Verified Construction of Fair Voting Rules

Michael Kirsten

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
kirsten@kit.edu

March 11, 2024

Abstract

Voting rules aggregate multiple individual preferences in order to make a collective decision. Commonly, these mechanisms are expected to respect a multitude of different notions of fairness and reliability, which must be carefully balanced to avoid inconsistencies.

This article contains a formalisation of a framework for the construction of such fair voting rules using composable modules [1, 2]. The framework is a formal and systematic approach for the flexible and verified construction of voting rules from individual composable modules to respect such social-choice properties by construction. Formal composition rules guarantee resulting social-choice properties from properties of the individual components which are of generic nature to be reused for various voting rules. We provide proofs for a selected set of structures and composition rules. The approach can be readily extended in order to support more voting rules, e.g., from the literature by extending the sets of modules and composition rules.

Contents

Chapter 1

Social-Choice Types

1.1 Preference Relation

theory *Preference-Relation*
imports *Main*
begin

The very core of the composable modules voting framework: types and functions, derivations, lemmas, operations on preference relations, etc.

1.1.1 Definition

Each voter expresses pairwise relations between all alternatives, thereby inducing a linear order.

type-synonym $'a$ *Preference-Relation* $=$ $'a$ *rel*

type-synonym $'a$ *Vote* $=$ $'a$ *set* $\times$ $'a$ *Preference-Relation*

fun *is-less-preferred-than* :: $'a \Rightarrow 'a$ *Preference-Relation* $\Rightarrow 'a \Rightarrow$ *bool*
(- $\preceq$- - [*50*, *1000*, *51*] *50*) **where**
$a \preceq_r b = ((a, b) \in r)$

fun *alts-*$\mathcal{V}$ :: $'a$ *Vote* $\Rightarrow 'a$ *set* **where**
alts-$\mathcal{V}$ V = *fst* V

fun *pref-*$\mathcal{V}$ :: $'a$ *Vote* $\Rightarrow 'a$ *Preference-Relation* **where**
pref-$\mathcal{V}$ V = *snd* V

lemma *lin-imp-antisym*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *linear-order-on* A r
shows *antisym* r

using *assms*
unfolding *linear-order-on-def partial-order-on-def*
by *simp*

lemma *lin-imp-trans*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *linear-order-on A r*
shows *trans r*
using *assms order-on-defs*
by *blast*

1.1.2 Ranking

fun *rank* :: $'a$ *Preference-Relation* $\Rightarrow$ $'a$ $\Rightarrow$ *nat* **where**
rank r a = *card* (*above r a*)

lemma *rank-gt-zero*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
refl: $a \preceq_r a$ **and**
fin: *finite r*
shows *rank r a* $\geq$ *1*
proof (*unfold rank.simps above-def*)
have $a \in \{b \in$ *Field r*. $(a, b) \in r\}$
using *FieldI2 refl*
by *fastforce*
hence $\{b \in$ *Field r*. $(a, b) \in r\} \neq \{\}$
by *blast*
hence *card* $\{b \in$ *Field r*. $(a, b) \in r\} \neq$ *0*
by (*simp add*: *fin finite-Field*)
thus *1* $\leq$ *card* $\{b.\ (a, b) \in r\}$
using *Collect-cong FieldI2 less-one not-le-imp-less*
by (*metis* (*no-types*, *lifting*))
qed

1.1.3 Limited Preference

definition *limited* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$ *bool* **where**
limited A r $\equiv$ $r \subseteq A \times A$

lemma *limited-dest*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$

assumes
$a \preceq_r b$ **and**
limited A r
shows $a \in A \wedge b \in A$
using *assms*
unfolding *limited-def*
by *auto*

fun *limit* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$ $'a$ *Preference-Relation* **where**
limit A $r = \{(a, b) \in r.\ a \in A \wedge b \in A\}$

definition *connex* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$ *bool* **where**
connex A $r \equiv$ *limited* A $r \wedge (\forall\ a \in A.\ \forall\ b \in A.\ a \preceq_r b \vee b \preceq_r a)$

lemma *connex-imp-refl*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *connex* A r
shows *refl-on* A r
proof
from *assms*
show $r \subseteq A \times A$
unfolding *connex-def limited-def*
by *simp*
next
fix a :: $'a$
assume $a \in A$
with *assms*
have $a \preceq_r a$
unfolding *connex-def*
by *metis*
thus $(a, a) \in r$
by *simp*
qed

lemma *lin-ord-imp-connex*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *linear-order-on* A r
shows *connex* A r
proof (*unfold connex-def limited-def*, *safe*)
fix
a :: $'a$ **and**
b :: $'a$
assume $(a, b) \in r$
moreover have *refl-on* A r
using *assms partial-order-onD*

unfolding *linear-order-on-def*
by *safe*
ultimately show $a \in A$
by (*simp add*: *refl-on-domain*)
next
fix
a :: $'a$ **and**
b :: $'a$
assume $(a, b) \in r$
moreover have *refl-on A r*
using *assms partial-order-onD*
unfolding *linear-order-on-def*
by *safe*
ultimately show $b \in A$
by (*simp add*: *refl-on-domain*)
next
fix
a :: $'a$ **and**
b :: $'a$
assume
$a \in A$ **and**
$b \in A$ **and**
$\neg\ b \preceq_r a$
moreover from *this*
have $(b, a) \notin r$
by *simp*
moreover from *this*
have *refl-on A r*
using *assms partial-order-onD*
unfolding *linear-order-on-def*
by *blast*
ultimately have $(a, b) \in r$
using *assms refl-onD*
unfolding *linear-order-on-def total-on-def*
by *metis*
thus $a \preceq_r b$
by *simp*
qed

lemma *connex-antsym-and-trans-imp-lin-ord*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
connex-r: *connex A r* **and**
antisym-r: *antisym r* **and**
trans-r: *trans r*
shows *linear-order-on A r*
proof (*unfold connex-def linear-order-on-def partial-order-on-def*

preorder-on-def refl-on-def total-on-def, *safe*)
fix
a :: $'a$ **and**
b :: $'a$
assume $(a, b) \in r$
thus $a \in A$
using *connex-r refl-on-domain connex-imp-refl*
by *metis*
next
fix
a :: $'a$ **and**
b :: $'a$
assume $(a, b) \in r$
thus $b \in A$
using *connex-r refl-on-domain connex-imp-refl*
by *metis*
next
fix a :: $'a$
assume $a \in A$
thus $(a, a) \in r$
using *connex-r connex-imp-refl refl-onD*
by *metis*
next
from *trans-r*
show *trans r*
by *simp*
next
from *antisym-r*
show *antisym r*
by *simp*
next
fix
a :: $'a$ **and**
b :: $'a$
assume
$a \in A$ **and**
$b \in A$ **and**
$(b, a) \notin r$
moreover from *this*
have $a \preceq_r b \vee b \preceq_r a$
using *connex-r*
unfolding *connex-def*
by *metis*
hence $(a, b) \in r \vee (b, a) \in r$
by *simp*
ultimately show $(a, b) \in r$
by *metis*
qed

lemma *limit-to-limits*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
shows *limited* A (*limit* A r)
unfolding *limited-def*
by *fastforce*

lemma *limit-presv-connex*:
fixes
$B :: 'a\ set$ **and**
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes
connex: *connex* B r **and**
subset: $A \subseteq B$
shows *connex* A (*limit* A r)
proof (*unfold connex-def limited-def*, *simp*, *safe*)
let $?s = \{(a, b).\ (a, b) \in r \wedge a \in A \wedge b \in A\}$
fix
$a :: 'a$ **and**
$b :: 'a$
assume
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
not-b-pref-r-a: $(b, a) \notin r$
have $b \preceq_r a \vee a \preceq_r b$
using *a-in-A b-in-A connex connex-def in-mono subset*
by *metis*
hence $a \preceq_{?s} b \vee b \preceq_{?s} a$
using *a-in-A b-in-A*
by *auto*
hence $a \preceq_{?s} b$
using *not-b-pref-r-a*
by *simp*
thus $(a, b) \in r$
by *simp*
qed

lemma *limit-presv-antisym*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes *antisym* r
shows *antisym* (*limit* A r)
using *assms*
unfolding *antisym-def*
by *simp*

lemma *limit-presv-trans*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes *trans r*
shows *trans* (*limit A r*)
unfolding *trans-def*
using *transE assms*
by *auto*

lemma *limit-presv-lin-ord*:
fixes
$A :: 'a\ set$ **and**
$B :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes
linear-order-on B r **and**
$A \subseteq B$
shows *linear-order-on A* (*limit A r*)
using *assms connex-antsym-and-trans-imp-lin-ord limit-presv-antisym limit-presv-connex limit-presv-trans lin-ord-imp-connex*
unfolding *preorder-on-def partial-order-on-def linear-order-on-def*
by *metis*

lemma *limit-presv-prefs*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$ **and**
$b :: 'a$
assumes
$a \preceq_r b$ **and**
$a \in A$ **and**
$b \in A$
shows $let\ s = limit\ A\ r\ in\ a \preceq_s b$
using *assms*
by *simp*

lemma *limit-rel-presv-prefs*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$ **and**
$b :: 'a$
assumes $(a, b) \in limit\ A\ r$
shows $a \preceq_r b$
using *mem-Collect-eq assms*
by *simp*

lemma *limit-trans*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes $A \subseteq B$
shows *limit* A r = *limit* A (*limit* B r)
using *assms*
by *auto*

lemma *lin-ord-not-empty*:
fixes r :: $'a$ *Preference-Relation*
assumes $r \neq \{\}$
shows $\neg$ *linear-order-on* $\{\}$ r
using *assms connex-imp-refl lin-ord-imp-connex refl-on-domain subrelI*
by *fastforce*

lemma *lin-ord-singleton*:
fixes a :: $'a$
shows $\forall$ r. *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof (*clarify*)
fix r :: $'a$ *Preference-Relation*
assume *lin-ord-r-a*: *linear-order-on* $\{a\}$ r
hence $a \preceq_r a$
using *lin-ord-imp-connex singletonI*
unfolding *connex-def*
by *metis*
moreover from *lin-ord-r-a*
have $\forall$ $(b, c) \in r$. $b = a \wedge c = a$
using *connex-imp-refl lin-ord-imp-connex refl-on-domain split-beta*
by *fastforce*
ultimately show $r = \{(a, a)\}$
by *auto*
qed

1.1.4 Auxiliary Lemmas

lemma *above-trans*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes
trans r **and**
$(a, b) \in r$
shows *above* r $b \subseteq$ *above* r a
using *Collect-mono assms transE*
unfolding *above-def*
by *metis*

lemma *above-refl*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
refl-on A r **and**
$a \in A$
shows $a \in$ *above* r a
using *assms refl-onD*
unfolding *above-def*
by *simp*

lemma *above-subset-geq-one*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
linear-order-on A r **and**
linear-order-on A r' **and**
above r $a \subseteq$ *above* r' a **and**
above r' $a = \{a\}$
shows *above* r $a = \{a\}$
using *assms connex-imp-refl above-refl insert-absorb lin-ord-imp-connex mem-Collect-eq refl-on-domain singletonI subset-singletonD*
unfolding *above-def*
by *metis*

lemma *above-connex*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
connex A r **and**
$a \in A$
shows $a \in$ *above* r a
using *assms connex-imp-refl above-refl*
by *metis*

lemma *pref-imp-in-above*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
shows $(a \preceq_r b) = (b \in$ *above* r $a)$

unfolding *above-def*
by *simp*

lemma *limit-presv-above*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes
$b \in$ *above* r a **and**
$a \in A$ **and**
$b \in A$
shows $b \in$ *above* (*limit* A r) a
using *assms pref-imp-in-above limit-presv-prefs*
by *metis*

lemma *limit-rel-presv-above*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes $b \in$ *above* (*limit* B r) a
shows $b \in$ *above* r a
using *assms limit-rel-presv-prefs mem-Collect-eq pref-imp-in-above*
unfolding *above-def*
by *metis*

lemma *above-one*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
lin-ord-r: *linear-order-on* A r **and**
fin-A: *finite* A **and**
non-empty-A: $A \neq \{\}$
shows $\exists\ a \in A.$ *above* r $a = \{a\} \land (\forall\ a' \in A.$ *above* r $a' = \{a'\} \longrightarrow a' = a)$
proof −
obtain n :: *nat* **where**
len-n-plus-one: $n + 1 =$ *card* A
using *Suc-eq-plus1 antisym-conv2 fin-A non-empty-A card-eq-0-iff*
gr0-implies-Suc le0
by *metis*
have *linear-order-on* A $r \land$ *finite* $A \land A \neq \{\} \land n + 1 =$ *card* $A \longrightarrow$
$(\exists\ a.\ a \in A \land$ *above* r $a = \{a\})$
proof (*induction* n *arbitrary*: A r)
case *0*

show *?case*
proof (*clarify*)
fix
A' :: $'a$ *set* **and**
r' :: $'a$ *Preference-Relation*
assume
lin-ord-r: *linear-order-on* A' r' **and**
len-A-is-one: $0 + 1 = card\ A'$
then obtain a **where** $A' = \{a\}$
using *card-1-singletonE add.left-neutral*
by *metis*
hence $a \in A' \wedge above\ r'\ a = \{a\}$
using *above-def lin-ord-r connex-imp-refl above-refl lin-ord-imp-connex refl-on-domain*
by *fastforce*
thus $\exists\ a'.\ a' \in A' \wedge above\ r'\ a' = \{a'\}$
by *metis*
qed
next
case (*Suc n*)
show *?case*
proof (*clarify*)
fix
A' :: $'a$ *set* **and**
r' :: $'a$ *Preference-Relation*
assume
lin-ord-r: *linear-order-on* A' r' **and**
fin-A: *finite* A' **and**
A-not-empty: $A' \neq \{\}$ **and**
len-A-n-plus-one: $Suc\ n + 1 = card\ A'$
then obtain B **where**
subset-B-card: $card\ B = n + 1 \wedge B \subseteq A'$
using *Suc-inject add-Suc card.insert-remove finite.cases insert-Diff-single subset-insertI*
by (*metis* (*mono-tags*, *lifting*))
then obtain a **where**
a: $A' - B = \{a\}$
using *Suc-eq-plus1 add-diff-cancel-left' fin-A len-A-n-plus-one card-1-singletonE card-Diff-subset finite-subset*
by *metis*
have $\exists\ a' \in B.\ above\ (limit\ B\ r')\ a' = \{a'\}$
using *subset-B-card Suc.IH add-diff-cancel-left' lin-ord-r card-eq-0-iff diff-le-self leD lessI limit-presv-lin-ord*
unfolding *One-nat-def*
by *metis*
then obtain b **where**
alt-b: $above\ (limit\ B\ r')\ b = \{b\}$
by *blast*
hence *b-above*: $\{a'.\ (b,\ a') \in limit\ B\ r'\} = \{b\}$

unfolding *above-def*
by *metis*
hence *b-pref-b*: $b \preceq_r{}' \ b$
using *CollectD limit-rel-presv-prefs singletonI*
by (*metis* (*lifting*))
show $\exists \ a'. \ a' \in A' \wedge \mathit{above}\ r' \ a' = \{a'\}$
proof (*cases*)
assume *a-pref-r-b*: $a \preceq_r{}' \ b$
have *refl-A*:
$\forall \ A'' \ r'' \ a' \ a''. \ \mathit{refl\text{-}on}\ A'' \ r'' \wedge (a'::'a, \ a'') \in r'' \longrightarrow a' \in A'' \wedge a'' \in A''$
using *refl-on-domain*
by *metis*
have *connex-refl*: $\forall \ A'' \ r''. \ \mathit{connex}\ (A''::'a\ set) \ r'' \longrightarrow \mathit{refl\text{-}on}\ A'' \ r''$
using *connex-imp-refl*
by *metis*
have $\forall \ A'' \ r''. \ \mathit{linear\text{-}order\text{-}on}\ (A''::'a\ set) \ r'' \longrightarrow \mathit{connex}\ A'' \ r''$
by (*simp add*: *lin-ord-imp-connex*)
hence *refl-A'*: *refl-on* $A' \ r'$
using *connex-refl lin-ord-r*
by *metis*
hence $a \in A' \wedge b \in A'$
using *refl-A a-pref-r-b*
by *simp*
hence *b-in-r*: $\forall \ a'. \ a' \in A' \longrightarrow b = a' \vee (b, \ a') \in r' \vee (a', \ b) \in r'$
using *lin-ord-r*
unfolding *linear-order-on-def total-on-def*
by *metis*
have *b-in-lim-B-r*: $(b, \ b) \in \mathit{limit}\ B \ r'$
using *alt-b mem-Collect-eq singletonI*
unfolding *above-def*
by *metis*
have *b-wins*: $\{a'. \ (b, \ a') \in \mathit{limit}\ B \ r'\} = \{b\}$
using *alt-b*
unfolding *above-def*
by (*metis* (*no-types*))
have *b-refl*: $(b, \ b) \in \{(a', \ a''). \ (a', \ a'') \in r' \wedge a' \in B \wedge a'' \in B\}$
using *b-in-lim-B-r*
by *simp*
moreover have *b-wins-B*: $\forall \ b' \in B. \ b \in \mathit{above}\ r' \ b'$
using *subset-B-card b-in-r b-wins b-refl CollectI Product-Type.Collect-case-prodD*
unfolding *above-def*
by *fastforce*
moreover have $b \in \mathit{above}\ r' \ a$
using *a-pref-r-b pref-imp-in-above*
by *metis*
ultimately have *b-wins*: $\forall \ a' \in A'. \ b \in \mathit{above}\ r' \ a'$
using *Diff-iff a empty-iff insert-iff*
by (*metis* (*no-types*))
hence $\forall \ a' \in A'. \ a' \in \mathit{above}\ r' \ b \longrightarrow a' = b$

using *CollectD lin-ord-r lin-imp-antisym*
unfolding *above-def antisym-def*
by *metis*
hence $\forall\ a' \in A'.\ (a' \in above\ r'\ b) = (a' = b)$
using *b-wins*
by *blast*
moreover have *above-b-in-A*: $above\ r'\ b \subseteq A'$
unfolding *above-def*
using *refl-A' refl-A*
by *auto*
ultimately have $above\ r'\ b = \{b\}$
using *alt-b*
unfolding *above-def*
by *fastforce*
thus *?thesis*
using *above-b-in-A*
by *blast*
next
assume $\neg\ a \preceq_r{}'\ b$
hence $b \preceq_r{}'\ a$
using *subset-B-card DiffE a lin-ord-r alt-b limit-to-limits limited-dest singletonI subset-iff lin-ord-imp-connex pref-imp-in-above*
unfolding *connex-def*
by *metis*
hence *b-smaller-a*: $(b,\ a) \in r'$
by *simp*
have *lin-ord-subset-A*:
$\forall\ B'\ B''\ r''.$
$linear\text{-}order\text{-}on\ (B''::'a\ set)\ r'' \wedge B' \subseteq B'' \longrightarrow$
$linear\text{-}order\text{-}on\ B'\ (limit\ B'\ r'')$
using *limit-presv-lin-ord*
by *metis*
have $\{a'.\ (b,\ a') \in limit\ B\ r'\} = \{b\}$
using *alt-b*
unfolding *above-def*
by *metis*
hence *b-in-B*: $b \in B$
by *auto*
have *limit-B*: $partial\text{-}order\text{-}on\ B\ (limit\ B\ r') \wedge total\text{-}on\ B\ (limit\ B\ r')$
using *lin-ord-subset-A subset-B-card lin-ord-r*
unfolding *linear-order-on-def*
by *metis*
have
$\forall\ A''\ r''.$
$total\text{-}on\ A''\ r'' =$
$(\forall\ a'.\ (a'::'a) \notin A'' \vee$
$(\forall\ a''.\ a'' \notin A'' \vee a' = a'' \vee (a',\ a'') \in r'' \vee (a'',\ a') \in r''))$
unfolding *total-on-def*
by *metis*

hence $\forall\ a'\ a''.\ a' \in B \longrightarrow a'' \in B \longrightarrow$
$a' = a'' \lor (a', a'') \in$ *limit* $B\ r' \lor (a'', a') \in$ *limit* $B\ r'$
using *limit-B*
by *simp*
hence $\forall\ a' \in B.\ b \in$ *above* $r'\ a'$
using *limit-rel-presv-prefs pref-imp-in-above singletonD mem-Collect-eq lin-ord-r alt-b b-above b-pref-b subset-B-card b-in-B*
by (*metis* (*lifting*))
hence $\forall\ a' \in B.\ a' \preceq_{r'} b$
unfolding *above-def*
by *simp*
hence *b-wins*: $\forall\ a' \in B.\ (a', b) \in r'$
by *simp*
have *trans* r'
using *lin-ord-r lin-imp-trans*
by *metis*
hence $\forall\ a' \in B.\ (a', a) \in r'$
using *transE b-smaller-a b-wins*
by *metis*
hence $\forall\ a' \in B.\ a' \preceq_{r'} a$
by *simp*
hence *nothing-above-a*: $\forall\ a' \in A'.\ a' \preceq_{r'} a$
using *a lin-ord-r lin-ord-imp-connex above-connex Diff-iff empty-iff insert-iff pref-imp-in-above*
by *metis*
have $\forall\ a' \in A'.\ (a' \in$ *above* $r'\ a) = (a' = a)$
using *lin-ord-r lin-imp-antisym nothing-above-a pref-imp-in-above CollectD*
unfolding *antisym-def above-def*
by *metis*
moreover have *above-a-in-A*: *above* $r'\ a \subseteq A'$
using *lin-ord-r connex-imp-refl lin-ord-imp-connex mem-Collect-eq refl-on-domain*
unfolding *above-def*
by *fastforce*
ultimately have *above* $r'\ a = \{a\}$
using *a*
unfolding *above-def*
by *blast*
thus *?thesis*
using *above-a-in-A*
by *blast*
qed
qed
qed
hence $\exists\ a.\ a \in A \land$ *above* $r\ a = \{a\}$
using *fin-A non-empty-A lin-ord-r len-n-plus-one*
by *blast*
thus *?thesis*
using *assms lin-ord-imp-connex pref-imp-in-above singletonD*
unfolding *connex-def*

by *metis*
qed

lemma *above-one-eq*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes
lin-ord: *linear-order-on* A r **and**
fin-A: *finite* A **and**
not-empty-A: $A \neq \{\}$ **and**
above-a: *above* r $a = \{a\}$ **and**
above-b: *above* r $b = \{b\}$
shows $a = b$
proof −
have $a \preceq_r a$
using *above-a singletonI pref-imp-in-above*
by *metis*
also have $b \preceq_r b$
using *above-b singletonI pref-imp-in-above*
by *metis*
moreover have
$\exists\ a' \in A.$ *above* r $a' = \{a'\} \wedge (\forall\ a'' \in A.$ *above* r $a'' = \{a''\} \longrightarrow a'' = a')$
using *lin-ord fin-A not-empty-A*
by (*simp add*: *above-one*)
moreover have *connex* A r
using *lin-ord*
by (*simp add*: *lin-ord-imp-connex*)
ultimately show $a = b$
using *above-a above-b limited-dest*
unfolding *connex-def*
by *metis*
qed

lemma *above-one-imp-rank-one*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *above* r $a = \{a\}$
shows *rank* r $a = 1$
using *assms*
by *simp*

lemma *rank-one-imp-above-one*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**

a :: *'a*
assumes
lin-ord: *linear-order-on A r* **and**
rank-one: *rank r a = 1*
shows *above r a* = {*a*}
proof −
from *lin-ord*
have *refl-on A r*
using *linear-order-on-def partial-order-onD*
by *blast*
moreover from *assms*
have *a* ∈ *A*
unfolding *rank.simps above-def linear-order-on-def partial-order-on-def preorder-on-def total-on-def*
using *card-1-singletonE insertI1 mem-Collect-eq refl-onD1*
by *metis*
ultimately have *a* ∈ *above r a*
using *above-refl*
by *fastforce*
with *rank-one*
show *above r a* = {*a*}
using *card-1-singletonE rank.simps singletonD*
by *metis*
qed

theorem *above-rank*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
a :: *'a*
assumes *linear-order-on A r*
shows (*above r a* = {*a*}) = (*rank r a = 1*)
using *assms above-one-imp-rank-one rank-one-imp-above-one*
by *metis*

lemma *rank-unique*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
a :: *'a* **and**
b :: *'a*
assumes
lin-ord: *linear-order-on A r* **and**
fin-A: *finite A* **and**
a-in-A: *a* ∈ *A* **and**
b-in-A: *b* ∈ *A* **and**
a-neq-b: *a* ≠ *b*
shows *rank r a* ≠ *rank r b*
proof (*unfold rank.simps above-def*, *clarify*)

assume *card-eq*: *card* $\{a'.\ (a,\ a') \in r\}$ = *card* $\{a'.\ (b,\ a') \in r\}$
have *refl-r*: *refl-on A r*
using *lin-ord*
by (*simp add*: *lin-ord-imp-connex connex-imp-refl*)
hence *rel-refl-b*: $(b,\ b) \in r$
using *b-in-A*
unfolding *refl-on-def*
by (*metis* (*no-types*))
have *rel-refl-a*: $(a,\ a) \in r$
using *a-in-A refl-r refl-onD*
by (*metis* (*full-types*))
obtain p :: $'a \Rightarrow$ *bool* **where**
rel-b: $\forall\ y.\ p\ y = ((b,\ y) \in r)$
using *is-less-preferred-than.simps*
by *metis*
hence *finite* (*Collect p*)
using *refl-r refl-on-domain fin-A rev-finite-subset mem-Collect-eq subsetI*
by *metis*
hence *finite* $\{a'.\ (b,\ a') \in r\}$
using *rel-b*
by (*simp add*: *Collect-mono rev-finite-subset*)
moreover from *this*
have *finite* $\{a'.\ (a,\ a') \in r\}$
using *card-eq card-gt-0-iff rel-refl-b*
by *force*
moreover have *trans r*
using *lin-ord lin-imp-trans*
by *metis*
moreover have $(a,\ b) \in r \lor (b,\ a) \in r$
using *lin-ord a-in-A b-in-A a-neq-b*
unfolding *linear-order-on-def total-on-def*
by *metis*
ultimately have *sets-eq*: $\{a'.\ (a,\ a') \in r\} = \{a'.\ (b,\ a') \in r\}$
using *card-eq above-trans card-seteq order-refl*
unfolding *above-def*
by *metis*
hence $(b,\ a) \in r$
using *rel-refl-a sets-eq*
by *blast*
hence $(a,\ b) \notin r$
using *lin-ord lin-imp-antisym a-neq-b antisymD*
by *metis*
thus *False*
using *lin-ord partial-order-onD sets-eq b-in-A*
unfolding *linear-order-on-def refl-on-def*
by *blast*
qed

lemma *above-presv-limit*:

fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$
shows *above* (*limit* A r) $a \subseteq A$
unfolding *above-def*
by *auto*

1.1.5 Lifting Property

definition *equiv-rel-except-a* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$
$'a$ *Preference-Relation* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
equiv-rel-except-a A r r' $a \equiv$
linear-order-on A r $\wedge$ *linear-order-on* A r' $\wedge$ $a \in A$ $\wedge$
$(\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ (a' \preceq_r b') = (a' \preceq_{r'} b'))$

definition *lifted* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$
$'a$ *Preference-Relation* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
lifted A r r' $a \equiv$
equiv-rel-except-a A r r' a $\wedge$ $(\exists\ a' \in A - \{a\}.\ a \preceq_r a' \wedge a' \preceq_{r'} a)$

lemma *trivial-equiv-rel*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *linear-order-on* A r
shows $\forall\ a \in A.$ *equiv-rel-except-a* A r r a
unfolding *equiv-rel-except-a-def*
using *assms*
by *simp*

lemma *lifted-imp-equiv-rel-except-a*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* A r r' a
shows *equiv-rel-except-a* A r r' a
using *assms*
unfolding *lifted-def equiv-rel-except-a-def*
by *simp*

lemma *lifted-imp-switched*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$

assumes *lifted* $A\ r\ r'\ a$
shows $\forall\ a' \in A - \{a\}.\ \neg\ (a' \preceq_r a \wedge a \preceq_{r'} a')$
proof (*safe*)
fix $b :: 'a$
assume
b-in-A: $b \in A$ **and**
b-neq-a: $b \neq a$ **and**
b-pref-a: $b \preceq_r a$ **and**
a-pref-b: $a \preceq_{r'} b$
hence *b-pref-a-rel*: $(b,\ a) \in r$
by *simp*
have *a-pref-b-rel*: $(a,\ b) \in r'$
using *a-pref-b*
by *simp*
have *antisym* r
using *assms lifted-imp-equiv-rel-except-a lin-imp-antisym*
unfolding *equiv-rel-except-a-def*
by *metis*
hence $\forall\ a'\ b'.\ (a',\ b') \in r \longrightarrow (b',\ a') \in r \longrightarrow a' = b'$
unfolding *antisym-def*
by *metis*
hence *imp-b-eq-a*: $(b,\ a) \in r \Longrightarrow (a,\ b) \in r \Longrightarrow b = a$
by *simp*
have $\exists\ a' \in A - \{a\}.\ a \preceq_r a' \wedge a' \preceq_{r'} a$
using *assms*
unfolding *lifted-def*
by *metis*
then obtain $c :: 'a$ **where**
$c \in A - \{a\} \wedge a \preceq_r c \wedge c \preceq_{r'} a$
by *metis*
hence *c-eq-r-s-exc-a*: $c \in A - \{a\} \wedge (a,\ c) \in r \wedge (c,\ a) \in r'$
by *simp*
have *equiv-r-s-exc-a*: *equiv-rel-except-a* $A\ r\ r'\ a$
using *assms*
unfolding *lifted-def*
by *metis*
hence $\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ (a' \preceq_r b') = (a' \preceq_{r'} b')$
unfolding *equiv-rel-except-a-def*
by *metis*
hence *equiv-r-s-exc-a-rel*:
$\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((a',\ b') \in r) = ((a',\ b') \in r')$
by *simp*
have $\forall\ a'\ b'\ c'.\ (a',\ b') \in r \longrightarrow (b',\ c') \in r \longrightarrow (a',\ c') \in r$
using *equiv-r-s-exc-a*
unfolding *equiv-rel-except-a-def linear-order-on-def partial-order-on-def preorder-on-def trans-def*
by *metis*
hence $(b,\ c) \in r'$
using *b-in-A b-neq-a b-pref-a-rel c-eq-r-s-exc-a equiv-r-s-exc-a equiv-r-s-exc-a-rel*

insertE insert-Diff
unfolding *equiv-rel-except-a-def*
by *metis*
hence $(a, c) \in r'$
using *a-pref-b-rel b-pref-a-rel imp-b-eq-a b-neq-a equiv-r-s-exc-a lin-imp-trans transE*
unfolding *equiv-rel-except-a-def*
by *metis*
thus *False*
using *c-eq-r-s-exc-a equiv-r-s-exc-a antisymD DiffD2 lin-imp-antisym singletonI*
unfolding *equiv-rel-except-a-def*
by *metis*
qed

lemma *lifted-mono*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
r' :: *'a Preference-Relation* **and**
a :: *'a* **and**
a' :: *'a*
assumes
lifted: *lifted* $A\ r\ r'\ a$ **and**
a'-pref-a: $a' \preceq_r a$
shows $a' \preceq_{r'} a$
proof (*simp*)
have *a'-pref-a-rel*: $(a', a) \in r$
using *a'-pref-a*
by *simp*
hence *a'-in-A*: $a' \in A$
using *lifted connex-imp-refl lin-ord-imp-connex refl-on-domain*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
have $\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ (b \preceq_r b') = (b \preceq_{r'} b')$
using *lifted*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((b, b') \in r) = ((b, b') \in r')$
by *simp*
have $\exists\ b \in A - \{a\}.\ a \preceq_r b \wedge b \preceq_{r'} a$
using *lifted*
unfolding *lifted-def*
by *metis*
hence *ex-lifted*: $\exists\ b \in A - \{a\}.\ (a, b) \in r \wedge (b, a) \in r'$
by *simp*
show $(a', a) \in r'$
proof (*cases* $a' = a$)
case *True*

thus *?thesis*
using *connex-imp-refl refl-onD lifted lin-ord-imp-connex*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
next
case *False*
thus *?thesis*
using *a′-pref-a-rel a′-in-A rest-eq ex-lifted insertE insert-Diff lifted lin-imp-trans lifted-imp-equiv-rel-except-a*
unfolding *equiv-rel-except-a-def trans-def*
by *metis*
qed
qed

lemma *lifted-above-subset*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* $A\ r\ r'\ a$
shows *above* $r'\ a \subseteq$ *above* $r\ a$
proof (*unfold above-def*, *safe*)
fix a' :: $'a$
assume *a-pref-x*: $(a,\ a') \in r'$
from *assms*
have $\exists\ b \in A - \{a\}.\ a \preceq_r b \land b \preceq_{r'} a$
unfolding *lifted-def*
by *metis*
hence *lifted-r*: $\exists\ b \in A - \{a\}.\ (a,\ b) \in r \land (b,\ a) \in r'$
by *simp*
from *assms*
have $\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ (b \preceq_r b') = (b \preceq_{r'} b')$
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*: $\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((b,\ b') \in r) = ((b,\ b') \in r')$
by *simp*
from *assms*
have *trans-r*: $\forall\ b\ c\ d.\ (b,\ c) \in r \longrightarrow (c,\ d) \in r \longrightarrow (b,\ d) \in r$
using *lin-imp-trans*
unfolding *trans-def lifted-def equiv-rel-except-a-def*
by *metis*
from *assms*
have *trans-s*: $\forall\ b\ c\ d.\ (b,\ c) \in r' \longrightarrow (c,\ d) \in r' \longrightarrow (b,\ d) \in r'$
using *lin-imp-trans*
unfolding *trans-def lifted-def equiv-rel-except-a-def*
by *metis*
from *assms*
have *refl-r*: $(a,\ a) \in r$

    using *connex-imp-refl lin-ord-imp-connex refl-onD*
    unfolding *equiv-rel-except-a-def lifted-def*
    by *metis*
  from *a-pref-x assms*
  have $a' \in A$
    using *connex-imp-refl lin-ord-imp-connex refl-onD2*
    unfolding *equiv-rel-except-a-def lifted-def*
    by *metis*
  with *a-pref-x lifted-r rest-eq trans-r trans-s refl-r*
  show $(a, a') \in r$
    using *Diff-iff singletonD*
    by (*metis* (*full-types*))
qed

lemma *lifted-above-mono*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation* **and**
    r' :: $'a$ *Preference-Relation* **and**
    a :: $'a$ **and**
    a' :: $'a$
  assumes
    lifted-a: *lifted* $A\ r\ r'\ a$ **and**
    a'*-in-A-sub-a*: $a' \in A - \{a\}$
  shows $above\ r\ a' \subseteq above\ r'\ a' \cup \{a\}$
proof (*safe*, *simp*)
  fix b :: $'a$
  assume
    b-in-above-r: $b \in above\ r\ a'$ **and**
    b-not-in-above-s: $b \notin above\ r'\ a'$
  have $\forall\ b' \in A - \{a\}.\ (a' \preceq_r b') = (a' \preceq_{r'} b')$
    using a'*-in-A-sub-a lifted-a*
    unfolding *lifted-def equiv-rel-except-a-def*
    by *metis*
  hence $\forall\ b' \in A - \{a\}.\ (b' \in above\ r\ a') = (b' \in above\ r'\ a')$
    unfolding *above-def*
    by *simp*
  hence $(b \in above\ r\ a') = (b \in above\ r'\ a')$
    using *lifted-a b-not-in-above-s lifted-mono limited-dest lifted-def lin-ord-imp-connex member-remove pref-imp-in-above*
    unfolding *equiv-rel-except-a-def remove-def connex-def*
    by *metis*
  thus $b = a$
    using *b-in-above-r b-not-in-above-s*
    by *simp*
qed

lemma *limit-lifted-imp-eq-or-lifted*:
  fixes

A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
lifted: *lifted* A' r r' a **and**
subset: $A \subseteq A'$
shows *limit* A r = *limit* A r' $\vee$ *lifted* A (*limit* A r) (*limit* A r') a
proof −
have $\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ (a' \preceq_r b') = (a' \preceq_r{}' b')$
using *lifted subset*
unfolding *lifted-def equiv-rel-except-a-def*
by *auto*
hence *eql-rs*:
$\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.$
$((a',\ b') \in (limit\ A\ r)) = ((a',\ b') \in (limit\ A\ r'))$
using *DiffD1 limit-presv-prefs limit-rel-presv-prefs*
by *simp*
have *lin-ord-r-s*: *linear-order-on* A (*limit* A r) $\wedge$ *linear-order-on* A (*limit* A r')
using *lifted subset lifted-def equiv-rel-except-a-def limit-presv-lin-ord*
by *metis*
show *?thesis*
proof (*cases*)
assume *a-in-A*: $a \in A$
thus *?thesis*
proof (*cases*)
assume $\exists\ a' \in A - \{a\}.\ a \preceq_r a' \wedge a' \preceq_r{}' a$
hence $\exists\ a' \in A - \{a\}.$
$(let\ q = limit\ A\ r\ in\ a \preceq_q a') \wedge (let\ u = limit\ A\ r'\ in\ a' \preceq_u a)$
using *DiffD1 limit-presv-prefs a-in-A*
by *simp*
thus *?thesis*
using *a-in-A eql-rs lin-ord-r-s*
unfolding *lifted-def equiv-rel-except-a-def*
by *simp*
next
assume $\neg$ $(\exists\ a' \in A - \{a\}.\ a \preceq_r a' \wedge a' \preceq_r{}' a)$
hence *strict-pref-to-a*: $\forall\ a' \in A - \{a\}.\ \neg\ (a \preceq_r a' \wedge a' \preceq_r{}' a)$
by *simp*
moreover have *not-worse*: $\forall\ a' \in A - \{a\}.\ \neg\ (a' \preceq_r a \wedge a \preceq_r{}' a')$
using *lifted subset lifted-imp-switched*
by *fastforce*
moreover have *connex*: *connex* A (*limit* A r) $\wedge$ *connex* A (*limit* A r')
using *lifted subset limit-presv-lin-ord lin-ord-imp-connex*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
moreover have
$\forall\ A''\ r''.$ *connex* A'' r'' =

$(limited\ A''\ r'' \wedge$
$(\forall\ b\ b'.\ (b::'a) \in A'' \longrightarrow b' \in A'' \longrightarrow (b \preceq_r''\ b' \vee b' \preceq_r''\ b)))$
unfolding *connex-def*
by (*simp add*: *Ball-def-raw*)
hence *limit-rel-r*:
$limited\ A\ (limit\ A\ r) \wedge$
$(\forall\ b\ b'.\ b \in A \wedge b' \in A \longrightarrow (b,\ b') \in limit\ A\ r \vee (b',\ b) \in limit\ A\ r)$
using *connex*
by *simp*
have *limit-imp-rel*: $\forall\ b\ b'\ A''\ r''.\ (b::'a,\ b') \in limit\ A''\ r'' \longrightarrow b \preceq_r''\ b'$
using *limit-rel-presv-prefs*
by *metis*
have *limit-rel-s*:
$limited\ A\ (limit\ A\ r') \wedge$
$(\forall\ b\ b'.\ b \in A \wedge b' \in A \longrightarrow (b,\ b') \in limit\ A\ r' \vee (b',\ b) \in limit\ A\ r')$
using *connex*
unfolding *connex-def*
by *simp*
ultimately have
$\forall\ a' \in A - \{a\}.\ a \preceq_r\ a' \wedge a \preceq_r'\ a' \vee a' \preceq_r\ a \wedge a' \preceq_r'\ a$
using *DiffD1 limit-rel-r limit-rel-presv-prefs a-in-A*
by *metis*
have $\forall\ a' \in A - \{a\}.\ ((a,\ a') \in (limit\ A\ r)) = ((a,\ a') \in (limit\ A\ r'))$
using *DiffD1 limit-imp-rel limit-rel-r limit-rel-s a-in-A*
strict-pref-to-a not-worse
by *metis*
hence
$\forall\ a' \in A - \{a\}.$
$(let\ q = limit\ A\ r\ in\ a \preceq_q\ a') = (let\ q = limit\ A\ r'\ in\ a \preceq_q\ a')$
by *simp*
moreover have
$\forall\ a' \in A - \{a\}.\ ((a',\ a) \in (limit\ A\ r)) = ((a',\ a) \in (limit\ A\ r'))$
using *a-in-A strict-pref-to-a not-worse DiffD1 limit-rel-presv-prefs*
limit-rel-s limit-rel-r
by *metis*
moreover have $(a,\ a) \in (limit\ A\ r) \wedge (a,\ a) \in (limit\ A\ r')$
using *a-in-A connex connex-imp-refl refl-onD*
by *metis*
ultimately show *?thesis*
using *eql-rs*
by *auto*
qed
next
assume $a \notin A$
thus *?thesis*
using *limit-to-limits limited-dest subrelI subset-antisym eql-rs*
by *auto*
qed
qed

lemma *negl-diff-imp-eq-limit*:
  fixes
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation* **and**
    r' :: $'a$ *Preference-Relation* **and**
    a :: $'a$
  assumes
    change: *equiv-rel-except-a* A' r r' a **and**
    subset: $A \subseteq A'$ **and**
    not-in-A: $a \notin A$
  shows *limit* A r = *limit* A r'
proof −
  have $A \subseteq A' - \{a\}$
    unfolding *subset-Diff-insert*
    using *not-in-A subset*
    by *simp*
  hence $\forall\ b \in A.\ \forall\ b' \in A.\ (b \preceq_r b') = (b \preceq_{r'} b')$
    using *change in-mono*
    unfolding *equiv-rel-except-a-def*
    by *metis*
  thus *?thesis*
    by *auto*
qed

theorem *lifted-above-winner-alts*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation* **and**
    r' :: $'a$ *Preference-Relation* **and**
    a :: $'a$ **and**
    a' :: $'a$
  assumes
    lifted-a: *lifted* A r r' a **and**
    a'-above-a': *above* r $a' = \{a'\}$ **and**
    fin-A: *finite* A
  shows *above* r' $a' = \{a'\} \vee$ *above* r' $a = \{a\}$
proof (*cases*)
  assume $a = a'$
  thus *?thesis*
    using *above-subset-geq-one lifted-a a'-above-a' lifted-above-subset*
    unfolding *lifted-def equiv-rel-except-a-def*
    by *metis*
next
  assume *a-neq-a'*: $a \neq a'$
  thus *?thesis*
  proof (*cases*)
    assume *above* r' $a' = \{a'\}$

thus *?thesis*
  by *simp*
next
  assume *a′-not-above-a′*: *above* r' $a' \neq \{a'\}$
  have $\forall\ a'' \in A.\ a'' \preceq_r a'$
  proof (*safe*)
    fix b :: $'a$
    assume *y-in-A*: $b \in A$
    hence $A \neq \{\}$
      by *blast*
    moreover have *linear-order-on* A r
      using *lifted-a*
      unfolding *equiv-rel-except-a-def lifted-def*
      by *simp*
    ultimately show $b \preceq_r a'$
      using *y-in-A a′-above-a′ lin-ord-imp-connex pref-imp-in-above*
        singletonD limited-dest singletonI
      unfolding *connex-def*
      by (*metis* (*no-types*))
  qed
  moreover have *equiv-rel-except-a* A r r' a
    using *lifted-a*
    unfolding *lifted-def*
    by *metis*
  moreover have $a' \in A - \{a\}$
    using *a-neq-a′ calculation member-remove*
      limited-dest lin-ord-imp-connex
    using *equiv-rel-except-a-def remove-def connex-def*
    by *metis*
  ultimately have $\forall\ a'' \in A - \{a\}.\ a'' \preceq_{r'} a'$
    using *DiffD1 lifted-a*
    unfolding *equiv-rel-except-a-def*
    by *metis*
  hence $\forall\ a'' \in A - \{a\}.$ *above* r' $a'' \neq \{a''\}$
    using *a′-not-above-a′ empty-iff insert-iff pref-imp-in-above*
    by *metis*
  hence *above* r' $a = \{a\}$
    using *Diff-iff all-not-in-conv lifted-a above-one singleton-iff fin-A*
    unfolding *lifted-def equiv-rel-except-a-def*
    by *metis*
  thus *above* r' $a' = \{a'\} \lor$ *above* r' $a = \{a\}$
    by *simp*
qed
qed

theorem *lifted-above-winner-single*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation* **and**

r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
lifted A r r' a **and**
above r a = $\{a\}$ **and**
finite A
shows *above* r' a = $\{a\}$
using *assms lifted-above-winner-alts*
by *metis*

theorem *lifted-above-winner-other*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
a' :: $'a$
assumes
lifted-a: *lifted* A r r' a **and**
a'*-above-*a': *above* r' a' = $\{a'\}$ **and**
fin-A: *finite* A **and**
a-not-a': $a \neq a'$
shows *above* r a' = $\{a'\}$
proof (*rule ccontr*)
assume *not-above-x*: *above* r $a' \neq \{a'\}$
then obtain b **where**
b-above-b: *above* r b = $\{b\}$
using *lifted-a fin-A insert-Diff insert-not-empty above-one*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence *above* r' b = $\{b\}$ $\vee$ *above* r' a = $\{a\}$
using *lifted-a fin-A lifted-above-winner-alts*
by *metis*
moreover have $\forall\ a''$. *above* r' $a'' = \{a''\} \longrightarrow a'' = a'$
using *all-not-in-conv lifted-a* a'*-above-*a' *fin-A above-one-eq*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
ultimately have $b = a'$
using *a-not-*a'
by *presburger*
moreover have $b \neq a'$
using *not-above-x b-above-b*
by *blast*
ultimately show *False*
by *simp*
qed

end

1.2 Norm

theory *Norm*
  imports *HOL−Library.Extended-Real*
    HOL−Combinatorics.List-Permutation
begin

A norm on R to n is a mapping N: $R \mapsto n$ on R that has the following properties:

- positive scalability: $N(a * u) = |a| * N(u)$ for all u in R to n and all a in R;
- positive semidefiniteness: $N(u) \geq 0$ for all u in R to n, and $N(u) = 0$ if and only if $u = (0,\ 0,\ \ldots,\ 0)$;
- triangle inequality: $N(u + v) \leq N(u) + N(v)$ for all u and v in R to n.

1.2.1 Definition

type-synonym *Norm* = *ereal list* ⇒ *ereal*

definition *norm* :: *Norm* ⇒ *bool* **where**
  norm n ≡ ∀ (*x*::*ereal list*). *n x* ≥ *0* ∧ (∀ *i* < *length x*. (*x*!*i* = *0*) ⟶ *n x* = *0*)

1.2.2 Auxiliary Lemmas

lemma *sum-over-image-of-bijection*:
  fixes
    A :: *′a set* **and**
    A′ :: *′b set* **and**
    f :: *′a* ⇒ *′b* **and**
    g :: *′a* ⇒ *ereal*
  assumes *bij-betw f A A′*
  shows (∑ *a* ∈ *A*. *g a*) = (∑ *a′* ∈ *A′*. *g* (*the-inv-into A f a′*))
  using *assms*
proof (*induction card A arbitrary*: *A A′*)
  case *0*
  hence *card A′* = *0*
    using *bij-betw-same-card assms*
    by *metis*
  hence (∑ *a* ∈ *A*. *g a*) = *0* ∧ (∑ *a′* ∈ *A′*. *g* (*the-inv-into A f a′*)) = *0*
    using *0 card-0-eq sum.empty sum.infinite*
    by *metis*

thus *?case*
  by *simp*
next
  case (*Suc x*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'b$ *set* **and**
    x :: *nat*
  assume
    IH: $\bigwedge A\ A'.\ x = card\ A \Longrightarrow$
      $bij\text{-}betw\ f\ A\ A' \Longrightarrow sum\ g\ A = (\sum\ a \in A'.\ g\ (the\text{-}inv\text{-}into\ A\ f\ a))$ **and**
    suc: $Suc\ x = card\ A$ **and**
    bij-A-A': $bij\text{-}betw\ f\ A\ A'$
  obtain a **where**
    a-in-A: $a \in A$
    using *suc card-eq-SucD insertI1*
    by *metis*
  have *a-compl-A*: $insert\ a\ (A - \{a\}) = A$
    using *a-in-A*
    by *blast*
  have *inj-on-A-A'*: $inj\text{-}on\ f\ A \wedge A' = f\ `\ A$
    using *bij-A-A'*
    unfolding *bij-betw-def*
    by *simp*
  hence *inj-on-A*: $inj\text{-}on\ f\ A$
    by *simp*
  have *img-of-A*: $A' = f\ `\ A$
    using *inj-on-A-A'*
    by *simp*
  have $inj\text{-}on\ f\ (insert\ a\ A)$
    using *inj-on-A a-compl-A*
    by *simp*
  hence *A'-sub-fa*: $A' - \{f\ a\} = f\ `\ (A - \{a\})$
    using *img-of-A*
    by *blast*
  hence *bij-without-a*: $bij\text{-}betw\ f\ (A - \{a\})\ (A' - \{f\ a\})$
    using *inj-on-A a-compl-A inj-on-insert*
    unfolding *bij-betw-def*
    by (*metis* (*no-types*))
  have $\forall\ f\ A\ A'.\ bij\text{-}betw\ f\ (A{::}'a\ set)\ (A'{::}'b\ set) = (inj\text{-}on\ f\ A \wedge f\ `\ A = A')$
    unfolding *bij-betw-def*
    by *simp*
  hence *inv-without-a*:
    $\forall\ a' \in A' - \{f\ a\}.\ the\text{-}inv\text{-}into\ (A - \{a\})\ f\ a' = the\text{-}inv\text{-}into\ A\ f\ a'$
    using *inj-on-A A'-sub-fa*
    by (*simp add*: *inj-on-diff the-inv-into-f-eq*)
  have *card-without-a*: $card\ (A - \{a\}) = x$
    using *suc a-in-A Diff-empty card-Diff-insert diff-Suc-1 empty-iff*
    by *simp*

hence *card-A′-from-x*: *card* $A' = $ *Suc* $x \wedge$ *card* $(A' - \{f\ a\}) = x$
  using *suc bij-A-A′ bij-without-a*
  by (*simp add*: *bij-betw-same-card*)
hence $(\sum a \in A.\ g\ a) = (\sum a \in (A - \{a\}).\ g\ a) + g\ a$
  using *suc add.commute card-Diff1-less-iff insert-Diff insert-Diff-single lessI*
    sum.insert-remove card-without-a
  by *metis*
also have $\ldots = (\sum a' \in (A' - \{f\ a\}).\ g$ (*the-inv-into* $(A - \{a\})\ f\ a')) + g\ a$
  using *IH bij-without-a card-without-a*
  by *simp*
also have $\ldots = (\sum a' \in (A' - \{f\ a\}).\ g$ (*the-inv-into* $A\ f\ a')) + g\ a$
  using *inv-without-a*
  by *simp*
also have $\ldots = (\sum a' \in (A' - \{f\ a\}).\ g$ (*the-inv-into* $A\ f\ a')) +$
    g (*the-inv-into* $A\ f\ (f\ a)$)
  using *a-in-A bij-A-A′*
  by (*simp add*: *bij-betw-imp-inj-on the-inv-into-f-f*)
also have $\ldots = (\sum a' \in A'.\ g$ (*the-inv-into* $A\ f\ a'$))
  using *add.commute card-Diff1-less-iff insert-Diff insert-Diff-single lessI*
    sum.insert-remove card-A′-from-x
  by *metis*
finally show $(\sum a \in A.\ g\ a) = (\sum a' \in A'.\ g$ (*the-inv-into* $A\ f\ a'$))
  by *simp*
qed

1.2.3 Common Norms

fun *l-one* :: *Norm* **where**
  l-one $x = (\sum i <$ *length* $x.\ |x!i|)$

1.2.4 Properties

definition *symmetry* :: *Norm* $\Rightarrow$ *bool* **where**
  symmetry $n \equiv \forall\ x\ y.\ x <\sim\sim> y \longrightarrow n\ x = n\ y$

1.2.5 Theorems

theorem *l-one-is-sym*: *symmetry l-one*
proof (*unfold symmetry-def*, *safe*)
  fix
    l :: *ereal list* **and**
    l' :: *ereal list*
  assume *perm*: $l <\sim\sim> l'$
  from *perm* **obtain** π
    where
      $perm_\pi$: π *permutes* $\{..<$ *length* $l\}$ **and**
      l_π: *permute-list* $\pi\ l = l'$
    using *mset-eq-permutation*
    by *metis*
  from $perm_\pi\ l_\pi$

have $(\sum\ i < \mathit{length}\ l.\ |l'!i|) = (\sum\ i < \mathit{length}\ l.\ |l!(\pi\ i)|)$
using *permute-list-nth*
by *fastforce*
also have $\ldots = (\sum\ i < \mathit{length}\ l.\ |l!(\pi\ (\mathit{inv}\ \pi\ i))|)$
using perm_π *permutes-inv-eq f-the-inv-into-f-bij-betw permutes-imp-bij sum.cong sum-over-image-of-bijection*
by (*smt* (*verit*, *ccfv-SIG*))
also have $\ldots = (\sum\ i < \mathit{length}\ l.\ |l!i|)$
using perm_π *permutes-inv-eq*
by *metis*
finally have $(\sum\ i < \mathit{length}\ l.\ |l'!i|) = (\sum\ i < \mathit{length}\ l.\ |l!i|)$
by *simp*
moreover have *length* l = *length* l'
using *perm perm-length*
by *metis*
ultimately show *l-one* l = *l-one* l'
using *l-one.elims*
by *metis*
qed

end

1.3 Electoral Result

theory *Result*
imports *Main*
begin

An electoral result is the principal result type of the composable modules voting framework, as it is a generalization of the set of winning alternatives from social choice functions. Electoral results are selections of the received (possibly empty) set of alternatives into the three disjoint groups of elected, rejected and deferred alternatives. Any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives.

1.3.1 Auxiliary Functions

type-synonym *'r Result* = *'r set* ∗ *'r set* ∗ *'r set*

A partition of a set A are pairwise disjoint sets that "set equals partition" A. For this specific predicate, we have three disjoint sets in a three-tuple.

fun *disjoint3* :: *'r Result* ⇒ *bool* **where**
disjoint3 (*e*, *r*, *d*) =

$((e \cap r = \{\}) \wedge$
$(e \cap d = \{\}) \wedge$
$(r \cap d = \{\}))$

fun *set-equals-partition* :: $'r\ set \Rightarrow 'r\ Result \Rightarrow bool$ **where**
set-equals-partition $X\ (e,\ r,\ d) = (e \cup r \cup d = X)$

1.3.2 Definition

A result generally is related to the alternative set A (of type 'a). A result should be well-formed on the alternatives. Also it should be possible to limit a well-formed result to a subset of the alternatives.

Specific result types like social choice results (sets of alternatives) can be realized via sublocales of the result locale.

locale *result* =
fixes
well-formed :: $'a\ set \Rightarrow ('r\ Result) \Rightarrow bool$ **and**
limit-set :: $'a\ set \Rightarrow 'r\ set \Rightarrow 'r\ set$
assumes $\bigwedge\ (A::('a\ set))\ (r::('r\ Result))$.
$(\textit{set-equals-partition}\ (\textit{limit-set}\ A\ UNIV)\ r \wedge \textit{disjoint3}\ r) \Longrightarrow \textit{well-formed}\ A\ r$

These three functions return the elect, reject, or defer set of a result.

fun (**in** *result*) *limit-res* :: $'a\ set \Rightarrow 'r\ Result \Rightarrow 'r\ Result$ **where**
limit-res $A\ (e,\ r,\ d) = (\textit{limit-set}\ A\ e,\ \textit{limit-set}\ A\ r,\ \textit{limit-set}\ A\ d)$

abbreviation *elect-r* :: $'r\ Result \Rightarrow 'r\ set$ **where**
elect-r $r \equiv \textit{fst}\ r$

abbreviation *reject-r* :: $'r\ Result \Rightarrow 'r\ set$ **where**
reject-r $r \equiv \textit{fst}\ (\textit{snd}\ r)$

abbreviation *defer-r* :: $'r\ Result \Rightarrow 'r\ set$ **where**
defer-r $r \equiv \textit{snd}\ (\textit{snd}\ r)$

end

1.4 Preference Profile

theory *Profile*
imports *Preference-Relation*
HOL−Library.Extended-Nat
HOL−Combinatorics.Permutations

begin

Preference profiles denote the decisions made by the individual voters on the eligible alternatives. They are represented in the form of one preference relation (e.g., selected on a ballot) per voter, collectively captured in a mapping of voters onto their respective preference relations. If there are finitely many voters, they can be enumerated and the mapping can be interpreted as a list of preference relations. Unlike the common preference profiles in the social-choice sense, the profiles described here consider only the (sub-)set of alternatives that are received.

1.4.1 Definition

A profile contains one ballot for each voter. An election consists of a set of participating voters, a set of eligible alternatives and a corresponding profile.

type-synonym $('a,\ 'v)$ *Profile* $=\ 'v \Rightarrow ('a$ *Preference-Relation*$)$

type-synonym $('a,\ 'v)$ *Election* $=\ 'a$ *set* $\times\ 'v$ *set* $\times\ ('a,\ 'v)$ *Profile*

fun *election-equality* :: $('a,\ 'v)$ *Election* $\Rightarrow ('a,\ 'v)$ *Election* $\Rightarrow$ *bool* **where**
election-equality $(A,\ V,\ p)\ (A',\ V',\ p') = (A = A' \wedge V = V' \wedge (\forall\ v \in V.\ p\ v = p'\ v))$

fun *alternatives-$\mathcal{E}$* :: $('a,\ 'v)$ *Election* $\Rightarrow\ 'a$ *set* **where**
alternatives-$\mathcal{E}$ $E =$ *fst* E

fun *voters-$\mathcal{E}$* :: $('a,\ 'v)$ *Election* $\Rightarrow\ 'v$ *set* **where**
voters-$\mathcal{E}$ $E =$ *fst* $($*snd* $E)$

fun *profile-$\mathcal{E}$* :: $('a,\ 'v)$ *Election* $\Rightarrow ('a,\ 'v)$ *Profile* **where**
profile-$\mathcal{E}$ $E =$ *snd* $($*snd* $E)$

A profile on a set of alternatives A and a voter set V consists of ballots that are linear orders on A for all voters in V. A finite profile is one with finitely many alternatives and voters.

definition *profile* :: $'v$ *set* $\Rightarrow\ 'a$ *set* $\Rightarrow ('a,\ 'v)$ *Profile* $\Rightarrow$ *bool* **where**
profile $V\ A\ p \equiv \forall\ v \in V.$ *linear-order-on* $A\ (p\ v)$

abbreviation *finite-profile* :: $'v$ *set* $\Rightarrow\ 'a$ *set* $\Rightarrow ('a,\ 'v)$ *Profile* $\Rightarrow$ *bool* **where**
finite-profile $V\ A\ p \equiv$ *finite* $A \wedge$ *finite* $V \wedge$ *profile* $V\ A\ p$

abbreviation *finite-election* :: $('a,'v)$ *Election* $\Rightarrow$ *bool* **where**
finite-election $E \equiv$ *finite-profile* $($*voters-$\mathcal{E}$* $E)$ $($*alternatives-$\mathcal{E}$* $E)$ $($*profile-$\mathcal{E}$* $E)$

definition *finite-voter-elections* :: $('a,\ 'v)$ *Election set* **where**
finite-voter-elections $= \{E :: ('a,\ 'v)$ *Election*. *finite* $($*voters-$\mathcal{E}$* $E)\}$

definition *finite-elections* :: ('*a*, '*v*) *Election set* **where**
finite-elections =
{*E* :: ('*a*, '*v*) *Election*. *finite-profile* (*voters-ℰ E*) (*alternatives-ℰ E*) (*profile-ℰ E*)}

definition *valid-elections* :: ('*a*,'*v*) *Election set* **where**
valid-elections = {*E*. *profile* (*voters-ℰ E*) (*alternatives-ℰ E*) (*profile-ℰ E*)}

— Elections with fixed alternatives, finite voters and a default value for the profile value on non-voters.
fun *fixed-alt-elections* :: '*a set* ⇒ ('*a*, '*v*) *Election set* **where**
fixed-alt-elections A = *valid-elections* ∩
{*E*. *alternatives-ℰ E* = *A* ∧ *finite* (*voters-ℰ E*) ∧ (∀ *v*. *v* ∉ *voters-ℰ E* ⟶ *profile-ℰ E v* = {})}

— Counts the occurrences of a ballot in an election, i.e., how many voters chose that exact ballot.
fun *vote-count* :: '*a Preference-Relation* ⇒ ('*a*, '*v*) *Election* ⇒ *nat* **where**
vote-count p E = *card* {*v* ∈ (*voters-ℰ E*). (*profile-ℰ E*) *v* = *p*}

1.4.2 Vote Count

lemma *sum-comp*:
fixes
f :: '*x* ⇒ '*z*::*comm-monoid-add* **and**
g :: '*y* ⇒ '*x* **and**
X :: '*x set* **and**
Y :: '*y set*
assumes *bij-betw g Y X*
shows *sum f X* = *sum* (*f* ∘ *g*) *Y*
using *assms*
proof (*induction card X arbitrary*: *X Y f g*)
case *0*
assume *bij-betw g Y X*
hence *card Y* = *0*
using *bij-betw-same-card 0.hyps*
unfolding *0.hyps*
by *simp*
hence *sum f X* = *0* ∧ *sum* (*f* ∘ *g*) *Y* = *0*
using *assms 0 card-0-eq sum.empty sum.infinite*
by *metis*
thus *?case*
by *simp*
next
case (*Suc n*)
assume
card-X: *Suc n* = *card X* **and**
bij: *bij-betw g Y X* **and**
hyp: ⋀ *X Y f g*. *n* = *card X* ⟹ *bij-betw g Y X* ⟹ *sum f X* = *sum* (*f* ∘ *g*) *Y*

then obtain $x :: 'x$
  where *x-in-X*: $x \in X$
  by *fastforce*
with *bij* **have** *bij-betw* g $(Y - \{the\text{-}inv\text{-}into\ Y\ g\ x\})$ $(X - \{x\})$
  using *bij-betw-DiffI bij-betw-apply bij-betw-singletonI bij-betw-the-inv-into empty-subsetI f-the-inv-into-f-bij-betw insert-subsetI*
  by (*metis* (*mono-tags*, *lifting*))
moreover have $n = card\ (X - \{x\})$
  using *card-X x-in-X*
  by *fastforce*
ultimately have $sum\ f\ (X - \{x\}) = sum\ (f \circ g)\ (Y - \{the\text{-}inv\text{-}into\ Y\ g\ x\})$
  using *hyp Suc*
  by *blast*
moreover have
  $sum\ (f \circ g)\ Y = f\ (g\ (the\text{-}inv\text{-}into\ Y\ g\ x)) + sum\ (f \circ g)\ (Y - \{the\text{-}inv\text{-}into\ Y\ g\ x\})$
  using *Suc.hyps*(*2*) *x-in-X bij bij-betw-def calculation card.infinite f-the-inv-into-f-bij-betw nat.discI sum.reindex sum.remove*
  by *metis*
moreover have $f\ (g\ (the\text{-}inv\text{-}into\ Y\ g\ x)) + sum\ (f \circ g)\ (Y - \{the\text{-}inv\text{-}into\ Y\ g\ x\}) =$
  $f\ x + sum\ (f \circ g)\ (Y - \{the\text{-}inv\text{-}into\ Y\ g\ x\})$
  using *x-in-X bij f-the-inv-into-f-bij-betw*
  by *metis*
moreover have $sum\ f\ X = f\ x + sum\ f\ (X - \{x\})$
  using *Suc.hyps*(*2*) *Zero-neq-Suc x-in-X card.infinite sum.remove*
  by *metis*
ultimately show *?case*
  by *simp*
qed

lemma *vote-count-sum*:
fixes $E :: ('a, 'v)\ Election$
assumes
  finite (*voters-$\mathcal{E}$ E*) **and**
  finite $(UNIV::('a \times 'a)\ set)$
shows $sum\ (\lambda\ p.\ vote\text{-}count\ p\ E)\ UNIV = card\ (voters\text{-}\mathcal{E}\ E)$
proof (*unfold vote-count.simps*)
have $\forall\ p.\ finite\ \{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\}$
  using *assms*
  by *force*
moreover have $disjoint\ \{\{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\} \mid p.\ p \in UNIV\}$
  unfolding *disjoint-def*
  by *blast*
moreover have *partition*:
  $voters\text{-}\mathcal{E}\ E = \bigcup\ \{\{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\} \mid p.\ p \in UNIV\}$
  using *Union-eq*[*of* $\{\{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\} \mid p.\ p \in UNIV\}$]
  by *blast*
ultimately have *card-eq-sum'*:

card (*voters-$\mathcal{E}$ E*) = *sum card* {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*. *p* ∈ *UNIV*}
using *card-Union-disjoint*[*of* {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*. *p* ∈ *UNIV*}]
by *auto*
have *finite* {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*. *p* ∈ *UNIV*}
using *partition assms*
by (*simp add*: *finite-UnionD*)
moreover have
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*. *p* ∈ *UNIV*} =
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}} ∪
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} = {}}
by *blast*
moreover have
{} = {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}} ∩
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} = {}}
by *blast*
ultimately have *sum card* {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*. *p* ∈ *UNIV*} =
sum card {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}} +
sum card {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} = {}}
using *sum.union-disjoint*[*of*
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}}
{{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} = {}}]
by *simp*
moreover have
∀ *X* ∈ {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} = {}}. *card X* = *0*
using *card-eq-0-iff*
by *fastforce*
ultimately have *card-eq-sum*:
card (*voters-$\mathcal{E}$ E*) = *sum card* {{*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} | *p*.
p ∈ *UNIV* ∧ {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}}
using *card-eq-sum'*
by *simp*
have *inj-on* (λ *p*. {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*})
{*p*. {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*} ≠ {}}
unfolding *inj-on-def*
by *blast*
moreover have
(λ *p*. {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* = *p*}) ‘ {*p*. {*v* ∈ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v*

$= p\} \neq \{\}\} \subseteq$
$\{\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \mid p.$
$p \in \textit{UNIV} \wedge \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
by *blast*
moreover have
$(\lambda\ p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\})$ ‘ $\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v$
$= p\} \neq \{\}\} \supseteq$
$\{\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \mid p.$
$p \in \textit{UNIV} \wedge \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
by *blast*
ultimately have *bij-betw* $(\lambda\ p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\})$
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\{\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \mid p.$
$p \in \textit{UNIV} \wedge \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
unfolding *bij-betw-def*
by *simp*
hence *sum-rewrite*:
$(\sum\ x \in \{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}.$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = x\}) =$
$\textit{sum card}\ \{\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \mid p.$
$p \in \textit{UNIV} \wedge \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
using *sum-comp*[*of*
$\lambda\ p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\}$
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\{\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \mid p.$
$p \in \textit{UNIV} \wedge \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
card]
unfolding *comp-def*
by *simp*
have $\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} = \{\}\} \cap$
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\} = \{\}$
by *blast*
moreover have $\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} = \{\}\} \cup$
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\} = \textit{UNIV}$
by *blast*
ultimately have $(\sum\ p \in \textit{UNIV}.\ \textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\}) =$
$(\sum\ x \in \{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}.$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = x\}) +$
$(\sum\ x \in \{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} = \{\}\}.$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = x\})$
using *assms sum.union-disjoint*[*of*
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} = \{\}\}$
$\{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$]
using *Finite-Set.finite-set add.commute finite-Un*
by (*metis* (*mono-tags*, *lifting*))
moreover have
$\forall\ x \in \{p.\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} = \{\}\}.$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = x\} = \mathit{0}$
using *card-eq-0-iff*

by *fastforce*
ultimately show $(\sum\ p \in \mathit{UNIV}.\ \mathit{card}\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}) = \mathit{card}\ (\mathit{voters}\text{-}\mathcal{E}\ E)$
using *card-eq-sum sum-rewrite*
by *simp*
qed

1.4.3 Voter Permutations

A common action of interest on elections is renaming the voters, e.g., when talking about anonymity.

fun *rename* :: $('v \Rightarrow 'v) \Rightarrow ('a, 'v)$ *Election* $\Rightarrow ('a, 'v)$ *Election* **where**
rename π $(A,\ V,\ p) = (A,\ \pi\ \text{`}\ V,\ p \circ (\mathit{the}\text{-}\mathit{inv}\ \pi))$

lemma *rename-sound*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
prof: *profile* $V\ A\ p$ **and**
renamed: $(A,\ V',\ q) =$ *rename* π $(A,\ V,\ p)$ **and**
bij: *bij* π
shows *profile* $V'\ A\ q$
proof (*unfold profile-def*, *safe*)
fix v' :: $'v$
assume v'-*in*-V': $v' \in V'$
let *?q-img* $= ((\mathit{the}\text{-}\mathit{inv})\ \pi)\ v'$
have $V' = \pi\ \text{`}\ V$
using *renamed*
by *simp*
hence *?q-img* $\in V$
using *UNIV-I* v'-*in*-V' *bij bij-is-inj bij-is-surj*
f-the-inv-into-f inj-image-mem-iff
by *metis*
hence *linear-order-on* A (p *?q-img*)
using *prof*
unfolding *profile-def*
by *simp*
moreover have $q\ v' = p$ *?q-img*
using *renamed bij*
by *simp*
ultimately show *linear-order-on* A $(q\ v')$
by *simp*
qed

lemma *rename-finite*:
fixes

```
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    π :: 'v ⇒ 'v
  assumes
    prof: finite-profile V A p and
    renamed: (A, V', q) = rename π (A, V, p) and
    bij: bij π
  shows finite-profile V' A q
proof (safe)
  show finite A
    using prof
    by simp
  show finite V'
    using bij renamed prof
    by simp
  show profile V' A q
    using assms rename-sound
    by metis
qed

lemma rename-inv:
  fixes
    π :: 'v ⇒ 'v and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes bij π
  shows rename π (rename (the-inv π) (A, V, p)) = (A, V, p)
proof −
  have rename π (rename (the-inv π) (A, V, p)) =
    (A, π ` (the-inv π) ` V, p ∘ (the-inv (the-inv π)) ∘ (the-inv π))
    by simp
  moreover have π ` (the-inv π) ` V = V
    using assms
    by (simp add: f-the-inv-into-f-bij-betw image-comp)
  moreover have (the-inv (the-inv π)) = π
    using assms bij-betw-def inj-on-the-inv-into surj-def surj-imp-inv-eq the-inv-f-f
    by (metis (mono-tags, opaque-lifting))
  moreover have π ∘ (the-inv π) = id
    using assms f-the-inv-into-f-bij-betw
    by fastforce
  ultimately show rename π (rename (the-inv π) (A, V, p)) = (A, V, p)
    by (simp add: rewriteR-comp-comp)
qed

lemma rename-inj:
  fixes π :: 'v ⇒ 'v
  assumes bij π
```

shows *inj* (*rename* π)
proof (*unfold inj-def*, *clarsimp*)
  fix
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    p' :: $('a, 'v)$ *Profile*
  assume
    eq-V: π ‘ $V = \pi$ ‘ V' **and**
    $p \circ$ *the-inv* $\pi = p' \circ$ *the-inv* π
  hence $p \circ$ *the-inv* $\pi \circ \pi = p' \circ$ *the-inv* $\pi \circ \pi$
    by *simp*
  hence $p = p'$
    using *assms bij-betw-the-inv-into bij-is-surj surj-fun-eq*
    by *metis*
  moreover have $V = V'$
    using *assms eq-V*
    by (*simp add*: *bij-betw-imp-inj-on inj-image-eq-iff*)
  ultimately show $V = V' \wedge p = p'$
    by *blast*
qed

lemma *rename-surj*:
  fixes π :: $'v \Rightarrow 'v$
  assumes *bij* π
  shows
    on-valid-els: *rename* π ‘ *valid-elections* = *valid-elections* **and**
    on-finite-els: *rename* π ‘ *finite-elections* = *finite-elections*
proof (*safe*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    p' :: $('a, 'v)$ *Profile*
  assume *valid*: $(A, V, p) \in$ *valid-elections*
  have *bij* (*the-inv* π)
    using *assms bij-betw-the-inv-into*
    by *blast*
  hence *rename* (*the-inv* π) $(A, V, p) \in$ *valid-elections*
    using *rename-sound valid*
    unfolding *valid-elections-def*
    by *fastforce*
  thus $(A, V, p) \in$ *rename* π ‘ *valid-elections*
    using *assms image-eqI rename-inv*[*of* π]
    by *metis*
  assume $(A', V', p') =$ *rename* π (A, V, p)
  thus $(A', V', p') \in$ *valid-elections*

using *rename-sound valid assms*
unfolding *valid-elections-def*
by *fastforce*
next
fix
A :: $'b$ *set* **and**
A' :: $'b$ *set* **and**
V :: $'v$ *set* **and**
V' :: $'v$ *set* **and**
p :: $('b, 'v)$ *Profile* **and**
p' :: $('b, 'v)$ *Profile*
assume *finite*: $(A, V, p) \in$ *finite-elections*
have *bij* (*the-inv* π)
using *assms bij-betw-the-inv-into*
by *blast*
hence *rename* (*the-inv* π) $(A, V, p) \in$ *finite-elections*
using *rename-finite finite*
unfolding *finite-elections-def*
by *fastforce*
thus $(A, V, p) \in$ *rename* π ‘ *finite-elections*
using *assms image-eqI rename-inv*[*of* π]
by *metis*
assume (A', V', p') = *rename* π (A, V, p)
thus $(A', V', p') \in$ *finite-elections*
using *rename-sound finite assms*
unfolding *finite-elections-def*
by *fastforce*
qed

1.4.4 List Representation for Ordered Voter Types

A profile on a voter set that has a natural order can be viewed as a list of ballots.

fun *to-list* :: $'v$::*linorder set* $\Rightarrow$ $('a, 'v)$ *Profile* $\Rightarrow$ $('a$ *Preference-Relation*$)$ *list*
where
to-list V p = (*if* (*finite* V)
then (*map* p (*sorted-list-of-set* V))
else [])

lemma *map2-helper*:
fixes
f :: $'x \Rightarrow 'y \Rightarrow 'z$ **and**
g :: $'x \Rightarrow 'x$ **and**
h :: $'y \Rightarrow 'y$ **and**
l1 :: $'x$ *list* **and**
l2 :: $'y$ *list*
shows *map2* f (*map* g *l1*) (*map* h *l2*) = *map2* (λ x y. f (g x) (h y)) *l1 l2*
proof –
have *map2* f (*map* g *l1*) (*map* h *l2*) = *map* (λ (x, y). f x y) (*zip* (*map* g *l1*) (*map*

$h\ l2$))
    by *simp*
  moreover have *map* (λ (x, y). $f\ x\ y$) (*zip* (*map* $g\ l1$) (*map* $h\ l2$)) =
    map (λ (x, y). $f\ x\ y$) (*map* (λ (x, y). ($g\ x$, $h\ y$)) (*zip* $l1\ l2$))
    using *zip-map-map*
    by *metis*
  moreover have *map* (λ (x, y). $f\ x\ y$) (*map* (λ (x, y). ($g\ x$, $h\ y$)) (*zip* $l1\ l2$)) =
    map ((λ (x, y). $f\ x\ y$) $\circ$ (λ (x, y). ($g\ x$, $h\ y$))) (*zip* $l1\ l2$)
    by *simp*
  moreover have *map* ((λ (x, y). $f\ x\ y$) $\circ$ (λ (x, y). ($g\ x$, $h\ y$))) (*zip* $l1\ l2$) =
    map (λ (x, y). f ($g\ x$) ($h\ y$)) (*zip* $l1\ l2$)
    by *auto*
  moreover have *map* (λ (x, y). f ($g\ x$) ($h\ y$)) (*zip* $l1\ l2$) = *map2* ($\lambda\ x\ y$. f ($g\ x$) ($h\ y$)) $l1\ l2$
    by *simp*
  ultimately show
    map2 f (*map* $g\ l1$) (*map* $h\ l2$) = *map2* ($\lambda\ x\ y$. f ($g\ x$) ($h\ y$)) $l1\ l2$
    by *simp*
qed

lemma *to-list-simp*:
  fixes
    i :: *nat* **and**
    V :: *'v::linorder set* **and**
    p :: (*'a*, *'v*) *Profile*
  assumes
    i < *card* V
  shows (*to-list* $V\ p$)!i = p ((*sorted-list-of-set* V)!i)
proof –
  have (*to-list* $V\ p$)!i = (*map* p (*sorted-list-of-set* V))!i
    by *simp*
  also have ... = p ((*sorted-list-of-set* V)!i)
    using *assms*
    by *simp*
  finally show *?thesis*
    by *simp*
qed

lemma *to-list-comp*:
  fixes
    V :: *'v::linorder set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    f :: *'a rel* $\Rightarrow$ *'a rel*
  shows *to-list* V ($f \circ p$) = *map* f (*to-list* $V\ p$)
proof –
  have $\forall$ i < *card* V. (*to-list* V ($f \circ p$))!i = ($f \circ p$) ((*sorted-list-of-set* V)!i)
    using *to-list-simp*
    by *blast*
  moreover have

$\forall\ i < card\ V.\ (f \circ p)\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i) = (map\ (f \circ p)\ (sorted\text{-}list\text{-}of\text{-}set\ V))!i$
unfolding *map-def*
by *simp*
moreover have
$\forall\ i < card\ V.\ (map\ (f \circ p)\ (sorted\text{-}list\text{-}of\text{-}set\ V))!i =$
$(map\ f\ (map\ p\ (sorted\text{-}list\text{-}of\text{-}set\ V)))!i$
by *simp*
moreover have $map\ p\ (sorted\text{-}list\text{-}of\text{-}set\ V) = to\text{-}list\ V\ p$
using *to-list-simp list-eq-iff-nth-eq*
by *simp*
ultimately have $\forall\ i < card\ V.\ (to\text{-}list\ V\ (f \circ p))!i = (map\ f\ (to\text{-}list\ V\ p))!i$
by *presburger*
moreover have $length\ (map\ f\ (to\text{-}list\ V\ p)) = card\ V$
by *simp*
moreover have $length\ (to\text{-}list\ V\ (f \circ p)) = card\ V$
by *simp*
ultimately show *?thesis*
using *nth-equalityI*
by *simp*
qed

lemma *set-card-upper-bound*:
fixes
i :: *nat* **and**
V :: *nat set*
assumes
fin-V: *finite V* **and**
bound-v: $\forall\ v \in V.\ i > v$
shows $i \geq card\ V$
proof $(cases\ V = \{\})$
case *True*
thus *?thesis*
by *simp*
next
case *False*
hence $Max\ V \in V$
using *fin-V*
by *simp*
moreover have $Max\ V \geq (card\ V) - 1$
using *False Max-ge-iff fin-V calculation card-Diff1-less finite-le-enumerate card-Diff-singleton finite-enumerate-in-set*
by *metis*
ultimately show *?thesis*
using *fin-V bound-v*
by *fastforce*
qed

lemma *sorted-list-of-set-nth-equals-card*:

fixes
  V :: $'v$::*linorder set* **and**
  x :: $'v$
assumes
  fin-V: *finite* V **and**
  x-V: $x \in V$
shows *sorted-list-of-set* $V!(card\ \{v \in V.\ v < x\}) = x$
proof −
  let $?c = card\ \{v \in V.\ v < x\}$ **and**
    $?set = \{v \in V.\ v < x\}$
  have *ex-index*: $\forall\ v \in V.\ \exists\ n.\ n < card\ V \wedge (sorted\text{-}list\text{-}of\text{-}set\ V!n) = v$
    using *sorted-list-of-set.distinct-sorted-key-list-of-set*
      sorted-list-of-set.length-sorted-key-list-of-set
      sorted-list-of-set.set-sorted-key-list-of-set
      distinct-Ex1 fin-V
    by *metis*
  then obtain φ **where**
    index-φ: $\forall\ v \in V.\ \varphi\ v < card\ V \wedge (sorted\text{-}list\text{-}of\text{-}set\ V!(\varphi\ v)) = v$
    by *metis*

  let $?i = \varphi\ x$
  have *inj-φ*: *inj-on* φ V
    using *inj-onI index-φ*
    by *metis*
  have *mono-φ*: $\forall\ v\ v'.\ v \in V \wedge v' \in V \wedge v < v' \longrightarrow \varphi\ v < \varphi\ v'$
    using *sorted-list-of-set.idem-if-sorted-distinct dual-order.strict-trans2 fin-V index-φ*
      finite-sorted-distinct-unique linorder-neqE-nat sorted-wrt-iff-nth-less
      sorted-list-of-set.length-sorted-key-list-of-set order-less-irrefl
    by (*metis* (*full-types*))
  have $\forall\ v \in ?set.\ v < x$
    by *simp*
  hence $\forall\ v \in ?set.\ \varphi\ v < ?i$
    using *mono-φ x-V*
    by *simp*
  hence $\forall\ j \in \{\varphi\ v \mid v.\ v \in ?set\}.\ ?i > j$
    by *blast*
  moreover have *fin-img*: *finite* $?set$
    using *fin-V*
    by *simp*
  ultimately have $?i \geq card\ \{\varphi\ v \mid v.\ v \in ?set\}$
    using *set-card-upper-bound*
    by *simp*
  also have $card\ \{\varphi\ v \mid v.\ v \in ?set\} = ?c$
    using *inj-φ*
    by (*simp add*: *card-image inj-on-subset setcompr-eq-image*)
  finally have *geq*: $?i \geq ?c$
    by *simp*
  have *sorted-φ*:

