s) > \text{size } R s$
 $\langle \text{proof} \rangle$

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

assumes $\bigwedge R. R \in \# \ R s \implies \text{linorder-on } A \ R \ R s \neq \{\#\} \ x \in A \ y \in A$
shows $x \prec [\text{majority-mset } R s] \ y \longleftrightarrow$
 $2 * \text{size } (\text{filter-mset } (\lambda R. R \ y \ x) \ R s) < \text{size } R s$
 $\langle \text{proof} \rangle$

context *preorder-family*

begin

lemma *majority-iff'*:

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

lemma *majority-iff*:

assumes $x \in \text{carrier} \ y \in \text{carrier}$
shows $\text{majority } R \ x \ y \longleftrightarrow \text{card } \{i \in \text{dom. } R \ i \ x \ y\} \geq \text{card } \{i \in \text{dom. } R \ i \ y \ x\}$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *strongly-preferred-majority-iff*:

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

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

<proof>

lemma *majority-not-outside*:

assumes *majority* $R \ x \ y$

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

<proof>

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>

lemma *majority-Pareto2*:

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

shows *majority* $R \ x \ z$

<proof>

lemma *majority-conv-majority-mset*:

assumes *finite dom*

shows *majority* $R = \text{majority-mset } (\text{image-mset } R \ (\text{mset-set dom}))$ (**is** *?lhs = ?rhs*)

<proof>

end

context *linorder-family*

begin

lemma *majority-iff-ge-half*:

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

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

<proof>

lemma *majority-iff-le-half*:

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

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

<proof>

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>

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*) $\longleftrightarrow$ $x \preceq[R] y \wedge (x \prec[R] y \vee \text{lexprod-list-aux } R \text{ } xs \text{ } ys)$

lemma *lexprod-list-aux-Nil-right-iff* [simp]: *lexprod-list-aux* *R* *xs* [] $\longleftrightarrow$ *xs* = []
 <proof>

lemma *lexprod-list-aux-refl*: $(\forall x \in \text{set } xs. R \text{ } x \text{ } x) \implies \text{lexprod-list-aux } R \text{ } xs \text{ } xs$
 <proof>

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

definition *lexprod-length-list* :: nat $\Rightarrow$ 'a relation $\Rightarrow$ 'a list relation **where**
lexprod-length-list *n* *R* = *restrict-relation* {*xs*. *length xs* = *n*} (*lexprod-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*
 <proof>

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

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

end

context *total-preorder-on*
begin

lemma *total-preorder-lexprod-list*: *total-preorder-on* (*lists carrier*) (*lexprod-list* *le*)
 <proof>

lemma *total-preorder-lexprod-length-list*:

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

end

context *order-on*
begin

lemma *order-lexprod-list*: *order-on* (*lists carrier*) (*lexprod-list le*)
 $\langle \text{proof} \rangle$

lemma *order-lexprod-length-list*:
order-on $\{xs. \text{ set } xs \subseteq \text{ carrier} \wedge \text{ length } xs = n\}$ (*lexprod-length-list* n le)
 $\langle \text{proof} \rangle$

end

context *linorder-on*
begin

lemma *order-lexprod-list*: *linorder-on* (*lists carrier*) (*lexprod-list le*)
 $\langle \text{proof} \rangle$

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

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* $R \ A = \text{Min-wrt-among } R' \ A$
 $\langle \text{proof} \rangle$

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* $R = \{x. R \ x \ x \wedge (\forall y. R \ y \ x \longrightarrow R \ x \ y)\}$
 $\langle \text{proof} \rangle$

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

$y\ x) \ A$
 $\langle \text{proof} \rangle$

context *preorder-on*
begin

lemma *Min-wrt-among-preorder:*
 $\text{Min-wrt-among } le\ A = \{x \in \text{carrier} \cap A. \forall y \in \text{carrier} \cap A. le\ y\ x \longrightarrow le\ x\ y\}$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-preorder:*
 $\text{Min-wrt } le = \{x \in \text{carrier}. \forall y \in \text{carrier}. le\ y\ x \longrightarrow le\ x\ y\}$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-among-subset:*
 $\text{Min-wrt-among } le\ A \subseteq \text{carrier } \text{Min-wrt-among } le\ A \subseteq A$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-subset:*
 $\text{Min-wrt } le \subseteq \text{carrier}$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-among-nonempty:*
assumes $B \cap \text{carrier} \neq \{\}$ *finite* $(B \cap \text{carrier})$
shows $\text{Min-wrt-among } le\ B \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-nonempty:*
 $\text{carrier} \neq \{\} \implies \text{finite } \text{carrier} \implies \text{Min-wrt } le \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-among-map-relation-vimage:*
 $f - ' \text{Min-wrt-among } le\ A \subseteq \text{Min-wrt-among } (\text{map-relation } f\ le)\ (f - ' A)$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-map-relation-vimage:*
 $f - ' \text{Min-wrt } le \subseteq \text{Min-wrt } (\text{map-relation } f\ le)$
 $\langle \text{proof} \rangle$

lemma *Min-wrt-among-map-relation-bij-subset:*
assumes $\text{bij } (f :: 'a \Rightarrow 'b)$
shows $f - ' \text{Min-wrt-among } le\ A \subseteq$
 $\text{Min-wrt-among } (\text{map-relation } (\text{inv } f)\ le)\ (f - ' A)$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\text{bij } f \implies f ' \text{Min-wrt } le = \text{Min-wrt } (\text{map-relation } (\text{inv } f) \text{ } le)$