$\forall\ i\,j.\ i < card\ V \wedge j < card\ V \wedge i < j$
$\longrightarrow (sorted\text{-}list\text{-}of\text{-}set\ V!i) < (sorted\text{-}list\text{-}of\text{-}set\ V!j)$
by (*simp add*: *sorted-wrt-nth-less*)
have *leq*: $?i \leq ?c$
proof (*rule ccontr*, *cases* $?c < card\ V$)
case *True*
let $?A = \lambda\ j.\ \{sorted\text{-}list\text{-}of\text{-}set\ V!j\}$
assume $\neg\ ?i \leq ?c$
hence $?i > ?c$
by *simp*
hence $\forall\ j \leq ?c.\ sorted\text{-}list\text{-}of\text{-}set\ V!j \in V \wedge sorted\text{-}list\text{-}of\text{-}set\ V!j < x$
using *sorted-φ dual-order.strict-trans2 geq index-φ x-V fin-V*
nth-mem sorted-list-of-set.length-sorted-key-list-of-set
sorted-list-of-set.set-sorted-key-list-of-set
by (*metis* (*mono-tags*, *lifting*))
hence $\{sorted\text{-}list\text{-}of\text{-}set\ V!j \mid j.\ j \leq ?c\} \subseteq \{v \in V.\ v < x\}$
by *blast*
also have $\{sorted\text{-}list\text{-}of\text{-}set\ V!j \mid j.\ j \leq ?c\}$
$= \{sorted\text{-}list\text{-}of\text{-}set\ V!j \mid j.\ j \in \{0\ ..< (?c+1)\}\}$
using *add.commute*
by *auto*
also have $\{sorted\text{-}list\text{-}of\text{-}set\ V!j \mid j.\ j \in \{0\ ..< (?c+1)\}\}$
$= (\bigcup\ j \in \{0\ ..< (?c+1)\}.\ \{sorted\text{-}list\text{-}of\text{-}set\ V!j\})$
by *blast*
finally have *subset*: $(\bigcup\ j \in \{0\ ..< (?c+1)\}.\ ?A\ j) \subseteq \{v \in V.\ v < x\}$
by *simp*
have $\forall\ i \leq ?c.\ \forall\ j \leq ?c.\ i \neq j \longrightarrow sorted\text{-}list\text{-}of\text{-}set\ V!i \neq sorted\text{-}list\text{-}of\text{-}set$
$V!j$
using *True*
by (*simp add*: *nth-eq-iff-index-eq*)
hence $\forall\ i \in \{0\ ..< (?c+1)\}.\ \forall\ j \in \{0\ ..< (?c+1)\}.$
$(i \neq j \longrightarrow \{sorted\text{-}list\text{-}of\text{-}set\ V!i\} \cap \{sorted\text{-}list\text{-}of\text{-}set\ V!j\} = \{\})$
by *fastforce*
hence *disjoint-family-on* $?A\ \{0\ ..< (?c+1)\}$
unfolding *disjoint-family-on-def*
by *simp*
moreover have *finite* $\{0\ ..< (?c+1)\}$
by *simp*
moreover have $\forall\ j \in \{0\ ..< (?c+1)\}.\ card\ (?A\ j) = 1$
by *simp*
ultimately have $card\ (\bigcup\ j \in \{0\ ..< (?c+1)\}.\ ?A\ j) = (\sum\ j \in \{0\ ..< (?c+1)\}.$
$1)$
using *card-UN-disjoint′*
by *fastforce*
also have $(\sum\ j \in \{0\ ..< (?c+1)\}.\ 1) = ?c + 1$
by *auto*
finally have $card\ (\bigcup\ j \in \{0\ ..< (?c+1)\}.\ ?A\ j) = ?c + 1$
by *simp*
hence $?c + 1 \leq ?c$

using *subset card-mono fin-img*
by (*metis* (*no-types*, *lifting*))
thus *False*
by *simp*
next
case *False*
assume $\neg$ *?i* $\leq$ *?c*
thus *False*
using *False x-V index-*φ *geq order-le-less-trans*
by *blast*
qed
thus *?thesis*
using *geq leq x-V index-*φ
by *simp*
qed

lemma *to-list-permutes-under-bij*:
fixes
π :: $'v$::*linorder* $\Rightarrow$ $'v$ **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile*
assumes
bij: *bij* π
shows
let φ = (λ *i*. *card* {$v \in \pi$ ‘ V. $v < \pi$ ((*sorted-list-of-set* V)!i)})
in (*to-list* V p) = *permute-list* φ (*to-list* (π ‘ V) (λ *x*. *p* (*the-inv* π *x*)))
proof (*cases finite* V)
case *False*

hence *to-list* V p = []
by *simp*
moreover have *to-list* (π ‘ V) (λ *x*. *p* (*the-inv* π *x*)) = []
proof −
have *infinite* (π ‘ V)
using *False assms bij-betw-finite bij-betw-subset top-greatest*
by *metis*
thus *?thesis*
by *simp*
qed
ultimately show *?thesis*
by *simp*
next
case *True*
let
?q = λ *x*. *p* (*the-inv* π *x*) **and**
?img = π ‘ V **and**
?n = *length* (*to-list* V p) **and**
?perm = λ *i*. *card* {$v \in \pi$ ‘ V. $v < \pi$ ((*sorted-list-of-set* V)!i)}

have *card-eq*: *card ?img* = *card V*
  using *assms bij-betw-same-card bij-betw-subset top-greatest*
  by *metis*
also have *card-length-V*: *?n* = *card V*
  by *simp*
also have *card-length-img*: *length* (*to-list ?img ?q*) = *card ?img*
  using *True*
  by *simp*
finally have *eq-length*: *length* (*to-list ?img ?q*) = *?n*
  by *simp*
show *?thesis*
proof (*unfold Let-def permute-list-def*, *rule nth-equalityI*)

  show *length* (*to-list V p*) =
        length
          (*map* (*λ i. to-list ?img ?q* ! *card* {*v* ∈ *?img*. *v* < *π* (*sorted-list-of-set*
V!*i*)})
              [*0* ..< *length* (*to-list ?img ?q*)])
    using *eq-length*
    by *simp*
next

  fix *i* :: *nat*
  assume *in-bnds*: *i* < *?n*
  let *?c* = *card* {*v* ∈ *?img*. *v* < *π* (*sorted-list-of-set V*!*i*)}
  have *map* (*λ i.* (*to-list ?img ?q*)!*?c*) [*0* ..< *?n*]!*i* = *p* ((*sorted-list-of-set V*)!*i*)
  proof −
    have ∀ *v. v* ∈ *?img* ⟶ {*v′* ∈ *?img. v′* < *v*} ⊆ *?img* − {*v*}
      by *blast*
    moreover have *elem-of-img*: *π* (*sorted-list-of-set V*!*i*) ∈ *?img*
      using *True in-bnds image-eqI nth-mem card-length-V*
          sorted-list-of-set.length-sorted-key-list-of-set
          sorted-list-of-set.set-sorted-key-list-of-set
      by *metis*
    ultimately have {*v* ∈ *?img. v* < *π* (*sorted-list-of-set V*!*i*)}
                ⊆ *?img* − {*π* (*sorted-list-of-set V*!*i*)}
      by *simp*
    hence {*v* ∈ *?img. v* < *π* (*sorted-list-of-set V*!*i*)} ⊂ *?img*
      using *elem-of-img*
      by *blast*
    moreover have *img-card-eq-V-length*: *card ?img* = *?n*
      using *card-eq card-length-V*
      by *presburger*
    ultimately have *card-in-bnds*: *?c* < *?n*
      using *True finite-imageI psubset-card-mono*
      by (*metis* (*mono-tags*, *lifting*))
    moreover have *img-list-map*:
      map (*λ i. to-list ?img ?q*!*?c*) [*0* ..< *?n*]!*i* = *to-list ?img ?q*!*?c*
      using *in-bnds*

```
      by simp
    also have img-list-card-eq-inv-img-list:
      to-list ?img ?q!?c = ?q ((sorted-list-of-set ?img)!?c)
      using in-bnds to-list-simp in-bnds img-card-eq-V-length card-in-bnds
      by (metis (no-types, lifting))
    also have img-card-eq-img-list-i:
      (sorted-list-of-set ?img)!?c = π (sorted-list-of-set V!i)
      using True elem-of-img sorted-list-of-set-nth-equals-card
      by blast
    finally show ?thesis
      using assms bij-betw-imp-inj-on the-inv-f-f
            img-list-map img-card-eq-img-list-i
            img-list-card-eq-inv-img-list
      by metis
  qed
  also have to-list V p!i = p ((sorted-list-of-set V)!i)
    using True in-bnds
    by simp
  finally show to-list V p!i =
      map (λ i. (to-list ?img ?q)!(card {v ∈ ?img. v < π (sorted-list-of-set V !
i)}))
       [0 ..< length (to-list ?img ?q)]!i
    using in-bnds eq-length Collect-cong card-eq
    by simp
 qed
qed
```

1.4.5 Preference Counts and Comparisons

The win count for an alternative a with respect to a finite voter set V in a profile p is the amount of ballots from V in p that rank alternative a in first position. If the voter set is infinite, counting is not generally possible.

```
fun win-count :: 'v set ⇒ ('a, 'v) Profile ⇒ 'a ⇒ enat where
 win-count V p a = (if (finite V)
  then card {v ∈ V. above (p v) a = {a}} else infinity)

fun prefer-count :: 'v set ⇒ ('a, 'v) Profile ⇒ 'a ⇒ 'a ⇒ enat where
 prefer-count V p x y = (if (finite V)
    then card {v ∈ V. (let r = (p v) in (y ≼_r x))} else infinity)

lemma pref-count-voter-set-card:
 fixes
  V :: 'v set and
  p :: ('a, 'v) Profile and
  a :: 'a and
  b :: 'a
 assumes fin-V: finite V
 shows prefer-count V p a b ≤ card V
proof (simp)
```

have $\{v \in V.\ (b,\ a) \in p\ v\} \subseteq V$
  by *simp*
hence $card\ \{v \in V.\ (b,\ a) \in p\ v\} \leq card\ V$
  using *fin-V Finite-Set.card-mono*
  by *metis*
thus $(finite\ V \longrightarrow card\ \{v \in V.\ (b,\ a) \in p\ v\} \leq card\ V) \wedge finite\ V$
  using *fin-V*
  by *simp*
qed

lemma *set-compr*:
  fixes
    $A :: 'a\ set$ **and**
    $f :: 'a \Rightarrow 'a\ set$
  shows $\{f\ x \mid x.\ x \in A\} = f\ `\ A$
  by *auto*

lemma *pref-count-set-compr*:
  fixes
    $A :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a,\ 'v)\ Profile$ **and**
    $a :: 'a$
  shows $\{prefer\text{-}count\ V\ p\ a\ a' \mid a'.\ a' \in A - \{a\}\} = (prefer\text{-}count\ V\ p\ a)\ `\ (A - \{a\})$
  by *auto*

lemma *pref-count*:
  fixes
    $A :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a,\ 'v)\ Profile$ **and**
    $a :: 'a$ **and**
    $b :: 'a$
  assumes
    prof: *profile* $V\ A\ p$ **and**
    fin: *finite* V **and**
    a-in-A: $a \in A$ **and**
    b-in-A: $b \in A$ **and**
    neq: $a \neq b$
  shows $prefer\text{-}count\ V\ p\ a\ b = card\ V - (prefer\text{-}count\ V\ p\ b\ a)$
proof −
  have $\forall\ v \in V.\ connex\ A\ (p\ v)$
    using *prof*
    unfolding *profile-def*
    by (*simp add*: *lin-ord-imp-connex*)
  hence *asym*: $\forall\ v \in V.\ \neg\ (let\ r = (p\ v)\ in\ (b \preceq_r a)) \longrightarrow (let\ r = (p\ v)\ in\ (a \preceq_r b))$
    using *a-in-A b-in-A*

unfolding *connex-def*
by *metis*
have $\forall\ v \in V.\ ((b, a) \in (p\ v) \longrightarrow (a, b) \notin (p\ v))$
using *antisymD neq lin-imp-antisym prof*
unfolding *profile-def*
by *metis*
hence $\{v \in V.\ (let\ r = (p\ v)\ in\ (b \preceq_r a))\} =$
$V - \{v \in V.\ (let\ r = (p\ v)\ in\ (a \preceq_r b))\}$
using *asym*
by *auto*
thus *?thesis*
by (*simp add*: *card-Diff-subset Collect-mono fin*)
qed

lemma *pref-count-sym*:
fixes
p :: (*'a*, *'v*) *Profile* **and**
V :: *'v set* **and**
a :: *'a* **and**
b :: *'a* **and**
c :: *'a*
assumes
pref-count-ineq: *prefer-count* $V\ p\ a\ c \geq$ *prefer-count* $V\ p\ c\ b$ **and**
prof: *profile* $V\ A\ p$ **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
c-in-A: $c \in A$ **and**
a-neq-c: $a \neq c$ **and**
c-neq-b: $c \neq b$
shows *prefer-count* $V\ p\ b\ c \geq$ *prefer-count* $V\ p\ c\ a$
proof (*cases*)
assume *fin-V*: *finite* V
have *nat1*: *prefer-count* $V\ p\ c\ a \in \mathbb{N}$
unfolding *Nats-def*
using *of-nat-eq-enat fin-V*
by *simp*
have *nat2*: *prefer-count* $V\ p\ b\ c \in \mathbb{N}$
unfolding *Nats-def*
using *of-nat-eq-enat fin-V*
by *simp*
have *smaller*: *prefer-count* $V\ p\ c\ a \leq$ *card* V
using *prof fin-V pref-count-voter-set-card*
by *metis*
have *prefer-count* $V\ p\ a\ c =$ *card* $V -$ (*prefer-count* $V\ p\ c\ a$)
using *pref-count prof a-in-A c-in-A a-neq-c fin-V*
by (*metis* (*no-types*, *opaque-lifting*))
moreover have *pref-count-b-eq*:
prefer-count $V\ p\ c\ b =$ *card* $V -$ (*prefer-count* $V\ p\ b\ c$)
using *pref-count prof a-in-A c-in-A a-neq-c b-in-A c-neq-b fin-V*

by *metis*
hence *ineq*: *card V* − (*prefer-count V p b c*) ≤ *card V* − (*prefer-count V p c a*)
using *calculation pref-count-ineq*
by *simp*
hence *card V* − (*prefer-count V p b c*) + (*prefer-count V p c a*) ≤
card V − (*prefer-count V p c a*) + (*prefer-count V p c a*)
using *pref-count-b-eq pref-count-ineq*
by *auto*
hence *card V* + (*prefer-count V p c a*) ≤ *card V* + (*prefer-count V p b c*)
using *nat1 nat2 fin-V smaller*
by *simp*
thus *?thesis*
by *simp*
next
assume *inf-V*: *infinite V*
have *prefer-count V p c a* = *infinity*
using *inf-V*
by *simp*
moreover have *prefer-count V p b c* = *infinity*
using *inf-V*
by *simp*
thus *?thesis*
by *simp*
qed

lemma *empty-prof-imp-zero-pref-count*:
fixes
p :: (′*a*, ′*v*) *Profile* **and**
V :: ′*v set* **and**
a :: ′*a* **and**
b :: ′*a*
assumes *V* = {}
shows *prefer-count V p a b* = *0*
unfolding *zero-enat-def*
using *assms*
by *simp*

fun *wins* :: ′*v set* ⇒ ′*a* ⇒ (′*a*, ′*v*) *Profile* ⇒ ′*a* ⇒ *bool* **where**
wins V a p b =
(*prefer-count V p a b* > *prefer-count V p b a*)

lemma *wins-inf-voters*:
fixes
p :: (′*a*, ′*v*) *Profile* **and**
a :: ′*a* **and**
b :: ′*a* **and**
V :: ′*v set*

assumes *infinite V*
shows *wins V b p a = False*
using *assms*
by *simp*

Alternative a wins against b implies that b does not win against a.

lemma *wins-antisym*:
fixes
p :: ('a, 'v) *Profile* **and**
a :: 'a **and**
b :: 'a **and**
V :: 'v *set*
assumes *wins V a p b*
shows ¬ *wins V b p a*
using *assms*
by *simp*

lemma *wins-irreflex*:
fixes
p :: ('a, 'v) *Profile* **and**
a :: 'a **and**
V :: 'v *set*
shows ¬ *wins V a p a*
using *wins-antisym*
by *metis*

1.4.6 Condorcet Winner

fun *condorcet-winner* :: 'v *set* ⇒ 'a *set* ⇒ ('a, 'v) *Profile* ⇒ 'a ⇒ *bool* **where**
condorcet-winner V A p a =
(*finite-profile V A p* ∧ $a \in A$ ∧ ($\forall\ x \in A - \{a\}$. *wins V a p x*))

lemma *cond-winner-unique-eq*:
fixes
V :: 'v *set* **and**
A :: 'a *set* **and**
p :: ('a, 'v) *Profile* **and**
a :: 'a **and**
b :: 'a
assumes
condorcet-winner V A p a **and**
condorcet-winner V A p b
shows $b = a$
proof (*rule ccontr*)
assume *b-neq-a*: $b \neq a$
have *wins V b p a*
using *b-neq-a insert-Diff insert-iff assms*
by *simp*

hence ¬ *wins V a p b*
  by (*simp add*: *wins-antisym*)
moreover have *a-wins-against-b*: *wins V a p b*
  using *Diff-iff b-neq-a singletonD assms*
  by *auto*
ultimately show *False*
  by *simp*
qed

lemma *cond-winner-unique*:
fixes
  A :: $'a$ *set* **and**
  p :: $('a, 'v)$ *Profile* **and**
  a :: $'a$
assumes *condorcet-winner V A p a*
shows $\{a' \in A.$ *condorcet-winner V A p a'*$\} = \{a\}$
proof (*safe*)
fix a' :: $'a$
assume *condorcet-winner V A p a'*
thus $a' = a$
  using *assms cond-winner-unique-eq*
  by *metis*
next
show $a \in A$
  using *assms*
  unfolding *condorcet-winner.simps*
  by (*metis* (*no-types*))
next
show *condorcet-winner V A p a*
  using *assms*
  by *presburger*
qed

lemma *cond-winner-unique-2*:
fixes
  V :: $'v$ *set* **and**
  A :: $'a$ *set* **and**
  p :: $('a, 'v)$ *Profile* **and**
  a :: $'a$ **and**
  b :: $'a$
assumes
  condorcet-winner V A p a **and**
  $b \neq a$
shows ¬ *condorcet-winner V A p b*
using *cond-winner-unique-eq assms*
by *metis*

1.4.7 Limited Profile

This function restricts a profile p to a set A of alternatives and a set V of voters s.t. voters outside of V do not have any preferences or do not cast a vote. This keeps all of A's preferences.

fun *limit-profile* :: *'a set* ⇒ (*'a*, *'v*) *Profile* ⇒ (*'a*, *'v*) *Profile* **where**
 limit-profile A p = (λ *v*. *limit A* (*p v*))

lemma *limit-prof-trans*:
 fixes
  A :: *'a set* **and**
  B :: *'a set* **and**
  C :: *'a set* **and**
  p :: (*'a*, *'v*) *Profile*
 assumes
  B ⊆ *A* **and**
  C ⊆ *B*
 shows *limit-profile C p* = *limit-profile C* (*limit-profile B p*)
 using *assms*
 by *auto*

lemma *limit-profile-sound*:
 fixes
  A :: *'a set* **and**
  B :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile*
 assumes
  profile: *profile V B p* **and**
  subset: *A* ⊆ *B*
 shows *profile V A* (*limit-profile A p*)
proof −
 have ∀ *v* ∈ *V*. *linear-order-on A* (*limit A* (*p v*))
  using *profile subset limit-presv-lin-ord*
  unfolding *profile-def*
  by *metis*
 hence ∀ *v* ∈ *V*. *linear-order-on A* ((*limit-profile A p*) *v*)
  by *simp*
 thus *?thesis*
  unfolding *profile-def*
  by *simp*
qed

1.4.8 Lifting Property

definition *equiv-prof-except-a* :: *'v set* ⇒ *'a set* ⇒ (*'a*, *'v*) *Profile* ⇒
  (*'a*, *'v*) *Profile* ⇒ *'a* ⇒ *bool* **where**
 equiv-prof-except-a V A p p' a ≡
  profile V A p ∧ *profile V A p'* ∧ *a* ∈ *A* ∧

$(\forall\ v \in V.\ \textit{equiv-rel-except-a}\ A\ (p\ v)\ (p'\ v)\ a)$

An alternative gets lifted from one profile to another iff its ranking increases in at least one ballot, and nothing else changes.

definition *lifted* :: $'v\ set \Rightarrow\ 'a\ set \Rightarrow ('a,\ 'v)\ \textit{Profile} \Rightarrow ('a,\ 'v)\ \textit{Profile} \Rightarrow\ 'a \Rightarrow$ *bool* **where**
lifted $V\ A\ p\ p'\ a \equiv$
finite-profile $V\ A\ p \land$ *finite-profile* $V\ A\ p' \land a \in A$
$\land (\forall\ v \in V.\ \neg\ \textit{Preference-Relation.lifted}\ A\ (p\ v)\ (p'\ v)\ a \longrightarrow (p\ v) = (p'\ v))$
$\land (\exists\ v \in V.\ \textit{Preference-Relation.lifted}\ A\ (p\ v)\ (p'\ v)\ a)$

lemma *lifted-imp-equiv-prof-except-a*:
fixes
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a,\ 'v)$ *Profile* **and**
p' :: $('a,\ 'v)$ *Profile* **and**
a :: $'a$
assumes *lifted* $V\ A\ p\ p'\ a$
shows *equiv-prof-except-a* $V\ A\ p\ p'\ a$
proof (*unfold equiv-prof-except-a-def*, *safe*)
from *assms*
show *profile* $V\ A\ p$
unfolding *lifted-def*
by *metis*
next
from *assms*
show *profile* $V\ A\ p'$
unfolding *lifted-def*
by *metis*
next
from *assms*
show $a \in A$
unfolding *lifted-def*
by *metis*
next
fix v :: $'v$
assume $v \in V$
with *assms*
show *equiv-rel-except-a* $A\ (p\ v)\ (p'\ v)\ a$
using *lifted-imp-equiv-rel-except-a trivial-equiv-rel*
unfolding *lifted-def profile-def*
by (*metis* (*no-types*))
qed

lemma *negl-diff-imp-eq-limit-prof*:
fixes
A :: $'a\ set$ **and**

A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
p' :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
change: *equiv-prof-except-a* V A' p q a **and**
subset: $A \subseteq A'$ **and**
not-in-A: $a \notin A$
shows $\forall\ v \in V.$ (*limit-profile* A p) v = (*limit-profile* A q) v
proof (*clarify*)
fix
v :: $'v$
assume $v \in V$
hence *equiv-rel-except-a* A' $(p\ v)$ $(q\ v)$ a
using *change equiv-prof-except-a-def*
by *metis*
hence *limit* A $(p\ v)$ = *limit* A $(q\ v)$
using *not-in-A negl-diff-imp-eq-limit subset*
by *metis*
thus *limit-profile* A p v = *limit-profile* A q v
by *simp*
qed

lemma *limit-prof-eq-or-lifted*:
fixes
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
p' :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
lifted-a: *lifted* V A' p p' a **and**
subset: $A \subseteq A'$
shows ($\forall\ v \in V.$ *limit-profile* A p v = *limit-profile* A p' v) $\vee$
lifted V A (*limit-profile* A p) (*limit-profile* A p') a
proof (*cases*)
assume *a-in-A*: $a \in A$
have $\forall\ v \in V.$ (*Preference-Relation.lifted* A' $(p\ v)$ $(p'\ v)$ a $\vee$ $(p\ v) = (p'\ v)$)
using *lifted-a*
unfolding *lifted-def*
by *metis*
hence *one*:
$\forall\ v \in V.$
(*Preference-Relation.lifted* A (*limit* A $(p\ v)$) (*limit* A $(p'\ v)$) a $\vee$
(*limit* A $(p\ v)$) = (*limit* A $(p'\ v)$))
using *limit-lifted-imp-eq-or-lifted subset*
by *metis*

```
  thus ?thesis
  proof (cases)
    assume ∀ v ∈ V. (limit A (p v)) = (limit A (p′ v))
    thus ?thesis
      by simp
  next
    assume forall-limit-p-q:
      ¬ (∀ v ∈ V. (limit A (p v)) = (limit A (p′ v)))
    let ?p = limit-profile A p
    let ?q = limit-profile A p′
    have profile V A ?p ∧ profile V A ?q
      using lifted-a limit-profile-sound subset
      unfolding lifted-def
      by metis
    moreover have
      ∃ v ∈ V. Preference-Relation.lifted A (?p v) (?q v) a
      using forall-limit-p-q lifted-a limit-profile.simps one
      unfolding lifted-def
      by (metis (no-types, lifting))
    moreover have
     ∀ v ∈ V. (¬ Preference-Relation.lifted A (?p v) (?q v) a) ⟶ (?p v) = (?q v)
      using lifted-a limit-profile.simps one
      unfolding lifted-def
      by metis
    ultimately have lifted V A ?p ?q a
      using a-in-A lifted-a rev-finite-subset subset
      unfolding lifted-def
      by (metis (no-types, lifting))
    thus ?thesis
      by simp
  qed
next
  assume a ∉ A
  thus ?thesis
    using lifted-a negl-diff-imp-eq-limit-prof subset lifted-imp-equiv-prof-except-a
    by metis
qed

end
```

1.5 Social Choice Result

```
theory Social-Choice-Result
  imports Result
begin
```

1.5.1 Social Choice Result

A social choice result contains three sets of alternatives: elected, rejected, and deferred alternatives.

fun *well-formed-$\mathcal{SCF}$* :: *'a set* $\Rightarrow$ *'a Result* $\Rightarrow$ *bool* **where**
well-formed-$\mathcal{SCF}$ A res = (*disjoint3 res* $\wedge$ *set-equals-partition A res*)

fun *limit-set-$\mathcal{SCF}$* :: *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **where**
limit-set-$\mathcal{SCF}$ A r = $A \cap r$

1.5.2 Auxiliary Lemmas

lemma *result-imp-rej*:
fixes
A :: *'a set* **and**
e :: *'a set* **and**
r :: *'a set* **and**
d :: *'a set*
assumes *well-formed-$\mathcal{SCF}$ A* (e, r, d)
shows $A - (e \cup d) = r$
proof (*safe*)
fix a :: *'a*
assume
$a \in A$ **and**
$a \notin r$ **and**
$a \notin d$
moreover have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
using *assms*
by *simp*
ultimately show $a \in e$
by *blast*
next
fix a :: *'a*
assume $a \in r$
moreover have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
using *assms*
by *simp*
ultimately show $a \in A$
by *blast*
next
fix a :: *'a*
assume
$a \in r$ **and**
$a \in e$
moreover have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
using *assms*

by *simp*
ultimately show *False*
by *auto*
next
fix $a :: {}'a$
assume
$a \in r$ **and**
$a \in d$
moreover have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
using *assms*
by *simp*
ultimately show *False*
by *blast*
qed

lemma *result-count*:
fixes
$A :: {}'a\ set$ **and**
$e :: {}'a\ set$ **and**
$r :: {}'a\ set$ **and**
$d :: {}'a\ set$
assumes
wf-result: *well-formed-$\mathcal{SCF}$* $A\ (e,\ r,\ d)$ **and**
fin-A: *finite* A
shows $card\ A = card\ e + card\ r + card\ d$
proof −
have $e \cup r \cup d = A$
using *wf-result*
by *simp*
moreover have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\})$
using *wf-result*
by *simp*
ultimately show *?thesis*
using *fin-A Int-Un-distrib2 finite-Un card-Un-disjoint sup-bot.right-neutral*
by *metis*
qed

lemma *defer-subset*:
fixes
$A :: {}'a\ set$ **and**
$r :: {}'a\ Result$
assumes *well-formed-$\mathcal{SCF}$* $A\ r$
shows $defer\text{-}r\ r \subseteq A$
proof (*safe*)
fix $a :: {}'a$
assume $a \in defer\text{-}r\ r$
moreover obtain
$f :: {}'a\ Result \Rightarrow {}'a\ set \Rightarrow {}'a\ set$ **and**

g :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Result* **where**
$A = f\ r\ A \wedge r = g\ r\ A \wedge$ *disjoint3* $(g\ r\ A) \wedge$ *set-equals-partition* $(f\ r\ A)\ (g\ r\ A)$
using *assms*
by *simp*
moreover have $\forall\ p.\ \exists\ e\ r\ d.$ *set-equals-partition* $A\ p \longrightarrow (e,\ r,\ d) = p \wedge e \cup r \cup d = A$
by *simp*
ultimately show $a \in A$
using *UnCI snd-conv*
by *metis*
qed

lemma *elect-subset*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Result*
assumes *well-formed-$\mathcal{SCF}$* $A\ r$
shows *elect-r* $r \subseteq A$
proof (*safe*)
fix a :: $'a$
assume $a \in$ *elect-r* r
moreover obtain
f :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* **and**
g :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Result* **where**
$A = f\ r\ A \wedge r = g\ r\ A \wedge$ *disjoint3* $(g\ r\ A) \wedge$ *set-equals-partition* $(f\ r\ A)\ (g\ r\ A)$
using *assms*
by *simp*
moreover have
$\forall\ p.\ \exists\ e\ r\ d.$ *set-equals-partition* $A\ p \longrightarrow (e,\ r,\ d) = p \wedge e \cup r \cup d = A$
by *simp*
ultimately show $a \in A$
using *UnCI assms fst-conv*
by *metis*
qed

lemma *reject-subset*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Result*
assumes *well-formed-$\mathcal{SCF}$* $A\ r$
shows *reject-r* $r \subseteq A$
proof (*safe*)
fix a :: $'a$
assume $a \in$ *reject-r* r
moreover obtain
f :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* **and**
g :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Result* **where**
$A = f\ r\ A \wedge r = g\ r\ A \wedge$ *disjoint3* $(g\ r\ A) \wedge$ *set-equals-partition* $(f\ r\ A)\ (g\ r\ A)$
using *assms*

by *simp*
moreover have
$\forall\ p.\ \exists\ e\ r\ d.\ \textit{set-equals-partition}\ A\ p \longrightarrow (e,\ r,\ d) = p \wedge e \cup r \cup d = A$
by *simp*
ultimately show $a \in A$
using *UnCI assms fst-conv snd-conv disjoint3.cases*
by *metis*
qed

end

1.6 Social Welfare Result

theory *Social-Welfare-Result*
imports *Result*
Preference-Relation
begin

1.6.1 Social Welfare Result

A social welfare result contains three sets of relations: elected, rejected, and deferred A well-formed social welfare result consists only of linear orders on the alternatives.

fun *well-formed-$\mathcal{SWF}$* :: $'a\ set \Rightarrow ('a\ \textit{Preference-Relation})\ \textit{Result} \Rightarrow \textit{bool}$ **where**
well-formed-$\mathcal{SWF}$ $A\ res = (\textit{disjoint3}\ res\ \wedge$
$\textit{set-equals-partition}\ \{r.\ \textit{linear-order-on}\ A\ r\}\ res)$

fun *limit-set-$\mathcal{SWF}$* ::
$'a\ set \Rightarrow ('a\ \textit{Preference-Relation})\ set \Rightarrow ('a\ \textit{Preference-Relation})\ set$ **where**
limit-set-$\mathcal{SWF}$ $A\ res = \{\textit{limit}\ A\ r \mid r.\ r \in res \wedge \textit{linear-order-on}\ A\ (\textit{limit}\ A\ r)\}$

end

1.7 Specific Electoral Result Types

theory *Result-Interpretations*
imports *Social-Choice-Result*
Social-Welfare-Result
Collections.Locale-Code
begin

Interpretations of the result locale are placed inside a Locale-Code block in order to enable code generation of later definitions in the locale. Those definitions need to be added via a Locale-Code block as well.

setup *Locale-Code.open-block*

Results from social choice functions ($\mathcal{SCF}s$), for the purpose of composability and modularity given as three sets of (potentially tied) alternatives. See `Social_Choice_Result.thy` for details.

global-interpretation $\mathcal{SCF}$*-result*:
result well-formed-$\mathcal{SCF}$ *limit-set-*$\mathcal{SCF}$
proof (*unfold-locales*, *simp*) **qed**

Results from committee functions. TODO: What is the semantics?

global-interpretation *committee-result*:
result λ *A r*. *set-equals-partition* (*Pow A*) *r* $\wedge$ *disjoint3 r* λ *A rs*. $\{r \cap A \mid r.\ r \in rs\}$
proof (*unfold-locales*, *safe*, *force*) **qed**

Results from social welfare functions ($\mathcal{SWF}s$), for the purpose of composability and modularity given as three linear orders over the alternatives. See `Social_Welfare_Result.thy` for details.

global-interpretation $\mathcal{SWF}$*-result*:
result well-formed-$\mathcal{SWF}$ *limit-set-*$\mathcal{SWF}$
proof (*unfold-locales*, *safe*)
fix
A :: *'a set* **and**
e :: (*'a Preference-Relation*) *set* **and**
r :: (*'a Preference-Relation*) *set* **and**
d :: (*'a Preference-Relation*) *set*
assume
partition: *set-equals-partition* (*limit-set-*$\mathcal{SWF}$ *A UNIV*) (*e*, *r*, *d*) **and**
disj: *disjoint3* (*e*, *r*, *d*)
have *limit-set-*$\mathcal{SWF}$ *A UNIV* =
$\{limit\ A\ r' \mid r'.\ r' \in UNIV \wedge linear\text{-}order\text{-}on\ A\ (limit\ A\ r')\}$
by *simp*
also have ... = $\{limit\ A\ r' \mid r'.\ r' \in UNIV\} \cap$
$\{limit\ A\ r' \mid r'.\ linear\text{-}order\text{-}on\ A\ (limit\ A\ r')\}$
by *blast*
also have ... = $\{limit\ A\ r' \mid r'.\ linear\text{-}order\text{-}on\ A\ (limit\ A\ r')\}$
by *blast*
also have ... = $\{r'.\ linear\text{-}order\text{-}on\ A\ r'\}$
proof (*safe*)
fix *r'* :: *'a Preference-Relation*
assume *lin-ord*: *linear-order-on A r'*
hence $\forall$ *a b*. $(a, b) \in r' \longrightarrow (a, b) \in limit\ A\ r'$
unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
by *force*
hence $r' \subseteq limit\ A\ r'$
by *slow*
moreover have $limit\ A\ r' \subseteq r'$
by *auto*

ultimately have $r' = \mathit{limit}\ A\ r'$
by *safe*
thus $\exists\ x.\ r' = \mathit{limit}\ A\ x \wedge \mathit{linear\text{-}order\text{-}on}\ A\ (\mathit{limit}\ A\ x)$
using *lin-ord*
by *metis*
qed
thus $\mathit{well\text{-}formed\text{-}}\mathcal{SWF}\ A\ (e,\ r,\ d)$
using *partition disj*
by *simp*
qed

setup *Locale-Code.close-block*

end

1.8 Function Symmetry Properties

theory *Symmetry-Of-Functions*
imports *HOL−Algebra.Group-Action*
HOL−Algebra.Generated-Groups
begin

1.8.1 Functions

type-synonym $('x,\ 'y)\ \mathit{binary\text{-}fun} = 'x \Rightarrow 'y \Rightarrow 'y$

fun *extensional-continuation* :: $('x \Rightarrow 'y) \Rightarrow 'x\ \mathit{set} \Rightarrow ('x \Rightarrow 'y)$ **where**
$\mathit{extensional\text{-}continuation}\ f\ s = (\lambda\ x.\ \mathit{if}\ (x \in s)\ \mathit{then}\ (f\ x)\ \mathit{else}\ \mathit{undefined})$

fun *preimg* :: $('x \Rightarrow 'y) \Rightarrow 'x\ \mathit{set} \Rightarrow 'y \Rightarrow 'x\ \mathit{set}$ **where**
$\mathit{preimg}\ f\ s\ x = \{x' \in s.\ f\ x' = x\}$

Relations

fun *restr-rel* :: $'x\ \mathit{rel} \Rightarrow 'x\ \mathit{set} \Rightarrow 'x\ \mathit{set} \Rightarrow 'x\ \mathit{rel}$ **where**
$\mathit{restr\text{-}rel}\ r\ s\ s' = r \cap s \times s'$

fun *closed-under-restr-rel* :: $'x\ \mathit{rel} \Rightarrow 'x\ \mathit{set} \Rightarrow 'x\ \mathit{set} \Rightarrow \mathit{bool}$ **where**
$\mathit{closed\text{-}under\text{-}restr\text{-}rel}\ r\ s\ t = ((\mathit{restr\text{-}rel}\ r\ t\ s)\ \text{``}\ t \subseteq t)$

fun *rel-induced-by-action* :: $'x\ \mathit{set} \Rightarrow 'y\ \mathit{set} \Rightarrow ('x,\ 'y)\ \mathit{binary\text{-}fun} \Rightarrow 'y\ \mathit{rel}$ **where**
$\mathit{rel\text{-}induced\text{-}by\text{-}action}\ s\ t\ \varphi = \{(y,\ y') \in t \times t.\ \exists\ x \in s.\ \varphi\ x\ y = y'\}$

fun *product-rel* :: $'x\ \mathit{rel} \Rightarrow ('x * 'x)\ \mathit{rel}$ **where**
$\mathit{product\text{-}rel}\ r = \{(p,\ p').\ (\mathit{fst}\ p,\ \mathit{fst}\ p') \in r \wedge (\mathit{snd}\ p,\ \mathit{snd}\ p') \in r\}$

fun *equivariance-rel* :: $'x\ \mathit{set} \Rightarrow 'y\ \mathit{set} \Rightarrow ('x,'y)\ \mathit{binary\text{-}fun} \Rightarrow ('y * 'y)\ \mathit{rel}$ **where**

equivariance-rel s t φ = $\{((u, v), (x, y)).\ (u, v) \in t \times t \wedge (\exists\ z \in s.\ x = \varphi\ z\ u \wedge y = \varphi\ z\ v)\}$

fun *set-closed-under-rel* :: $'x$ *set* $\Rightarrow$ $'x$ *rel* $\Rightarrow$ *bool* **where**
set-closed-under-rel s r = $(\forall\ x\ y.\ (x, y) \in r \longrightarrow x \in s \longrightarrow y \in s)$

fun *singleton-set-system* :: $'x$ *set* $\Rightarrow$ $'x$ *set set* **where**
singleton-set-system s = $\{\{x\} \mid x.\ x \in s\}$

fun *set-action* :: $('x, 'r)$ *binary-fun* $\Rightarrow$ $('x, 'r$ *set*$)$ *binary-fun* **where**
set-action ψ x = *image* $(\psi\ x)$

1.8.2 Invariance and Equivariance

Invariance and equivariance are symmetry properties of functions: Invariance means that related preimages have identical images and equivariance denotes consistent changes.

datatype $('x, 'y)$ *property* =
Invariance $'x$ *rel* |
Equivariance $'x$ *set* $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set*

fun *satisfies* :: $('x \Rightarrow 'y) \Rightarrow ('x, 'y)$ *property* $\Rightarrow$ *bool* **where**
satisfies f (*Invariance r*) = $(\forall\ x.\ \forall\ y.\ (x, y) \in r \longrightarrow f\ x = f\ y)$ |
satisfies f (*Equivariance s* τ) = $(\forall\ (\varphi, \psi) \in \tau.\ \forall\ x \in s.\ \varphi\ x \in s \longrightarrow f\ (\varphi\ x) = \psi\ (f\ x))$

definition *equivar-ind-by-act* :: $'z$ *set* $\Rightarrow$ $'x$ *set* $\Rightarrow$ $('z, 'x)$ *binary-fun*
$\Rightarrow$ $('z, 'y)$ *binary-fun* $\Rightarrow$ $('x,'y)$ *property* **where**
equivar-ind-by-act s t φ ψ = *Equivariance t* $\{(\varphi\ x, \psi\ x) \mid x.\ x \in s\}$

1.8.3 Auxiliary Lemmas

lemma *inj-imp-inj-on-set-system*:
fixes $f :: 'x \Rightarrow 'y$
assumes *inj f*
shows *inj* $(\lambda\ s.\ \{f\ `\ x \mid x.\ x \in s\})$
proof (*unfold inj-def*, *safe*)
fix
$s :: 'x$ *set set* **and**
$t :: 'x$ *set set* **and**
$x :: 'x$ *set*
assume *f-elem-s-eq-f-elem-t*: $\{f\ `\ x' \mid x'.\ x' \in s\} = \{f\ `\ x' \mid x'.\ x' \in t\}$
then obtain $y :: 'x$ *set* **where**
$f\ `\ y = f\ `\ x$
by *metis*
hence *y-eq-x*: $y = x$
using *image-inv-f-f assms*
by *metis*
moreover have

$x \in t \longrightarrow f \text{ ` } x \in \{f \text{ ` } x' \mid x'. x' \in s\}$ **and**
$x \in s \longrightarrow f \text{ ` } x \in \{f \text{ ` } x' \mid x'. x' \in t\}$
using *f-elem-s-eq-f-elem-t*
by *auto*
ultimately have $x \in t \longrightarrow y \in s$ **and** $x \in s \longrightarrow y \in t$
using *assms*
by (*simp add*: *inj-image-eq-iff*, *simp add*: *inj-image-eq-iff*)
thus $x \in t \Longrightarrow x \in s$ **and** $x \in s \Longrightarrow x \in t$
using *y-eq-x*
by (*simp*, *simp*)
qed

lemma *inj-and-surj-imp-surj-on-set-system*:
fixes $f :: 'x \Rightarrow 'y$
assumes
inj f **and**
surj f
shows *surj* $(\lambda\ s.\ \{f \text{ ` } x \mid x.\ x \in s\})$
proof (*unfold surj-def*, *safe*)
fix $s :: 'y\ set\ set$
have $\forall\ x.\ f \text{ ` } (\textit{the-inv}\ f) \text{ ` } x = x$
using *image-f-inv-f assms surj-imp-inv-eq the-inv-f-f*
by (*metis* (*no-types*, *opaque-lifting*))
hence $s = \{f \text{ ` } (\textit{the-inv}\ f) \text{ ` } x \mid x.\ x \in s\}$
by *simp*
also have $\{f \text{ ` } (\textit{the-inv}\ f) \text{ ` } x \mid x.\ x \in s\} =$
$\{f \text{ ` } x \mid x.\ x \in \{(\textit{the-inv}\ f) \text{ ` } x \mid x.\ x \in s\}\}$
by *blast*
finally show $\exists\ t.\ s = \{f \text{ ` } x \mid x.\ x \in t\}$
by *blast*
qed

lemma *bij-imp-bij-on-set-system*:
fixes $f :: 'x \Rightarrow 'y$
assumes *bij f*
shows *bij* $(\lambda\ s.\ \{f \text{ ` } x \mid x.\ x \in s\})$
proof (*unfold bij-def*)
have *range f* = *UNIV*
using *assms*
unfolding *bij-betw-def*
by *safe*
moreover have *inj f*
using *assms*
unfolding *bij-betw-def*
by *safe*
ultimately show *inj* $(\lambda\ s.\ \{f \text{ ` } x \mid x.\ x \in s\}) \wedge$ *surj* $(\lambda\ s.\ \{f \text{ ` } x \mid x.\ x \in s\})$
using *inj-imp-inj-on-set-system*
by (*simp add*: *inj-and-surj-imp-surj-on-set-system*)
qed

lemma *un-left-inv-singleton-set-system*: $\bigcup \circ$ *singleton-set-system* = *id*
proof
  fix s :: $'x$ *set*
  have ($\bigcup \circ$ *singleton-set-system*) s = $\{x.\ \exists\ s' \in$ *singleton-set-system* $s.\ x \in s'\}$
    by *auto*
  also have $\{x.\ \exists\ s' \in$ *singleton-set-system* $s.\ x \in s'\} = \{x.\ \{x\} \in$ *singleton-set-system* $s\}$
    by *auto*
  also have $\{x.\ \{x\} \in$ *singleton-set-system* $s\} = \{x.\ \{x\} \in \{\{x\} \mid x.\ x \in s\}\}$
    by *simp*
  finally show ($\bigcup \circ$ *singleton-set-system*) s = *id* s
    by *simp*
qed

lemma *the-inv-comp*:
  fixes
    f :: $'y \Rightarrow 'z$ **and**
    g :: $'x \Rightarrow 'y$ **and**
    s :: $'x$ *set* **and**
    t :: $'y$ *set* **and**
    u :: $'z$ *set* **and**
    x :: $'z$
  assumes
    bij-betw f t u **and**
    bij-betw g s t **and**
    $x \in u$
  shows *the-inv-into* s ($f \circ g$) x = ((*the-inv-into* s g) $\circ$ (*the-inv-into* t f)) x
proof (*clarsimp*)
  have *el-Y*: *the-inv-into* t f $x \in t$
    using *assms bij-betw-apply bij-betw-the-inv-into*
    by *metis*
  hence g (*the-inv-into* s g (*the-inv-into* t f x)) = *the-inv-into* t f x
    using *assms f-the-inv-into-f-bij-betw*
    by *metis*
  moreover have f (*the-inv-into* t f x) = x
    using *el-Y assms f-the-inv-into-f-bij-betw*
    by *metis*
  ultimately have ($f \circ g$) (*the-inv-into* s g (*the-inv-into* t f x)) = x
    by *simp*
  hence *the-inv-into* s ($f \circ g$) x =
      the-inv-into s ($f \circ g$) (($f \circ g$) (*the-inv-into* s g (*the-inv-into* t f x)))
    by *presburger*
  also have
    the-inv-into s ($f \circ g$) (($f \circ g$) (*the-inv-into* s g (*the-inv-into* t f x))) =
      the-inv-into s g (*the-inv-into* t f x)
   using *assms bij-betw-apply bij-betw-imp-inj-on bij-betw-the-inv-into bij-betw-trans the-inv-into-f-eq*
    by (*metis* (*no-types*, *lifting*))

finally show *the-inv-into* s $(f \circ g)$ x = *the-inv-into* s g (*the-inv-into* t f x)
by *blast*
qed

lemma *preimg-comp*:
fixes
f :: $'x \Rightarrow 'y$ **and**
g :: $'x \Rightarrow 'x$ **and**
s :: $'x$ *set* **and**
x :: $'y$
shows *preimg* f $(g$ ‘ $s)$ x = g ‘ *preimg* $(f \circ g)$ s x
proof (*safe*)
fix y :: $'x$
assume $y \in$ *preimg* f $(g$ ‘ $s)$ x
then obtain z :: $'x$ **where**
$g\ z = y$ **and**
$z \in$ *preimg* $(f \circ g)$ s x
unfolding *comp-def*
by *fastforce*
thus $y \in g$ ‘ *preimg* $(f \circ g)$ s x
by *blast*
next
fix y :: $'x$
assume $y \in$ *preimg* $(f \circ g)$ s x
thus $g\ y \in$ *preimg* f $(g$ ‘ $s)$ x
by *simp*
qed

1.8.4 Rewrite Rules

theorem *rewrite-invar-as-equivar*:
fixes
f :: $'x \Rightarrow 'y$ **and**
s :: $'x$ *set* **and**
t :: $'z$ *set* **and**
φ :: $('z, 'x)$ *binary-fun*
shows *satisfies* f (*Invariance* (*rel-induced-by-action* t s φ)) =
satisfies f (*equivar-ind-by-act* t s φ $(\lambda\ g.\ id)$)
proof (*unfold equivar-ind-by-act-def*, *simp*, *safe*)
fix
x :: $'x$ **and**
y :: $'z$
assume
$x \in s$ **and**
$y \in t$ **and**
$\varphi\ y\ x \in s$
thus
$(\forall\ x'\ y'.\ x' \in s \land y' \in s \land (\exists\ z \in t.\ \varphi\ z\ x' = y') \longrightarrow f\ x' = f\ y')$
$\Longrightarrow (f\ (\varphi\ y\ x) = id\ (f\ x))$ **and**

$(\forall\ x'\ y'.\ (\exists\ z.\ x' = \varphi\ z \wedge y' = id \wedge z \in t) \longrightarrow$
$(\forall\ z \in s.\ x'\ z \in s \longrightarrow f\ (x'\ z) = y'\ (f\ z)))$
$\Longrightarrow (f\ x = f\ (\varphi\ y\ x))$
unfolding *id-def*
by (*metis*, *metis*)
qed

lemma *rewrite-invar-ind-by-act*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$s :: 'z\ set$ **and**
$t :: 'x\ set$ **and**
$\varphi :: ('z, 'x)\ binary\text{-}fun$
shows *satisfies* f (*Invariance* (*rel-induced-by-action* $s\ t\ \varphi$)) $=$
$(\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y \in t \longrightarrow f\ y = f\ (\varphi\ x\ y))$
proof (*safe*)
fix
$y :: 'x$ **and**
$x :: 'z$
assume
satisfies f (*Invariance* (*rel-induced-by-action* $s\ t\ \varphi$)) **and**
$y \in t$ **and**
$x \in s$ **and**
$\varphi\ x\ y \in t$
moreover from *this* **have** $(y, \varphi\ x\ y) \in$ *rel-induced-by-action* $s\ t\ \varphi$
unfolding *rel-induced-by-action.simps*
by *blast*
ultimately show $f\ y = f\ (\varphi\ x\ y)$
by *simp*
next
assume $\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y \in t \longrightarrow f\ y = f\ (\varphi\ x\ y)$
moreover have
$\forall\ (x, y) \in$ *rel-induced-by-action* $s\ t\ \varphi.\ x \in t \wedge y \in t \wedge (\exists\ z \in s.\ y = \varphi\ z\ x)$
by *auto*
ultimately show *satisfies* f (*Invariance* (*rel-induced-by-action* $s\ t\ \varphi$))
by *auto*
qed

lemma *rewrite-equivar-ind-by-act*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$s :: 'z\ set$ **and**
$t :: 'x\ set$ **and**
$\varphi :: ('z, 'x)\ binary\text{-}fun$ **and**
$\psi :: ('z, 'y)\ binary\text{-}fun$
shows *satisfies* f (*equivar-ind-by-act* $s\ t\ \varphi\ \psi$) $=$
$(\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y \in t \longrightarrow f\ (\varphi\ x\ y) = \psi\ x\ (f\ y))$
unfolding *equivar-ind-by-act-def*
by *auto*

lemma *rewrite-group-act-img*:
  fixes
    $m :: 'x\ monoid$ **and**
    $s :: 'y\ set$ **and**
    $\varphi :: ('x, 'y)\ binary\text{-}fun$ **and**
    $t :: 'y\ set$ **and**
    $x :: 'x$ **and**
    $y :: 'x$
  assumes
    $t \subseteq s$ **and**
    $x \in carrier\ m$ **and**
    $y \in carrier\ m$ **and**
    $group\text{-}action\ m\ s\ \varphi$
  shows $\varphi\ (x \otimes_{m} y)$ ‘ $t = \varphi\ x$ ‘ $\varphi\ y$ ‘ t
proof (*safe*)
  fix $z :: 'y$
  assume *z-in-t*: $z \in t$
  hence $\varphi\ (x \otimes_{m} y)\ z = \varphi\ x\ (\varphi\ y\ z)$
    using *assms group-action.composition-rule*[*of m s*]
    by *blast*
  thus
    $\varphi\ (x \otimes_{m} y)\ z \in \varphi\ x$ ‘ $\varphi\ y$ ‘ t **and**
    $\varphi\ x\ (\varphi\ y\ z) \in \varphi\ (x \otimes_{m} y)$ ‘ t
    using *z-in-t*
    by (*blast*, *force*)
qed

lemma *rewrite-carrier*: $carrier\ (BijGroup\ UNIV) = \{f'.\ bij\ f'\}$
  unfolding *BijGroup-def Bij-def*
  by *simp*

lemma *universal-set-carrier-imp-bij-group*:
  fixes $f :: 'a \Rightarrow 'a$
  assumes $f \in carrier\ (BijGroup\ UNIV)$
  shows $bij\ f$
  using *rewrite-carrier assms*
  by *blast*

lemma *rewrite-sym-group*:
  fixes
    $f :: 'a \Rightarrow 'a$ **and**
    $g :: 'a \Rightarrow 'a$ **and**
    $s :: 'a\ set$
  assumes
    f-carrier: $f \in carrier\ (BijGroup\ s)$ **and**
    g-carrier: $g \in carrier\ (BijGroup\ s)$
  shows
    rewrite-mult: $f \otimes_{BijGroup\ s} g = extensional\text{-}continuation\ (f \circ g)\ s$ **and**

rewrite-mult-univ: $s = UNIV \longrightarrow f \otimes_{BijGroup\ s} g = f \circ g$
proof −
show $f \otimes_{BijGroup\ s} g = extensional\text{-}continuation\ (f \circ g)\ s$
using *f-carrier g-carrier*
unfolding *BijGroup-def compose-def comp-def restrict-def*
by *simp*
next
show $s = UNIV \longrightarrow f \otimes_{BijGroup\ s} g = f \circ g$
using *f-carrier g-carrier*
unfolding *BijGroup-def compose-def comp-def restrict-def*
by *fastforce*
qed

lemma *simp-extensional-univ*:
fixes $f :: 'a \Rightarrow 'b$
shows *extensional-continuation f UNIV* $=$ *f*
unfolding *If-def*
by *simp*

lemma *extensional-continuation-subset*:
fixes
$f :: 'a \Rightarrow 'b$ **and**
$s :: 'a\ set$ **and**
$t :: 'a\ set$ **and**
$x :: 'a$
assumes
$t \subseteq s$ **and**
$x \in t$
shows *extensional-continuation f s x* $=$ *extensional-continuation f t x*
using *assms*
unfolding *subset-iff*
by *simp*

lemma *rel-ind-by-coinciding-action-on-subset-eq-restr*:
fixes
$\varphi :: ('a, 'b)\ binary\text{-}fun$ **and**
$\psi :: ('a, 'b)\ binary\text{-}fun$ **and**
$s :: 'a\ set$ **and**
$t :: 'b\ set$ **and**
$u :: 'b\ set$
assumes
$u \subseteq t$ **and**
$\forall\ x \in s.\ \forall\ y \in u.\ \psi\ x\ y = \varphi\ x\ y$
shows *rel-induced-by-action s u* $\psi =$ *Restr* (*rel-induced-by-action s t* φ) *u*
proof (*unfold rel-induced-by-action.simps*)
have $\{(x, y).\ (x, y) \in u \times u \land (\exists\ z \in s.\ \psi\ z\ x = y)\}$
$= \{(x, y).\ (x, y) \in u \times u \land (\exists\ z \in s.\ \varphi\ z\ x = y)\}$
using *assms*
by *auto*

also have ... = *Restr* $\{(x, y). (x, y) \in t \times t \wedge (\exists\ z \in s.\ \varphi\ z\ x = y)\}$ *u*
  using *assms*
  by *blast*
finally show $\{(x, y). (x, y) \in u \times u \wedge (\exists\ z \in s.\ \psi\ z\ x = y)\} =$
    Restr $\{(x, y). (x, y) \in t \times t \wedge (\exists\ z \in s.\ \varphi\ z\ x = y)\}$ *u*
  by *simp*
qed

lemma *coinciding-actions-ind-equal-rel*:
  fixes
    s :: *'x set* **and**
    t :: *'y set* **and**
    φ :: *('x, 'y) binary-fun* **and**
    ψ :: *('x, 'y) binary-fun*
  assumes $\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y = \psi\ x\ y$
  shows *rel-induced-by-action s t* φ = *rel-induced-by-action s t* ψ
  unfolding *extensional-continuation.simps*
  using *assms*
  by *auto*

1.8.5 Group Actions

lemma *const-id-is-group-act*:
  fixes *m* :: *'x monoid*
  assumes *group m*
  shows *group-action m UNIV* (λ *x. id*)
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*, *safe*)
  show *group m*
    using *assms*
    by *blast*
next
  show *group* (*BijGroup UNIV*)
    using *group-BijGroup*
    by *metis*
next
  show *id* $\in$ *carrier* (*BijGroup UNIV*)
    unfolding *BijGroup-def Bij-def*
    by *simp*
  thus $id = id \otimes_{BijGroup\ UNIV} id$
    using *rewrite-mult-univ comp-id*
    by *metis*
qed

theorem *group-act-induces-set-group-act*:
  fixes
    m :: *'x monoid* **and**
    s :: *'y set* **and**
    φ :: *('x, 'y) binary-fun*
  defines φ*-img* $\equiv$ (λ *x. extensional-continuation* (*image* (φ *x*)) (*Pow s*))

assumes *group-action m s φ*
shows *group-action m (Pow s) φ-img*
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*, *safe*)
show *group m*
using *assms*
unfolding *group-action-def group-hom-def*
by *simp*
next
show *group (BijGroup (Pow s))*
using *group-BijGroup*
by *metis*
next
{
fix $x :: 'x$
assume *car-x*: $x \in$ *carrier m*
hence *bij-betw (φ x) s s*
using *assms group-action.surj-prop*
unfolding *bij-betw-def*
by (*simp add*: *group-action.inj-prop*)
hence *bij-betw (image (φ x)) (Pow s) (Pow s)*
using *bij-betw-Pow*
by *metis*
moreover have $\forall$ $t \in$ *Pow s*. *φ-img x t* = *image (φ x) t*
unfolding *φ-img-def*
by *simp*
ultimately have *bij-betw (φ-img x) (Pow s) (Pow s)*
using *bij-betw-cong*
by *fastforce*
moreover have *φ-img x* $\in$ *extensional (Pow s)*
unfolding *φ-img-def extensional-def*
by *simp*
ultimately show *φ-img x* $\in$ *carrier (BijGroup (Pow s))*
unfolding *BijGroup-def Bij-def*
by *simp*
}
fix
$x :: 'x$ **and**
$y :: 'x$
note
car-x-el = ‹$x \in$ *carrier m* $\Longrightarrow$ *φ-img x* $\in$ *carrier (BijGroup (Pow s))*› **and**
car-y-el = ‹$y \in$ *carrier m* $\Longrightarrow$ *φ-img y* $\in$ *carrier (BijGroup (Pow s))*›
assume
car-x: $x \in$ *carrier m* **and**
car-y: $y \in$ *carrier m*
hence *car-els*: *φ-img x* $\in$ *carrier (BijGroup (Pow s))* $\wedge$ *φ-img y* $\in$ *carrier (BijGroup (Pow s))*
using *car-x-el car-y-el car-y*
by *blast*
hence *h-closed*: $\forall$ *t*. $t \in$ *Pow s* $\longrightarrow$ *φ-img y t* $\in$ *Pow s*

using *bij-betw-apply Int-Collect partial-object.select-convs*(*1*)
unfolding *BijGroup-def Bij-def*
by *metis*
from *car-els*
have φ*-img* $x \otimes_{\,BijGroup\ (Pow\ s)}$ φ*-img* $y =$
extensional-continuation (φ*-img* $x \circ \varphi$*-img* y) (*Pow* s)
using *rewrite-mult*
by *blast*
moreover have
$\forall$ t. $t \notin$ *Pow* $s \longrightarrow$ *extensional-continuation* (φ*-img* $x \circ \varphi$*-img* y) (*Pow* s) $t =$ *undefined*
by *simp*
moreover have $\forall$ t. $t \notin$ *Pow* $s \longrightarrow \varphi$*-img* ($x \otimes_{\,m}$ y) $t =$ *undefined*
unfolding φ*-img-def*
by *simp*
moreover have
$\forall$ t. $t \in$ *Pow* $s \longrightarrow$ *extensional-continuation* (φ*-img* $x \circ \varphi$*-img* y) (*Pow* s) $t =$ φ x ‘ φ y ‘ t
using *h-closed*
unfolding φ*-img-def*
by *simp*
moreover have $\forall$ t. $t \in$ *Pow* $s \longrightarrow \varphi$*-img* ($x \otimes_{\,m}$ y) $t = \varphi$ x ‘ φ y ‘ t
unfolding φ*-img-def extensional-continuation.simps*
using *rewrite-group-act-img car-x car-y assms PowD*
by *metis*
ultimately have $\forall$ t. φ*-img* ($x \otimes_{\,m}$ y) $t =$ (φ*-img* $x \otimes_{\,BijGroup\ (Pow\ s)}$ φ*-img* y) t
by *metis*
thus φ*-img* ($x \otimes_{\,m}$ y) $= \varphi$*-img* $x \otimes_{\,BijGroup\ (Pow\ s)}$ φ*-img* y
by *blast*
qed

1.8.6 Invariance and Equivariance

It suffices to show invariance under the group action of a generating set of a group to show invariance under the group action of the whole group. For example, it is enough to show invariance under transpositions to show invariance under a complete finite symmetric group.

theorem *invar-generating-system-imp-invar*:
fixes
f :: $'x \Rightarrow$ $'y$ **and**
m :: $'z$ *monoid* **and**
s :: $'z$ *set* **and**
t :: $'x$ *set* **and**
φ :: ($'z$, $'x$) *binary-fun*
assumes
invar: *satisfies f* (*Invariance* (*rel-induced-by-action* s t φ)) **and**
action-φ: *group-action* m t φ **and**

```
    gen: carrier m = generate m s
  shows satisfies f (Invariance (rel-induced-by-action (carrier m) t φ))
proof (unfold satisfies.simps rel-induced-by-action.simps, safe)
  fix
    g :: 'z and
    x :: 'x
  assume
    group-elem: g ∈ carrier m and
    x-in-t: x ∈ t
  interpret interpr-action-φ: group-action m t φ
    using action-φ
    by blast
  have g ∈ generate m s
    using group-elem gen
    by blast
  hence ∀ x ∈ t. f x = f (φ g x)
  proof (induct g rule: generate.induct)
    case one
    hence ∀ x ∈ t. φ 1 m x = x
      using action-φ group-action.id-eq-one restrict-apply
      by metis
    thus ?case
      by simp
  next
    case (incl g)
    hence ∀ x ∈ t. (x, φ g x) ∈ rel-induced-by-action s t φ
      using gen action-φ generate.incl group-action.element-image
      unfolding rel-induced-by-action.simps
      by fastforce
    thus ?case
      using invar
      unfolding satisfies.simps
      by blast
  next
    case (inv g)
    hence ∀ x ∈ t. φ (inv m g) x ∈ t
      using action-φ gen generate.inv group-action.element-image
      by metis
    hence ∀ x ∈ t. f (φ g (φ (inv m g) x)) = f (φ (inv m g) x)
      using gen generate.incl group-action.element-image action-φ
          invar local.inv rewrite-invar-ind-by-act
      by metis
    moreover have ∀ x ∈ t. φ g (φ (inv m g) x) = x
      using action-φ gen generate.incl group.inv-closed group-action.orbit-sym-aux
          group.inv-inv group-hom.axioms(1) interpr-action-φ.group-hom local.inv
      by (metis (full-types))
    ultimately show ?case
      by simp
  next
```

case ($eng\ g_1\ g_2$)
assume
invar$_1$: $\forall\ x \in t.\ f\,x = f\ (\varphi\ g_1\ x)$ **and**
invar$_2$: $\forall\ x \in t.\ f\,x = f\ (\varphi\ g_2\ x)$ **and**
gen$_1$: $g_1 \in generate\ m\ s$ **and**
gen$_2$: $g_2 \in generate\ m\ s$
hence $\forall\ x \in t.\ \varphi\ g_2\ x \in t$
using *gen interpr-action-*φ*.element-image*
by *blast*
hence $\forall\ x \in t.\ f\ (\varphi\ g_1\ (\varphi\ g_2\ x)) = f\ (\varphi\ g_2\ x)$
using *invar*$_1$
by *simp*
moreover have $\forall\ x \in t.\ f\ (\varphi\ g_2\ x) = f\,x$
using *invar*$_2$
by *simp*
moreover have $\forall\ x \in t.\ f\ (\varphi\ (g_1 \otimes_m g_2)\ x) = f\ (\varphi\ g_1\ (\varphi\ g_2\ x))$
using *action-*φ *gen interpr-action-*φ*.composition-rule gen*$_1$ *gen*$_2$
by *simp*
ultimately show *?case*
by *simp*
qed
thus $f\,x = f\ (\varphi\ g\ x)$
using *x-in-t*
by *simp*
qed

lemma *invar-parameterized-fun*:
fixes
$f :: 'x \Rightarrow ('x \Rightarrow 'y)$ **and**
$r :: 'x\ rel$
assumes
param-invar: $\forall\ x.\ satisfies\ (f\,x)\ (Invariance\ r)$ **and**
invar: $satisfies\ f\ (Invariance\ r)$
shows $satisfies\ (\lambda\ x.\ f\,x\,x)\ (Invariance\ r)$
using *invar param-invar*
by *auto*

lemma *invar-under-subset-rel*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$r :: 'x\ rel$
assumes
subset: $r \subseteq rel$ **and**
invar: $satisfies\ f\ (Invariance\ rel)$
shows $satisfies\ f\ (Invariance\ r)$
using *assms*
by *auto*

lemma *equivar-ind-by-act-coincide*:

fixes
  s :: $'x$ *set* **and**
  t :: $'y$ *set* **and**
  f :: $'y \Rightarrow 'z$ **and**
  φ :: $('x, 'y)$ *binary-fun* **and**
  φ' :: $('x, 'y)$ *binary-fun* **and**
  ψ :: $('x, 'z)$ *binary-fun*
assumes $\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y = \varphi'\ x\ y$
shows *satisfies* f (*equivar-ind-by-act* $s\ t\ \varphi\ \psi$) = *satisfies* f (*equivar-ind-by-act* $s\ t\ \varphi'\ \psi$)
using *assms*
unfolding *rewrite-equivar-ind-by-act*
by *simp*

lemma *equivar-under-subset*:
fixes
  f :: $'x \Rightarrow 'y$ **and**
  s :: $'x$ *set* **and**
  t :: $'x$ *set* **and**
  τ :: $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set*
assumes
  satisfies f (*Equivariance* $s\ \tau$) **and**
  $t \subseteq s$
shows *satisfies* f (*Equivariance* $t\ \tau$)
using *assms*
unfolding *satisfies.simps*
by *blast*

lemma *equivar-under-subset'*:
fixes
  f :: $'x \Rightarrow 'y$ **and**
  s :: $'x$ *set* **and**
  τ :: $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set* **and**
  υ :: $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set*
assumes
  satisfies f (*Equivariance* $s\ \tau$) **and**
  $\upsilon \subseteq \tau$
shows *satisfies* f (*Equivariance* $s\ \upsilon$)
using *assms*
unfolding *satisfies.simps*
by *blast*

theorem *group-act-equivar-f-imp-equivar-preimg*:
fixes
  f :: $'x \Rightarrow 'y$ **and**
  $\mathcal{D}_f$ :: $'x$ *set* **and**
  s :: $'x$ *set* **and**
  m :: $'z$ *monoid* **and**
  φ :: $('z, 'x)$ *binary-fun* **and**

ψ :: ($'z$, $'y$) *binary-fun* **and**
x :: $'z$
defines *equivar-prop* $\equiv$ *equivar-ind-by-act* (*carrier m*) $\mathcal{D}_f$ φ ψ
assumes
action-φ: *group-action m s* φ **and**
action-res: *group-action m UNIV* ψ **and**
dom-in-s: $\mathcal{D}_f \subseteq s$ **and**
closed-domain:
closed-under-restr-rel (*rel-induced-by-action* (*carrier m*) *s* φ) *s* $\mathcal{D}_f$ **and**
equivar-f: *satisfies f equivar-prop* **and**
group-elem-x: $x \in$ *carrier m*
shows $\forall$ *y*. *preimg f* $\mathcal{D}_f$ (ψ *x y*) = (φ *x*) ‘ (*preimg f* $\mathcal{D}_f$ *y*)
proof (*safe*)
interpret *action-φ*: *group-action m s* φ
using *action-φ*
by *simp*
interpret *action-results*: *group-action m UNIV* ψ
using *action-res*
by *simp*
have *group-elem-inv*: (*inv* $_m$ *x*) $\in$ *carrier m*
using *group.inv-closed group-hom.axioms*(*1*) *action-φ.group-hom group-elem-x*
by *metis*
fix
y :: $'y$ **and**
z :: $'x$
assume *preimg-el*: $z \in$ *preimg f* $\mathcal{D}_f$ (ψ *x y*)
obtain a :: $'x$ **where**
img: $a = \varphi$ (*inv* $_m$ *x*) *z*
by *simp*
have *domain*: $z \in \mathcal{D}_f \land z \in s$
using *preimg-el dom-in-s*
by *auto*
hence $a \in s$
using *dom-in-s action-φ group-elem-inv preimg-el img action-φ.element-image*
by *auto*
hence (z, a) $\in$ (*rel-induced-by-action* (*carrier m*) *s* φ) $\cap$ ($\mathcal{D}_f \times s$)
using *img preimg-el domain group-elem-inv*
by *auto*
hence $a \in$ ((*rel-induced-by-action* (*carrier m*) *s* φ) $\cap$ ($\mathcal{D}_f \times s$)) ‘‘ $\mathcal{D}_f$
using *img preimg-el domain group-elem-inv*
by *auto*
hence *a-in-domain*: $a \in \mathcal{D}_f$
using *closed-domain*
by *auto*
moreover have (φ (*inv* $_m$ *x*), ψ (*inv* $_m$ *x*)) $\in$ {(φ *g*, ψ *g*) | *g*. *g* $\in$ *carrier m*}
using *group-elem-inv*
by *auto*
ultimately have *f a* = ψ (*inv* $_m$ *x*) (*f z*)
using *domain equivar-f img*

unfolding *equivar-prop-def equivar-ind-by-act-def*
by *simp*
also have $f\ z = \psi\ x\ y$
using *preimg-el*
by *simp*
also have $\psi\ (inv\ _m\ x)\ (\psi\ x\ y) = y$
using *action-results.group-hom action-results.orbit-sym-aux group-elem-x*
by *simp*
finally have $f\ a = y$
by *simp*
hence $a \in preimg\ f\ \mathcal{D}_f\ y$
using *a-in-domain*
by *simp*
moreover have $z = \varphi\ x\ a$
using *group-hom.axioms(1) action-φ.group-hom action-φ.orbit-sym-aux img domain a-in-domain group-elem-x group-elem-inv group.inv-inv*
by *metis*
ultimately show $z \in (\varphi\ x)$ ‘ $(preimg\ f\ \mathcal{D}_f\ y)$
by *simp*
next
fix
$y :: 'y$ **and**
$z :: 'x$
assume *preimg-el*: $z \in preimg\ f\ \mathcal{D}_f\ y$
hence *domain*: $f\ z = y \wedge z \in \mathcal{D}_f \wedge z \in s$
using *dom-in-s*
by *auto*
hence $\varphi\ x\ z \in s$
using *group-elem-x group-action.element-image action-φ*
by *metis*
hence $(z, \varphi\ x\ z) \in (rel\text{-}induced\text{-}by\text{-}action\ (carrier\ m)\ s\ \varphi) \cap (\mathcal{D}_f \times s) \cap \mathcal{D}_f \times s$
using *group-elem-x domain*
by *auto*
hence $\varphi\ x\ z \in \mathcal{D}_f$
using *closed-domain*
by *auto*
moreover have $(\varphi\ x, \psi\ x) \in \{(\varphi\ a, \psi\ a) \mid a.\ a \in carrier\ m\}$
using *group-elem-x*
by *blast*
ultimately show $\varphi\ x\ z \in preimg\ f\ \mathcal{D}_f\ (\psi\ x\ y)$
using *equivar-f domain*
unfolding *equivar-prop-def equivar-ind-by-act-def*
by *simp*
qed

Invariance and Equivariance Function Composition

lemma *invar-comp*:
fixes

    f :: $'x \Rightarrow 'y$ **and**
    g :: $'y \Rightarrow 'z$ **and**
    r :: $'x$ *rel*
  assumes *satisfies* f (*Invariance* r)
  shows *satisfies* $(g \circ f)$ (*Invariance* r)
  using *assms*
  by *simp*

lemma *equivar-comp*:
  fixes
    f :: $'x \Rightarrow 'y$ **and**
    g :: $'y \Rightarrow 'z$ **and**
    s :: $'x$ *set* **and**
    t :: $'y$ *set* **and**
    τ :: $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set* **and**
    υ :: $(('y \Rightarrow 'y) \times ('z \Rightarrow 'z))$ *set*
  defines
    transitive-acts $\equiv$
      $\{(\varphi, \psi).\ \exists\ \chi :: 'y \Rightarrow 'y.\ (\varphi, \chi) \in \tau \wedge (\chi, \psi) \in \upsilon \wedge \chi\ `\ f\ `\ s \subseteq t\}$
  assumes
    $f\ `\ s \subseteq t$ **and**
    satisfies f (*Equivariance* s τ) **and**
    satisfies g (*Equivariance* t υ)
  shows *satisfies* $(g \circ f)$ (*Equivariance* s *transitive-acts*)
proof (*unfold transitive-acts-def*, *simp*, *safe*)
  fix
    φ :: $'x \Rightarrow 'x$ **and**
    χ :: $'y \Rightarrow 'y$ **and**
    ψ :: $'z \Rightarrow 'z$ **and**
    x :: $'x$
  assume
    x-in-X: $x \in s$ **and**
    φ-x-in-X: $\varphi\ x \in s$ **and**
    χ-img$_f$-img$_s$-in-t: $\chi\ `\ f\ `\ s \subseteq t$ **and**
    act-f: $(\varphi, \chi) \in \tau$ **and**
    act-g: $(\chi, \psi) \in \upsilon$
  hence $f\ x \in t \wedge \chi\ (f\ x) \in t$
    using *assms*
    by *blast*
  hence $\psi\ (g\ (f\ x)) = g\ (\chi\ (f\ x))$
    using *act-g assms*
    by *fastforce*
  also have $g\ (f\ (\varphi\ x)) = g\ (\chi\ (f\ x))$
    using *assms act-f x-in-X φ-x-in-X*
    by *fastforce*
  finally show $g\ (f\ (\varphi\ x)) = \psi\ (g\ (f\ x))$
    by *simp*
qed

lemma *equivar-ind-by-act-comp*:
  fixes
    f :: $'x \Rightarrow 'y$ **and**
    g :: $'y \Rightarrow 'z$ **and**
    s :: $'w$ *set* **and**
    t :: $'x$ *set* **and**
    u :: $'y$ *set* **and**
    φ :: $('w, 'x)$ *binary-fun* **and**
    χ :: $('w, 'y)$ *binary-fun* **and**
    ψ :: $('w, 'z)$ *binary-fun*
  assumes
    f ‘ $t \subseteq u$ **and**
    $\forall\ x \in s.\ \chi\ x$ ‘ f ‘ $t \subseteq u$ **and**
    satisfies f (*equivar-ind-by-act* $s\ t\ \varphi\ \chi$) **and**
    satisfies g (*equivar-ind-by-act* $s\ u\ \chi\ \psi$)
  shows *satisfies* $(g \circ f)$ (*equivar-ind-by-act* $s\ t\ \varphi\ \psi$)
proof −
  let $?a_\varphi = \{(\varphi\ a, \chi\ a) \mid a.\ a \in s\}$ **and**
      $?a_\psi = \{(\chi\ a, \psi\ a) \mid a.\ a \in s\}$
  have $\forall\ a \in s.\ (\varphi\ a, \chi\ a) \in \{(\varphi\ a, \chi\ a) \mid b.\ b \in s\} \wedge$
                $(\chi\ a, \psi\ a) \in \{(\chi\ b, \psi\ b) \mid b.\ b \in s\} \wedge \chi\ a$ ‘ f ‘ $t \subseteq u$
    using *assms*
    by *blast*
  hence $\{(\varphi\ a, \psi\ a) \mid a.\ a \in s\} \subseteq$
        $\{(\varphi, \psi).\ \exists\ \upsilon.\ (\varphi, \upsilon) \in ?a_\varphi \wedge (\upsilon, \psi) \in ?a_\psi \wedge \upsilon$ ‘ f ‘ $t \subseteq u\}$
    by *blast*
  hence *satisfies* $(g \circ f)$ (*Equivariance* t $\{(\varphi\ a, \psi\ a) \mid a.\ a \in s\}$)
    using *assms equivar-comp*[*of f t u* $?a_\varphi$ *g* $?a_\psi$] *equivar-under-subset′*
    unfolding *equivar-ind-by-act-def*
    by (*metis* (*no-types*, *lifting*))
  thus *?thesis*
    unfolding *equivar-ind-by-act-def*
    by *blast*
qed

lemma *equivar-set-minus*:
  fixes
    f :: $'x \Rightarrow 'y$ *set* **and**
    g :: $'x \Rightarrow 'y$ *set* **and**
    s :: $'z$ *set* **and**
    t :: $'x$ *set* **and**
    φ :: $('z, 'x)$ *binary-fun* **and**
    ψ :: $('z, 'y)$ *binary-fun*
  assumes
    f-equivar: *satisfies* f (*equivar-ind-by-act* $s\ t\ \varphi$ (*set-action* ψ)) **and**
    g-equivar: *satisfies* g (*equivar-ind-by-act* $s\ t\ \varphi$ (*set-action* ψ)) **and**
    bij-a: $\forall\ a \in s.$ *bij* $(\psi\ a)$
  shows *satisfies* $(\lambda\ b.\ f\ b - g\ b)$ (*equivar-ind-by-act* $s\ t\ \varphi$ (*set-action* ψ))
proof −

have $\forall\ a \in s.\ \forall\ x \in t.\ \varphi\ a\ x \in t \longrightarrow f\ (\varphi\ a\ x) = \psi\ a\ `\ (f\ x)$
  using *f-equivar*
  unfolding *rewrite-equivar-ind-by-act*
  by *simp*
moreover have $\forall\ a \in s.\ \forall\ x \in t.\ \varphi\ a\ x \in t \longrightarrow g\ (\varphi\ a\ x) = \psi\ a\ `\ (g\ x)$
  using *g-equivar*
  unfolding *rewrite-equivar-ind-by-act*
  by *simp*
ultimately have
$\forall\ a \in s.\ \forall\ b \in t.\ \varphi\ a\ b \in t \longrightarrow f\ (\varphi\ a\ b) - g\ (\varphi\ a\ b) = \psi\ a\ `\ (f\ b) - \psi\ a\ `\ (g$
$b)$
  by *blast*
moreover have $\forall\ a \in s.\ \forall\ u\ v.\ \psi\ a\ `\ u - \psi\ a\ `\ v = \psi\ a\ `\ (u - v)$
  using *bij-a image-set-diff*
  unfolding *bij-def*
  by *blast*
ultimately show *?thesis*
  unfolding *set-action.simps*
  using *rewrite-equivar-ind-by-act*
  by *fastforce*
qed

lemma *equivar-union-under-img-act*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$s :: 'z\ set$ **and**
$\varphi :: ('z, 'x)\ binary\text{-}fun$
shows *satisfies* $\bigcup$ (*equivar-ind-by-act s UNIV*
(*set-action* (*set-action* φ)) (*set-action* φ))
proof (*unfold equivar-ind-by-act-def*, *clarsimp*, *safe*)
fix
$x :: 'z$ **and**
$ts :: 'x\ set\ set$ **and**
$t :: 'x\ set$ **and**
$y :: 'x$
assume
$y \in t$ **and**
$t \in ts$
thus
$\varphi\ x\ y \in \varphi\ x\ `\ \bigcup\ ts$ **and**
$\varphi\ x\ y \in \bigcup\ ((`)\ (\varphi\ x)\ `\ ts)$
by (*blast*, *blast*)
qed

end

1.9 Symmetry Properties of Voting Rules

theory *Voting-Symmetry*
imports *Symmetry-Of-Functions*
Social-Choice-Result
Social-Welfare-Result
Profile
begin

1.9.1 Definitions

fun (**in** *result*) *results-closed-under-rel* :: $('a, 'v)$ *Election rel* $\Rightarrow$ *bool* **where**
results-closed-under-rel r =
($\forall$ $(e, e') \in r$. *limit-set* (*alternatives-$\mathcal{E}$ e*) *UNIV* = *limit-set* (*alternatives-$\mathcal{E}$ e'*) *UNIV*)

fun *result-action* :: $('x, 'r)$ *binary-fun* $\Rightarrow$ $('x, 'r$ *Result*) *binary-fun* **where**
result-action ψ x = (λ r. (ψ x ‘ *elect-r r*, ψ x ‘ *reject-r r*, ψ x ‘ *defer-r r*))

Anonymity

definition *anonymity$_{\mathcal{G}}$* :: $('v \Rightarrow 'v)$ *monoid* **where**
anonymity$_{\mathcal{G}}$ = *BijGroup* (*UNIV*::$'v$ *set*)

fun *φ-anon* :: $('a, 'v)$ *Election set* $\Rightarrow$ $('v \Rightarrow 'v) \Rightarrow (('a, 'v)$ *Election*
$\Rightarrow ('a, 'v)$ *Election*) **where**
φ-anon $\mathcal{E}$ π = *extensional-continuation* (*rename* π) $\mathcal{E}$

fun *anonymity$_{\mathcal{R}}$* :: $('a, 'v)$ *Election set* $\Rightarrow$ $('a, 'v)$ *Election rel* **where**
anonymity$_{\mathcal{R}}$ $\mathcal{E}$ = *rel-induced-by-action* (*carrier anonymity$_{\mathcal{G}}$*) $\mathcal{E}$ (*φ-anon* $\mathcal{E}$)