$\langle \text{proof} \rangle$

lemma *Min-wrt-among-mono:*

$le \ y \ x \implies x \in \text{Min-wrt-among } le \ A \implies y \in A \implies y \in \text{Min-wrt-among } le \ A$

$\langle \text{proof} \rangle$

lemma *Min-wrt-mono:*

$le \ y \ x \implies x \in \text{Min-wrt } le \implies y \in \text{Min-wrt } le$

$\langle \text{proof} \rangle$

end

context *total-preorder-on*

begin

lemma *Min-wrt-among-total-preorder:*

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

$\langle \text{proof} \rangle$

lemma *Min-wrt-total-preorder:*

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

$\langle \text{proof} \rangle$

lemma *decompose-Min:*

assumes $A: A \subseteq \text{carrier}$

defines $M \equiv \text{Min-wrt-among } le \ A$

shows $\text{restrict-relation } A \ le = (\lambda x \ y. x \in M \wedge y \in A \vee (y \notin M \wedge \text{restrict-relation } (A - M) \ le \ x \ y))$

$\langle \text{proof} \rangle$

end

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

$\text{min-wrt-among } R \ A = \text{the-elem } (\text{Min-wrt-among } R \ A)$

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

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

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

$\text{max-wrt-among } R \ A = \text{the-elem } (\text{Max-wrt-among } R \ A)$

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 $is-singleton\ (Max-wrt-among\ le\ A)$
 $\langle proof \rangle$

lemma *max-wrt-among-inside:*
assumes $A \neq \{\}$ $A \subseteq carrier$
shows $max-wrt-among\ le\ A \in A$
 $\langle proof \rangle$

lemma *le-max-wrt-among:*
assumes $y \in A$ $A \subseteq carrier$
shows $le\ y\ (max-wrt-among\ le\ A)$
 $\langle proof \rangle$

end

context *finite-linorder-on*
begin

lemma *Min-wrt-among-singleton:*
assumes $A \neq \{\}$ $A \subseteq carrier$
shows $is-singleton\ (Min-wrt-among\ le\ A)$
 $\langle proof \rangle$

lemma *min-wrt-among-inside:*
assumes $A \neq \{\}$ $A \subseteq carrier$
shows $min-wrt-among\ le\ A \in A$
 $\langle proof \rangle$

lemma *le-min-wrt-among:*
assumes $y \in A$ $A \subseteq carrier$
shows $le\ (min-wrt-among\ le\ A)\ y$
 $\langle proof \rangle$

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]: agents ≠ {} and nonempty-alts [simp]: alts ≠ {}
  assumes prefs-wf [simp]: i ∈ agents ⇒ finite-linorder-on alts (R i)
  assumes prefs-undefined [simp]: i ∉ agents ⇒ ¬R i x y
begin

```

```

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

```

```

lemma prefs-wf' [simp]:
  i ∈ agents ⇒ linorder-on alts (R i)
⟨proof⟩

```

```

lemma not-outside:
  assumes x ≤[R i] y
  shows i ∈ agents x ∈ alts y ∈ alts
⟨proof⟩

```

```

sublocale linorder-family agents alts R
⟨proof⟩

```

```

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

```

```

lemma wf-update:
  assumes i ∈ agents linorder-on alts Ri'
  shows pref-profile-linorder-wf agents alts (R(i := Ri'))
⟨proof⟩

```

```

lemma wf-permute-agents:
  assumes σ permutes agents
  shows pref-profile-linorder-wf agents alts (R ∘ σ)
⟨proof⟩

```

```

lemma (in −) pref-profile-eqI:
  assumes pref-profile-linorder-wf agents alts R1 pref-profile-linorder-wf agents alts R2
  assumes ∧x. x ∈ agents ⇒ R1 x = R2 x
  shows R1 = R2

```

<proof>

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 *card agents* ≤ 2 **and** [*simp*]: *finite agents*
 assumes *linorder-on alts* (*majority R*)
 shows *majority R* = *Pareto R*
<proof>

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

abbreviation *is-pref-profile* $\equiv$ *pref-profile-linorder-wf agents alts*

lemma *finite-linorder-on-iff'* [*simp*]:
 finite-linorder-on alts R $\longleftrightarrow$ *linorder-on alts R*
<proof>

lemma *finite-pref-profiles* [*intro*]: *finite* {*R. is-pref-profile R*}
 and *card-pref-profiles*: *card* {*R. is-pref-profile R*} = *fact* (*card alts*) $\wedge$ *card agents*
<proof>

lemma *pref-profile-exists*: $\exists R. is-pref-profile R$
<proof>

lemma *pref-profile-wfI'* [*intro?*]:
 ($\bigwedge i. i \in agents \implies linorder-on alts (R i)$) $\implies$
 ($\bigwedge i. i \notin agents \implies R i = (\lambda -. False)$) $\implies is-pref-profile R$
<proof>

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

lemma *election* [*simp*, *intro*]: *linorder-election agents alts*
<proof>

end

2.2 Definition and desirable properties of SWFs

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

lemma *swf-wf'*:
assumes *is-pref-profile* *R*
shows *finite-linorder-on alts* (*swf* *R*)
 <proof>

end

lemma (in *linorder-election*) *social-welfare-functionI* [intro]:
 ($\bigwedge R. \text{is-pref-profile } R \Rightarrow \text{linorder-on alts } (swf\ R)$) $\Rightarrow$ *social-welfare-function agents alts swf*
 <proof>

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*: π *permutes agents* $\Rightarrow$ *is-pref-profile* *R* $\Rightarrow$ *swf* (*R* $\circ$ π) = *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>

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*: σ *permutes alts* $\Rightarrow$ *is-pref-profile* *R* $\Rightarrow$
 swf (*map-relation* $\sigma \circ R$) = *map-relation* σ (*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* $\Rightarrow$ $\forall i \in \text{agents}. x \succ [R\ i] y \Rightarrow x \succ [swf\ R] y$
begin

lemma *unanimous'*:
assumes *is-pref-profile* $R \ \forall i \in \text{agents}. x \succeq [R \ i] \ y$
shows $x \succeq [\text{swf } R] \ y$
 $\langle \text{proof} \rangle$

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

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*:
 $\text{is-pref-profile } R \implies i \in \text{agents} \implies \text{linorder-on alts } R' \implies$
 $\text{swap-dist-relation } (R \ i) (\text{swf } R) \leq \text{swap-dist-relation } (R \ i) (\text{swf } (R(i := R')))$

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*:
 $\text{is-pref-profile } R \implies \text{linorder-on alts } (\text{majority } R) \implies \text{swf } R = \text{majority } R$

locale *majcons-kstratproof-swf* =
 $\text{majority-consistent-swf agents alts swf} +$
 $\text{kemeny-strategyproof-swf agents alts swf}$
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
   $\langle \text{proof} \rangle$ 