Neutrality

fun *rel-rename* :: $('a \Rightarrow 'a, 'a$ *Preference-Relation*) *binary-fun* **where**
rel-rename π r = $\{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in r\}$

fun *alternatives-rename* :: $('a \Rightarrow 'a, ('a, 'v)$ *Election*) *binary-fun* **where**
alternatives-rename π $\mathcal{E}$ = (π ‘ (*alternatives-$\mathcal{E}$* $\mathcal{E}$), *voters-$\mathcal{E}$* $\mathcal{E}$, (*rel-rename* π) $\circ$ (*profile-$\mathcal{E}$* $\mathcal{E}$))

definition *neutrality$_{\mathcal{G}}$* :: $('a \Rightarrow 'a)$ *monoid* **where**
neutrality$_{\mathcal{G}}$ = *BijGroup* (*UNIV*::$'a$ *set*)

fun *φ-neutr* :: $('a, 'v)$ *Election set* $\Rightarrow$ $('a \Rightarrow 'a, ('a, 'v)$ *Election*) *binary-fun* **where**
φ-neutr $\mathcal{E}$ π = *extensional-continuation* (*alternatives-rename* π) $\mathcal{E}$

fun *neutrality$_{\mathcal{R}}$* :: $('a, 'v)$ *Election set* $\Rightarrow$ $('a, 'v)$ *Election rel* **where**
neutrality$_{\mathcal{R}}$ $\mathcal{E}$ = *rel-induced-by-action* (*carrier neutrality$_{\mathcal{G}}$*) $\mathcal{E}$ (*φ-neutr* $\mathcal{E}$)

fun *ψ-neutr$_{c}$* :: $('a \Rightarrow 'a, 'a)$ *binary-fun* **where**

$\psi\text{-}neutr_{\mathrm{c}}\ \pi\ r = \pi\ r$

fun $\psi\text{-}neutr_{\mathrm{w}}$ :: $('a \Rightarrow 'a,\ 'a\ rel)$ *binary-fun* **where**
$\psi\text{-}neutr_{\mathrm{w}}\ \pi\ r = rel\text{-}rename\ \pi\ r$

Homogeneity

fun $homogeneity_{\mathcal{R}}$ :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$homogeneity_{\mathcal{R}}\ \mathcal{E} =$
$\{(E,\ E') \in \mathcal{E} \times \mathcal{E}.$
$alternatives\text{-}\mathcal{E}\ E = alternatives\text{-}\mathcal{E}\ E' \wedge finite\ (voters\text{-}\mathcal{E}\ E) \wedge finite\ (voters\text{-}\mathcal{E}\ E') \wedge$
$(\exists\ n > 0.\ \forall\ r{::}('a\ Preference\text{-}Relation).\ vote\text{-}count\ r\ E = n * (vote\text{-}count\ r\ E'))\}$

fun *copy-list* :: $nat \Rightarrow 'x\ list \Rightarrow 'x\ list$ **where**
$copy\text{-}list\ 0\ l = []\ |$
$copy\text{-}list\ (Suc\ n)\ l = copy\text{-}list\ n\ l\ @\ l$

fun $homogeneity_{\mathcal{R}}'$ :: $('a,\ 'v{::}linorder)$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$homogeneity_{\mathcal{R}}'\ \mathcal{E} =$
$\{(E,\ E') \in \mathcal{E} \times \mathcal{E}.$
$alternatives\text{-}\mathcal{E}\ E = alternatives\text{-}\mathcal{E}\ E' \wedge finite\ (voters\text{-}\mathcal{E}\ E) \wedge finite\ (voters\text{-}\mathcal{E}\ E') \wedge$
$(\exists\ n > 0.\ to\text{-}list\ (voters\text{-}\mathcal{E}\ E')\ (profile\text{-}\mathcal{E}\ E') =$
$copy\text{-}list\ n\ (to\text{-}list\ (voters\text{-}\mathcal{E}\ E)\ (profile\text{-}\mathcal{E}\ E)))\}$

Reversal Symmetry

fun *rev-rel* :: $'a\ rel \Rightarrow 'a\ rel$ **where**
$rev\text{-}rel\ r = \{(a,\ b).\ (b,\ a) \in r\}$

fun *rel-app* :: $('a\ rel \Rightarrow 'a\ rel) \Rightarrow ('a,\ 'v)\ Election \Rightarrow ('a,\ 'v)\ Election$ **where**
$rel\text{-}app\ f\ (A,\ V,\ p) = (A,\ V,\ f \circ p)$

definition $reversal_{\mathcal{G}}$ :: $('a\ rel \Rightarrow 'a\ rel)$ *monoid* **where**
$reversal_{\mathcal{G}} = (\!|carrier = \{rev\text{-}rel,\ id\},\ monoid.mult = comp,\ one = id|\!)$

fun φ*-rev* :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $('a\ rel \Rightarrow 'a\ rel,\ ('a,\ 'v)\ Election)$ *binary-fun* **where**
$\varphi\text{-}rev\ \mathcal{E}\ \varphi = extensional\text{-}continuation\ (rel\text{-}app\ \varphi)\ \mathcal{E}$

fun ψ*-rev* :: $('a\ rel \Rightarrow 'a\ rel,\ 'a\ rel)$ *binary-fun* **where**
$\psi\text{-}rev\ \varphi\ r = \varphi\ r$

fun $reversal_{\mathcal{R}}$ :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$reversal_{\mathcal{R}}\ \mathcal{E} = rel\text{-}induced\text{-}by\text{-}action\ (carrier\ reversal_{\mathcal{G}})\ \mathcal{E}\ (\varphi\text{-}rev\ \mathcal{E})$

1.9.2 Auxiliary Lemmas

fun *n-app* :: *nat* ⇒ ('*x* ⇒ '*x*) ⇒ ('*x* ⇒ '*x*) **where**
  n-app 0 f = *id* |
  n-app (*Suc n*) *f* = *f* ∘ *n-app n f*

lemma *n-app-rewrite*:
  fixes
    f :: '*x* ⇒ '*x* **and**
    n :: *nat* **and**
    x :: '*x*
  shows (*f* ∘ *n-app n f*) *x* = (*n-app n f* ∘ *f*) *x*
proof (*clarsimp*, *induction n f arbitrary*: *x rule*: *n-app.induct*)
  case (*1 f*)
  fix
    f :: '*x* ⇒ '*x* **and**
    x :: '*x*
  show *f* (*n-app 0 f x*) = *n-app 0 f* (*f x*)
    by *simp*
next
  case (*2 n f*)
  fix
    f :: '*x* ⇒ '*x* **and**
    n :: *nat* **and**
    x :: '*x*
  assume ⋀ *y*. *f* (*n-app n f y*) = *n-app n f* (*f y*)
  thus *f* (*n-app* (*Suc n*) *f x*) = *n-app* (*Suc n*) *f* (*f x*)
    by *simp*
qed

lemma *n-app-leaves-set*:
  fixes
    A :: '*x set* **and**
    B :: '*x set* **and**
    f :: '*x* ⇒ '*x* **and**
    x :: '*x*
  assumes
    fin-A: *finite A* **and**
    fin-B: *finite B* **and**
    x-el: *x* ∈ *A* − *B* **and**
    bij: *bij-betw f A B*
  obtains *n* :: *nat* **where**
    n > *0* **and**
    n-app n f x ∈ *B* − *A* **and**
    ∀ *m* > *0*. *m* < *n* ⟶ *n-app m f x* ∈ *A* ∩ *B*
proof −
  assume *existence-witness*:
    ⋀ *n*. *0* < *n* ⟹ *n-app n f x* ∈ *B* − *A* ⟹ ∀ *m* > *0*. *m* < *n* ⟶ *n-app m f x* ∈ *A* ∩ *B* ⟹ *?thesis*
  have *ex-A*: ∃ *n* > *0*. *n-app n f x* ∈ *B* − *A* ∧ (∀ *m* > *0*. *m* < *n* ⟶ *n-app m f*

$x \in A$)
  proof (*rule ccontr*, *clarsimp*)
    assume *nex*:
      $\forall\ n.\ n\text{-}app\ n\ f\ x \in B \longrightarrow n = 0 \vee n\text{-}app\ n\ f\ x \in A \vee (\exists\ m > 0.\ m < n \wedge n\text{-}app\ m\ f\ x \notin A)$
    hence $\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in B \longrightarrow n\text{-}app\ n\ f\ x \in A \vee (\exists\ m > 0.\ m < n \wedge n\text{-}app\ m\ f\ x \notin A)$
      by *blast*
    moreover have $(\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in B \longrightarrow n\text{-}app\ n\ f\ x \in A) \longrightarrow False$
    proof (*safe*)
      assume *in-A*: $\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in B \longrightarrow n\text{-}app\ n\ f\ x \in A$
      hence $\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in A \longrightarrow n\text{-}app\ (Suc\ n)\ f\ x \in A$
        using *n-app.simps bij*
        unfolding *bij-betw-def*
        by *force*
      hence *in-AB-imp-in-AB*:
        $\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in A \cap B \longrightarrow n\text{-}app\ (Suc\ n)\ f\ x \in A \cap B$
        using *n-app.simps bij*
        unfolding *bij-betw-def*
        by *auto*
      have *in-int*: $\forall\ n > 0.\ n\text{-}app\ n\ f\ x \in A \cap B$
      proof (*clarify*)
        fix $n :: nat$
        assume $n > 0$
        thus $n\text{-}app\ n\ f\ x \in A \cap B$
        proof (*induction n*)
          case *0*
          thus *?case*
            by *safe*
        next
          case (*Suc n*)
          assume $0 < n \Longrightarrow n\text{-}app\ n\ f\ x \in A \cap B$
          moreover have $n = 0 \longrightarrow n\text{-}app\ (Suc\ n)\ f\ x = f\ x$
            by *simp*
          ultimately show $n\text{-}app\ (Suc\ n)\ f\ x \in A \cap B$
            using *x-el bij in-A in-AB-imp-in-AB*
            unfolding *bij-betw-def*
            by *blast*
        qed
      qed
      hence $\{n\text{-}app\ n\ f\ x \mid n.\ n > 0\} \subseteq A \cap B$
        by *blast*
      hence *finite* $\{n\text{-}app\ n\ f\ x \mid n.\ n > 0\}$
        using *fin-A fin-B rev-finite-subset*
        by *blast*
      moreover have
        $inj\text{-}on\ (\lambda\ n.\ n\text{-}app\ n\ f\ x)\ \{n.\ n > 0\} \longrightarrow infinite\ ((\lambda\ n.\ n\text{-}app\ n\ f\ x)\ \text{`}\ \{n.\ n > 0\})$
        using *diff-is-0-eq′ finite-imageD finite-nat-set-iff-bounded lessI*

less-imp-diff-less mem-Collect-eq nless-le
by *metis*
moreover have $(\lambda\ n.\ n\text{-}app\ n\ f\ x)$ ‘ $\{n.\ n > 0\} = \{n\text{-}app\ n\ f\ x \mid n.\ n > 0\}$
by *auto*
ultimately have $\neg$ *inj-on* $(\lambda\ n.\ n\text{-}app\ n\ f\ x)\ \{n.\ n > 0\}$
by *metis*
hence $\exists\ n.\ n > 0 \wedge (\exists\ m > n.\ n\text{-}app\ n\ f\ x = n\text{-}app\ m\ f\ x)$
using *linorder-inj-onI′ mem-Collect-eq*
by *metis*
hence $\exists\ n\text{-}min.\ 0 < n\text{-}min \wedge (\exists\ m > n\text{-}min.\ n\text{-}app\ n\text{-}min\ f\ x = n\text{-}app\ m\ f$
$x) \wedge$
$(\forall\ n < n\text{-}min.\ \neg\ (0 < n \wedge (\exists\ m > n.\ n\text{-}app\ n\ f\ x = n\text{-}app\ m\ f\ x)))$
using *exists-least-iff* $[of\ \lambda\ n.\ n > 0 \wedge (\exists\ m > n.\ n\text{-}app\ n\ f\ x = n\text{-}app\ m\ f$
$x)]$
by *presburger*
then obtain *n-min* :: *nat* **where**
n-min-pos: $n\text{-}min > 0$ **and**
$\exists\ m > n\text{-}min.\ n\text{-}app\ n\text{-}min\ f\ x = n\text{-}app\ m\ f\ x$ **and**
neq: $\forall\ n < n\text{-}min.\ \neg\ (n > 0 \wedge (\exists\ m > n.\ n\text{-}app\ n\ f\ x = n\text{-}app\ m\ f\ x))$
by *blast*
then obtain *m* :: *nat* **where**
m-gt-n-min: $m > n\text{-}min$ **and**
$n\text{-}app\ n\text{-}min\ f\ x = f\ (n\text{-}app\ (m - 1)\ f\ x)$
using *comp-apply diff-Suc-1 less-nat-zero-code n-app.elims*
by (*metis* (*mono-tags*, *lifting*))
moreover have $n\text{-}app\ n\text{-}min\ f\ x = f\ (n\text{-}app\ (n\text{-}min - 1)\ f\ x)$
using *Suc-pred′ n-min-pos comp-eq-id-dest id-comp diff-Suc-1*
less-nat-zero-code n-app.elims
by (*metis* (*mono-tags*, *opaque-lifting*))
moreover have $n\text{-}app\ (m - 1)\ f\ x \in A \wedge n\text{-}app\ (n\text{-}min - 1)\ f\ x \in A$
using *in-int x-el n-min-pos m-gt-n-min Diff-iff IntD1 diff-le-self id-apply*
nless-le
cancel-comm-monoid-add-class.diff-cancel n-app.simps(*1*)
by *metis*
ultimately have *eq*: $n\text{-}app\ (m - 1)\ f\ x = n\text{-}app\ (n\text{-}min - 1)\ f\ x$
using *bij*
unfolding *bij-betw-def inj-def inj-on-def*
by *simp*
moreover have $m - 1 > n\text{-}min - 1$
using *m-gt-n-min n-min-pos*
by *simp*
ultimately have *case-greater-0*: $n\text{-}min - 1 > 0 \longrightarrow False$
using *neq n-min-pos diff-less zero-less-one*
by *metis*
have $n\text{-}app\ (m - 1)\ f\ x \in B$
using *in-int m-gt-n-min n-min-pos*
by *simp*
thus *False*
using *x-el eq case-greater-0*

by *simp*
qed
ultimately have $\exists\ n > 0.\ \exists\ m > 0.\ m < n \wedge$ *n-app m f x* $\notin A$
by *blast*
hence $\exists\ n.\ n > 0 \wedge$ *n-app n f x* $\notin A \wedge (\forall\ m < n.\ \neg\ (m > 0 \wedge$ *n-app m f x* $\notin A))$
using *exists-least-iff*[*of* $\lambda\ n.\ n > 0 \wedge$ *n-app n f x* $\notin A$]
by *blast*
then obtain n :: *nat* **where**
n-pos: $n > 0$ **and**
not-in-A: *n-app n f x* $\notin A$ **and**
less-in-A: $\forall\ m.\ (0 < m \wedge m < n) \longrightarrow$ *n-app m f x* $\in A$
by *blast*
moreover have *n-app 0 f x* $\in A$
using *x-el*
by *simp*
ultimately have *n-app* $(n - 1)$ *f x* $\in A$
using *bot-nat-0.not-eq-extremum diff-less less-numeral-extra*(*1*)
by *metis*
moreover have *n-app n f x* $=$ *f* (*n-app* $(n - 1)$ *f x*)
using *n-app.simps*(*2*) *Suc-pred′ n-pos comp-eq-id-dest fun.map-id*
by (*metis* (*mono-tags*, *opaque-lifting*))
ultimately show *False*
using *bij nex not-in-A n-pos less-in-A*
unfolding *bij-betw-def*
by *blast*
qed
moreover have *n-app-f-x-in-A*: *n-app 0 f x* $\in A$
using *x-el*
by *simp*
ultimately have
$\forall\ n.\ (\forall\ m > 0.\ m < n \longrightarrow$ *n-app m f x* $\in A) \longrightarrow (\forall\ m > 0.\ m < n \longrightarrow$ *n-app* $(m - 1)$ *f x* $\in A)$
using *bot-nat-0.not-eq-extremum less-imp-diff-less*
by *metis*
moreover have $\forall\ m > 0.$ *n-app m f x* $=$ *f* (*n-app* $(m - 1)$ *f x*)
using *bot-nat-0.not-eq-extremum comp-apply diff-Suc-1 n-app.elims*
by (*metis* (*mono-tags*, *lifting*))
ultimately have
$\forall\ n.\ (\forall\ m > 0.\ m < n \longrightarrow$ *n-app m f x* $\in A) \longrightarrow (\forall\ m > 0.\ m \leq n \longrightarrow$ *n-app m f x* $\in B)$
using *bij n-app.simps*(*1*) *n-app-f-x-in-A diff-Suc-1 gr0-conv-Suc imageI linorder-not-le nless-le not-less-eq-eq*
unfolding *bij-betw-def*
by *metis*
hence $\exists\ n > 0.$ *n-app n f x* $\in B - A \wedge (\forall\ m > 0.\ m < n \longrightarrow$ *n-app m f x* $\in A \cap B)$
using *IntI nless-le ex-A*
by *metis*

thus *?thesis*
  using *existence-witness*
  by *blast*
qed

lemma *n-app-rev*:
 fixes
  $A :: 'x\ set$ **and**
  $B :: 'x\ set$ **and**
  $f :: 'x \Rightarrow 'x$ **and**
  $n :: nat$ **and**
  $m :: nat$ **and**
  $x :: 'x$ **and**
  $y :: 'x$
 assumes
  x-in-A: $x \in A$ **and**
  y-in-A: $y \in A$ **and**
  n-geq-m: $n \geq m$ **and**
  n-app-eq-m-n: *n-app* $n\ f\ x =$ *n-app* $m\ f\ y$ **and**
  n-app-x-in-A: $\forall\ n' < n.$ *n-app* $n'\ f\ x \in A$ **and**
  n-app-y-in-A: $\forall\ m' < m.$ *n-app* $m'\ f\ y \in A$ **and**
  fin-A: *finite* A **and**
  fin-B: *finite* B **and**
  bij-f-A-B: *bij-betw* $f\ A\ B$
 shows *n-app* $(n - m)\ f\ x = y$
 using *assms*
proof (*induction n f arbitrary*: *m x y rule*: *n-app.induct*)
 case (*1 f*)
 fix
  $f :: 'x \Rightarrow 'x$ **and**
  $m :: nat$ **and**
  $x :: 'x$ **and**
  $y :: 'x$
 assume
  $m \leq 0$ **and**
  n-app $0\ f\ x =$ *n-app* $m\ f\ y$
 thus *n-app* $(0 - m)\ f\ x = y$
  by *simp*
next
 case (*2 n f*)
 fix
  $f :: 'x \Rightarrow 'x$ **and**
  $n :: nat$ **and**
  $m :: nat$ **and**
  $x :: 'x$ **and**
  $y :: 'x$
 assume
  bij: *bij-betw* $f\ A\ B$ **and**
  x-in-A: $x \in A$ **and**

y-in-A: $y \in A$ **and**
m-leq-suc-n: $m \leq Suc\ n$ **and**
x-dom: $\forall\ n' < Suc\ n.\ n\text{-}app\ n'\ f\ x \in A$ **and**
y-dom: $\forall\ m' < m.\ n\text{-}app\ m'\ f\ y \in A$ **and**
eq: $n\text{-}app\ (Suc\ n)\ f\ x = n\text{-}app\ m\ f\ y$ **and**
hyp:
$\bigwedge m\ x\ y.$
$x \in A \Longrightarrow$
$y \in A \Longrightarrow$
$m \leq n \Longrightarrow$
$n\text{-}app\ n\ f\ x = n\text{-}app\ m\ f\ y \Longrightarrow$
$\forall\ n' < n.\ n\text{-}app\ n'\ f\ x \in A \Longrightarrow$
$\forall\ m' < m.\ n\text{-}app\ m'\ f\ y \in A \Longrightarrow$
$finite\ A \Longrightarrow finite\ B \Longrightarrow bij\text{-}betw\ f\ A\ B \Longrightarrow n\text{-}app\ (n - m)\ f\ x = y$
hence $m > 0 \longrightarrow f\ (n\text{-}app\ n\ f\ x) = f\ (n\text{-}app\ (m - 1)\ f\ y)$
using *Suc-pred′ comp-apply n-app.simps*(*2*)
by (*metis* (*mono-tags*, *opaque-lifting*))
moreover have $n\text{-}app\ n\ f\ x \in A$
using *x-in-A x-dom*
by *blast*
moreover have $m > 0 \longrightarrow n\text{-}app\ (m - 1)\ f\ y \in A$
using *y-dom*
by *simp*
ultimately have $m > 0 \longrightarrow n\text{-}app\ n\ f\ x = n\text{-}app\ (m - 1)\ f\ y$
using *bij*
unfolding *bij-betw-def inj-on-def*
by *blast*
moreover have $m - 1 \leq n$
using *m-leq-suc-n*
by *simp*
hence $m > 0 \longrightarrow n\text{-}app\ (n - (m - 1))\ f\ x = y$
using *hyp x-in-A y-in-A x-dom y-dom Suc-pred fin-A fin-B bij calculation less-SucI*
unfolding *One-nat-def*
by *metis*
hence $m > 0 \longrightarrow n\text{-}app\ (Suc\ n - m)\ f\ x = y$
using *Suc-diff-eq-diff-pred*
by *presburger*
moreover have $m = 0 \longrightarrow n\text{-}app\ (Suc\ n - m)\ f\ x = y$
using *eq*
by *simp*
ultimately show $n\text{-}app\ (Suc\ n - m)\ f\ x = y$
by *blast*
qed

lemma *n-app-inv*:
fixes
A :: *′x set* **and**
B :: *′x set* **and**

```
    f :: 'x ⇒ 'x and
    n :: nat and
    x :: 'x
  assumes
    x ∈ B and
    ∀ m ≥ 0. m < n ⟶ n-app m (the-inv-into A f) x ∈ B and
    bij-betw f A B
  shows n-app n f (n-app n (the-inv-into A f) x) = x
  using assms
proof (induction n f arbitrary: x rule: n-app.induct)
  case (1 f)
  fix f :: 'x ⇒ 'x
  show ?case
    by simp
next
  case (2 n f)
  fix
    n :: nat and
    f :: 'x ⇒ 'x and
    x :: 'x
  assume
    x-in-B: x ∈ B and
    bij: bij-betw f A B and
    stays-in-B: ∀ m ≥ 0. m < Suc n ⟶ n-app m (the-inv-into A f) x ∈ B and
    hyp: ⋀ x. x ∈ B ⟹
            ∀ m ≥ 0. m < n ⟶ n-app m (the-inv-into A f) x ∈ B ⟹
            bij-betw f A B ⟹ n-app n f (n-app n (the-inv-into A f) x) = x
  have n-app (Suc n) f (n-app (Suc n) (the-inv-into A f) x) =
    n-app n f (f (n-app (Suc n) (the-inv-into A f) x))
    using n-app-rewrite
    by simp
  also have ... = n-app n f (n-app n (the-inv-into A f) x)
    using stays-in-B bij
    by (simp add: f-the-inv-into-f-bij-betw)
  finally show n-app (Suc n) f (n-app (Suc n) (the-inv-into A f) x) = x
    using hyp bij stays-in-B x-in-B
    by simp
qed

lemma bij-betw-finite-ind-global-bij:
  fixes
    A :: 'x set and
    B :: 'x set and
    f :: 'x ⇒ 'x
  assumes
    fin-A: finite A and
    fin-B: finite B and
    bij: bij-betw f A B
  obtains g :: 'x ⇒ 'x where
```

```
    bij g and
    ∀ a ∈ A. g a = f a and
    ∀ b ∈ B − A. g b ∈ A − B ∧ (∃ n > 0. n-app n f (g b) = b) and
    ∀ x ∈ UNIV − A − B. g x = x
proof −
  assume existence-witness:
    ⋀ g. bij g ⟹
          ∀ a ∈ A. g a = f a ⟹
          ∀ b ∈ B − A. g b ∈ A − B ∧ (∃ n > 0. n-app n f (g b) = b) ⟹
          ∀ x ∈ UNIV − A − B. g x = x ⟹ ?thesis
  have bij-inv: bij-betw (the-inv-into A f) B A
    using bij bij-betw-the-inv-into
    by blast
  then obtain g' :: 'x ⇒ nat where
    greater-0: ∀ x ∈ B − A. g' x > 0 and
    in-set-diff: ∀ x ∈ B − A. n-app (g' x) (the-inv-into A f) x ∈ A − B and
    minimal: ∀ x ∈ B − A. ∀ n > 0. n < g' x ⟶ n-app n (the-inv-into A f) x
∈ B ∩ A
    using n-app-leaves-set[of B A - the-inv-into A f False] fin-A fin-B
    by metis
  obtain g :: 'x ⇒ 'x where
    def-g:
      g = (λ x. if x ∈ A then f x else
               (if x ∈ B − A then n-app (g' x) (the-inv-into A f) x else x))
    by simp
  hence coincide: ∀ a ∈ A. g a = f a
    by simp
  have id: ∀ x ∈ UNIV − A − B. g x = x
    using def-g
    by simp
  have ∀ x ∈ B − A. n-app 0 (the-inv-into A f) x ∈ B
    by simp
  moreover have ∀ x ∈ B − A. ∀ n > 0. n < g' x ⟶ n-app n (the-inv-into A
f) x ∈ B
    using minimal
    by blast
  ultimately have ∀ x ∈ B − A. n-app (g' x) f (n-app (g' x) (the-inv-into A f)
x) = x
    using n-app-inv bij DiffD1 antisym-conv2
    by metis
  hence ∀ x ∈ B − A. n-app (g' x) f (g x) = x
    using def-g
    by simp
  with greater-0 in-set-diff
  have reverse: ∀ x ∈ B − A. g x ∈ A − B ∧ (∃ n > 0. n-app n f (g x) = x)
    using def-g
    by auto
  have ∀ x ∈ UNIV − A − B. g x = id x
    using def-g
```

by *simp*
hence g ‘ $(UNIV - A - B) = UNIV - A - B$
by *simp*
moreover have g ‘ $A = B$
using *def-g bij*
unfolding *bij-betw-def*
by *simp*
moreover have $A \cup (UNIV - A - B) = UNIV - (B - A) \wedge B \cup (UNIV - A - B) = UNIV - (A - B)$
by *blast*
ultimately have *surj-cases-13*: g ‘ $(UNIV - (B - A)) = UNIV - (A - B)$
using *image-Un*
by *metis*
have *inj-on* g $A \wedge$ *inj-on* g $(UNIV - A - B)$
using *def-g bij*
unfolding *bij-betw-def inj-on-def*
by *simp*
hence *inj-cases-13*: *inj-on* g $(UNIV - (B - A))$
unfolding *inj-on-def*
using *DiffD2 DiffI bij bij-betwE def-g*
by (*metis* (*no-types*, *lifting*))
have *card* $A =$ *card* B
using *fin-A fin-B bij bij-betw-same-card*
by *blast*
with *fin-A fin-B*
have *finite* $(B - A) \wedge$ *finite* $(A - B) \wedge$ *card* $(B - A) =$ *card* $(A - B)$
using *card-le-sym-Diff finite-Diff2 nle-le*
by *metis*
moreover have $(\lambda\ x.$ *n-app* $(g'\ x)$ (*the-inv-into* A f) $x)$ ‘ $(B - A) \subseteq A - B$
using *in-set-diff*
by *blast*
moreover have *inj-on* $(\lambda\ x.$ *n-app* $(g'\ x)$ (*the-inv-into* A f) $x)$ $(B - A)$
proof (*unfold inj-on-def*, *safe*)
fix
$x :: 'x$ **and**
$y :: 'x$
assume
x-in-B: $x \in B$ **and**
x-not-in-A: $x \notin A$ **and**
y-in-B: $y \in B$ **and**
y-not-in-A: $y \notin A$ **and**
n-app $(g'\ x)$ (*the-inv-into* A f) $x =$ *n-app* $(g'\ y)$ (*the-inv-into* A f) y
moreover from *this* **have**
$\forall\ n < g'\ x.$ *n-app* n (*the-inv-into* A f) $x \in B$ **and**
$\forall\ n < g'\ y.$ *n-app* n (*the-inv-into* A f) $y \in B$
using *minimal Diff-iff Int-iff bot-nat-0.not-eq-extremum eq-id-iff n-app.simps*(*1*)
by (*metis*, *metis*)
ultimately have *x-to-y*:
n-app $(g'\ x - g'\ y)$ (*the-inv-into* A f) $x = y \vee$

n-app $(g'\ y - g'\ x)$ (*the-inv-into* $A\ f$) $y = x$
using *x-in-B y-in-B bij-inv fin-A fin-B*
n-app-rev[*of x*] *n-app-rev*[*of y B x g′ x g′ y*]
by *fastforce*
hence $g'\ x \neq g'\ y \longrightarrow$
$((\exists\ n > 0.\ n < g'\ x \wedge$ *n-app n* (*the-inv-into* $A\ f$) $x \in B - A) \vee$
$(\exists\ n > 0.\ n < g'\ y \wedge$ *n-app n* (*the-inv-into* $A\ f$) $y \in B - A))$
using *greater-0 x-in-B x-not-in-A y-in-B y-not-in-A Diff-iff diff-less-mono2*
diff-zero id-apply less-Suc-eq-0-disj n-app.elims
by (*metis* (*full-types*))
thus $x = y$
using *minimal x-in-B x-not-in-A y-in-B y-not-in-A x-to-y*
by *force*
qed
ultimately have *bij-betw* (λ *x. n-app* $(g'\ x)$ (*the-inv-into* $A\ f$) x) $(B - A)$ $(A - B)$
unfolding *bij-betw-def*
by (*simp add*: *card-image card-subset-eq*)
hence *bij-case2*: *bij-betw* g $(B - A)$ $(A - B)$
using *def-g*
unfolding *bij-betw-def inj-on-def*
by *simp*
hence g ‘ *UNIV* = *UNIV*
using *surj-cases-13 Un-Diff-cancel2 image-Un sup-top-left*
unfolding *bij-betw-def*
by *metis*
moreover have *inj g*
using *inj-cases-13 bij-case2 DiffD2 DiffI imageI surj-cases-13*
unfolding *bij-betw-def inj-def inj-on-def*
by *metis*
ultimately have *bij g*
unfolding *bij-def*
by *safe*
thus *?thesis*
using *coincide id reverse existence-witness*
by *blast*
qed

lemma *bij-betw-ext*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$X :: 'x\ set$ **and**
$Y :: 'y\ set$
assumes *bij-betw f X Y*
shows *bij-betw* (*extensional-continuation f X*) *X Y*
proof −
have $\forall\ x \in X.$ *extensional-continuation f X x = f x*
by *simp*
thus *?thesis*

using *assms bij-betw-cong*
by *metis*
qed

1.9.3 Anonymity Lemmas

lemma *anon-rel-vote-count*:
fixes
$\mathcal{E}$ :: $('a, 'v)$ *Election set* **and**
E :: $('a, 'v)$ *Election* **and**
E' :: $('a, 'v)$ *Election*
assumes
finite (*voters-$\mathcal{E}$* E) **and**
$(E, E') \in$ *anonymity*$_{\mathcal{R}}$ $\mathcal{E}$
shows *alternatives-$\mathcal{E}$* E = *alternatives-$\mathcal{E}$* $E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E}$
$\wedge$ ($\forall$ *p*. *vote-count p E* = *vote-count p E'*)
proof −
have $E \in \mathcal{E}$
using *assms*
unfolding *anonymity*$_{\mathcal{R}}$*.simps rel-induced-by-action.simps*
by *safe*
with *assms*
obtain π :: $'v \Rightarrow 'v$ **where**
bijection-π: *bij* π **and**
renamed: E' = *rename* π E
unfolding *anonymity*$_{\mathcal{R}}$*.simps anonymity*$_{\mathcal{G}}$*-def*
using *universal-set-carrier-imp-bij-group*
by *auto*
have *eq-alts*: *alternatives-$\mathcal{E}$* E' = *alternatives-$\mathcal{E}$* E
using *eq-fst-iff rename.simps alternatives-$\mathcal{E}$.elims renamed*
by (*metis* (*no-types*))
have $\forall$ $v \in$ *voters-$\mathcal{E}$* E'. (*profile-$\mathcal{E}$* E') v = (*profile-$\mathcal{E}$* E) (*the-inv* π v)
unfolding *profile-$\mathcal{E}$.simps*
using *renamed rename.simps comp-apply prod.collapse snd-conv*
by (*metis* (*no-types*, *lifting*))
hence *rewrite*:
$\forall$ *p*. {$v \in$ (*voters-$\mathcal{E}$* E'). (*profile-$\mathcal{E}$* E') $v = p$}
= {$v \in$ (*voters-$\mathcal{E}$* E'). (*profile-$\mathcal{E}$* E) (*the-inv* π v) = p}
by *blast*
have $\forall$ $v \in$ *voters-$\mathcal{E}$* E'. *the-inv* π $v \in$ *voters-$\mathcal{E}$* E
unfolding *voters-$\mathcal{E}$.simps*
using *renamed UNIV-I bijection-π bij-betw-imp-surj bij-is-inj f-the-inv-into-f*
prod.sel inj-image-mem-iff prod.collapse rename.simps
by (*metis* (*no-types*, *lifting*))
hence
$\forall$ *p*. $\forall$ $v \in$ *voters-$\mathcal{E}$* E'. (*profile-$\mathcal{E}$* E) (*the-inv* π v) = $p \longrightarrow$
$v \in \pi$ ‘ {$v \in$ *voters-$\mathcal{E}$* E. (*profile-$\mathcal{E}$* E) $v = p$}
using *bijection-π f-the-inv-into-f-bij-betw image-iff*
by *fastforce*

hence *subset*:
$\forall\ p.\ \{v \in voters\text{-}\mathcal{E}\ E'.\ (profile\text{-}\mathcal{E}\ E)\ (the\text{-}inv\ \pi\ v) = p\} \subseteq$
$\pi\ `\ \{v \in voters\text{-}\mathcal{E}\ E.\ (profile\text{-}\mathcal{E}\ E)\ v = p\}$
by *blast*
from *renamed* **have** $\forall\ v \in voters\text{-}\mathcal{E}\ E.\ \pi\ v \in voters\text{-}\mathcal{E}\ E'$
unfolding *voters-$\mathcal{E}$.simps*
using *bijection-π bij-is-inj prod.sel inj-image-mem-iff prod.collapse rename.simps*
by (*metis* (*mono-tags*, *lifting*))
hence
$\forall\ p.\ \pi\ `\ \{v \in voters\text{-}\mathcal{E}\ E.\ (profile\text{-}\mathcal{E}\ E)\ v = p\} \subseteq$
$\{v \in voters\text{-}\mathcal{E}\ E'.\ (profile\text{-}\mathcal{E}\ E)\ (the\text{-}inv\ \pi\ v) = p\}$
using *bijection-π bij-is-inj the-inv-f-f*
by *fastforce*
hence $\forall\ p.\ \{v \in voters\text{-}\mathcal{E}\ E'.\ (profile\text{-}\mathcal{E}\ E')\ v = p\} = \pi\ `\ \{v \in voters\text{-}\mathcal{E}\ E.\ (profile\text{-}\mathcal{E}\ E)\ v = p\}$
using *subset rewrite*
by (*simp add*: *subset-antisym*)
moreover have
$\forall\ p.\ card\ (\pi\ `\ \{v \in voters\text{-}\mathcal{E}\ E.\ (profile\text{-}\mathcal{E}\ E)\ v = p\})$
$= card\ \{v \in voters\text{-}\mathcal{E}\ E.\ (profile\text{-}\mathcal{E}\ E)\ v = p\}$
using *bijection-π bij-betw-same-card bij-betw-subset top-greatest*
by (*metis* (*no-types*, *lifting*))
ultimately show
$alternatives\text{-}\mathcal{E}\ E = alternatives\text{-}\mathcal{E}\ E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E} \wedge (\forall\ p.\ vote\text{-}count\ p\ E = vote\text{-}count\ p\ E')$
using *eq-alts assms*
by *simp*
qed

lemma *vote-count-anon-rel*:
fixes
$\mathcal{E}$ :: (*'a*, *'v*) *Election set* **and**
E :: (*'a*, *'v*) *Election* **and**
E' :: (*'a*, *'v*) *Election*
assumes
fin-voters-E: *finite* (*voters-$\mathcal{E}$ E*) **and**
fin-voters-E': *finite* (*voters-$\mathcal{E}$ E'*) **and**
default-non-v: $\forall\ v.\ v \notin voters\text{-}\mathcal{E}\ E \longrightarrow profile\text{-}\mathcal{E}\ E\ v = \{\}$ **and**
default-non-v': $\forall\ v.\ v \notin voters\text{-}\mathcal{E}\ E' \longrightarrow profile\text{-}\mathcal{E}\ E'\ v = \{\}$ **and**
eq: $alternatives\text{-}\mathcal{E}\ E = alternatives\text{-}\mathcal{E}\ E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E}$
$\wedge (\forall\ p.\ vote\text{-}count\ p\ E = vote\text{-}count\ p\ E')$
shows $(E, E') \in anonymity_{\mathcal{R}}\ \mathcal{E}$
proof –
have $\forall\ p.\ card\ \{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\} = card\ \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = p\}$
using *eq*
unfolding *vote-count.simps*
by *blast*
moreover have

$\forall\ p.\ finite\ \{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\}$
$\quad \wedge\ finite\ \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = p\}$
using *assms*
by *simp*
ultimately have
$\forall\ p.\ \exists\ \pi_p.\ bij\text{-}betw\ \pi_p\ \{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\}$
$\quad \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = p\}$
using *bij-betw-iff-card*
by *blast*
then obtain π :: $'a\ Preference\text{-}Relation \Rightarrow ('v \Rightarrow 'v)$ **where**
bij: $\forall\ p.\ bij\text{-}betw\ (\pi\ p)\ \{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\}$
$\quad \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = p\}$
by (*metis* (*no-types*))
obtain π' :: $'v \Rightarrow 'v$ **where**
π'*-def*: $\forall\ v \in voters\text{-}\mathcal{E}\ E.\ \pi'\ v = \pi\ (profile\text{-}\mathcal{E}\ E\ v)\ v$
by *fastforce*
hence $\forall\ v\ v'.\ v \in voters\text{-}\mathcal{E}\ E \wedge v' \in voters\text{-}\mathcal{E}\ E \longrightarrow$
$\pi'\ v = \pi'\ v' \longrightarrow \pi\ (profile\text{-}\mathcal{E}\ E\ v)\ v = \pi\ (profile\text{-}\mathcal{E}\ E\ v')\ v'$
by *simp*
moreover have
$\forall\ w\ w'.\ w \in voters\text{-}\mathcal{E}\ E \wedge w' \in voters\text{-}\mathcal{E}\ E \longrightarrow \pi\ (profile\text{-}\mathcal{E}\ E\ w)\ w = \pi\ (profile\text{-}\mathcal{E}\ E\ w')\ w' \longrightarrow$
$\{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = profile\text{-}\mathcal{E}\ E\ w\}$
$\cap\ \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = profile\text{-}\mathcal{E}\ E\ w'\} \neq \{\}$
using *bij*
unfolding *bij-betw-def*
by *blast*
moreover have
$\forall\ w\ w'.$
$\{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = profile\text{-}\mathcal{E}\ E\ w\}$
$\cap\ \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = profile\text{-}\mathcal{E}\ E\ w'\} \neq \{\}$
$\longrightarrow profile\text{-}\mathcal{E}\ E\ w = profile\text{-}\mathcal{E}\ E\ w'$
by *blast*
ultimately have *eq-prof*:
$\forall\ v\ v'.\ v \in voters\text{-}\mathcal{E}\ E \wedge v' \in voters\text{-}\mathcal{E}\ E \longrightarrow \pi'\ v = \pi'\ v' \longrightarrow profile\text{-}\mathcal{E}\ E\ v = profile\text{-}\mathcal{E}\ E\ v'$
by *presburger*
hence $\forall\ v\ v'.\ v \in voters\text{-}\mathcal{E}\ E \wedge v' \in voters\text{-}\mathcal{E}\ E \longrightarrow \pi'\ v = \pi'\ v' \longrightarrow$
$\quad \pi\ (profile\text{-}\mathcal{E}\ E\ v)\ v = \pi\ (profile\text{-}\mathcal{E}\ E\ v)\ v'$
using π'*-def*
by *metis*
hence $\forall\ v\ v'.\ v \in voters\text{-}\mathcal{E}\ E \wedge v' \in voters\text{-}\mathcal{E}\ E \longrightarrow \pi'\ v = \pi'\ v' \longrightarrow v = v'$
using *bij eq-prof*
unfolding *bij-betw-def inj-on-def*
by *simp*
hence *inj*: $inj\text{-}on\ \pi'\ (voters\text{-}\mathcal{E}\ E)$
unfolding *inj-on-def*
by *simp*
have $\pi'\ `\ voters\text{-}\mathcal{E}\ E = \{\pi\ (profile\text{-}\mathcal{E}\ E\ v)\ v \mid v.\ v \in voters\text{-}\mathcal{E}\ E\}$

using *π′-def*
unfolding *Setcompr-eq-image*
by *simp*
also have
... = $\bigcup$ {*π p* ‘ {*v* ∈ *voters-ℰ E*. *profile-ℰ E v* = *p*} | *p*. *p* ∈ *UNIV*}
unfolding *Union-eq*
by *blast*
also have
... = $\bigcup$ {{*v* ∈ *voters-ℰ E′*. *profile-ℰ E′ v* = *p*} | *p*. *p* ∈ *UNIV*}
using *bij*
unfolding *bij-betw-def*
by (*metis* (*mono-tags*, *lifting*))
finally have *π′* ‘ *voters-ℰ E* = *voters-ℰ E′*
by *blast*
with *inj* **have** *bij′*: *bij-betw π′* (*voters-ℰ E*) (*voters-ℰ E′*)
using *bij*
unfolding *bij-betw-def*
by *blast*
then obtain *π-global* :: *′v* ⇒ *′v* **where**
bijection-π$_g$: *bij π-global* **and**
π-global-def: ∀ *v* ∈ *voters-ℰ E*. *π-global v* = *π′ v* **and**
π-global-def′:
∀ *v* ∈ *voters-ℰ E′* − *voters-ℰ E*.
π-global v ∈ *voters-ℰ E* − *voters-ℰ E′* ∧
(∃ *n* > *0*. *n-app n π′* (*π-global v*) = *v*) **and**
π-global-non-voters: ∀ *v* ∈ *UNIV* − *voters-ℰ E* − *voters-ℰ E′*. *π-global v* = *v*
using *fin-voters-E fin-voters-E′ bij-betw-finite-ind-global-bij*
by *blast*
hence *inv*: ∀ *v v′*. (*π-global v′* = *v*) = (*v′* = *the-inv π-global v*)
using *UNIV-I bij-betw-imp-inj-on bij-betw-imp-surj-on f-the-inv-into-f the-inv-f-f*
by *metis*
moreover have
∀ *v* ∈ *UNIV* − (*voters-ℰ E′* − *voters-ℰ E*). *π-global v* ∈ *UNIV* − (*voters-ℰ E* − *voters-ℰ E′*)
using *π-global-def π-global-non-voters bij′ bijection-π*$_g$ *DiffD1 DiffD2 DiffI bij-betwE*
by (*metis* (*no-types*, *lifting*))
ultimately have ∀ *v* ∈ *voters-ℰ E* − *voters-ℰ E′*. *the-inv π-global v* ∈ *voters-ℰ E′* − *voters-ℰ E*
using *bijection-π*$_g$ *π-global-def′ DiffD2 DiffI UNIV-I*
by *metis*
hence ∀ *v* ∈ *voters-ℰ E* − *voters-ℰ E′*. ∀ *n* > *0*. *profile-ℰ E* (*the-inv π-global v*) = {}
using *default-non-v*
by *simp*
moreover have ∀ *v* ∈ *voters-ℰ E* − *voters-ℰ E′*. *profile-ℰ E′ v* = {}
using *default-non-v′*
by *simp*
ultimately have *case-1*:

$\forall\ v \in$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* $=$ (*profile-$\mathcal{E}$ E* $\circ$ *the-inv* π*-global*) *v*
by *auto*
have $\forall\ v \in$ *voters-$\mathcal{E}$ E'*. $\exists\ v' \in$ *voters-$\mathcal{E}$ E*. π*-global* $v' = v \wedge \pi'\ v' = v$
using *bij' imageE* π*-global-def*
unfolding *bij-betw-def*
by (*metis* (*mono-tags*, *opaque-lifting*))
hence $\forall\ v \in$ *voters-$\mathcal{E}$ E'*. $\exists\ v' \in$ *voters-$\mathcal{E}$ E*. $v' =$ *the-inv* π*-global* $v \wedge \pi'\ v' = v$
using *inv*
by *metis*
hence $\forall\ v \in$ *voters-$\mathcal{E}$ E'*. *the-inv* π*-global* $v \in$ *voters-$\mathcal{E}$ E* $\wedge\ \pi'$ (*the-inv* π*-global* *v*) $= v$
by *blast*
moreover have $\forall\ v' \in$ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E'* ($\pi'\ v'$) $=$ *profile-$\mathcal{E}$ E v'*
using π'*-def bij bij-betwE mem-Collect-eq*
by *fastforce*
ultimately have *case-2*: $\forall\ v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* $=$ (*profile-$\mathcal{E}$ E* $\circ$ *the-inv* π*-global*) *v*
unfolding *comp-def*
by *metis*
have $\forall\ v \in$ *UNIV* $-$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* $=$ (*profile-$\mathcal{E}$ E* $\circ$ *the-inv* π*-global*) *v*
using π*-global-non-voters default-non-v default-non-v' inv*
by *simp*
hence *profile-$\mathcal{E}$ E'* $=$ *profile-$\mathcal{E}$ E* $\circ$ *the-inv* π*-global*
using *case-1 case-2*
by *blast*
moreover have π*-global* ‘ (*voters-$\mathcal{E}$ E*) $=$ *voters-$\mathcal{E}$ E'*
using π*-global-def bij' bij-betw-imp-surj-on*
by *fastforce*
ultimately have $E' =$ *rename* π*-global E*
using *rename.simps eq prod.collapse*
unfolding *voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps alternatives-$\mathcal{E}$.simps*
by *metis*
thus *?thesis*
unfolding *extensional-continuation.simps anonymity$_{\mathcal{R}}$.simps*
rel-induced-by-action.simps φ-anon.simps anonymity$_{\mathcal{G}}$-def
using *eq bijection-π_g case-prodI rewrite-carrier*
by *auto*
qed

lemma *rename-comp*:
fixes
π :: $'v \Rightarrow 'v$ **and**
π' :: $'v \Rightarrow 'v$
assumes
bij π **and**
bij π'
shows *rename* $\pi \circ$ *rename* $\pi' =$ *rename* ($\pi \circ \pi'$)

proof
  fix E :: $('a, 'v)$ *Election*
  have *rename* π' E = (*alternatives-$\mathcal{E}$* E, π' ‘ (*voters-$\mathcal{E}$* E), (*profile-$\mathcal{E}$* E) $\circ$ (*the-inv* π'))
    unfolding *alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps*
    using *prod.collapse rename.simps*
    by *metis*
  hence
    (*rename* π $\circ$ *rename* π') E =
      rename π (*alternatives-$\mathcal{E}$* E, π' ‘ (*voters-$\mathcal{E}$* E), (*profile-$\mathcal{E}$* E) $\circ$ (*the-inv* π'))
    unfolding *comp-def*
    by *presburger*
  also have
    ... = (*alternatives-$\mathcal{E}$* E, π ‘ π' ‘ (*voters-$\mathcal{E}$* E), (*profile-$\mathcal{E}$* E) $\circ$ (*the-inv* π') $\circ$ (*the-inv* π))
    by *simp*
  also have ... = (*alternatives-$\mathcal{E}$* E, (π $\circ$ π') ‘ (*voters-$\mathcal{E}$* E), (*profile-$\mathcal{E}$* E) $\circ$ *the-inv* (π $\circ$ π'))
    using *assms the-inv-comp*[*of* π *UNIV UNIV* π']
    unfolding *comp-def image-image*
    by *simp*
  finally show (*rename* π $\circ$ *rename* π') E = *rename* (π $\circ$ π') E
    unfolding *alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps*
    using *prod.collapse rename.simps*
    by *metis*
qed

interpretation *anonymous-group-action*:
  group-action anonymity$_{\mathcal{G}}$ valid-elections φ-anon valid-elections
proof (*unfold group-action-def group-hom-def anonymity$_{\mathcal{G}}$-def group-hom-axioms-def hom-def*,
    safe, (*rule group-BijGroup*)+)
  show *bij-car-el*:
    $\bigwedge$ π. π $\in$ *carrier* (*BijGroup UNIV*) $\Longrightarrow$
      φ-anon valid-elections π $\in$ *carrier* (*BijGroup valid-elections*)
  proof −
    fix π :: $'v \Rightarrow 'v$
    assume π $\in$ *carrier* (*BijGroup UNIV*)
    hence *bij*: *bij* π
      using *rewrite-carrier*
      by *blast*
    hence *rename* π ‘ *valid-elections* = *valid-elections*
      using *rename-surj bij*
      by *blast*
    moreover have *inj-on* (*rename* π) *valid-elections*
      using *rename-inj bij subset-inj-on*
      by *blast*
    ultimately have *bij-betw* (*rename* π) *valid-elections valid-elections*
      unfolding *bij-betw-def*

by *blast*
hence *bij-betw* (φ*-anon valid-elections* π) *valid-elections valid-elections*
unfolding φ*-anon.simps extensional-continuation.simps*
using *bij-betw-ext*
by *simp*
moreover have φ*-anon valid-elections* $\pi \in$ *extensional valid-elections*
unfolding *extensional-def*
by *force*
ultimately show φ*-anon valid-elections* $\pi \in$ *carrier* (*BijGroup valid-elections*)
unfolding *BijGroup-def Bij-def*
by *simp*
qed
fix
π :: $'v \Rightarrow 'v$ **and**
π' :: $'v \Rightarrow 'v$
assume
bij: $\pi \in$ *carrier* (*BijGroup UNIV*) **and**
bij': $\pi' \in$ *carrier* (*BijGroup UNIV*)
hence *car-els*: φ*-anon valid-elections* $\pi \in$ *carrier* (*BijGroup valid-elections*) $\wedge$
φ*-anon valid-elections* $\pi' \in$ *carrier* (*BijGroup valid-elections*)
using *bij-car-el*
by *metis*
hence *bij-betw* (φ*-anon valid-elections* π') *valid-elections valid-elections*
unfolding *BijGroup-def Bij-def extensional-def*
by *auto*
hence *valid-closed'*: φ*-anon valid-elections* π' ‘ *valid-elections* $\subseteq$ *valid-elections*
using *bij-betw-imp-surj-on*
by *blast*
from *car-els*
have φ*-anon valid-elections* $\pi \otimes_{\textit{BijGroup valid-elections}}$ (φ*-anon valid-elections*) π' =
extensional-continuation
(φ*-anon valid-elections* $\pi \circ \varphi$*-anon valid-elections* π') *valid-elections*
using *rewrite-mult*
by *blast*
moreover have
$\forall$ *E*. *E* $\in$ *valid-elections* $\longrightarrow$
extensional-continuation
(φ*-anon valid-elections* $\pi \circ \varphi$*-anon valid-elections* π') *valid-elections E* =
(φ*-anon valid-elections* $\pi \circ \varphi$*-anon valid-elections* π') *E*
by *simp*
moreover have
$\forall$ *E*. *E* $\in$ *valid-elections* $\longrightarrow$
(φ*-anon valid-elections* $\pi \circ \varphi$*-anon valid-elections* π') *E* = *rename* π (*rename* π' *E*)
unfolding φ*-anon.simps*
using *valid-closed'*
by *auto*
moreover have $\forall$ *E*. *E* $\in$ *valid-elections* $\longrightarrow$ *rename* π (*rename* π' *E*) = *rename*

$(\pi \circ \pi')$ *E*
using *rename-comp bij bij' universal-set-carrier-imp-bij-group comp-apply*
by *metis*
moreover have
$\forall$ *E*. *E* $\in$ *valid-elections* $\longrightarrow$
rename $(\pi \circ \pi')$ *E* = φ*-anon valid-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ *E*
using *rewrite-mult-univ bij bij'*
unfolding φ*-anon.simps*
by *force*
moreover have
$\forall$ *E*. *E* $\notin$ *valid-elections* $\longrightarrow$
extensional-continuation
(φ*-anon valid-elections* $\pi \circ \varphi$*-anon valid-elections* π') *valid-elections* *E* = *undefined*
by *simp*
moreover have
$\forall$ *E*. *E* $\notin$ *valid-elections* $\longrightarrow$ φ*-anon valid-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ *E* = *undefined*
by *simp*
ultimately have
$\forall$ *E*. φ*-anon valid-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ *E* =
(φ*-anon valid-elections* $\pi \otimes_{\mathit{BijGroup\ valid\text{-}elections}}$ φ*-anon valid-elections* π') *E*
by *metis*
thus
φ*-anon valid-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ =
φ*-anon valid-elections* $\pi \otimes_{\mathit{BijGroup\ valid\text{-}elections}}$ φ*-anon valid-elections* π'
by *blast*
qed

lemma (**in** *result*) *well-formed-res-anon*:
satisfies (λ *E*. *limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) (*Invariance* (*anonymity*$_{\mathcal{R}}$ *valid-elections*))
proof (*unfold anonymity*$_{\mathcal{R}}$*.simps*, *clarsimp*) **qed**

1.9.4 Neutrality Lemmas

lemma *rel-rename-helper*:
fixes
r :: *'a rel* **and**
π :: *'a* $\Rightarrow$ *'a* **and**
a :: *'a* **and**
b :: *'a*
assumes *bij* π
shows $(\pi\ a, \pi\ b) \in \{(\pi\ x, \pi\ y) \mid x\ y.\ (x, y) \in r\} \longleftrightarrow (a, b) \in \{(x, y) \mid x\ y.\ (x, y) \in r\}$
proof (*safe*, *simp*)
fix
x :: *'a* **and**

$y :: 'a$
assume
$(x, y) \in r$ **and**
$\pi\ a = \pi\ x$ **and**
$\pi\ b = \pi\ y$
thus $(a, b) \in r$
using *assms bij-is-inj the-inv-f-f*
by *metis*
next
fix
$x :: 'a$ **and**
$y :: 'a$
assume $(a, b) \in r$
thus $\exists\ x\ y.\ (\pi\ a, \pi\ b) = (\pi\ x, \pi\ y) \wedge (x, y) \in r$
by *metis*
qed

lemma *rel-rename-comp*:
fixes
$\pi :: 'a \Rightarrow 'a$ **and**
$\pi' :: 'a \Rightarrow 'a$
shows *rel-rename* $(\pi \circ \pi') =$ *rel-rename* $\pi \circ$ *rel-rename* π'
proof
fix $r :: 'a\ rel$
have *rel-rename* $(\pi \circ \pi')\ r = \{(\pi\ (\pi'\ a), \pi\ (\pi'\ b)) \mid a\ b.\ (a, b) \in r\}$
by *simp*
also have ... $= \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in$ *rel-rename* $\pi'\ r\}$
unfolding *rel-rename.simps*
by *blast*
finally show *rel-rename* $(\pi \circ \pi')\ r = ($*rel-rename* $\pi \circ$ *rel-rename* $\pi')\ r$
unfolding *comp-def*
by *simp*
qed

lemma *rel-rename-sound*:
fixes
$\pi :: 'a \Rightarrow 'a$ **and**
$r :: 'a\ rel$ **and**
$A :: 'a\ set$
assumes *inj* π
shows
refl-on $A\ r \longrightarrow$ *refl-on* $(\pi\ `\ A)$ (*rel-rename* $\pi\ r$) **and**
antisym $r \longrightarrow$ *antisym* (*rel-rename* $\pi\ r$) **and**
total-on $A\ r \longrightarrow$ *total-on* $(\pi\ `\ A)$ (*rel-rename* $\pi\ r$) **and**
Relation.trans $r \longrightarrow$ *Relation.trans* (*rel-rename* $\pi\ r$)
proof (*unfold antisym-def total-on-def Relation.trans-def*, *safe*)
assume *refl-on* $A\ r$
thus *refl-on* $(\pi\ `\ A)$ (*rel-rename* $\pi\ r$)
unfolding *refl-on-def rel-rename.simps*

by *blast*
next
fix
a :: $'a$ **and**
b :: $'a$
assume
$(a, b) \in$ *rel-rename* π r **and**
$(b, a) \in$ *rel-rename* π r
then obtain
c :: $'a$ **and**
d :: $'a$ **and**
c' :: $'a$ **and**
d' :: $'a$ **where**
c-rel-d: $(c, d) \in r$ **and**
d'*-rel-*c': $(d', c') \in r$ **and**
π_c*-eq-a*: π $c = a$ **and**
$\pi_c{}'$*-eq-a*: π $c' = a$ **and**
π_d*-eq-b*: π $d = b$ **and**
$\pi_d{}'$*-eq-b*: π $d' = b$
unfolding *rel-rename.simps*
by *auto*
hence $c = c' \wedge d = d'$
using *assms*
unfolding *inj-def*
by *presburger*
moreover assume $\forall$ a b. $(a, b) \in r \longrightarrow (b, a) \in r \longrightarrow a = b$
ultimately have $c = d$
using d'*-rel-*c' *c-rel-d*
by *simp*
thus $a = b$
using π_c*-eq-a* π_d*-eq-b*
by *simp*
next
fix
a :: $'a$ **and**
b :: $'a$
assume
total: $\forall$ $x \in A$. $\forall$ $y \in A$. $x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r$ **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
π_a*-neq-*π_b: π $a \neq \pi$ b **and**
π_b*-not-rel-*π_a: $(\pi$ b, π $a) \notin$ *rel-rename* π r
hence $(b, a) \notin r \wedge a \neq b$
unfolding *rel-rename.simps*
by *blast*
hence $(a, b) \in r$
using *a-in-A* *b-in-A* *total*
by *blast*
thus $(\pi$ a, π $b) \in$ *rel-rename* π r

unfolding *rel-rename.simps*
by *blast*
next
fix
a :: $'a$ **and**
b :: $'a$ **and**
c :: $'a$
assume
$(a,\ b) \in$ *rel-rename* $\pi\ r$ **and**
$(b,\ c) \in$ *rel-rename* $\pi\ r$
then obtain
d :: $'a$ **and**
e :: $'a$ **and**
s :: $'a$ **and**
t :: $'a$ **where**
d-rel-e: $(d,\ e) \in r$ **and**
s-rel-t: $(s,\ t) \in r$ **and**
π_d*-eq-a*: $\pi\ d = a$ **and**
π_s*-eq-b*: $\pi\ s = b$ **and**
π_t*-eq-c*: $\pi\ t = c$ **and**
π_e*-eq-b*: $\pi\ e = b$
unfolding *alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps*
using *rel-rename.simps Pair-inject mem-Collect-eq*
by *auto*
hence $s = e$
using *assms rangeI range-ex1-eq*
by *metis*
hence $(d,\ e) \in r \wedge (e,\ t) \in r$
using *d-rel-e s-rel-t*
by *simp*
moreover assume $\forall\ x\ y\ z.\ (x,\ y) \in r \longrightarrow (y,\ z) \in r \longrightarrow (x,\ z) \in r$
ultimately have $(d,\ t) \in r$
by *blast*
thus $(a,\ c) \in$ *rel-rename* $\pi\ r$
unfolding *rel-rename.simps*
using π_d*-eq-a* π_t*-eq-c*
by *blast*
qed

lemma *rel-rename-bij*:
fixes π :: $'a \Rightarrow 'a$
assumes *bij-π*: *bij* π
shows *bij* (*rel-rename* π)
proof (*unfold bij-def inj-def surj-def*, *safe*)
show *subset*:
$\bigwedge\ r\ s\ a\ b.$ *rel-rename* $\pi\ r =$ *rel-rename* $\pi\ s \Longrightarrow (a,\ b) \in r \Longrightarrow (a,\ b) \in s$
proof −
fix
r :: $'a$ *rel* **and**

s :: $'a$ *rel* **and**
a :: $'a$ **and**
b :: $'a$
assume
rel-rename π r = *rel-rename* π s **and**
$(a, b) \in r$
hence $(\pi\ a, \pi\ b) \in \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in s\}$
unfolding *rel-rename.simps*
by *blast*
hence $\exists\ c\ d.\ (c, d) \in s \wedge \pi\ c = \pi\ a \wedge \pi\ d = \pi\ b$
by *fastforce*
moreover have $\forall\ c\ d.\ \pi\ c = \pi\ d \longrightarrow c = d$
using *bij-π bij-pointE*
by *metis*
ultimately show $(a, b) \in s$
by *blast*
qed
fix
r :: $'a$ *rel* **and**
s :: $'a$ *rel* **and**
a :: $'a$ **and**
b :: $'a$
assume
rel-rename π r = *rel-rename* π s **and**
$(a, b) \in s$
thus $(a, b) \in r$
using *subset*
by *presburger*
next
fix r :: $'a$ *rel*
have *rel-rename* π $\{((\textit{the-inv}\ \pi)\ a, (\textit{the-inv}\ \pi)\ b) \mid a\ b.\ (a, b) \in r\}$ =
$\{(\pi\ ((\textit{the-inv}\ \pi)\ a), \pi\ ((\textit{the-inv}\ \pi)\ b)) \mid a\ b.\ (a, b) \in r\}$
by *auto*
also have ... = $\{(a, b) \mid a\ b.\ (a, b) \in r\}$
using *the-inv-f-f bij-π*
by (*simp add*: *f-the-inv-into-f-bij-betw*)
finally have *rel-rename* π (*rel-rename* (*the-inv* π) r) = r
by *simp*
thus $\exists\ s.\ r$ = *rel-rename* π s
by *blast*
qed

lemma *alternatives-rename-comp*:
fixes
π :: $'a \Rightarrow 'a$ **and**
π' :: $'a \Rightarrow 'a$
shows *alternatives-rename* $\pi \circ$ *alternatives-rename* π' = *alternatives-rename* $(\pi \circ \pi')$
proof

fix $\mathcal{E}$ :: ($'a$, $'v$) *Election*
have (*alternatives-rename* π $\circ$ *alternatives-rename* π') $\mathcal{E}$
= (π ‘ π' ‘ (*alternatives-$\mathcal{E}$* $\mathcal{E}$), *voters-$\mathcal{E}$* $\mathcal{E}$, (*rel-rename* π) $\circ$ (*rel-rename* π') $\circ$ (*profile-$\mathcal{E}$* $\mathcal{E}$))
by (*simp add*: *fun.map-comp*)
also have
... = (($\pi \circ \pi'$) ‘ (*alternatives-$\mathcal{E}$* $\mathcal{E}$), *voters-$\mathcal{E}$* $\mathcal{E}$, (*rel-rename* ($\pi \circ \pi'$)) $\circ$ (*profile-$\mathcal{E}$* $\mathcal{E}$))
using *rel-rename-comp image-comp*
by *metis*
also have ... = *alternatives-rename* ($\pi \circ \pi'$) $\mathcal{E}$
by *simp*
finally show (*alternatives-rename* π $\circ$ *alternatives-rename* π') $\mathcal{E}$ = *alternatives-rename* ($\pi \circ \pi'$) $\mathcal{E}$
by *blast*
qed

lemma *alternatives-rename-bij*:
fixes π :: ($'a \Rightarrow 'a$)
assumes *bij-π*: *bij* π
shows *bij-betw* (*alternatives-rename* π) *valid-elections valid-elections*
proof (*unfold bij-betw-def*, *safe*, *intro inj-onI*, *clarsimp*)
fix
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
p' :: ($'a$, $'v$) *Profile*
assume
(A, V, p) $\in$ *valid-elections* **and**
(A', V, p') $\in$ *valid-elections* **and**
π-eq-img-A-A': π ‘ A = π ‘ A' **and**
rel-rename π $\circ$ p = *rel-rename* π $\circ$ p'
hence
(*the-inv* (*rel-rename* π)) $\circ$ *rel-rename* π $\circ$ p = (*the-inv* (*rel-rename* π)) $\circ$ *rel-rename* π $\circ$ p'
using *fun.map-comp*
by *metis*
also have (*the-inv* (*rel-rename* π)) $\circ$ *rel-rename* π = *id*
using *bij-π rel-rename-bij inv-o-cancel surj-imp-inv-eq the-inv-f-f*
unfolding *bij-betw-def*
by (*metis* (*no-types*, *opaque-lifting*))
finally have $p = p'$
by *simp*
thus $A = A' \wedge p = p'$
using *bij-π π-eq-img-A-A' bij-betw-imp-inj-on inj-image-eq-iff*
by *metis*
next
fix

```
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile
assume valid-elects: (A, V, p) ∈ valid-elections
have valid-elects-closed:
  ⋀ A' V' p' π.
    bij π ⟹ (A', V', p') = alternatives-rename π (A, V, p) ⟹
      (A', V', p') ∈ valid-elections
proof −
  fix
    A' :: 'a set and
    V' :: 'v set and
    p' :: ('a, 'v) Profile and
    π :: 'a ⇒ 'a
  assume renamed: (A', V', p') = alternatives-rename π (A, V, p)
  hence rewr: V = V' ∧ A' = π ` A
    by simp
  hence ∀ v ∈ V'. linear-order-on A (p v)
    using valid-elects
    unfolding valid-elections-def profile-def
    by simp
  moreover have ∀ v ∈ V'. p' v = rel-rename π (p v)
    using renamed
    by simp
  moreover assume bij-π: bij π
  ultimately have ∀ v ∈ V'. linear-order-on A' (p' v)
    unfolding linear-order-on-def partial-order-on-def preorder-on-def
    using rewr rel-rename-sound bij-is-inj
    by metis
  thus (A', V', p') ∈ valid-elections
    unfolding valid-elections-def profile-def
    by simp
qed
thus ⋀ A' V' p'.
        (A', V', p') = alternatives-rename π (A, V, p) ⟹
          (A, V, p) ∈ valid-elections ⟹ (A', V', p') ∈ valid-elections
  using bij-π valid-elects
  by blast
have alternatives-rename (the-inv π) (A, V, p)
        = ((the-inv π) ` A, V, rel-rename (the-inv π) ∘ p)
  by simp
also have
  alternatives-rename π ((the-inv π) ` A, V, rel-rename (the-inv π) ∘ p) =
    (π ` (the-inv π) ` A, V, rel-rename π ∘ rel-rename (the-inv π) ∘ p)
  by auto
also have ... = (A, V, rel-rename (π ∘ the-inv π) ∘ p)
  using bij-π rel-rename-comp[of π] the-inv-f-f
  by (simp add: bij-betw-imp-surj-on bij-is-inj f-the-inv-into-f image-comp)
also have (A, V, rel-rename (π ∘ the-inv π) ∘ p) = (A, V, rel-rename id ∘ p)
```

using *UNIV-I assms comp-apply f-the-inv-into-f-bij-betw id-apply*
by *metis*
finally have *alternatives-rename* π (*alternatives-rename* (*the-inv* π) (A, V, p)) = (A, V, p)
unfolding *rel-rename.simps*
by *auto*
moreover have *alternatives-rename* (*the-inv* π) (A, V, p) $\in$ *valid-elections*
using *valid-elects-closed bij-*π
by (*simp add*: *bij-betw-the-inv-into valid-elects*)
ultimately show (A, V, p) $\in$ *alternatives-rename* π ‘ *valid-elections*
using *image-eqI*
by *metis*
qed

interpretation *φ-neutr-act*:
group-action neutrality$_{\mathcal{G}}$ valid-elections φ-neutr valid-elections
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def neutrality$_{\mathcal{G}}$-def*,
safe, (*rule group-BijGroup*)+)
show *bij-car-el*:
$\bigwedge \pi.\ \pi \in$ *carrier* (*BijGroup UNIV*) $\Longrightarrow$
φ-neutr valid-elections $\pi \in$ *carrier* (*BijGroup valid-elections*)
proof −
fix $\pi :: 'c \Rightarrow 'c$
assume $\pi \in$ *carrier* (*BijGroup UNIV*)
hence *bij-betw* (*φ-neutr valid-elections* π) *valid-elections valid-elections*
using *universal-set-carrier-imp-bij-group*
unfolding *φ-neutr.simps*
using *alternatives-rename-bij bij-betw-ext*
by *metis*
thus *φ-neutr valid-elections* $\pi \in$ *carrier* (*BijGroup valid-elections*)
unfolding *φ-neutr.simps BijGroup-def Bij-def extensional-def*
by *simp*
qed
fix
$\pi :: 'a \Rightarrow 'a$ **and**
$\pi' :: 'a \Rightarrow 'a$
assume
bij: $\pi \in$ *carrier* (*BijGroup UNIV*) **and**
bij': $\pi' \in$ *carrier* (*BijGroup UNIV*)
hence *car-els*: *φ-neutr valid-elections* $\pi \in$ *carrier* (*BijGroup valid-elections*) $\wedge$
φ-neutr valid-elections $\pi' \in$ *carrier* (*BijGroup valid-elections*)
using *bij-car-el*
by *metis*
hence *bij-betw* (*φ-neutr valid-elections* π') *valid-elections valid-elections*
unfolding *BijGroup-def Bij-def extensional-def*
by *auto*
hence *valid-closed'*: *φ-neutr valid-elections* π' ‘ *valid-elections* $\subseteq$ *valid-elections*
using *bij-betw-imp-surj-on*

by *blast*
have *φ-neutr valid-elections* π ⊗ *BijGroup valid-elections* *φ-neutr valid-elections* π′ =
extensional-continuation
(*φ-neutr valid-elections* π ∘ *φ-neutr valid-elections* π′) *valid-elections*
using *car-els rewrite-mult*
by *auto*
moreover have
∀ ℰ. ℰ ∈ *valid-elections* ⟶
extensional-continuation
(*φ-neutr valid-elections* π ∘ *φ-neutr valid-elections* π′) *valid-elections* ℰ =
(*φ-neutr valid-elections* π ∘ *φ-neutr valid-elections* π′) ℰ
by *simp*
moreover have
∀ ℰ. ℰ ∈ *valid-elections* ⟶
(*φ-neutr valid-elections* π ∘ *φ-neutr valid-elections* π′) ℰ =
alternatives-rename π (*alternatives-rename* π′ ℰ)
unfolding *φ-neutr.simps*
using *valid-closed′*
by *auto*
moreover have
∀ ℰ. ℰ ∈ *valid-elections*
⟶ *alternatives-rename* π (*alternatives-rename* π′ ℰ) = *alternatives-rename* (π ∘ π′) ℰ
using *alternatives-rename-comp bij bij′ comp-apply*
by *metis*
moreover have
∀ ℰ. ℰ ∈ *valid-elections* ⟶ *alternatives-rename* (π ∘ π′) ℰ =
φ-neutr valid-elections (π ⊗ *BijGroup UNIV* π′) ℰ
using *rewrite-mult-univ bij bij′*
unfolding *φ-anon.simps*
by *force*
moreover have
∀ ℰ. ℰ ∉ *valid-elections* ⟶
extensional-continuation
(*φ-neutr valid-elections* π ∘ *φ-neutr valid-elections* π′) *valid-elections* ℰ = *undefined*
by *simp*
moreover have
∀ ℰ. ℰ ∉ *valid-elections* ⟶ *φ-neutr valid-elections* (π ⊗ *BijGroup UNIV* π′) ℰ = *undefined*
by *simp*
ultimately have
∀ ℰ. *φ-neutr valid-elections* (π ⊗ *BijGroup UNIV* π′) ℰ =
(*φ-neutr valid-elections* π ⊗ *BijGroup valid-elections* *φ-neutr valid-elections* π′) ℰ
by *metis*
thus
φ-neutr valid-elections (π ⊗ *BijGroup UNIV* π′) =

φ*-neutr valid-elections* $\pi \otimes_{\ BijGroup\ valid\text{-}elections}$ φ*-neutr valid-elections* π'
  by *blast*
qed

interpretation ψ*-neutr*$_\mathrm{c}$*-act*: *group-action neutrality*$_\mathcal{G}$ *UNIV* ψ*-neutr*$_\mathrm{c}$
proof (*unfold group-action-def group-hom-def hom-def neutrality*$_\mathcal{G}$*-def group-hom-axioms-def*,

    safe, (*rule group-BijGroup*)+)
 fix $\pi :: 'a \Rightarrow 'a$
 assume $\pi \in$ *carrier* (*BijGroup UNIV*)
 hence *bij* π
  unfolding *BijGroup-def Bij-def*
  by *simp*
 thus ψ*-neutr*$_\mathrm{c}$ $\pi \in$ *carrier* (*BijGroup UNIV*)
  unfolding ψ*-neutr*$_\mathrm{c}$*.simps*
  using *rewrite-carrier*
  by *blast*
next
 fix
  $\pi :: 'a \Rightarrow 'a$ **and**
  $\pi' :: 'a \Rightarrow 'a$
 show ψ*-neutr*$_\mathrm{c}$ $(\pi \otimes_{\ BijGroup\ UNIV}\ \pi') =$
     ψ*-neutr*$_\mathrm{c}$ $\pi \otimes_{\ BijGroup\ UNIV}$ ψ*-neutr*$_\mathrm{c}$ π'
  unfolding ψ*-neutr*$_\mathrm{c}$*.simps*
  by *simp*
qed

interpretation ψ*-neutr*$_\mathrm{w}$*-act*: *group-action neutrality*$_\mathcal{G}$ *UNIV* ψ*-neutr*$_\mathrm{w}$
proof (*unfold group-action-def group-hom-def hom-def neutrality*$_\mathcal{G}$*-def group-hom-axioms-def*,

    safe, (*rule group-BijGroup*)+)
 show *group-elem*:
  $\bigwedge \pi.\ \pi \in$ *carrier* (*BijGroup UNIV*) $\Longrightarrow$ ψ*-neutr*$_\mathrm{w}$ $\pi \in$ *carrier* (*BijGroup UNIV*)
 proof −
  fix $\pi :: 'c \Rightarrow 'c$
  assume $\pi \in$ *carrier* (*BijGroup UNIV*)
  hence *bij* π
   unfolding *neutrality*$_\mathcal{G}$*-def BijGroup-def Bij-def*
   by *simp*
  hence *bij* (ψ*-neutr*$_\mathrm{w}$ π)
   unfolding *neutrality*$_\mathcal{G}$*-def BijGroup-def Bij-def* ψ*-neutr*$_\mathrm{w}$*.simps*
   using *rel-rename-bij*
   by *blast*
  thus ψ*-neutr*$_\mathrm{w}$ $\pi \in$ *carrier* (*BijGroup UNIV*)
   using *rewrite-carrier*
   by *blast*
 qed
 fix
  $\pi :: 'a \Rightarrow 'a$ **and**

$\pi' :: 'a \Rightarrow 'a$
assume
$\pi \in$ *carrier* (*BijGroup UNIV*) **and**
$\pi' \in$ *carrier* (*BijGroup UNIV*)
moreover from *this* **have**
ψ-*neutr*$_\mathrm{w}$ $\pi \in$ *carrier* (*BijGroup UNIV*) $\wedge$ ψ-*neutr*$_\mathrm{w}$ $\pi' \in$ *carrier* (*BijGroup UNIV*)
using *group-elem*
by *blast*
ultimately show ψ-*neutr*$_\mathrm{w}$ ($\pi \otimes_{BijGroup\ UNIV} \pi'$) = ψ-*neutr*$_\mathrm{w}$ $\pi \otimes_{BijGroup\ UNIV}$ ψ-*neutr*$_\mathrm{w}$ π'
unfolding ψ-*neutr*$_\mathrm{w}$.*simps*
using *rel-rename-comp rewrite-mult-univ*
by *metis*
qed

lemma *wf-result-neutrality-*$\mathcal{SCF}$:
satisfies (λ $\mathcal{E}$. *limit-set-*$\mathcal{SCF}$ (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*equivar-ind-by-act* (*carrier neutrality*$_\mathcal{G}$) *valid-elections*
(φ-*neutr valid-elections*) (*set-action* ψ-*neutr*$_\mathrm{c}$))
proof (*unfold rewrite-equivar-ind-by-act*, *safe*, *auto*) **qed**

lemma *wf-result-neutrality-*$\mathcal{SWF}$:
satisfies (λ $\mathcal{E}$. *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*equivar-ind-by-act* (*carrier neutrality*$_\mathcal{G}$) *valid-elections*
(φ-*neutr valid-elections*) (*set-action* ψ-*neutr*$_\mathrm{w}$))
proof (*unfold rewrite-equivar-ind-by-act voters-*$\mathcal{E}$.*simps profile-*$\mathcal{E}$.*simps set-action.simps*, *safe*)
show *lim-el-*π:
$\bigwedge$ π *A V p r*. $\pi \in$ *carrier neutrality*$_\mathcal{G}$ $\Longrightarrow$ (*A*, *V*, *p*) $\in$ *valid-elections* $\Longrightarrow$
φ-*neutr valid-elections* π (*A*, *V*, *p*) $\in$ *valid-elections* $\Longrightarrow$
r $\in$ *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (φ-*neutr valid-elections* π (*A*, *V*, *p*))) *UNIV* $\Longrightarrow$
r $\in$ ψ-*neutr*$_\mathrm{w}$ π ‘ *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (*A*, *V*, *p*)) *UNIV*
proof –
fix
$\pi :: 'c \Rightarrow 'c$ **and**
$A :: 'c$ *set* **and**
$V :: 'v$ *set* **and**
$p :: ('c, 'v)$ *Profile* **and**
$r :: 'c$ *rel*
let *?r-inv* = ψ-*neutr*$_\mathrm{w}$ (*the-inv* π) *r*
assume
carrier-π: $\pi \in$ *carrier neutrality*$_\mathcal{G}$ **and**
prof: (*A*, *V*, *p*) $\in$ *valid-elections* **and**
φ-*neutr valid-elections* π (*A*, *V*, *p*) $\in$ *valid-elections* **and**
lim-el: *r* $\in$ *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (φ-*neutr valid-elections* π (*A*, *V*, *p*))) *UNIV*
hence *inv-carrier*: *the-inv* $\pi \in$ *carrier neutrality*$_\mathcal{G}$

unfolding *neutrality*$_{\mathcal{G}}$*-def rewrite-carrier*
using *bij-betw-the-inv-into*
by *simp*
moreover have *the-inv* $\pi \circ \pi = id$
using *carrier-*π *universal-set-carrier-imp-bij-group bij-is-inj the-inv-f-f*
unfolding *neutrality*$_{\mathcal{G}}$*-def*
by *fastforce*
moreover have $\mathbf{1}$ $_{neutrality_{\mathcal{G}}}$ $= id$
unfolding *neutrality*$_{\mathcal{G}}$*-def BijGroup-def*
by *auto*
ultimately have *the-inv* $\pi \otimes_{neutrality_{\mathcal{G}}} \pi = \mathbf{1}_{neutrality_{\mathcal{G}}}$
using *carrier-*π
unfolding *neutrality*$_{\mathcal{G}}$*-def*
using *rewrite-mult-univ*
by *metis*
hence *inv-eq*: *inv* $_{neutrality_{\mathcal{G}}}$ $\pi =$ *the-inv* π
using *carrier-*π *inv-carrier* ψ*-neutr*$_{\mathrm{c}}$*-act.group-hom group.inv-closed group.inv-solve-right group.l-inv group-BijGroup group-hom.hom-one group-hom.one-closed*
unfolding *neutrality*$_{\mathcal{G}}$*-def*
by *metis*
have $r \in$ *limit-set-*$\mathcal{SWF}$ (π ‘ A) *UNIV*
unfolding φ*-neutr.simps*
using *prof lim-el*
by *simp*
hence *lin*: *linear-order-on* (π ‘ A) r
by *auto*
have *bij-inv*: *bij* (*the-inv* π)
using *carrier-*π *bij-betw-the-inv-into universal-set-carrier-imp-bij-group*
unfolding *neutrality*$_{\mathcal{G}}$*-def*
by *blast*
hence (*the-inv* π) ‘ π ‘ $A = A$
using *carrier-*π *UNIV-I bij-betw-imp-surj universal-set-carrier-imp-bij-group f-the-inv-into-f-bij-betw image-f-inv-f surj-imp-inv-eq*
unfolding *neutrality*$_{\mathcal{G}}$*-def*
by *metis*
hence *lin-inv*: *linear-order-on* A *?r-inv*
using *rel-rename-sound bij-inv lin bij-is-inj*
unfolding ψ*-neutr*$_{\mathrm{w}}$*.simps linear-order-on-def preorder-on-def partial-order-on-def*
by *metis*
hence $\forall\ a\ b.\ (a, b) \in$ *?r-inv* $\longrightarrow a \in A \land b \in A$
using *linear-order-on-def partial-order-onD*(*1*) *refl-on-def*
by *blast*
hence *limit* A *?r-inv* $= \{(a, b).\ (a, b) \in$ *?r-inv*$\}$
by *auto*
also have ... = *?r-inv*
by *blast*
finally have ... = *limit* A *?r-inv*
by *blast*
hence *?r-inv* $\in$ *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*

unfolding *limit-set-$\mathcal{SWF}$.simps*
using *lin-inv UNIV-I fst-conv mem-Collect-eq alternatives-$\mathcal{E}$.elims iso-tuple-UNIV-I CollectI*
by (*metis* (*mono-tags*, *lifting*))
moreover have $r = \psi$*-neutr*$_{\mathrm{w}}$ π *?r-inv*
using *carrier-π inv-eq inv-carrier iso-tuple-UNIV-I ψ-neutr$_{\mathrm{w}}$-act.orbit-sym-aux*
by *metis*
ultimately show $r \in \psi$*-neutr*$_{\mathrm{w}}$ π ‘ *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* (A, V, p)) *UNIV*
by *blast*
qed
fix
π :: $'a \Rightarrow 'a$ **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
r :: $'a$ *rel*
let *?r-inv* = ψ*-neutr*$_{\mathrm{w}}$ (*the-inv* π) r
assume
carrier-π: $\pi \in$ *carrier neutrality*$_{\mathcal{G}}$ **and**
prof: (A, V, p) $\in$ *valid-elections* **and**
prof-π: φ*-neutr valid-elections* π (A, V, p) $\in$ *valid-elections* **and**
$r \in$ *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* (A, V, p)) *UNIV*
hence
$r \in$ *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* (φ*-neutr valid-elections* (*inv* $_{neutrality_{\mathcal{G}}}$ π)
(φ*-neutr valid-elections* π (A, V, p))))) *UNIV*
using *φ-neutr-act.orbit-sym-aux*
by *metis*
moreover have *inv-group-elem*: *inv* $_{neutrality_{\mathcal{G}}}$ $\pi \in$ *carrier neutrality*$_{\mathcal{G}}$
using *carrier-π ψ-neutr$_{\mathrm{c}}$-act.group-hom group.inv-closed group-hom-def*
by *metis*
moreover have
φ*-neutr valid-elections* (*inv* $_{neutrality_{\mathcal{G}}}$ π)
(φ*-neutr valid-elections* π (A, V, p)) $\in$ *valid-elections*
using *prof φ-neutr-act.element-image inv-group-elem prof-π*
by *metis*
ultimately have
$r \in \psi$*-neutr*$_{\mathrm{w}}$ (*inv* $_{neutrality_{\mathcal{G}}}$ π) ‘
limit-set-$\mathcal{SWF}$ (*alternatives-$\mathcal{E}$* (φ*-neutr valid-elections* π (A, V, p))) *UNIV*
using *prof-π lim-el-π prod.collapse*
by *metis*
thus
ψ*-neutr*$_{\mathrm{w}}$ π $r \in$ *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* (φ*-neutr valid-elections* π (A, V, p))) *UNIV*
using *carrier-π ψ-neutr$_{\mathrm{w}}$-act.group-action-axioms ψ-neutr$_{\mathrm{w}}$-act.inj-prop group-action.orbit-sym-aux inj-image-mem-iff inv-group-elem iso-tuple-UNIV-I*

by (*metis* (*no-types*, *lifting*))
qed

1.9.5 Homogeneity Lemmas

lemma *refl-homogeneity*$_{\mathcal{R}}$:
fixes $\mathcal{E}$ :: (*'a*, *'v*) *Election set*
assumes $\mathcal{E} \subseteq$ *finite-voter-elections*
shows *refl-on* $\mathcal{E}$ (*homogeneity*$_{\mathcal{R}}$ $\mathcal{E}$)
using *assms*
unfolding *refl-on-def finite-voter-elections-def*
by *auto*

lemma (**in** *result*) *well-formed-res-homogeneity*:
satisfies (λ $\mathcal{E}$. *limit-set* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*) (*Invariance* (*homogeneity*$_{\mathcal{R}}$ *UNIV*))
by *simp*

lemma *refl-homogeneity*$_{\mathcal{R}}$*'*:
fixes $\mathcal{E}$ :: (*'a*, *'v*::*linorder*) *Election set*
assumes $\mathcal{E} \subseteq$ *finite-voter-elections*
shows *refl-on* $\mathcal{E}$ (*homogeneity*$_{\mathcal{R}}$*'* $\mathcal{E}$)
using *assms*
unfolding *homogeneity*$_{\mathcal{R}}$*'.simps refl-on-def finite-voter-elections-def*
by *auto*

lemma (**in** *result*) *well-formed-res-homogeneity'*:
satisfies (λ $\mathcal{E}$. *limit-set* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*) (*Invariance* (*homogeneity*$_{\mathcal{R}}$*'* *UNIV*))
by *simp*