```

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  $\implies i \in \text{agents} \implies \text{linorder-on alts } S \implies$ 
     $\text{linorder-on alts } (\text{majority } (R(i := S))) \implies$ 
     $\text{swap-dist-relation } (R \ i) \ (\text{swf } R) \leq \text{swap-dist-relation } (R \ i) \ (\text{majority } (R(i := S)))$ 

```

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  $\in \text{agents}$ 
begin

```

```

  sublocale social-welfare-function agents alts  $\lambda R. R \ \text{dictator}$ 
   $\langle \text{proof} \rangle$ 

```

```

  sublocale neutral-swf agents alts  $\lambda R. R \ \text{dictator}$ 
   $\langle \text{proof} \rangle$ 

```

```

  sublocale unanimous-swf agents alts  $\lambda R. R \ \text{dictator}$ 
   $\langle \text{proof} \rangle$ 

```

```

  sublocale kemeny-strategyproof-swf agents alts  $\lambda R. R \ \text{dictator}$ 
   $\langle \text{proof} \rangle$ 

```

```

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  $\lambda$ -. ranking
   $\langle$ proof $\rangle$ 

sublocale anonymous-swf agents alts  $\lambda$ -. ranking
   $\langle$ proof $\rangle$ 

sublocale kemeny-strategyproof-swf agents alts  $\lambda$ -. ranking
   $\langle$ proof $\rangle$ 

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'
 $\langle$ proof $\rangle$ 

```

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* $\wedge$ ($\forall R \in \#Rs$. *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** $is\text{-}pref\text{-}profile\ R\ Rs = image\text{-}mset\ R\ (mset\text{-}set\ agents)$
 $\langle proof \rangle$

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 $aswf\text{-}welldefined$:
assumes $is\text{-}pref\text{-}profile\ R$
defines $R_s \equiv image\text{-}mset\ R\ (mset\text{-}set\ agents)$
shows $aswf\ R_s = swf\ R$
 $\langle proof \rangle$

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

lemma $aswf\text{-}wf$:
assumes $is\text{-}apref\text{-}profile\ Rs$
shows $linorder\text{-}on\ alts\ (aswf\ Rs)$
 $\langle proof \rangle$

lemma $aswf\text{-}wf'$:
assumes $is\text{-}apref\text{-}profile\ Rs$
shows $finite\text{-}linorder\text{-}on\ alts\ (aswf\ Rs)$
 $\langle proof \rangle$

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 $aswf'\ Rs = ranking\ (aswf\ (image\text{-}mset\ of\text{-}ranking\ Rs))$

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

lemma $is\text{-}apref\text{-}profile'\text{-}imp\text{-}is\text{-}apref\text{-}profile$:
assumes $is\text{-}apref\text{-}profile'\ Rs$
shows $is\text{-}apref\text{-}profile\ (image\text{-}mset\ of\text{-}ranking\ Rs)$
 $\langle proof \rangle$

lemma $aswf'\text{-}wf$:
assumes $is\text{-}apref\text{-}profile'\ Rs$
shows $aswf'\ Rs \in permutations\text{-}of\text{-}set\ alts$
 $\langle proof \rangle$

end

locale $anonymous\text{-}unanimous\text{-}swf =$
 $anonymous\text{-}swf\ agents\ alts\ swf +$

unanimous-swf agents alts swf
for *agents* :: 'agent set **and** *alts* :: 'alt set **and** *swf*
begin

lemma *unanimous-aswf*:
assumes *is-apref-profile* *Rs* $\forall R \in \#Rs. x \succ [R] y$
shows $x \succ [aswf\ Rs] y$
 $\langle proof \rangle$

lemma *unanimous-aswf'*:
assumes *is-apref-profile* *Rs* $\forall R \in \#Rs. x \succeq [R] y$
shows $x \succeq [aswf\ Rs] y$
 $\langle proof \rangle$