1.9.6 Reversal Symmetry Lemmas

lemma *rev-rev-id*: *rev-rel* $\circ$ *rev-rel* = *id*
by *auto*

lemma *rev-rel-limit*:
fixes
A :: *'a set* **and**
r :: *'a rel*
shows *rev-rel* (*limit A r*) = *limit A* (*rev-rel r*)
unfolding *rev-rel.simps limit.simps*
by *blast*

lemma *rev-rel-lin-ord*:
fixes
A :: *'a set* **and**
r :: *'a rel*
assumes *linear-order-on A r*
shows *linear-order-on A* (*rev-rel r*)

using *assms*
unfolding *rev-rel.simps linear-order-on-def partial-order-on-def total-on-def antisym-def preorder-on-def refl-on-def trans-def*
by *blast*

interpretation $\mathit{reversal}_{\mathcal{G}}$*-group*: *group* $\mathit{reversal}_{\mathcal{G}}$
proof
show $\mathbf{1}\ _{\mathit{reversal}_{\mathcal{G}}} \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*
by *simp*
next
show $\mathit{carrier}\ \mathit{reversal}_{\mathcal{G}} \subseteq \mathit{Units}\ \mathit{reversal}_{\mathcal{G}}$
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def Units-def*
using *rev-rev-id*
by *auto*
next
fix $\alpha :: {'a}\ \mathit{rel} \Rightarrow {'a}\ \mathit{rel}$
show $\alpha \otimes_{\mathit{reversal}_{\mathcal{G}}} \mathbf{1}\ _{\mathit{reversal}_{\mathcal{G}}} = \alpha$
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*
by *auto*
assume α*-elem*: $\alpha \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
thus $\mathbf{1}\ _{\mathit{reversal}_{\mathcal{G}}} \otimes_{\mathit{reversal}_{\mathcal{G}}} \alpha = \alpha$
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*
by *auto*
fix $\alpha' :: {'a}\ \mathit{rel} \Rightarrow {'a}\ \mathit{rel}$
assume α'*-elem*: $\alpha' \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
thus $\alpha \otimes_{\mathit{reversal}_{\mathcal{G}}} \alpha' \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
using α*-elem rev-rev-id*
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*
by *auto*
fix $z :: {'a}\ \mathit{rel} \Rightarrow {'a}\ \mathit{rel}$
assume $z \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
thus $\alpha \otimes_{\mathit{reversal}_{\mathcal{G}}} \alpha' \otimes_{\mathit{reversal}_{\mathcal{G}}} z = \alpha \otimes_{\mathit{reversal}_{\mathcal{G}}} (\alpha' \otimes_{\mathit{reversal}_{\mathcal{G}}} z)$
using α*-elem* α'*-elem*
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*
by *auto*
qed

interpretation φ*-rev-act*: *group-action* $\mathit{reversal}_{\mathcal{G}}$ *valid-elections* φ*-rev valid-elections*
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*, *safe*, *rule group-BijGroup*)
show *car-el*:
$\bigwedge \pi.\ \pi \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}} \Longrightarrow \varphi\text{-}\mathit{rev}\ \mathit{valid\text{-}elections}\ \pi \in \mathit{carrier}\ (\mathit{BijGroup}\ \mathit{valid\text{-}elections})$
proof −
fix $\pi :: {'c}\ \mathit{rel} \Rightarrow {'c}\ \mathit{rel}$
assume $\pi \in \mathit{carrier}\ \mathit{reversal}_{\mathcal{G}}$
hence π*-cases*: $\pi \in \{\mathit{id}, \mathit{rev\text{-}rel}\}$
unfolding $\mathit{reversal}_{\mathcal{G}}$*-def*

by *auto*
hence *inv-rel-app*: *rel-app* $\pi \circ$ *rel-app* $\pi =$ *id*
using *rev-rev-id*
by *fastforce*
have *id*: $\forall\ \mathcal{E}$. *rel-app* π (*rel-app* $\pi\ \mathcal{E}$) $= \mathcal{E}$
by (*simp add*: *inv-rel-app pointfree-idE*)
have $\forall\ \mathcal{E} \in$ *valid-elections*. *rel-app* $\pi\ \mathcal{E} \in$ *valid-elections*
unfolding *valid-elections-def profile-def*
using π*-cases rev-rel-lin-ord rel-app.simps fun.map-id*
by *fastforce*
hence *rel-app* π ‘ *valid-elections* $\subseteq$ *valid-elections*
by *blast*
with *id* **have** *bij-betw* (*rel-app* π) *valid-elections valid-elections*
using *bij-betw-byWitness*[*of valid-elections*]
by *blast*
hence *bij-betw* (φ*-rev valid-elections* π) *valid-elections valid-elections*
unfolding φ*-rev.simps*
using *bij-betw-ext*
by *blast*
moreover have φ*-rev valid-elections* $\pi \in$ *extensional valid-elections*
unfolding *extensional-def*
by *simp*
ultimately show φ*-rev valid-elections* $\pi \in$ *carrier* (*BijGroup valid-elections*)
unfolding *BijGroup-def Bij-def*
by *simp*
qed
fix
π :: $'a$ *rel* $\Rightarrow$ $'a$ *rel* **and**
π' :: $'a$ *rel* $\Rightarrow$ $'a$ *rel*
assume
rev: $\pi \in$ *carrier* $reversal_{\mathcal{G}}$ **and**
rev': $\pi' \in$ *carrier* $reversal_{\mathcal{G}}$
hence φ*-rev valid-elections* $(\pi \otimes_{reversal_{\mathcal{G}}} \pi') =$
extensional-continuation (*rel-app* $(\pi \circ \pi')$) *valid-elections*
unfolding $reversal_{\mathcal{G}}$*-def*
by *simp*
also have *rel-app* $(\pi \circ \pi') =$ *rel-app* $\pi \circ$ *rel-app* π'
using *rel-app.simps*
by *fastforce*
finally have *rewrite*:
φ*-rev valid-elections* $(\pi \otimes_{reversal_{\mathcal{G}}} \pi') =$
extensional-continuation (*rel-app* $\pi \circ$ *rel-app* π') *valid-elections*
by *blast*
have $\forall\ \mathcal{E} \in$ *valid-elections*. φ*-rev valid-elections* $\pi'\ \mathcal{E} \in$ *valid-elections*
using *car-el rev'*
unfolding *BijGroup-def Bij-def bij-betw-def*
by *auto*
hence *extensional-continuation*
(φ*-rev valid-elections* $\pi \circ \varphi$*-rev valid-elections* π') *valid-elections* $=$

extensional-continuation (*rel-app* $\pi \circ$ *rel-app* π') *valid-elections*
unfolding *extensional-continuation.simps* φ*-rev.simps*
by *fastforce*
also have
extensional-continuation (φ*-rev valid-elections* $\pi \circ \varphi$*-rev valid-elections* π')
valid-elections
= φ*-rev valid-elections* $\pi \otimes_{\textit{BijGroup valid-elections}}$ φ*-rev valid-elections* π'
using *car-el rewrite-mult rev rev'*
by *metis*
finally show
φ*-rev valid-elections* $(\pi \otimes_{\textit{reversal}_{\mathcal{G}}} \pi') =$
φ*-rev valid-elections* $\pi \otimes_{\textit{BijGroup valid-elections}}$ φ*-rev valid-elections* π'
using *rewrite*
by *metis*
qed

interpretation ψ*-rev-act*: *group-action* $\textit{reversal}_{\mathcal{G}}$ *UNIV* ψ*-rev*
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def* ψ*-rev.simps*,
safe, *rule group-BijGroup*)
fix π :: *'a rel* $\Rightarrow$ *'a rel*
show *bij*: $\bigwedge \pi.\ \pi \in$ *carrier* $\textit{reversal}_{\mathcal{G}} \Longrightarrow \pi \in$ *carrier* (*BijGroup UNIV*)
proof −
fix π :: *'b rel* $\Rightarrow$ *'b rel*
assume $\pi \in$ *carrier* $\textit{reversal}_{\mathcal{G}}$
hence $\pi \in \{id,\ \textit{rev-rel}\}$
unfolding $\textit{reversal}_{\mathcal{G}}$*-def*
by *auto*
hence *bij* π
using *rev-rev-id bij-id insertE o-bij singleton-iff*
by *metis*
thus $\pi \in$ *carrier* (*BijGroup UNIV*)
using *rewrite-carrier*
by *blast*
qed
fix
π :: *'a rel* $\Rightarrow$ *'a rel* **and**
π' :: *'a rel* $\Rightarrow$ *'a rel*
assume
rev: $\pi \in$ *carrier* $\textit{reversal}_{\mathcal{G}}$ **and**
rev': $\pi' \in$ *carrier* $\textit{reversal}_{\mathcal{G}}$
hence $\pi \otimes_{\textit{BijGroup UNIV}} \pi' = \pi \circ \pi'$
using *bij rewrite-mult-univ*
by *blast*
also from *rev rev'* **have** ... $= \pi \otimes_{\textit{reversal}_{\mathcal{G}}} \pi'$
unfolding $\textit{reversal}_{\mathcal{G}}$*-def*
by *simp*
finally show $\pi \otimes_{\textit{reversal}_{\mathcal{G}}} \pi' = \pi \otimes_{\textit{BijGroup UNIV}} \pi'$
by *simp*
qed

lemma φ-ψ-*rev-well-formed*:
shows *satisfies* (λ $\mathcal{E}$. *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* $\mathcal{E}$) *UNIV*)
(*equivar-ind-by-act* (*carrier reversal$_\mathcal{G}$*) *valid-elections*
(φ-*rev valid-elections*) (*set-action* ψ-*rev*))
proof (*unfold rewrite-equivar-ind-by-act*, *clarify*)
fix
π :: $'a$ *rel* $\Rightarrow$ $'a$ *rel* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile*
assume
π $\in$ *carrier reversal$_\mathcal{G}$* **and**
(A, V, p) $\in$ *valid-elections*
moreover from *this* **have** *cases*: π $\in$ {*id*, *rev-rel*}
unfolding *reversal$_\mathcal{G}$-def*
by *auto*
ultimately have *eq-A*: *alternatives-$\mathcal{E}$* (φ-*rev valid-elections* π (A, V, p)) = A
by *simp*
have
$\forall$ r $\in$ {*limit* A r | r. r $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A r)}. $\exists$ r' $\in$ *UNIV*.
rev-rel r = *limit* A (*rev-rel* r') $\wedge$
rev-rel r' $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A (*rev-rel* r'))
using *rev-rel-limit*[*of* A] *rev-rel-lin-ord*[*of* A]
by *force*
hence
$\forall$ r $\in$ {*limit* A r | r. r $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A r)}.
rev-rel r $\in$
{*limit* A (*rev-rel* r') | r'. *rev-rel* r' $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A (*rev-rel* r'))}
by *blast*
moreover have
{*limit* A (*rev-rel* r') | r'. *rev-rel* r' $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A (*rev-rel* r'))} $\subseteq$
{*limit* A r | r. r $\in$ *UNIV* $\wedge$ *linear-order-on* A (*limit* A r)}
by *blast*
ultimately have $\forall$ r $\in$ *limit-set-$\mathcal{SWF}$* A *UNIV*. *rev-rel* r $\in$ *limit-set-$\mathcal{SWF}$* A *UNIV*
unfolding *limit-set-$\mathcal{SWF}$.simps*
by *blast*
hence *subset*: $\forall$ r $\in$ *limit-set-$\mathcal{SWF}$* A *UNIV*. π r $\in$ *limit-set-$\mathcal{SWF}$* A *UNIV*
using *cases*
by *fastforce*
hence $\forall$ r $\in$ *limit-set-$\mathcal{SWF}$* A *UNIV*. r $\in$ π ‘ *limit-set-$\mathcal{SWF}$* A *UNIV*
using *rev-rev-id comp-apply empty-iff id-apply image-eqI insert-iff cases*
by *metis*
hence π ‘ *limit-set-$\mathcal{SWF}$* A *UNIV* = *limit-set-$\mathcal{SWF}$* A *UNIV*
using *subset*
by *blast*

hence *set-action* ψ*-rev* π (*limit-set-*$\mathcal{SWF}$ *A UNIV*) = *limit-set-*$\mathcal{SWF}$ *A UNIV*
unfolding *set-action.simps*
by *simp*
also have
... = *limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (φ*-rev valid-elections* π (*A*, *V*, *p*))) *UNIV*
using *eq-A*
by *simp*
finally show
limit-set-$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (φ*-rev valid-elections* π (*A*, *V*, *p*))) *UNIV* =
set-action ψ*-rev* π (*limit-set-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (*A*, *V*, *p*)) *UNIV*)
by *simp*
qed

end

1.10 Result-Dependent Voting Rule Properties

theory *Property-Interpretations*
imports *Voting-Symmetry*
Result-Interpretations
begin

1.10.1 Properties Dependent on the Result Type

The interpretation of equivariance properties generally depends on the result type. For example, neutrality for social choice rules means that single winners are renamed when the candidates in the votes are consistently renamed. For social welfare results, the complete result rankings must be renamed.

New result-type-dependent definitions for properties can be added here.

locale *result-properties* = *result* +
fixes ψ*-neutr* :: ('*a* ⇒ '*a*, '*b*) *binary-fun* **and**
$\mathcal{E}$ :: ('*a*, '*v*) *Election*
assumes
act-neutr: *group-action neutrality*$_{\mathcal{G}}$ *UNIV* ψ*-neutr* **and**
well-formed-res-neutr:
satisfies (λ $\mathcal{E}$ :: ('*a*, '*v*) *Election. limit-set* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*equivar-ind-by-act* (*carrier neutrality*$_{\mathcal{G}}$)
valid-elections (φ*-neutr valid-elections*) (*set-action* ψ*-neutr*))

sublocale *result-properties* ⊆ *result*
using *result-axioms*
by *simp*

1.10.2 Interpretations

global-interpretation $\mathcal{SCF}$*-properties*:

result-properties well-formed-$\mathcal{SCF}$ *limit-set-*$\mathcal{SCF}$ ψ*-neutr*$_{\mathrm{c}}$
unfolding *result-properties-def result-properties-axioms-def*
using *wf-result-neutrality-*$\mathcal{SCF}$ ψ*-neutr*$_{\mathrm{c}}$*-act.group-action-axioms*
$\mathcal{SCF}$*-result.result-axioms*
by *blast*

global-interpretation $\mathcal{SWF}$*-properties*:
result-properties well-formed-$\mathcal{SWF}$ *limit-set-*$\mathcal{SWF}$ ψ*-neutr*$_{\mathrm{w}}$
unfolding *result-properties-def result-properties-axioms-def*
using *wf-result-neutrality-*$\mathcal{SWF}$ ψ*-neutr*$_{\mathrm{w}}$*-act.group-action-axioms*
$\mathcal{SWF}$*-result.result-axioms*
by *blast*

end

Chapter 2

Refined Types

2.1 Preference List

theory *Preference-List*
  imports *../Preference-Relation*
    HOL−Combinatorics.Multiset-Permutations
    List−Index.List-Index
begin

Preference lists derive from preference relations, ordered from most to least preferred alternative.

2.1.1 Well-Formedness

type-synonym $'a$ *Preference-List* $=$ $'a$ *list*

abbreviation *well-formed-l* :: $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
  well-formed-l $l \equiv$ *distinct* l

2.1.2 Auxiliary Lemmas About Lists

lemma *is-arg-min-equal*:
  fixes
    f :: $'a \Rightarrow 'b{::}ord$ **and**
    g :: $'a \Rightarrow 'b$ **and**
    S :: $'a$ *set* **and**
    x :: $'a$
  assumes $\forall\ x \in S.\ f\, x = g\ x$
  shows *is-arg-min* f $(\lambda\ s.\ s \in S)$ x $=$ *is-arg-min* g $(\lambda\ s.\ s \in S)$ x
proof (*unfold is-arg-min-def*, *cases* $x \notin S$, *clarsimp*)
  case *x-in-S*: *False*
  thus $(x \in S \wedge (\nexists\ y.\ y \in S \wedge f\, y < f\, x)) = (x \in S \wedge (\nexists\ y.\ y \in S \wedge g\ y < g\ x))$
  proof (*cases* $\exists\ y.\ (\lambda\ s.\ s \in S)\ y \wedge f\, y < f\, x$)
    case y: *True*
    then obtain y :: $'a$ **where**

$(\lambda\ s.\ s \in S)\ y \wedge f\ y < f\ x$
by *metis*
hence $(\lambda\ s.\ s \in S)\ y \wedge g\ y < g\ x$
using *x-in-S assms*
by *metis*
thus *?thesis*
using *y*
by *metis*
next
case *not-y*: *False*
have $\neg\ (\exists\ y.\ (\lambda\ s.\ s \in S)\ y \wedge g\ y < g\ x)$
proof (*safe*)
fix $y :: 'a$
assume
y-in-S: $y \in S$ **and**
g-y-lt-g-x: $g\ y < g\ x$
have *f-eq-g-for-elems-in-S*: $\forall\ a.\ a \in S \longrightarrow f\ a = g\ a$
using *assms*
by *simp*
hence $g\ x = f\ x$
using *x-in-S*
by *presburger*
thus *False*
using *f-eq-g-for-elems-in-S g-y-lt-g-x not-y y-in-S*
by (*metis* (*no-types*))
qed
thus *?thesis*
using *x-in-S not-y*
by *simp*
qed
qed

lemma *list-cons-presv-finiteness*:
fixes
$A :: 'a\ set$ **and**
$S :: 'a\ list\ set$
assumes
fin-A: *finite* A **and**
fin-B: *finite* S
shows *finite* $\{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
proof −
let $?P = \lambda\ A.\ finite\ \{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
have $\forall\ a\ A'.\ finite\ A' \longrightarrow a \notin A' \longrightarrow\ ?P\ A' \longrightarrow\ ?P\ (insert\ a\ A')$
proof (*safe*)
fix
$a :: 'a$ **and**
$A' :: 'a\ set$
assume *finite* $\{a\#l \mid a\ l.\ a \in A' \wedge l \in S\}$
moreover have

$\{a'\#l \mid a'\ l.\ a' \in \mathit{insert}\ a\ A' \wedge l \in S\} =$
$\{a\#l \mid a\ l.\ a \in A' \wedge l \in S\} \cup \{a\#l \mid l.\ l \in S\}$
by *blast*
moreover have $\mathit{finite}\ \{a\#l \mid l.\ l \in S\}$
using *fin-B*
by *simp*
ultimately have $\mathit{finite}\ \{a'\#l \mid a'\ l.\ a' \in \mathit{insert}\ a\ A' \wedge l \in S\}$
by *simp*
thus $?P\ (\mathit{insert}\ a\ A')$
by *simp*
qed
moreover have $?P\ \{\}$
by *simp*
ultimately show $?P\ A$
using *finite-induct*$[\mathit{of}\ A\ ?P]$ *fin-A*
by *simp*
qed

lemma *listset-finiteness*:
fixes $l :: 'a\ \mathit{set}\ \mathit{list}$
assumes $\forall\ i{::}\mathit{nat}.\ i < \mathit{length}\ l \longrightarrow \mathit{finite}\ (l!i)$
shows $\mathit{finite}\ (\mathit{listset}\ l)$
using *assms*
proof (*induct l*, *simp*)
case (*Cons a l*)
fix
$a :: 'a\ \mathit{set}$ **and**
$l :: 'a\ \mathit{set}\ \mathit{list}$
assume
elems-fin-then-set-fin: $\forall\ i{::}\mathit{nat} < \mathit{length}\ l.\ \mathit{finite}\ (l!i) \Longrightarrow \mathit{finite}\ (\mathit{listset}\ l)$ **and**
fin-all-elems: $\forall\ i{::}\mathit{nat} < \mathit{length}\ (a\#l).\ \mathit{finite}\ ((a\#l)!i)$
hence *finite a*
by *auto*
moreover from *fin-all-elems*
have $\forall\ i < \mathit{length}\ l.\ \mathit{finite}\ (l!i)$
by *auto*
hence $\mathit{finite}\ (\mathit{listset}\ l)$
using *elems-fin-then-set-fin*
by *simp*
ultimately have $\mathit{finite}\ \{a'\#l' \mid a'\ l'.\ a' \in a \wedge l' \in (\mathit{listset}\ l)\}$
using *list-cons-presv-finiteness*
by *auto*
thus $\mathit{finite}\ (\mathit{listset}\ (a\#l))$
by (*simp add*: *set-Cons-def*)
qed

lemma *all-ls-elems-same-len*:
fixes $l :: 'a\ \mathit{set}\ \mathit{list}$
shows $\forall\ l'{::}('a\ \mathit{list}).\ l' \in \mathit{listset}\ l \longrightarrow \mathit{length}\ l' = \mathit{length}\ l$

```
proof (induct l, simp)
  case (Cons a l)
  fix
    a :: 'a set and
    l :: 'a set list
  assume ∀ l'. l' ∈ listset l ⟶ length l' = length l
  moreover have
    ∀ a' l'::('a set list). listset (a'#l') = {b#m | b m. b ∈ a' ∧ m ∈ listset l'}
    by (simp add: set-Cons-def)
  ultimately show ∀ l'. l' ∈ listset (a#l) ⟶ length l' = length (a#l)
    using local.Cons
    by force
qed

lemma all-ls-elems-in-ls-set:
  fixes l :: 'a set list
  shows ∀ l' i::nat. l' ∈ listset l ∧ i < length l' ⟶ l'!i ∈ l!i
proof (induct l, simp, safe)
  case (Cons a l)
  fix
    a :: 'a set and
    l :: 'a set list and
    l' :: 'a list and
    i :: nat
  assume elems-in-set-then-elems-pos:
    ∀ l' i::nat. l' ∈ listset l ∧ i < length l' ⟶ l'!i ∈ l!i and
    l-prime-in-set-a-l: l' ∈ listset (a#l) and
    i-lt-len-l-prime: i < length l'
  have l' ∈ set-Cons a (listset l)
    using l-prime-in-set-a-l
    by simp
  hence l' ∈ {m. ∃ b m'. m = b#m' ∧ b ∈ a ∧ m' ∈ (listset l)}
    unfolding set-Cons-def
    by simp
  hence ∃ b m. l' = b#m ∧ b ∈ a ∧ m ∈ (listset l)
    by simp
  thus l'!i ∈ (a#l)!i
    using elems-in-set-then-elems-pos i-lt-len-l-prime nth-Cons-Suc
          Suc-less-eq gr0-conv-Suc length-Cons nth-non-equal-first-eq
    by metis
qed

lemma all-ls-in-ls-set:
  fixes l :: 'a set list
  shows ∀ l'. length l' = length l ∧ (∀ i < length l'. l'!i ∈ l!i) ⟶ l' ∈ listset l
proof (induction l, safe, simp)
  case (Cons a l)
  fix
    l :: 'a set list and
```

```
    l' :: 'a list and
    s :: 'a set
  assume
    all-ls-in-ls-set-induct:
    ∀ m. length m = length l ∧ (∀ i < length m. m!i ∈ l!i) ⟶ m ∈ listset l and
    len-eq: length l' = length (s#l) and
    elems-pos-in-cons-ls-pos: ∀ i < length l'. l'!i ∈ (s#l)!i
  then obtain t and x where
    l'-cons: l' = x#t
    using length-Suc-conv
    by metis
  hence x ∈ s
    using elems-pos-in-cons-ls-pos
    by force
  moreover have t ∈ listset l
    using l'-cons all-ls-in-ls-set-induct len-eq diff-Suc-1 diff-Suc-eq-diff-pred
          elems-pos-in-cons-ls-pos length-Cons nth-Cons-Suc zero-less-diff
    by metis
  ultimately show l' ∈ listset (s#l)
    using l'-cons
    unfolding listset-def set-Cons-def
    by simp
qed
```

2.1.3 Ranking

Rank 1 is the top preference, rank 2 the second, and so on. Rank 0 does not exist.

```
fun rank-l :: 'a Preference-List ⇒ 'a ⇒ nat where
  rank-l l a = (if a ∈ set l then index l a + 1 else 0)

fun rank-l-idx :: 'a Preference-List ⇒ 'a ⇒ nat where
  rank-l-idx l a =
    (let i = index l a in
      if i = length l then 0 else i + 1)

lemma rank-l-equiv: rank-l = rank-l-idx
  unfolding member-def
  by (simp add: ext index-size-conv)

lemma rank-zero-imp-not-present:
  fixes
    p :: 'a Preference-List and
    a :: 'a
  assumes rank-l p a = 0
  shows a ∉ set p
  using assms
  by force
```

definition *above-l* :: *'a Preference-List* ⇒ *'a* ⇒ *'a Preference-List* **where**
above-l r a ≡ *take* (*rank-l r a*) *r*

2.1.4 Definition

fun *is-less-preferred-than-l* :: *'a* ⇒ *'a Preference-List* ⇒ *'a* ⇒ *bool*
(- $\lesssim$- - [*50*, *1000*, *51*] *50*) **where**
$a \lesssim_l b = (a \in set\ l \wedge b \in set\ l \wedge index\ l\ a \geq index\ l\ b)$

lemma *rank-gt-zero*:
fixes
l :: *'a Preference-List* **and**
a :: *'a*
assumes $a \lesssim_l a$
shows *rank-l l a* ≥ *1*
using *assms*
by *simp*

definition *pl-α* :: *'a Preference-List* ⇒ *'a Preference-Relation* **where**
pl-α l ≡ $\{(a, b).\ a \lesssim_l b\}$

lemma *rel-trans*:
fixes *l* :: *'a Preference-List*
shows *Relation.trans* (*pl-α l*)
unfolding *Relation.trans-def pl-α-def*
by *simp*

lemma *pl-α-lin-order*:
fixes
A :: *'a set* **and**
r :: *'a rel*
assumes *el*: *r* ∈ *pl-α* ‘ *permutations-of-set A*
shows *linear-order-on A r*
proof (*cases A* = {})
case *True*
thus *?thesis*
using *assms*
unfolding *pl-α-def is-less-preferred-than-l.simps*
by *simp*
next
case *False*
thus *?thesis*
proof (*unfold linear-order-on-def total-on-def antisym-def*
partial-order-on-def preorder-on-def, *safe*)
have *A* ≠ {}
using *False*
by *simp*
hence ∀ *l* ∈ *permutations-of-set A*. *l* ≠ []
using *assms permutations-of-setD*(*1*)

by *force*
hence $\forall\ a \in A.\ \forall\ l \in$ *permutations-of-set* $A.\ a \lesssim_l a$
using *is-less-preferred-than-l.simps*
unfolding *permutations-of-set-def*
by *simp*
hence $\forall\ a \in A.\ \forall\ l \in$ *permutations-of-set* $A.\ (a,\ a) \in$ *pl-*α l
unfolding *pl-*α*-def*
by *simp*
hence $\forall\ a \in A.\ (a,\ a) \in r$
using *el*
by *auto*
moreover have $r \subseteq A \times A$
using *el*
unfolding *pl-*α*-def permutations-of-set-def*
by *auto*
ultimately show *refl-on* $A\ r$
unfolding *refl-on-def*
by *simp*
next
show *Relation.trans* r
using *el rel-trans*
by *auto*
next
fix
$x :: 'a$ **and**
$y :: 'a$
assume
x-rel-y: $(x,\ y) \in r$ **and**
y-rel-x: $(y,\ x) \in r$
have $\forall\ x\ y.\ \forall\ l \in$ *permutations-of-set* $A.\ x \lesssim_l y \wedge y \lesssim_l x \longrightarrow x = y$
using *is-less-preferred-than-l.simps index-eq-index-conv nle-le*
unfolding *permutations-of-set-def*
by *metis*
hence $\forall\ x\ y.\ \forall\ l \in$ *pl-*α ‘ *permutations-of-set* $A.\ (x,\ y) \in l \wedge (y,\ x) \in l \longrightarrow x = y$
unfolding *pl-*α*-def permutations-of-set-def antisym-on-def*
by *blast*
thus $x = y$
using *y-rel-x x-rel-y el*
by *auto*
next
fix
$x :: 'a$ **and**
$y :: 'a$
assume
x-in-A: $x \in A$ **and**
y-in-A: $y \in A$ **and**
x-neq-y: $x \neq y$ **and**
not-y-x-rel: $(y,\ x) \notin r$

have $\forall\ x\ y.\ \forall\ l \in$ *permutations-of-set* $A.\ x \in A \wedge y \in A \wedge x \neq y \wedge (\neg\ y \lesssim_l x) \longrightarrow x \lesssim_l y$
using *is-less-preferred-than-l.simps*
unfolding *permutations-of-set-def*
by *auto*
hence $\forall\ x\ y.\ \forall\ l \in$ *pl-α* ‘ *permutations-of-set* A.
$x \in A \wedge y \in A \wedge x \neq y \wedge (y, x) \notin l \longrightarrow (x, y) \in l$
unfolding *pl-α-def permutations-of-set-def*
by *blast*
thus $(x, y) \in r$
using *x-in-A y-in-A x-neq-y not-y-x-rel el*
by *auto*
qed
qed

lemma *lin-order-pl-α*:
fixes
r :: *'a rel* **and**
A :: *'a set*
assumes
lin-order: *linear-order-on* $A\ r$ **and**
fin: *finite* A
shows $r \in$ *pl-α* ‘ *permutations-of-set* A
proof −
let $?\varphi = \lambda\ a.$ *card* ((*underS* $r\ a) \cap A$)
let *?inv* = *the-inv-into* A $?\varphi$
let $?l$ = *map* ($\lambda\ x.$ *?inv* x) (*rev* [0 ..< *card* A])
have *antisym*: $\forall\ a\ b.\ a \in$ ((*underS* $r\ b) \cap A) \wedge b \in$ ((*underS* $r\ a) \cap A) \longrightarrow$ *False*
using *lin-order*
unfolding *underS-def linear-order-on-def partial-order-on-def antisym-def*
by *auto*
hence $\forall\ a\ b\ c.\ a \in$ (*underS* $r\ b) \cap A \longrightarrow b \in$ (*underS* $r\ c) \cap A \longrightarrow a \in$ (*underS* $r\ c) \cap A$
using *lin-order CollectD CollectI transD IntE IntI*
unfolding *underS-def linear-order-on-def partial-order-on-def preorder-on-def*
by (*metis* (*mono-tags*, *lifting*))
hence $\forall\ a\ b.\ a \in$ (*underS* $r\ b) \cap A \longrightarrow$ (*underS* $r\ a) \cap A \subset$ (*underS* $r\ b) \cap A$
using *antisym*
by *blast*
hence *mon*: $\forall\ a\ b.\ a \in$ (*underS* $r\ b) \cap A \longrightarrow\ ?\varphi\ a <\ ?\varphi\ b$
using *fin*
by (*simp add*: *psubset-card-mono*)
moreover have *total-underS*:
$\forall\ a\ b.\ a \in A \wedge b \in A \wedge a \neq b \longrightarrow a \in$ ((*underS* $r\ b) \cap A) \vee b \in$ ((*underS* $r\ a) \cap A$)
using *lin-order totalp-onD totalp-on-total-on-eq*
unfolding *underS-def linear-order-on-def partial-order-on-def antisym-def*
by *fastforce*
ultimately have $\forall\ a\ b.\ a \in A \wedge b \in A \wedge a \neq b \longrightarrow\ ?\varphi\ a \neq\ ?\varphi\ b$

using *order-less-imp-not-eq2*
by *metis*
hence *inj*: *inj-on ?φ A*
using *inj-on-def*
by *blast*
have *in-bounds*: $\forall\ a \in A.\ ?\varphi\ a < card\ A$
using *CollectD IntD1 card-seteq fin inf-sup-ord*(*2*) *linorder-le-less-linear*
unfolding *underS-def*
by (*metis* (*mono-tags*, *lifting*))
hence $?\varphi$ ‘ $A \subseteq \{0\ ..< card\ A\}$
using *atLeast0LessThan*
by *blast*
moreover have *card* ($?\varphi$ ‘ A) = *card A*
using *inj fin card-image*
by *blast*
ultimately have $?\varphi$ ‘ $A = \{0\ ..< card\ A\}$
by (*simp add*: *card-subset-eq*)
hence *bij*: *bij-betw* $?\varphi\ A\ \{0\ ..< card\ A\}$
using *inj*
unfolding *bij-betw-def*
by *safe*
hence *bij-inv*: *bij-betw ?inv* $\{0\ ..< card\ A\}\ A$
using *bij-betw-the-inv-into*
by *metis*
hence *?inv* ‘ $\{0\ ..< card\ A\} = A$
unfolding *bij-betw-def*
by *metis*
hence *set ?l* = *A*
by *simp*
moreover have *dist-l*: *distinct ?l*
using *bij-inv*
unfolding *distinct-map*
using *bij-betw-imp-inj-on*
by *simp*
ultimately have $?l \in$ *permutations-of-set A*
by *auto*
moreover have *index-eq*: $\forall\ a \in A.\ index\ ?l\ a = card\ A - 1 - ?\varphi\ a$
proof
fix $a :: 'a$
assume *a-in-A*: $a \in A$
have $\forall\ xs.\ \forall\ i < length\ xs.\ (rev\ xs)!i = xs!(length\ xs - 1 - i)$
using *rev-nth*
by *auto*
hence $\forall\ i < length\ [0\ ..< card\ A].\ (rev\ [0\ ..< card\ A])!i$
$= [0\ ..< card\ A]!(length\ [0\ ..< card\ A] - 1 - i)$
by *blast*
moreover have $\forall\ i < card\ A.\ [0\ ..< card\ A]!i = i$
by *simp*
moreover have *card-A-len*: $length\ [0\ ..< card\ A] = card\ A$

by *simp*
ultimately have $\forall\ i < card\ A.\ (rev\ [0\ ..< card\ A])!i = card\ A - 1 - i$
using *diff-Suc-eq-diff-pred diff-less diff-self-eq-0 less-imp-diff-less zero-less-Suc*
by *metis*
moreover have $\forall\ i < card\ A.\ ?l!i = ?inv\ ((rev\ [0\ ..< card\ A])!i)$
by *simp*
ultimately have $\forall\ i < card\ A.\ ?l!i = ?inv\ (card\ A - 1 - i)$
by *presburger*
moreover have $card\ A - 1 - (card\ A - 1 - card\ (underS\ r\ a \cap A)) = card\ (underS\ r\ a \cap A)$
using *in-bounds a-in-A*
by *auto*
moreover have $?inv\ (card\ (underS\ r\ a \cap A)) = a$
using *a-in-A inj the-inv-into-f-f*
by *fastforce*
ultimately have $?l!(card\ A - 1 - card\ (underS\ r\ a \cap A)) = a$
using *in-bounds a-in-A card-Diff-singleton card-Suc-Diff1 diff-less-Suc fin*
by *metis*
thus $index\ ?l\ a = card\ A - 1 - card\ (underS\ r\ a \cap A)$
using *bij-inv dist-l a-in-A card-A-len card-Diff-singleton card-Suc-Diff1 diff-less-Suc fin index-nth-id length-map length-rev*
by *metis*
qed
moreover have $pl\text{-}\alpha\ ?l = r$
proof
show $r \subseteq pl\text{-}\alpha\ ?l$
proof (*unfold pl-α-def*, *auto*)
fix
$a :: 'a$ **and**
$b :: 'a$
assume $(a, b) \in r$
hence $a \in A$
using *lin-order*
unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
by *auto*
thus $a \in ?inv\ `\ \{0\ ..< card\ A\}$
using *bij-inv bij-betw-def*
by *metis*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume $(a, b) \in r$
hence $b \in A$
using *lin-order*
unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
by *auto*
thus $b \in ?inv\ `\ \{0\ ..< card\ A\}$
using *bij-inv bij-betw-def*

```
        by metis
    next
      fix
        a :: 'a and
        b :: 'a
      assume rel: (a, b) ∈ r
      hence el-A: a ∈ A ∧ b ∈ A
        using lin-order
       unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
        by auto
      moreover have a ∈ underS r b ∨ a = b
        using lin-order rel
        unfolding underS-def
        by simp
      ultimately have ?φ a ≤ ?φ b
        using mon le-eq-less-or-eq
        by auto
      thus index ?l b ≤ index ?l a
        using index-eq el-A diff-le-mono2
        by metis
    qed
  next
    show pl-α ?l ⊆ r
    proof (unfold pl-α-def, auto)
      fix
        a :: nat and
        b :: nat
      assume
        in-bnds-a: a < card A and
        in-bnds-b: b < card A and
        index-rel: index ?l (?inv b) ≤ index ?l (?inv a)
      have el-a: ?inv a ∈ A
        using bij-inv in-bnds-a atLeast0LessThan
        unfolding bij-betw-def
        by auto
      moreover have el-b: ?inv b ∈ A
        using bij-inv in-bnds-b atLeast0LessThan
        unfolding bij-betw-def
        by auto
      ultimately have leq-diff: card A − 1 − (?φ (?inv b)) ≤ card A − 1 − (?φ
(?inv a))
        using index-rel index-eq
        by metis
      have ∀ a < card A. ?φ (?inv a) < card A
        using fin bij-inv bij
        unfolding bij-betw-def
        by fastforce
      hence ?φ (?inv b) ≤ card A − 1 ∧ ?φ (?inv a) ≤ card A − 1
        using in-bnds-a in-bnds-b fin
```

```
        by fastforce
      hence ?φ (?inv b) ≥ ?φ (?inv a)
        using fin leq-diff le-diff-iff′
        by blast
      hence cases: ?φ (?inv a) < ?φ (?inv b) ∨ ?φ (?inv a) = ?φ (?inv b)
        by auto
      have ∀ a b. a ∈ A ∧ b ∈ A ∧ ?φ a < ?φ b ⟶ a ∈ underS r b
        using mon total-underS antisym IntD1 order-less-not-sym
        by metis
      hence ?φ (?inv a) < ?φ (?inv b) ⟶ ?inv a ∈ underS r (?inv b)
        using el-a el-b
        by blast
      hence cases-less: ?φ (?inv a) < ?φ (?inv b) ⟶ (?inv a, ?inv b) ∈ r
        unfolding underS-def
        by simp
      have ∀ a b. a ∈ A ∧ b ∈ A ∧ ?φ a = ?φ b ⟶ a = b
        using mon total-underS antisym order-less-not-sym
        by metis
      hence ?φ (?inv a) = ?φ (?inv b) ⟶ ?inv a = ?inv b
        using el-a el-b
        by simp
      hence cases-eq: ?φ (?inv a) = ?φ (?inv b) ⟶ (?inv a, ?inv b) ∈ r
        using lin-order el-a el-b
       unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
        by auto
      show (?inv a, ?inv b) ∈ r
        using cases cases-less cases-eq
        by auto
    qed
  qed
  ultimately show r ∈ pl-α ‘ permutations-of-set A
    by auto
qed

lemma index-helper:
  fixes
    xs :: ′x list and
    x :: ′x
  assumes
    fin-set-xs: finite (set xs) and
    dist-xs: distinct xs and
    x ∈ set xs
  shows index xs x = card {y ∈ set xs. index xs y < index xs x}
proof −
  have bij: bij-betw (index xs) (set xs) {0 ..< length xs}
    using assms bij-betw-index
    by blast
  hence card {y ∈ set xs. index xs y < index xs x}
        = card (index xs ‘ {y ∈ set xs. index xs y < index xs x})
```

```
    using CollectD bij-betw-same-card bij-betw-subset subsetI
    by (metis (no-types, lifting))
  also have index xs ‘ {y ∈ set xs. index xs y < index xs x}
        = {m | m. m ∈ index xs ‘ (set xs) ∧ m < index xs x}
    by blast
  also have {m | m. m ∈ index xs ‘ (set xs) ∧ m < index xs x} = {m | m. m <
index xs x}
    using bij assms atLeastLessThan-iff bot-nat-0.extremum
          index-image index-less-size-conv order-less-trans
    by metis
  also have card {m | m. m < index xs x} = index xs x
    by simp
  finally show ?thesis
    by simp
qed

lemma pl-α-eq-imp-list-eq:
  fixes
    xs :: 'x list and
    ys :: 'x list
  assumes
    fin-set-xs: finite (set xs) and
    set-eq: set xs = set ys and
    dist-xs: distinct xs and
    dist-ys: distinct ys and
    pl-α-eq: pl-α xs = pl-α ys
  shows xs = ys
proof (rule ccontr)
  assume xs ≠ ys
  moreover with this
  have xs ≠ [] ∧ ys ≠ []
    using set-eq
    by auto
  ultimately obtain
    i :: nat and
    x :: 'x where
      i < length xs and
      xs!i ≠ ys!i and
      x = xs!i and
    x-in-xs: x ∈ set xs
    using dist-xs dist-ys distinct-remdups-id
          length-remdups-card-conv nth-equalityI nth-mem set-eq
    by metis
  moreover with this
    have neq-ind: index xs x ≠ index ys x
    using dist-xs index-nth-id nth-index set-eq
    by metis
  ultimately have
     card {y ∈ set xs. index xs y < index xs x} ≠ card {y ∈ set xs. index ys y <
```

index ys x}
  using *dist-xs dist-ys set-eq index-helper fin-set-xs*
  by (*metis* (*mono-tags*))
 then obtain y :: *'x* **where**
  y-in-set-xs: $y \in$ *set xs* **and**
  y-neq-x: $y \neq x$ **and**
  neq-indices:
   (*index xs y* < *index xs x* ∧ *index ys y* > *index ys x*) ∨
    (*index ys y* < *index ys x* ∧ *index xs y* > *index xs x*)
  using *index-eq-index-conv not-less-iff-gr-or-eq set-eq*
  by (*metis* (*mono-tags*, *lifting*))
 hence (*is-less-preferred-than-l x xs y* ∧ *is-less-preferred-than-l y ys x*)
     ∨ (*is-less-preferred-than-l x ys y* ∧ *is-less-preferred-than-l y xs x*)
  unfolding *is-less-preferred-than-l.simps*
  using *y-in-set-xs less-imp-le-nat set-eq x-in-xs*
  by *blast*
 hence ((x, y) ∈ *pl-α xs* ∧ (x, y) ∉ *pl-α ys*) ∨ ((x, y) ∈ *pl-α ys* ∧ (x, y) ∉ *pl-α xs*)
  unfolding *pl-α-def*
  using *is-less-preferred-than-l.simps y-neq-x neq-indices*
    case-prod-conv linorder-not-less mem-Collect-eq
  by *metis*
 thus *False*
  using *pl-α-eq*
  by *blast*
qed

lemma *pl-α-bij-betw*:
 fixes X :: *'x set*
 assumes *finite X*
 shows *bij-betw pl-α* (*permutations-of-set X*) {*r*. *linear-order-on X r*}
proof (*unfold bij-betw-def*, *safe*)
 show *inj-on pl-α* (*permutations-of-set X*)
  unfolding *inj-on-def permutations-of-set-def*
  using *pl-α-eq-imp-list-eq assms*
  by *fastforce*
next
 fix *xs* :: *'x list*
 assume *xs* ∈ *permutations-of-set X*
 thus *linear-order-on X* (*pl-α xs*)
  using *assms pl-α-lin-order*
  by *blast*
next
 fix *r* :: *'x rel*
 assume *linear-order-on X r*
 thus *r* ∈ *pl-α* ‘ *permutations-of-set X*
  using *assms lin-order-pl-α*
  by *blast*
qed

2.1.5 Limited Preference

definition *limited* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
  limited A $r \equiv \forall\ a.\ a \in$ *set* $r \longrightarrow\ a \in A$

fun *limit-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ $'a$ *Preference-List* **where**
  limit-l A l = *List.filter* $(\lambda\ a.\ a \in A)$ l

lemma *limited-dest*:
  fixes
    A :: $'a$ *set* **and**
    l :: $'a$ *Preference-List* **and**
    a :: $'a$ **and**
    b :: $'a$
  assumes
    $a \lesssim_l b$ **and**
    limited A l
  shows $a \in A \wedge b \in A$
  using *assms*
  unfolding *limited-def*
  by *simp*

lemma *limit-equiv*:
  fixes
    A :: $'a$ *set* **and**
    l :: $'a$ *list*
  assumes *well-formed-l* l
  shows *pl-*α (*limit-l* A l) = *limit* A (*pl-*α l)
  using *assms*
proof (*induction* l)
  case *Nil*
  thus *pl-*α (*limit-l* A []) = *limit* A (*pl-*α [])
    unfolding *pl-*α*-def*
    by *simp*
next
  case (*Cons* a l)
  fix
    a :: $'a$ **and**
    l :: $'a$ *list*
  assume
    wf-imp-limit: *well-formed-l* $l \Longrightarrow$ *pl-*α (*limit-l* A l) = *limit* A (*pl-*α l) **and**
    wf-a-l: *well-formed-l* $(a\#l)$
  show *pl-*α (*limit-l* A $(a\#l)$) = *limit* A (*pl-*α $(a\#l)$)
    using *wf-imp-limit wf-a-l*
  proof (*clarsimp*, *safe*)
    fix
      b :: $'a$ **and**
      c :: $'a$
    assume *b-less-c*: $(b,\ c) \in$ *pl-*α $(a\#($*filter* $(\lambda\ a.\ a \in A)$ $l))$
    have *limit-preference-list-assoc*: *pl-*α (*limit-l* A l) = *limit* A (*pl-*α l)

using *wf-a-l wf-imp-limit*
by *simp*
thus $(b,\ c) \in pl\text{-}\alpha\ (a\#l)$
proof (*unfold pl-α-def is-less-preferred-than-l.simps*, *safe*)
show $b \in set\ (a\#l)$
using *b-less-c*
unfolding *pl-α-def*
by *fastforce*
next
show $c \in set\ (a\#l)$
using *b-less-c*
unfolding *pl-α-def*
by *fastforce*
next
have $\forall\ a'\ l'\ a''.\ (a'{::}'a) \lesssim_{l}{}'\ a'' =$
$(a' \in set\ l' \wedge a'' \in set\ l' \wedge index\ l'\ a'' \leq index\ l'\ a')$
using *is-less-preferred-than-l.simps*
by *blast*
moreover from *this*
have $\{(a',\ b').\ a' \lesssim_{(}limit\text{-}l\ A\ l)\ b'\} =$
$\{(a',\ a'').\ a' \in set\ (limit\text{-}l\ A\ l) \wedge a'' \in set\ (limit\text{-}l\ A\ l) \wedge$
$index\ (limit\text{-}l\ A\ l)\ a'' \leq index\ (limit\text{-}l\ A\ l)\ a'\}$
by *presburger*
moreover from *this*
have $\{(a',\ b').\ a' \lesssim_{l}\ b'\} =$
$\{(a',\ a'').\ a' \in set\ l \wedge a'' \in set\ l \wedge index\ l\ a'' \leq index\ l\ a'\}$
using *is-less-preferred-than-l.simps*
by *auto*
ultimately have $\{(a',\ b').$
$a' \in set\ (limit\text{-}l\ A\ l) \wedge b' \in set\ (limit\text{-}l\ A\ l) \wedge$
$index\ (limit\text{-}l\ A\ l)\ b' \leq index\ (limit\text{-}l\ A\ l)\ a'\} =$
$limit\ A\ \{(a',\ b').\ a' \in set\ l \wedge b' \in set\ l \wedge index\ l\ b' \leq index\ l\ a'\}$
using *pl-α-def limit-preference-list-assoc*
by (*metis* (*no-types*))
hence *idx-imp*:
$b \in set\ (limit\text{-}l\ A\ l) \wedge c \in set\ (limit\text{-}l\ A\ l) \wedge$
$index\ (limit\text{-}l\ A\ l)\ c \leq index\ (limit\text{-}l\ A\ l)\ b \longrightarrow$
$b \in set\ l \wedge c \in set\ l \wedge index\ l\ c \leq index\ l\ b$
by *auto*
have $b \lesssim_{(}a\#(filter\ (\lambda\ a.\ a \in A)\ l))\ c$
using *b-less-c case-prodD mem-Collect-eq*
unfolding *pl-α-def*
by *metis*
moreover obtain
$f :: {}'a \Rightarrow {}'a\ list \Rightarrow {}'a \Rightarrow {}'a$ **and**
$g :: {}'a \Rightarrow {}'a\ list \Rightarrow {}'a \Rightarrow {}'a\ list$ **and**
$h :: {}'a \Rightarrow {}'a\ list \Rightarrow {}'a \Rightarrow {}'a$ **where**
$\forall\ d\ s\ e.\ d \lesssim_{s}\ e \longrightarrow$
$d = f\ e\ s\ d \wedge s = g\ e\ s\ d \wedge e = h\ e\ s\ d \wedge f\ e\ s\ d \in set\ (g\ e\ s\ d) \wedge$

```
          index (g e s d) (h e s d) ≤ index (g e s d) (f e s d) ∧
            h e s d ∈ set (g e s d)
      by fastforce
    ultimately have
      b = f c (a#(filter (λ a. a ∈ A) l)) b ∧
        a#(filter (λ a. a ∈ A) l) = g c (a#(filter (λ a. a ∈ A) l)) b ∧
        c = h c (a#(filter (λ a. a ∈ A) l)) b ∧
       f c (a#(filter (λ a. a ∈ A) l)) b ∈ set (g c (a#(filter (λ a. a ∈ A) l)) b) ∧
       h c (a#(filter (λ a. a ∈ A) l)) b ∈ set (g c (a#(filter (λ a. a ∈ A) l)) b) ∧
        index (g c (a#(filter (λ a. a ∈ A) l)) b)
            (h c (a#(filter (λ a. a ∈ A) l)) b) ≤
          index (g c (a#(filter (λ a. a ∈ A) l)) b)
            (f c (a#(filter (λ a. a ∈ A) l)) b)
      by blast
    moreover have filter (λ a. a ∈ A) l = limit-l A l
      by simp
    ultimately have a ≠ c ⟶ index (a#l) c ≤ index (a#l) b
      using idx-imp
      by force
    thus index (a#l) c ≤ index (a#l) b
      by force
  qed
next
  fix
    b :: 'a and
    c :: 'a
  assume
     a ∈ A and
    (b, c) ∈ pl-α (a#(filter (λ a. a ∈ A) l))
  thus c ∈ A
    unfolding pl-α-def
    by fastforce
next
  fix
    b :: 'a and
    c :: 'a
  assume
    a ∈ A and
    (b, c) ∈ pl-α (a#(filter (λ a. a ∈ A) l))
  thus b ∈ A
    unfolding pl-α-def
    using case-prodD insert-iff mem-Collect-eq set-filter inter-set-filter IntE
    by auto
next
  fix
    b :: 'a and
    c :: 'a
  assume
    b-less-c: (b, c) ∈ pl-α (a#l) and
```

b-in-A: $b \in A$ **and**
c-in-A: $c \in A$
show $(b,\ c) \in$ *pl-α* $(a\#($*filter* $(\lambda\ a.\ a \in A)\ l))$
proof (*unfold pl-α-def is-less-preferred-than.simps*, *safe*)
show $b \lesssim_{(} a\#($*filter* $(\lambda\ a.\ a \in A)\ l))\ c$
proof (*unfold is-less-preferred-than-l.simps*, *safe*)
show $b \in$ *set* $(a\#($*filter* $(\lambda\ a.\ a \in A)\ l))$
using *b-less-c b-in-A*
unfolding *pl-α-def*
by *fastforce*
next
show $c \in$ *set* $(a\#($*filter* $(\lambda\ a.\ a \in A)\ l))$
using *b-less-c c-in-A*
unfolding *pl-α-def*
by *fastforce*
next
have $(b,\ c) \in$ *pl-α* $(a\#l)$
by (*simp add*: *b-less-c*)
hence $b \lesssim_{(} a\#l)\ c$
using *case-prodD mem-Collect-eq*
unfolding *pl-α-def*
by *metis*
moreover have
pl-α (*filter* $(\lambda\ a.\ a \in A)\ l) = \{(a,\ b).\ (a,\ b) \in$ *pl-α* $l \wedge a \in A \wedge b \in A\}$
using *wf-a-l wf-imp-limit*
by *simp*
ultimately show
index $(a\#($*filter* $(\lambda\ a.\ a \in A)\ l))\ c \leq$ *index* $(a\#($*filter* $(\lambda\ a.\ a \in A)\ l))\ b$
unfolding *pl-α-def*
using *add-leE add-le-cancel-right case-prodI c-in-A b-in-A index-Cons set-ConsD*
in-rel-Collect-case-prod-eq linorder-le-cases mem-Collect-eq not-one-le-zero
by *fastforce*
qed
qed
next
fix
b :: $'a$ **and**
c :: $'a$
assume
a-not-in-A: $a \notin A$ **and**
b-less-c: $(b,\ c) \in$ *pl-α* l
show $(b,\ c) \in$ *pl-α* $(a\#l)$
proof (*unfold pl-α-def is-less-preferred-than-l.simps*, *safe*)
show $b \in$ *set* $(a\#l)$
using *b-less-c*
unfolding *pl-α-def*
by *fastforce*
next

```
    show c ∈ set (a#l)
      using b-less-c
      unfolding pl-α-def
      by fastforce
  next
    show index (a#l) c ≤ index (a#l) b
    proof (unfold index-def, simp, safe)
      assume a = b
      thus False
        using a-not-in-A b-less-c case-prod-conv is-less-preferred-than-l.elims
              mem-Collect-eq set-filter wf-a-l
        unfolding pl-α-def
        by simp
    next
      show find-index (λ x. x = c) l ≤ find-index (λ x. x = b) l
        using b-less-c case-prodD mem-Collect-eq
        unfolding pl-α-def
        by (simp add: index-def)
    qed
  qed
next
  fix
    b :: 'a and
    c :: 'a
  assume
    a-not-in-l: a ∉ set l and
    a-not-in-A: a ∉ A and
    b-in-A: b ∈ A and
    c-in-A: c ∈ A and
    b-less-c: (b, c) ∈ pl-α (a#l)
  thus (b, c) ∈ pl-α l
  proof (unfold pl-α-def is-less-preferred-than-l.simps, safe)
    assume b ∈ set (a#l)
    thus b ∈ set l
      using a-not-in-A b-in-A
      by fastforce
  next
    assume c ∈ set (a#l)
    thus c ∈ set l
      using a-not-in-A c-in-A
      by fastforce
  next
    assume index (a#l) c ≤ index (a#l) b
    thus index l c ≤ index l b
      using a-not-in-l a-not-in-A c-in-A add-le-cancel-right
            index-Cons index-le-size size-index-conv
      by (metis (no-types, lifting))
  qed
qed
```

qed

2.1.6 Auxiliary Definitions

definition *total-on-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
total-on-l A l $\equiv \forall\ a \in A.\ a \in$ *set l*

definition *refl-on-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
refl-on-l A l $\equiv (\forall\ a.\ a \in$ *set l* $\longrightarrow a \in A) \land (\forall\ a \in A.\ a \lesssim_l a)$

definition *trans* :: $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
trans l $\equiv \forall\ (a, b, c) \in$ *set l* $\times$ *set l* $\times$ *set l*. $a \lesssim_l b \land b \lesssim_l c \longrightarrow a \lesssim_l c$

definition *preorder-on-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
preorder-on-l A l $\equiv$ *refl-on-l A l* $\land$ *trans l*

definition *antisym-l* :: $'a$ *list* $\Rightarrow$ *bool* **where**
antisym-l l $\equiv \forall\ a\ b.\ a \lesssim_l b \land b \lesssim_l a \longrightarrow a = b$

definition *partial-order-on-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
partial-order-on-l A l $\equiv$ *preorder-on-l A l* $\land$ *antisym-l l*

definition *linear-order-on-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
linear-order-on-l A l $\equiv$ *partial-order-on-l A l* $\land$ *total-on-l A l*

definition *connex-l* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
connex-l A l $\equiv$ *limited A l* $\land (\forall\ a \in A.\ \forall\ b \in A.\ a \lesssim_l b \lor b \lesssim_l a)$

abbreviation *ballot-on* :: $'a$ *set* $\Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
ballot-on A l $\equiv$ *well-formed-l l* $\land$ *linear-order-on-l A l*

2.1.7 Auxiliary Lemmas

lemma *list-trans*[*simp*]:
fixes *l* :: $'a$ *Preference-List*
shows *trans l*
unfolding *trans-def*
by *simp*

lemma *list-antisym*[*simp*]:
fixes *l* :: $'a$ *Preference-List*
shows *antisym-l l*
unfolding *antisym-l-def*
by *auto*

lemma *lin-order-equiv-list-of-alts*:
fixes
A :: $'a$ *set* **and**
l :: $'a$ *Preference-List*
shows *linear-order-on-l A l* $=$ ($A =$ *set l*)

unfolding *linear-order-on-l-def total-on-l-def partial-order-on-l-def preorder-on-l-def refl-on-l-def*
by *auto*

lemma *connex-imp-refl*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List*
assumes *connex-l A l*
shows *refl-on-l A l*
unfolding *refl-on-l-def*
using *assms connex-l-def Preference-List.limited-def*
by *metis*

lemma *lin-ord-imp-connex-l*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List*
assumes *linear-order-on-l A l*
shows *connex-l A l*
using *assms linorder-le-cases*
unfolding *connex-l-def linear-order-on-l-def preorder-on-l-def limited-def refl-on-l-def partial-order-on-l-def is-less-preferred-than-l.simps*
by *metis*

lemma *above-trans*:
fixes
l :: *'a Preference-List* **and**
a :: *'a* **and**
b :: *'a*
assumes
trans l **and**
$a \lesssim_l b$
shows *set* (*above-l l b*) $\subseteq$ *set* (*above-l l a*)
using *assms set-take-subset-set-take rank-l.simps Suc-le-mono add.commute add-0 add-Suc*
unfolding *above-l-def Preference-List.is-less-preferred-than-l.simps One-nat-def*
by *metis*

lemma *less-preferred-l-rel-equiv*:
fixes
l :: *'a Preference-List* **and**
a :: *'a* **and**
b :: *'a*
shows $a \lesssim_l b$ = *Preference-Relation.is-less-preferred-than a* (*pl-*α *l*) *b*
unfolding *pl-*α*-def*
by *simp*

theorem *above-equiv*:

fixes
l :: *'a Preference-List* **and**
a :: *'a*
shows *set* (*above-l l a*) = *above* (*pl-α l*) *a*
proof (*safe*)
fix *b* :: *'a*
assume *b-member*: *b* ∈ *set* (*above-l l a*)
hence *index l b* ≤ *index l a*
unfolding *rank-l.simps above-l-def*
using *Suc-eq-plus1 Suc-le-eq index-take linorder-not-less bot-nat-0.extremum-strict*
by (*metis* (*full-types*))
hence $a \lesssim_l b$
using *Suc-le-mono add-Suc le-antisym take-0 b-member in-set-takeD index-take le0 rank-l.simps*
unfolding *above-l-def is-less-preferred-than-l.simps*
by *metis*
thus *b* ∈ *above* (*pl-α l*) *a*
using *less-preferred-l-rel-equiv pref-imp-in-above*
by *metis*
next
fix *b* :: *'a*
assume *b* ∈ *above* (*pl-α l*) *a*
hence $a \lesssim_l b$
using *pref-imp-in-above less-preferred-l-rel-equiv*
by *metis*
thus *b* ∈ *set* (*above-l l a*)
unfolding *above-l-def is-less-preferred-than-l.simps rank-l.simps*
using *Suc-eq-plus1 Suc-le-eq index-less-size-conv set-take-if-index le-imp-less-Suc*
by (*metis* (*full-types*))
qed

theorem *rank-equiv*:
fixes
l :: *'a Preference-List* **and**
a :: *'a*
assumes *well-formed-l l*
shows *rank-l l a* = *rank* (*pl-α l*) *a*
proof (*simp*, *safe*)
assume *a* ∈ *set l*
moreover have *above* (*pl-α l*) *a* = *set* (*above-l l a*)
unfolding *above-equiv*
by *simp*
moreover have *distinct* (*above-l l a*)
unfolding *above-l-def*
using *assms distinct-take*
by *blast*
moreover from *this*
have *card* (*set* (*above-l l a*)) = *length* (*above-l l a*)

  using *distinct-card*
  by *blast*
 moreover have *length* (*above-l l a*) = *rank-l l a*
  unfolding *above-l-def*
  using *Suc-le-eq*
  by (*simp add*: *in-set-member*)
 ultimately show *Suc* (*index l a*) = *card* (*above* (*pl-*α *l*) *a*)
  by *simp*
next
 assume $a \notin$ *set l*
 hence *above* (*pl-*α *l*) *a* = {}
  unfolding *above-def*
  using *less-preferred-l-rel-equiv*
  by *fastforce*
 thus *card* (*above* (*pl-*α *l*) *a*) = *0*
  by *fastforce*
qed

lemma *lin-ord-equiv*:
 fixes
  A :: *'a set* **and**
  l :: *'a Preference-List*
 shows *linear-order-on-l A l* = *linear-order-on A* (*pl-*α *l*)
 unfolding *pl-*α*-def linear-order-on-l-def linear-order-on-def refl-on-l-def*
   Relation.trans-def preorder-on-l-def partial-order-on-l-def partial-order-on-def
    total-on-l-def preorder-on-def refl-on-def antisym-def total-on-def
    is-less-preferred-than-l.simps
 by *auto*

2.1.8 First Occurrence Indices

lemma *pos-in-list-yields-rank*:
 fixes
  l :: *'a Preference-List* **and**
  a :: *'a* **and**
  n :: *nat*
 assumes
  $\forall$ (*j*::*nat*) $\leq$ *n*. *l*!*j* $\neq$ *a* **and**
  l!(*n* − *1*) = *a*
 shows *rank-l l a* = *n*
 using *assms*
proof (*induction l arbitrary*: *n*, *simp-all*) **qed**

lemma *ranked-alt-not-at-pos-before*:
 fixes
  l :: *'a Preference-List* **and**
  a :: *'a* **and**
  n :: *nat*
 assumes

```
    a ∈ set l and
    n < (rank-l l a) − 1
  shows l!n ≠ a
  using assms add-diff-cancel-right′ index-first member-def rank-l.simps
  by metis

lemma pos-in-list-yields-pos:
  fixes
    l :: ′a Preference-List and
    a :: ′a
  assumes a ∈ set l
  shows l!(rank-l l a − 1) = a
  using assms
proof (induction l, simp)
  fix
    l :: ′a Preference-List and
    b :: ′a
  case (Cons b l)
  assume a ∈ set (b#l)
  moreover from this
  have rank-l (b#l) a = 1 + index (b#l) a
    using Suc-eq-plus1 add-Suc add-cancel-left-left rank-l.simps
    by metis
  ultimately show (b#l)!(rank-l (b#l) a − 1) = a
    using diff-add-inverse nth-index
    by metis
qed


lemma rel-of-pref-pred-for-set-eq-list-to-rel:
  fixes l :: ′a Preference-List
  shows relation-of (λ y z. y ≲l z) (set l) = pl-α l
proof (unfold relation-of-def, safe)
  fix
    a :: ′a and
    b :: ′a
  assume a ≲l b
  moreover have (a ≲l b) = (a ≼(pl-α l) b)
    using less-preferred-l-rel-equiv
    by (metis (no-types))
  ultimately show (a, b) ∈ pl-α l
    by simp
next
  fix
    a :: ′a and
    b :: ′a
  assume (a, b) ∈ pl-α l
  thus a ≲l b
    using less-preferred-l-rel-equiv
```

unfolding *is-less-preferred-than.simps*
by *metis*
thus
$a \in set\ l$ **and**
$b \in set\ l$
by (*simp*, *simp*)
qed

end

2.2 Preference (List) Profile

theory *Profile-List*
imports ../*Profile*
Preference-List
begin

2.2.1 Definition

A profile (list) contains one ballot for each voter.

type-synonym *'a Profile-List* = *'a Preference-List list*

type-synonym *'a Election-List* = *'a set* × *'a Profile-List*

Abstraction from profile list to profile.

fun *pl-to-pr-α* :: *'a Profile-List* ⇒ (*'a*, *nat*) *Profile* **where**
pl-to-pr-α pl = (λ *n*. *if* (*n* < *length pl* ∧ *n* ≥ *0*)
then (*map* (*Preference-List.pl-α*) *pl*)!*n*
else {})

lemma *prof-abstr-presv-size*:
fixes *p* :: *'a Profile-List*
shows *length p* = *length* (*to-list* {*0* ..< *length p*} (*pl-to-pr-α p*))
by *simp*

A profile on a finite set of alternatives A contains only ballots that are lists of linear orders on A.

definition *profile-l* :: *'a set* ⇒ *'a Profile-List* ⇒ *bool* **where**
profile-l A p ≡ ∀ *i* < *length p*. *ballot-on A* (*p*!*i*)

lemma *refinement*:
fixes
A :: *'a set* **and**
p :: *'a Profile-List*
assumes *profile-l A p*
shows *profile* {*0* ..< *length p*} *A* (*pl-to-pr-α p*)

```
proof (unfold profile-def, safe)
  fix i :: nat
  assume in-range: i ∈ {0 ..< length p}
  moreover have well-formed-l (p!i)
    using assms in-range
    unfolding profile-l-def
    by simp
  moreover have linear-order-on-l A (p!i)
    using assms in-range
    unfolding profile-l-def
    by simp
  ultimately show linear-order-on A (pl-to-pr-α p i)
    using lin-ord-equiv length-map nth-map
    by auto
qed

end
```

2.3 Ordered Relation Type

```
theory Ordered-Relation
  imports Preference-Relation
          ./Refined-Types/Preference-List
          HOL−Combinatorics.Multiset-Permutations
begin

lemma fin-ordered:
  fixes X :: 'x set
  assumes finite X
  obtains ord :: 'x rel where
    linear-order-on X ord
proof −
  assume
    ex: ⋀ ord. linear-order-on X ord ⟹ ?thesis
  obtain l :: 'x list where
    set-l: set l = X
    using finite-list assms
    by blast
  let ?r = pl-α l
  have antisym ?r
    using set-l Collect-mono-iff antisym index-eq-index-conv pl-α-def
    unfolding antisym-def
    by fastforce
  moreover have refl-on X ?r
    using set-l
    unfolding refl-on-def pl-α-def is-less-preferred-than-l.simps
    by blast
```

moreover have *Relation.trans ?r*
unfolding *Relation.trans-def pl-α-def is-less-preferred-than-l.simps*
by *auto*
moreover have *total-on X ?r*
using *set-l*
unfolding *total-on-def pl-α-def is-less-preferred-than-l.simps*
by *force*
ultimately have *linear-order-on X ?r*
unfolding *linear-order-on-def preorder-on-def partial-order-on-def*
by *blast*
thus *?thesis*
using *ex*
by *blast*
qed

typedef *'a Ordered-Preference* =
{*p* :: *'a*::*finite Preference-Relation. linear-order-on* (*UNIV*::*'a set*) *p*}
morphisms *ord2pref pref2ord*
proof (*simp*)
have *finite* (*UNIV*::*'a set*)
by *simp*
then obtain *p* :: *'a Preference-Relation* **where**
linear-order-on (*UNIV*::*'a set*) *p*
using *fin-ordered*
by *metis*
thus ∃ *p*::*'a Preference-Relation. linear-order p*
by *blast*
qed

instance *Ordered-Preference* :: (*finite*) *finite*
proof
have (*UNIV*::*'a Ordered-Preference set*) =
pref2ord ' {*p* :: *'a Preference-Relation. linear-order-on* (*UNIV*::*'a set*) *p*}
using *type-definition.Abs-image type-definition-Ordered-Preference*
by *blast*
moreover have *finite* {*p* :: *'a Preference-Relation. linear-order-on* (*UNIV*::*'a set*) *p*}
by *simp*
ultimately show *finite* (*UNIV*::*'a Ordered-Preference set*)
using *finite-imageI*
by *metis*
qed

lemma *range-ord2pref*: *range ord2pref* = {*p. linear-order p*}
using *type-definition.Rep-range type-definition-Ordered-Preference*
by *metis*

lemma *card-ord-pref*: *card* (*UNIV*::*'a*::*finite Ordered-Preference set*) = *fact* (*card* (*UNIV*::*'a set*))

proof −
  let *?n* = *card* (*UNIV*::′*a set*) **and**
    ?perm = *permutations-of-set* (*UNIV* :: ′*a set*)
  have (*UNIV*::(′*a Ordered-Preference set*)) =
    pref2ord ‘ {*p* :: ′*a Preference-Relation*. *linear-order-on* (*UNIV*::′*a set*) *p*}
    using *type-definition-Ordered-Preference type-definition.Abs-image*
    by *blast*
  moreover have
    inj-on pref2ord {*p* :: ′*a Preference-Relation*. *linear-order-on* (*UNIV*::′*a set*) *p*}
    using *inj-onCI pref2ord-inject*
    by *metis*
  ultimately have
    bij-betw pref2ord
      {*p* :: ′*a Preference-Relation*. *linear-order-on* (*UNIV*::′*a set*) *p*}
      (*UNIV*::(′*a Ordered-Preference set*))
    using *bij-betw-imageI*
    by *metis*
  hence *card* (*UNIV*::(′*a Ordered-Preference set*)) =
    card {*p* :: ′*a Preference-Relation*. *linear-order-on* (*UNIV*::′*a set*) *p*}
    using *bij-betw-same-card*
    by *metis*
  moreover have *card ?perm* = *fact ?n*
    by *simp*
  ultimately show *?thesis*
    using *bij-betw-same-card pl-α-bij-betw finite*
    by *metis*
qed

end

2.4 Alternative Election Type

theory *Quotient-Type-Election*
  imports *Profile*
begin

lemma *election-equality-equiv*:
  election-equality E E **and**
  election-equality E E′ ⟹ *election-equality E*′ *E* **and**
  election-equality E E′ ⟹ *election-equality E*′ *F* ⟹ *election-equality E F*
proof −
  have ∀ *E*. *E* = (*fst E*, *fst* (*snd E*), *snd* (*snd E*))
    by *simp*
  thus
    election-equality E E **and**
    election-equality E E′ ⟹ *election-equality E*′ *E* **and**
    election-equality E E′ ⟹ *election-equality E*′ *F* ⟹ *election-equality E F*

using *election-equality.simps*[*of fst E fst* (*snd E*) *snd* (*snd E*)]
election-equality.simps[*of*
fst E′ fst (*snd E′*) *snd* (*snd E′*) *fst E fst* (*snd E*) *snd* (*snd E*)]
election-equality.simps[*of*
fst E′ fst (*snd E′*) *snd* (*snd E′*) *fst F fst* (*snd F*) *snd* (*snd F*)]
by (*metis*, *metis*, *metis*)
qed

quotient-type (′*a*, ′*v*) *Election-Alt* =
′*a set* × ′*v set* × (′*a*, ′*v*) *Profile* / *election-equality*
unfolding *equivp-reflp-symp-transp reflp-def symp-def transp-def*
using *election-equality-equiv*
by *simp*

fun *fst-alt* :: (′*a*, ′*v*) *Election-Alt* ⇒ ′*a set* **where**
fst-alt E = *Product-Type.fst* (*rep-Election-Alt E*)

fun *snd-alt* :: (′*a*, ′*v*) *Election-Alt* ⇒ ′*v set* × (′*a*, ′*v*) *Profile* **where**
snd-alt E = *Product-Type.snd* (*rep-Election-Alt E*)

abbreviation *alternatives-$\mathcal{E}$-alt* :: (′*a*, ′*v*) *Election-Alt* ⇒ ′*a set* **where**
alternatives-$\mathcal{E}$-alt E ≡ *fst-alt E*

abbreviation *voters-$\mathcal{E}$-alt* :: (′*a*, ′*v*) *Election-Alt* ⇒ ′*v set* **where**
voters-$\mathcal{E}$-alt E ≡ *Product-Type.fst* (*snd-alt E*)

abbreviation *profile-$\mathcal{E}$-alt* :: (′*a*, ′*v*) *Election-Alt* ⇒ (′*a*, ′*v*) *Profile* **where**
profile-$\mathcal{E}$-alt E ≡ *Product-Type.snd* (*snd-alt E*)

end

Chapter 3

Quotient Rules

3.1 Quotients of Equivalence Relations

theory *Relation-Quotients*
  imports *../Social-Choice-Types/Symmetry-Of-Functions*
begin

3.1.1 Definitions

fun *singleton-set* :: $'x$ *set* $\Rightarrow$ $'x$ **where**
  singleton-set s = (*if* (*card* s = *1*) *then* (*the-inv* (λ x. $\{x\}$) s) *else undefined*)
— This is undefined if card s != 1. Note that "undefined = undefined" is the only provable equality for undefined.

For a given function, we define a function on sets that maps each set to the unique image under f of its elements, if one exists. Otherwise, the result is undefined.

fun $\pi_{\mathcal{Q}}$ :: ($'x \Rightarrow 'y$) $\Rightarrow$ ($'x$ *set* $\Rightarrow$ $'y$) **where**
  $\pi_{\mathcal{Q}}$ f s = *singleton-set* (f ‘ s)

For a given function f on sets and a mapping from elements to sets, we define a function on the set element type that maps each element to the image of its corresponding set under f. A natural mapping is from elements to their classes under a relation.

fun *inv-*$\pi_{\mathcal{Q}}$ :: ($'x \Rightarrow 'x$ *set*) $\Rightarrow$ ($'x$ *set* $\Rightarrow$ $'y$) $\Rightarrow$ ($'x \Rightarrow 'y$) **where**
  inv-$\pi_{\mathcal{Q}}$ *cls* f x = f (*cls* x)

fun *relation-class* :: $'x$ *rel* $\Rightarrow$ $'x \Rightarrow 'x$ *set* **where**
  relation-class r x = r ‘‘ $\{x\}$

3.1.2 Well-Definedness

lemma *singleton-set-undef-if-card-neq-one*:
  fixes s :: $'x$ *set*

assumes *card s* $\neq$ *1*
shows *singleton-set s* = *undefined*
using *assms*
by *simp*

lemma *singleton-set-def-if-card-one*:
fixes *s* :: *'x set*
assumes *card s* = *1*
shows $\exists!$ *x*. *x* = *singleton-set s* $\wedge$ $\{x\}$ = *s*
using *assms card-1-singletonE inj-def singleton-inject the-inv-f-f*
unfolding *singleton-set.simps*
by (*metis* (*mono-tags*, *lifting*))

If the given function is invariant under an equivalence relation, the induced function on sets is well-defined for all equivalence classes of that relation.

theorem *pass-to-quotient*:
fixes
f :: *'x* $\Rightarrow$ *'y* **and**
r :: *'x rel* **and**
s :: *'x set*
assumes
f respects r **and**
equiv s r
shows $\forall$ *t* $\in$ *s* // *r*. $\forall$ *x* $\in$ *t*. $\pi_{\mathcal{Q}}$ *f t* = *f x*
proof (*safe*)
fix
t :: *'x set* **and**
x :: *'x*
have $\forall$ *y* $\in$ *r*``$\{x\}$. (*x*, *y*) $\in$ *r*
unfolding *Image-def*
by *simp*
hence *func-eq-x*: $\{f\ y \mid y.\ y \in r``\{x\}\} = \{f\ x \mid y.\ y \in r``\{x\}\}$
using *assms*
unfolding *congruent-def*
by *fastforce*
assume
t $\in$ *s* // *r* **and**
x-in-t: *x* $\in$ *t*
moreover from *this* **have** *r* `` $\{x\}$ $\in$ *s* // *r*
using *assms quotient-eq-iff equiv-class-eq-iff quotientI*
by *metis*
ultimately have *r-img-elem-x-eq-t*: *r* `` $\{x\}$ = *t*
using *assms quotient-eq-iff Image-singleton-iff*
by *metis*
hence $\{f\ x \mid y.\ y \in r``\{x\}\} = \{f\ x\}$
using *x-in-t*
by *blast*
hence *f* ` *t* = $\{f\ x\}$
using *Setcompr-eq-image r-img-elem-x-eq-t func-eq-x*

by *metis*
thus $\pi_\mathcal{Q}$ *f t* = *f x*
using *singleton-set-def-if-card-one is-singletonI is-singleton-altdef the-elem-eq*
unfolding $\pi_\mathcal{Q}$*.simps*
by *metis*
qed

A function on sets induces a function on the element type that is invariant under a given equivalence relation.

theorem *pass-to-quotient-inv*:
fixes
f :: *'x set* ⇒ *'x* **and**
r :: *'x rel* **and**
s :: *'x set*
assumes *equiv s r*
defines *induced-fun* ≡ (*inv*-$\pi_\mathcal{Q}$ (*relation-class r*) *f*)
shows
induced-fun respects r **and**
∀ *A* ∈ *s* // *r*. $\pi_\mathcal{Q}$ *induced-fun A* = *f A*
proof (*safe*)
have ∀ (*a*, *b*) ∈ *r*. *relation-class r a* = *relation-class r b*
using *assms equiv-class-eq*
unfolding *relation-class.simps*
by *fastforce*
hence ∀ (*a*, *b*) ∈ *r*. *induced-fun a* = *induced-fun b*
unfolding *induced-fun-def inv*-$\pi_\mathcal{Q}$*.simps*
by *auto*
thus *induced-fun respects r*
unfolding *congruent-def*
by *metis*
moreover fix *A* :: *'x set*
assume *A* ∈ *s* // *r*
moreover with *assms*
obtain *a* :: *'x* **where**
a ∈ *A* **and**
A-eq-rel-class-r-a: *A* = *relation-class r a*
using *equiv-Eps-in proj-Eps*
unfolding *proj-def relation-class.simps*
by *metis*
ultimately have $\pi_\mathcal{Q}$ *induced-fun A* = *induced-fun a*
using *pass-to-quotient assms*
by *blast*
thus $\pi_\mathcal{Q}$ *induced-fun A* = *f A*
using *A-eq-rel-class-r-a*
unfolding *induced-fun-def*
by *simp*
qed

3.1.3 Equivalence Relations

lemma *equiv-rel-restr*:
  fixes
    s :: $'x$ *set* **and**
    t :: $'x$ *set* **and**
    r :: $'x$ *rel*
  assumes
    equiv s r **and**
    $t \subseteq s$
  shows *equiv* t (*Restr* r t)
proof (*unfold equiv-def refl-on-def*, *safe*)
  fix x :: $'x$
  assume $x \in t$
  thus $(x, x) \in r$
    using *assms*
    unfolding *equiv-def refl-on-def*
    by *blast*
next
  show *sym* (*Restr* r t)
    using *assms*
    unfolding *equiv-def sym-def*
    by *blast*
next
  show *Relation.trans* (*Restr* r t)
    using *assms*
    unfolding *equiv-def Relation.trans-def*
    by *blast*
qed

lemma *rel-ind-by-group-act-equiv*:
  fixes
    m :: $'x$ *monoid* **and**
    s :: $'y$ *set* **and**
    φ :: $('x, 'y)$ *binary-fun*
  assumes *group-action* m s φ
  shows *equiv* s (*rel-induced-by-action* (*carrier* m) s φ)
proof (*unfold equiv-def refl-on-def sym-def Relation.trans-def rel-induced-by-action.simps*,
    clarsimp, *safe*)
  fix y :: $'y$
  assume $y \in s$
  hence $\varphi\ \mathbf{1}\ _m\ y = y$
    using *assms group-action.id-eq-one restrict-apply'*
    by *metis*
  thus $\exists\ g \in$ *carrier* m. $\varphi\ g\ y = y$
    using *assms group.is-monoid group-hom.axioms*
    unfolding *group-action-def*
    by *blast*
next
  fix

$y :: 'y$ **and**
$g :: 'x$
assume
y-in-s: $y \in s$ **and**
carrier-g: $g \in carrier\ m$
hence $y = \varphi\ (inv\ _m\ g)\ (\varphi\ g\ y)$
using *assms*
by (*simp add*: *group-action.orbit-sym-aux*)
thus $\exists\ h \in carrier\ m.\ \varphi\ h\ (\varphi\ g\ y) = y$
using *assms carrier-g group.inv-closed group-action.group-hom group-hom.axioms*(*1*)
by *metis*
next
fix
$y :: 'y$ **and**
$g :: 'x$ **and**
$h :: 'x$
assume
y-in-s: $y \in s$ **and**
carrier-g: $g \in carrier\ m$ **and**
carrier-h: $h \in carrier\ m$
hence $\varphi\ (h \otimes\ _m\ g)\ y = \varphi\ h\ (\varphi\ g\ y)$
using *assms*
by (*simp add*: *group-action.composition-rule*)
thus $\exists\ f \in carrier\ m.\ \varphi\ f\ y = \varphi\ h\ (\varphi\ g\ y)$
using *assms carrier-g carrier-h group-action.group-hom*
group-hom.axioms(*1*) *monoid.m-closed*
unfolding *group-def*
by *metis*
qed

end

3.2 Quotients of Equivalence Relations on Election Sets

theory *Election-Quotients*
imports *Relation-Quotients*
../Social-Choice-Types/Voting-Symmetry
../Social-Choice-Types/Ordered-Relation
HOL−Analysis.Convex
HOL−Analysis.Cartesian-Space
begin

3.2.1 Auxiliary Lemmas

lemma *obtain-partition*:

fixes
$X :: 'x\ set$ **and**
$N :: 'y \Rightarrow nat$ **and**
$Y :: 'y\ set$
assumes
finite X **and**
finite Y **and**
sum $N\ Y = card\ X$
shows $\exists\ \mathcal{X}.\ X = \bigcup\ \{\mathcal{X}\ i \mid i.\ i \in Y\} \wedge (\forall\ i \in Y.\ card\ (\mathcal{X}\ i) = N\ i) \wedge$
$(\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$
using *assms*
proof (*induction card Y arbitrary*: $X\ Y$)
case *0*
fix
$X :: 'x\ set$ **and**
$Y :: 'y\ set$
assume
fin-X: *finite* X **and**
card-X: *sum* $N\ Y = card\ X$ **and**
fin-Y: *finite* Y **and**
card-Y: $0 = card\ Y$
let $?\mathcal{X} = \lambda\ y.\ \{\}$
have *Y-empty*: $Y = \{\}$
using *0 fin-Y card-Y*
by *simp*
hence *sum* $N\ Y = 0$
by *simp*
hence $X = \{\}$
using *fin-X card-X*
by *simp*
hence $X = \bigcup\ \{?\mathcal{X}\ i \mid i.\ i \in Y\}$
by *blast*
moreover have $\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow ?\mathcal{X}\ i \cap ?\mathcal{X}\ j = \{\}$
by *blast*
ultimately show
$\exists\ \mathcal{X}.\ X = \bigcup\ \{\mathcal{X}\ i \mid i.\ i \in Y\} \wedge$
$(\forall\ i \in Y.\ card\ (\mathcal{X}\ i) = N\ i) \wedge$
$(\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$
using *Y-empty*
by *simp*
next
case (*Suc x*)
fix
$x :: nat$ **and**
$X :: 'x\ set$ **and**
$Y :: 'y\ set$
assume
card-Y: *Suc* $x = card\ Y$ **and**
fin-Y: *finite* Y **and**

fin-X: *finite X* **and**
card-X: *sum N Y* = *card X* **and**
hyp:
$\bigwedge$ *Y* (*X*::*'x set*).
x = *card Y* $\Longrightarrow$
finite X $\Longrightarrow$
finite Y $\Longrightarrow$
sum N Y = *card X* $\Longrightarrow$
$\exists$ $\mathcal{X}$.
$X = \bigcup \{\mathcal{X}\ i \mid i.\ i \in Y\} \wedge$
$(\forall\ i \in Y.\ card\ (\mathcal{X}\ i) = N\ i) \wedge$
$(\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$
then obtain
Y' :: *'y set* **and**
y :: *'y* **where**
ins-Y: $Y = insert\ y\ Y'$ **and**
card-Y': $card\ Y' = x$ **and**
fin-Y': $finite\ Y'$ **and**
y-not-in-Y': $y \notin Y'$
using *card-Suc-eq-finite*
by (*metis* (*no-types*, *lifting*))
hence $N\ y \leq card\ X$
using *card-X card-Y fin-Y le-add1 n-not-Suc-n sum.insert*
by *metis*
then obtain X' :: *'x set* **where**
X'-in-X: $X' \subseteq X$ **and**
card-X': $card\ X' = N\ y$
using *fin-X ex-card*
by *metis*
hence $finite\ (X - X') \wedge card\ (X - X') = sum\ N\ Y'$
using *card-Y card-X fin-X fin-Y ins-Y card-Y' fin-Y'*
Suc-n-not-n add-diff-cancel-left' card-Diff-subset card-insert-if
finite-Diff finite-subset sum.insert
by *metis*
then obtain $\mathcal{X}$ :: *'y* $\Rightarrow$ *'x set* **where**
part: $X - X' = \bigcup \{\mathcal{X}\ i \mid i.\ i \in Y'\}$ **and**
disj: $\forall\ i\ j.\ i \neq j \longrightarrow i \in Y' \wedge j \in Y' \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\}$ **and**
card: $\forall\ i \in Y'.\ card\ (\mathcal{X}\ i) = N\ i$
using *hyp*[*of* Y' $X - X'$] *fin-Y' card-Y'*
by *auto*
then obtain $\mathcal{X}'$ :: *'y* $\Rightarrow$ *'x set* **where**
map': $\mathcal{X}' = (\lambda\ z.\ if\ (z = y)\ then\ X'\ else\ \mathcal{X}\ z)$
by *simp*
hence *eq-$\mathcal{X}$*: $\forall\ i \in Y'.\ \mathcal{X}'\ i = \mathcal{X}\ i$
using *y-not-in-Y'*
by *simp*
have $Y = \{y\} \cup Y'$
using *ins-Y*
by *simp*

hence $\forall\ f.\ \{f\ i \mid i.\ i \in Y\} = \{f\ y\} \cup \{f\ i \mid i.\ i \in Y'\}$
  by *blast*
hence $\{\mathcal{X}'\ i \mid i.\ i \in Y\} = \{\mathcal{X}'\ y\} \cup \{\mathcal{X}'\ i \mid i.\ i \in Y'\}$
  by *metis*
hence $\bigcup\ \{\mathcal{X}'\ i \mid i.\ i \in Y\} = \mathcal{X}'\ y \cup \bigcup\ \{\mathcal{X}'\ i \mid i.\ i \in Y'\}$
  by *simp*
also have $\mathcal{X}'\ y = X'$
  using *map'*
  by *presburger*
also have $\bigcup\ \{\mathcal{X}'\ i \mid i.\ i \in Y'\} = \bigcup\ \{\mathcal{X}\ i \mid i.\ i \in Y'\}$
  using *eq-$\mathcal{X}$*
  by *blast*
finally have *part'*: $X = \bigcup\ \{\mathcal{X}'\ i \mid i.\ i \in Y\}$
  using *part Diff-partition X'-in-X*
  by *metis*
have $\forall\ i \in Y'.\ \mathcal{X}'\ i \subseteq X - X'$
  using *part eq-$\mathcal{X}$ Setcompr-eq-image UN-upper*
  by *metis*
hence $\forall\ i \in Y'.\ \mathcal{X}'\ i \cap X' = \{\}$
  by *blast*
hence $\forall\ i \in Y'.\ \mathcal{X}'\ i \cap \mathcal{X}'\ y = \{\}$
  using *map'*
  by *simp*
hence $\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X}'\ i \cap \mathcal{X}'\ j = \{\}$
  using *map' disj ins-Y inf.commute insertE*
  by (*metis* (*no-types*, *lifting*))
moreover have $\forall\ i \in Y.\ card\ (\mathcal{X}'\ i) = N\ i$
  using *map' card card-X' ins-Y*
  by *simp*
ultimately show
  $\exists\ \mathcal{X}.\ X = \bigcup\ \{\mathcal{X}\ i \mid i.\ i \in Y\} \wedge$
        $(\forall\ i \in Y.\ card\ (\mathcal{X}\ i) = N\ i) \wedge$
          $(\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$
  using *part'*
  by *blast*
qed

3.2.2 Anonymity Quotient - Grid

fun *anonymity*$_{\mathcal{Q}}$ :: *'a set* $\Rightarrow$ (*'a*, *'v*) *Election set set* **where**
  anonymity$_{\mathcal{Q}}$ A = *quotient* (*fixed-alt-elections* A) (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections* A))

— Counts the occurrences of a ballot per election in a set of elections if the occurrences of the ballot per election coincide for all elections in the set.
fun *vote-count*$_{\mathcal{Q}}$ :: *'a Preference-Relation* $\Rightarrow$ (*'a*, *'v*) *Election set* $\Rightarrow$ *nat* **where**
  vote-count$_{\mathcal{Q}}$ $p = \pi_{\mathcal{Q}}$ (*vote-count* p)

fun *anon-class-to-vec* :: (*'a::finite*, *'v*) *Election set* $\Rightarrow$ (*nat*, *'a Ordered-Preference*)

vec **where**
  anon-class-to-vec $X = (\chi\ p.\ \textit{vote-count}_{\mathcal{Q}}\ (\textit{ord2pref}\ p)\ X)$

We assume all our elections to consist of a fixed finite alternative set of size n and finite subsets of an infinite voter universe. Profiles are linear orders on the alternatives. Then we can work on the natural-number-vectors of dimension n! instead of the equivalence classes of the anonymity relation: Each dimension corresponds to one possible linear order on the alternative set, i.e., the possible preferences. Each equivalence class of elections corresponds to a vector whose entries denote the amount of voters per election in that class who vote the respective corresponding preference.

theorem $\textit{anonymity}_{\mathcal{Q}}$*-iso*:
  assumes *infinite* (*UNIV*::(′*v set*))
  shows *bij-betw* (*anon-class-to-vec*::(′*a*::*finite*, ′*v*) *Election set* ⇒ *nat*^(′*a Ordered-Preference*))
        ($\textit{anonymity}_{\mathcal{Q}}$ (*UNIV*::′*a set*)) (*UNIV*::(*nat*^(′*a Ordered-Preference*)) *set*)
proof (*unfold bij-betw-def inj-on-def*, *standard*, *standard*, *standard*, *standard*)
  fix
    X :: (′*a*, ′*v*) *Election set* **and**
    Y :: (′*a*, ′*v*) *Election set*
  assume
    class-X: *X* ∈ $\textit{anonymity}_{\mathcal{Q}}$ *UNIV* **and**
    class-Y: *Y* ∈ $\textit{anonymity}_{\mathcal{Q}}$ *UNIV* **and**
    eq-vec: *anon-class-to-vec X* = *anon-class-to-vec Y*
  have ∀ *E* ∈ *fixed-alt-elections UNIV*. *finite* (*voters-*$\mathcal{E}$ *E*)
    by *simp*
  hence ∀ (*E*, *E*′) ∈ $\textit{anonymity}_{\mathcal{R}}$ (*fixed-alt-elections UNIV*). *finite* (*voters-*$\mathcal{E}$ *E*)
    by *simp*
  moreover have *subset*: *fixed-alt-elections UNIV* ⊆ *valid-elections*
    by *simp*
  ultimately have
    ∀ (*E*, *E*′) ∈ $\textit{anonymity}_{\mathcal{R}}$ (*fixed-alt-elections UNIV*). ∀ *p*. *vote-count p E* = *vote-count p E*′
    using *anon-rel-vote-count*
    by *blast*
  hence *vote-count-invar*: ∀ *p*. (*vote-count p*) *respects* ($\textit{anonymity}_{\mathcal{R}}$ (*fixed-alt-elections UNIV*))
    unfolding *congruent-def*
    by *blast*
  have *foo*: *equiv valid-elections* ($\textit{anonymity}_{\mathcal{R}}$ *valid-elections*)
    using *rel-ind-by-group-act-equiv*[*of* $\textit{anonymity}_{\mathcal{G}}$ *valid-elections* *φ-anon valid-elections*]
        rel-ind-by-coinciding-action-on-subset-eq-restr
    by (*simp add*: *anonymous-group-action.group-action-axioms*)
  moreover have
    ∀ π ∈ *carrier* $\textit{anonymity}_{\mathcal{G}}$.
      ∀ *E* ∈ *fixed-alt-elections UNIV*.
        φ-anon (*fixed-alt-elections UNIV*) π *E* = *φ-anon valid-elections* π *E*

by *simp*
ultimately have *equiv-rel*:
equiv (*fixed-alt-elections UNIV*) (*anonymity*$_\mathcal{R}$ (*fixed-alt-elections UNIV*))
using *subset rel-ind-by-coinciding-action-on-subset-eq-restr*[*of fixed-alt-elections UNIV*
valid-elections carrier anonymity$_\mathcal{G}$ φ*-anon* (*fixed-alt-elections UNIV*)]
equiv-rel-restr
unfolding *anonymity*$_\mathcal{R}$*.simps*
by (*metis* (*no-types*))
with *vote-count-invar*
have *quotient-count*: $\forall$ $X \in$ *anonymity*$_\mathcal{Q}$ *UNIV*. $\forall$ *p*. $\forall$ $E \in X$. *vote-count*$_\mathcal{Q}$ *p X* = *vote-count p E*
using *pass-to-quotient*[*of anonymity*$_\mathcal{R}$ (*fixed-alt-elections UNIV*)]
unfolding *anonymity*$_\mathcal{Q}$*.simps anonymity*$_\mathcal{R}$*.simps vote-count*$_\mathcal{Q}$*.simps*
by *metis*
moreover from *equiv-rel*
obtain
E :: ('*a*, '*v*) *Election* **and**
E' :: ('*a*, '*v*) *Election* **where**
E-in-X: $E \in X$ **and**
E'-in-Y: $E' \in Y$
using *class-X class-Y equiv-Eps-in*
unfolding *anonymity*$_\mathcal{Q}$*.simps*
by *metis*
ultimately have $\forall$ *p*. *vote-count*$_\mathcal{Q}$ *p X* = *vote-count p E* $\wedge$ *vote-count*$_\mathcal{Q}$ *p Y* = *vote-count p E'*
using *class-X class-Y*
by *blast*
moreover with *eq-vec* **have** $\forall$ *p*. *vote-count*$_\mathcal{Q}$ (*ord2pref p*) *X* = *vote-count*$_\mathcal{Q}$ (*ord2pref p*) *Y*
unfolding *anon-class-to-vec.simps*
using *UNIV-I vec-lambda-inverse*
by *metis*
ultimately have $\forall$ *p*. *vote-count* (*ord2pref p*) *E* = *vote-count* (*ord2pref p*) *E'*
by *simp*
hence *eq*: $\forall$ $p \in \{p.$ *linear-order-on* (*UNIV*::'*a set*) *p*$\}$. *vote-count p E* = *vote-count p E'*
using *pref2ord-inverse*
by *metis*
from *equiv-rel class-X class-Y* **have** *subset-fixed-alts*:
$X \subseteq$ *fixed-alt-elections UNIV* $\wedge$ $Y \subseteq$ *fixed-alt-elections UNIV*
unfolding *anonymity*$_\mathcal{Q}$*.simps*
using *in-quotient-imp-subset*
by *blast*
hence *eq-alts*: *alternatives-*$\mathcal{E}$ *E* = *UNIV* $\wedge$ *alternatives-*$\mathcal{E}$ *E'* = *UNIV*
using *E-in-X E'-in-Y*
unfolding *fixed-alt-elections.simps*
by *blast*
with *subset-fixed-alts* **have** *eq-complement*:

$\forall\ p \in UNIV - \{p.\ linear\text{-}order\text{-}on\ (UNIV::'a\ set)\ p\}.$
$\{v \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E\ v = p\} = \{\} \wedge \{v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = p\} = \{\}$
using *E-in-X E′-in-Y*
unfolding *fixed-alt-elections.simps valid-elections-def profile-def*
by *auto*
hence $\forall\ p \in UNIV - \{p.\ linear\text{-}order\text{-}on\ (UNIV::'a\ set)\ p\}.$
$vote\text{-}count\ p\ E = 0 \wedge vote\text{-}count\ p\ E' = 0$
unfolding *card-eq-0-iff vote-count.simps*
by *simp*
with *eq* **have** *eq-vote-count*: $\forall\ p.\ vote\text{-}count\ p\ E = vote\text{-}count\ p\ E'$
using *DiffI UNIV-I*
by *metis*
moreover from *subset-fixed-alts E-in-X E′-in-Y*
have *finite* (*voters-*$\mathcal{E}$ *E*) $\wedge$ *finite* (*voters-*$\mathcal{E}$ *E′*)
unfolding *fixed-alt-elections.simps*
by *blast*
moreover from *subset-fixed-alts E-in-X E′-in-Y*
have $(E, E') \in (fixed\text{-}alt\text{-}elections\ UNIV) \times (fixed\text{-}alt\text{-}elections\ UNIV)$
by *blast*
moreover from *this*
have
$(\forall\ v.\ v \notin voters\text{-}\mathcal{E}\ E \longrightarrow profile\text{-}\mathcal{E}\ E\ v = \{\}) \wedge (\forall\ v.\ v \notin voters\text{-}\mathcal{E}\ E' \longrightarrow profile\text{-}\mathcal{E}\ E'\ v = \{\})$
by *simp*
ultimately have $(E, E') \in anonymity_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)$
using *eq-alts vote-count-anon-rel*
by *metis*
hence $anonymity_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ ``\ \{E\} =$
$anonymity_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ ``\ \{E'\}$
using *equiv-rel equiv-class-eq*
by *metis*
also have $anonymity_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ ``\ \{E\} = X$
using *E-in-X class-X equiv-rel Image-singleton-iff equiv-class-eq quotientE*
unfolding $anonymity_{\mathcal{Q}}.simps$
by (*metis* (*no-types*, *lifting*))
also have $anonymity_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ ``\ \{E'\} = Y$
using *E′-in-Y class-Y equiv-rel Image-singleton-iff equiv-class-eq quotientE*
unfolding $anonymity_{\mathcal{Q}}.simps$
by (*metis* (*no-types*, *lifting*))
finally show $X = Y$
by *simp*
next
have *subset*: $fixed\text{-}alt\text{-}elections\ UNIV \subseteq valid\text{-}elections$
by *simp*
have *equiv valid-elections* ($anonymity_{\mathcal{R}}$ *valid-elections*)
using *rel-ind-by-group-act-equiv*[*of* $anonymity_{\mathcal{G}}$ *valid-elections* φ*-anon valid-elections*]
rel-ind-by-coinciding-action-on-subset-eq-restr
by (*simp add*: *anonymous-group-action.group-action-axioms*)

moreover have
  $\forall\ \pi \in$ *carrier anonymity*$_{\mathcal{G}}$.
    $\forall\ E \in$ *fixed-alt-elections UNIV*.
      φ*-anon* (*fixed-alt-elections UNIV*) π E = φ*-anon valid-elections* π E
  using *subset*
  unfolding φ*-anon.simps*
  by *simp*
ultimately have *equiv-rel*:
  equiv (*fixed-alt-elections UNIV*) (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*))
  using *subset equiv-rel-restr rel-ind-by-coinciding-action-on-subset-eq-restr*[*of*
      fixed-alt-elections UNIV valid-elections carrier anonymity$_{\mathcal{G}}$
      φ*-anon* (*fixed-alt-elections UNIV*)]
  unfolding *anonymity*$_{\mathcal{R}}$*.simps*
  by (*metis* (*no-types*))
have (*UNIV*::((*nat*, *'a Ordered-Preference*) *vec set*)) $\subseteq$
    (*anon-class-to-vec*::(*'a*, *'v*) *Election set* $\Rightarrow$ (*nat*, *'a Ordered-Preference*) *vec*) ‘
    anonymity$_{\mathcal{Q}}$ *UNIV*
proof (*unfold anon-class-to-vec.simps*, *safe*)
  fix *x* :: (*nat*, *'a Ordered-Preference*) *vec*
  have *finite* (*UNIV*::(*'a Ordered-Preference set*))
    by *simp*
  hence *finite* $\{x\$i \mid i.\ i \in UNIV\}$
    using *finite-Atleast-Atmost-nat*
    by *blast*
  hence *sum* (λ *i*. *x*\$*i*) *UNIV* $< \infty$
    using *enat-ord-code*
    by *simp*
  moreover have $0 \leq$ *sum* (λ *i*. *x*\$*i*) *UNIV*
    by *blast*
  ultimately obtain *V* :: *'v set* **where**
    fin-V: *finite V* **and**
    card V = *sum* (λ *i*. *x*\$*i*) *UNIV*
    using *assms infinite-arbitrarily-large*
    by *metis*
  then obtain *X'* :: *'a Ordered-Preference* $\Rightarrow$ *'v set* **where**
    card': $\forall\ i.$ *card* ($X'\ i$) = $x\$i$ **and**
    partition': $V = \bigcup\ \{X'\ i \mid i.\ i \in UNIV\}$ **and**
    disjoint': $\forall\ i\ j.\ i \neq j \longrightarrow X'\ i \cap X'\ j = \{\}$
    using *obtain-partition*[*of V UNIV* (\$) *x*]
    by *auto*
  obtain *X* :: *'a Preference-Relation* $\Rightarrow$ *'v set* **where**
    def-X: *X* = (λ *i*. *if* ($i \in \{i.$ *linear-order i*$\}$) *then* X' (*pref2ord i*) *else* $\{\}$)
    by *simp*
  hence $\{X\ i \mid i.\ i \notin \{i.$ *linear-order i*$\}\} \subseteq \{\{\}\}$
    by *auto*
  moreover have
    $\{X\ i \mid i.\ i \in \{i.$ *linear-order i*$\}\} = \{X'$ (*pref2ord i*) $\mid i.\ i \in \{i.$ *linear-order*
i$\}\}$

using *def-X*
by *metis*
moreover have
$\{X\ i \mid i.\ i \in \mathit{UNIV}\} =$
$\{X\ i \mid i.\ i \in \{i.\ \mathit{linear\text{-}order}\ i\}\} \cup \{X\ i \mid i.\ i \in \mathit{UNIV} - \{i.\ \mathit{linear\text{-}order}$
$i\}\}$
by *blast*
ultimately have
$\{X\ i \mid i.\ i \in \mathit{UNIV}\} = \{X'\ (\mathit{pref2ord}\ i) \mid i.\ i \in \{i.\ \mathit{linear\text{-}order}\ i\}\} \vee$
$\{X\ i \mid i.\ i \in \mathit{UNIV}\} = \{X'\ (\mathit{pref2ord}\ i) \mid i.\ i \in \{i.\ \mathit{linear\text{-}order}\ i\}\} \cup \{\{\}\}$
by *auto*
also have $\{X'\ (\mathit{pref2ord}\ i) \mid i.\ i \in \{i.\ \mathit{linear\text{-}order}\ i\}\} = \{X'\ i \mid i.\ i \in \mathit{UNIV}\}$
using *iso-tuple-UNIV-I pref2ord-cases*
by *metis*
finally have
$\{X\ i \mid i.\ i \in \mathit{UNIV}\} = \{X'\ i \mid i.\ i \in \mathit{UNIV}\} \vee$
$\{X\ i \mid i.\ i \in \mathit{UNIV}\} = \{X'\ i \mid i.\ i \in \mathit{UNIV}\} \cup \{\{\}\}$
by *simp*
hence $\bigcup\ \{X\ i \mid i.\ i \in \mathit{UNIV}\} = \bigcup\ \{X'\ i \mid i.\ i \in \mathit{UNIV}\}$
using *Sup-union-distrib ccpo-Sup-singleton sup-bot.right-neutral*
by (*metis* (*no-types*, *lifting*))
hence *partition*: $V = \bigcup\ \{X\ i \mid i.\ i \in \mathit{UNIV}\}$
using *partition′*
by *simp*
moreover have $\forall\ i\ j.\ i \neq j \longrightarrow X\ i \cap X\ j = \{\}$
using *disjoint′ def-X pref2ord-inject*
by *auto*
ultimately have $\forall\ v \in V.\ \exists!\ i.\ v \in X\ i$
by *auto*
then obtain $p' :: {'v} \Rightarrow {'a}\ \mathit{Preference\text{-}Relation}$ **where**
p-X: $\forall\ v \in V.\ v \in X\ (p'\ v)$ **and**
p-disj: $\forall\ v \in V.\ \forall\ i.\ i \neq p'\ v \longrightarrow v \notin X\ i$
by *metis*
then obtain $p :: {'v} \Rightarrow {'a}\ \mathit{Preference\text{-}Relation}$ **where**
p-def: $p = (\lambda\ v.\ \mathit{if}\ v \in V\ \mathit{then}\ p'\ v\ \mathit{else}\ \{\})$
by *simp*
hence *lin-ord*: $\forall\ v \in V.\ \mathit{linear\text{-}order}\ (p\ v)$
using *def-X p-X p-disj*
by *fastforce*
hence *valid*: $(\mathit{UNIV},\ V,\ p) \in \mathit{fixed\text{-}alt\text{-}elections}\ \mathit{UNIV}$
using *fin-V*
unfolding *p-def fixed-alt-elections.simps valid-elections-def profile-def*
by *auto*
hence $\forall\ i.\ \forall\ E \in \mathit{anonymity}_{\mathcal{R}}\ (\mathit{fixed\text{-}alt\text{-}elections}\ \mathit{UNIV})\ ``\ \{(\mathit{UNIV},\ V,\ p)\}.$
$\mathit{vote\text{-}count}\ i\ E = \mathit{vote\text{-}count}\ i\ (\mathit{UNIV},\ V,\ p)$
using *anon-rel-vote-count*[*of* $(\mathit{UNIV},\ V,\ p)$ - *fixed-alt-elections UNIV*]
fin-V subset
by *simp*
moreover have $(\mathit{UNIV},\ V,\ p) \in \mathit{anonymity}_{\mathcal{R}}\ (\mathit{fixed\text{-}alt\text{-}elections}\ \mathit{UNIV})\ ``$

$\{(UNIV, V, p)\}$
using *equiv-rel valid*
unfolding *Image-def equiv-def refl-on-def*
by *blast*
ultimately have *eq-vote-count*:
$\forall i.$ *vote-count* i ‘ (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\}$) =
$\{$*vote-count* i $(UNIV, V, p)\}$
by *blast*
have $\forall i. \forall v \in V. p\ v = i \longleftrightarrow v \in X\ i$
using *p-X p-disj*
unfolding *p-def*
by *metis*
hence $\forall i. \{v \in V. p\ v = i\} = \{v \in V. v \in X\ i\}$
by *blast*
moreover have $\forall i. X\ i \subseteq V$
using *partition*
by *blast*
ultimately have *rewr-preimg*: $\forall i. \{v \in V. p\ v = i\} = X\ i$
by *auto*
hence $\forall i \in \{i.$ *linear-order* $i\}.$ *vote-count* i $(UNIV, V, p) = x\$($*pref2ord* $i)$
using *def-X card'*
by *simp*
hence $\forall i \in \{i.$ *linear-order* $i\}.$
vote-count i ‘ (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\}$) = $\{x\$($*pref2ord* $i)\}$
using *eq-vote-count*
by *metis*
hence
$\forall i \in \{i.$ *linear-order* $i\}.$
vote-count$_{\mathcal{Q}}$ i (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\}$) = $x\$($*pref2ord* $i)$
unfolding *vote-count*$_{\mathcal{Q}}$*.simps* $\pi_{\mathcal{Q}}$*.simps singleton-set.simps*
using *is-singleton-altdef singleton-set-def-if-card-one*
by *fastforce*
hence $\forall i.$ *vote-count*$_{\mathcal{Q}}$ (*ord2pref* i) (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\}$)
$= x\$i$
using *ord2pref ord2pref-inverse*
by *metis*
hence *anon-class-to-vec* (*anonymity*$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\}$) $= x$
using *anon-class-to-vec.simps vec-lambda-unique*
by (*metis* (*no-types*, *lifting*))
moreover have
anonymity$_{\mathcal{R}}$ (*fixed-alt-elections UNIV*) ‘‘ $\{(UNIV, V, p)\} \in$ *anonymity*$_{\mathcal{Q}}$ *UNIV*
using *valid*
unfolding *anonymity*$_{\mathcal{Q}}$*.simps quotient-def*

by *blast*
ultimately show
$x \in (\lambda\ X::(('a,\ 'v)\ Election\ set).\ \chi\ p.\ vote\text{-}count_{\mathcal{Q}}\ (ord2pref\ p)\ X)$ ‘ $anonymity_{\mathcal{Q}}$ $UNIV$
using *anon-class-to-vec.elims*
by *blast*
qed
thus $(anon\text{-}class\text{-}to\text{-}vec::('a,\ 'v)\ Election\ set \Rightarrow (nat,\ 'a\ Ordered\text{-}Preference)\ vec)$ ‘
$anonymity_{\mathcal{Q}}\ UNIV = (UNIV::((nat,\ 'a\ Ordered\text{-}Preference)\ vec\ set))$
by *blast*
qed

3.2.3 Homogeneity Quotient - Simplex

fun $vote\text{-}fraction :: 'a\ Preference\text{-}Relation \Rightarrow ('a,\ 'v)\ Election \Rightarrow rat$ **where**
$vote\text{-}fraction\ r\ E =$
$(if\ (finite\ (voters\text{-}\mathcal{E}\ E) \wedge voters\text{-}\mathcal{E}\ E \neq \{\})$
$then\ (Fract\ (vote\text{-}count\ r\ E)\ (card\ (voters\text{-}\mathcal{E}\ E)))\ else\ 0)$

fun $anon\text{-}hom_{\mathcal{R}} :: ('a,\ 'v)\ Election\ set \Rightarrow ('a,\ 'v)\ Election\ rel$ **where**
$anon\text{-}hom_{\mathcal{R}}\ \mathcal{E} =$
$\{(E,\ E')\ |\ E\ E'.\ E \in \mathcal{E} \wedge E' \in \mathcal{E} \wedge (finite\ (voters\text{-}\mathcal{E}\ E) = finite\ (voters\text{-}\mathcal{E}\ E'))$ $\wedge$
$(\forall\ r.\ vote\text{-}fraction\ r\ E = vote\text{-}fraction\ r\ E')\}$

fun $anon\text{-}hom_{\mathcal{Q}} :: 'a\ set \Rightarrow ('a,\ 'v)\ Election\ set\ set$ **where**
$anon\text{-}hom_{\mathcal{Q}}\ A = quotient\ (fixed\text{-}alt\text{-}elections\ A)\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ A))$

fun $vote\text{-}fraction_{\mathcal{Q}} :: 'a\ Preference\text{-}Relation \Rightarrow ('a,\ 'v)\ Election\ set \Rightarrow rat$ **where**
$vote\text{-}fraction_{\mathcal{Q}}\ p = \pi_{\mathcal{Q}}\ (vote\text{-}fraction\ p)$

fun $anon\text{-}hom\text{-}class\text{-}to\text{-}vec :: ('a::finite,\ 'v)\ Election\ set$
$\Rightarrow (rat,\ 'a\ Ordered\text{-}Preference)\ vec$ **where**
$anon\text{-}hom\text{-}class\text{-}to\text{-}vec\ \mathcal{E} = (\chi\ p.\ vote\text{-}fraction_{\mathcal{Q}}\ (ord2pref\ p)\ \mathcal{E})$

Maps each rational real vector entry to the corresponding rational. If the entry is not rational, the corresponding entry will be undefined.

fun $rat\text{-}vec :: real\hat{}'b \Rightarrow rat\hat{}'b$ **where**
$rat\text{-}vec\ v = (\chi\ p.\ the\text{-}inv\ of\text{-}rat\ (v\$p))$

fun $rat\text{-}vec\text{-}set :: (real\hat{}'b)\ set \Rightarrow (rat\hat{}'b)\ set$ **where**
$rat\text{-}vec\text{-}set\ V = rat\text{-}vec$ ‘ $\{v \in V.\ \forall\ i.\ v\$i \in \mathbb{Q}\}$

definition $standard\text{-}basis :: (real\hat{}'b)\ set$ **where**
$standard\text{-}basis = \{v.\ \exists\ b.\ v\$b = 1 \wedge (\forall\ c \neq b.\ v\$c = 0)\}$

The rational points in the simplex.

definition *vote-simplex* :: $(rat^\frown{}'b)$ *set* **where**
vote-simplex = *insert 0* (*rat-vec-set* (*convex hull* (*standard-basis* :: $(real^\frown{}'b)$ *set*)))

Auxiliary Lemmas

lemma *convex-combination-in-convex-hull*:
fixes
$X :: (real^\frown{}'b)$ *set* **and**
$x :: real^\frown{}'b$
assumes $\exists\ f::(real^\frown{}'b) \Rightarrow real.$
$sum\ f\ X = 1 \wedge (\forall\ x \in X.\ f\ x \geq 0) \wedge x = sum\ (\lambda\ x.\ (f\ x) *_R\ x)\ X$
shows $x \in convex\ hull\ X$
using *assms*
proof (*induction card X arbitrary*: *X x*)
case *0*
fix
$X :: (real^\frown{}'b)$ *set* **and**
$x :: real^\frown{}'b$
assume
$0 = card\ X$ **and**
$\exists\ f.\ sum\ f\ X = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x = (\sum\ x \in X.\ f\ x *_R\ x)$
hence $(\forall\ f.\ sum\ f\ X = 0) \wedge (\exists\ f.\ sum\ f\ X = 1)$
using *card-0-eq empty-iff sum.infinite sum.neutral zero-neq-one*
by *metis*
hence $\exists\ f.\ sum\ f\ X = 1 \wedge sum\ f\ X = 0$
by *metis*
hence *False*
using *zero-neq-one*
by *metis*
thus *?case*
by *simp*
next
case (*Suc n*)
fix
$X :: (real^\frown{}'b)$ *set* **and**
$x :: real^\frown{}'b$ **and**
$n :: nat$
assume
card: $Suc\ n = card\ X$ **and**
$\exists\ f.\ sum\ f\ X = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x = (\sum\ x \in X.\ f\ x *_R\ x)$ **and**
hyp: $\bigwedge\ (X::(real^\frown{}'b)\ set)\ x.$
$n = card\ X \Longrightarrow$
$\exists\ f.\ sum\ f\ X = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x = (\sum\ x \in X.\ f\ x *_R\ x) \Longrightarrow$
$x \in convex\ hull\ X$
then obtain $f :: (real^\frown{}'b) \Rightarrow real$ **where**
sum: $sum\ f\ X = 1$ **and**
nonneg: $\forall\ x \in X.\ 0 \leq f\ x$ **and**
x-sum: $x = (\sum\ x \in X.\ f\ x *_R\ x)$
by *blast*

have *card X > 0*
  using *card*
  by *linarith*
hence *fin*: *finite X*
  using *card-gt-0-iff*
  by *blast*
have *n = 0* $\longrightarrow$ *card X = 1*
  using *card*
  by *presburger*
hence *n = 0* $\longrightarrow$ ($\exists$ *y*. *X* = {*y*} $\wedge$ *f y = 1*)
  using *sum nonneg One-nat-def add.right-neutral card-1-singleton-iff empty-iff finite.emptyI sum.insert sum.neutral*
  by (*metis* (*no-types*, *opaque-lifting*))
hence *n = 0* $\longrightarrow$ ($\exists$ *y*. *X* = {*y*} $\wedge$ *x = y*)
  using *x-sum*
  by *fastforce*
hence *n = 0* $\longrightarrow$ *x* $\in$ *X*
  by *blast*
moreover have *n > 0* $\longrightarrow$ *x* $\in$ *convex hull X*
proof (*safe*)
  assume *0 < n*
  hence *card-X-gt-1*: *card X > 1*
    using *card*
    by *simp*
  have ($\forall$ *y* $\in$ *X*. *f y* $\geq$ *1*) $\longrightarrow$ *sum f X* $\geq$ *sum* (λ *x*. *1*) *X*
    using *fin sum-mono*
    by *metis*
  moreover have *sum* (λ *x*. *1*) *X = card X*
    by *force*
  ultimately have ($\forall$ *y* $\in$ *X*. *f y* $\geq$ *1*) $\longrightarrow$ *card X* $\leq$ *sum f X*
    by *force*
  hence ($\forall$ *y* $\in$ *X*. *f y* $\geq$ *1*) $\longrightarrow$ *1 < sum f X*
    using *card-X-gt-1*
    by *linarith*
  then obtain *y* :: *real^'b* **where**
    y-in-X: *y* $\in$ *X* **and**
    f-y-lt-one: *f y < 1*
    using *sum*
    by *auto*
  hence *1 − f y* $\neq$ *0* $\wedge$ $x = f\ y *_R y + (\sum x \in X - \{y\}.\ f\ x *_R x)$
    using *fin sum.remove x-sum*
    by *simp*
  moreover have $\forall\ \alpha \neq 0.\ (\sum x \in X - \{y\}.\ f\ x *_R x) = \alpha *_R (\sum x \in X - \{y\}.\ (f\ x\ /\ \alpha) *_R x)$
    unfolding *scaleR-sum-right*
    by *simp*
  ultimately have *convex-comb*:
    $x = f\ y *_R y + (1 - f\ y) *_R (\sum x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R x)$
    by *simp*

obtain f' :: $real^{\frown\prime}b \Rightarrow real$ **where**
def': $f' = (\lambda\ x.\ f\ x\ /\ (1\ -\ f\ y))$
by *simp*
hence $\forall\ x \in X - \{y\}.\ f'\ x \geq 0$
using *nonneg f-y-lt-one*
by *fastforce*
moreover have $sum\ f'\ (X - \{y\}) = (sum\ (\lambda\ x.\ f\ x)\ (X - \{y\}))\ /\ (1 - f\ y)$
unfolding *def' sum-divide-distrib*
by *simp*
moreover have $(sum\ (\lambda\ x.\ f\ x)\ (X - \{y\}))\ /\ (1 - f\ y) = (1 - f\ y)\ /\ (1 - f$
$y)$
using *sum y-in-X*
by (*simp add*: *fin sum.remove*)
moreover have $(1 - f\ y)\ /\ (1 - f\ y) = 1$
using *f-y-lt-one*
by *simp*
ultimately have
$sum\ f'\ (X - \{y\}) = 1 \wedge (\forall\ x \in X - \{y\}.\ 0 \leq f'\ x) \wedge$
$(\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R x) = (\sum\ x \in X - \{y\}.\ f'\ x *_R x)$
using *def'*
by *metis*
hence $\exists\ f'.\ sum\ f'\ (X - \{y\}) = 1 \wedge (\forall\ x \in X - \{y\}.\ 0 \leq f'\ x) \wedge$
$(\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R x) = (\sum\ x \in X - \{y\}.\ f'\ x *_R$
$x)$
by *metis*
moreover have $card\ (X - \{y\}) = n$
using *card y-in-X*
by *simp*
ultimately have $(\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R x) \in convex\ hull\ (X$
$- \{y\})$
using *hyp*
by *blast*
hence $(\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R x) \in convex\ hull\ X$
using *Diff-subset hull-mono in-mono*
by (*metis* (*no-types*, *lifting*))
moreover have $f\ y \geq 0 \wedge 1 - f\ y \geq 0$
using *f-y-lt-one nonneg y-in-X*
by *simp*
moreover have $f\ y + (1 - f\ y) \geq 0$
by *simp*
moreover have $y \in convex\ hull\ X$
using *y-in-X*
by (*simp add*: *hull-inc*)
moreover have
$\forall\ x\ y.\ x \in convex\ hull\ X \wedge y \in convex\ hull\ X \longrightarrow$
$(\forall\ a \geq 0.\ \forall\ b \geq 0.\ a + b = 1 \longrightarrow a *_R x + b *_R y \in convex\ hull\ X)$
using *convex-def convex-convex-hull*
by (*metis* (*no-types*, *opaque-lifting*))
ultimately show $x \in convex\ hull\ X$

using *convex-comb*
by *simp*
qed
ultimately show $x \in$ *convex hull X*
using *hull-inc*
by *fastforce*
qed

lemma *standard-simplex-rewrite*: *convex hull standard-basis*
$= \{v::(real^{\frown\prime}b).\ (\forall\ i.\ v\$i \geq 0) \wedge sum\ ((\$)\ v)\ UNIV = 1\}$
proof (*unfold convex-def hull-def*, *standard*)
let $?simplex = \{v::\ (real^{\frown\prime}b).\ (\forall\ i.\ v\$i \geq 0) \wedge sum\ ((\$)\ v)\ UNIV = 1\}$
have *fin-dim*: *finite* (*UNIV*::$'b$ *set*)
by *simp*
have $\forall\ x::(real^{\frown\prime}b).\ \forall\ y.\ sum\ ((\$)\ (x + y))\ UNIV = sum\ ((\$)\ x)\ UNIV + sum\ ((\$)\ y)\ UNIV$
by (*simp add*: *sum.distrib*)
hence $\forall\ x::(real^{\frown\prime}b).\ \forall\ y.\ \forall\ u\ v.$
$sum\ ((\$)\ (u *_R x + v *_R y))\ UNIV = sum\ ((\$)\ (u *_R x))\ UNIV + sum\ ((\$)\ (v *_R y))\ UNIV$
by *blast*
moreover have $\forall\ x\ u.\ sum\ ((\$)\ (u *_R x))\ UNIV = u *_R (sum\ ((\$)\ x)\ UNIV)$
using *scaleR-right.sum sum.cong vector-scaleR-component*
by (*metis* (*mono-tags*, *lifting*))
ultimately have $\forall\ x::(real^{\frown\prime}b).\ \forall\ y.\ \forall\ u\ v.$
$sum\ ((\$)\ (u *_R x + v *_R y))\ UNIV = u *_R (sum\ ((\$)\ x)\ UNIV) + v *_R (sum\ ((\$)\ y)\ UNIV)$
by (*metis* (*no-types*))
moreover have $\forall\ x \in ?simplex.\ sum\ ((\$)\ x)\ UNIV = 1$
by *simp*
ultimately have
$\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u\ v.\ sum\ ((\$)\ (u *_R x + v *_R y))\ UNIV = u *_R 1 + v *_R 1$
by (*metis* (*no-types*, *lifting*))
hence $\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u\ v.\ sum\ ((\$)\ (u *_R x + v *_R y))\ UNIV = u + v$
by *simp*
moreover have
$\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u \geq 0.\ \forall\ v \geq 0.$
$u + v = 1 \longrightarrow (\forall\ i.\ (u *_R x + v *_R y)\$i \geq 0)$
by *simp*
ultimately have *simplex-convex*:
$\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u \geq 0.\ \forall\ v \geq 0.$
$u + v = 1 \longrightarrow u *_R x + v *_R y \in ?simplex$
by *simp*
have *entries*: $\forall\ v::(real^{\frown\prime}b) \in standard\text{-}basis.\ \exists\ b.\ v\$b = 1 \wedge (\forall\ c.\ c \neq b \longrightarrow v\$c = 0)$
unfolding *standard-basis-def*
by *simp*

then obtain *one* :: $real^\frown{}'b \Rightarrow{} 'b$ **where**
def: $\forall\ v \in standard\text{-}basis.\ v\$(one\ v) = 1 \wedge (\forall\ i \neq one\ v.\ v\$i = 0)$
by *metis*
hence $\forall\ v{::}(real^\frown{}'b) \in standard\text{-}basis.\ \forall\ b.\ v\$b = 0 \vee v\$b = 1$
by *metis*
hence *geq-0*: $\forall\ v{::}(real^\frown{}'b) \in standard\text{-}basis.\ \forall\ b.\ v\$b \geq 0$
using *dual-order.refl zero-less-one-class.zero-le-one*
by *metis*
moreover have $\forall\ v{::}(real^\frown{}'b) \in standard\text{-}basis.$
$sum\ ((\$)\ v)\ UNIV = sum\ ((\$)\ v)\ (UNIV - \{one\ v\}) + v\$(one\ v)$
unfolding *def*
using *add.commute finite insert-UNIV sum.insert-remove*
by *metis*
moreover have $\forall\ v \in standard\text{-}basis.\ sum\ ((\$)\ v)\ (UNIV - \{one\ v\}) + v\$(one\ v) = 1$
using *def*
by *simp*
ultimately have $standard\text{-}basis \subseteq ?simplex$
by *force*
with *simplex-convex*
have $?simplex \in$
$\{t.\ (\forall\ x \in t.\ \forall\ y \in t.\ \forall\ u \geq 0.\ \forall\ v \geq 0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in t) \wedge$
$standard\text{-}basis \subseteq t\}$
by *blast*
thus $\bigcap\ \{t.\ (\forall\ x \in t.\ \forall\ y \in t.\ \forall\ u \geq 0.\ \forall\ v \geq 0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in t) \wedge$
$standard\text{-}basis \subseteq t\} \subseteq ?simplex$
by *blast*
next
show $\{v.\ (\forall\ i.\ 0 \leq v\ \$\ i) \wedge sum\ ((\$)\ v)\ UNIV = 1\} \subseteq$
$\bigcap\ \{t.\ (\forall\ x \in t.\ \forall\ y \in t.\ \forall\ u \geq 0.\ \forall\ v \geq 0.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in t) \wedge$
$(standard\text{-}basis{::}((real^\frown{}'b)\ set)) \subseteq t\}$
proof
fix
$x :: real^\frown{}'b$ **and**
$X :: (real^\frown{}'b)\ set$
assume *convex-comb*: $x \in \{v.\ (\forall\ i.\ 0 \leq v\ \$\ i) \wedge sum\ ((\$)\ v)\ UNIV = 1\}$
have $\forall\ v \in standard\text{-}basis.\ \exists\ b.\ v\$b = 1 \wedge (\forall\ b' \neq b.\ v\$b' = 0)$
unfolding *standard-basis-def*
by *simp*
then obtain *ind* :: $(real^\frown{}'b) \Rightarrow{} 'b$ **where**
ind-1: $\forall\ v \in standard\text{-}basis.\ v\$(ind\ v) = 1$ **and**
ind-0: $\forall\ v \in standard\text{-}basis.\ \forall\ b \neq (ind\ v).\ v\$b = 0$
by *metis*
hence $\forall\ v\ v'.\ v \in standard\text{-}basis \wedge v' \in standard\text{-}basis \longrightarrow ind\ v = ind\ v' \longrightarrow$
$(\forall\ b.\ v\$b = v'\$b)$
by *metis*

hence *inj-ind*:
$\forall\ v\ v'.\ v \in \textit{standard-basis} \wedge v' \in \textit{standard-basis} \longrightarrow \textit{ind}\ v = \textit{ind}\ v' \longrightarrow v = v'$
unfolding *vec-eq-iff*
by *simp*
hence *inj-on ind standard-basis*
unfolding *inj-on-def*
by *blast*
hence *bij*: *bij-betw ind standard-basis* (*ind* ‘ *standard-basis*)
unfolding *bij-betw-def*
by *simp*
obtain *ind-inv* :: $'b \Rightarrow (\textit{real}^\frown{}'b)$ **where**
char-vec: $\textit{ind-inv} = (\lambda\ b.\ (\chi\ i.\ \textit{if}\ i = b\ \textit{then}\ 1\ \textit{else}\ 0))$
by *blast*
hence *in-basis*: $\forall\ b.\ \textit{ind-inv}\ b \in \textit{standard-basis}$
unfolding *standard-basis-def*
by *simp*
moreover from *this*
have *ind-inv-map*: $\forall\ b.\ \textit{ind}\ (\textit{ind-inv}\ b) = b$
using *char-vec ind-0 ind-1 axis-def axis-nth zero-neq-one*
by *metis*
ultimately have $\forall\ b.\ \exists\ v.\ v \in \textit{standard-basis} \wedge b = \textit{ind}\ v$
by *metis*
hence *univ*: *ind* ‘ *standard-basis* = *UNIV*
by *blast*
have *bij-inv*: *bij-betw ind-inv UNIV standard-basis*
using *ind-inv-map bij bij-betw-byWitness*[*of UNIV ind*] *in-basis inj-ind*
unfolding *image-subset-iff*
by *simp*
obtain *f* :: $(\textit{real}^\frown{}'b) \Rightarrow \textit{real}$ **where**
def: $f = (\lambda\ v.\ \textit{if}\ v \in \textit{standard-basis}\ \textit{then}\ x\$(\textit{ind}\ v)\ \textit{else}\ 0)$
by *blast*
hence $\textit{sum}\ f\ \textit{standard-basis} = \textit{sum}\ (\lambda\ v.\ x\$(\textit{ind}\ v))\ \textit{standard-basis}$
by *simp*
also have $\textit{sum}\ (\lambda\ v.\ x\$(\textit{ind}\ v))\ \textit{standard-basis} = \textit{sum}\ ((\$)\ x \circ \textit{ind})\ \textit{standard-basis}$
unfolding *comp-def*
by *simp*
also have ... = *sum* (($) *x*) (*ind* ‘ *standard-basis*)
using *sum-comp*[*of ind standard-basis ind* ‘ *standard-basis* ($) *x*] *bij*
by *simp*
also have ... = *sum* (($) *x*) *UNIV*
using *univ*
by *simp*
finally have *sum f standard-basis* = *sum* (($) *x*) *UNIV*
using *univ*
by *simp*
hence *sum-1*: *sum f standard-basis* = *1*
using *convex-comb*
by *simp*

have *nonneg*: $\forall\ v \in$ *standard-basis*. $f\ v \geq 0$
using *def convex-comb*
by *simp*
have $\forall\ v \in$ *standard-basis*. $\forall\ i$. $v\$i$ = (*if* i = *ind* v *then* 1 *else* 0)
using *ind-1 ind-0*
by *fastforce*
hence $\forall\ v \in$ *standard-basis*. $\forall\ i$. $x\$(ind\ v) * v\$i$ = (*if* i = *ind* v *then* $x\$(ind\ v)$ *else* 0)
by *auto*
hence $\forall\ v \in$ *standard-basis*. (χ i. $x\$(ind\ v) * v\$i$)
= (χ i. *if* i = *ind* v *then* $x\$(ind\ v)$ *else* 0)
by *fastforce*
moreover have $\forall\ v$. $(x\$(ind\ v)) *_R v$ = (χ i. $x\$(ind\ v) * v\$i$)
unfolding *scaleR-vec-def*
by *simp*
ultimately have
$\forall\ v \in$ *standard-basis*. $(x\$(ind\ v)) *_R v$ = (χ i. *if* i = *ind* v *then* $x\$(ind\ v)$ *else* 0)
by *simp*
moreover have *sum* (λ x. $(f\ x) *_R x$) *standard-basis*
= *sum* (λ v. $(x\$(ind\ v)) *_R v$) *standard-basis*
unfolding *def*
by *simp*
ultimately have *sum* (λ x. $(f\ x) *_R x$) *standard-basis*
= *sum* (λ v. (χ i. *if* i = *ind* v *then* $x\$(ind\ v)$ *else* 0)) *standard-basis*
by *force*
also have ... = *sum* (λ b. (χ i. *if* i = *ind* (*ind-inv* b) *then* $x\$(ind\ (ind\text{-}inv\ b))$ *else* 0)) *UNIV*
using *bij-inv sum-comp*
unfolding *comp-def*
by *blast*
also have ... = *sum* (λ b. (χ i. *if* i = b *then* $x\$b$ *else* 0)) *UNIV*
using *ind-inv-map*
by *presburger*
finally have *sum* (λ x. $(f\ x) *_R x$) *standard-basis*
= *sum* (λ b. (χ i. *if* i = b *then* $x\$b$ *else* 0)) *UNIV*
by *simp*
moreover have $\forall\ b$. (*sum* (λ b. (χ i. *if* i = b *then* $x\$b$ *else* 0)) *UNIV*)$\$b$
= *sum* (λ b'. (χ i. *if* i = b' *then* $x\$b'$ *else* 0)$\$b$) *UNIV*
using *sum-component*
by *blast*
moreover have $\forall\ b$. (λ b'. (χ i. *if* i = b' *then* $x\$b'$ *else* 0)$\$b$)
= (λ b'. *if* b' = b *then* $x\$b$ *else* 0)
by *force*
moreover have $\forall\ b$. *sum* (λ b'. *if* b' = b *then* $x\$b$ *else* 0) *UNIV*
= $x\$b$ + *sum* (λ b'. 0) (*UNIV* $-$ $\{b\}$)
by *simp*
ultimately have $\forall\ b$. (*sum* (λ x. $(f\ x) *_R x$) *standard-basis*)$\$b$ = $x\$b$
by *simp*

hence *sum* (λ *x*. (*f x*) $*_R$ *x*) *standard-basis* = *x*
unfolding *vec-eq-iff*
by *simp*
hence $\exists$ *f*::(*real*^′*b*) $\Rightarrow$ *real*.
sum f standard-basis = *1* $\wedge$
($\forall$ *x* $\in$ *standard-basis*. *f x* $\geq$ *0*) $\wedge$
x = *sum* (λ *x*. (*f x*) $*_R$ *x*) *standard-basis*
using *sum-1 nonneg*
by *blast*
hence *x* $\in$ *convex hull* (*standard-basis*::((*real*^′*b*) *set*))
using *convex-combination-in-convex-hull*
by *blast*
thus *x* $\in \bigcap$ {*t*. ($\forall$ *x* $\in$ *t*. $\forall$ *y* $\in$ *t*. $\forall$ *u* $\geq$ *0*. $\forall$ *v* $\geq$ *0*. *u* + *v* = *1* $\longrightarrow$ *u* $*_R$ *x*
+ *v* $*_R$ *y* $\in$ *t*) $\wedge$
(*standard-basis*::((*real*^′*b*) *set*)) $\subseteq$ *t*}
unfolding *convex-def hull-def*
by *blast*
qed
qed

lemma *fract-distr-helper*:
fixes
a :: *int* **and**
b :: *int* **and**
c :: *int*
assumes *c* $\neq$ *0*
shows *Fract a c* + *Fract b c* = *Fract* (*a* + *b*) *c*
using *add-rat assms mult.commute mult-rat-cancel distrib-right*
by *metis*

lemma *anon-hom-equiv-rel*:
fixes *X* :: (′*a*, ′*v*) *Election set*
assumes $\forall$ *E* $\in$ *X*. *finite* (*voters-*$\mathcal{E}$ *E*)
shows *equiv X* (*anon-hom*$_\mathcal{R}$ *X*)
proof (*unfold equiv-def*, *safe*)
show *refl-on X* (*anon-hom*$_\mathcal{R}$ *X*)
unfolding *refl-on-def anon-hom*$_\mathcal{R}$*.simps*
by *blast*
next
show *sym* (*anon-hom*$_\mathcal{R}$ *X*)
unfolding *sym-def anon-hom*$_\mathcal{R}$*.simps*
using *sup-commute*
by *simp*
next
show *Relation.trans* (*anon-hom*$_\mathcal{R}$ *X*)
proof
fix
E :: (′*a*, ′*v*) *Election* **and**
E′ :: (′*a*, ′*v*) *Election* **and**

    F :: $('a, 'v)$ *Election*
  assume
    rel: $(E, E') \in$ *anon-hom*$_\mathcal{R}$ X **and**
    rel': $(E', F) \in$ *anon-hom*$_\mathcal{R}$ X
  hence *fin*: *finite* (*voters-*$\mathcal{E}$ E')
    unfolding *anon-hom*$_\mathcal{R}$*.simps*
    using *assms*
    by *fastforce*
  from *rel rel'* **have** *eq-frac*:
    $(\forall\ r.$ *vote-fraction* $r\ E =$ *vote-fraction* $r\ E') \wedge$
      $(\forall\ r.$ *vote-fraction* $r\ E' =$ *vote-fraction* $r\ F)$
    unfolding *anon-hom*$_\mathcal{R}$*.simps*
    by *blast*
  hence $\forall\ r.$ *vote-fraction* $r\ E =$ *vote-fraction* $r\ F$
    by *metis*
  thus $(E, F) \in$ *anon-hom*$_\mathcal{R}$ X
    using *rel rel' snd-conv*
    unfolding *anon-hom*$_\mathcal{R}$*.simps*
    by *blast*
  qed
qed

lemma *fract-distr*:
  fixes
    A :: *'x set* **and**
    f :: *'x* $\Rightarrow$ *int* **and**
    b :: *int*
  assumes
    finite A **and**
    $b \neq 0$
  shows *sum* (λ *a. Fract* (*f a*) *b*) $A =$ *Fract* (*sum f A*) *b*
  using *assms*
proof (*induction card A arbitrary*: *A f b*)
  case *0*
  fix
    A :: *'x set* **and**
    f :: *'x* $\Rightarrow$ *int* **and**
    b :: *int*
  assume
    $0 =$ *card* A **and**
    finite A **and**
    $b \neq 0$
  hence *sum* (λ *a. Fract* (*f a*) *b*) $A = 0 \wedge$ *sum f* $A = 0$
    by *simp*
  thus *?case*
    using *0 rat-number-collapse*
    by *simp*
next
  case (*Suc n*)

fix
  A :: $'x$ *set* **and**
  f :: $'x \Rightarrow$ *int* **and**
  b :: *int* **and**
  n :: *nat*
assume
  card-A: *Suc* n = *card* A **and**
  fin-A: *finite* A **and**
  b-non-zero: $b \neq 0$ **and**
  hyp: $\bigwedge$ A f b.
    n = *card* (A::$'x$ *set*) $\Longrightarrow$
    finite $A \Longrightarrow b \neq 0 \Longrightarrow (\sum\ a \in A.\ Fract\ (f\ a)\ b) = Fract\ (sum\ f\ A)\ b$
hence $A \neq \{\}$
  by *auto*
then obtain c :: $'x$ **where**
  c-in-A: $c \in A$
  by *blast*
hence $(\sum\ a \in A.\ Fract\ (f\ a)\ b) = (\sum\ a \in A - \{c\}.\ Fract\ (f\ a)\ b) + Fract\ (f$
$c)\ b$
  using *fin-A*
  by (*simp add*: *sum-diff1*)
also have ... $= Fract\ (sum\ f\ (A - \{c\}))\ b + Fract\ (f\ c)\ b$
  using *hyp card-A fin-A b-non-zero c-in-A Diff-empty card-Diff-singleton diff-Suc-1 finite-Diff-insert*
  by *metis*
also have ... $= Fract\ (sum\ f\ (A - \{c\}) + f\ c)\ b$
  using *c-in-A b-non-zero fract-distr-helper*
  by *metis*
also have ... $= Fract\ (sum\ f\ A)\ b$
  using *c-in-A fin-A*
  by (*simp add*: *sum-diff1*)
finally show $(\sum\ a \in A.\ Fract\ (f\ a)\ b) = Fract\ (sum\ f\ A)\ b$
  by *blast*
qed

Simplex Bijection

We assume all our elections to consist of a fixed finite alternative set of size n and finite subsets of an infinite voter universe. Profiles are linear orders on the alternatives. Then we can work on the standard simplex of dimension n! instead of the equivalence classes of the equivalence relation for anonymous + homogeneous voting rules (anon hom): Each dimension corresponds to one possible linear order on the alternative set, i.e., the possible preferences. Each equivalence class of elections corresponds to a vector whose entries denote the fraction of voters per election in that class who vote the respective corresponding preference.

theorem *anon-hom*$_{\mathcal{Q}}$*-iso*:
  assumes *infinite* (*UNIV*::($'v$ *set*))

shows
bij-betw (*anon-hom-class-to-vec*::(′*a*::*finite*, ′*v*) *Election set* ⇒ *rat*^(′*a Ordered-Preference*))
(*anon-hom*$_\mathcal{Q}$ (*UNIV*::′*a set*)) (*vote-simplex* :: (*rat*^(′*a Ordered-Preference*))
set)
proof (*unfold bij-betw-def inj-on-def*, *standard*, *standard*, *standard*, *standard*)
fix
X :: (′*a*, ′*v*) *Election set* **and**
Y :: (′*a*, ′*v*) *Election set*
assume
class-X: *X* ∈ *anon-hom*$_\mathcal{Q}$ *UNIV* **and**
class-Y: *Y* ∈ *anon-hom*$_\mathcal{Q}$ *UNIV* **and**
eq-vec: *anon-hom-class-to-vec X* = *anon-hom-class-to-vec Y*
have *equiv*: *equiv* (*fixed-alt-elections UNIV*) (*anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*))
using *anon-hom-equiv-rel CollectD IntD1 inf-commute*
unfolding *fixed-alt-elections.simps*
by (*metis* (*no-types*, *lifting*))
hence *subset*: *X* ≠ {} ∧ *X* ⊆ *fixed-alt-elections UNIV* ∧ *Y* ≠ {} ∧ *Y* ⊆
fixed-alt-elections UNIV
using *class-X class-Y in-quotient-imp-non-empty in-quotient-imp-subset*
unfolding *anon-hom*$_\mathcal{Q}$*.simps*
by *blast*
then obtain *E* :: (′*a*, ′*v*) *Election* **and**
E′ :: (′*a*, ′*v*) *Election* **where**
E-in-X: *E* ∈ *X* **and**
E′*-in-Y*: *E*′ ∈ *Y*
by *blast*
hence *class-X-E*: *anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) ‘‘ {*E*} = *X*
using *class-X equiv Image-singleton-iff equiv-class-eq quotientE*
unfolding *anon-hom*$_\mathcal{Q}$*.simps*
by (*metis* (*no-types*, *opaque-lifting*))
hence ∀ *F* ∈ *X*. (*E*, *F*) ∈ *anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*)
unfolding *Image-def*
by *blast*
hence ∀ *F* ∈ *X*. ∀ *p*. *vote-fraction p F* = *vote-fraction p E*
unfolding *anon-hom*$_\mathcal{R}$*.simps*
by *fastforce*
hence ∀ *p*. *vote-fraction p* ‘ *X* = {*vote-fraction p E*}
using *E-in-X*
by *blast*
hence ∀ *p*. *vote-fraction*$_\mathcal{Q}$ *p X* = *vote-fraction p E*
using *is-singletonI singleton-set-def-if-card-one the-elem-eq*
unfolding *is-singleton-altdef vote-fraction*$_\mathcal{Q}$*.simps* $\pi_\mathcal{Q}$*.simps singleton-set.simps*
by *metis*
hence *eq-X-E*: ∀ *p*. (*anon-hom-class-to-vec X*)$*p* = *vote-fraction* (*ord2pref p*) *E*
unfolding *anon-hom-class-to-vec.simps*
using *vec-lambda-beta*
by *metis*
have *class-Y-E*′: *anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) ‘‘ {*E*′} = *Y*
using *class-Y equiv E*′*-in-Y Image-singleton-iff equiv-class-eq quotientE*

unfolding $\textit{anon-hom}_{\mathcal{Q}}\textit{.simps}$
by (*metis* (*no-types*, *opaque-lifting*))
hence $\forall\ F \in Y.\ (E', F) \in \textit{anon-hom}_{\mathcal{R}}\ (\textit{fixed-alt-elections UNIV})$
unfolding *Image-def*
by *blast*
hence $\forall\ F \in Y.\ \forall\ p.\ \textit{vote-fraction}\ p\ E' = \textit{vote-fraction}\ p\ F$
unfolding $\textit{anon-hom}_{\mathcal{R}}\textit{.simps}$
by *blast*
hence $\forall\ p.\ \textit{vote-fraction}\ p\ `\ Y = \{\textit{vote-fraction}\ p\ E'\}$
using E'*-in-*Y
by *fastforce*
hence $\forall\ p.\ \textit{vote-fraction}_{\mathcal{Q}}\ p\ Y = \textit{vote-fraction}\ p\ E'$
using *is-singletonI singleton-set-def-if-card-one the-elem-eq*
unfolding *is-singleton-altdef* $\textit{vote-fraction}_{\mathcal{Q}}\textit{.simps}$ $\pi_{\mathcal{Q}}\textit{.simps}$ *singleton-set.simps*
by *metis*
hence *eq-Y-E'*: $\forall\ p.\ (\textit{anon-hom-class-to-vec}\ Y)\$p = \textit{vote-fraction}\ (\textit{ord2pref}\ p)$
E'
unfolding *anon-hom-class-to-vec.simps*
using *vec-lambda-beta*
by *metis*
with *eq-X-E eq-vec*
have $\forall\ p.\ \textit{vote-fraction}\ (\textit{ord2pref}\ p)\ E = \textit{vote-fraction}\ (\textit{ord2pref}\ p)\ E'$
by *metis*
hence *eq-ord*: $\forall\ p.\ \textit{linear-order}\ p \longrightarrow \textit{vote-fraction}\ p\ E = \textit{vote-fraction}\ p\ E'$
using *mem-Collect-eq pref2ord-inverse*
by *metis*
have $(\forall\ v.\ v \in \textit{voters-}\mathcal{E}\ E \longrightarrow \textit{linear-order}\ (\textit{profile-}\mathcal{E}\ E\ v)) \wedge$
$(\forall\ v.\ v \in \textit{voters-}\mathcal{E}\ E' \longrightarrow \textit{linear-order}\ (\textit{profile-}\mathcal{E}\ E'\ v))$
using *subset E-in-X* E'*-in-*Y
unfolding *fixed-alt-elections.simps valid-elections-def profile-def*
by *fastforce*
hence $\forall\ p.\ \neg\ \textit{linear-order}\ p \longrightarrow \textit{vote-count}\ p\ E = 0 \wedge \textit{vote-count}\ p\ E' = 0$
unfolding *vote-count.simps*
using *card.infinite card-0-eq Collect-empty-eq*
by (*metis* (*mono-tags*, *lifting*))
hence $\forall\ p.\ \neg\ \textit{linear-order}\ p \longrightarrow \textit{vote-fraction}\ p\ E = 0 \wedge \textit{vote-fraction}\ p\ E' = 0$
using *int-ops rat-number-collapse*
by *simp*
with *eq-ord* **have** $\forall\ p.\ \textit{vote-fraction}\ p\ E = \textit{vote-fraction}\ p\ E'$
by *metis*
hence $(E, E') \in \textit{anon-hom}_{\mathcal{R}}\ (\textit{fixed-alt-elections UNIV})$
using *subset E-in-X* E'*-in-*Y *fixed-alt-elections.simps*
unfolding $\textit{anon-hom}_{\mathcal{R}}\textit{.simps}$
by *blast*
thus $X = Y$
using *class-X-E class-Y-E' equiv equiv-class-eq*
by (*metis* (*no-types*, *lifting*))
next
show $(\textit{anon-hom-class-to-vec}::(\prime a, \prime v)\ \textit{Election set} \Rightarrow \textit{rat}\hat{}(\prime a\ \textit{Ordered-Preference}))$

```
      ‘ anon-hom𝒬 UNIV = vote-simplex
  proof (unfold vote-simplex-def, safe)
    fix X :: ('a, 'v) Election set
    assume
      quot: X ∈ anon-hom𝒬 UNIV and
     not-simplex: anon-hom-class-to-vec X ∉ rat-vec-set (convex hull standard-basis)
    have equiv-rel:
      equiv (fixed-alt-elections UNIV) (anon-homℛ (fixed-alt-elections UNIV))
     using anon-hom-equiv-rel[of fixed-alt-elections UNIV] fixed-alt-elections.simps
      by blast
    then obtain E :: ('a, 'v) Election where
      E-in-X: E ∈ X and
      X = anon-homℛ (fixed-alt-elections UNIV) ‘‘ {E}
      using quot anon-hom𝒬.simps equiv-Eps-in proj-Eps
      unfolding proj-def
      by metis
    hence rel: ∀ E' ∈ X. (E, E') ∈ anon-homℛ (fixed-alt-elections UNIV)
      by simp
     hence ∀ p. ∀ E' ∈ X. vote-fraction (ord2pref p) E' = vote-fraction (ord2pref
p) E
      unfolding anon-homℛ.simps
      by fastforce
    hence ∀ p. vote-fraction (ord2pref p) ‘ X = {vote-fraction (ord2pref p) E}
      using E-in-X
      by blast
    hence repr: ∀ p. vote-fraction𝒬 (ord2pref p) X = vote-fraction (ord2pref p) E
      using is-singletonI singleton-set-def-if-card-one the-elem-eq
      unfolding vote-fraction𝒬.simps π𝒬.simps is-singleton-altdef
      by metis
    have ∀ p. vote-count (ord2pref p) E ≥ 0
      by simp
    hence ∀ p. card (voters-ℰ E) > 0 ⟶
        Fract (int (vote-count (ord2pref p) E)) (int (card (voters-ℰ E))) ≥ 0
      using zero-le-Fract-iff
      by simp
    hence ∀ p. vote-fraction (ord2pref p) E ≥ 0
      unfolding vote-fraction.simps card-gt-0-iff
      by simp
    hence ∀ p. vote-fraction𝒬 (ord2pref p) X ≥ 0
      using repr
      by simp
    hence geq-0: ∀ p. real-of-rat (vote-fraction𝒬 (ord2pref p) X) ≥ 0
      using zero-le-of-rat-iff
      by blast
    have voters-ℰ E = {} ∨ infinite (voters-ℰ E) ⟶
       (∀ p. real-of-rat (vote-fraction p E) = 0)
      by simp
    hence zero-case:
      voters-ℰ E = {} ∨ infinite (voters-ℰ E) ⟶
```

$(\chi$ *p. real-of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref p*$)$ *X*$)) = $ *0*
using *repr*
unfolding *zero-vec-def*
by *simp*
let *?sum* = *sum* (λ *p. vote-count p E*) *UNIV*
have *finite* (*UNIV*::('*a* × '*a*) *set*)
by *simp*
hence *eq-card*: *finite* (*voters-ℰ E*) ⟶ *card* (*voters-ℰ E*) = *?sum*
using *vote-count-sum*
by *metis*
hence *finite* (*voters-ℰ E*) ∧ *voters-ℰ E* ≠ {} ⟶
sum (λ *p. vote-fraction p E*) *UNIV* =
sum (λ *p. Fract* (*vote-count p E*) *?sum*) *UNIV*
unfolding *vote-fraction.simps*
by *presburger*
moreover have *gt-0*: *finite* (*voters-ℰ E*) ∧ *voters-ℰ E* ≠ {} ⟶ *?sum* > *0*
using *eq-card*
by *fastforce*
hence *finite* (*voters-ℰ E*) ∧ *voters-ℰ E* ≠ {} ⟶
sum (λ *p. Fract* (*vote-count p E*) *?sum*) *UNIV* = *Fract ?sum ?sum*
using *fract-distr*[*of UNIV ?sum* λ *p. int* (*vote-count p E*)]
card-0-eq eq-card finite-class.finite-UNIV
of-nat-eq-0-iff of-nat-sum sum.cong
by (*metis* (*no-types*, *lifting*))
moreover have *finite* (*voters-ℰ E*) ∧ *voters-ℰ E* ≠ {} ⟶ *Fract ?sum ?sum*
= *1*
using *gt-0 One-rat-def eq-rat*(*1*)[*of ?sum 1 ?sum 1*]
by *linarith*
ultimately have *sum-1*:
finite (*voters-ℰ E*) ∧ *voters-ℰ E* ≠ {} ⟶ *sum* (λ *p. vote-fraction p E*) *UNIV*
= *1*
by *presburger*
have *inv-of-rat*: ∀ *x* ∈ ℚ. *the-inv of-rat* (*of-rat x*) = *x*
unfolding *Rats-def*
using *the-inv-f-f injI of-rat-eq-iff*
by *metis*
have *E* ∈ *fixed-alt-elections UNIV*
using *quot E-in-X equiv-class-eq-iff equiv-rel rel*
unfolding *anon-hom*$_{\mathcal{Q}}$*.simps quotient-def*
by *fastforce*
hence ∀ *v* ∈ *voters-ℰ E. linear-order* (*profile-ℰ E v*)
unfolding *fixed-alt-elections.simps valid-elections-def profile-def*
by *fastforce*
hence ∀ *p*. ¬ *linear-order p* ⟶ *vote-count p E* = *0*
unfolding *vote-count.simps*
using *card.infinite card-0-eq*
by *blast*
hence ∀ *p*. ¬ *linear-order p* ⟶ *vote-fraction p E* = *0*
using *rat-number-collapse*

by *simp*
moreover have *sum* (*λ p. vote-fraction p E*) *UNIV* =
sum (*λ p. vote-fraction p E*) {*p. linear-order p*} +
sum (*λ p. vote-fraction p E*) (*UNIV* − {*p. linear-order p*})
using *finite CollectD Collect-mono UNIV-I add.commute sum.subset-diff top-set-def*
by *metis*
ultimately have *sum* (*λ p. vote-fraction p E*) *UNIV* =
sum (*λ p. vote-fraction p E*) {*p. linear-order p*}
by *simp*
moreover have *bij-betw ord2pref UNIV* {*p. linear-order p*}
using *inj-def ord2pref-inject range-ord2pref*
unfolding *bij-betw-def*
by *blast*
ultimately have
sum (*λ p. vote-fraction p E*) *UNIV* = *sum* (*λ p. vote-fraction* (*ord2pref p*) *E*) *UNIV*
using *comp-def*[*of λ p. vote-fraction p E ord2pref*]
sum-comp[*of ord2pref UNIV* {*p. linear-order p*} *λ p. vote-fraction p E*]
by *auto*
hence *finite* (*voters-$\mathcal{E}$ E*) ∧ *voters-$\mathcal{E}$ E* ≠ {} ⟶
sum (*λ p. vote-fraction* (*ord2pref p*) *E*) *UNIV* = *1*
using *sum-1*
by *presburger*
hence *finite* (*voters-$\mathcal{E}$ E*) ∧ *voters-$\mathcal{E}$ E* ≠ {} ⟶
sum (*λ p. real-of-rat* (*vote-fraction* (*ord2pref p*) *E*)) *UNIV* = *1*
using *of-rat-1 of-rat-sum*
by *metis*
with *zero-case*
have (*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)) = *0* ∨
sum (*λ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)) *UNIV* = *1*
using *repr*
by *force*
hence (*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)) = *0* ∨
((∀ *p.* (*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*))\$*p* ≥ *0*) ∧
sum ((\$) (*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*))) *UNIV* = *1*)
using *geq-0*
by *force*
moreover have *rat-entries*: ∀ *p.* (*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*))\$*p* ∈ ℚ
by *simp*
ultimately have *simplex-el*:
(*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)) ∈
{*x* ∈ *insert 0* (*convex hull standard-basis*). ∀ *i. x*\$*i* ∈ ℚ}
using *standard-simplex-rewrite*
by *blast*
moreover have
∀ *p.* (*rat-vec* (*χ p. of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)))\$*p*
= *the-inv real-of-rat* ((*χ p. real-of-rat* (*vote-fraction$_{\mathcal{Q}}$* (*ord2pref p*) *X*)) \$ *p*)

unfolding *rat-vec.simps*
using *vec-lambda-beta*
by *blast*
moreover have
$\forall\ p.$ *the-inv real-of-rat* $((\chi\ p.$ *real-of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X))\ \$\ p)$
$=$
the-inv real-of-rat $($*real-of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X))$
by *simp*
moreover have
$\forall\ p.$ *the-inv real-of-rat* $($*real-of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X)) =$
vote-fraction$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X$
using *rat-entries inv-of-rat Rats-eq-range-nat-to-rat-surj surj-nat-to-rat-surj*
by *blast*
moreover have $\forall\ p.$ *vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X = ($*anon-hom-class-to-vec*
$X)\$p$
by *simp*
ultimately have
$\forall\ p.$ $($*rat-vec* $(\chi\ p.$ *of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X)))\$p =$
$($*anon-hom-class-to-vec* $X)\$p$
by *metis*
hence *rat-vec* $(\chi\ p.$ *of-rat* $($*vote-fraction*$_{\mathcal{Q}}$ $($*ord2pref* $p)\ X)) =$ *anon-hom-class-to-vec*
X
by *simp*
with *simplex-el*
have $\exists\ x \in \{x \in$ *insert* 0 $($*convex hull standard-basis*$).\ \forall\ i.\ x\ \$\ i \in \mathbb{Q}\}.$
rat-vec $x =$ *anon-hom-class-to-vec* X
by *blast*
with *not-simplex*
have *rat-vec* $0 =$ *anon-hom-class-to-vec* X
using *image-iff insertE mem-Collect-eq*
unfolding *rat-vec-set.simps*
by (*metis* (*mono-tags*, *lifting*))
thus *anon-hom-class-to-vec* $X = 0$
unfolding *rat-vec.simps*
using *Rats-0 inv-of-rat of-rat-0 vec-lambda-unique zero-index*
by (*metis* (*no-types*, *lifting*))
next
have *non-empty*:
$(UNIV, \{\}, \lambda\ v.\ \{\}) \in ($*anon-hom*$_{\mathcal{R}}$ $($*fixed-alt-elections* $UNIV)$ ‘‘ $\{(UNIV, \{\},$
$\lambda\ v.\ \{\})\})$
unfolding *anon-hom*$_{\mathcal{R}}$*.simps Image-def fixed-alt-elections.simps*
valid-elections-def profile-def
by *simp*
have *in-els*: $(UNIV, \{\}, \lambda\ v.\ \{\}) \in$ *fixed-alt-elections* $UNIV$
unfolding *fixed-alt-elections.simps valid-elections-def profile-def*
by *simp*
have $\forall\ r$::('*a Preference-Relation*). *vote-fraction* r $(UNIV, \{\}, (\lambda\ v.\ \{\})) = 0$
by *simp*
hence

$\forall\ E \in (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV))\ \text{``}\ \{(UNIV, \{\}, (\lambda\ v.\ \{\}))\}.$
$\forall\ r.\ vote\text{-}fraction\ r\ E = 0$
unfolding *anon-hom*$_{\mathcal{R}}$*.simps*
by *auto*
moreover have
$\forall\ E \in (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV))\ \text{``}\ \{(UNIV, \{\}, (\lambda\ v.\ \{\}))\}.$
$finite\ (voters\text{-}\mathcal{E}\ E)$
unfolding *Image-def anon-hom*$_{\mathcal{R}}$*.simps*
by *fastforce*
ultimately have *all-zero*:
$\forall\ r.\ \forall\ E \in (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV))\ \text{``}\ \{(UNIV, \{\}, (\lambda\ v.$
$\{\}))\}.$
$vote\text{-}fraction\ r\ E = 0$
by *blast*
hence $\forall\ r.\ 0 \in$
$vote\text{-}fraction\ r\ \text{`}\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV))\ \text{``}\ \{(UNIV, \{\}, (\lambda$
$v.\ \{\}))\}$
using *non-empty image-eqI*
by (*metis* (*mono-tags*, *lifting*))
hence $\forall\ r.\ \{0\} \subseteq vote\text{-}fraction\ r\ \text{`}$
$(anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\}, \lambda\ v.\ \{\})\})$
by *blast*
moreover have $\forall\ r.\ \{0\} \supseteq vote\text{-}fraction\ r\ \text{`}$
$(anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\}, \lambda\ v.\ \{\})\})$
using *all-zero*
by *blast*
ultimately have $\forall\ r.$
$vote\text{-}fraction\ r\ \text{`}\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\}, \lambda\ v.$
$\{\})\}) = \{0\}$
by *blast*
hence
$\forall\ r.$
$card\ (vote\text{-}fraction\ r\ \text{`}\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\},$
$\lambda\ v.\ \{\})\})) = 1$
$\wedge\ the\text{-}inv\ (\lambda\ x.\ \{x\})$
$(vote\text{-}fraction\ r\ \text{`}\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\}, \lambda$
$v.\ \{\})\})) = 0$
using *is-singletonI singleton-insert-inj-eq' singleton-set-def-if-card-one*
unfolding *is-singleton-altdef singleton-set.simps*
by *metis*
hence
$\forall\ r.\ vote\text{-}fraction_{\mathcal{Q}}\ r\ (anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\},$
$\lambda\ v.\ \{\})\}) = 0$
unfolding *vote-fraction*$_{\mathcal{Q}}$*.simps* $\pi_{\mathcal{Q}}$*.simps singleton-set.simps*
by *metis*
hence $\forall\ r::('a\ Ordered\text{-}Preference).\ vote\text{-}fraction_{\mathcal{Q}}\ (ord2pref\ r)$
$(anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{``}\ \{(UNIV, \{\}, \lambda\ v.\ \{\})\}) = 0$
by *metis*
hence $\forall\ r::('a\ Ordered\text{-}Preference).$