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

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

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 $\exists R \in \#(R1 - R2). swap-dist-relation\ R\ S1 > swap-dist-relation\ R\ S2$
shows $aswf\ R1 \neq S1 \vee aswf\ R2 \neq S2$
 $\langle proof \rangle$

lemma *kemeny-strategyproof-aswf-strong*:
assumes *is-apref-profile* *R1* *is-apref-profile* *R2*
assumes $size\ (R1 - R2) = 1$
assumes $(\exists R \in \#R1 - R2. swap-dist-relation\ R\ S1 > swap-dist-relation\ R\ S2) \vee$
 $(\exists R \in \#R2 - R1. swap-dist-relation\ R\ S2 > swap-dist-relation\ R\ S1)$
shows $aswf\ R1 \neq S1 \vee aswf\ R2 \neq S2$
 $\langle proof \rangle$

lemma *kemeny-strategyproof-aswf'*:

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

lemma *kemeny-strategyproof-aswf'-strong*:
assumes *is-apref-profile'* $R1$ *is-apref-profile'* $R2$
assumes $size (R1 - R2) = 1$
assumes $(\exists R \in \#(R1 - R2). swap-dist R S1 > swap-dist R S2) \vee$
 $(\exists R \in \#(R2 - R1). swap-dist R S2 > swap-dist R S1)$
shows $aswf' R1 \neq S1 \vee aswf' R2 \neq S2$
<proof>

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 $R1 - R2 = replicate-mset n A$
shows $swap-dist A (aswf' R1) \leq swap-dist A (aswf' R2)$
<proof>

lemma *kemeny-strategyproof-aswf'-clones*:
assumes *is-apref-profile'* $R1$ *is-apref-profile'* $R2$
assumes $R1 - R2 = replicate-mset n A$
assumes $swap-dist A S1 > swap-dist A S2$
shows $aswf' R1 \neq S1 \vee aswf' R2 \neq S2$
<proof>

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 *is-apref-profile'* R *is-apref-profile'* R' $add-mset S R' = add-mset S' R$
shows $aswf' R = S \vee aswf' R' \neq S$
<proof>

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 $R_s S \longleftrightarrow$

$\text{majority-mset} (\text{image-mset of-ranking } Rs) = \text{of-ranking } S \wedge \text{distinct } S$

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 Rs linorder-on alts (majority-mset Rs)
shows aswf Rs = majority-mset Rs
 ⟨proof⟩

lemma *majority-consistent-aswf'*:
assumes is-apref-profile' Rs majority-rel-mset Rs S
shows aswf' Rs = S
 ⟨proof⟩

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
 ⟨proof⟩

lemma *alts-conv-alts-list*: alts = set alts-list
 ⟨proof⟩

lemma *card-alts [simp]*: card alts = length alts-list
 ⟨proof⟩

lemma *distinct-agents-list*: distinct agents-list
 ⟨proof⟩

lemma *agents-conv-agents-list*: $agents = set\ agents\text{-}list$
 ⟨proof⟩

lemma *card-agents*: $card\ agents = length\ agents\text{-}list$
 ⟨proof⟩

lemma *mset-eq-alts-list-iff*: $mset\ xs = mset\ alts\text{-}list \longleftrightarrow distinct\ xs \wedge set\ xs = alts$
 ⟨proof⟩

lemma *mset-eq-agents-list-iff*: $mset\ xs = mset\ agents\text{-}list \longleftrightarrow distinct\ xs \wedge set\ xs = agents$
 ⟨proof⟩

definition *prefs-from-rankings*
 :: 'alt list list $\Rightarrow$ ('agent $\Rightarrow$ 'alt relation) **where**
prefs-from-rankings rs =
 ($\lambda i.$ if $i \in agents$ then *of-ranking* (rs ! index *agents-list* i) else ($\lambda -.$ False))

definition *prefs-from-rankings-wf* :: 'alt list list $\Rightarrow$ bool **where**
prefs-from-rankings-wf rs $\longleftrightarrow$
 length rs = card agents $\wedge$ list-all ($\lambda r.$ mset r = mset *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⟩

lemma *prefs-from-rankings-nth*:
assumes *prefs-from-rankings-wf* R1 $i < card\ agents$
shows *prefs-from-rankings* R1 (*agents-list* ! i) = *of-ranking* (R1 ! i)
 ⟨proof⟩

lemma *prefs-from-rankings-outside*:
assumes $i \notin agents$
shows *prefs-from-rankings* R1 i = ($\lambda -.$ False)
 ⟨proof⟩

lemma *prefs-from-rankings-update*:
assumes *prefs-from-rankings-wf* R1 $i < card\ agents$ mset xs = mset *alts-list*
shows *prefs-from-rankings* (R1[i := xs]) =
 (*prefs-from-rankings* R1)(*agents-list* ! i := *of-ranking* xs)
 ⟨proof⟩

lemma *prefs-from-rankings-wf-update*:
assumes *prefs-from-rankings-wf* R1 $i < card\ agents$ mset xs = mset *alts-list*
shows *prefs-from-rankings-wf* (R1[i := xs])
 ⟨proof⟩

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>

lemma *majority-prefs-from-rankings-eq-of-ranking*:
assumes $\text{prefs-from-rankings-wf } R$ $\text{majority-rel-mset} (\text{mset } R)$ ys
shows $\text{majority} (\text{prefs-from-rankings } R) = \text{of-ranking } ys$
 <proof>

lemma *majority-rel-mset-imp-mset*:
assumes $\text{prefs-from-rankings-wf } R$ $\text{majority-rel-mset} (\text{mset } R)$ xs
shows $\text{mset } xs = \text{mset alts-list}$
 <proof>

end

locale *social-welfare-function-explicit* =
social-welfare-function agents alts swf +
linorder-election-explicit agents alts agents-list alts-list
for $\text{agents} :: \text{'agent set}$ **and** $\text{alts} :: \text{'alt set}$ **and** $\text{swf agents-list alts-list}$
begin