$(anon\text{-}hom\text{-}class\text{-}to\text{-}vec\ ((anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)$
$`` \{(UNIV, \{\}, \lambda\ v.\ \{\})\})))\$r = 0$
unfolding *anon-hom-class-to-vec.simps*
using *vec-lambda-beta*
by (*metis* (*no-types*))
moreover have $\forall\ r::('a\ Ordered\text{-}Preference).\ 0\$r = 0$
by *simp*
ultimately have $\forall\ r::('a\ Ordered\text{-}Preference).$
$(anon\text{-}hom\text{-}class\text{-}to\text{-}vec$
$((anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ `` \{(UNIV, \{\}, \lambda\ v.\ \{\})\})))\$r$
$= (0::(rat\widehat{\ }('a\ Ordered\text{-}Preference)))\$r$
by (*metis* (*no-types*))
hence *anon-hom-class-to-vec*
$((anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ `` \{(UNIV, \{\}, \lambda\ v.\ \{\})\}))$
$= (0::(rat\widehat{\ }('a\ Ordered\text{-}Preference)))$
using *vec-eq-iff*
by *blast*
moreover have
$(anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ `` \{(UNIV, \{\}, \lambda\ v.\ \{\})\}) \in anon\text{-}hom_{\mathcal{Q}}$
$UNIV$
unfolding *anon-hom*$_{\mathcal{Q}}$*.simps quotient-def*
using *in-els*
by *blast*
ultimately show $(0::(rat\widehat{\ }('a\ Ordered\text{-}Preference))) \in anon\text{-}hom\text{-}class\text{-}to\text{-}vec$ `
$anon\text{-}hom_{\mathcal{Q}}\ UNIV$
using *image-eqI*
by (*metis* (*no-types*))
next
fix $x :: rat\widehat{\ }('a\ Ordered\text{-}Preference)$
assume $x \in rat\text{-}vec\text{-}set\ (convex\ hull\ standard\text{-}basis)$
— Convert rat vector x to real vector x'.
then obtain $x' :: real\widehat{\ }('a\ Ordered\text{-}Preference)$ **where**
conv: $x' \in convex\ hull\ standard\text{-}basis$ **and**
inv: $\forall\ p.\ x\$p = the\text{-}inv\ real\text{-}of\text{-}rat\ (x'\$p)$ **and**
rat: $\forall\ p.\ x'\$p \in \mathbb{Q}$
unfolding *rat-vec-set.simps rat-vec.simps*
by *force*
hence *convex*: $(\forall\ p.\ 0 \le x'\$p) \wedge sum\ ((\$)\ x')\ UNIV = 1$
using *standard-simplex-rewrite*
by *blast*
have *map*: $\forall\ p.\ real\text{-}of\text{-}rat\ (x\$p) = x'\$p$
using *inv rat the-inv-f-f*[*of real-of-rat*] *f-the-inv-into-f inj-onCI of-rat-eq-iff*
unfolding *Rats-def*
by *metis*
have $\forall\ p.\ \exists\ fract.\ Fract\ (fst\ fract)\ (snd\ fract) = x\$p \wedge 0 < snd\ fract$
using *quotient-of-unique*
by *metis*
then obtain $fraction' :: 'a\ Ordered\text{-}Preference \Rightarrow (int \times int)$ **where**
$\forall\ p.\ x\$p = Fract\ (fst\ (fraction'\ p))\ (snd\ (fraction'\ p))$ **and**

pos': $\forall$ *p*. *0* $<$ *snd* (*fraction'* *p*)
by *metis*
with *map*
have *fract'*: $\forall$ *p*. *x'*\$*p* = (*fst* (*fraction'* *p*)) / (*snd* (*fraction'* *p*))
using *div-by-0 divide-less-cancel of-int-0 of-int-pos of-rat-rat*
by *metis*
with *convex*
have $\forall$ *p*. (*fst* (*fraction'* *p*)) / (*snd* (*fraction'* *p*)) $\geq$ *0*
by *fastforce*
with *pos'*
have $\forall$ *p*. *fst* (*fraction'* *p*) $\geq$ *0*
using *not-less of-int-0-le-iff of-int-pos zero-le-divide-iff*
by *metis*
with *pos'*
have $\forall$ *p*. *fst* (*fraction'* *p*) $\in \mathbb{N} \land$ *snd* (*fraction'* *p*) $\in \mathbb{N}$
using *nonneg-int-cases of-nat-in-Nats order-less-le*
by *metis*
hence $\forall$ *p*. $\exists$ (*n*::*nat*) (*m*::*nat*). *fst* (*fraction'* *p*) = *n* $\land$ *snd* (*fraction'* *p*) = *m*
using *Nats-cases*
by *metis*
hence $\forall$ *p*. $\exists$ *m*::*nat* $\times$ *nat*. *fst* (*fraction'* *p*) = *int* (*fst* *m*) $\land$ *snd* (*fraction'* *p*) = *int* (*snd* *m*)
by *simp*
then obtain *fraction* :: *'a Ordered-Preference* $\Rightarrow$ (*nat* $\times$ *nat*) **where**
eq: $\forall$ *p*. *fst* (*fraction'* *p*) = *int* (*fst* (*fraction* *p*)) $\land$
snd (*fraction'* *p*) = *int* (*snd* (*fraction* *p*))
by *metis*
with *fract'*
have *fract*: $\forall$ *p*. *x'*\$*p* = (*fst* (*fraction* *p*)) / (*snd* (*fraction* *p*))
by *simp*
from *eq pos'*
have *pos*: $\forall$ *p*. *0* $<$ *snd* (*fraction* *p*)
by *simp*
let *?prod* = *prod* (λ *p*. *snd* (*fraction* *p*)) *UNIV*
have *fin*: *finite* (*UNIV*::(*'a Ordered-Preference set*))
by *simp*
hence *finite* {*snd* (*fraction* *p*) | *p*. *p* $\in$ *UNIV*}
using *finite-Atleast-Atmost-nat*
by *simp*
have *pos-prod*: *?prod* $>$ *0*
using *pos*
by *simp*
hence $\forall$ *p*. *?prod mod* (*snd* (*fraction* *p*)) = *0*
using *pos finite UNIV-I bits-mod-0 mod-prod-eq mod-self prod-zero*
by (*metis* (*mono-tags*, *lifting*))
hence *div*: $\forall$ *p*. (*?prod div* (*snd* (*fraction* *p*))) $*$ (*snd* (*fraction* *p*)) = *?prod*
using *add.commute add-0 div-mult-mod-eq*
by *metis*
obtain *voter-amount* :: *'a Ordered-Preference* $\Rightarrow$ *nat* **where**

def: *voter-amount* = (λ *p*. (*fst* (*fraction p*)) ∗ (*?prod div* (*snd* (*fraction p*))))
  by *blast*
have *rewrite-div*: ∀ *p*. *?prod div* (*snd* (*fraction p*)) = *?prod* / (*snd* (*fraction p*))
  using *div less-imp-of-nat-less nonzero-mult-div-cancel-right*
    of-nat-less-0-iff of-nat-mult pos
  by *metis*
hence *sum voter-amount UNIV* =
    sum (λ *p*. (*fst* (*fraction p*)) ∗ (*?prod* / (*snd* (*fraction p*)))) *UNIV*
  using *def*
  by *simp*
hence *sum voter-amount UNIV* =
    ?prod ∗ (*sum* (λ *p*. (*fst* (*fraction p*)) / (*snd* (*fraction p*))) *UNIV*)
  using *mult-of-nat-commute sum.cong times-divide-eq-right*
    vector-space-over-itself.scale-sum-right
  by (*metis* (*mono-tags*, *lifting*))
hence *rewrite-sum*: *sum voter-amount UNIV* = *?prod*
  using *fract convex mult-cancel-left1 of-nat-eq-iff sum.cong*
  by (*metis* (*mono-tags*, *lifting*))
obtain *V* :: *'v set* **where**
  fin-V: *finite V* **and**
  card-V-eq-sum: *card V* = *sum voter-amount UNIV*
  using *assms infinite-arbitrarily-large*
  by *metis*
then obtain *part* :: *'a Ordered-Preference* ⇒ *'v set* **where**
  partition: *V* = ⋃ {*part p* | *p*. *p* ∈ *UNIV*} **and**
  disjoint: ∀ *p p'*. *p* ≠ *p'* ⟶ *part p* ∩ *part p'* = {} **and**
  card: ∀ *p*. *card* (*part p*) = *voter-amount p*
  using *obtain-partition*[*of V UNIV voter-amount*]
  by *auto*
hence *exactly-one-prof*: ∀ *v* ∈ *V*. ∃!*p*. *v* ∈ *part p*
  by *blast*
then obtain *prof'* :: *'v* ⇒ *'a Ordered-Preference* **where**
  maps-to-prof': ∀ *v* ∈ *V*. *v* ∈ *part* (*prof' v*)
  by *metis*
then obtain *prof* :: *'v* ⇒ *'a Preference-Relation* **where**
  prof: *prof* = (λ *v*. *if v* ∈ *V then ord2pref* (*prof' v*) *else* {})
  by *blast*
hence *election*: (*UNIV*, *V*, *prof*) ∈ *fixed-alt-elections UNIV*
  unfolding *fixed-alt-elections.simps valid-elections-def profile-def*
  using *fin-V ord2pref*
  by *auto*
have ∀ *p*. {*v* ∈ *V*. *prof' v* = *p*} = {*v* ∈ *V*. *v* ∈ *part p*}
  using *maps-to-prof' exactly-one-prof*
  by *blast*
hence ∀ *p*. {*v* ∈ *V*. *prof' v* = *p*} = *part p*
  using *partition*
  by *fastforce*
hence ∀ *p*. *card* {*v* ∈ *V*. *prof' v* = *p*} = *voter-amount p*
  using *card*

by *presburger*
moreover have $\forall\ p.\ \forall\ v.\ (v \in \{v \in V.\ prof'\ v = p\}) = (v \in \{v \in V.\ prof\ v = (ord2pref\ p)\})$
using *prof*
by (*simp add*: *ord2pref-inject*)
ultimately have $\forall\ p.\ card\ \{v \in V.\ prof\ v = (ord2pref\ p)\} = voter\text{-}amount\ p$
by *simp*
hence $\forall\ p{::}'a\ Ordered\text{-}Preference.$
$vote\text{-}fraction\ (ord2pref\ p)\ (UNIV,\ V,\ prof) = Fract\ (voter\text{-}amount\ p)\ (card\ V)$
using *rat-number-collapse fin-V*
by *simp*
moreover have $\forall\ p.\ Fract\ (voter\text{-}amount\ p)\ (card\ V) = (voter\text{-}amount\ p)\ /\ (card\ V)$
unfolding *Fract-of-int-quotient of-rat-divide*
by *simp*
moreover have
$\forall\ p.\ (voter\text{-}amount\ p)\ /\ (card\ V) =$
$((fst\ (fraction\ p)) * (?prod\ div\ (snd\ (fraction\ p))))\ /\ ?prod$
using *card def card-V-eq-sum rewrite-sum*
by *presburger*
moreover have
$\forall\ p.\ ((fst\ (fraction\ p)) * (?prod\ div\ (snd\ (fraction\ p))))\ /\ ?prod =$
$(fst\ (fraction\ p))\ /\ (snd\ (fraction\ p))$
using *rewrite-div pos-prod*
by *auto*
— The percentages of voters voting for each linearly ordered profile in (UNIV, V, prof) equal the entries of the given vector.
ultimately have *eq-vec*:
$\forall\ p :: 'a\ Ordered\text{-}Preference.\ vote\text{-}fraction\ (ord2pref\ p)\ (UNIV,\ V,\ prof) = x'\$p$
using *fract*
by *presburger*
moreover have $\forall\ E \in anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{‘‘}\ \{(UNIV,\ V,\ prof)\}.$
$\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)\ E = vote\text{-}fraction\ (ord2pref\ p)\ (UNIV,\ V,\ prof)$
unfolding *anon-hom*$_{\mathcal{R}}$*.simps*
by *fastforce*
ultimately have $\forall\ E \in anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{‘‘}\ \{(UNIV,\ V,\ prof)\}.$
$\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)\ E = x'\$p$
by *simp*
hence $\forall\ E \in anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{‘‘}\ \{(UNIV,\ V,\ prof)\}.$
$\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)\ E = x'\$p$
using *eq-vec*
by *metis*
hence *vec-entries-match-E-vote-frac*:
$\forall\ p.\ \forall\ E \in anon\text{-}hom_{\mathcal{R}}\ (fixed\text{-}alt\text{-}elections\ UNIV)\ \text{‘‘}\ \{(UNIV,\ V,\ prof)\}.$

vote-fraction (*ord2pref p*) *E* = *x′*$*p*
by *blast*
have ∀ *x* ∈ ℚ. ∀ *y*. *complex-of-rat y* = *complex-of-real x* ⟶ *real-of-rat y* = *x*
using *Re-complex-of-real Re-divide-of-real of-rat.rep-eq of-real-of-int-eq*
by *metis*
hence ∀ *x* ∈ ℚ. ∀ *y*. *complex-of-rat y* = *complex-of-real x* ⟶ *y* = *the-inv real-of-rat x*
using *injI of-rat-eq-iff the-inv-f-f*
by *metis*
with *vec-entries-match-E-vote-frac*
have *all-eq-vec*:
∀ *p*. ∀ *E* ∈ *anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)}.
vote-fraction (*ord2pref p*) *E* = *x*$*p*
using *rat inv*
by *metis*
moreover have (*UNIV*, *V*, *prof*) ∈ *anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)}
using *anon-hom*$_\mathcal{R}$*.simps election*
by *blast*
ultimately have ∀ *p*. *vote-fraction* (*ord2pref p*) `
anon-hom$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)} ⊇ {*x*$*p*}
using *image-insert insert-iff mk-disjoint-insert singletonD subsetI*
by (*metis* (*no-types*, *lifting*))
with *all-eq-vec*
have ∀ *p*. *vote-fraction* (*ord2pref p*) `
anon-hom$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)} = {*x*$*p*}
by *blast*
hence ∀ *p*. *vote-fraction*$_\mathcal{Q}$ (*ord2pref p*)
(*anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)}) = *x*$*p*
using *is-singletonI singleton-inject singleton-set-def-if-card-one*
unfolding *is-singleton-altdef vote-fraction*$_\mathcal{Q}$*.simps* $\pi_\mathcal{Q}$*.simps*
by *metis*
hence *x* = *anon-hom-class-to-vec* (*anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*) `` {(*UNIV*, *V*, *prof*)})
unfolding *anon-hom-class-to-vec.simps*
using *vec-lambda-unique*
by (*metis* (*no-types*, *lifting*))
moreover have (*anon-hom*$_\mathcal{R}$ (*fixed-alt-elections UNIV*)) `` {(*UNIV*, *V*, *prof*)} ∈ *anon-hom*$_\mathcal{Q}$ *UNIV*
unfolding *anon-hom*$_\mathcal{Q}$*.simps quotient-def*
using *election*
by *blast*
ultimately show
x ∈ (*anon-hom-class-to-vec* :: (*′a*, *′v*) *Election set* ⇒ *rat*^(*′a Ordered-Preference*))
` *anon-hom*$_\mathcal{Q}$ *UNIV*
by *blast*
qed
qed

end

Chapter 4

Component Types

4.1 Distance

theory *Distance*
imports *HOL−Library.Extended-Real*
Social-Choice-Types/Voting-Symmetry
begin

A general distance on a set X is a mapping d: $X \times X \mapsto R \cup \{+\infty\}$ such that for every x, y, z in X, the following four conditions are satisfied:

- $d(x,\ y) \geq 0$ (non-negativity);
- $d(x,\ y) = 0$ if and only if $x = y$ (identity of indiscernibles);
- $d(x,\ y) = d(y,\ x)$ (symmetry);
- $d(x,\ y) \leq d(x,\ z) + d(z,\ y)$ (triangle inequality).

  Moreover, a mapping that satisfies all but the second conditions is called a pseudo-distance, whereas a quasi-distance needs to satisfy the first three conditions (and not necessarily the last one).

4.1.1 Definition

type-synonym $'a$ *Distance* $=$ $'a \Rightarrow 'a \Rightarrow$ *ereal*

The un-curried version of a distance is defined on tuples.

fun *tup* :: $'a$ *Distance* $\Rightarrow$ ($'a * 'a \Rightarrow$ *ereal*) **where**
tup d = (λ *pair*. *d* (*fst pair*) (*snd pair*))

definition *distance* :: $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$ *bool* **where**
distance S d $\equiv \forall\ x\ y.\ x \in S \wedge y \in S \longrightarrow d\ x\ x = 0 \wedge 0 \leq d\ x\ y$

4.1.2 Conditions

definition *symmetric* :: $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$ *bool* **where**
symmetric $S\ d \equiv \forall\ x\ y.\ x \in S \land y \in S \longrightarrow d\ x\ y = d\ y\ x$

definition *triangle-ineq* :: $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$ *bool* **where**
triangle-ineq $S\ d \equiv \forall\ x\ y\ z.\ x \in S \land y \in S \land z \in S \longrightarrow d\ x\ z \leq d\ x\ y + d\ y\ z$

definition *eq-if-zero* :: $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$ *bool* **where**
eq-if-zero $S\ d \equiv \forall\ x\ y.\ x \in S \land y \in S \longrightarrow d\ x\ y = 0 \longrightarrow x = y$

definition *vote-distance* :: ($'a$ *Vote set* $\Rightarrow$ $'a$ *Vote Distance* $\Rightarrow$ *bool*) $\Rightarrow$
$'a$ *Vote Distance* $\Rightarrow$ *bool* **where**
vote-distance $\pi\ d \equiv \pi\ \{(A,\ p).$ *linear-order-on* $A\ p \land$ *finite* $A\}\ d$

definition *election-distance* :: (($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool*) $\Rightarrow$
($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
election-distance $\pi\ d \equiv \pi\ \{(A,\ V,\ p).$ *finite-profile* $V\ A\ p\}\ d$

4.1.3 Standard Distance Property

definition *standard* :: ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
standard $d \equiv \forall\ A\ A'\ V\ V'\ p\ p'.\ A \neq A' \lor V \neq V' \longrightarrow d\ (A,\ V,\ p)\ (A',\ V',\ p') = \infty$

4.1.4 Auxiliary Lemmas

fun *arg-min-set* :: ($'b \Rightarrow\ 'a$ :: *ord*) $\Rightarrow$ $'b$ *set* $\Rightarrow$ $'b$ *set* **where**
arg-min-set $f\ A$ = *Collect* (*is-arg-min* f ($\lambda\ a.\ a \in A$))

lemma *arg-min-subset*:
fixes
B :: $'b$ *set* **and**
f :: $'b \Rightarrow\ 'a$ :: *ord*
shows *arg-min-set* $f\ B \subseteq B$
proof (*auto*, *unfold is-arg-min-def*, *simp*)
qed

lemma *sum-monotone*:
fixes
A :: $'a$ *set* **and**
f :: $'a \Rightarrow$ *int* **and**
g :: $'a \Rightarrow$ *int*
assumes $\forall\ a \in A.\ f\ a \leq g\ a$
shows $(\sum\ a \in A.\ f\ a) \leq (\sum\ a \in A.\ g\ a)$
using *assms*
by (*induction A rule*: *infinite-finite-induct*, *simp-all*)

lemma *distrib*:
  fixes
    $A :: 'a\ set$ **and**
    $f :: 'a \Rightarrow int$ **and**
    $g :: 'a \Rightarrow int$
  shows $(\sum a \in A.\ f\ a) + (\sum a \in A.\ g\ a) = (\sum a \in A.\ f\ a + g\ a)$
  using *sum.distrib*
  by *metis*

lemma *distrib-ereal*:
  fixes
    $A :: 'a\ set$ **and**
    $f :: 'a \Rightarrow int$ **and**
    $g :: 'a \Rightarrow int$
  shows $ereal\ (real\text{-}of\text{-}int\ ((\sum a \in A.\ (f{::}'a \Rightarrow int)\ a) + (\sum a \in A.\ g\ a))) =$
    $ereal\ (real\text{-}of\text{-}int\ ((\sum a \in A.\ (f\ a) + (g\ a))))$
  using *distrib*[*of f*]
  by *simp*

lemma *uneq-ereal*:
  fixes
    $x :: int$ **and**
    $y :: int$
  assumes $x \leq y$
  shows $ereal\ (real\text{-}of\text{-}int\ x) \leq ereal\ (real\text{-}of\text{-}int\ y)$
  using *assms*
  by *simp*

4.1.5 Swap Distance

fun *neq-ord* :: $'a$ *Preference-Relation* $\Rightarrow$ $'a$ *Preference-Relation* $\Rightarrow$
        $'a \Rightarrow 'a \Rightarrow bool$ **where**
  $neq\text{-}ord\ r\ s\ a\ b = ((a \preceq_r b \land b \preceq_s a) \lor (b \preceq_r a \land a \preceq_s b))$

fun *pairwise-disagreements* :: $'a\ set \Rightarrow 'a$ *Preference-Relation* $\Rightarrow$
        $'a$ *Preference-Relation* $\Rightarrow ('a \times 'a)\ set$ **where**
  $pairwise\text{-}disagreements\ A\ r\ s = \{(a, b) \in A \times A.\ a \neq b \land neq\text{-}ord\ r\ s\ a\ b\}$

fun *pairwise-disagreements'* :: $'a\ set \Rightarrow 'a$ *Preference-Relation* $\Rightarrow$
        $'a$ *Preference-Relation* $\Rightarrow ('a \times 'a)\ set$ **where**
  $pairwise\text{-}disagreements'\ A\ r\ s =$
      $Set.filter\ (\lambda\ (a, b).\ a \neq b \land neq\text{-}ord\ r\ s\ a\ b)\ (A \times A)$

lemma *set-eq-filter*:
  fixes
    $X :: 'a\ set$ **and**
    $P :: 'a \Rightarrow bool$
  shows $\{x \in X.\ P\ x\} = Set.filter\ P\ X$
  by *auto*

lemma *pairwise-disagreements-eq*[*code*]: *pairwise-disagreements* = *pairwise-disagreements'*
  unfolding *pairwise-disagreements.simps pairwise-disagreements'.simps*
  by *fastforce*

fun *swap* :: *'a Vote Distance* **where**
  swap (*A*, *r*) (*A'*, *r'*) =
    (*if A* = *A'*
    then card (*pairwise-disagreements A r r'*)
    else ∞)

lemma *swap-case-infinity*:
  fixes
    x :: *'a Vote* **and**
    y :: *'a Vote*
  assumes *alts-𝒱 x* ≠ *alts-𝒱 y*
  shows *swap x y* = ∞
  using *assms*
  by (*induction rule*: *swap.induct*, *simp*)

lemma *swap-case-fin*:
  fixes
    x :: *'a Vote* **and**
    y :: *'a Vote*
  assumes *alts-𝒱 x* = *alts-𝒱 y*
  shows *swap x y* = *card* (*pairwise-disagreements* (*alts-𝒱 x*) (*pref-𝒱 x*) (*pref-𝒱 y*))
  using *assms*
  by (*induction rule*: *swap.induct*, *simp*)

4.1.6 Spearman Distance

fun *spearman* :: *'a Vote Distance* **where**
  spearman (*A*, *x*) (*A'*, *y*) =
    (*if A* = *A'*
    then ∑ *a* ∈ *A*. *abs* (*int* (*rank x a*) − *int* (*rank y a*))
    else ∞)

lemma *spearman-case-inf*:
  fixes
    x :: *'a Vote* **and**
    y :: *'a Vote*
  assumes *alts-𝒱 x* ≠ *alts-𝒱 y*
  shows *spearman x y* = ∞
  using *assms*
  by (*induction rule*: *spearman.induct*, *simp*)

lemma *spearman-case-fin*:
  fixes
    x :: *'a Vote* **and**

y :: $'a$ *Vote*
assumes *alts-*$\mathcal{V}$ x = *alts-*$\mathcal{V}$ y
shows *spearman* x y =
($\sum$ $a \in$ *alts-*$\mathcal{V}$ x. *abs* (*int* (*rank* (*pref-*$\mathcal{V}$ x) a) $-$ *int* (*rank* (*pref-*$\mathcal{V}$ y) a)))
using *assms*
by (*induction rule*: *spearman.induct*, *simp*)

4.1.7 Properties

Distances that are invariant under specific relations induce symmetry properties in distance rationalized voting rules.

Definitions

fun *totally-invariant-dist* :: $'x$ *Distance* $\Rightarrow$ $'x$ *rel* $\Rightarrow$ *bool* **where**
totally-invariant-dist d rel = *satisfies* (*tup d*) (*Invariance* (*product-rel rel*))

fun *invariant-dist* :: $'y$ *Distance* $\Rightarrow$ $'x$ *set* $\Rightarrow$ $'y$ *set* $\Rightarrow$ ($'x$, $'y$) *binary-fun* $\Rightarrow$ *bool* **where**
invariant-dist d X Y φ = *satisfies* (*tup d*) (*Invariance* (*equivariance-rel X Y* φ))

definition *distance-anonymity* :: ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-anonymity d $\equiv$
$\forall$ A A' V V' p p' π::($'v \Rightarrow 'v$).
(*bij* π $\longrightarrow$
(d (A, V, p) (A', V', p')) =
(d (*rename* π (A, V, p))) (*rename* π (A', V', p')))

fun *distance-anonymity'* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-anonymity' X d = *invariant-dist d* (*carrier anonymity*$_{\mathcal{G}}$) X (φ*-anon X*)

fun *distance-neutrality* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-neutrality X d = *invariant-dist d* (*carrier neutrality*$_{\mathcal{G}}$) X (φ*-neutr X*)

fun *distance-reversal-symmetry* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-reversal-symmetry X d = *invariant-dist d* (*carrier reversal*$_{\mathcal{G}}$) X (φ*-rev X*)

definition *distance-homogeneity'* :: ($'a$, $'v$::*linorder*) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-homogeneity' X d = *totally-invariant-dist d* (*homogeneity*$_{\mathcal{R}}$$'$ X)

definition *distance-homogeneity* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Election Distance* $\Rightarrow$ *bool* **where**
distance-homogeneity X d = *totally-invariant-dist d* (*homogeneity*$_{\mathcal{R}}$ X)

Auxiliary Lemmas

lemma *rewrite-totally-invariant-dist*:
  fixes
    d :: $'x$ *Distance* **and**
    r :: $'x$ *rel*
  shows *totally-invariant-dist* $d\ r = (\forall\ (x, y) \in r.\ \forall\ (a, b) \in r.\ d\ a\ x = d\ b\ y)$
proof (*safe*)
  fix
    a :: $'x$ **and**
    b :: $'x$ **and**
    x :: $'x$ **and**
    y :: $'x$
  assume
    inv: *totally-invariant-dist* $d\ r$ **and**
    $(a, b) \in r$ **and**
    $(x, y) \in r$
  hence *rel*: $((a, x), (b, y)) \in$ *product-rel* r
    by *simp*
  hence *tup* $d\ (a, x) =$ *tup* $d\ (b, y)$
    using *inv*
    unfolding *totally-invariant-dist.simps satisfies.simps*
    by *simp*
  thus $d\ a\ x = d\ b\ y$
    by *simp*
next
  show $\forall\ (x, y) \in r.\ \forall\ (a, b) \in r.\ d\ a\ x = d\ b\ y \Longrightarrow$ *totally-invariant-dist* $d\ r$
  proof (*unfold totally-invariant-dist.simps satisfies.simps product-rel.simps*, *safe*)
    fix
      a :: $'x$ **and**
      b :: $'x$ **and**
      x :: $'x$ **and**
      y :: $'x$
    assume
      $\forall\ (x, y) \in r.\ \forall\ (a, b) \in r.\ d\ a\ x = d\ b\ y$ **and**
      (*fst* (x, a), *fst* $(y, b)) \in r$ **and**
      (*snd* (x, a), *snd* $(y, b)) \in r$
    hence $d\ x\ a = d\ y\ b$
      by *auto*
    thus *tup* $d\ (x, a) =$ *tup* $d\ (y, b)$
      by *simp*
  qed
qed

lemma *rewrite-invariant-dist*:
  fixes
    d :: $'y$ *Distance* **and**
    X :: $'x$ *set* **and**
    Y :: $'y$ *set* **and**
    φ :: $('x, 'y)$ *binary-fun*

shows *invariant-dist* $d\ X\ Y\ \varphi = (\forall\ x \in X.\ \forall\ y \in Y.\ \forall\ z \in Y.\ d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z))$
proof (*safe*)
fix
$x :: 'x$ **and**
$y :: 'y$ **and**
$z :: 'y$
assume
$x \in X$ **and**
$y \in Y$ **and**
$z \in Y$ **and**
invariant-dist $d\ X\ Y\ \varphi$
thus $d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z)$
by *fastforce*
next
show $\forall\ x \in X.\ \forall\ y \in Y.\ \forall\ z \in Y.\ d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z) \Longrightarrow$ *invariant-dist* $d\ X\ Y\ \varphi$
proof (*unfold invariant-dist.simps satisfies.simps equivariance-rel.simps*, *safe*)
fix
$x :: 'x$ **and**
$a :: 'y$ **and**
$b :: 'y$
assume
$\forall\ x \in X.\ \forall\ y \in Y.\ \forall\ z \in Y.\ d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z)$ **and**
$x \in X$ **and**
$a \in Y$ **and**
$b \in Y$
hence $d\ a\ b = d\ (\varphi\ x\ a)\ (\varphi\ x\ b)$
by *blast*
thus *tup* $d\ (a,\ b) =$ *tup* $d\ (\varphi\ x\ a,\ \varphi\ x\ b)$
by *simp*
qed
qed

lemma *invar-dist-image*:
fixes
$d :: 'y$ *Distance* **and**
$G :: 'x$ *monoid* **and**
$Y :: 'y$ *set* **and**
$Y' :: 'y$ *set* **and**
$\varphi :: ('x,\ 'y)$ *binary-fun* **and**
$y :: 'y$ **and**
$g :: 'x$
assumes
invar-d: *invariant-dist* d (*carrier* G) $Y\ \varphi$ **and**
Y'-*in*-Y: $Y' \subseteq Y$ **and**
action-φ: *group-action* $G\ Y\ \varphi$ **and**
g-carrier: $g \in$ *carrier* G **and**
y-in-Y: $y \in Y$

shows $d\ (\varphi\ g\ y)$ ‘ $(\varphi\ g)$ ‘ $Y' = d\ y$ ‘ Y'
proof (*safe*)
fix $y' :: 'y$
assume y'-*in*-Y': $y' \in Y'$
hence $((y, y'), ((\varphi\ g\ y), (\varphi\ g\ y'))) \in$ *equivariance-rel* (*carrier* G) $Y\ \varphi$
using Y'-*in*-Y *y-in-Y g-carrier*
unfolding *equivariance-rel.simps*
by *blast*
hence *eq-dist*: *tup* $d\ ((\varphi\ g\ y), (\varphi\ g\ y')) =$ *tup* $d\ (y, y')$
using *invar-d*
unfolding *invariant-dist.simps*
by *fastforce*
thus $d\ (\varphi\ g\ y)\ (\varphi\ g\ y') \in d\ y$ ‘ Y'
using y'-*in*-Y'
by *simp*
have $\varphi\ g\ y' \in \varphi\ g$ ‘ Y'
using y'-*in*-Y'
by *simp*
thus $d\ y\ y' \in d\ (\varphi\ g\ y)$ ‘ $\varphi\ g$ ‘ Y'
using *eq-dist*
by (*simp add*: *rev-image-eqI*)
qed

lemma *swap-neutral*: *invariant-dist swap* (*carrier neutrality*$_{\mathcal{G}}$)
UNIV $(\lambda\ \pi\ (A, q).\ (\pi$ ‘ A, *rel-rename* $\pi\ q))$
proof (*simp only*: *rewrite-invariant-dist*, *safe*)
fix
$\pi :: 'a \Rightarrow 'a$ **and**
$A :: 'a\ set$ **and**
$q :: 'a\ rel$ **and**
$A' :: 'a\ set$ **and**
$q' :: 'a\ rel$
assume $\pi \in$ *carrier neutrality*$_{\mathcal{G}}$
hence *bij*: *bij* π
unfolding *neutrality*$_{\mathcal{G}}$-*def*
using *rewrite-carrier*
by *blast*
show *swap* $(A, q)\ (A', q') =$ *swap* $(\pi$ ‘ A, *rel-rename* $\pi\ q)$ $(\pi$ ‘ A', *rel-rename* π
$q')$
proof (*cases* $A = A'$)
let *?f* $= (\lambda\ (a, b).\ (\pi\ a, \pi\ b))$
let *?swap-set* $= \{(a, b) \in A \times A.\ a \neq b \wedge$ *neq-ord* $q\ q'\ a\ b\}$
let *?swap-set'* $=$
$\{(a, b) \in \pi$ ‘ $A \times \pi$ ‘ $A.\ a \neq b \wedge$ *neq-ord* (*rel-rename* $\pi\ q$) (*rel-rename* $\pi\ q'$)
$a\ b\}$
let *?rel* $= \{(a, b) \in A \times A.\ a \neq b \wedge$ *neq-ord* $q\ q'\ a\ b\}$
case *True*
hence π ‘ $A = \pi$ ‘ A'
by *simp*

hence *swap* (π ‘ *A*, *rel-rename* π *q*) (π ‘ *A′*, *rel-rename* π *q′*) = *card ?swap-set′*
by *simp*
moreover have *bij-betw ?f ?swap-set ?swap-set′*
proof (*unfold bij-betw-def inj-on-def*, *standard*, *standard*, *standard*, *standard*)
fix
x :: $'a \times 'a$ **and**
y :: $'a \times 'a$
assume
$x \in$ *?swap-set* **and**
$y \in$ *?swap-set* **and**
?f x = *?f y*
hence π (*fst x*) = π (*fst y*) $\wedge$ π (*snd x*) = π (*snd y*)
by *auto*
hence *fst x* = *fst y* $\wedge$ *snd x* = *snd y*
using *bij bij-pointE*
by *metis*
thus $x = y$
using *prod.expand*
by *metis*
next
show *?f* ‘ *?swap-set* = *?swap-set′*
proof
have $\forall$ *a b*. (*a*, *b*) $\in A \times A \longrightarrow$ (π *a*, π *b*) $\in \pi$ ‘ $A \times \pi$ ‘ A
by *simp*
moreover have $\forall$ *a b*. $a \neq b \longrightarrow \pi\ a \neq \pi\ b$
using *bij bij-pointE*
by *metis*
moreover have
$\forall$ *a b*. *neq-ord q q′ a b* $\longrightarrow$ *neq-ord* (*rel-rename* π *q*) (*rel-rename* π *q′*) (π *a*) (π *b*)
unfolding *neq-ord.simps rel-rename.simps*
by *auto*
ultimately show *?f* ‘ *?swap-set* $\subseteq$ *?swap-set′*
by *auto*
next
have $\forall$ *a b*. (*a*, *b*) $\in$ (*rel-rename* π *q*) $\longrightarrow$ (*the-inv* π *a*, *the-inv* π *b*) $\in q$
unfolding *rel-rename.simps*
using *bij bij-is-inj the-inv-f-f*
by *fastforce*
moreover have $\forall$ *a b*. (*a*, *b*) $\in$ (*rel-rename* π *q′*) $\longrightarrow$ (*the-inv* π *a*, *the-inv* π *b*) $\in q'$
unfolding *rel-rename.simps*
using *bij bij-is-inj the-inv-f-f*
by *fastforce*
ultimately have $\forall$ *a b*. *neq-ord* (*rel-rename* π *q*) (*rel-rename* π *q′*) *a b* $\longrightarrow$
neq-ord q q′ (*the-inv* π *a*) (*the-inv* π *b*)
by *simp*
moreover have $\forall$ *a b*. (*a*, *b*) $\in \pi$ ‘ $A \times \pi$ ‘ $A \longrightarrow$ (*the-inv* π *a*, *the-inv* π *b*) $\in A \times A$

```
      using bij bij-is-inj f-the-inv-into-f inj-image-mem-iff
      by fastforce
    moreover have ∀ a b. a ≠ b ⟶ the-inv π a ≠ the-inv π b
      using bij UNIV-I bij-betw-imp-surj bij-is-inj f-the-inv-into-f
      by metis
    ultimately have
      ∀ a b. (a, b) ∈ ?swap-set' ⟶ (the-inv π a, the-inv π b) ∈ ?swap-set
      by blast
    moreover have ∀ a b. (a, b) = ?f (the-inv π a, the-inv π b)
      using f-the-inv-into-f-bij-betw bij
      by fastforce
    ultimately show ?swap-set' ⊆ ?f ` ?swap-set
      by blast
  qed
qed
moreover have card ?swap-set = swap (A, q) (A', q')
  using True
  by simp
ultimately show ?thesis
  by (simp add: bij-betw-same-card)
next
  case False
  hence π ` A ≠ π ` A'
    using bij bij-is-inj inj-image-eq-iff
    by metis
  hence swap (A, q) (A', q') = ∞ ∧
    swap (π ` A, rel-rename π q) (π ` A', rel-rename π q') = ∞
    using False
    by simp
  thus ?thesis
    by simp
qed
qed

end
```

4.2 Votewise Distance

```
theory Votewise-Distance
  imports Social-Choice-Types/Norm
          Distance
begin
```

Votewise distances are a natural class of distances on elections which depend on the submitted votes in a simple and transparent manner. They are formed by using any distance d on individual orders and combining the components with a norm on $\mathbb{R}^n$.

4.2.1 Definition

fun *votewise-distance* :: *'a Vote Distance* ⇒ *Norm*
⇒ (*'a*,*'v*::*linorder*) *Election Distance* **where**
votewise-distance d n (*A*, *V*, *p*) (*A'*, *V'*, *p'*) =
(*if* (*finite V*) ∧ *V* = *V'* ∧ (*V* ≠ {} ∨ *A* = *A'*)
then n (*map2* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*)) (*to-list V p*) (*to-list V' p'*))
else ∞)

4.2.2 Inference Rules

lemma *symmetric-norm-inv-under-map2-permute*:
fixes
d :: *'a Vote Distance* **and**
n :: *Norm* **and**
A :: *'a set* **and**
A' :: *'a set* **and**
φ :: *nat* ⇒ *nat* **and**
p :: (*'a Preference-Relation*) *list* **and**
p' :: (*'a Preference-Relation*) *list*
assumes
perm: φ *permutes* {*0* ..< *length p*} **and**
len-eq: *length p* = *length p'* **and**
sym-n: *symmetry n*
shows *n* (*map2* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*)) *p p'*)
= *n* (*map2* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*)) (*permute-list* φ *p*) (*permute-list* φ *p'*))
proof −
let *?z* = *zip p p'* **and**
?lt-len = λ *i*. {..< *length i*} **and**
?c-prod = *case-prod* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*))
let *?listpi* = λ *q*. *permute-list* φ *q*
let *?q* = *?listpi p* **and**
?q' = *?listpi p'*
have *listpi-sym*: ∀ *l*. (*length l* = *length p* ⟶ *?listpi l* <~~> *l*)
using *mset-permute-list perm atLeast-upt*
by *simp*
moreover have *length* (*map2* (λ *x y*. *d* (*A*, *x*) (*A'*, *y*)) *p p'*) = *length p*
using *len-eq*
by *simp*
ultimately have (*map2* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*)) *p p'*)
<~~> (*?listpi* (*map2* (λ *x y*. *d* (*A*, *x*) (*A'*, *y*)) *p p'*))
by *metis*
hence *n* (*map2* (λ *q q'*. *d* (*A*, *q*) (*A'*, *q'*)) *p p'*)
= *n* (*?listpi* (*map2* (λ *x y*. *d* (*A*, *x*) (*A'*, *y*)) *p p'*))
using *sym-n*
unfolding *symmetry-def*
by *blast*
also have ... = *n* (*map* (*case-prod* (λ *x y*. *d* (*A*, *x*) (*A'*, *y*)))
(*?listpi* (*zip p p'*)))
using *permute-list-map*[*of* φ *?z ?c-prod*] *perm len-eq atLeast-upt*

by *simp*
also have ... = n (*map2* (λ x y. d (A, x) (A', y)) (*?listpi* p) (*?listpi* p'))
using *len-eq perm atLeast-upt*
by (*simp add*: *permute-list-zip*)
finally show *?thesis*
by *simp*
qed

lemma *permute-invariant-under-map*:
fixes
l :: *'a list* **and**
ls :: *'a list*
assumes l <~~> *ls*
shows *map f l* <~~> *map f ls*
using *assms*
by *simp*

lemma *linorder-rank-injective*:
fixes
V :: *'v::linorder set* **and**
v :: *'v* **and**
v' :: *'v*
assumes
v-in-V: $v \in V$ **and**
v'-in-V: $v' \in V$ **and**
v'-neq-v: $v' \neq v$ **and**
fin-V: *finite* V

shows *card* $\{x \in V.\ x < v\} \neq$ *card* $\{x \in V.\ x < v'\}$
proof –
have $v < v' \vee v' < v$
using *v'-neq-v linorder-less-linear*
by *metis*
hence $\{x \in V.\ x < v\} \subset \{x \in V.\ x < v'\} \vee \{x \in V.\ x < v'\} \subset \{x \in V.\ x < v\}$
using *v-in-V v'-in-V dual-order.strict-trans*
by *blast*
thus *?thesis*
using *assms sorted-list-of-set-nth-equals-card*
by (*metis* (*full-types*))
qed

lemma *permute-invariant-under-coinciding-funs*:
fixes
l :: *'v list* **and**
π*-1* :: *nat* $\Rightarrow$ *nat* **and**
π*-2* :: *nat* $\Rightarrow$ *nat*
assumes $\forall$ i < *length* l. π-*1* i = π-*2* i
shows *permute-list* π*-1* l = *permute-list* π*-2* l
using *assms*

unfolding *permute-list-def*
by *simp*

lemma *symmetric-norm-imp-distance-anonymous*:
fixes
d :: *'a Vote Distance* **and**
n :: *Norm*
assumes *symmetry n*
shows *distance-anonymity* (*votewise-distance d n*)
proof (*unfold distance-anonymity-def*, *safe*)
fix
A :: *'a set* **and**
A' :: *'a set* **and**
V :: *'v::linorder set* **and**
V' :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
p' :: (*'a*, *'v*) *Profile* **and**
π :: $'v \Rightarrow 'v$
let *?rn1* = *rename* π (A, V, p) **and**
?rn2 = *rename* π (A', V', p') **and**
?rn-V = π ‘ V **and**
?rn-V' = π ‘ V' **and**
?rn-p = $p \circ$ (*the-inv* π) **and**
?rn-p' = $p' \circ$ (*the-inv* π) **and**
?len = *length* (*to-list V p*) **and**
?sl-V = *sorted-list-of-set V*
let *?perm* = λ *i*. (*card* ({$v \in$ *?rn-V*. $v < \pi$ (*?sl-V*!*i*)})) **and**
?perm-total = (λ *i*. (*if* (*i* < *?len*)
then card ({$v \in$ *?rn-V*. $v < \pi$ (*?sl-V*!*i*)})
else i))
assume *bij*: *bij* π
show *votewise-distance d n* (A, V, p) (A', V', p') = *votewise-distance d n ?rn1 ?rn2*
proof −
have *rn-A-eq-A*: *fst ?rn1* = A
by *simp*
have *rn-A'-eq-A'*: *fst ?rn2* = A'
by *simp*
have *rn-V-eq-pi-V*: *fst* (*snd ?rn1*) = *?rn-V*
by *simp*
have *rn-V'-eq-pi-V'*: *fst* (*snd ?rn2*) = *?rn-V'*
by *simp*
have *rn-p-eq-pi-p*: *snd* (*snd ?rn1*) = *?rn-p*
by *simp*
have *rn-p'-eq-pi-p'*: *snd* (*snd ?rn2*) = *?rn-p'*
by *simp*
show *?thesis*
proof (*cases finite* $V \wedge V = V' \wedge (V \neq \{\} \vee A = A')$)
case *False*

hence *inf-dist*: *votewise-distance d n* $(A,\ V,\ p)\ (A',\ V',\ p') = \infty$
by *auto*
moreover have *infinite V* $\Longrightarrow$ *infinite ?rn-V*
using *False bij bij-betw-finite bij-betw-subset False subset-UNIV*
by *metis*
moreover have $V \neq V' \Longrightarrow$ *?rn-V* $\neq$ *?rn-V'*
using *bij bij-def inj-image-mem-iff subsetI subset-antisym*
by *metis*
moreover have $V = \{\} \Longrightarrow$ *?rn-V* $= \{\}$
using *bij*
by *simp*
ultimately have *inf-dist-rename*: *votewise-distance d n ?rn1 ?rn2* $= \infty$
using *False*
by *auto*
thus *votewise-distance d n* $(A,\ V,\ p)\ (A',\ V',\ p')$ = *votewise-distance d n ?rn1 ?rn2*
using *inf-dist*
by *simp*
next
case *True*

have *perm-funs-coincide*: $\forall\ i <$ *?len. ?perm i* = *?perm-total i*
by *presburger*

have *lengths-eq*: *?len* = *length* (*to-list* V' p')
using *True*
by *simp*

have *rn-V-permutes*: (*to-list V p*) = *permute-list ?perm* (*to-list ?rn-V ?rn-p*)
using *assms to-list-permutes-under-bij bij to-list-permutes-under-bij*
unfolding *comp-def*
by (*metis* (*no-types*))
hence *len-V-rn-V-eq*: *?len* = *length* (*to-list ?rn-V ?rn-p*)
by *simp*
hence *permute-list ?perm* (*to-list ?rn-V ?rn-p*)
= *permute-list ?perm-total* (*to-list ?rn-V ?rn-p*)
using *permute-invariant-under-coinciding-funs*[*of* (*to-list ?rn-V ?rn-p*)]
perm-funs-coincide
by *presburger*
hence *rn-list-perm-list-V*: (*to-list V p*) = *permute-list ?perm-total* (*to-list ?rn-V ?rn-p*)
using *rn-V-permutes*
by *metis*

have *rn-V'-permutes*: (*to-list* V' p') = *permute-list ?perm* (*to-list ?rn-V' ?rn-p'*)
unfolding *comp-def*

using *True bij to-list-permutes-under-bij*
by (*metis* (*no-types*))
hence *permute-list ?perm* (*to-list ?rn-V′ ?rn-p′*)
= *permute-list ?perm-total* (*to-list ?rn-V′ ?rn-p′*)
using *permute-invariant-under-coinciding-funs*[*of* (*to-list ?rn-V′ ?rn-p′*)]
perm-funs-coincide lengths-eq
by *fastforce*
hence *rn-list-perm-list-V′*:
(*to-list V′ p′*) = *permute-list ?perm-total* (*to-list ?rn-V′ ?rn-p′*)
using *rn-V′-permutes*
by *metis*

have *rn-lengths-eq*: *length* (*to-list ?rn-V ?rn-p*) = *length* (*to-list ?rn-V′ ?rn-p′*)
using *len-V-rn-V-eq lengths-eq rn-V′-permutes*
by *simp*
have *perm*: *?perm-total permutes* {*0* ..< *?len*}
proof −

have ∀ *i j*. (*i* < *?len* ∧ *j* < *?len* ∧ *i* ≠ *j*
⟶ π ((*sorted-list-of-set V*)!*i*) ≠ π ((*sorted-list-of-set V*)!*j*))
using *bij bij-pointE True nth-eq-iff-index-eq length-map*
sorted-list-of-set.distinct-sorted-key-list-of-set to-list.elims
by (*metis* (*mono-tags*, *opaque-lifting*))
moreover have *in-bnds-imp-img-el*: ∀ *i*. *i* < *?len* ⟶ π ((*sorted-list-of-set V*)!*i*) ∈ π ‘ *V*
using *True image-eqI nth-mem sorted-list-of-set*(*1*) *to-list.simps length-map*
by *metis*
ultimately have ∀ *i* < *?len*. ∀ *j* < *?len*. (*?perm-total i* = *?perm-total j* ⟶ *i* = *j*)
using *linorder-rank-injective Collect-cong True finite-imageI*
by (*metis* (*no-types*, *lifting*))
moreover have ∀ *i*. *i* < *?len* ⟶ *i* ∈ {*0* ..< *?len*}
by *simp*
ultimately have ∀ *i* ∈ {*0* ..< *?len*}. ∀ *j* ∈ {*0* ..< *?len*}.
(*?perm-total i* = *?perm-total j* ⟶ *i* = *j*)
by *simp*
hence *inj*: *inj-on ?perm-total* {*0* ..< *?len*}
unfolding *inj-on-def*
by *simp*
have ∀ *v′* ∈ (π ‘ *V*). (*card* ({*v*∈(π ‘ *V*). *v* < *v′*})) < *card* (π ‘ *V*)
using *card-seteq True finite-imageI less-irrefl linorder-not-le mem-Collect-eq subsetI*
by (*metis* (*no-types*, *lifting*))
moreover have ∀ *i* < *?len*. π ((*sorted-list-of-set V*)!*i*) ∈ π ‘ *V*
using *in-bnds-imp-img-el*
by *simp*
moreover have *card* (π ‘ *V*) = *card V*
using *bij bij-betw-same-card bij-betw-subset top-greatest*
by *metis*

moreover have *card V* = *?len*
by *simp*
ultimately have *bounded-img*: ∀ *i*. (*i* < *?len* ⟶ *?perm-total i* ∈ {*0* ..< *?len*})
using *atLeast0LessThan lessThan-iff*
by (*metis* (*full-types*))
hence ∀ *i*. *i* < *?len* ⟶ *?perm-total i* ∈ {*0* ..< *?len*}
by *simp*
moreover have ∀ *i*. *i* ∈ {*0* ..< *?len*} ⟶ *i* < *?len*
using *atLeastLessThan-iff*
by *blast*
ultimately have ∀ *i*. *i* ∈ {*0* ..< *?len*} ⟶ *?perm-total i* ∈ {*0* .. *?len*}
by *fastforce*
hence *?perm-total* ‘ {*0* ..< *?len*} ⊆ {*0* ..< *?len*}
using *bounded-img*
by *force*
hence *?perm-total* ‘ {*0* ..< *?len*} = {*0* ..< *?len*}
using *inj card-image card-subset-eq finite-atLeastLessThan*
by *blast*
hence *bij-perm*: *bij-betw ?perm-total* {*0* ..< *?len*} {*0* ..< *?len*}
using *inj bij-betw-def atLeast0LessThan*
by *blast*
thus *?thesis*
using *atLeast0LessThan bij-imp-permutes*
by *fastforce*
qed
have *votewise-distance d n ?rn1 ?rn2*
= *n* (*map2* (λ *q q′*. *d* (*A*, *q*) (*A′*, *q′*)) (*to-list ?rn-V ?rn-p*) (*to-list ?rn-V′ ?rn-p′*))
using *True rn-A-eq-A rn-A′-eq-A′ rn-V-eq-pi-V rn-V′-eq-pi-V′ rn-p-eq-pi-p rn-p′-eq-pi-p′*
by *force*
also have ... = *n* (*map2* (λ *q q′*. *d* (*A*, *q*) (*A′*, *q′*))
(*permute-list ?perm-total* (*to-list ?rn-V ?rn-p*))
(*permute-list ?perm-total* (*to-list ?rn-V′ ?rn-p′*)))
using *symmetric-norm-inv-under-map2-permute*[*of ?perm-total to-list ?rn-V ?rn-p*]
assms perm rn-lengths-eq len-V-rn-V-eq
by *simp*
also have ... = *n* (*map2* (λ *q q′*. *d* (*A*, *q*) (*A′*, *q′*)) (*to-list V p*) (*to-list V′ p′*))
using *rn-list-perm-list-V rn-list-perm-list-V′*
by *presburger*
also have *votewise-distance d n* (*A*, *V*, *p*) (*A′*, *V′*, *p′*)
= *n* (*map2* (λ *q q′*. *d* (*A*, *q*) (*A′*, *q′*)) (*to-list V p*) (*to-list V′ p′*))
using *True*
by *force*
finally show *votewise-distance d n* (*A*, *V*, *p*) (*A′*, *V′*, *p′*)
= *votewise-distance d n ?rn1 ?rn2*

```
      by linarith
    qed
  qed
qed

lemma neutral-dist-imp-neutral-votewise-dist:
  fixes
    d :: 'a Vote Distance and
    n :: Norm
  defines vote-action ≡ (λ π (A, q). (π ‘ A, rel-rename π q))
  assumes invar: invariant-dist d (carrier neutrality_𝒢) UNIV vote-action
  shows distance-neutrality valid-elections (votewise-distance d n)
proof (unfold distance-neutrality.simps,
      simp only: rewrite-invariant-dist,
      safe)
  fix
    A :: 'a set and
    A' :: 'a set and
    V :: 'v::linorder set and
    V' :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile and
    π :: 'a ⇒ 'a
  assume
    carrier: π ∈ carrier neutrality_𝒢 and
    valid: (A, V, p) ∈ valid-elections and
    valid': (A', V', p') ∈ valid-elections
  hence bij: bij π
    unfolding neutrality_𝒢-def
    using rewrite-carrier
    by blast
  thus votewise-distance d n (A, V, p) (A', V', p') =
        votewise-distance d n
          (φ-neutr valid-elections π (A, V, p)) (φ-neutr valid-elections π (A', V',
p'))
  proof (cases finite V ∧ V = V' ∧ (V ≠ {} ∨ A = A'))
    case True
    hence finite V ∧ V = V' ∧ (V ≠ {} ∨ π ‘ A = π ‘ A')
      by metis
    hence votewise-distance d n
          (φ-neutr valid-elections π (A, V, p)) (φ-neutr valid-elections π (A', V',
p'))
      = n (map2 (λ q q'. d (π ‘ A, q) (π ‘ A', q'))
         (to-list V (rel-rename π ∘ p)) (to-list V' (rel-rename π ∘ p')))
      using valid valid'
      by auto
    also have (map2 (λ q q'. d (π ‘ A, q) (π ‘ A', q'))
          (to-list V (rel-rename π ∘ p)) (to-list V' (rel-rename π ∘ p')))
      = (map2 (λ q q'. d (π ‘ A, q) (π ‘ A', q'))
```

```
      (map (rel-rename π) (to-list V p)) (map (rel-rename π) (to-list V′ p′)))
    using to-list-comp
    by metis
  also have (map2 (λ q q′. d (π ‘ A, q) (π ‘ A′, q′))
        (map (rel-rename π) (to-list V p)) (map (rel-rename π) (to-list V′ p′)))
      = (map2 (λ q q′. d (π ‘ A, rel-rename π q) (π ‘ A′, rel-rename π q′))
        (to-list V p) (to-list V′ p′))
    using map2-helper
    by blast
  also have (λ q q′. d (π ‘ A, rel-rename π q) (π ‘ A′, rel-rename π q′))
      = (λ q q′. d (A, q) (A′, q′))
    using rewrite-invariant-dist[of d carrier neutrality_G UNIV vote-action]
        invar carrier UNIV-I case-prod-conv
    unfolding vote-action-def
    by (metis (no-types, lifting))
  finally have votewise-distance d n
      (φ-neutr valid-elections π (A, V, p)) (φ-neutr valid-elections π (A′, V′, p′))
      = n (map2 (λ q q′. d (A, q) (A′, q′)) (to-list V p) (to-list V′ p′))
    by simp
  also have votewise-distance d n (A, V, p) (A′, V′, p′)
      = n (map2 (λ q q′. d (A, q) (A′, q′)) (to-list V p) (to-list V′ p′))
    using True
    by auto
  finally show ?thesis
    by simp
 next
  case False
  hence ¬ (finite V ∧ V = V′ ∧ (V ≠ {} ∨ π ‘ A = π ‘ A′))
    using bij bij-is-inj inj-image-eq-iff
    by metis
  hence votewise-distance d n
      (φ-neutr valid-elections π (A, V, p)) (φ-neutr valid-elections π (A′, V′, p′))
= ∞
    using valid valid′
    by auto
  also have votewise-distance d n (A, V, p) (A′, V′, p′) = ∞
    using False
    by auto
  finally show ?thesis
    by simp
 qed
qed

end
```

4.3 Consensus

theory *Consensus*
imports *Social-Choice-Types/Voting-Symmetry*
begin

An election consisting of a set of alternatives and preferential votes for each voter (a profile) is a consensus if it has an undisputed winner reflecting a certain concept of fairness in the society.

4.3.1 Definition

type-synonym $('a, 'v)$ *Consensus* $= ('a, 'v)$ *Election* $\Rightarrow$ *bool*

4.3.2 Consensus Conditions

Nonempty alternative set.

fun $\textit{nonempty-set}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{nonempty-set}_\mathcal{C}\ (A,\ V,\ p) = (A \neq \{\})$

Nonempty profile, i.e., nonempty voter set. Note that this is also true if p v = for all voters v in V.

fun $\textit{nonempty-profile}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{nonempty-profile}_\mathcal{C}\ (A,\ V,\ p) = (V \neq \{\})$

Equal top ranked alternatives.

fun $\textit{equal-top}_\mathcal{C}{'}$ :: $'a \Rightarrow ('a, 'v)$ *Consensus* **where**
$\textit{equal-top}_\mathcal{C}{'}\ a\ (A,\ V,\ p) = (a \in A \wedge (\forall\ v \in V.\ \textit{above}\ (p\ v)\ a = \{a\}))$

fun $\textit{equal-top}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{equal-top}_\mathcal{C}\ c = (\exists\ a.\ \textit{equal-top}_\mathcal{C}{'}\ a\ c)$

Equal votes.

fun $\textit{equal-vote}_\mathcal{C}{'}$ :: $'a$ *Preference-Relation* $\Rightarrow ('a, 'v)$ *Consensus* **where**
$\textit{equal-vote}_\mathcal{C}{'}\ r\ (A,\ V,\ p) = (\forall\ v \in V.\ (p\ v) = r)$

fun $\textit{equal-vote}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{equal-vote}_\mathcal{C}\ c = (\exists\ r.\ \textit{equal-vote}_\mathcal{C}{'}\ r\ c)$

Unanimity condition.

fun $\textit{unanimity}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{unanimity}_\mathcal{C}\ c = (\textit{nonempty-set}_\mathcal{C}\ c \wedge \textit{nonempty-profile}_\mathcal{C}\ c \wedge \textit{equal-top}_\mathcal{C}\ c)$

Strong unanimity condition.

fun $\textit{strong-unanimity}_\mathcal{C}$ :: $('a, 'v)$ *Consensus* **where**
$\textit{strong-unanimity}_\mathcal{C}\ c = (\textit{nonempty-set}_\mathcal{C}\ c \wedge \textit{nonempty-profile}_\mathcal{C}\ c \wedge \textit{equal-vote}_\mathcal{C}\ c)$

4.3.3 Properties

definition *consensus-anonymity* :: ($'a$, $'v$) *Consensus* $\Rightarrow$ *bool* **where**
  consensus-anonymity $c \equiv$
    ($\forall$ A V p π::($'v \Rightarrow 'v$).
      bij $\pi \longrightarrow$
        (*let* (A', V', q) = (*rename* π (A, V, p)) *in*
          profile V A p $\longrightarrow$ *profile* V' A' q
          $\longrightarrow$ c (A, V, p) $\longrightarrow$ c (A', V', q)))

fun *consensus-neutrality* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$) *Consensus* $\Rightarrow$ *bool* **where**
  consensus-neutrality X c = *satisfies* c (*Invariance* (*neutrality*$_\mathcal{R}$ X))

4.3.4 Auxiliary Lemmas

lemma *cons-anon-conj*:
  fixes
    c1 :: ($'a$, $'v$) *Consensus* **and**
    c2 :: ($'a$, $'v$) *Consensus*
  assumes
    anon1: *consensus-anonymity c1* **and**
    anon2: *consensus-anonymity c2*
  shows *consensus-anonymity* (λ *e*. *c1 e* $\wedge$ *c2 e*)
proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    q :: ($'a$, $'v$) *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume
    bij: *bij* π **and**
    prof: *profile* V A p **and**
    renamed: *rename* π (A, V, p) = (A', V', q) **and**
    c1: *c1* (A, V, p) **and**
    c2: *c2* (A, V, p)
  hence *profile* V' A' q
    using *rename-sound renamed bij fst-conv rename.simps*
    by *metis*
  thus *c1* (A', V', q) $\wedge$ *c2* (A', V', q)
    using *bij renamed c1 c2 assms prof*
    unfolding *consensus-anonymity-def*
    by *auto*
qed

theorem *cons-conjunction-invariant*:
  fixes
    $\mathfrak{C}$ :: ($'a$, $'v$) *Consensus set* **and**

rel :: $('a, 'v)$ *Election rel*
defines $C \equiv (\lambda\ E.\ (\forall\ C' \in \mathfrak{C}.\ C'\ E))$
assumes $\bigwedge C'.\ C' \in \mathfrak{C} \Longrightarrow$ *satisfies* C' (*Invariance rel*)
shows *satisfies* C (*Invariance rel*)
proof (*unfold satisfies.simps*, *standard*, *standard*, *standard*)
fix
E :: $('a,'v)$ *Election* **and**
E' :: $('a,'v)$ *Election*
assume $(E,E') \in rel$
hence $\forall\ C' \in \mathfrak{C}.\ C'\ E = C'\ E'$
using *assms*
unfolding *satisfies.simps*
by *blast*
thus $C\ E = C\ E'$
unfolding *C-def*
by *blast*
qed

lemma *cons-anon-invariant*:
fixes
c :: $('a, 'v)$ *Consensus* **and**
A :: *'a set* **and**
A' :: *'a set* **and**
V :: *'v set* **and**
V' :: *'v set* **and**
p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
anon: *consensus-anonymity* c **and**
bij: *bij* π **and**
prof-p: *profile* $V\ A\ p$ **and**
renamed: *rename* $\pi\ (A,\ V,\ p) = (A',\ V',\ q)$ **and**
cond-c: $c\ (A,\ V,\ p)$
shows $c\ (A',\ V',\ q)$
proof −
have *profile* $V'\ A'\ q$
using *rename-sound bij renamed prof-p*
by *fastforce*
thus *?thesis*
using *anon cond-c renamed rename-finite bij prof-p*
unfolding *consensus-anonymity-def Let-def*
by *auto*
qed

lemma *ex-anon-cons-imp-cons-anonymous*:
fixes
b :: $('a, 'v)$ *Consensus* **and**
b':: $'b \Rightarrow ('a, 'v)$ *Consensus*

assumes
  general-cond-b: $b = (\lambda\ E.\ \exists\ x.\ b'\ x\ E)$ **and**
  all-cond-anon: $\forall\ x.$ *consensus-anonymity* $(b'\ x)$
shows *consensus-anonymity b*
proof (*unfold consensus-anonymity-def Let-def*, *safe*)
  fix
    $A :: 'a\ set$ **and**
    $A' :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $V' :: 'v\ set$ **and**
    $p :: ('a, 'v)$ *Profile* **and**
    $q :: ('a, 'v)$ *Profile* **and**
    $\pi :: 'v \Rightarrow 'v$
  assume
    bij: *bij* π **and**
    cond-b: $b\ (A,\ V,\ p)$ **and**
    prof-p: *profile* $V\ A\ p$ **and**
    renamed: *rename* $\pi\ (A,\ V,\ p) = (A',\ V',\ q)$
  have $\exists\ x.\ b'\ x\ (A,\ V,\ p)$
    using *cond-b general-cond-b*
    by *simp*
  then obtain $x :: 'b$ **where**
    $b'\ x\ (A,\ V,\ p)$
    by *blast*
  moreover have *consensus-anonymity* $(b'\ x)$
    using *all-cond-anon*
    by *simp*
  moreover have *profile* $V'\ A'\ q$
    using *prof-p renamed bij rename-sound*
    by *fastforce*
  ultimately have $b'\ x\ (A',\ V',\ q)$
    using *all-cond-anon bij prof-p renamed*
    unfolding *consensus-anonymity-def*
    by *auto*
  hence $\exists\ x.\ b'\ x\ (A',\ V',\ q)$
    by *metis*
  thus $b\ (A',\ V',\ q)$
    using *general-cond-b*
    by *simp*
qed

4.3.5 Theorems

Anonymity

lemma *nonempty-set-cons-anonymous*: *consensus-anonymity* $\textit{nonempty-set}_{\mathcal{C}}$
  unfolding *consensus-anonymity-def*
  by *simp*

lemma *nonempty-profile-cons-anonymous*: *consensus-anonymity* $\textit{nonempty-profile}_{\mathcal{C}}$

proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    q :: $('a, 'v)$ *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume
    bij: *bij* π **and**
    prof-p: *profile* V A p **and**
    renamed: *rename* π $(A, V, p) = (A', V', q)$ **and**
    not-empty-p: *nonempty-profile*$_{\mathcal{C}}$ (A, V, p)
  have *card* V = *card* V'
    using *renamed bij rename.simps Pair-inject*
        bij-betw-same-card bij-betw-subset top-greatest
    by (*metis* (*mono-tags*, *lifting*))
  thus *nonempty-profile*$_{\mathcal{C}}$ (A', V', q)
    using *not-empty-p length-0-conv renamed*
    unfolding *nonempty-profile*$_{\mathcal{C}}$*.simps*
    by *auto*
qed

lemma *equal-top-cons'-anonymous*:
  fixes a :: $'a$
  shows *consensus-anonymity* (*equal-top*$_{\mathcal{C}}$$'$ a)
proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    q :: $('a, 'v)$ *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume
    bij: *bij* π **and**
    prof-p: *profile* V A p **and**
    renamed: *rename* π $(A, V, p) = (A', V', q)$ **and**
    top-cons-a: *equal-top*$_{\mathcal{C}}$$'$ a (A, V, p)
  have $\forall\ v' \in V'.\ q\ v' = p$ ((*the-inv* π) v')
    using *renamed*
    by *auto*
  moreover have $\forall\ v' \in V'.$ (*the-inv* π) $v' \in V$
    using *bij renamed rename.simps bij-is-inj*
        f-the-inv-into-f-bij-betw inj-image-mem-iff
    by *fastforce*
  moreover have *winner*: $\forall\ v \in V.$ *above* $(p\ v)\ a = \{a\}$

using *top-cons-a*
by *simp*
ultimately have $\forall\ v' \in V'.$ *above* $(q\ v')\ a = \{a\}$
by *simp*
moreover have $a \in A$
using *top-cons-a*
by *simp*
ultimately show *equal-top*$_{\mathcal{C}}$*'* $a\ (A',\ V',\ q)$
using *renamed*
unfolding *equal-top*$_{\mathcal{C}}$*'.simps*
by *simp*
qed

lemma *eq-top-cons-anon*: *consensus-anonymity equal-top*$_{\mathcal{C}}$
using *equal-top-cons'-anonymous*
ex-anon-cons-imp-cons-anonymous[*of equal-top*$_{\mathcal{C}}$ *equal-top*$_{\mathcal{C}}$*'*]
by *fastforce*

lemma *eq-vote-cons'-anonymous*:
fixes r :: *'a Preference-Relation*
shows *consensus-anonymity* (*equal-vote*$_{\mathcal{C}}$*'* r)
proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
fix
A :: *'a set* **and**
A' :: *'a set* **and**
V :: *'v set* **and**
V' :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
π :: *'v* $\Rightarrow$ *'v*
assume
bij: *bij* π **and**
prof-p: *profile* $V\ A\ p$ **and**
renamed: *rename* $\pi\ (A,\ V,\ p) = (A',\ V',\ q)$ **and**
eq-vote: *equal-vote*$_{\mathcal{C}}$*'* $r\ (A,\ V,\ p)$
have $\forall\ v' \in V'.\ q\ v' = p$ ((*the-inv* π) v')
using *renamed*
by *auto*
moreover have $\forall\ v' \in V'.$ (*the-inv* π) $v' \in V$
using *bij renamed rename.simps bij-is-inj*
f-the-inv-into-f-bij-betw inj-image-mem-iff
by *fastforce*
moreover have *winner*: $\forall\ v \in V.\ p\ v = r$
using *eq-vote*
by *simp*
ultimately have $\forall\ v' \in V'.\ q\ v' = r$
by *simp*
thus *equal-vote*$_{\mathcal{C}}$*'* $r\ (A',\ V',\ q)$
unfolding *equal-vote*$_{\mathcal{C}}$*'.simps*

by *metis*
qed

lemma *eq-vote-cons-anonymous*: *consensus-anonymity equal-vote*$_{\mathcal{C}}$
unfolding *equal-vote*$_{\mathcal{C}}$*.simps*
using *eq-vote-cons′-anonymous ex-anon-cons-imp-cons-anonymous*
by *blast*

Neutrality

lemma *nonempty-set*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality valid-elections nonempty-set*$_{\mathcal{C}}$
proof (*simp*, *unfold valid-elections-def*, *safe*) **qed**

lemma *nonempty-profile*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality valid-elections nonempty-profile*$_{\mathcal{C}}$
proof (*simp*, *unfold valid-elections-def*, *safe*) **qed**

lemma *equal-vote*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality valid-elections equal-vote*$_{\mathcal{C}}$
proof (*simp*, *unfold valid-elections-def*, *clarsimp*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a,\ 'v)$ *Profile* **and**
π :: $'a \Rightarrow\ 'a$ **and**
r :: $'a$ *rel*
show $\forall\ v \in V.\ p\ v = r \Longrightarrow \exists\ r.\ \forall\ v \in V.\ \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p\ v\} = r$
by *simp*
assume *bij*: $\pi \in$ *carrier neutrality*$_{\mathcal{G}}$
hence *bij* π
unfolding *neutrality*$_{\mathcal{G}}$*-def*
using *rewrite-carrier*
by *blast*
hence $\forall\ a.$ *the-inv* $\pi\ (\pi\ a) = a$
using *bij-is-inj the-inv-f-f*
by *metis*
moreover have
$\forall\ v \in V.\ \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p\ v\} = r \Longrightarrow$
$\forall\ v \in V.\ \{(\textit{the-inv}\ \pi\ (\pi\ a), \textit{the-inv}\ \pi\ (\pi\ b)) \mid a\ b.\ (a, b) \in p\ v\} =$
$\{(\textit{the-inv}\ \pi\ a, \textit{the-inv}\ \pi\ b) \mid a\ b.\ (a, b) \in r\}$
by *fastforce*
ultimately have
$\forall\ v \in V.\ \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p\ v\} = r \Longrightarrow$
$\forall\ v \in V.\ \{(a, b) \mid a\ b.\ (a, b) \in p\ v\} =$
$\{(\textit{the-inv}\ \pi\ a, \textit{the-inv}\ \pi\ b) \mid a\ b.\ (a, b) \in r\}$
by *auto*
hence $\forall\ v \in V.\ \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p\ v\} = r \Longrightarrow$
$\forall\ v \in V.\ p\ v = \{(\textit{the-inv}\ \pi\ a, \textit{the-inv}\ \pi\ b) \mid a\ b.\ (a, b) \in r\}$
by *simp*
thus $\forall\ v \in V.\ \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p\ v\} = r \Longrightarrow \exists\ r.\ \forall\ v \in V.\ p\ v = r$
by *simp*

qed

lemma *strong-unanimity*$_{\mathcal{C}}$*-neutral*:
consensus-neutrality valid-elections strong-unanimity$_{\mathcal{C}}$
using *nonempty-set*$_{\mathcal{C}}$*-neutral equal-vote*$_{\mathcal{C}}$*-neutral nonempty-profile*$_{\mathcal{C}}$*-neutral*
cons-conjunction-invariant[*of*
$\{$*nonempty-set*$_{\mathcal{C}}$, *nonempty-profile*$_{\mathcal{C}}$, *equal-vote*$_{\mathcal{C}}\}$ *neutrality*$_{\mathcal{R}}$ *valid-elections*]
unfolding *strong-unanimity*$_{\mathcal{C}}$*.simps*
by *fastforce*

end

4.4 Electoral Module

theory *Electoral-Module*
imports *Social-Choice-Types/Property-Interpretations*
begin

Electoral modules are the principal component type of the composable modules voting framework, as they are a generalization of voting rules in the sense of social choice functions. These are only the types used for electoral modules. Further restrictions are encompassed by the electoral-module predicate.

An electoral module does not need to make final decisions for all alternatives, but can instead defer the decision for some or all of them to other modules. Hence, electoral modules partition the received (possibly empty) set of alternatives into elected, rejected and deferred alternatives. In particular, any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives. Just like a voting rule, an electoral module also receives a profile which holds the voters preferences, which, unlike a voting rule, consider only the (sub-)set of alternatives that the module receives.

4.4.1 Definition

An electoral module maps an election to a result. To enable currying, the Election type is not used here because that would require tuples.

type-synonym ($'a$, $'v$, $'r$) *Electoral-Module* = $'v$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ ($'a$, $'v$) *Profile* $\Rightarrow$ $'r$

fun $fun_{\mathcal{E}}$ :: $('v\ set \Rightarrow 'a\ set \Rightarrow ('a, 'v)\ Profile \Rightarrow 'r) \Rightarrow (('a, 'v)\ Election \Rightarrow 'r)$
where
$fun_{\mathcal{E}}\ m = (\lambda\ E.\ m\ (voters\text{-}\mathcal{E}\ E)\ (alternatives\text{-}\mathcal{E}\ E)\ (profile\text{-}\mathcal{E}\ E))$

The next three functions take an electoral module and turn it into a function only outputting the elect, reject, or defer set respectively.

abbreviation *elect* :: $('a, 'v, 'r\ Result)\ Electoral\text{-}Module \Rightarrow 'v\ set \Rightarrow 'a\ set$
$\Rightarrow ('a, 'v)\ Profile \Rightarrow 'r\ set$ **where**
$elect\ m\ V\ A\ p \equiv elect\text{-}r\ (m\ V\ A\ p)$

abbreviation *reject* :: $('a, 'v, 'r\ Result)\ Electoral\text{-}Module \Rightarrow 'v\ set \Rightarrow 'a\ set$
$\Rightarrow ('a, 'v)\ Profile \Rightarrow 'r\ set$ **where**
$reject\ m\ V\ A\ p \equiv reject\text{-}r\ (m\ V\ A\ p)$

abbreviation *defer* :: $('a, 'v, 'r\ Result)\ Electoral\text{-}Module \Rightarrow 'v\ set \Rightarrow 'a\ set$
$\Rightarrow ('a, 'v)\ Profile \Rightarrow 'r\ set$ **where**
$defer\ m\ V\ A\ p \equiv defer\text{-}r\ (m\ V\ A\ p)$

4.4.2 Auxiliary Definitions

Electoral modules partition a given set of alternatives A into a set of elected alternatives e, a set of rejected alternatives r, and a set of deferred alternatives d, using a profile. e, r, and d partition A. Electoral modules can be used as voting rules. They can also be composed in multiple structures to create more complex electoral modules.

definition (**in** *result*) *electoral-module* :: $('a, 'v, ('r\ Result))\ Electoral\text{-}Module \Rightarrow bool$ **where**
$electoral\text{-}module\ m \equiv \forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow well\text{-}formed\ A\ (m\ V\ A\ p)$

definition *only-voters-vote* :: $('a, 'v, ('r\ Result))\ Electoral\text{-}Module \Rightarrow bool$ **where**
$only\text{-}voters\text{-}vote\ m \equiv \forall\ A\ V\ p\ p'.\ (\forall\ v \in V.\ p\ v = p'\ v) \longrightarrow m\ V\ A\ p = m\ V\ A\ p'$

lemma (**in** *result*) *electoral-modI*:
fixes m :: $('a, 'v, ('r\ Result))\ Electoral\text{-}Module$
assumes $\bigwedge\ A\ V\ p.\ profile\ V\ A\ p \Longrightarrow well\text{-}formed\ A\ (m\ V\ A\ p)$
shows *electoral-module m*
unfolding *electoral-module-def*
using *assms*
by *simp*

4.4.3 Properties

We only require voting rules to behave a specific way on admissible elections, i.e., elections that are valid profiles (= votes are linear orders on the alternatives). Note that we do not assume finiteness of voter or alternative sets by default.

Anonymity

An electoral module is anonymous iff the result is invariant under renamings of voters, i.e., any permutation of the voter set that does not change the preferences leads to an identical result.

definition (**in** *result*) *anonymity* :: ('*a*, '*v*, ('*r Result*)) *Electoral-Module* ⇒ *bool*
where
  anonymity m ≡
   electoral-module m ∧
    (∀ *A V p* π::('*v* ⇒ '*v*).
     bij π ⟶ (*let* (*A*', *V*', *q*) = (*rename* π (*A*, *V*, *p*)) *in*
       finite-profile V A p ∧ *finite-profile V' A' q* ⟶ *m V A p* = *m V' A' q*))

Anonymity can alternatively be described as invariance under the voter permutation group acting on elections via the rename function.

fun *anonymity'* :: ('*a*, '*v*) *Election set* ⇒ ('*a*, '*v*, '*r*) *Electoral-Module* ⇒ *bool* **where**
  anonymity' X m = *satisfies* ($fun_{\mathcal{E}}$ *m*) (*Invariance* ($anonymity_{\mathcal{R}}$ *X*))

Homogeneity

A voting rule is homogeneous if copying an election does not change the result. For ordered voter types and finite elections, we use the notion of copying ballot lists to define copying an election. The more general definition of homogeneity for unordered voter types already implies anonymity.

fun (**in** *result*) *homogeneity* :: ('*a*, '*v*) *Election set* ⇒ ('*a*, '*v*, ('*r Result*)) *Electoral-Module*
    ⇒ *bool* **where**
  homogeneity X m = *satisfies* ($fun_{\mathcal{E}}$ *m*) (*Invariance* ($homogeneity_{\mathcal{R}}$ *X*))
— This does not require any specific behaviour on infinite voter sets ... Might make sense to extend the definition to that case somehow.

fun *homogeneity'* :: ('*a*, '*v*::*linorder*) *Election set* ⇒ ('*a*, '*v*, '*b Result*) *Electoral-Module*
    ⇒ *bool* **where**
  homogeneity' X m = *satisfies* ($fun_{\mathcal{E}}$ *m*) (*Invariance* ($homogeneity_{\mathcal{R}}'$ *X*))

lemma (**in** *result*) *hom-imp-anon*:
  fixes *X* :: ('*a*, '*v*) *Election set*
  assumes
   homogeneity X m **and**
   ∀ *E* ∈ *X*. *finite* (*voters-*$\mathcal{E}$ *E*)
  shows *anonymity' X m*
proof (*unfold anonymity'.simps satisfies.simps*, *standard*, *standard*, *standard*)
  fix
   E :: ('*a*, '*v*) *Election* **and**
   E' :: ('*a*, '*v*) *Election*
  assume *rel*: (*E*, *E'*) ∈ $anonymity_{\mathcal{R}}$ *X*
  hence *E* ∈ *X* ∧ *E'* ∈ *X*

unfolding *anonymity*$_\mathcal{R}$*.simps rel-induced-by-action.simps*
by *blast*
moreover with *this*
have *fin*: *finite* (*voters-*$\mathcal{E}$ *E*) $\wedge$ *finite* (*voters-*$\mathcal{E}$ *E*$'$)
using *assms*
by *simp*
moreover with *this*
have $\forall$ *r*. *vote-count r E* = *1* $*$ (*vote-count r E*$'$)
using *anon-rel-vote-count rel mult-1*
by *metis*
moreover with *fin*
have *alternatives-*$\mathcal{E}$ *E* = *alternatives-*$\mathcal{E}$ *E*$'$
using *anon-rel-vote-count rel*
by *blast*
ultimately show *fun*$_\mathcal{E}$ *m E* = *fun*$_\mathcal{E}$ *m E*$'$
using *assms zero-less-one*
unfolding *homogeneity.simps satisfies.simps homogeneity*$_\mathcal{R}$*.simps*
by *blast*
qed

Neutrality

Neutrality is equivariance under consistent renaming of candidates in the candidate set and election results.

fun (**in** *result-properties*) *neutrality* :: ($'a$, $'v$) *Election set*
$\Rightarrow$ ($'a$, $'v$, $'b$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
neutrality X m = *satisfies* (*fun*$_\mathcal{E}$ *m*)
(*equivar-ind-by-act* (*carrier neutrality*$_\mathcal{G}$) *X* (φ*-neutr X*) (*result-action* ψ*-neutr*))

4.4.4 Reversal Symmetry of Social Welfare Rules

A social welfare rule is reversal symmetric if reversing all voters' preferences reverses the result rankings as well.

definition *reversal-symmetry* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$, $'a$ *rel Result*) *Electoral-Module*
$\Rightarrow$ *bool* **where**
reversal-symmetry X m = *satisfies* (*fun*$_\mathcal{E}$ *m*)
(*equivar-ind-by-act* (*carrier reversal*$_\mathcal{G}$) *X* (φ*-rev X*) (*result-action* ψ*-rev*))

4.4.5 Social Choice Modules

The following results require electoral modules to return social choice results, i.e., sets of elected, rejected and deferred alternatives. In order to export code, we use the hack provided by Locale-Code.