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

lemma $\text{swf}'\text{-wf: } \text{prefs-from-rankings-wf } R \Longrightarrow \text{mset} (\text{swf}' R) = \text{mset-set alts}$
 <proof>

end

locale *majority-consistent-swf-explicit* =
social-welfare-function-explicit agents alts swf agents-list alts-list +
majority-consistent-swf agents alts swf
for $\text{agents} :: \text{'agent set}$ **and** $\text{alts} :: \text{'alt set}$ **and** $\text{swf agents-list alts-list}$
begin

lemma *majority-consistent-swf'*:
assumes $\text{prefs-from-rankings-wf } R$ $\text{majority-rel-mset} (\text{mset } R)$ ys
shows $\text{swf}' R = ys$
 <proof>

end

locale *majcons-kstratproof-swf-explicit* =
social-welfare-function-explicit agents alts swf agents-list alts-list +
majcons-kstratproof-swf agents alts swf
for $\text{agents} :: \text{'agent set}$ **and** $\text{alts} :: \text{'alt set}$ **and** $\text{swf agents-list alts-list}$
begin

sublocale *majority-consistent-swf-explicit* $\langle \text{proof} \rangle$

sublocale *majority-consistent-weak-kstratproof-swf*
 $\langle \text{proof} \rangle$

lemma *distinct-alts-list-aux: distinct alts-list*
 $\langle \text{proof} \rangle$

lemma *distinct-agents-list-aux: distinct agents-list*
 $\langle \text{proof} \rangle$

lemma *prefs-from-rankings-wf-iff:*
prefs-from-rankings-wf $xss \longleftrightarrow$
 $\text{length } xss = \text{length agents-list} \wedge \text{list-all } (\lambda ys. \text{mset } ys = \text{mset alts-list}) \ xss$
 $\langle \text{proof} \rangle$

lemma *swf'-in-all-rankings:*
assumes *prefs-from-rankings-wf* xss *permutations-of-set-list* $\text{alts-list} = yss$
shows $\text{list-ex } (\lambda ys. \text{swf}' \ xss = ys) \ yss$
 $\langle \text{proof} \rangle$

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

lemma *kemeny-strategyproof-swf'-aux:*
assumes *prefs-from-rankings-wf* xss *prefs-from-rankings-wf* yss
assumes $\text{map } (\text{index } ys) \ S1 = S1' \ \text{map } (\text{index } ys) \ S2 = S2'$
assumes $\text{inversion-number } S1' = d1 \ \text{inversion-number } S2' = d2$
assumes $d1 > d2 \wedge i < \text{length agents-list} \wedge ys = xss \ ! \ i \wedge yss = xss[i := zs]$
shows $\text{swf}' \ xss \neq S1 \vee \text{swf}' \ yss \neq S2$
 $\langle \text{proof} \rangle$

end

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

sublocale *majority-consistent-swf-explicit* $\text{agents alts swf agents-list alts-list} \langle \text{proof} \rangle$

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⟩

```

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⟩

```

```

sublocale new: linorder-election agents alts'
⟨proof⟩

```

```

lemma dummy-alts: distinct dummy-alts set dummy-alts = alts - alts'
⟨proof⟩

```

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 ⇒ '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)
 $\langle \text{proof} \rangle$

lemma *restrict-extend-ranking*:
assumes *linorder-on alts'* R
shows *restrict-relation alts'* (*extend-ranking* R) $= R$
 $\langle \text{proof} \rangle$

lemma *swap-dist-extend-ranking*:
assumes *linorder-on alts'* R *linorder-on alts'* S
shows *swap-dist-relation* (*extend-ranking* R) (*extend-ranking* S) $= \text{swap-dist-relation } R S$
 $\langle \text{proof} \rangle$

lemma *extend-ranking-eq-iff*:
assumes $\bigwedge x y. R x y \implies x \in \text{alts}' \wedge y \in \text{alts}' \bigwedge x y. S x y \implies x \in \text{alts}' \wedge y \in \text{alts}'$
shows *extend-ranking* $R = \text{extend-ranking } S \longleftrightarrow R = S$
 $\langle \text{proof} \rangle$

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

lemma *count-extend-ranking-multiset*:
assumes $\bigwedge R. R \in \# Rs \implies \text{linorder-on alts}' R$ **and** *xy*: $x \in \text{alts } y \in \text{alts}$
shows *size* $\{\#R \in \#Rs. \text{extend-ranking } R x y\} =$
 $(\text{if } x \in \text{alts}' \wedge y \in \text{alts}' \text{ then } \text{size } \{\#R \in \#Rs. R x y\}$
 $\text{else if } x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking dummy-alts } x y) \text{ then } \text{size } Rs \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *count-extend-profile*:
assumes *new.is-pref-profile* R **and** *xy*: $x \in \text{alts } y \in \text{alts}$
shows *card* $\{i \in \text{agents}. \text{extend-profile } R i x y\} =$
 $(\text{if } x \in \text{alts}' \wedge y \in \text{alts}' \text{ then } \text{card } \{i \in \text{agents}. R i x y\}$
 $\text{else if } x \notin \text{alts}' \wedge (y \in \text{alts}' \vee \text{of-ranking dummy-alts } x y) \text{ then } \text{card agents} \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *majority-extend-profile*:
assumes *new.is-pref-profile* R
shows *majority* (*extend-profile* R) $= \text{extend-ranking } (\text{majority } R)$

<proof>

lemma *majority-mset-extend-profile:*

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

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

<proof>

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* :: $('agent \Rightarrow 'alt \text{ relation}) \Rightarrow 'alt \text{ relation}$ **where**

$\text{swf-low } R = \text{restrict-relation } \text{alts}' (\text{swf } (\text{extend-profile } R))$

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

<proof>

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>

lemma *unanimous-restrict:*

assumes *unanimous-swf agents alts swf*

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

<proof>

lemma *majority-consistent-restrict:*

assumes *majority-consistent-swf agents alts swf*

shows *majority-consistent-swf agents alts' swf-low*

<proof>

end

locale *unanimous-swf-restrict-alts* =

$\text{swf-restrict-alts agents alts swf dummy-alts alts}' +$

$\text{unanimous-swf agents alts swf}$

for *agents* :: $'agent \text{ set}$ **and** *alts* :: $'alt \text{ set}$ **and** $\text{swf dummy-alts alts}'$

begin

sublocale *new: unanimous-swf agents alts' swf-low*

<proof>

lemma *swf-dummy-alts-least-preferred:*

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

shows $x \succ_{[\text{swf } (\text{extend-profile } R)]} y$

<proof>

lemma *swf-strongly-preferred-dummy-alts*:

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

assumes $x \succ_{[\text{of-ranking dummy-alts}]} y$

shows $x \succ_{[\text{swf (extend-profile } R)]} y$

<proof>

lemma *swf-preferred-dummy-alts-iff*:

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

shows $x \succeq_{[\text{of-ranking dummy-alts}]} y \longleftrightarrow x \succeq_{[\text{swf (extend-profile } R)]} y$

<proof>

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 $x \succ_{[\text{swf (extend-profile } R)]} y \longleftrightarrow x \succ_{[\text{of-ranking dummy-alts}]} y$

<proof>

lemma *extend-ranking-swf-low*:

assumes *new.is-pref-profile* R

shows *extend-ranking (swf-low* R) = *swf (extend-profile* R)

<proof>

lemma *kemeny-strategyproof-restrict*:

assumes *kemeny-strategyproof-swf* *agents* *alts* *swf*

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

<proof>

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*

<proof>

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

<proof>

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*
 $\langle \text{proof} \rangle$

sublocale *new: social-welfare-function-explicit agents alts' swf-low agents-list alts-list'*
 $\langle \text{proof} \rangle$

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 = $\{\}$*
 $\langle \text{proof} \rangle$

lemma *wf-extend:*
assumes *new.prefs-from-rankings-wf R*
shows *prefs-from-rankings-wf (map extend R)*
 $\langle \text{proof} \rangle$

lemma *of-ranking-extend:*
assumes *mset xs = mset alts-list'*
shows *of-ranking (extend xs) = extend-ranking (of-ranking xs)*
 $\langle \text{proof} \rangle$

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*
 $\langle \text{proof} \rangle$

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

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) $\longleftrightarrow$ majority-rel-mset (mset R) S*
 $\langle \text{proof} \rangle$

lemma *new-swf'-eq:*
assumes *R: new.prefs-from-rankings-wf R*
shows *new.swf' R = filter ($\lambda x. x \in \text{alts}'$) (swf' (map extend R))*
 $\langle \text{proof} \rangle$

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 $A \setminus B$ to the original element in B that it was cloned from.

```

locale cloning =
  fixes  $A\ B\ unclone$ 
  assumes  $subset: A \subseteq B$ 
  assumes  $finite: finite\ B$ 
  assumes  $unclone: \bigwedge x. x \in B \implies unclone\ x \in A$ 
  assumes  $unclone-ident: \bigwedge x. x \in A \implies unclone\ x = x$ 
  assumes  $card-unclone:$ 
     $x \in A \implies y \in A \implies card\ (unclone - ' \{x\} \cap B) = card\ (unclone - ' \{y\} \cap B)$ 
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  $i$ )
  <proof>

```

```

lemma clones-outside:  $i \notin A \implies clones\ i = \{\}$ 
  <proof>

```

```

lemma card-clones':
  assumes  $i \in A$ 
  shows card (clones  $i$ ) * card  $A = card\ B$ 
  <proof>

```

```

lemma card-clones:
  assumes  $i \in A$ 
  shows card (clones  $i$ ) = factor
  <proof>

```

```

lemma image-mset-unclone:
  image-mset unclone (mset-set  $B$ ) = repeat-mset factor (mset-set  $A$ )
  (is ?lhs = ?rhs)
  <proof>

```

```

lemma factor-pos:  $B \neq \{\} \implies factor > 0$ 
  <proof>

```

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* $B \setminus A \neq \{\}$ $\text{card } A \text{ dvd } \text{card } B$

shows $\exists \text{ unclone. cloning } A \ B \ \text{unclone}$

<proof>

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'* $\neq \{\}$

<proof>

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

<proof>

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 $\Rightarrow$ 'alt relation) $\Rightarrow$ 'agent $\Rightarrow$ 'alt relation **where**

extend-profile *R* *i* = (if *i* $\in$ *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>

lemma *count-extend-profile*:

$\text{card } \{i \in \text{agents. } \text{extend-profile } R \ i \ x \ y\} = \text{clone.factor} * \text{card } \{i \in \text{agents'}. \ R \ i \ x \ y\}$

<proof>

lemma *majority-extend-profile*:

assumes *new.is-pref-profile* *R*

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

<proof>

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*⟩

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*⟩

Unanimity is obviously preserved as well.