"defers n" is true for all electoral modules that defer exactly n alternatives, whenever there are n or more alternatives.

definition *defers* :: *nat* $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
defers n m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
($\forall$ *A V p.* (*card A* $\geq$ *n* $\wedge$ *finite A* $\wedge$ *profile V A p*) $\longrightarrow$ *card* (*defer m V A p*) = *n*)

"rejects n" is true for all electoral modules that reject exactly n alternatives, whenever there are n or more alternatives.

definition *rejects* :: *nat* $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
rejects n m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
($\forall$ *A V p.* (*card A* $\geq$ *n* $\wedge$ *finite A* $\wedge$ *profile V A p*) $\longrightarrow$ *card* (*reject m V A p*) = *n*)

As opposed to "rejects", "eliminates" allows to stop rejecting if no alternatives were to remain.

definition *eliminates* :: *nat* $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
eliminates n m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
($\forall$ *A V p.* (*card A* > *n* $\wedge$ *profile V A p*) $\longrightarrow$ *card* (*reject m V A p*) = *n*)

"elects n" is true for all electoral modules that elect exactly n alternatives, whenever there are n or more alternatives.

definition *elects* :: *nat* $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
elects n m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
($\forall$ *A V p.* (*card A* $\geq$ *n* $\wedge$ *profile V A p*) $\longrightarrow$ *card* (*elect m V A p*) = *n*)

An electoral module is independent of an alternative a iff a's ranking does not influence the outcome.

definition *indep-of-alt* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ $'v$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$
$\Rightarrow$ *bool* **where**
indep-of-alt m V A a $\equiv$
$\mathcal{SCF}$*-result.electoral-module m*
$\wedge$ ($\forall$ *p q. equiv-prof-except-a V A p q a* $\longrightarrow$ *m V A p* = *m V A q*)

definition *unique-winner-if-profile-non-empty* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
$\Rightarrow$ *bool* **where**
unique-winner-if-profile-non-empty m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
($\forall$ *A V p.* (*A* $\neq$ {} $\wedge$ *V* $\neq$ {} $\wedge$ *profile V A p*) $\longrightarrow$
($\exists$ *a* $\in$ *A. m V A p* = ({*a*}, *A* − {*a*}, {})))

4.4.6 Equivalence Definitions

definition *prof-contains-result* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ $'v$ *set* $\Rightarrow$ $'a$ *set*

$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool*
where
prof-contains-result $m\ V\ A\ p\ q\ a \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
profile $V\ A\ p \wedge$ *profile* $V\ A\ q \wedge a \in A \wedge$
$(a \in$ *elect* $m\ V\ A\ p \longrightarrow a \in$ *elect* $m\ V\ A\ q) \wedge$
$(a \in$ *reject* $m\ V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ q) \wedge$
$(a \in$ *defer* $m\ V\ A\ p \longrightarrow a \in$ *defer* $m\ V\ A\ q)$

definition *prof-leq-result* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
prof-leq-result $m\ V\ A\ p\ q\ a \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
profile $V\ A\ p \wedge$ *profile* $V\ A\ q \wedge a \in A \wedge$
$(a \in$ *reject* $m\ V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ q) \wedge$
$(a \in$ *defer* $m\ V\ A\ p \longrightarrow a \notin$ *elect* $m\ V\ A\ q)$

definition *prof-geq-result* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
prof-geq-result $m\ V\ A\ p\ q\ a \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
profile $V\ A\ p \wedge$ *profile* $V\ A\ q \wedge a \in A \wedge$
$(a \in$ *elect* $m\ V\ A\ p \longrightarrow a \in$ *elect* $m\ V\ A\ q) \wedge$
$(a \in$ *defer* $m\ V\ A\ p \longrightarrow a \notin$ *reject* $m\ V\ A\ q)$

definition *mod-contains-result* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
$\Rightarrow ('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
mod-contains-result $m\ n\ V\ A\ p\ a \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$\mathcal{SCF}$*-result.electoral-module* $n \wedge$
profile $V\ A\ p \wedge a \in A \wedge$
$(a \in$ *elect* $m\ V\ A\ p \longrightarrow a \in$ *elect* $n\ V\ A\ p) \wedge$
$(a \in$ *reject* $m\ V\ A\ p \longrightarrow a \in$ *reject* $n\ V\ A\ p) \wedge$
$(a \in$ *defer* $m\ V\ A\ p \longrightarrow a \in$ *defer* $n\ V\ A\ p)$

definition *mod-contains-result-sym* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
$\Rightarrow ('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
mod-contains-result-sym $m\ n\ V\ A\ p\ a \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$\mathcal{SCF}$*-result.electoral-module* $n \wedge$
profile $V\ A\ p \wedge a \in A \wedge$
$(a \in$ *elect* $m\ V\ A\ p \longleftrightarrow a \in$ *elect* $n\ V\ A\ p) \wedge$
$(a \in$ *reject* $m\ V\ A\ p \longleftrightarrow a \in$ *reject* $n\ V\ A\ p) \wedge$
$(a \in$ *defer* $m\ V\ A\ p \longleftrightarrow a \in$ *defer* $n\ V\ A\ p)$

4.4.7 Auxiliary Lemmas

lemma *elect-rej-def-combination*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    V :: $'v$ *set* **and**
    A :: $'a$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    e :: $'a$ *set* **and**
    r :: $'a$ *set* **and**
    d :: $'a$ *set*
  assumes
    elect m V A p = e **and**
    reject m V A p = r **and**
    defer m V A p = d
  shows m V A p = (e, r, d)
  using *assms*
  by *auto*

lemma *par-comp-result-sound*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    p :: ($'a$, $'v$) *Profile*
  assumes
    $\mathcal{SCF}$-*result.electoral-module* m **and**
    profile V A p
  shows *well-formed*-$\mathcal{SCF}$ A (m V A p)
  using *assms*
  unfolding $\mathcal{SCF}$-*result.electoral-module-def*
  by *simp*

lemma *result-presv-alts*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile*
  assumes
    $\mathcal{SCF}$-*result.electoral-module* m **and**
    profile V A p
  shows (*elect* m V A p) $\cup$ (*reject* m V A p) $\cup$ (*defer* m V A p) = A
proof (*safe*)
  fix a :: $'a$
  assume $a \in$ *elect* m V A p
  moreover have
    $\forall$ p'. *set-equals-partition* A p' $\longrightarrow$
      ($\exists$ E R D. p' = (E, R, D) $\wedge$ $E \cup R \cup D = A$)
    by *simp*
  moreover have *set-equals-partition* A (m V A p)

    using *assms*
    unfolding $\mathcal{SCF}$*-result.electoral-module-def*
    by *simp*
  ultimately show $a \in A$
    using *UnI1 fstI*
    by (*metis* (*no-types*))
next
  fix $a :: 'a$
  assume $a \in$ *reject* $m\ V\ A\ p$
  moreover have
    $\forall\ p'.$ *set-equals-partition* $A\ p' \longrightarrow$
        $(\exists\ E\ R\ D.\ p' = (E, R, D) \land E \cup R \cup D = A)$
    by *simp*
  moreover have *set-equals-partition* $A\ (m\ V\ A\ p)$
    using *assms*
    unfolding $\mathcal{SCF}$*-result.electoral-module-def*
    by *simp*
  ultimately show $a \in A$
    using *UnI1 fstI sndI subsetD sup-ge2*
    by *metis*
next
  fix $a :: 'a$
  assume $a \in$ *defer* $m\ V\ A\ p$
  moreover have
    $\forall\ p'.$ *set-equals-partition* $A\ p' \longrightarrow$
        $(\exists\ E\ R\ D.\ p' = (E, R, D) \land E \cup R \cup D = A)$
    by *simp*
  moreover have *set-equals-partition* $A\ (m\ V\ A\ p)$
    using *assms*
    unfolding $\mathcal{SCF}$*-result.electoral-module-def*
    by *simp*
  ultimately show $a \in A$
    using *sndI subsetD sup-ge2*
    by *metis*
next
  fix $a :: 'a$
  assume
    $a \in A$ **and**
    $a \notin$ *defer* $m\ V\ A\ p$ **and**
    $a \notin$ *reject* $m\ V\ A\ p$
  moreover have
    $\forall\ p'.$ *set-equals-partition* $A\ p' \longrightarrow$
        $(\exists\ E\ R\ D.\ p' = (E, R, D) \land E \cup R \cup D = A)$
    by *simp*
  moreover have *set-equals-partition* $A\ (m\ V\ A\ p)$
    using *assms*
    unfolding $\mathcal{SCF}$*-result.electoral-module-def*
    by *simp*
  ultimately show $a \in$ *elect* $m\ V\ A\ p$

```
    using fst-conv snd-conv Un-iff
    by metis
qed

lemma result-disj:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    p :: ('a, 'v) Profile and
    V :: 'v set
  assumes
    SCF-result.electoral-module m and
    profile V A p
  shows
    (elect m V A p) ∩ (reject m V A p) = {} ∧
        (elect m V A p) ∩ (defer m V A p) = {} ∧
        (reject m V A p) ∩ (defer m V A p) = {}
proof (safe)
  fix a :: 'a
  assume
    a ∈ elect m V A p and
    a ∈ reject m V A p
  moreover have well-formed-SCF A (m V A p)
    using assms
    unfolding SCF-result.electoral-module-def
    by metis
  ultimately show a ∈ {}
    using prod.exhaust-sel DiffE UnCI result-imp-rej
    by (metis (no-types))
next
  fix a :: 'a
  assume
    elect-a: a ∈ elect m V A p and
    defer-a: a ∈ defer m V A p
  have disj:
    ∀ p'. disjoint3 p' ⟶
      (∃ B C D. p' = (B, C, D) ∧ B ∩ C = {} ∧ B ∩ D = {} ∧ C ∩ D = {})
    by simp
  have well-formed-SCF A (m V A p)
    using assms
    unfolding SCF-result.electoral-module-def
    by metis
  hence disjoint3 (m V A p)
    by simp
  then obtain
    e :: 'a Result ⇒ 'a set and
    r :: 'a Result ⇒ 'a set and
    d :: 'a Result ⇒ 'a set
    where
```

$m\ V\ A\ p =$
$(e\ (m\ V\ A\ p),\ r\ (m\ V\ A\ p),\ d\ (m\ V\ A\ p)) \wedge$
$e\ (m\ V\ A\ p) \cap r\ (m\ V\ A\ p) = \{\} \wedge$
$e\ (m\ V\ A\ p) \cap d\ (m\ V\ A\ p) = \{\} \wedge$
$r\ (m\ V\ A\ p) \cap d\ (m\ V\ A\ p) = \{\}$
using *elect-a defer-a disj*
by *metis*
hence $((\mathit{elect}\ m\ V\ A\ p) \cap (\mathit{reject}\ m\ V\ A\ p) = \{\}) \wedge$
$((\mathit{elect}\ m\ V\ A\ p) \cap (\mathit{defer}\ m\ V\ A\ p) = \{\}) \wedge$
$((\mathit{reject}\ m\ V\ A\ p) \cap (\mathit{defer}\ m\ V\ A\ p) = \{\})$
using *eq-snd-iff fstI*
by *metis*
thus $a \in \{\}$
using *elect-a defer-a disjoint-iff-not-equal*
by (*metis* (*no-types*))
next
fix $a :: 'a$
assume
$a \in \mathit{reject}\ m\ V\ A\ p$ **and**
$a \in \mathit{defer}\ m\ V\ A\ p$
moreover have *well-formed-$\mathcal{SCF}$* $A\ (m\ V\ A\ p)$
using *assms*
unfolding *$\mathcal{SCF}$-result.electoral-module-def*
by *simp*
ultimately show $a \in \{\}$
using *prod.exhaust-sel DiffE UnCI result-imp-rej*
by (*metis* (*no-types*))
qed

lemma *elect-in-alts*:
fixes
$m :: ('a, 'v, 'a\ \mathit{Result})\ \mathit{Electoral\text{-}Module}$ **and**
$A :: 'a\ \mathit{set}$ **and**
$p :: ('a, 'v)\ \mathit{Profile}$
assumes
$\mathcal{SCF}$-result.electoral-module m **and**
profile $V\ A\ p$
shows $\mathit{elect}\ m\ V\ A\ p \subseteq A$
using *le-supI1 assms result-presv-alts sup-ge1*
by *metis*

lemma *reject-in-alts*:
fixes
$m :: ('a, 'v, 'a\ \mathit{Result})\ \mathit{Electoral\text{-}Module}$ **and**
$A :: 'a\ \mathit{set}$ **and**
$V :: 'v\ \mathit{set}$ **and**
$p :: ('a, 'v)\ \mathit{Profile}$
assumes
$\mathcal{SCF}$-result.electoral-module m **and**

profile V A p
shows *reject m V A p* $\subseteq$ *A*
using *le-supI1 assms result-presv-alts sup-ge2*
by *fastforce*

lemma *defer-in-alts*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p
shows *defer m V A p* $\subseteq$ *A*
using *assms result-presv-alts*
by *fastforce*

lemma *def-presv-prof*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p
shows *let new-A* = *defer m V A p in profile V new-A* (*limit-profile new-A p*)
using *defer-in-alts limit-profile-sound assms*
by *metis*

An electoral module can never reject, defer or elect more than |A| alternatives.

lemma *upper-card-bounds-for-result*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p **and**
finite A
shows
upper-card-bound-for-elect: *card* (*elect m V A p*) $\leq$ *card A* **and**
upper-card-bound-for-reject: *card* (*reject m V A p*) $\leq$ *card A* **and**
upper-card-bound-for-defer: *card* (*defer m V A p*) $\leq$ *card A*
proof −
show *card* (*elect m V A p*) $\leq$ *card A*
using *assms card-mono elect-in-alts*

by *metis*
next
show *card* (*reject m V A p*) $\leq$ *card A*
using *assms card-mono reject-in-alts*
by *metis*
next
show *card* (*defer m V A p*) $\leq$ *card A*
using *assms card-mono defer-in-alts*
by *metis*
qed

lemma *reject-not-elec-or-def*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p
shows *reject m V A p* = *A* − (*elect m V A p*) − (*defer m V A p*)
proof −
have *well-formed-*$\mathcal{SCF}$ *A* (*m V A p*)
using *assms*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *simp*
hence (*elect m V A p*) ∪ (*reject m V A p*) ∪ (*defer m V A p*) = *A*
using *assms result-presv-alts*
by *simp*
moreover have
(*elect m V A p*) ∩ (*reject m V A p*) = {} ∧ (*reject m V A p*) ∩ (*defer m V A p*) = {}
using *assms result-disj*
by *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *elec-and-def-not-rej*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p
shows *elect m V A p* ∪ *defer m V A p* = *A* − (*reject m V A p*)
proof −

have (*elect* m V A p) ∪ (*reject* m V A p) ∪ (*defer* m V A p) = A
using *assms result-presv-alts*
by *blast*
moreover have
(*elect* m V A p) ∩ (*reject* m V A p) = {} ∧ (*reject* m V A p) ∩ (*defer* m V A p) = {}
using *assms result-disj*
by *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *defer-not-elec-or-rej*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
profile V A p
shows *defer* m V A p = A − (*elect* m V A p) − (*reject* m V A p)
proof −
have *well-formed-*$\mathcal{SCF}$ A (m V A p)
using *assms*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *simp*
hence (*elect* m V A p) ∪ (*reject* m V A p) ∪ (*defer* m V A p) = A
using *assms result-presv-alts*
by *simp*
moreover have
(*elect* m V A p) ∩ (*defer* m V A p) = {} ∧ (*reject* m V A p) ∩ (*defer* m V A p) = {}
using *assms result-disj*
by *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *electoral-mod-defer-elem*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
a :: '*a*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
profile V A p **and**
$a \in A$ **and**

$a \notin$ *elect* $m\ V\ A\ p$ **and**
$a \notin$ *reject* $m\ V\ A\ p$
shows $a \in$ *defer* $m\ V\ A\ p$
using *DiffI assms reject-not-elec-or-def*
by *metis*

lemma *mod-contains-result-comm*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
a :: $'a$
assumes *mod-contains-result* $m\ n\ V\ A\ p\ a$
shows *mod-contains-result* $n\ m\ V\ A\ p\ a$
proof (*unfold mod-contains-result-def*, *safe*)
from *assms*
show $\mathcal{SCF}$*-result.electoral-module* n
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show $\mathcal{SCF}$*-result.electoral-module* m
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show *profile* $V\ A\ p$
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show $a \in A$
unfolding *mod-contains-result-def*
by *safe*
next
assume $a \in$ *elect* $n\ V\ A\ p$
thus $a \in$ *elect* $m\ V\ A\ p$
using *IntI assms electoral-mod-defer-elem empty-iff result-disj*
unfolding *mod-contains-result-def*
by (*metis* (*mono-tags*, *lifting*))
next
assume $a \in$ *reject* $n\ V\ A\ p$
thus $a \in$ *reject* $m\ V\ A\ p$
using *IntI assms electoral-mod-defer-elem empty-iff result-disj*
unfolding *mod-contains-result-def*
by (*metis* (*mono-tags*, *lifting*))
next

assume $a \in$ *defer n V A p*
thus $a \in$ *defer m V A p*
using *IntI assms electoral-mod-defer-elem empty-iff result-disj*
unfolding *mod-contains-result-def*
by (*metis* (*mono-tags*, *lifting*))
qed

lemma *not-rej-imp-elec-or-def*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
a :: '*a*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
profile V A p **and**
$a \in A$ **and**
$a \notin$ *reject m V A p*
shows $a \in$ *elect m V A p* $\vee$ $a \in$ *defer m V A p*
using *assms electoral-mod-defer-elem*
by *metis*

lemma *single-elim-imp-red-def-set*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
eliminates 1 m **and**
card A $>$ *1* **and**
profile V A p
shows *defer m V A p* $\subset$ *A*
using *Diff-eq-empty-iff Diff-subset card-eq-0-iff defer-in-alts eliminates-def eq-iff not-one-le-zero psubsetI reject-not-elec-or-def assms*
by (*metis* (*no-types*, *lifting*))

lemma *eq-alts-in-profs-imp-eq-results*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
q :: ('*a*, '*v*) *Profile*
assumes
eq: $\forall$ $a \in A$. *prof-contains-result m V A p q a* **and**
mod-m: $\mathcal{SCF}$*-result.electoral-module m* **and**
prof-p: *profile V A p* **and**

```
    prof-q: profile V A q
  shows m V A p = m V A q
proof -
  have elected-in-A: elect m V A q ⊆ A
    using elect-in-alts mod-m prof-q
    by metis
  have rejected-in-A: reject m V A q ⊆ A
    using reject-in-alts mod-m prof-q
    by metis
  have deferred-in-A: defer m V A q ⊆ A
    using defer-in-alts mod-m prof-q
    by metis
  have ∀ a ∈ elect m V A p. a ∈ elect m V A q
    using elect-in-alts eq prof-contains-result-def mod-m prof-p in-mono
    by metis
  moreover have ∀ a ∈ elect m V A q. a ∈ elect m V A p
  proof
    fix a :: 'a
    assume q-elect-a: a ∈ elect m V A q
    hence a ∈ A
      using elected-in-A
      by blast
    moreover have a ∉ defer m V A q
      using q-elect-a prof-q mod-m result-disj
      by blast
    moreover have a ∉ reject m V A q
      using q-elect-a disjoint-iff-not-equal prof-q mod-m result-disj
      by metis
    ultimately show a ∈ elect m V A p
      using electoral-mod-defer-elem eq prof-contains-result-def
      by fastforce
  qed
  moreover have ∀ a ∈ reject m V A p. a ∈ reject m V A q
    using reject-in-alts eq prof-contains-result-def mod-m prof-p
    by fastforce
  moreover have ∀ a ∈ reject m V A q. a ∈ reject m V A p
  proof
    fix a :: 'a
    assume q-rejects-a: a ∈ reject m V A q
    hence a ∈ A
      using rejected-in-A
      by blast
    moreover have a-not-deferred-q: a ∉ defer m V A q
      using q-rejects-a prof-q mod-m result-disj
      by blast
    moreover have a-not-elected-q: a ∉ elect m V A q
      using q-rejects-a disjoint-iff-not-equal prof-q mod-m result-disj
      by metis
    ultimately show a ∈ reject m V A p
```

using *electoral-mod-defer-elem eq prof-contains-result-def*
by *fastforce*
qed
moreover have $\forall\ a \in$ *defer m V A p*. $a \in$ *defer m V A q*
using *defer-in-alts eq prof-contains-result-def mod-m prof-p*
by *fastforce*
moreover have $\forall\ a \in$ *defer m V A q*. $a \in$ *defer m V A p*
proof
fix a :: $'a$
assume *q-defers-a*: $a \in$ *defer m V A q*
moreover have $a \in A$
using *q-defers-a deferred-in-A*
by *blast*
moreover have $a \notin$ *elect m V A q*
using *q-defers-a prof-q mod-m result-disj*
by *blast*
moreover have $a \notin$ *reject m V A q*
using *q-defers-a prof-q disjoint-iff-not-equal mod-m result-disj*
by *metis*
ultimately show $a \in$ *defer m V A p*
using *electoral-mod-defer-elem eq prof-contains-result-def*
by *fastforce*
qed
ultimately show *?thesis*
using *prod.collapse subsetI subset-antisym*
by (*metis* (*no-types*))
qed

lemma *eq-def-and-elect-imp-eq*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
q :: ($'a$, $'v$) *Profile*
assumes
mod-m: $\mathcal{SCF}$*-result.electoral-module m* **and**
mod-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
fin-p: *profile V A p* **and**
fin-q: *profile V A q* **and**
elec-eq: *elect m V A p* = *elect n V A q* **and**
def-eq: *defer m V A p* = *defer n V A q*
shows *m V A p* = *n V A q*
proof –
have *reject m V A p* = *A* − ((*elect m V A p*) ∪ (*defer m V A p*))
using *mod-m fin-p elect-rej-def-combination result-imp-rej*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *metis*

moreover have *reject n V A q* = *A* − ((*elect n V A q*) ∪ (*defer n V A q*))
using *mod-n fin-q elect-rej-def-combination result-imp-rej*
unfolding *$\mathcal{SCF}$-result.electoral-module-def*
by *metis*
ultimately show *?thesis*
using *elec-eq def-eq prod-eqI*
by *metis*
qed

4.4.8 Non-Blocking

An electoral module is non-blocking iff this module never rejects all alternatives.

definition *non-blocking* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ *bool* **where**
non-blocking m ≡
$\mathcal{SCF}$-result.electoral-module m ∧
(∀ *A V p*. ((*A* ≠ {} ∧ *finite A* ∧ *profile V A p*) ⟶ *reject m V A p* ≠ *A*))

4.4.9 Electing

An electoral module is electing iff it always elects at least one alternative.

definition *electing* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ *bool* **where**
electing m ≡
$\mathcal{SCF}$-result.electoral-module m ∧
(∀ *A V p*. (*A* ≠ {} ∧ *finite A* ∧ *profile V A p*) ⟶ *elect m V A p* ≠ {})

lemma *electing-for-only-alt*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
one-alt: *card A* = *1* **and**
electing: *electing m* **and**
prof: *profile V A p*
shows *elect m V A p* = *A*
proof (*safe*)
fix *a* :: '*a*
assume *elect-a*: *a* ∈ *elect m V A p*
have *$\mathcal{SCF}$-result.electoral-module m* ⟶ *elect m V A p* ⊆ *A*
using *prof elect-in-alts*
by *blast*
hence *elect m V A p* ⊆ *A*
using *electing*
unfolding *electing-def*
by *metis*
thus *a* ∈ *A*

using *elect-a*
by *blast*
next
fix $a :: 'a$
assume $a \in A$
thus $a \in$ *elect m V A p*
using *electing prof one-alt One-nat-def Suc-leI card-seteq card-gt-0-iff elect-in-alts infinite-super lessI*
unfolding *electing-def*
by *metis*
qed

theorem *electing-imp-non-blocking*:
fixes $m :: ('a, 'v, 'a\ Result)$ *Electoral-Module*
assumes *electing m*
shows *non-blocking m*
proof (*unfold non-blocking-def*, *safe*)
from *assms*
show $\mathcal{SCF}$*-result.electoral-module m*
unfolding *electing-def*
by *simp*
next
fix
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)$ *Profile* **and**
$a :: 'a$
assume
profile V A p **and**
finite A **and**
reject m V A p $= A$ **and**
$a \in A$
moreover have
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
$(\forall\ A\ V\ q.\ A \neq \{\} \wedge \mathit{finite}\ A \wedge \mathit{profile}\ V\ A\ q \longrightarrow \mathit{elect}\ m\ V\ A\ q \neq \{\})$
using *assms*
unfolding *electing-def*
by *metis*
ultimately show $a \in \{\}$
using *Diff-cancel Un-empty elec-and-def-not-rej*
by *metis*
qed

4.4.10 Properties

An electoral module is non-electing iff it never elects an alternative.

definition *non-electing* $:: ('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ *bool* **where**
non-electing m $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$ $(\forall\ A\ V\ p.\ \mathit{profile}\ V\ A\ p \longrightarrow \mathit{elect}\ m\ V\ A\ p =$

{})

lemma *single-rej-decr-def-card*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
rejecting: *rejects 1 m* **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile V A p*
shows *card* (*defer m V A p*) = *card A* − *1*
proof −
have *no-elect*:
$\mathcal{SCF}$*-result.electoral-module m* ∧ (∀ *V A q*. *profile V A q* ⟶ *elect m V A q* =
{})
using *non-electing*
unfolding *non-electing-def*
by (*metis* (*no-types*))
hence *reject m V A p* ⊆ *A*
using *f-prof reject-in-alts*
by *metis*
moreover have *A* = *A* − *elect m V A p*
using *no-elect f-prof*
by *blast*
ultimately show *?thesis*
using *f-prof no-elect rejecting card-Diff-subset card-gt-0-iff*
defer-not-elec-or-rej less-one order-less-imp-le Suc-leI
bot.extremum-unique card.empty diff-is-0-eq' One-nat-def
unfolding *rejects-def*
by *metis*
qed

lemma *single-elim-decr-def-card-2*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
eliminating: *eliminates 1 m* **and**
non-electing: *non-electing m* **and**
not-empty: *card A* > *1* **and**
prof-p: *profile V A p*
shows *card* (*defer m V A p*) = *card A* − *1*
proof −
have *no-elect*:
$\mathcal{SCF}$*-result.electoral-module m* ∧ (∀ *A V q*. *profile V A q* ⟶ *elect m V A q* =

{})
    using *non-electing*
    unfolding *non-electing-def*
    by (*metis* (*no-types*))
  hence *reject m V A p* $\subseteq$ *A*
    using *prof-p reject-in-alts*
    by *metis*
  moreover have $A = A -$ *elect m V A p*
    using *no-elect prof-p*
    by *blast*
  ultimately show *?thesis*
    using *prof-p not-empty no-elect eliminating card-ge-0-finite*
        card-Diff-subset defer-not-elec-or-rej zero-less-one
    unfolding *eliminates-def*
    by (*metis* (*no-types*, *lifting*))
qed

An electoral module is defer-deciding iff this module chooses exactly 1 alternative to defer and rejects any other alternative. Note that 'rejects n-1 m' can be omitted due to the well-formedness property.

definition *defer-deciding* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
  defer-deciding m $\equiv$
    $\mathcal{SCF}$*-result.electoral-module m* $\wedge$ *non-electing m* $\wedge$ *defers 1 m*

An electoral module decrements iff this module rejects at least one alternative whenever possible (|A| > 1).

definition *decrementing* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
  decrementing m $\equiv$
    $\mathcal{SCF}$*-result.electoral-module m* $\wedge$
      ($\forall$ *A V p. profile V A p* $\wedge$ *card A* $>$ *1* $\longrightarrow$ *card* (*reject m V A p*) $\geq$ *1*)

definition *defer-condorcet-consistency* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
  defer-condorcet-consistency m $\equiv$
    $\mathcal{SCF}$*-result.electoral-module m* $\wedge$
    ($\forall$ *A V p a. condorcet-winner V A p a* $\longrightarrow$
      (*m V A p* = ({}, *A* $-$ (*defer m V A p*), {*d* $\in$ *A. condorcet-winner V A p d*})))

definition *condorcet-compatibility* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
  condorcet-compatibility m $\equiv$
    $\mathcal{SCF}$*-result.electoral-module m* $\wedge$
    ($\forall$ *A V p a. condorcet-winner V A p a* $\longrightarrow$
      (*a* $\notin$ *reject m V A p* $\wedge$
        ($\forall$ *b.* $\neg$ *condorcet-winner V A p b* $\longrightarrow$ *b* $\notin$ *elect m V A p*) $\wedge$
          (*a* $\in$ *elect m V A p* $\longrightarrow$
            ($\forall$ *b* $\in$ *A.* $\neg$ *condorcet-winner V A p b* $\longrightarrow$ *b* $\in$ *reject m V A p*))))

An electoral module is defer-monotone iff, when a deferred alternative is lifted, this alternative remains deferred.

definition *defer-monotonicity* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
defer-monotonicity $m \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$(\forall\ A\ V\ p\ q\ a.$
$(a \in \textit{defer}\ m\ V\ A\ p \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow a \in \textit{defer}\ m\ V\ A\ q)$

An electoral module is defer-lift-invariant iff lifting a deferred alternative does not affect the outcome.

definition *defer-lift-invariance* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
defer-lift-invariance $m \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$(\forall\ A\ V\ p\ q\ a.\ (a \in (\textit{defer}\ m\ V\ A\ p) \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow m\ V\ A\ p = m\ V$
$A\ q)$

Two electoral modules are disjoint-compatible if they only make decisions over disjoint sets of alternatives. Electoral modules reject alternatives for which they make no decision.

definition *disjoint-compatibility* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$
($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
disjoint-compatibility $m\ n \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge \mathcal{SCF}$*-result.electoral-module* $n \wedge$
$(\forall\ V.$
$(\forall\ A.$
$(\exists\ B \subseteq A.$
$(\forall\ a \in B.\ \textit{indep-of-alt}\ m\ V\ A\ a \wedge$
$(\forall\ p.\ \textit{profile}\ V\ A\ p \longrightarrow a \in \textit{reject}\ m\ V\ A\ p)) \wedge$
$(\forall\ a \in A - B.\ \textit{indep-of-alt}\ n\ V\ A\ a \wedge$
$(\forall\ p.\ \textit{profile}\ V\ A\ p \longrightarrow a \in \textit{reject}\ n\ V\ A\ p)))))$

Lifting an elected alternative a from an invariant-monotone electoral module either does not change the elect set, or makes a the only elected alternative.

definition *invariant-monotonicity* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool*
where
invariant-monotonicity $m \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$(\forall\ A\ V\ p\ q\ a.\ (a \in \textit{elect}\ m\ V\ A\ p \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow$
$(\textit{elect}\ m\ V\ A\ q = \textit{elect}\ m\ V\ A\ p \vee \textit{elect}\ m\ V\ A\ q = \{a\}))$

Lifting a deferred alternative a from a defer-invariant-monotone electoral module either does not change the defer set, or makes a the only deferred alternative.

definition *defer-invariant-monotonicity* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$
bool **where**
defer-invariant-monotonicity $m \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$ *non-electing* $m \wedge$
$(\forall\ A\ V\ p\ q\ a.\ (a \in \textit{defer}\ m\ V\ A\ p \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow$
$(\textit{defer}\ m\ V\ A\ q = \textit{defer}\ m\ V\ A\ p \vee \textit{defer}\ m\ V\ A\ q = \{a\}))$

4.4.11 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    a :: *'a*
  assumes
    ccomp: *condorcet-compatibility m* **and**
    dd: *defer-deciding m* **and**
    winner: *condorcet-winner V A p a*
  shows *defer m V A p* = {*a*}
proof (*rule ccontr*)
  assume *not-w*: *defer m V A p* ≠ {*a*}
  have *def-one*: *defers 1 m*
    using *dd*
    unfolding *defer-deciding-def*
    by *metis*
  hence *c-win*: *finite-profile V A p* ∧ *a* ∈ *A* ∧ (∀ *b* ∈ *A* − {*a*}. *wins V a p b*)
    using *winner*
    by *auto*
  hence *card* (*defer m V A p*) = *1*
    using *Suc-leI card-gt-0-iff def-one equals0D*
    unfolding *One-nat-def defers-def*
    by *metis*
  hence ∃ *b* ∈ *A*. *defer m V A p* = {*b*}
    using *card-1-singletonE dd defer-in-alts insert-subset c-win*
    unfolding *defer-deciding-def*
    by *metis*
  hence ∃ *b* ∈ *A*. *b* ≠ *a* ∧ *defer m V A p* = {*b*}
    using *not-w*
    by *metis*
  hence *not-in-defer*: *a* ∉ *defer m V A p*
    by *auto*
  have *non-electing m*
    using *dd*
    unfolding *defer-deciding-def*
    by *simp*
  hence *a* ∉ *elect m V A p*
    using *c-win equals0D*
    unfolding *non-electing-def*
    by *simp*
  hence *a* ∈ *reject m V A p*
    using *not-in-defer ccomp c-win electoral-mod-defer-elem*
    unfolding *condorcet-compatibility-def*
    by *metis*
  moreover have *a* ∉ *reject m V A p*
    using *ccomp c-win winner*

    unfolding *condorcet-compatibility-def*
    by *simp*
  ultimately show *False*
    by *simp*
qed

theorem *ccomp-and-dd-imp-dcc*[*simp*]:
  fixes m :: ('a, 'v, 'a *Result*) *Electoral-Module*
  assumes
    ccomp: *condorcet-compatibility* m **and**
    dd: *defer-deciding* m
  shows *defer-condorcet-consistency* m
proof (*unfold defer-condorcet-consistency-def*, *simp*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* m
    using *dd*
    unfolding *defer-deciding-def*
    by *metis*
next
  fix
    A :: 'a *set* **and**
    V :: 'v *set* **and**
    p :: ('a, 'v) *Profile* **and**
    a :: 'a
  assume
    prof-A: *profile* V A p **and**
    a-in-A: $a \in A$ **and**
    fin-A: *finite* A **and**
    fin-V: *finite* V **and**
    c-winner:
      $\forall\ x \in A - \{a\}$.
          (*finite* $V \longrightarrow$ *card* $\{v \in V.\ (a, x) \in p\ v\}$ < *card* $\{v \in V.\ (x, a) \in p\ v\}$)
$\wedge$ *finite* V
  hence *winner*: *condorcet-winner* V A p a
    by *simp*
  hence *elect-empty*: *elect* m V A p = {}
    using *dd*
    unfolding *defer-deciding-def non-electing-def*
    by *simp*
  have *cond-winner-a*: $\{a\} = \{c \in A.$ *condorcet-winner* V A p $c\}$
    using *cond-winner-unique winner*
    by *metis*
  have *defer-a*: *defer* m V A p = $\{a\}$
    using *winner dd ccomp ccomp-and-dd-imp-def-only-winner winner*
    by *simp*
  hence *reject* m V A p = A − *defer* m V A p
    using *Diff-empty dd reject-not-elec-or-def winner elect-empty*
    unfolding *defer-deciding-def*
    by *fastforce*
  hence m V A p = ({}, A − *defer* m V A p, $\{a\}$)

using *elect-empty defer-a elect-rej-def-combination*
by *metis*
hence $m\ V\ A\ p = (\{\}, A - \mathit{defer}\ m\ V\ A\ p, \{c \in A.\ \mathit{condorcet\text{-}winner}\ V\ A\ p\ c\})$
using *cond-winner-a*
by *simp*
thus $m\ V\ A\ p =$
$(\{\}, A - \mathit{defer}\ m\ V\ A\ p,$
$\{d \in A.\ \forall\ x \in A - \{d\}.\ \mathit{card}\ \{v \in V.\ (d, x) \in p\ v\} < \mathit{card}\ \{v \in V.\ (x, d) \in p\ v\}\})$
using *fin-A fin-V prof-A winner Collect-cong*
by *simp*
qed

If m and n are disjoint compatible, so are n and m.

theorem *disj-compat-comm*[*simp*]:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module*
assumes *disjoint-compatibility m n*
shows *disjoint-compatibility n m*
proof (*unfold disjoint-compatibility-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module m*
using *assms*
unfolding *disjoint-compatibility-def*
by *simp*
next
show $\mathcal{SCF}$*-result.electoral-module n*
using *assms*
unfolding *disjoint-compatibility-def*
by *simp*
next
fix
A :: 'a *set* **and**
V :: 'v *set*
obtain B **where**
$B \subseteq A \land$
$(\forall\ a \in B.$
$\mathit{indep\text{-}of\text{-}alt}\ m\ V\ A\ a \land (\forall\ p.\ \mathit{profile}\ V\ A\ p \longrightarrow a \in \mathit{reject}\ m\ V\ A\ p)) \land$
$(\forall\ a \in A - B.$
$\mathit{indep\text{-}of\text{-}alt}\ n\ V\ A\ a \land (\forall\ p.\ \mathit{profile}\ V\ A\ p \longrightarrow a \in \mathit{reject}\ n\ V\ A\ p))$
using *assms*
unfolding *disjoint-compatibility-def*
by *metis*
hence
$\exists\ B \subseteq A.$
$(\forall\ a \in A - B.$
$\mathit{indep\text{-}of\text{-}alt}\ n\ V\ A\ a \land (\forall\ p.\ \mathit{profile}\ V\ A\ p \longrightarrow a \in \mathit{reject}\ n\ V\ A\ p)) \land$
$(\forall\ a \in B.$
$\mathit{indep\text{-}of\text{-}alt}\ m\ V\ A\ a \land (\forall\ p.\ \mathit{profile}\ V\ A\ p \longrightarrow a \in \mathit{reject}\ m\ V\ A\ p))$

by *auto*
hence $\exists\ B \subseteq A.$
$(\forall\ a \in A - B.$
indep-of-alt $n\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $n\ V\ A\ p)) \wedge$
$(\forall\ a \in A - (A - B).$
indep-of-alt $m\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ p))$
using *double-diff order-refl*
by *metis*
thus $\exists\ B \subseteq A.$
$(\forall\ a \in B.$
indep-of-alt $n\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $n\ V\ A\ p)) \wedge$
$(\forall\ a \in A - B.$
indep-of-alt $m\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ p))$
by *fastforce*
qed

Every electoral module which is defer-lift-invariant is also defer-monotone.

theorem *dl-inv-imp-def-mono*[*simp*]:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes *defer-lift-invariance m*
shows *defer-monotonicity m*
using *assms*
unfolding *defer-monotonicity-def defer-lift-invariance-def*
by *metis*

4.4.12 Social Choice Properties

Condorcet Consistency

definition *condorcet-consistency* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$ *bool*
where
condorcet-consistency $m \equiv$
$\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$(\forall\ A\ V\ p\ a.$ *condorcet-winner* $V\ A\ p\ a \longrightarrow$
$(m\ V\ A\ p = (\{e \in A.$ *condorcet-winner* $V\ A\ p\ e\}, A - ($*elect* $m\ V\ A\ p), \{\})))$

lemma *condorcet-consistency'*:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
shows *condorcet-consistency* $m =$
$(\mathcal{SCF}$*-result.electoral-module* $m \wedge$
$(\forall\ A\ V\ p\ a.$ *condorcet-winner* $V\ A\ p\ a \longrightarrow$
$(m\ V\ A\ p = (\{a\}, A - ($*elect* $m\ V\ A\ p), \{\}))))$
proof (*safe*)
assume *condorcet-consistency m*
thus $\mathcal{SCF}$*-result.electoral-module* m
unfolding *condorcet-consistency-def*
by *metis*
next
fix
A :: $'a$ *set* **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assume
condorcet-consistency m **and**
condorcet-winner V A p a
thus $m\ V\ A\ p = (\{a\}, A - elect\ m\ V\ A\ p, \{\})$
using *cond-winner-unique*
unfolding *condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
next
assume
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\forall\ A\ V\ p\ a.\ condorcet\text{-}winner\ V\ A\ p\ a \longrightarrow m\ V\ A\ p = (\{a\}, A - elect\ m\ V\ A\ p, \{\})$
moreover have
$\forall\ A\ V\ p\ a.\ condorcet\text{-}winner\ V\ A\ p\ (a{::}'a) \longrightarrow$
$\{b \in A.\ condorcet\text{-}winner\ V\ A\ p\ b\} = \{a\}$
using *cond-winner-unique*
by (*metis* (*full-types*))
ultimately show *condorcet-consistency* m
unfolding *condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
qed

lemma *condorcet-consistency''*:
fixes m :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
shows *condorcet-consistency* m =
($\mathcal{SCF}$*-result.electoral-module* m $\wedge$
$(\forall\ A\ V\ p\ a.$
$condorcet\text{-}winner\ V\ A\ p\ a \longrightarrow m\ V\ A\ p = (\{a\}, A - \{a\}, \{\})))$
proof (*simp only*: *condorcet-consistency'*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assume
e-mod: $\mathcal{SCF}$*-result.electoral-module* m **and**
cc: $\forall\ A\ V\ p\ a'.\ condorcet\text{-}winner\ V\ A\ p\ a' \longrightarrow$
$m\ V\ A\ p = (\{a'\}, A - elect\ m\ V\ A\ p, \{\})$ **and**
c-win: *condorcet-winner* V A p a
show $m\ V\ A\ p = (\{a\}, A - \{a\}, \{\})$
using *cc c-win fst-conv*
by *metis*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**

p :: (′*a*, ′*v*) *Profile* **and**
a :: ′*a*
assume
e-mod: *SCF-result.electoral-module m* **and**
cc: ∀ *A V p a*′. *condorcet-winner V A p a*′ ⟶ *m V A p* = ({*a*′}, *A* − {*a*′}, {}) **and**
c-win: *condorcet-winner V A p a*
show *m V A p* = ({*a*}, *A* − *elect m V A p*, {})
using *cc c-win fst-conv*
by *metis*
qed

(Weak) Monotonicity

An electoral module is monotone iff when an elected alternative is lifted, this alternative remains elected.

definition *monotonicity* :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* ⇒ *bool* **where**
monotonicity m ≡
SCF-result.electoral-module m ∧
(∀ *A V p q a*. *a* ∈ *elect m V A p* ∧ *lifted V A p q a* ⟶ *a* ∈ *elect m V A q*)

end

4.5 Electoral Module on Election Quotients

theory *Quotient-Module*
imports *Quotients/Relation-Quotients*
Electoral-Module
begin

lemma *invariance-is-congruence*:
fixes
m :: (′*a*, ′*v*, ′*r*) *Electoral-Module* **and**
r :: (′*a*, ′*v*) *Election rel*
shows (*satisfies* (*fun*$_\mathcal{E}$ *m*) (*Invariance r*)) = (*fun*$_\mathcal{E}$ *m respects r*)
unfolding *satisfies.simps congruent-def*
by *blast*

lemma *invariance-is-congruence*′:
fixes
f :: ′*x* ⇒ ′*y* **and**
r :: ′*x rel*
shows (*satisfies f* (*Invariance r*)) = (*f respects r*)
unfolding *satisfies.simps congruent-def*
by *blast*

theorem *pass-to-election-quotient*:
fixes
m :: ($'a$, $'v$, $'r$) *Electoral-Module* **and**
r :: ($'a$, $'v$) *Election rel* **and**
X :: ($'a$, $'v$) *Election set*
assumes
equiv X r **and**
satisfies ($fun_{\mathcal{E}}$ m) (*Invariance* r)
shows $\forall\ A \in X\ //\ r.\ \forall\ E \in A.\ \pi_{\mathcal{Q}}\ (fun_{\mathcal{E}}\ m)\ A = fun_{\mathcal{E}}\ m\ E$
using *invariance-is-congruence pass-to-quotient assms*
by *blast*

end

4.6 Evaluation Function

theory *Evaluation-Function*
imports *Social-Choice-Types/Profile*
begin

This is the evaluation function. From a set of currently eligible alternatives, the evaluation function computes a numerical value that is then to be used for further (s)election, e.g., by the elimination module.

4.6.1 Definition

type-synonym ($'a$, $'v$) *Evaluation-Function* =
$'v$ *set* ⇒ $'a$ ⇒ $'a$ *set* ⇒ ($'a$, $'v$) *Profile* ⇒ *enat*

4.6.2 Property

An Evaluation function is a Condorcet-rating iff the following holds: If a Condorcet Winner w exists, w and only w has the highest value.

definition *condorcet-rating* :: ($'a$, $'v$) *Evaluation-Function* ⇒ *bool* **where**
condorcet-rating f ≡
$\forall\ A\ V\ p\ w\ .\ condorcet\text{-}winner\ V\ A\ p\ w \longrightarrow$
$(\forall\ l \in A\ .\ l \neq w \longrightarrow f\ V\ l\ A\ p < f\ V\ w\ A\ p)$

An Evaluation function is dependent only on the participating voters iff it is invariant under profile changes that only impact non-voters.

definition *only-voters-count* :: ($'a$, $'v$) *Evaluation-Function* ⇒ *bool* **where**
only-voters-count f ≡
$\forall\ A\ V\ p\ p'.\ (\forall\ v \in V.\ p\ v = p'\ v) \longrightarrow$
$(\forall\ a \in A.\ f\ V\ a\ A\ p = f\ V\ a\ A\ p')$

4.6.3 Theorems

If e is Condorcet-rating, the following holds: If a Condorcet winner w exists, w has the maximum evaluation value.

```
theorem cond-winner-imp-max-eval-val:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    a :: 'a
  assumes
    rating: condorcet-rating e and
    f-prof: finite-profile V A p and
    winner: condorcet-winner V A p a
  shows e V a A p = Max {e V b A p | b. b ∈ A}
proof −
  let ?set = {e V b A p | b. b ∈ A} and
      ?eMax = Max {e V b A p | b. b ∈ A} and
      ?eW = e V a A p
  have ?eW ∈ ?set
    using CollectI condorcet-winner.simps winner
    by (metis (mono-tags, lifting))
  moreover have ∀ e ∈ ?set. e ≤ ?eW
  proof (safe)
    fix b :: 'a
    assume b ∈ A
    moreover have ∀ n n'. (n::nat) = n' ⟶ n ≤ n'
      by simp
    ultimately show e V b A p ≤ e V a A p
      using less-imp-le rating winner order-refl
      unfolding condorcet-rating-def
      by metis
  qed
  ultimately have ?eW ∈ ?set ∧ (∀ e ∈ ?set. e ≤ ?eW)
    by blast
  moreover have finite ?set
    using f-prof
    by simp
  moreover have ?set ≠ {}
    using condorcet-winner.simps winner
    by fastforce
  ultimately show ?thesis
    using Max-eq-iff
    by (metis (no-types, lifting))
qed
```

If e is Condorcet-rating, the following holds: If a Condorcet Winner w exists, a non-Condorcet winner has a value lower than the maximum evaluation value.

theorem *non-cond-winner-not-max-eval*:
  fixes
    e :: ($'a$, $'v$) *Evaluation-Function* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    a :: $'a$ **and**
    b :: $'a$
  assumes
    rating: *condorcet-rating* e **and**
    f-prof: *finite-profile* V A p **and**
    winner: *condorcet-winner* V A p a **and**
    lin-A: $b \in A$ **and**
    loser: $a \neq b$
  shows $e\ V\ b\ A\ p < Max\ \{e\ V\ c\ A\ p \mid c.\ c \in A\}$
proof $-$
  have $e\ V\ b\ A\ p < e\ V\ a\ A\ p$
    using *lin-A loser rating winner*
    unfolding *condorcet-rating-def*
    by *metis*
  also have $e\ V\ a\ A\ p = Max\ \{e\ V\ c\ A\ p \mid c.\ c \in A\}$
    using *cond-winner-imp-max-eval-val f-prof rating winner*
    by *fastforce*
  finally show *?thesis*
    by *simp*
qed

end

4.7 Elimination Module

theory *Elimination-Module*
  imports *Evaluation-Function*
          Electoral-Module
begin

This is the elimination module. It rejects a set of alternatives only if these are not all alternatives. The alternatives potentially to be rejected are put in a so-called elimination set. These are all alternatives that score below a preset threshold value that depends on the specific voting rule.

4.7.1 General Definitions

type-synonym *Threshold-Value* = *enat*

type-synonym *Threshold-Relation* = *enat* ⇒ *enat* ⇒ *bool*

type-synonym (*'a*, *'v*) *Electoral-Set* = *'v set* ⇒ *'a set* ⇒ (*'a*, *'v*) *Profile* ⇒ *'a set*

fun *elimination-set* :: (*'a*, *'v*) *Evaluation-Function* ⇒ *Threshold-Value* ⇒
Threshold-Relation ⇒ (*'a*, *'v*) *Electoral-Set* **where**
elimination-set e t r V A p = $\{a \in A\ .\ r\ (e\ V\ a\ A\ p)\ t\}$

fun *average* :: (*'a*, *'v*) *Evaluation-Function* ⇒ *'v set* ⇒
'a set ⇒ (*'a*, *'v*) *Profile* ⇒ *Threshold-Value* **where**
average e V A p = (*let sum* = ($\sum$ *x* ∈ *A*. *e V x A p*) *in*
(*if* (*sum* = *infinity*) *then* (*infinity*)
else ((*the-enat sum*) *div* (*card A*))))

4.7.2 Social Choice Definitions

fun *elimination-module* :: (*'a*, *'v*) *Evaluation-Function* ⇒
Threshold-Value ⇒ *Threshold-Relation* ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module*
where
elimination-module e t r V A p =
(*if* (*elimination-set e t r V A p*) ≠ *A*
then ({}, (*elimination-set e t r V A p*), *A* − (*elimination-set e t r V A p*))
else ({}, {}, *A*))

4.7.3 Common Social Choice Eliminators

fun *less-eliminator* :: (*'a*, *'v*) *Evaluation-Function* ⇒
Threshold-Value ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
less-eliminator e t V A p = *elimination-module e t* (<) *V A p*

fun *max-eliminator* ::
(*'a*, *'v*) *Evaluation-Function* ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
max-eliminator e V A p =
less-eliminator e (*Max* {*e V x A p* | *x*. *x* ∈ *A*}) *V A p*
find-theorems *max-eliminator*

fun *leq-eliminator* ::
(*'a*, *'v*) *Evaluation-Function* ⇒ *Threshold-Value* ⇒
(*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
leq-eliminator e t V A p = *elimination-module e t* (≤) *V A p*

fun *min-eliminator* ::
(*'a*, *'v*) *Evaluation-Function* ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
min-eliminator e V A p =
leq-eliminator e (*Min* {*e V x A p* | *x*. *x* ∈ *A*}) *V A p*

fun *less-average-eliminator* ::
(*'a*, *'v*) *Evaluation-Function* ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
less-average-eliminator e V A p = *less-eliminator e* (*average e V A p*) *V A p*

fun *leq-average-eliminator* ::
$('a, 'v)$ *Evaluation-Function* $\Rightarrow$ $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **where**
leq-average-eliminator e V A p = *leq-eliminator e* (*average e V A p*) *V A p*

4.7.4 Soundness

lemma *elim-mod-sound*[*simp*]:
fixes
e :: $('a, 'v)$ *Evaluation-Function* **and**
t :: *Threshold-Value* **and**
r :: *Threshold-Relation*
shows $\mathcal{SCF}$*-result.electoral-module* (*elimination-module e t r*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *less-elim-sound*[*simp*]:
fixes
e :: $('a, 'v)$ *Evaluation-Function* **and**
t :: *Threshold-Value*
shows $\mathcal{SCF}$*-result.electoral-module* (*less-eliminator e t*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *leq-elim-sound*[*simp*]:
fixes
e :: $('a, 'v)$ *Evaluation-Function* **and**
t :: *Threshold-Value*
shows $\mathcal{SCF}$*-result.electoral-module* (*leq-eliminator e t*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *max-elim-sound*[*simp*]:
fixes *e* :: $('a, 'v)$ *Evaluation-Function*
shows $\mathcal{SCF}$*-result.electoral-module* (*max-eliminator e*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *min-elim-sound*[*simp*]:
fixes *e* :: $('a, 'v)$ *Evaluation-Function*
shows $\mathcal{SCF}$*-result.electoral-module* (*min-eliminator e*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *less-avg-elim-sound*[*simp*]:
fixes *e* :: $('a, 'v)$ *Evaluation-Function*
shows $\mathcal{SCF}$*-result.electoral-module* (*less-average-eliminator e*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *leq-avg-elim-sound*[*simp*]:
  fixes $e :: ('a, 'v)$ *Evaluation-Function*
  shows $\mathcal{SCF}$*-result.electoral-module* (*leq-average-eliminator* e)
  unfolding $\mathcal{SCF}$*-result.electoral-module-def*
  by *auto*

4.7.5 Only participating voters impact the result

lemma *elim-mod-only-voters*[*simp*]:
  fixes
    $e :: ('a, 'v)$ *Evaluation-Function* **and**
    $t ::$ *Threshold-Value* **and**
    $r ::$ *Threshold-Relation*
  assumes *only-voters-count* e
  shows *only-voters-vote* (*elimination-module* e t r)
proof (*unfold only-voters-vote-def elimination-module.simps*, *safe*)
  fix
    $A :: 'a$ *set* **and**
    $V :: 'v$ *set* **and**
    $p :: ('a, 'v)$ *Profile* **and**
    $p' :: ('a, 'v)$ *Profile*
  assume $\forall\ v \in V.\ p\ v = p'\ v$
  hence $\forall\ a \in A.\ (e\ V\ a\ A\ p) = (e\ V\ a\ A\ p')$
    using *assms*
    unfolding *only-voters-count-def*
    by *simp*
  hence $\{a \in A.\ r\ (e\ V\ a\ A\ p)\ t\} = \{a \in A.\ r\ (e\ V\ a\ A\ p')\ t\}$
    by *metis*
  hence *elimination-set* $e\ t\ r\ V\ A\ p$ = *elimination-set* $e\ t\ r\ V\ A\ p'$
    unfolding *elimination-set.simps*
    by *presburger*
  thus (*if elimination-set* $e\ t\ r\ V\ A\ p \neq A$
       then ($\{\}$, *elimination-set* $e\ t\ r\ V\ A\ p$, A − *elimination-set* $e\ t\ r\ V\ A\ p$) *else*
($\{\}$, $\{\}$, A)) =
     (*if elimination-set* $e\ t\ r\ V\ A\ p' \neq A$
       then ($\{\}$, *elimination-set* $e\ t\ r\ V\ A\ p'$, A − *elimination-set* $e\ t\ r\ V\ A\ p'$) *else*
($\{\}$, $\{\}$, A))
    by *presburger*
qed

lemma *less-elim-only-voters*[*simp*]:
  fixes
    $e :: ('a, 'v)$ *Evaluation-Function* **and**
    $t ::$ *Threshold-Value*
  assumes *only-voters-count* e
  shows *only-voters-vote* (*less-eliminator* e t)
  unfolding *less-eliminator.simps*
  using *only-voters-vote-def elim-mod-only-voters assms*

by *simp*

lemma *leq-elim-only-voters*[*simp*]:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
assumes *only-voters-count e*
shows *only-voters-vote* (*leq-eliminator e t*)
unfolding *leq-eliminator.simps*
using *only-voters-vote-def elim-mod-only-voters assms*
by *simp*

lemma *max-elim-only-voters*[*simp*]:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
assumes *only-voters-count e*
shows *only-voters-vote* (*max-eliminator e*)
proof (*unfold max-eliminator.simps only-voters-vote-def*, *safe*)
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
p' :: (*'a*, *'v*) *Profile*
assume *coinciding*: $\forall\ v \in V.\ p\ v = p'\ v$
hence $\forall\ x \in A.\ e\ V\ x\ A\ p = e\ V\ x\ A\ p'$
using *assms*
unfolding *only-voters-count-def*
by *simp*
hence $Max\ \{e\ V\ x\ A\ p \mid x.\ x \in A\} = Max\ \{e\ V\ x\ A\ p' \mid x.\ x \in A\}$
by *metis*
thus *less-eliminator e* ($Max\ \{e\ V\ x\ A\ p \mid x.\ x \in A\}$) $V\ A\ p$ =
less-eliminator e ($Max\ \{e\ V\ x\ A\ p' \mid x.\ x \in A\}$) $V\ A\ p'$
using *coinciding assms less-elim-only-voters*
unfolding *only-voters-vote-def*
by (*metis* (*no-types*, *lifting*))
qed

lemma *min-elim-only-voters*[*simp*]:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
assumes *only-voters-count e*
shows *only-voters-vote* (*min-eliminator e*)
proof (*unfold min-eliminator.simps only-voters-vote-def*, *safe*)
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
p' :: (*'a*, *'v*) *Profile*
assume
coinciding: $\forall\ v \in V.\ p\ v = p'\ v$
hence $\forall\ x \in A.\ e\ V\ x\ A\ p = e\ V\ x\ A\ p'$

using *assms*
unfolding *only-voters-count-def*
by *simp*
hence *Min* $\{e\ V\ x\ A\ p \mid x.\ x \in A\} = $ *Min* $\{e\ V\ x\ A\ p' \mid x.\ x \in A\}$
by *metis*
thus *leq-eliminator e* (*Min* $\{e\ V\ x\ A\ p \mid x.\ x \in A\}$) $V\ A\ p =$
leq-eliminator e (*Min* $\{e\ V\ x\ A\ p' \mid x.\ x \in A\}$) $V\ A\ p'$
using *coinciding assms leq-elim-only-voters*
unfolding *only-voters-vote-def*
by (*metis* (*no-types*, *lifting*))
qed

lemma *less-avg-only-voters*[*simp*]:
fixes *e* :: ('*a*, '*v*) *Evaluation-Function*
assumes *only-voters-count e*
shows *only-voters-vote* (*less-average-eliminator e*)
proof (*unfold less-average-eliminator.simps only-voters-vote-def*, *safe*)
fix
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
p' :: ('*a*, '*v*) *Profile*
assume *coinciding*: $\forall\ v \in V.\ p\ v = p'\ v$
hence $\forall\ x \in A.\ e\ V\ x\ A\ p = e\ V\ x\ A\ p'$
using *assms*
unfolding *only-voters-count-def*
by *simp*
hence *average e V A p* = *average e V A p'*
unfolding *average.simps*
by *auto*
thus *less-eliminator e* (*average e V A p*) $V\ A\ p =$
less-eliminator e (*average e V A p'*) $V\ A\ p'$
using *coinciding assms less-elim-only-voters*
unfolding *only-voters-vote-def*
by (*metis* (*no-types*, *lifting*))
qed

lemma *leq-avg-only-voters*[*simp*]:
fixes *e* :: ('*a*, '*v*) *Evaluation-Function*
assumes *only-voters-count e*
shows *only-voters-vote* (*leq-average-eliminator e*)
proof (*unfold leq-average-eliminator.simps only-voters-vote-def*, *safe*)
fix
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
p' :: ('*a*, '*v*) *Profile*
assume *coinciding*: $\forall\ v \in V.\ p\ v = p'\ v$
hence $\forall\ x \in A.\ e\ V\ x\ A\ p = e\ V\ x\ A\ p'$

using *assms*
unfolding *only-voters-count-def*
by *simp*
hence *average e V A p = average e V A p′*
unfolding *average.simps*
by *auto*
thus *leq-eliminator e (average e V A p) V A p =*
leq-eliminator e (average e V A p′) V A p′
using *coinciding assms leq-elim-only-voters*
unfolding *only-voters-vote-def*
by (*metis* (*no-types*, *lifting*))
qed

4.7.6 Non-Blocking

lemma *elim-mod-non-blocking*:
fixes
e :: (′*a*, ′*v*) *Evaluation-Function* **and**
t :: *Threshold-Value* **and**
r :: *Threshold-Relation*
shows *non-blocking* (*elimination-module e t r*)
unfolding *non-blocking-def*
by *auto*

lemma *less-elim-non-blocking*:
fixes
e :: (′*a*, ′*v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
shows *non-blocking* (*less-eliminator e t*)
unfolding *less-eliminator.simps*
using *elim-mod-non-blocking*
by *auto*

lemma *leq-elim-non-blocking*:
fixes
e :: (′*a*, ′*v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
shows *non-blocking* (*leq-eliminator e t*)
unfolding *leq-eliminator.simps*
using *elim-mod-non-blocking*
by *auto*

lemma *max-elim-non-blocking*:
fixes *e* :: (′*a*, ′*v*) *Evaluation-Function*
shows *non-blocking* (*max-eliminator e*)
unfolding *non-blocking-def*
using $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *min-elim-non-blocking*:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
shows *non-blocking* (*min-eliminator e*)
unfolding *non-blocking-def*
using $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *less-avg-elim-non-blocking*:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
shows *non-blocking* (*less-average-eliminator e*)
unfolding *non-blocking-def*
using $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

lemma *leq-avg-elim-non-blocking*:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
shows *non-blocking* (*leq-average-eliminator e*)
unfolding *non-blocking-def*
using $\mathcal{SCF}$*-result.electoral-module-def*
by *auto*

4.7.7 Non-Electing

lemma *elim-mod-non-electing*:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *Threshold-Value* **and**
r :: *Threshold-Relation*
shows *non-electing* (*elimination-module e t r*)
unfolding *non-electing-def*
by *simp*

lemma *less-elim-non-electing*:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
shows *non-electing* (*less-eliminator e t*)
using *elim-mod-non-electing less-elim-sound*
unfolding *non-electing-def*
by *simp*

lemma *leq-elim-non-electing*:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
shows *non-electing* (*leq-eliminator e t*)
unfolding *non-electing-def*
by *simp*

lemma *max-elim-non-electing*:
  fixes e :: ($'a$, $'v$) *Evaluation-Function*
  shows *non-electing* (*max-eliminator* e)
  unfolding *non-electing-def*
  by *simp*

lemma *min-elim-non-electing*:
  fixes e :: ($'a$, $'v$) *Evaluation-Function*
  shows *non-electing* (*min-eliminator* e)
  unfolding *non-electing-def*
  by *simp*

lemma *less-avg-elim-non-electing*:
  fixes e :: ($'a$, $'v$) *Evaluation-Function*
  shows *non-electing* (*less-average-eliminator* e)
  unfolding *non-electing-def*
  by *auto*

lemma *leq-avg-elim-non-electing*:
  fixes e :: ($'a$, $'v$) *Evaluation-Function*
  shows *non-electing* (*leq-average-eliminator* e)
  unfolding *non-electing-def*
  by *simp*

4.7.8 Inference Rules

If the used evaluation function is Condorcet rating, max-eliminator is Condorcet compatible.

theorem *cr-eval-imp-ccomp-max-elim*[*simp*]:
  fixes e :: ($'a$, $'v$) *Evaluation-Function*
  assumes *condorcet-rating* e
  shows *condorcet-compatibility* (*max-eliminator* e)
proof (*unfold condorcet-compatibility-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* (*max-eliminator* e)
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    a :: $'a$
  assume
    c-win: *condorcet-winner* V A p a **and**
    rej-a: $a \in$ *reject* (*max-eliminator* e) V A p
  have $e\ V\ a\ A\ p = Max\ \{e\ V\ b\ A\ p \mid b.\ b \in A\}$
    using *c-win cond-winner-imp-max-eval-val assms*
    by *fastforce*
  hence $a \notin$ *reject* (*max-eliminator* e) V A p
    by *simp*

thus *False*
  using *rej-a*
  by *linarith*
next
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    a :: $'a$
  assume $a \in$ *elect* (*max-eliminator* e) V A p
  moreover have $a \notin$ *elect* (*max-eliminator* e) V A p
    by *simp*
  ultimately show *False*
    by *linarith*
next
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    a :: $'a$ **and**
    a' :: $'a$
  assume
    condorcet-winner V A p a **and**
    $a \in$ *elect* (*max-eliminator* e) V A p
  thus $a' \in$ *reject* (*max-eliminator* e) V A p
    using *condorcet-winner.elims*(*2*) *empty-iff max-elim-non-electing*
    unfolding *non-electing-def*
    by *metis*
qed

If the used evaluation function is Condorcet rating, max-eliminator is defer-Condorcet-consistent.

theorem *cr-eval-imp-dcc-max-elim*[*simp*]:
  fixes e :: $('a, 'v)$ *Evaluation-Function*
  assumes *condorcet-rating* e
  shows *defer-condorcet-consistency* (*max-eliminator* e)
proof (*unfold defer-condorcet-consistency-def*, *safe*, *simp*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    a :: $'a$
  assume
    winner: *condorcet-winner* V A p a
  hence *f-prof*: *finite-profile* V A p
    by *simp*
  let *?trsh* = *Max* $\{e\ V\ b\ A\ p \mid b.\ b \in A\}$
  show
    max-eliminator e V A p =

```
    ({},
      A − defer (max-eliminator e) V A p,
      {b ∈ A. condorcet-winner V A p b})
  proof (cases elimination-set e (?trsh) (<) V A p ≠ A)
    have e V a A p = Max {e V x A p | x. x ∈ A}
      using winner assms cond-winner-imp-max-eval-val
      by fastforce
    hence ∀ b ∈ A. b ≠ a ⟷ b ∈ {c ∈ A. e V c A p < Max {e V b A p | b. b ∈
A}}
      using winner assms mem-Collect-eq linorder-neq-iff
      unfolding condorcet-rating-def
      by (metis (mono-tags, lifting))
    hence elim-set: (elimination-set e ?trsh (<) V A p) = A − {a}
      unfolding elimination-set.simps
      by blast
    case True
    hence
      max-eliminator e V A p =
        ({},
          (elimination-set e ?trsh (<) V A p),
          A − (elimination-set e ?trsh (<) V A p))
      by simp
    also have ... = ({}, A − {a}, {a})
      using elim-set winner
      by auto
    also have ... = ({},A − defer (max-eliminator e) V A p, {a})
      using calculation
      by simp
    also have
      ... = ({},
            A − defer (max-eliminator e) V A p,
            {b ∈ A. condorcet-winner V A p b})
      using cond-winner-unique winner Collect-cong
      by (metis (no-types, lifting))
    finally show ?thesis
      using winner
      by metis
  next
    case False
    moreover have ?trsh = e V a A p
      using assms winner cond-winner-imp-max-eval-val
      by fastforce
    ultimately show ?thesis
      using winner
      by auto
  qed
qed

end
```

4.8 Aggregator

theory *Aggregator*
  imports *Social-Choice-Types/Social-Choice-Result*
begin

An aggregator gets two partitions (results of electoral modules) as input and output another partition. They are used to aggregate results of parallel composed electoral modules. They are commutative, i.e., the order of the aggregated modules does not affect the resulting aggregation. Moreover, they are conservative in the sense that the resulting decisions are subsets of the two given partitions' decisions.

4.8.1 Definition

type-synonym $'a$ *Aggregator* $=$ $'a$ *set* $\Rightarrow$ $'a$ *Result* $\Rightarrow$ $'a$ *Result* $\Rightarrow$ $'a$ *Result*

definition *aggregator* :: $'a$ *Aggregator* $\Rightarrow$ *bool* **where**
  aggregator agg $\equiv$
    $\forall\ A\ e\ e'\ d\ d'\ r\ r'.$
      $(\textit{well-formed-}\mathcal{SCF}\ A\ (e,\ r,\ d) \wedge \textit{well-formed-}\mathcal{SCF}\ A\ (e',\ r',\ d')) \longrightarrow$
      $\textit{well-formed-}\mathcal{SCF}\ A\ (\textit{agg}\ A\ (e,\ r,\ d)\ (e',\ r',\ d'))$

4.8.2 Properties

definition *agg-commutative* :: $'a$ *Aggregator* $\Rightarrow$ *bool* **where**
  agg-commutative agg $\equiv$
    $\textit{aggregator agg} \wedge (\forall\ A\ e\ e'\ d\ d'\ r\ r'.$
      $\textit{agg}\ A\ (e,\ r,\ d)\ (e',\ r',\ d') = \textit{agg}\ A\ (e',\ r',\ d')\ (e,\ r,\ d))$

definition *agg-conservative* :: $'a$ *Aggregator* $\Rightarrow$ *bool* **where**
  agg-conservative agg $\equiv$
    $\textit{aggregator agg} \wedge$
    $(\forall\ A\ e\ e'\ d\ d'\ r\ r'.$
      $((\textit{well-formed-}\mathcal{SCF}\ A\ (e,\ r,\ d) \wedge \textit{well-formed-}\mathcal{SCF}\ A\ (e',\ r',\ d')) \longrightarrow$
        $\textit{elect-r}\ (\textit{agg}\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq (e \cup e') \wedge$
        $\textit{reject-r}\ (\textit{agg}\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq (r \cup r') \wedge$
        $\textit{defer-r}\ (\textit{agg}\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq (d \cup d')))$

end

4.9 Maximum Aggregator

theory *Maximum-Aggregator*
 imports *Aggregator*
begin

The max(imum) aggregator takes two partitions of an alternative set A as input. It returns a partition where every alternative receives the maximum result of the two input partitions.

4.9.1 Definition

fun *max-aggregator* :: $'a$ *Aggregator* **where**
 max-aggregator A $(e,\ r,\ d)$ $(e',\ r',\ d') =$
  $(e \cup e',$
  $A - (e \cup e' \cup d \cup d'),$
  $(d \cup d') - (e \cup e'))$

4.9.2 Auxiliary Lemma

lemma *max-agg-rej-set*:
 fixes
  A :: $'a$ *set* **and**
  e :: $'a$ *set* **and**
  e' :: $'a$ *set* **and**
  d :: $'a$ *set* **and**
  d' :: $'a$ *set* **and**
  r :: $'a$ *set* **and**
  r' :: $'a$ *set* **and**
  a :: $'a$
 assumes
  wf-first-mod: *well-formed-$\mathcal{SCF}$* A $(e,\ r,\ d)$ **and**
  wf-second-mod: *well-formed-$\mathcal{SCF}$* A $(e',\ r',\ d')$
 shows *reject-r* (*max-aggregator* A $(e,\ r,\ d)$ $(e',\ r',\ d')) = r \cap r'$
proof –
 have $A - (e \cup d) = r$
  using *wf-first-mod result-imp-rej*
  by *metis*
 moreover have $A - (e' \cup d') = r'$
  using *wf-second-mod result-imp-rej*
  by *metis*
 ultimately have $A - (e \cup e' \cup d \cup d') = r \cap r'$
  by *blast*
 moreover have $\{l \in A.\ l \notin e \cup e' \cup d \cup d'\} = A - (e \cup e' \cup d \cup d')$
  unfolding *set-diff-eq*
  by *simp*

ultimately show *reject-r* (*max-aggregator* A (e, r, d) (e', r', d')) $= r \cap r'$
  by *simp*
qed

4.9.3 Soundness

theorem *max-agg-sound*[*simp*]: *aggregator max-aggregator*
proof (*unfold aggregator-def*, *simp*, *safe*)
  fix
    A :: $'a$ *set* **and**
    e :: $'a$ *set* **and**
    e' :: $'a$ *set* **and**
    d :: $'a$ *set* **and**
    d' :: $'a$ *set* **and**
    r :: $'a$ *set* **and**
    r' :: $'a$ *set* **and**
    a :: $'a$
  assume
    $e' \cup r' \cup d' = e \cup r \cup d$ **and**
    $a \notin d$ **and**
    $a \notin r$ **and**
    $a \in e'$
  thus $a \in e$
    by *auto*
next
  fix
    A :: $'a$ *set* **and**
    e :: $'a$ *set* **and**
    e' :: $'a$ *set* **and**
    d :: $'a$ *set* **and**
    d' :: $'a$ *set* **and**
    r :: $'a$ *set* **and**
    r' :: $'a$ *set* **and**
    a :: $'a$
  assume
    $e' \cup r' \cup d' = e \cup r \cup d$ **and**
    $a \notin d$ **and**
    $a \notin r$ **and**
    $a \in d'$
  thus $a \in e$
    by *auto*
qed

4.9.4 Properties

The max-aggregator is conservative.

theorem *max-agg-consv*[*simp*]: *agg-conservative max-aggregator*
proof (*unfold agg-conservative-def*, *safe*)
  show *aggregator max-aggregator*

using *max-agg-sound*
by *metis*
next
fix
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**
r' :: $'a$ *set* **and**
a :: $'a$
assume
elect-a: $a \in$ *elect-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**
a-not-in-e': $a \notin e'$
have $a \in e \cup e'$
using *elect-a*
by *simp*
thus $a \in e$
using *a-not-in-e'*
by *simp*
next
fix
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**
r' :: $'a$ *set* **and**
a :: $'a$
assume
wf-result: *well-formed-*$\mathcal{SCF}$ A (e', r', d') **and**
reject-a: $a \in$ *reject-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**
a-not-in-r': $a \notin r'$
have $a \in r \cup r'$
using *wf-result reject-a*
by *force*
thus $a \in r$
using *a-not-in-r'*
by *simp*
next
fix
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**

r' :: $'a$ *set* **and**
a :: $'a$
assume
defer-a: $a \in$ *defer-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**
a-not-in-d': $a \notin d'$
have $a \in d \cup d'$
using *defer-a*
by *force*
thus $a \in d$
using *a-not-in-d'*
by *simp*
qed

The max-aggregator is commutative.

theorem *max-agg-comm*[*simp*]: *agg-commutative max-aggregator*
unfolding *agg-commutative-def*
by *auto*

end

4.10 Termination Condition

theory *Termination-Condition*
imports *Social-Choice-Types/Result*
begin

The termination condition is used in loops. It decides whether or not to terminate the loop after each iteration, depending on the current state of the loop.

4.10.1 Definition

type-synonym $'r$ *Termination-Condition* = $'r$ *Result* $\Rightarrow$ *bool*

end

4.11 Defer Equal Condition

theory *Defer-Equal-Condition*
imports *Termination-Condition*

begin

This is a family of termination conditions. For a natural number n, the according defer-equal condition is true if and only if the given result's defer-set contains exactly n elements.

4.11.1 Definition

fun *defer-equal-condition* :: $nat \Rightarrow 'a$ *Termination-Condition* **where**
defer-equal-condition $n\ (e,\ r,\ d) = (card\ d = n)$

end

Chapter 5

Basic Modules

5.1 Defer Module

theory *Defer-Module*
imports *Component-Types/Electoral-Module*
begin

The defer module is not concerned about the voter's ballots, and simply defers all alternatives. It is primarily used for defining an empty loop.

5.1.1 Definition

fun *defer-module* :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
defer-module $V\ A\ p = (\{\}, \{\}, A)$

5.1.2 Soundness

theorem *def-mod-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module defer-module*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *simp*

5.1.3 Properties

theorem *def-mod-non-electing*: *non-electing defer-module*
unfolding *non-electing-def*
by *simp*

theorem *def-mod-def-lift-inv*: *defer-lift-invariance defer-module*
unfolding *defer-lift-invariance-def*
by *simp*

end

5.2 Elect First Module

theory *Elect-First-Module*
  imports *Component-Types/Electoral-Module*
begin

The elect first module elects the alternative that is most preferred on the first ballot and rejects all other alternatives.

5.2.1 Definition

fun *least* :: $'v$::*wellorder set* $\Rightarrow$ $'v$ **where**
  least V = (*Least* (λ v. $v \in V$))

fun *elect-first-module* :: ($'a$, $'v$::*wellorder*, $'a$ *Result*) *Electoral-Module* **where**
  elect-first-module V A p =
    ($\{a \in A.$ *above* (p (*least* V)) $a = \{a\}\}$,
    $\{a \in A.$ *above* (p (*least* V)) $a \neq \{a\}\}$,
    $\{\}$)

5.2.2 Soundness

theorem *elect-first-mod-sound*: $\mathcal{SCF}$*-result.electoral-module elect-first-module*
proof (*intro* $\mathcal{SCF}$*-result.electoral-modI*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$::*wellorder set* **and**
    p :: ($'a$, $'v$) *Profile*
  have $\{a \in A.$ *above* (p (*least* V)) $a = \{a\}\} \cup \{a \in A.$ *above* (p (*least* V)) $a \neq \{a\}\} = A$
    by *blast*
  hence *set-equals-partition* A (*elect-first-module* V A p)
    by *simp*
  moreover have
    $\forall$ $a \in A$. ($a \notin \{a' \in A.$ *above* (p (*least* V)) $a' = \{a'\}\} \vee$
              $a \notin \{a' \in A.$ *above* (p (*least* V)) $a' \neq \{a'\}\}$)
    by *simp*
  hence $\{a \in A.$ *above* (p (*least* V)) $a = \{a\}\} \cap \{a \in A.$ *above* (p (*least* V)) $a \neq \{a\}\} = \{\}$
    by *blast*
  hence *disjoint3* (*elect-first-module* V A p)
    by *simp*
  ultimately show *well-formed-*$\mathcal{SCF}$ A (*elect-first-module* V A p)
    by *simp*
qed

end

5.3 Consensus Class

theory *Consensus-Class*
  imports *Consensus*
    ../Defer-Module
    ../Elect-First-Module
begin

A consensus class is a pair of a set of elections and a mapping that assigns a unique alternative to each election in that set (of elections). This alternative is then called the consensus alternative (winner). Here, we model the mapping by an electoral module that defers alternatives which are not in the consensus.