lemma *unanimous-clone:*
assumes *unanimous-swf agents alts swf*
shows *unanimous-swf agents' alts swf-low*
 ⟨*proof*⟩

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*⟩

lemma *majority-consistent-clone:*
assumes *majority-consistent-swf agents alts swf*
shows *majority-consistent-swf agents' alts swf-low*
 ⟨*proof*⟩

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*
 $\langle proof \rangle$

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

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

lemma *is-pref-profile-extend-profile* [intro]:
assumes *new.is-pref-profile R*
shows *is-pref-profile* (*extend-profile R*)
 $\langle proof \rangle$

lemma *count-extend-profile*:
assumes *new.is-pref-profile R* $x \in alts$ $y \in alts$
shows $card\ \{i \in agents. extend-profile\ R\ i\ x\ y\} =$
 $card\ \{i \in agents - agents1 - agents2. R\ i\ x\ y\} +$
 $(\text{if } x = y \text{ then } 2 \text{ else } 1) * card\ agents1$
 $\langle proof \rangle$

lemma *majority-extend-profile*:
assumes *new.is-pref-profile R*
shows *majority* (*extend-profile R*) = *majority R*
 $\langle proof \rangle$

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*
 $\langle proof \rangle$

lemma *kemeny-strategyproof-reduce*:
assumes *kemeny-strategyproof-swf agents alts swf*
shows *kemeny-strategyproof-swf* (*agents – agents1 – agents2*) *alts swf-low*
 $\langle proof \rangle$

lemma *majority-consistent-reduce*:
assumes *majority-consistent-swf agents alts swf*
shows *majority-consistent-swf* (*agents – agents1 – agents2*) *alts swf-low*
 $\langle proof \rangle$

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'
⟨proof⟩
```

lemma *topo-sorts-aux-Cons'*:

```
fixes x xs defines rel ≡ x # xs
shows topo-sorts-aux rel = topo-sorts-aux-step rel rel
⟨proof⟩
```

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}
⟨proof⟩ 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* <proof>

sublocale *anonymous-kemeny-strategyproof-swf agents alts swf* <proof>

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*
 <proof>

lemma *alts-conv-alts-list*: *alts* = *set alts-list*
 <proof>

lemma *card-alts* [simp]: *card alts* = *length alts-list*
 <proof>

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)* = $\{(x, y). x \neq y \wedge \text{of-ranking } xs \ x \ y\}$
 <proof>

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 $A \equiv \text{set } (\text{topo-sorts-aux } (\text{map } (\lambda x. (x, \{\}))) \text{ alts-list})$
shows $\text{aswf}' R \in A$
<proof>

lemma *aswf'-in-allowed-results:*
assumes *is-apref-profile' Rs*
shows $\text{aswf}' Rs \in \text{allowed-results } Rs$
<proof>

lemma *is-apref-profile'-iff:*
 $\text{is-apref-profile}' Rs \longleftrightarrow (\text{size } Rs = \text{agent-card} \wedge (\forall R \in \#Rs. \text{mset } R = \text{mset alts-list}))$
<proof>

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 $\text{inversion-number } S1' = d1 \text{ inversion-number } S2' = d2$
assumes $\text{map } (\text{index } T) S1 = S1' \text{ map } (\text{index } T) S2 = S2'$
assumes $R12: \text{add-mset } T' R1 \equiv \text{add-mset } T R2$
assumes $d2 < d1$
shows $\text{aswf}' R1 \neq S1 \vee \text{aswf}' R2 \neq S2$
<proof>

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 $\text{aswf}' R = S \vee \text{aswf}' R' \neq S$
<proof>

3.4 Automation for majority consistency

fun *majority-rel-mset-aux* :: *'a list multiset $\Rightarrow$ 'a list $\Rightarrow$ bool* **where**
 $\text{majority-rel-mset-aux } Rs [] \longleftrightarrow \text{True}$
 $\mid \text{majority-rel-mset-aux } Rs (x \# xs) \longleftrightarrow$
 $(\forall y \in \text{set } xs. 2 * \text{size } (\text{filter-mset } (\lambda R. \text{of-ranking } R y x) Rs) > \text{size } Rs) \wedge$
 $\text{majority-rel-mset-aux } Rs xs$

fun *majority-rel-list-aux* :: *'a list list $\Rightarrow$ 'a list $\Rightarrow$ bool* **where**
 $\text{majority-rel-list-aux } Rs [] \longleftrightarrow \text{True}$
 $\mid \text{majority-rel-list-aux } Rs (x \# xs) \longleftrightarrow$
 $\text{list-all } (\lambda y. 2 * \text{length } (\text{filter } (\lambda R. \text{of-ranking } R y x) Rs) > \text{length } Rs) xs \wedge$
 $\text{majority-rel-list-aux } Rs xs$

lemma *majority-rel-mset-aux-mset:*

majority-rel-mset-aux (mset Rs) ys $\longleftrightarrow$ majority-rel-list-aux Rs ys

<proof>

lemma *majority-rel-mset-aux-correct:*

assumes $\bigwedge R. R \in \# Rs \implies \text{distinct } R \wedge \text{set } R = A \text{ } Rs \neq \{\#\} \text{ distinct } zs \text{ set } zs \subseteq A$

defines $R s' \equiv \text{image-mset of-ranking } Rs$

defines $M \equiv \text{majority-mset } R s'$

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)

<proof>

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 *set S = A distinct S*

assumes *majority-rel-mset-aux Rs S*

shows *majority-rel-mset Rs S*

<proof>

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 mset ys = mset alts-list*

shows *majority-rel-mset (mset R) ys*

<proof>

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*

<proof>

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>

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*

<proof>

lemma *permutations-of-set-aux-list-Nil*: *permutations-of-set-aux-list acc [] = [acc]*
 <proof>

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))
 <proof>

<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

<ML>

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* $\langle \text{proof} \rangle$
sublocale *anonymous-kemeny-strategyproof-swf agents alts swf* $\langle \text{proof} \rangle$

lemma *symmetry-breaking-aux1*:
assumes *distinct*: *distinct* $[a, b, c, d, e]$ **and** *alts-eq*: $\text{alts} = \{a, b, c, d, e\}$
defines $R \equiv \{\# [a, b, c, d, e], [a, c, b, e, d] \# \}$
assumes R : $\text{aswf}' R = [a, c, b, d, e]$
shows $\text{aswf}' \{\# [a, b, c, e, d], [a, c, b, d, e] \# \} \in \{[a, b, c, e, d], [a, c, b, d, e]\}$
 $\langle \text{proof} \rangle$

lemma *symmetry-breaking-aux2*:
obtains *abcde* **where**
distinct abcde $\text{alts} = \text{set } abcde$ *length abcde* $= 5$
case abcde of $[a, b, c, d, e] \Rightarrow \text{aswf}' \{\# [a, b, c, d, e], [a, c, b, e, d] \# \} = [a, b, c, d, e]$
 $\langle \text{proof} \rangle$

lemma *contradiction*: *False*
 $\langle \text{proof} \rangle$

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-kemeny-sp-swf*) *impossibility*:
assumes *even* (*card agents*) **and** *card alts* ≥ 5
shows *False*
 $\langle \text{proof} \rangle$

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-kemeny-sp-swf-explicit-4-4* =
anonymous-unanimous-kemeny-sp-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*: $\text{aswf}' \{\# [a, b, c, d], [a, b, c, d], [b, a, d, c], [b, a, d, c] \# \} = [a, b, c, d]$
begin

$\langle ML \rangle$

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-kemenysp-swf-4-4* = *anonymous-unanimous-kemenysp-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* $\langle \text{proof} \rangle$

sublocale *anonymous-kemeny-strategyproof-swf agents alts swf* $\langle \text{proof} \rangle$

lemma *symmetry-breaking-aux1*:
assumes *distinct*: *distinct* $[a, b, c, d]$ **and** *alts-eq*: *alts* = $\{a, b, c, d\}$
defines $R \equiv \text{repeat-mset } 2 \ \{ \# [a, b, c, d], [b, a, d, c] \# \}$
assumes R : *aswf'* $R = [b, a, c, d]$
shows *aswf'* $(\text{repeat-mset } 2 \ \{ \# [a, b, d, c], [b, a, c, d] \# \}) \in \{[a, b, d, c], [b, a, c, d]\}$
 $\langle \text{proof} \rangle$

lemma *symmetry-breaking-aux2*:
obtains *abcd* **where**
distinct *abcd* *alts* = *set abcd* *length abcd* = 4
case abcd of $[a, b, c, d] \Rightarrow \text{aswf}' (\text{repeat-mset } 2 \ \{ \# [a, b, c, d], [b, a, d, c] \# \}) = [a, b, c, d]$
 $\langle \text{proof} \rangle$

lemma *contradiction*: *False*
 $\langle \text{proof} \rangle$

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*
 $\langle \text{proof} \rangle$

The following collects the two impossibility results in one theorem.

theorem *anonymous-unanimous-kemenysp-impossibility*:

```

assumes (card alts = 4  $\wedge$  4 dvd card agents)  $\vee$  (card alts  $\geq$  5  $\wedge$  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>

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 ( $\lambda x. x \in \text{set } xs$ ) 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>

```

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
<proof>

```

We consider the following profile R . This profile does not have a linear majority relation, but many manipulations of it do.

definition $R :: 'alt \text{ list list}$ **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*
<proof>

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>

We now consider a second profile, which differs from R only by a manipulation of the third agent.

definition *S :: 'alt list list 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-wf [simp]: prefs-from-rankings-wf S*
<proof>

We similarly show that $f(S) = dcba$.

lemma *swf'-S: swf' S = [d,c,b,a]*
<proof>

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 <proof>*

lemma *contradiction-eq9-ge4-aux:*
assumes *card agents = 9 card alts ≥ 4*
shows *False*
<proof>

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-eq18-ge4-aux*:
assumes *card agents = 18 card alts ≥ 4*
shows *False*
 ⟨*proof*⟩

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 $n ≥ 9$ except for $n ∈ \{10, 12, 14, 16\}$.

lemma *contradiction-geq9-ge4-aux*:
assumes *card agents ∈ {9, 11, 13, 15} ∪ {17..} card alts ≥ 4*
shows *False*
 ⟨*proof*⟩

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*⟩

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

⟨*ML*⟩

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

$\langle ML \rangle$

end

context *majcons-kstratproof-swf-explicit*
begin

lemma *contradiction-ge3-eq4*:
 assumes *card agents* ≥ 3 *card alts* = 4
 shows *False*
 $\langle proof \rangle$

end

We now have everything to put together the final impossibility theorem.

theorem *majcons-kstratproof-impossibility*:
 assumes (*card alts* = 4 $\wedge$ *card agents* ≥ 3) $\vee$
 (*card alts* ≥ 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*
 $\langle proof \rangle$

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. Froleyks, 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.