5.3.1 Definition

type-synonym $('a,\ 'v,\ 'r)$ *Consensus-Class* $=$ $('a,\ 'v)$ *Consensus* $\times$ $('a,\ 'v,\ 'r)$ *Electoral-Module*

fun *consensus-*$\mathcal{K}$ :: $('a,\ 'v,\ 'r)$ *Consensus-Class* $\Rightarrow$ $('a,\ 'v)$ *Consensus*
  where *consensus-*$\mathcal{K}$ K $=$ *fst* K

fun *rule-*$\mathcal{K}$ :: $('a,\ 'v,\ 'r)$ *Consensus-Class* $\Rightarrow$ $('a,\ 'v,\ 'r)$ *Electoral-Module*
  where *rule-*$\mathcal{K}$ K $=$ *snd* K

5.3.2 Consensus Choice

Returns those consensus elections on a given alternative and voter set from a given consensus that are mapped to the given unique winner by a given consensus rule.

fun $\mathcal{K}_{\mathcal{E}}$ :: $('a,\ 'v,\ 'r\ Result)$ *Consensus-Class* $\Rightarrow$ $'r$ $\Rightarrow$ $('a,\ 'v)$ *Election set* **where**
  $\mathcal{K}_{\mathcal{E}}\ K\ w =$
    $\{(A,\ V,\ p)\ |\ A\ V\ p.\ (\textit{consensus-}\mathcal{K}\ K)\ (A,\ V,\ p) \wedge \textit{finite-profile}\ V\ A\ p$
      $\wedge\ \textit{elect}\ (\textit{rule-}\mathcal{K}\ K)\ V\ A\ p = \{w\}\}$

fun *elections-*$\mathcal{K}$ :: $('a,\ 'v,\ 'r\ Result)$ *Consensus-Class* $\Rightarrow$ $('a,\ 'v)$ *Election set* **where**
  elections-$\mathcal{K}$ $K = \bigcup\ ((\mathcal{K}_{\mathcal{E}}\ K)\ `\ \textit{UNIV})$

A consensus class is deemed well-formed if the result of its mapping is completely determined by its consensus, the elected set of the electoral module's result.

definition *well-formed* :: $('a,\ 'v)$ *Consensus* $\Rightarrow$ $('a,\ 'v,\ 'r)$ *Electoral-Module* $\Rightarrow$ *bool*
where
  well-formed $c\ m \equiv$
    $\forall\ A\ V\ V'\ p\ p'.\ \textit{profile}\ V\ A\ p \wedge \textit{profile}\ V'\ A\ p' \wedge c\ (A,\ V,\ p) \wedge c\ (A,\ V',\ p')$
$\longrightarrow$
      $m\ V\ A\ p = m\ V'\ A\ p'$

A sensible social choice rule for a given arbitrary consensus and social choice rule r is the one that chooses the result of r for all consensus elections and defers all candidates otherwise.

fun *consensus-choice* :: ('a, 'v) *Consensus* ⇒ ('a, 'v, 'a *Result*) *Electoral-Module*
    ⇒ ('a, 'v, 'a *Result*) *Consensus-Class* **where**
  consensus-choice c m =
   (*let*
    w = (λ *V A p*. *if c* (*A*, *V*, *p*) *then m V A p else defer-module V A p*)
    in (*c*, *w*))

5.3.3 Auxiliary Lemmas

lemma *unanimity'-consensus-imp-elect-fst-mod-well-formed*:
  fixes *a* :: '*a*
  shows *well-formed* (λ *c*. *nonempty-set*$_{\mathcal{C}}$ *c* ∧ *nonempty-profile*$_{\mathcal{C}}$ *c* ∧ *equal-top*$_{\mathcal{C}}$' *a c*)
      elect-first-module
proof (*unfold well-formed-def*, *safe*)
  fix
   a :: '*a* **and**
   A :: '*a set* **and**
   V :: '*v*::*wellorder set* **and**
   V' :: '*v set* **and**
   p :: ('*a*, '*v*) *Profile* **and**
   p' :: ('*a*, '*v*) *Profile*
  let *?cond* = λ *c*. *nonempty-set*$_{\mathcal{C}}$ *c* ∧ *nonempty-profile*$_{\mathcal{C}}$ *c* ∧ *equal-top*$_{\mathcal{C}}$' *a c*
  assume
   prof-p: *profile V A p* **and**
   prof-p': *profile V' A p'* **and**
   eq-top-p: *equal-top*$_{\mathcal{C}}$' *a* (*A*, *V*, *p*) **and**
   eq-top-p': *equal-top*$_{\mathcal{C}}$' *a* (*A*, *V'*, *p'*) **and**
   not-empty-A: *nonempty-set*$_{\mathcal{C}}$ (*A*, *V*, *p*) **and**
   not-empty-A': *nonempty-set*$_{\mathcal{C}}$ (*A*, *V'*, *p'*) **and**
   not-empty-p: *nonempty-profile*$_{\mathcal{C}}$ (*A*, *V*, *p*) **and**
   not-empty-p': *nonempty-profile*$_{\mathcal{C}}$ (*A*, *V'*, *p'*)
  hence
   cond-Ap: *?cond* (*A*, *V*, *p*) **and**
   cond-Ap': *?cond* (*A*, *V'*, *p'*)
   by *simp-all*
  have ∀ *a'* ∈ *A*. ((*above* (*p* (*least V*)) *a'* = {*a'*}) = (*above* (*p'* (*least V'*)) *a'* = {*a'*}))
  proof
   fix *a'* :: '*a*
   assume *a'-in-A*: *a'* ∈ *A*
   show (*above* (*p* (*least V*)) *a'* = {*a'*}) = (*above* (*p'* (*least V'*)) *a'* = {*a'*})
   proof (*cases*)
    assume *a'* = *a*
    thus *?thesis*
    using *cond-Ap cond-Ap' Collect-mem-eq LeastI empty-Collect-eq equal-top*$_{\mathcal{C}}$'*.simps*

nonempty-profile$_{\mathcal{C}}$.simps least.simps
by (*metis* (*no-types*, *lifting*))
next
assume *a′-neq-a*: $a' \neq a$
have *non-empty*: $V \neq \{\} \wedge V' \neq \{\}$
using *not-empty-p not-empty-p′*
by *simp*
hence $A \neq \{\} \wedge$ *linear-order-on* A (p (*least* V))
$\wedge$ *linear-order-on* A (p' (*least* V'))
using *not-empty-A not-empty-A′ prof-p prof-p′*
a′-in-A card.remove enumerate.simps(*1*)
enumerate-in-set finite-enumerate-in-set
least.elims all-not-in-conv
zero-less-Suc
unfolding *profile-def*
by *metis*
hence ($a \in$ *above* (p (*least* V)) $a' \vee a' \in$ *above* (p (*least* V)) a) $\wedge$
($a \in$ *above* (p' (*least* V')) $a' \vee a' \in$ *above* (p' (*least* V')) a)
using *a′-in-A a′-neq-a eq-top-p*
unfolding *above-def linear-order-on-def total-on-def*
by *auto*
hence (*above* (p (*least* V)) $a = \{a\} \wedge$ *above* (p (*least* V)) $a' = \{a'\} \longrightarrow a =$
a') $\wedge$
(*above* (p' (*least* V')) $a = \{a\} \wedge$ *above* (p' (*least* V')) $a' = \{a'\} \longrightarrow a$
$= a'$)
by *auto*
thus *?thesis*
using *bot-nat-0.not-eq-extremum card-0-eq cond-Ap cond-Ap′*
enumerate.simps(*1*) *enumerate-in-set equal-top$_{\mathcal{C}}$′.simps*
finite-enumerate-in-set non-empty least.simps
by *metis*
qed
qed
thus *elect-first-module* V A p = *elect-first-module* V' A p'
by *auto*
qed

lemma *strong-unanimity′consensus-imp-elect-fst-mod-completely-determined*:
fixes r :: *′a Preference-Relation*
shows *well-formed*
(λ c. *nonempty-set$_{\mathcal{C}}$* $c \wedge$ *nonempty-profile$_{\mathcal{C}}$* $c \wedge$ *equal-vote$_{\mathcal{C}}$′* r c) *elect-first-module*
proof (*unfold well-formed-def*, *clarify*)
fix
a :: *′a* **and**
A :: *′a set* **and**
V :: *′v::wellorder set* **and**
V' :: *′v set* **and**
p :: (*′a*, *′v*) *Profile* **and**
p' :: (*′a*, *′v*) *Profile*

let *?cond* = λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c* $\wedge$ *equal-vote*$_\mathcal{C}$*'* *r c*
assume
prof-p: *profile V A p* **and**
prof-p': *profile V' A p'* **and**
eq-vote-p: *equal-vote*$_\mathcal{C}$*'* *r* (*A*, *V*, *p*) **and**
eq-vote-p': *equal-vote*$_\mathcal{C}$*'* *r* (*A*, *V'*, *p'*) **and**
not-empty-A: *nonempty-set*$_\mathcal{C}$ (*A*, *V*, *p*) **and**
not-empty-A': *nonempty-set*$_\mathcal{C}$ (*A*, *V'*, *p'*) **and**
not-empty-p: *nonempty-profile*$_\mathcal{C}$ (*A*, *V*, *p*) **and**
not-empty-p': *nonempty-profile*$_\mathcal{C}$ (*A*, *V'*, *p'*)
hence
cond-Ap: *?cond* (*A*, *V*, *p*) **and**
cond-Ap': *?cond* (*A*, *V'*, *p'*)
by *simp-all*
have *p* (*least V*) = *r* $\wedge$ *p'* (*least V'*) = *r*
using *eq-vote-p eq-vote-p' not-empty-p not-empty-p'*
bot-nat-0.not-eq-extremum card-0-eq enumerate.simps(*1*)
enumerate-in-set equal-vote$_\mathcal{C}$*'.simps finite-enumerate-in-set*
nonempty-profile$_\mathcal{C}$*.simps least.elims*
by (*metis* (*no-types*, *lifting*))
thus *elect-first-module V A p* = *elect-first-module V' A p'*
by *auto*
qed

lemma *strong-unanimity'consensus-imp-elect-fst-mod-well-formed*:
fixes *r* :: *'a Preference-Relation*
shows *well-formed* (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c* $\wedge$ *equal-vote*$_\mathcal{C}$*'*
r c)
elect-first-module
using *strong-unanimity'consensus-imp-elect-fst-mod-completely-determined*
by *blast*

lemma *cons-domain-valid*:
fixes *C* :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
shows *elections-*$\mathcal{K}$ *C* $\subseteq$ *valid-elections*
proof
fix *E* :: (*'a*,*'v*) *Election*
assume *E* $\in$ *elections-*$\mathcal{K}$ *C*
hence *fun*$_\mathcal{E}$ *profile E*
unfolding $\mathcal{K}_\mathcal{E}$*.simps*
by *force*
thus *E* $\in$ *valid-elections*
unfolding *valid-elections-def*
by *simp*
qed

lemma *cons-domain-finite*:
fixes *C* :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
shows

finite: *elections-*$\mathcal{K}$ $C \subseteq$ *finite-elections* **and**
finite-voters: *elections-*$\mathcal{K}$ $C \subseteq$ *finite-voter-elections*
proof −
have $\forall$ $E \in$ *elections-*$\mathcal{K}$ C. *fun*$_{\mathcal{E}}$ *profile* $E \wedge$ *finite* (*alternatives-*$\mathcal{E}$ E) $\wedge$ *finite* (*voters-*$\mathcal{E}$ E)
unfolding $\mathcal{K}_{\mathcal{E}}$*.simps*
by *force*
thus *elections-*$\mathcal{K}$ $C \subseteq$ *finite-elections*
unfolding *finite-elections-def fun*$_{\mathcal{E}}$*.simps*
by *blast*
thus *elections-*$\mathcal{K}$ $C \subseteq$ *finite-voter-elections*
unfolding *finite-elections-def finite-voter-elections-def*
by *blast*
qed

5.3.4 Consensus Rules

definition *non-empty-set* :: ($'a$, $'v$, $'r$) *Consensus-Class* $\Rightarrow$ *bool* **where**
non-empty-set $c \equiv \exists$ K. *consensus-*$\mathcal{K}$ c K

Unanimity condition.

definition *unanimity* :: ($'a$, $'v$::*wellorder*, $'a$ *Result*) *Consensus-Class* **where**
unanimity = *consensus-choice unanimity*$_{\mathcal{C}}$ *elect-first-module*

Strong unanimity condition.

definition *strong-unanimity* :: ($'a$, $'v$::*wellorder*, $'a$ *Result*) *Consensus-Class* **where**
strong-unanimity = *consensus-choice strong-unanimity*$_{\mathcal{C}}$ *elect-first-module*

5.3.5 Properties

definition *consensus-rule-anonymity* :: ($'a$, $'v$, $'r$) *Consensus-Class* $\Rightarrow$ *bool* **where**
consensus-rule-anonymity $c \equiv$
($\forall$ A V p π::($'v \Rightarrow 'v$).
bij $\pi \longrightarrow$
(*let* (A', V', q) = (*rename* π (A, V, p)) *in*
profile V A $p \longrightarrow$ *profile* V' A' q
$\longrightarrow$ *consensus-*$\mathcal{K}$ c (A, V, p)
$\longrightarrow$ (*consensus-*$\mathcal{K}$ c (A', V', q) $\wedge$ (*rule-*$\mathcal{K}$ c V A p = *rule-*$\mathcal{K}$ c V' A' q))))

fun *consensus-rule-anonymity'* :: ($'a$, $'v$) *Election set* $\Rightarrow$ ($'a$, $'v$, $'r$ *Result*) *Consensus-Class*
$\Rightarrow$ *bool* **where**
consensus-rule-anonymity' X C =
satisfies (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*Invariance* (*anonymity*$_{\mathcal{R}}$ X))

fun (**in** *result-properties*) *consensus-rule-neutrality* :: ($'a$, $'v$) *Election set*
$\Rightarrow$ ($'a$, $'v$, $'b$ *Result*) *Consensus-Class* $\Rightarrow$ *bool* **where**
consensus-rule-neutrality X C = *satisfies* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C))
(*equivar-ind-by-act* (*carrier neutrality*$_{\mathcal{G}}$) X (φ*-neutr* X) (*set-action* ψ*-neutr*))

fun *consensus-rule-reversal-symmetry* :: ('*a*, '*v*) *Election set*
⇒ ('*a*, '*v*, '*a rel Result*) *Consensus-Class* ⇒ *bool* **where**
consensus-rule-reversal-symmetry X C = *satisfies* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*))
(*equivar-ind-by-act* (*carrier reversal*$_{\mathcal{G}}$) *X* (φ*-rev X*) (*set-action* ψ*-rev*))

5.3.6 Inference Rules

lemma *consensus-choice-equivar*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
c :: ('*a*, '*v*) *Consensus* **and**
G :: '*x set* **and**
X :: ('*a*, '*v*) *Election set* **and**
φ :: ('*x*, ('*a*, '*v*) *Election*) *binary-fun* **and**
ψ :: ('*x*, '*a*) *binary-fun* **and**
f :: '*a Result* ⇒ '*a set*
defines *equivar* ≡ *equivar-ind-by-act G X* φ (*set-action* ψ)
assumes
equivar-m: *satisfies* (*f* ∘ $fun_{\mathcal{E}}$ *m*) *equivar* **and**
equivar-defer: *satisfies* (*f* ∘ $fun_{\mathcal{E}}$ *defer-module*) *equivar* **and**
— Could be generalized to arbitrary modules instead of defer-module
invar-cons: *satisfies c* (*Invariance* (*rel-induced-by-action G X* φ))
shows *satisfies* (*f* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ (*consensus-choice c m*)))
(*equivar-ind-by-act G X* φ (*set-action* ψ))
proof (*simp only*: *rewrite-equivar-ind-by-act*, *standard*, *standard*, *standard*)
fix
E :: ('*a*, '*v*) *Election* **and**
g :: '*x*
assume
g-in-G: *g* ∈ *G* **and**
E-in-X: *E* ∈ *X* **and**
φ*-g-E-in-X*: φ *g E* ∈ *X*
show (*f* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ (*consensus-choice c m*))) (φ *g E*) =
set-action ψ *g* ((*f* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ (*consensus-choice c m*))) *E*)
proof (*cases c E*)
case *True*
hence *c* (φ *g E*)
using *invar-cons rewrite-invar-ind-by-act g-in-G* φ*-g-E-in-X E-in-X*
by *metis*
hence (*f* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ (*consensus-choice c m*))) (φ *g E*) = (*f* ∘ $fun_{\mathcal{E}}$ *m*) (φ *g E*)
by *simp*
also have (*f* ∘ $fun_{\mathcal{E}}$ *m*) (φ *g E*) =
set-action ψ *g* ((*f* ∘ $fun_{\mathcal{E}}$ *m*) *E*)
using *equivar-m E-in-X* φ*-g-E-in-X g-in-G rewrite-equivar-ind-by-act*
unfolding *equivar-def*
by (*metis* (*mono-tags*, *lifting*))
also have (*f* ∘ $fun_{\mathcal{E}}$ *m*) *E* =

$(f \circ fun_{\mathcal{E}}$ $(rule\text{-}\mathcal{K}$ $(consensus\text{-}choice$ c $m)))$ E
    using *True E-in-X g-in-G invar-cons*
    by *simp*
  finally show *?thesis*
    by *simp*
next
  case *False*
  hence $\neg$ c $(\varphi$ g $E)$
    using *invar-cons rewrite-invar-ind-by-act g-in-G* φ*-g-E-in-X E-in-X*
    by *metis*
  hence $(f \circ fun_{\mathcal{E}}$ $(rule\text{-}\mathcal{K}$ $(consensus\text{-}choice$ c $m)))$ $(\varphi$ g $E)$ $=$
    $(f \circ fun_{\mathcal{E}}$ $defer\text{-}module)$ $(\varphi$ g $E)$
    by *simp*
  also have $(f \circ fun_{\mathcal{E}}$ $defer\text{-}module)$ $(\varphi$ g $E)$ $=$
    $set\text{-}action$ ψ g $((f \circ fun_{\mathcal{E}}$ $defer\text{-}module)$ $E)$
    using *equivar-defer E-in-X g-in-G* φ*-g-E-in-X rewrite-equivar-ind-by-act*
    unfolding *equivar-def*
    by (*metis* (*mono-tags*, *lifting*))
  also have $(f \circ fun_{\mathcal{E}}$ $defer\text{-}module)$ E $=$
    $(f \circ fun_{\mathcal{E}}$ $(rule\text{-}\mathcal{K}$ $(consensus\text{-}choice$ c $m)))$ E
    using *False E-in-X g-in-G invar-cons*
    by *simp*
  finally show *?thesis*
    by *simp*
qed
qed

lemma *consensus-choice-anonymous*:
  fixes
    α :: $('a, 'v)$ *Consensus* **and**
    β :: $('a, 'v)$ *Consensus* **and**
    m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
    β' :: $'b \Rightarrow ('a, 'v)$ *Consensus*
  assumes
    beta-sat: $\beta = (\lambda\ E.\ \exists\ a.\ \beta'\ a\ E)$ **and**
    beta'-anon: $\forall\ x.$ *consensus-anonymity* $(\beta'\ x)$ **and**
    anon-cons-cond: *consensus-anonymity* α **and**
    conditions-univ: $\forall\ x.$ *well-formed* $(\lambda\ E.\ \alpha\ E \wedge \beta'\ x\ E)$ m
  shows *consensus-rule-anonymity* (*consensus-choice* $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)$ m)
proof (*unfold consensus-rule-anonymity-def Let-def*, *safe*)
  fix
    A :: $'a$ *set* **and**
    A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    V' :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    q :: $('a, 'v)$ *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume

bij: *bij* π **and**
prof-p: *profile* V A p **and**
prof-q: *profile* V' A' q **and**
renamed: *rename* π $(A,\ V,\ p) = (A',\ V',\ q)$ **and**
consensus-cond: *consensus-*$\mathcal{K}$ (*consensus-choice* $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)$ m) $(A,\ V,$
$p)$
hence $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ (A,\ V,\ p)$
by *simp*
hence
alpha-Ap: $\alpha\ (A,\ V,\ p)$ **and**
beta-Ap: $\beta\ (A,\ V,\ p)$
by *simp-all*
have *alpha-A-perm-p*: $\alpha\ (A',\ V',\ q)$
using *anon-cons-cond alpha-Ap bij prof-p prof-q renamed*
unfolding *consensus-anonymity-def*
by *fastforce*
moreover have $\beta\ (A',\ V',\ q)$
using *beta′-anon beta-Ap beta-sat ex-anon-cons-imp-cons-anonymous*[*of* β β']
bij
prof-p renamed beta′-anon cons-anon-invariant[*of* β]
unfolding *consensus-anonymity-def*
by *blast*
ultimately show *em-cond-perm*:
consensus-$\mathcal{K}$ (*consensus-choice* $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)$ m) $(A',\ V',\ q)$
using *beta-Ap beta-sat ex-anon-cons-imp-cons-anonymous bij*
prof-p prof-q
by *simp*
have $\exists\ x.\ \beta'\ x\ (A,\ V,\ p)$
using *beta-Ap beta-sat*
by *simp*
then obtain x **where**
beta′-x-Ap: $\beta'\ x\ (A,\ V,\ p)$
by *metis*
hence *beta′-x-A-perm-p*: $\beta'\ x\ (A',\ V',\ q)$
using *beta′-anon bij prof-p renamed*
cons-anon-invariant prof-q
unfolding *consensus-anonymity-def*
by *auto*
have $m\ V\ A\ p = m\ V'\ A'\ q$
using *alpha-Ap alpha-A-perm-p beta′-x-Ap beta′-x-A-perm-p*
conditions-univ prof-p prof-q rename.simps prod.inject renamed
unfolding *well-formed-def*
by *metis*
thus *rule-*$\mathcal{K}$ (*consensus-choice* $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)$ m) $V\ A\ p =$
rule-$\mathcal{K}$ (*consensus-choice* $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)$ m) $V'\ A'\ q$
using *consensus-cond em-cond-perm*
by *simp*
qed

5.3.7 Theorems

Anonymity

lemma *unanimity-anonymous*: *consensus-rule-anonymity unanimity*
proof (*unfold unanimity-def*)
  let *?ne-cond* = (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c*)
  have *consensus-anonymity ?ne-cond*
   using *nonempty-set-cons-anonymous nonempty-profile-cons-anonymous cons-anon-conj*
    by *auto*
  moreover have *equal-top*$_\mathcal{C}$ = (λ *c*. $\exists$ *a. equal-top*$_\mathcal{C}$*'* *a c*)
    by *fastforce*
  ultimately have *consensus-rule-anonymity*
     (*consensus-choice*
     (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c* $\wedge$ *equal-top*$_\mathcal{C}$ *c*) *elect-first-module*)
    using *consensus-choice-anonymous*[*of equal-top*$_\mathcal{C}$]
      equal-top-cons'-anonymous unanimity'-consensus-imp-elect-fst-mod-well-formed
    by *fastforce*
  moreover have *consensus-choice*
    (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c* $\wedge$ *equal-top*$_\mathcal{C}$ *c*)
      elect-first-module =
        consensus-choice unanimity$_\mathcal{C}$ *elect-first-module*
    using *unanimity*$_\mathcal{C}$*.simps*
    by *metis*
 ultimately show *consensus-rule-anonymity* (*consensus-choice unanimity*$_\mathcal{C}$ *elect-first-module*)
    by (*metis* (*no-types*))
qed

lemma *strong-unanimity-anonymous*: *consensus-rule-anonymity strong-unanimity*
proof (*unfold strong-unanimity-def*)
  have *consensus-anonymity* (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c*)
   using *nonempty-set-cons-anonymous nonempty-profile-cons-anonymous cons-anon-conj*
    unfolding *consensus-anonymity-def*
    by *simp*
  moreover have *equal-vote*$_\mathcal{C}$ = (λ *c*. $\exists$ *v. equal-vote*$_\mathcal{C}$*'* *v c*)
    by *fastforce*
  ultimately have *consensus-rule-anonymity*
      (*consensus-choice*
     (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c* $\wedge$ *equal-vote*$_\mathcal{C}$ *c*) *elect-first-module*)
    using *consensus-choice-anonymous*[*of equal-vote*$_\mathcal{C}$]
      nonempty-set-cons-anonymous nonempty-profile-cons-anonymous eq-vote-cons'-anonymous
        strong-unanimity'consensus-imp-elect-fst-mod-well-formed
    by *fastforce*
  moreover have *consensus-choice* (λ *c. nonempty-set*$_\mathcal{C}$ *c* $\wedge$ *nonempty-profile*$_\mathcal{C}$ *c*
$\wedge$ *equal-vote*$_\mathcal{C}$ *c*)
           elect-first-module =
             consensus-choice strong-unanimity$_\mathcal{C}$ *elect-first-module*
    using *strong-unanimity*$_\mathcal{C}$*.elims*(*2*, *3*)
    by *metis*
  ultimately show

consensus-rule-anonymity (*consensus-choice strong-unanimity*$_{\mathcal{C}}$ *elect-first-module*)
    by (*metis* (*no-types*))
qed

Neutrality

lemma *defer-winners-equivar*:
  fixes
    G :: $'x$ *set* **and**
    X :: ($'a$, $'v$) *Election set* **and**
    φ :: ($'x$, ($'a$, $'v$) *Election*) *binary-fun* **and**
    ψ :: ($'x$, $'a$) *binary-fun*
  shows *satisfies* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *defer-module*)
        (*equivar-ind-by-act* G X φ (*set-action* ψ))
  using *rewrite-equivar-ind-by-act*
  by *fastforce*

lemma *elect-first-winners-neutral*: *satisfies* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*)
        (*equivar-ind-by-act* (*carrier neutrality*$_{\mathcal{G}}$)
          valid-elections (φ-*neutr valid-elections*) (*set-action* ψ-*neutr*$_{\mathrm{c}}$))
proof (*simp only*: *rewrite-equivar-ind-by-act*, *clarify*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$::*wellorder set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    π :: $'a \Rightarrow 'a$
  assume
    bij: $\pi \in$ *carrier neutrality*$_{\mathcal{G}}$ **and**
    valid: (A, V, p) $\in$ *valid-elections*
  hence *bijective-*π: *bij* π
    unfolding *neutrality*$_{\mathcal{G}}$*-def*
    using *rewrite-carrier*
    by *blast*
  hence *inv*: $\forall$ a. $a = \pi$ (*the-inv* π a)
    by (*simp add*: *f-the-inv-into-f-bij-betw*)
  from *bij valid* **have**
    (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*) (φ-*neutr valid-elections* π (A, V, p)) =
      {$a \in \pi$ ‘ A. *above* (*rel-rename* π (p (*least* V))) a = {a}}
    by *simp*
  moreover have
    {$a \in \pi$ ‘ A. *above* (*rel-rename* π (p (*least* V))) a = {a}} =
      {$a \in \pi$ ‘ A. {b. (a, b) $\in$ {(π a, π b) | a b. (a, b) $\in$ p (*least* V)}} = {a}}
    unfolding *above-def*
    by *simp*
  ultimately have *elect-simp*:
    (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*) (φ-*neutr valid-elections* π (A, V, p)) =
      {$a \in \pi$ ‘ A. {b. (a, b) $\in$ {(π a, π b) | a b. (a, b) $\in$ p (*least* V)}} = {a}}
    by *simp*
  have $\forall$ $a \in \pi$ ‘ A. {b. (a, b) $\in$ {(π x, π y) | x y. (x, y) $\in$ p (*least* V)}} =

{π b | b. (a, π b) ∈ {(π x, π y) | x y. (x, y) ∈ p (*least* V)}}
by *blast*
moreover have ∀ a ∈ π ‘ A.
{π b | b. (a, π b) ∈ {(π x, π y) | x y. (x, y) ∈ p (*least* V)}} =
{π b | b. (π (*the-inv* π a), π b) ∈ {(π x, π y) | x y. (x, y) ∈ p (*least* V)}}
using *bijective-π*
by (*simp add*: *f-the-inv-into-f-bij-betw*)
moreover have ∀ a ∈ π ‘ A. ∀ b.
((π (*the-inv* π a), π b) ∈ {(π x, π y) | x y. (x, y) ∈ p (*least* V)}) =
((*the-inv* π a, b) ∈ {(x, y) | x y. (x, y) ∈ p (*least* V)})
using *bijective-π rel-rename-helper*[*of* π]
by *auto*
moreover have {(x, y) | x y. (x, y) ∈ p (*least* V)} = p (*least* V)
by *simp*
ultimately have
∀ a ∈ π ‘ A. ({b. (a, b) ∈ {(π a, π b) | a b. (a, b) ∈ p (*least* V)}} = {a}) =
({π b | b. (*the-inv* π a, b) ∈ p (*least* V)} = {a})
by *force*
hence {a ∈ π ‘ A. {b. (a, b) ∈ {(π a, π b) | a b. (a, b) ∈ p (*least* V)}} = {a}}
=
{a ∈ π ‘ A. {π b | b. (*the-inv* π a, b) ∈ p (*least* V)} = {a}}
by *auto*
hence (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*) (*φ-neutr valid-elections* π (A, V, p)) =
{a ∈ π ‘ A. {π b | b. (*the-inv* π a, b) ∈ p (*least* V)} = {a}}
using *elect-simp*
by *simp*
also have {a ∈ π ‘ A. {π b | b. (*the-inv* π a, b) ∈ p (*least* V)} = {a}} =
{π a | a. a ∈ A ∧ {π b | b. (a, b) ∈ p (*least* V)} = {π a}}
using *bijective-π inv bij-is-inj the-inv-f-f*
by *fastforce*
also have {π a | a. a ∈ A ∧ {π b | b. (a, b) ∈ p (*least* V)} = {π a}} =
π ‘ {a ∈ A. {π b | b. (a, b) ∈ p (*least* V)} = {π a}}
by *blast*
also have π ‘ {a ∈ A. {π b | b. (a, b) ∈ p (*least* V)} = {π a}} =
π ‘ {a ∈ A. π ‘ {b | b. (a, b) ∈ p (*least* V)} = π ‘ {a}}
by *blast*
finally have
(*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*) (*φ-neutr valid-elections* π (A, V, p)) =
π ‘ {a ∈ A. π ‘ (*above* (p (*least* V)) a) = π ‘ {a}}
unfolding *above-def*
by *simp*
moreover have
∀ a. (π ‘ (*above* (p (*least* V)) a) = π ‘ {a}) =
(*the-inv* π ‘ π ‘ *above* (p (*least* V)) a = *the-inv* π ‘ π ‘ {a})
using ‹*bij* π› *bij-betw-the-inv-into bij-def inj-image-eq-iff*
by *metis*
moreover have ∀ a. (*the-inv* π ‘ π ‘ *above* (p (*least* V)) a = *the-inv* π ‘ π ‘
{a}) =
(*above* (p (*least* V)) a = {a})

using *bijective-π bij-betw-imp-inj-on bij-betw-the-inv-into inj-image-eq-iff*
by *metis*
ultimately have (*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*) (*φ-neutr valid-elections* π (*A*, *V*, *p*)) =
π ‘ {*a* ∈ *A*. *above* (*p* (*least V*)) *a* = {*a*}}
by *presburger*
moreover have *elect elect-first-module V A p* = {*a* ∈ *A*. *above* (*p* (*least V*)) *a* = {*a*}}
by *simp*
moreover have *set-action* ψ-$neutr_{c}$ π
((*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*) (*A*, *V*, *p*)) =
π ‘ (*elect elect-first-module V A p*)
by *auto*
ultimately show
(*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*) (*φ-neutr valid-elections* π (*A*, *V*, *p*)) =
set-action ψ-$neutr_{c}$ π
((*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*) (*A*, *V*, *p*))
by *blast*
qed

lemma *strong-unanimity-neutral*:
defines *domain* ≡ *valid-elections* ∩ *Collect* $strong\text{-}unanimity_{\mathcal{C}}$
— We want to show neutrality on a set as general as possible, as it implies subset neutrality.
shows $\mathcal{SCF}$*-properties.consensus-rule-neutrality domain strong-unanimity*
proof −
have *coincides*: ∀ π. ∀ *E* ∈ *domain*. *φ-neutr domain* π *E* = *φ-neutr valid-elections* π *E*
unfolding *domain-def φ-neutr.simps*
by *auto*
have *consensus-neutrality domain* $strong\text{-}unanimity_{\mathcal{C}}$
using $strong\text{-}unanimity_{\mathcal{C}}$*-neutral invar-under-subset-rel*
unfolding *domain-def*
by *simp*
hence *satisfies* $strong\text{-}unanimity_{\mathcal{C}}$
(*Invariance* (*rel-induced-by-action* (*carrier* $neutrality_{\mathcal{G}}$) *domain* (*φ-neutr valid-elections*)))
unfolding *consensus-neutrality.simps* $neutrality_{\mathcal{R}}$*.simps*
using *coincides coinciding-actions-ind-equal-rel*
by *metis*
moreover have *satisfies* (*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*)
(*equivar-ind-by-act* (*carrier* $neutrality_{\mathcal{G}}$)
domain (*φ-neutr valid-elections*) (*set-action* ψ-$neutr_{c}$))
using *elect-first-winners-neutral*
unfolding *domain-def equivar-ind-by-act-def*
using *equivar-under-subset*
by *blast*
ultimately have *satisfies* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*))
(*equivar-ind-by-act* (*carrier* $neutrality_{\mathcal{G}}$) *domain*
(*φ-neutr valid-elections*) (*set-action* ψ-$neutr_{c}$))

using *defer-winners-equivar*[*of*
carrier neutrality$_{\mathcal{G}}$ *domain* φ*-neutr valid-elections* ψ*-neutr*$_{\mathrm{c}}$]
consensus-choice-equivar[*of*
elect-r elect-first-module carrier neutrality$_{\mathcal{G}}$ *domain*
φ*-neutr valid-elections* ψ*-neutr*$_{\mathrm{c}}$ *strong-unanimity*$_{\mathcal{C}}$]
unfolding *strong-unanimity-def*
by *metis*
thus *?thesis*
unfolding $\mathcal{SCF}$*-properties.consensus-rule-neutrality.simps*
using *coincides equivar-ind-by-act-coincide*
by (*metis* (*no-types*, *lifting*))
qed

lemma *strong-unanimity-neutral'*: $\mathcal{SCF}$*-properties.consensus-rule-neutrality*
(*elections-*$\mathcal{K}$ *strong-unanimity*) *strong-unanimity*
proof −
have *elections-*$\mathcal{K}$ *strong-unanimity* $\subseteq$ *valid-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$
unfolding *valid-elections-def* $\mathcal{K}_{\mathcal{E}}$*.simps strong-unanimity-def*
by *force*
moreover from *this* **have** *coincide*:
$\forall$ π. $\forall$ $E \in$ *elections-*$\mathcal{K}$ *strong-unanimity.*
φ*-neutr* (*valid-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$) π E =
φ*-neutr* (*elections-*$\mathcal{K}$ *strong-unanimity*) π E
unfolding φ*-neutr.simps*
using *extensional-continuation-subset*
by (*metis* (*no-types*, *lifting*))
ultimately have
satisfies (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*))
(*equivar-ind-by-act* (*carrier neutrality*$_{\mathcal{G}}$) (*elections-*$\mathcal{K}$ *strong-unanimity*)
(φ*-neutr* (*valid-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$)) (*set-action* ψ*-neutr*$_{\mathrm{c}}$))
using *strong-unanimity-neutral*
equivar-under-subset[*of*
elect-r $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)
valid-elections $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$
{(φ*-neutr* (*valid-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$) g, *set-action*
ψ*-neutr*$_{\mathrm{c}}$ g) | g.
$g \in$ *carrier neutrality*$_{\mathcal{G}}$} *elections-*$\mathcal{K}$ *strong-unanimity*]
unfolding *equivar-ind-by-act-def* $\mathcal{SCF}$*-properties.consensus-rule-neutrality.simps*
by *blast*
thus *?thesis*
unfolding $\mathcal{SCF}$*-properties.consensus-rule-neutrality.simps*
using *coincide*
equivar-ind-by-act-coincide[*of*
carrier neutrality$_{\mathcal{G}}$ *elections-*$\mathcal{K}$ *strong-unanimity*
φ*-neutr* (*elections-*$\mathcal{K}$ *strong-unanimity*)
φ*-neutr* (*valid-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$)
elect-r $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*) *set-action* ψ*-neutr*$_{\mathrm{c}}$]
by (*metis* (*no-types*))
qed

lemma *strong-unanimity-closed-under-neutrality*: *closed-under-restr-rel* ($neutrality_{\mathcal{R}}$ *valid-elections*) *valid-elections* (*elections-$\mathcal{K}$ strong-unanimity*)
proof (*unfold closed-under-restr-rel.simps restr-rel.simps* $neutrality_{\mathcal{R}}$*.simps rel-induced-by-action.simps elections-$\mathcal{K}$.simps*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'b$ *set* **and**
p :: $('a, 'b)$ *Profile* **and**
A' :: $'a$ *set* **and**
V' :: $'b$ *set* **and**
p' :: $('a, 'b)$ *Profile* **and**
π :: $'a \Rightarrow 'a$ **and**
a :: $'a$
assume
prof: $(A, V, p) \in$ *valid-elections* **and**
cons: $(A, V, p) \in \mathcal{K}_{\mathcal{E}}$ *strong-unanimity* a **and**
bij: $\pi \in$ *carrier* $neutrality_{\mathcal{G}}$ **and**
img: φ*-neutr valid-elections* π $(A, V, p) = (A', V', p')$
hence *fin*: $(A, V, p) \in$ *finite-elections*
unfolding $\mathcal{K}_{\mathcal{E}}$*.simps finite-elections-def*
by *simp*
hence *valid'*: $(A', V', p') \in$ *valid-elections*
using *bij img* φ*-neutr-act.group-action-axioms group-action.element-image prof*
unfolding *finite-elections-def*
by (*metis* (*mono-tags*, *lifting*))
moreover have $V' = V \wedge A' = \pi$ ‘ A
using *img fin alternatives-rename.elims fstI prof sndI*
unfolding *extensional-continuation.simps* φ*-neutr.simps alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps*
by (*metis* (*no-types*, *lifting*))
ultimately have *prof'*: *finite-profile* V' A' p'
using *fin bij CollectD finite-imageI fst-eqD snd-eqD*
unfolding *finite-elections-def valid-elections-def alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps*
by (*metis* (*no-types*, *lifting*))
let *?domain* = *valid-elections* $\cap$ *Collect* $strong\text{-}unanimity_{\mathcal{C}}$
have $((A, V, p), (A', V', p')) \in neutrality_{\mathcal{R}}$ *valid-elections*
using *bij img fin valid'*
unfolding $neutrality_{\mathcal{R}}$*.simps rel-induced-by-action.simps finite-elections-def valid-elections-def*
by *blast*
moreover have *unanimous*: $(A, V, p) \in$ *?domain*
using *cons fin*
unfolding $\mathcal{K}_{\mathcal{E}}$*.simps strong-unanimity-def valid-elections-def*
by *simp*
ultimately have *unanimous'*: $(A', V', p') \in$ *?domain*
using $strong\text{-}unanimity_{\mathcal{C}}$*-neutral*
by *force*

have *rewrite*: $\forall\ \pi \in$ *carrier neutrality*$_{\mathcal{G}}$.
  φ*-neutr ?domain* π $(A,\ V,\ p) \in$ *?domain* $\longrightarrow$
    (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) (φ*-neutr ?domain* π $(A,\ V,\ p)$) =
      set-action ψ*-neutr*$_{\mathrm{c}}$ π ((*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) $(A,\ V,\ p)$)
  using *strong-unanimity-neutral unanimous*
    rewrite-equivar-ind-by-act[*of*
      elect-r $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)
      carrier neutrality$_{\mathcal{G}}$ *?domain*
      φ*-neutr ?domain set-action* ψ*-neutr*$_{\mathrm{c}}$]
  unfolding $\mathcal{SCF}$*-properties.consensus-rule-neutrality.simps*
  by *blast*
have *img'*: φ*-neutr ?domain* π $(A,\ V,\ p) = (A',\ V',\ p')$
  using *img unanimous*
  by *simp*
hence *elect* (*rule-*$\mathcal{K}$ *strong-unanimity*) $V'\ A'\ p'$ =
    (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) (φ*-neutr ?domain* π $(A,\ V,\ p)$)
  by *simp*
also have (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) (φ*-neutr ?domain* π $(A,\ V,\ p)$) =
  set-action ψ*-neutr*$_{\mathrm{c}}$ π
    ((*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) $(A,\ V,\ p)$)
  using *bij img' unanimous' rewrite*
  by *fastforce*
also have (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*)) $(A,\ V,\ p) = \{a\}$
  using *cons*
  unfolding $\mathcal{K}_{\mathcal{E}}$*.simps*
  by *simp*
finally have *elect* (*rule-*$\mathcal{K}$ *strong-unanimity*) $V'\ A'\ p' = \{\psi$*-neutr*$_{\mathrm{c}}$ $\pi\ a\}$
  by *simp*
hence $(A',\ V',\ p') \in \mathcal{K}_{\mathcal{E}}$ *strong-unanimity* (ψ*-neutr*$_{\mathrm{c}}$ $\pi\ a$)
  unfolding $\mathcal{K}_{\mathcal{E}}$*.simps strong-unanimity-def consensus-choice.simps*
  using *unanimous' prof'*
  by *simp*
hence $(A',\ V',\ p') \in$ *elections-*$\mathcal{K}$ *strong-unanimity*
  by *simp*
hence $((A,\ V,\ p),\ (A',\ V',\ p'))$
    $\in \bigcup$ (*range* ($\mathcal{K}_{\mathcal{E}}$ *strong-unanimity*)) $\times \bigcup$ (*range* ($\mathcal{K}_{\mathcal{E}}$ *strong-unanimity*))
  unfolding *elections-*$\mathcal{K}$*.simps*
  using *cons*
  by *blast*
moreover have $\exists\ \pi \in$ *carrier neutrality*$_{\mathcal{G}}$. φ*-neutr valid-elections* π $(A,\ V,\ p) = (A',\ V',\ p')$
  using *img bij*
  unfolding *neutrality*$_{\mathcal{G}}$*-def*
  by *blast*
ultimately show $(A',\ V',\ p') \in \bigcup$ (*range* ($\mathcal{K}_{\mathcal{E}}$ *strong-unanimity*))
  by *blast*

qed

end

5.4 Distance Rationalization

theory *Distance-Rationalization*
  imports *Social-Choice-Types/Refined-Types/Preference-List*
    Consensus-Class
    Distance
begin

A distance rationalization of a voting rule is its interpretation as a procedure that elects an uncontroversial winner if there is one, and otherwise elects the alternatives that are as close to becoming an uncontroversial winner as possible. Within general distance rationalization, a voting rule is characterized by a distance on profiles and a consensus class.

5.4.1 Definitions

Returns the distance of an election to the preimage of a unique winner under the given consensus elections and consensus rule.

fun $\textit{score} :: ('a,\ 'v)\ \textit{Election Distance} \Rightarrow ('a,\ 'v,\ 'r\ \textit{Result})\ \textit{Consensus-Class}$
    $\Rightarrow ('a,\ 'v)\ \textit{Election} \Rightarrow\ 'r \Rightarrow \textit{ereal}$ **where**
  $\textit{score}\ d\ K\ E\ w = \textit{Inf}\ (d\ E\ `\ (\mathcal{K}_{\mathcal{E}}\ K\ w))$

fun (**in** *result*) $\mathcal{R}_{\mathcal{W}} :: ('a,\ 'v)\ \textit{Election Distance} \Rightarrow ('a,\ 'v,\ 'r\ \textit{Result})\ \textit{Consensus-Class} \Rightarrow$
    $'v\ \textit{set} \Rightarrow\ 'a\ \textit{set} \Rightarrow ('a,\ 'v)\ \textit{Profile} \Rightarrow\ 'r\ \textit{set}$ **where**
  $\mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p = \textit{arg-min-set}\ (\textit{score}\ d\ K\ (A,\ V,\ p))\ (\textit{limit-set}\ A\ \textit{UNIV})$

fun (**in** *result*) $\textit{distance-}\mathcal{R} :: ('a,\ 'v)\ \textit{Election Distance} \Rightarrow ('a,\ 'v,\ 'r\ \textit{Result})\ \textit{Consensus-Class} \Rightarrow$
    $('a,\ 'v,\ 'r\ \textit{Result})\ \textit{Electoral-Module}$ **where**
  $\textit{distance-}\mathcal{R}\ d\ K\ V\ A\ p = (\mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p,\ (\textit{limit-set}\ A\ \textit{UNIV}) - \mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p,\ \{\})$

5.4.2 Standard Definitions

definition $\textit{standard} :: ('a,\ 'v)\ \textit{Election Distance} \Rightarrow \textit{bool}$ **where**
  $\textit{standard}\ d \equiv \forall\ A\ A'\ V\ V'\ p\ p'.\ (V \neq V' \lor A \neq A') \longrightarrow d\ (A,\ V,\ p)\ (A',\ V',\ p') = \infty$

definition $\textit{non-voters-irrelevant} :: ('a,\ 'v)\ \textit{Election Distance} \Rightarrow \textit{bool}$ **where**
  $\textit{non-voters-irrelevant}\ d \equiv \forall\ A\ A'\ V\ V'\ p\ q\ p'.$

$(\forall\ v \in V.\ p\ v = q\ v) \longrightarrow (d\ (A,\ V,\ p)\ (A',\ V',\ p') = d\ (A,\ V,\ q)\ (A',\ V',\ p')$
$\wedge\ (d\ (A',\ V',\ p')\ (A,\ V,\ p) = d\ (A',\ V',\ p')\ (A,\ V,\ q)))$

Creates a set of all possible profiles on a finite alternative set that are empty everywhere outside of a given finite voter set.

fun *all-profiles* :: $'v$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $(('a,\ 'v)$ *Profile*) *set* **where**
all-profiles $V\ A$ =
(*if* (*infinite* A $\vee$ *infinite* V)
then {} *else* {p. p ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)})

fun $\mathcal{K}_{\mathcal{E}}$*-std* :: $('a,\ 'v,\ 'r$ *Result*) *Consensus-Class* $\Rightarrow$ $'r$ $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'v$ *set*
$\Rightarrow$ $('a,\ 'v)$ *Election set* **where**
$\mathcal{K}_{\mathcal{E}}$*-std* $K\ w\ A\ V$ =
(λ p. $(A,\ V,\ p)$) ‘ (*Set.filter*
(λ p. (*consensus-*$\mathcal{K}$ K) $(A,\ V,\ p)$ $\wedge$ *elect* (*rule-*$\mathcal{K}$ K) $V\ A\ p$ =
{w})
(*all-profiles* $V\ A$))

Returns those consensus elections on a given alternative and voter set from a given consensus that are mapped to the given unique winner by a given consensus rule.

fun *score-std* :: $('a,\ 'v)$ *Election Distance* $\Rightarrow$ $('a,\ 'v,\ 'r$ *Result*) *Consensus-Class* $\Rightarrow$
$('a,\ 'v)$ *Election* $\Rightarrow$ $'r$ $\Rightarrow$ *ereal* **where**
score-std $d\ K\ E\ w$ =
(*if* $\mathcal{K}_{\mathcal{E}}$*-std* $K\ w$ (*alternatives-*$\mathcal{E}$ E) (*voters-*$\mathcal{E}$ E) = {}
then ∞ *else Min* ($d\ E$ ‘ ($\mathcal{K}_{\mathcal{E}}$*-std* $K\ w$ (*alternatives-*$\mathcal{E}$ E) (*voters-*$\mathcal{E}$ E))))

fun (**in** *result*) $\mathcal{R}_{\mathcal{W}}$*-std* :: $('a,\ 'v)$ *Election Distance* $\Rightarrow$ $('a,\ 'v,\ 'r$ *Result*) *Consensus-Class* $\Rightarrow$
$'v$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $('a,\ 'v)$ *Profile* $\Rightarrow$ $'r$ *set* **where**
$\mathcal{R}_{\mathcal{W}}$*-std* $d\ K\ V\ A\ p$ = *arg-min-set* (*score-std* $d\ K$ $(A,\ V,\ p)$) (*limit-set* A *UNIV*)

fun (**in** *result*) *distance-*$\mathcal{R}$*-std* :: $('a,\ 'v)$ *Election Distance* $\Rightarrow$
$('a,\ 'v,\ 'r$ *Result*) *Consensus-Class* $\Rightarrow$ $('a,\ 'v,\ 'r$ *Result*) *Electoral-Module*
where
distance-$\mathcal{R}$*-std* $d\ K\ V\ A\ p$ = ($\mathcal{R}_{\mathcal{W}}$*-std* $d\ K\ V\ A\ p$, (*limit-set* A *UNIV*) $-$ $\mathcal{R}_{\mathcal{W}}$*-std* $d\ K\ V\ A\ p$, {})

5.4.3 Auxiliary Lemmas

lemma *fin-*$\mathcal{K}_{\mathcal{E}}$:
fixes C :: $('a,\ 'v,\ 'r$ *Result*) *Consensus-Class*
shows *elections-*$\mathcal{K}$ $C \subseteq$ *finite-elections*
proof
fix E :: $('a,'v)$ *Election*
assume $E \in$ *elections-*$\mathcal{K}$ C
hence *finite-election* E
unfolding $\mathcal{K}_{\mathcal{E}}$*.simps*
by *force*

thus $E \in$ *finite-elections*
unfolding *finite-elections-def*
by *simp*
qed

lemma *univ-*$\mathcal{K}_{\mathcal{E}}$:
fixes C :: $('a, 'v, 'r$ *Result*) *Consensus-Class*
shows *elections-*$\mathcal{K}$ $C \subseteq$ *UNIV*
by *simp*

lemma *list-cons-presv-finiteness*:
fixes
A :: $'a$ *set* **and**
S :: $'a$ *list set*
assumes
fin-A: *finite* A **and**
fin-B: *finite* S
shows *finite* $\{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
proof −
let *?P* $= \lambda\ A.$ *finite* $\{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
have $\bigwedge\ a\ A'.$ *finite* $A' \Longrightarrow a \notin A' \Longrightarrow$ *?P* $A' \Longrightarrow$ *?P* (*insert* $a\ A'$)
proof −
fix
a :: $'a$ **and**
A' :: $'a$ *set*
assume
fin: *finite* A' **and**
not-in: $a \notin A'$ **and**
fin-set: *finite* $\{a\#l \mid a\ l.\ a \in A' \wedge l \in S\}$
have $\{a'\#l \mid a'\ l.\ a' \in$ *insert* $a\ A' \wedge l \in S\}$
$= \{a\#l \mid a\ l.\ a \in A' \wedge l \in S\} \cup \{a\#l \mid l.\ l \in S\}$
by *auto*
moreover have *finite* $\{a\#l \mid l.\ l \in S\}$
using *fin-B*
by *simp*
ultimately have *finite* $\{a'\#l \mid a'\ l.\ a' \in$ *insert* $a\ A' \wedge l \in S\}$
using *fin-set*
by *simp*
thus *?P* (*insert* $a\ A'$)
by *simp*
qed
moreover have *?P* $\{\}$
by *simp*
ultimately show *?P* A
using *finite-induct*[*of* A *?P*] *fin-A*
by *simp*
qed

lemma *listset-finiteness*:

fixes *l* :: *'a set list*
assumes ∀ *i*::*nat*. *i* < *length l* ⟶ *finite* (*l*!*i*)
shows *finite* (*listset l*)
using *assms*
proof (*induct l*, *simp*)
case (*Cons a l*)
fix
a :: *'a set* **and**
l :: *'a set list*
assume
elems-fin-then-set-fin: ∀ *i*::*nat* < *length l*. *finite* (*l*!*i*) ⟹ *finite* (*listset l*) **and**
fin-all-elems: ∀ *i*::*nat* < *length* (*a*#*l*). *finite* ((*a*#*l*)!*i*)
hence *finite a*
by *auto*
moreover from *fin-all-elems*
have ∀ *i* < *length l*. *finite* (*l*!*i*)
by *auto*
hence *finite* (*listset l*)
using *elems-fin-then-set-fin*
by *simp*
ultimately have *finite* {*a'*#*l'* | *a' l'*. *a'* ∈ *a* ∧ *l'* ∈ (*listset l*)}
using *list-cons-presv-finiteness*
by *auto*
thus *finite* (*listset* (*a*#*l*))
by (*simp add*: *set-Cons-def*)
qed

lemma *ls-entries-empty-imp-ls-set-empty*:
fixes *l* :: *'a set list*
assumes
0 < *length l* **and**
∀ *i* ::*nat*. *i* < *length l* ⟶ *l*!*i* = {}
shows *listset l* = {}
using *assms*
proof (*induct l*, *simp*)
case (*Cons a l*)
fix
a :: *'a set* **and**
l :: *'a set list*
assume *all-elems-empty*: ∀ *i*::*nat* < *length* (*a*#*l*). (*a*#*l*)!*i* = {}
hence *a* = {}
by *auto*
moreover from *all-elems-empty*
have ∀ *i* < *length l*. *l*!*i* = {}
by *auto*
ultimately have {*a'*#*l'* | *a' l'*. *a'* ∈ *a* ∧ *l'* ∈ (*listset l*)} = {}
by *simp*
thus *listset* (*a*#*l*) = {}
by (*simp add*: *set-Cons-def*)

qed

lemma *all-ls-elems-same-len*:
  fixes l :: $'a$ *set list*
  shows $\forall\ l'::('a\ list).\ l' \in listset\ l \longrightarrow length\ l' = length\ l$
proof (*induct* l, *simp*)
  case (*Cons a l*)
  fix
    a :: $'a$ *set* **and**
    l :: $'a$ *set list*
  assume $\forall\ l'.\ l' \in listset\ l \longrightarrow length\ l' = length\ l$
  moreover have
    $\forall\ a'\ l'::('a\ set\ list).\ listset\ (a'\#l') = \{b\#m \mid b\ m.\ b \in a' \wedge m \in listset\ l'\}$
    by (*simp add*: *set-Cons-def*)
  ultimately show $\forall\ l'.\ l' \in listset\ (a\#l) \longrightarrow length\ l' = length\ (a\#l)$
    using *local.Cons*
    by *fastforce*
qed

lemma *all-ls-elems-in-ls-set*:
  fixes l :: $'a$ *set list*
  shows $\forall\ l'\ i::nat.\ l' \in listset\ l \wedge i < length\ l' \longrightarrow l'!i \in l!i$
proof (*induct* l, *simp*, *safe*)
  case (*Cons a l*)
  fix
    a :: $'a$ *set* **and**
    l :: $'a$ *set list* **and**
    l' :: $'a$ *list* **and**
    i :: *nat*
  assume *elems-in-set-then-elems-pos*:
    $\forall\ l'\ i::nat.\ l' \in listset\ l \wedge i < length\ l' \longrightarrow l'!i \in l!i$ **and**
    l-prime-in-set-a-l: $l' \in listset\ (a\#l)$ **and**
    i-lt-len-l-prime: $i < length\ l'$
  have $l' \in set\text{-}Cons\ a\ (listset\ l)$
    using *l-prime-in-set-a-l*
    by *simp*
  hence $l' \in \{m.\ \exists\ b\ m'.\ m = b\#m' \wedge b \in a \wedge m' \in (listset\ l)\}$
    unfolding *set-Cons-def*
    by *simp*
  hence $\exists\ b\ m.\ l' = b\#m \wedge b \in a \wedge m \in (listset\ l)$
    by *simp*
  thus $l'!i \in (a\#l)!i$
    using *elems-in-set-then-elems-pos i-lt-len-l-prime nth-Cons-Suc*
        Suc-less-eq gr0-conv-Suc length-Cons nth-non-equal-first-eq
    by *metis*
qed

lemma *fin-all-profs*:
  fixes

A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
x :: $'a$ *Preference-Relation*
assumes
fin-A: *finite* A **and**
fin-V: *finite* V
shows *finite* (*all-profiles* V A $\cap$ $\{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = x\}$)
proof (*cases* $A = \{\}$)
let *?profs* = *all-profiles* V A $\cap$ $\{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = x\}$
case *True*
hence *permutations-of-set* $A = \{[]\}$
unfolding *permutations-of-set-def*
by *fastforce*
hence *pl-α* ' *permutations-of-set* $A = \{\{\}\}$
unfolding *pl-α-def*
by *simp*
hence $\forall$ $p \in$ *all-profiles* V A. $\forall\ v.\ v \in V \longrightarrow p\ v = \{\}$
by (*simp add*: *image-subset-iff*)
hence $\forall$ $p \in$ *?profs*. $(\forall\ v.\ v \in V \longrightarrow p\ v = \{\}) \land (\forall\ v.\ v \notin V \longrightarrow p\ v = x)$
by *simp*
hence $\forall$ $p \in$ *?profs*. $p = (\lambda\ v.$ *if* $v \in V$ *then* $\{\}$ *else* $x)$
by (*metis* (*no-types*, *lifting*))
hence *?profs* $\subseteq$ $\{\lambda\ v.$ *if* $v \in V$ *then* $\{\}$ *else* $x\}$
by *blast*
thus *finite ?profs*
using *finite.emptyI finite-insert finite-subset*
by (*metis* (*no-types*, *lifting*))
next
let *?profs* = (*all-profiles* V A $\cap$ $\{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = x\}$)
case *False*
from *fin-V* **obtain** *ord* **where** *linear-order-on* V *ord*
using *finite-list lin-ord-equiv lin-order-equiv-list-of-alts*
by *metis*
then obtain *list-V* **where**
len: *length list-V* = *card* V **and**
pl: *ord* = *pl-α list-V* **and**
perm: *list-V* $\in$ *permutations-of-set* V
using *lin-order-pl-α fin-V image-iff length-finite-permutations-of-set*
by *metis*
let *?map* = λ *p*::($'a$, $'v$) *Profile*. *map p list-V*
have $\forall$ $p \in$ *all-profiles* V A. $\forall$ $v \in V$. $p\ v \in$ (*pl-α* ' *permutations-of-set* A)
by (*simp add*: *image-subset-iff*)
hence $\forall$ $p \in$ *all-profiles* V A. ($\forall$ $v \in V$. *linear-order-on* A $(p\ v)$)
using *pl-α-lin-order fin-A False*
by *metis*
moreover have $\forall$ $p \in$ *?profs*. $\forall$ $i <$ *length* (*?map p*). (*?map p*)!i = p (*list-V*!i)
by *simp*
moreover have $\forall$ $i <$ *length list-V*. *list-V*!$i \in V$
using *perm nth-mem permutations-of-setD*(*1*)

by *metis*
moreover have *lens-eq*: $\forall\ p \in$ *?profs*. *length* (*?map p*) = *length list-V*
by *simp*
ultimately have $\forall\ p \in$ *?profs*. $\forall\ i <$ *length* (*?map p*). *linear-order-on A* ((*?map p*)!*i*)
by *simp*
hence *subset*: *?map* ` *?profs* $\subseteq$ {*xs*. *length xs* = *card V* $\wedge$
($\forall\ i <$ *length xs*. *linear-order-on A* (*xs*!*i*))}
using *len lens-eq*
by *fastforce*
have $\forall$ *p1 p2*. *p1* $\in$ *?profs* $\wedge$ *p2* $\in$ *?profs* $\wedge$ *p1* $\neq$ *p2* $\longrightarrow$ ($\exists\ v \in V$. *p1 v* $\neq$ *p2 v*)
by *fastforce*
hence $\forall$ *p1 p2*. *p1* $\in$ *?profs* $\wedge$ *p2* $\in$ *?profs* $\wedge$ *p1* $\neq$ *p2* $\longrightarrow$ ($\exists\ v \in$ *set list-V*. *p1 v* $\neq$ *p2 v*)
using *perm*
unfolding *permutations-of-set-def*
by *simp*
hence $\forall$ *p1 p2*. *p1* $\in$ *?profs* $\wedge$ *p2* $\in$ *?profs* $\wedge$ *p1* $\neq$ *p2* $\longrightarrow$ *?map p1* $\neq$ *?map p2*
by *simp*
hence *inj-on ?map ?profs*
unfolding *inj-on-def*
by *blast*
moreover have *finite* {*xs*. *length xs* = *card V* $\wedge$ ($\forall\ i <$ *length xs*. *linear-order-on A* (*xs*!*i*))}
proof −
have *finite* {*r*. *linear-order-on A r*}
using *fin-A*
unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
by *simp*
hence *finSupset*: $\forall\ n$. *finite* {*xs*. *length xs* = *n* $\wedge$ *set xs* $\subseteq$ {*r*. *linear-order-on A r*}}
using *Collect-mono finite-lists-length-eq rev-finite-subset*
by (*metis* (*no-types*, *lifting*))
have $\forall\ l \in$ {*xs*. *length xs* = *card V* $\wedge$
($\forall\ i <$ *length xs*. *linear-order-on A* (*xs*!*i*))}.
set l $\subseteq$ {*r*. *linear-order-on A r*}
using *in-set-conv-nth mem-Collect-eq subsetI*
by (*metis* (*no-types*, *lifting*))
hence {*xs*. *length xs* = *card V* $\wedge$
($\forall\ i <$ *length xs*. *linear-order-on A* (*xs*!*i*))}
$\subseteq$ {*xs*. *length xs* = *card V* $\wedge$ *set xs* $\subseteq$ {*r*. *linear-order-on A r*}}
by *blast*
thus *?thesis*
using *finSupset rev-finite-subset*
by *blast*
qed
moreover have $\forall$ *f X Y*. *inj-on f X* $\wedge$ *finite Y* $\wedge$ *f* ` *X* $\subseteq$ *Y* $\longrightarrow$ *finite X*
using *finite-imageD finite-subset*

    by *metis*
  ultimately show *finite ?profs*
    using *subset*
    by *blast*
qed

lemma *profile-permutation-set*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set*
  shows *all-profiles* V A =
        $\{p' :: ('a, 'v)$ *Profile*. *finite-profile* V A $p'\}$
proof (*cases finite* $A \wedge$ *finite* $V \wedge A \neq \{\}$, *clarsimp*)
  assume
    fin-A: *finite* A **and**
    fin-V: *finite* V **and**
    non-empty: $A \neq \{\}$
  show $\{\pi.\ \pi$ ‘ $V \subseteq$ *pl-*α ‘ *permutations-of-set* $A\} = \{p'.$ *profile* V A $p'\}$
  proof
    show $\{\pi.\ \pi$ ‘ $V \subseteq$ *pl-*α ‘ *permutations-of-set* $A\} \subseteq \{p'.$ *profile* V A $p'\}$
    proof (*rule*, *clarify*)
      fix p' :: $'v \Rightarrow 'a$ *Preference-Relation*
      assume
        subset: p' ‘ $V \subseteq$ *pl-*α ‘ *permutations-of-set* A
      hence $\forall\ v \in V.\ p'\ v \in$ *pl-*α ‘ *permutations-of-set* A
        by *blast*
      hence $\forall\ v \in V.$ *linear-order-on* A $(p'\ v)$
        using *fin-A pl-*α*-lin-order non-empty*
        by *metis*
      thus *profile* V A p'
        unfolding *profile-def*
        by *simp*
    qed
  next
    show $\{p'.$ *profile* V A $p'\} \subseteq \{\pi.\ \pi$ ‘ $V \subseteq$ *pl-*α ‘ *permutations-of-set* $A\}$
    proof (*rule*, *clarify*)
      fix
        p' :: $('a, 'v)$ *Profile* **and**
        v :: $'v$
      assume
        prof: *profile* V A p' **and**
        el: $v \in V$
      hence *linear-order-on* A $(p'\ v)$
        unfolding *profile-def*
        by *simp*
      thus $(p'\ v) \in$ *pl-*α ‘ *permutations-of-set* A
        using *fin-A lin-order-pl-*α
        by *simp*
    qed

qed
next
assume *not-fin-empty*: $\neg$ (*finite* A $\wedge$ *finite* V $\wedge$ $A \neq \{\}$)
have *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\} \Longrightarrow$ *permutations-of-set* $A = \{[]\}$
unfolding *permutations-of-set-def*
by *fastforce*
hence *pl-empty*: *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\} \Longrightarrow$ *pl-*α ‘ *permutations-of-set* A $= \{\{\}\}$
unfolding *pl-*α*-def*
by *simp*
hence *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\} \Longrightarrow$
$\forall\ \pi \in \{\pi.\ \pi$ ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\}.\ \forall\ v \in V.\ \pi\ v = \{\}$
by *fastforce*
hence *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\} \Longrightarrow$
$\{\pi.\ \pi$ ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\} = \{\pi.\ \forall\ v \in V.\ \pi\ v = \{\}\}$
using *image-subset-iff singletonD singletonI pl-empty*

by *fastforce*
moreover have *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow$ *all-profiles* $V\ A = \{\pi.\ \pi$ ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\}$
by *simp*
ultimately have *all-prof-eq*: *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow$ *all-profiles* $V\ A = \{\pi.\ \forall\ v \in V.\ \pi\ v = \{\}\}$
by *simp*
have *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow \forall\ p' \in \{p'.$ *finite-profile* $V\ A\ p' \wedge (\forall\ v'.\ v' \notin V \longrightarrow p'\ v' = \{\})\}.$
$(\forall\ v \in V.$ *linear-order-on* $\{\}\ (p'\ v))$
unfolding *profile-def*
by *simp*
moreover have $\forall\ r.$ *linear-order-on* $\{\}\ r \longrightarrow r = \{\}$
using *lin-ord-not-empty*
by *metis*
ultimately have *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow \forall\ p' \in \{p'.$ *finite-profile* $V\ A\ p' \wedge (\forall\ v'.\ v' \notin V \longrightarrow p'\ v' = \{\})\}.$
$\forall\ v.\ p'\ v = \{\}$
by *blast*
hence *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow \{p'.$ *finite-profile* $V\ A\ p'\} = \{p'.\ \forall\ v \in V.\ p'\ v = \{\}\}$
using *lin-ord-not-empty lnear-order-on-empty*
unfolding *profile-def*
by (*metis* (*no-types*, *opaque-lifting*))
hence *finite* A $\wedge$ *finite* V $\wedge$ $A = \{\}$
$\Longrightarrow$ *all-profiles* $V\ A = \{p'.$ *finite-profile* $V\ A\ p'\}$
using *all-prof-eq*
by *simp*
moreover have *infinite* A $\vee$ *infinite* $V \Longrightarrow$ *all-profiles* $V\ A = \{\}$
by *simp*
moreover have *infinite* A $\vee$ *infinite* $V \Longrightarrow$
$\{p'.$ *finite-profile* $V\ A\ p' \wedge (\forall\ v'.\ v' \notin V \longrightarrow p'\ v' = \{\})\} = \{\}$

by *auto*
moreover have *infinite* $A \lor$ *infinite* $V \lor A = \{\}$
using *not-fin-empty*
by *simp*
ultimately show *all-profiles* $V\ A = \{p'.$ *finite-profile* $V\ A\ p'\}$
by *blast*
qed

5.4.4 Soundness

lemma (**in** *result*) $\mathcal{R}$*-sound*:
fixes
K :: ($'a$, $'v$, $'r$ *Result*) *Consensus-Class* **and**
d :: ($'a$, $'v$) *Election Distance*
shows *electoral-module* (*distance-*$\mathcal{R}$ d K)
proof (*unfold electoral-module-def*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile*
have $\mathcal{R}_{\mathcal{W}}$ $d\ K\ V\ A\ p \subseteq$ (*limit-set* A *UNIV*)
using $\mathcal{R}_{\mathcal{W}}$*.simps arg-min-subset*
by *metis*
hence *set-equals-partition* (*limit-set* A *UNIV*) (*distance-*$\mathcal{R}$ $d\ K\ V\ A\ p$)
by *auto*
moreover have *disjoint3* (*distance-*$\mathcal{R}$ $d\ K\ V\ A\ p$)
by *simp*
ultimately show *well-formed* A (*distance-*$\mathcal{R}$ $d\ K\ V\ A\ p$)
using *result-axioms*
unfolding *result-def*
by *simp*
qed

5.4.5 Inference Rules

lemma *is-arg-min-equal*:
fixes
f :: $'a \Rightarrow$ $'b$::*ord* **and**
g :: $'a \Rightarrow$ $'b$ **and**
S :: $'a$ *set* **and**
x :: $'a$
assumes $\forall\ x \in S.\ f\ x = g\ x$
shows *is-arg-min* f (λ $s.$ $s \in S$) x = *is-arg-min* g (λ $s.$ $s \in S$) x
proof (*unfold is-arg-min-def*, *cases* $x \in S$)
case *False*
thus $(x \in S \land (\nexists\ y.\ y \in S \land f\ y < f\ x)) = (x \in S \land (\nexists\ y.\ y \in S \land g\ y < g\ x))$
by *simp*
next
case *x-in-S*: *True*
thus $(x \in S \land (\nexists\ y.\ y \in S \land f\ y < f\ x)) = (x \in S \land (\nexists\ y.\ y \in S \land g\ y < g\ x))$

```
  proof (cases ∃ y. (λ s. s ∈ S) y ∧ f y < f x)
    case y: True
    then obtain y :: 'a where
      (λ s. s ∈ S) y ∧ f y < f x
      by metis
    hence (λ s. s ∈ S) y ∧ g y < g x
      using x-in-S assms
      by metis
    thus ?thesis
      using y
      by metis
  next
    case not-y: False
    have ¬ (∃ y. (λ s. s ∈ S) y ∧ g y < g x)
    proof (safe)
      fix y :: 'a
      assume
        y-in-S: y ∈ S and
        g-y-lt-g-x: g y < g x
      have f-eq-g-for-elems-in-S: ∀ a. a ∈ S ⟶ f a = g a
        using assms
        by simp
      hence g x = f x
        using x-in-S
        by presburger
      thus False
        using f-eq-g-for-elems-in-S g-y-lt-g-x not-y y-in-S
        by (metis (no-types))
    qed
    thus ?thesis
      using x-in-S not-y
      by simp
  qed
qed

lemma (in result) standard-distance-imp-equal-score:
  fixes
    d :: ('a, 'v) Election Distance and
    K :: ('a, 'v, 'r Result) Consensus-Class and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    w :: 'r
  assumes
    irr-non-V: non-voters-irrelevant d and
    std: standard d
  shows score d K (A, V, p) w = score-std d K (A, V, p) w
proof -
  have profile-perm-set:
```

all-profiles V A =
{p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'}
using *profile-permutation-set*
by *metis*
hence *eq-intersect*: $\mathcal{K}_\mathcal{E}$*-std* K w A V =
$\mathcal{K}_\mathcal{E}$ K w ∩ *Pair* A ‘ *Pair* V ‘ {p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'}
by *force*
have *inf-eq-inf-for-std-cons*:
Inf (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w)) =
Inf (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w ∩
Pair A ‘ *Pair* V ‘ {p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'}))
proof −
have ($\mathcal{K}_\mathcal{E}$ K w ∩ *Pair* A ‘ *Pair* V ‘ {p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'})
⊆ ($\mathcal{K}_\mathcal{E}$ K w)
by *simp*
hence *Inf* (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w)) ≤
Inf (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w ∩
Pair A ‘ *Pair* V ‘ {p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'}))
using *INF-superset-mono dual-order.refl*
by *metis*
moreover have *Inf* (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w)) ≥
Inf (d (A, V, p) ‘ ($\mathcal{K}_\mathcal{E}$ K w ∩
Pair A ‘ *Pair* V ‘ {p' :: ($'a$, $'v$) *Profile*. *finite-profile* V A p'}))
proof (*rule INF-greatest*)
let *?inf* = *Inf* (d (A, V, p) ‘
($\mathcal{K}_\mathcal{E}$ K w ∩ *Pair* A ‘ *Pair* V ‘ {p'. *finite-profile* V A p'}))
let *?compl* = ($\mathcal{K}_\mathcal{E}$ K w) −
($\mathcal{K}_\mathcal{E}$ K w ∩ *Pair* A ‘ *Pair* V ‘ {p'. *finite-profile* V A p'})
fix i :: ($'a$, $'v$) *Election*
assume *el*: i ∈ $\mathcal{K}_\mathcal{E}$ K w
have *in-intersect*: i ∈ ($\mathcal{K}_\mathcal{E}$ K w ∩ *Pair* A ‘ *Pair* V ‘ {p'. *finite-profile* V A
p'})
⟹ *?inf* ≤ d (A, V, p) i
using *Complete-Lattices.complete-lattice-class.INF-lower*
by *metis*
have i ∈ *?compl* ⟹ (V ≠ *fst* (*snd* i)
∨ A ≠ *fst* i
∨ ¬ *finite-profile* V A (*snd* (*snd* i)))
by *fastforce*
moreover have V ≠ *fst* (*snd* i) ⟹ d (A, V, p) i = ∞
using *std prod.collapse*
unfolding *standard-def*
by *metis*
moreover have A ≠ *fst* i ⟹ d (A, V, p) i = ∞
using *std prod.collapse*
unfolding *standard-def*
by *metis*
moreover have V = *fst* (*snd* i) ∧ A = *fst* i
∧ ¬ *finite-profile* V A (*snd* (*snd* i)) ⟶ *False*

using *el*
by *fastforce*
ultimately have
$i \in ?compl \Longrightarrow Inf\ (d\ (A,\ V,\ p)\ `$
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap Pair\ A\ `\ Pair\ V\ `\ \{p'.\ finite\text{-}profile\ V\ A\ p'\}))$
$\leq d\ (A,\ V,\ p)\ i$
using *ereal-less-eq*
by *metis*
thus $Inf\ (d\ (A,\ V,\ p)\ `$
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap$
$Pair\ A\ `\ Pair\ V\ `\ \{p'.\ finite\text{-}profile\ V\ A\ p'\}))$
$\leq d\ (A,\ V,\ p)\ i$
using *in-intersect el*
by *blast*
qed
ultimately show
$Inf\ (d\ (A,\ V,\ p)\ `\ \mathcal{K}_{\mathcal{E}}\ K\ w) =$
$Inf\ (d\ (A,\ V,\ p)\ `$
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap Pair\ A\ `\ Pair\ V\ `\ \{p'.\ finite\text{-}profile\ V\ A\ p'\}))$
by *simp*
qed
also have *inf-eq-min-for-std-cons*:
$\ldots = score\text{-}std\ d\ K\ (A,\ V,\ p)\ w$
proof $(cases\ \mathcal{K}_{\mathcal{E}}\text{-}std\ K\ w\ A\ V = \{\})$
case *True*
hence $Inf\ (d\ (A,\ V,\ p)\ `$
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap Pair\ A\ `\ Pair\ V\ `$
$\{p'.\ finite\text{-}profile\ V\ A\ p'\})) = \infty$
using *eq-intersect*
using *top-ereal-def*
by *simp*
also have $score\text{-}std\ d\ K\ (A,\ V,\ p)\ w = \infty$
using *True*
unfolding *Let-def*
by *simp*
finally show *?thesis*
by *simp*
next
case *False*
hence *fin*: $finite\ A \wedge finite\ V$
using *eq-intersect*
by *blast*
have $finite\ (d\ (A,\ V,\ p)\ `(\mathcal{K}_{\mathcal{E}}\text{-}std\ K\ w\ A\ V))$
proof −
have $\mathcal{K}_{\mathcal{E}}\text{-}std\ K\ w\ A\ V = (\mathcal{K}_{\mathcal{E}}\ K\ w) \cap$
$\{(A,\ V,\ p') \mid p'.\ finite\text{-}profile\ V\ A\ p'\}$
using *eq-intersect*
by *blast*
hence *subset*: $d\ (A,\ V,\ p)\ `(\mathcal{K}_{\mathcal{E}}\text{-}std\ K\ w\ A\ V) \subseteq$

$d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$
by *blast*
let *?finite-prof* $= \lambda\ p'\ v.$ (*if* $(v \in V)$ *then* $p'\ v$ *else* $\{\})$
have $\forall\ p'.$ *finite-profile* $V\ A\ p' \longrightarrow$
finite-profile $V\ A$ (*?finite-prof* p')
unfolding *If-def profile-def*
by *simp*
moreover have $\forall\ p'.\ (\forall\ v.\ v \notin V \longrightarrow$ *?finite-prof* $p'\ v = \{\})$
by *simp*
ultimately have
$\forall\ (A',\ V',\ p') \in \{(A',\ V',\ p').\ A' = A \wedge V' = V \wedge$ *finite-profile* $V\ A\ p'\}.$
$(A',\ V',$ *?finite-prof* $p') \in \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$
by *force*
moreover have $\forall\ p'.\ d\ (A,\ V,\ p)\ (A,\ V,\ p') = d\ (A,\ V,\ p)\ (A,\ V,$ *?finite-prof*
$p')$
using *irr-non-V*
unfolding *non-voters-irrelevant-def*
by *simp*
ultimately have
$\forall\ (A',\ V',\ p') \in \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}.$
$(\exists\ (X,\ Y,\ z) \in \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}.$
$d\ (A,\ V,\ p)\ (A',\ V',\ p') = d\ (A,\ V,\ p)\ (X,\ Y,\ z))$
by *fastforce*
hence $\forall\ (A',\ V',\ p') \in \{(A',\ V',\ p').\ A' = A \wedge V' = V \wedge$ *finite-profile* $V\ A$
$p'\}.$
$d\ (A,\ V,\ p)\ (A',\ V',\ p') \in$
$d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
by *fastforce*
hence *subset-2*: $d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$
$\subseteq d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
by *fastforce*
have $\forall\ (A',\ V',\ p') \in \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}.$
$(\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v))$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})$
using *fin*
unfolding *profile-def*
by *simp*
hence $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
$\subseteq \{(A,\ V,\ p') \mid p'.\ p' \in \{p'.\ (\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v))$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}\}$
by *blast*
moreover have *finite* $\{(A,\ V,\ p') \mid p'.\ p' \in \{p'.\ (\forall\ v \in V.$ *linear-order-on* A
$(p'\ v))$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}\}$

proof −
have $\{p'.\ (\forall\ v \in V.\ \textit{linear-order-on}\ A\ (p'\ v)) \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
$\subseteq \textit{all-profiles}\ V\ A \cap \{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = \{\}\}$
using *lin-order-pl-α fin*
by *fastforce*
moreover have $\textit{finite}\ (\textit{all-profiles}\ V\ A \cap \{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = \{\}\})$
using *fin fin-all-profs*
by *blast*
ultimately have $\textit{finite}\ \{p'.\ (\forall\ v \in V.\ \textit{linear-order-on}\ A\ (p'\ v))$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
using *rev-finite-subset*
by *blast*
thus *?thesis*
by *simp*
qed
ultimately have $\textit{finite}\ \{(A, V, p') \mid p'.\ \textit{finite-profile}\ V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
using *rev-finite-subset*
by *simp*
hence $\textit{finite}\ (d\ (A, V, p)\ \text{‘}\ \{(A, V, p') \mid p'.\ \textit{finite-profile}\ V\ A\ p'$
$\wedge\ (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\})$
by *simp*
hence $\textit{finite}\ (d\ (A, V, p)\ \text{‘}\ \{(A, V, p') \mid p'.\ \textit{finite-profile}\ V\ A\ p'\})$
using *subset-2 rev-finite-subset*
by *simp*
thus *?thesis*
using *subset rev-finite-subset*
by *blast*
qed
moreover have $d\ (A, V, p)\ \text{‘}\ (\mathcal{K}_{\mathcal{E}}\textit{-std}\ K\ w\ A\ V) \neq \{\}$
using *False*
by *simp*
ultimately have $\textit{Inf}\ (d\ (A, V, p)\ \text{‘}\ (\mathcal{K}_{\mathcal{E}}\textit{-std}\ K\ w\ A\ V)) = \textit{Min}\ (d\ (A, V, p)\ \text{‘}\ (\mathcal{K}_{\mathcal{E}}\textit{-std}\ K\ w\ A\ V))$
using *Min-Inf False*
by *metis*
also have $... = \textit{score-std}\ d\ K\ (A, V, p)\ w$
using *False*
by *simp*
also have $\textit{Inf}\ (d\ (A, V, p)\ \text{‘}\ (\mathcal{K}_{\mathcal{E}}\textit{-std}\ K\ w\ A\ V)) =$
$\textit{Inf}\ (d\ (A, V, p)\ \text{‘}\ (\mathcal{K}_{\mathcal{E}}\ K\ w\ \cap$
$\textit{Pair}\ A\ \text{‘}\ \textit{Pair}\ V\ \text{‘}\ \{p'.\ \textit{finite-profile}\ V\ A\ p'\}))$
using *eq-intersect*
by *simp*
ultimately show *?thesis*
by *simp*
qed
finally show $\textit{score}\ d\ K\ (A, V, p)\ w = \textit{score-std}\ d\ K\ (A, V, p)\ w$

by *simp*
qed

lemma (**in** *result*) *anonymous-distance-and-consensus-imp-rule-anonymity*:
fixes
d :: $('a, 'v)$ *Election Distance* **and**
K :: $('a, 'v, 'r$ *Result*$)$ *Consensus-Class*
assumes
d-anon: *distance-anonymity* d **and**
K-anon: *consensus-rule-anonymity* K
shows *anonymity* (*distance-*$\mathcal{R}$ d K)
proof (*unfold anonymity-def Let-def*, *safe*)
show *electoral-module* (*distance-*$\mathcal{R}$ d K)
using $\mathcal{R}$*-sound*
by *metis*
next
fix
A :: *'a set* **and**
A' :: *'a set* **and**
V :: *'v set* **and**
V' :: *'v set* **and**
p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile* **and**
π :: $'v \Rightarrow 'v$
assume
fin-A: *finite* A **and**
fin-V: *finite* V **and**
profile-p: *profile* V A p **and**
profile-q: *profile* V' A' q **and**
bij: *bij* π **and**
renamed: *rename* π $(A, V, p) = (A', V', q)$
have $A = A'$
using *bij renamed*
by *simp*
hence *eq-univ*: *limit-set* A *UNIV* = *limit-set* A' *UNIV*
by *simp*
hence $\mathcal{R}_{\mathcal{W}}$ d K V A p = $\mathcal{R}_{\mathcal{W}}$ d K V' A' q
proof −
have *dist-rename-inv*:
$\forall$ E::$('a, 'v)$ *Election*. d (A, V, p) E = d (A', V', q) (*rename* π E)
using *d-anon bij renamed surj-pair*
unfolding *distance-anonymity-def*
by *metis*
hence $\forall$ S::$('a, 'v)$ *Election set*.
(d (A, V, p) ‘ S) $\subseteq$ (d (A', V', q) ‘ (*rename* π ‘ S))
by *blast*
moreover have $\forall$ S::$('a, 'v)$ *Election set*.
((d (A', V', q) ‘ (*rename* π ‘ S)) $\subseteq$ (d (A, V, p) ‘ S))
proof (*clarify*)

fix
S :: $('a,\ 'v)$ *Election set* **and**
X :: $'a$ *set* **and**
X' :: $'a$ *set* **and**
Y :: $'v$ *set* **and**
Y' :: $'v$ *set* **and**
z :: $('a,\ 'v)$ *Profile* **and**
z' :: $('a,\ 'v)$ *Profile*
assume
$(X',\ Y',\ z') = \mathit{rename}\ \pi\ (X,\ Y,\ z)$ **and**
el: $(X,\ Y,\ z) \in S$
hence $d\ (A',\ V',\ q)\ (X',\ Y',\ z') = d\ (A,\ V,\ p)\ (X,\ Y,\ z)$
using *dist-rename-inv*
by *simp*
thus $d\ (A',\ V',\ q)\ (X',\ Y',\ z') \in d\ (A,\ V,\ p)$ ‘ S
using *el*
by *simp*
qed
ultimately have *eq-range*: $\forall\ S$::$('a,\ 'v)$ *Election set*.
$(d\ (A,\ V,\ p)$ ‘ $S) = (d\ (A',\ V',\ q)$ ‘ $(\mathit{rename}\ \pi$ ‘ $S))$
by *blast*
have $\forall\ w.\ \mathit{rename}\ \pi$ ‘ $(\mathcal{K}_{\mathcal{E}}\ K\ w) \subseteq (\mathcal{K}_{\mathcal{E}}\ K\ w)$
proof (*clarify*)
fix
w :: $'r$ **and**
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
V' :: $'v$ *set* **and**
p :: $('a,\ 'v)$ *Profile* **and**
p' :: $('a,\ 'v)$ *Profile*
assume
renamed: $(A',\ V',\ p') = \mathit{rename}\ \pi\ (A,\ V,\ p)$ **and**
consensus: $(A,\ V,\ p) \in \mathcal{K}_{\mathcal{E}}\ K\ w$
hence *cons*:
$(\mathit{consensus}\text{-}\mathcal{K}\ K)\ (A,\ V,\ p) \wedge \mathit{finite\text{-}profile}\ V\ A\ p \wedge \mathit{elect}\ (\mathit{rule}\text{-}\mathcal{K}\ K)\ V\ A\ p$
$= \{w\}$
by *simp*
hence *fin-img*: $\mathit{finite\text{-}profile}\ V'\ A'\ p'$
using *renamed bij rename.simps fst-conv rename-finite*
by *metis*
hence *cons-img*: $\mathit{consensus}\text{-}\mathcal{K}\ K\ (A',\ V',\ p') \wedge (\mathit{rule}\text{-}\mathcal{K}\ K\ V\ A\ p = \mathit{rule}\text{-}\mathcal{K}\ K$
$V'\ A'\ p')$
using *K-anon renamed bij cons*
unfolding *consensus-rule-anonymity-def Let-def*
by *simp*
hence $\mathit{elect}\ (\mathit{rule}\text{-}\mathcal{K}\ K)\ V'\ A'\ p' = \{w\}$
using *cons*
by *simp*

thus $(A', V', p') \in \mathcal{K}_{\mathcal{E}}\ K\ w$
  using *cons-img fin-img*
  by *simp*
qed
moreover have $\forall\ w.\ (\mathcal{K}_{\mathcal{E}}\ K\ w) \subseteq rename\ \pi\ `\ (\mathcal{K}_{\mathcal{E}}\ K\ w)$
proof (*clarify*)
  fix
    $w :: 'r$ **and**
    $A :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a, 'v)\ Profile$
  assume *consensus*: $(A, V, p) \in \mathcal{K}_{\mathcal{E}}\ K\ w$
  let $?inv = rename\ (the\text{-}inv\ \pi)\ (A, V, p)$
  have *inv-inv-id*: $the\text{-}inv\ (the\text{-}inv\ \pi) = \pi$
    using *the-inv-f-f bij bij-betw-imp-inj-on bij-betw-imp-surj*
      inj-on-the-inv-into surj-imp-inv-eq the-inv-into-onto
    by (*metis* (*no-types*, *opaque-lifting*))
  hence $?inv = (A, ((the\text{-}inv\ \pi)\ `\ V), p \circ (the\text{-}inv\ (the\text{-}inv\ \pi)))$
    by *simp*
  moreover have $(p \circ (the\text{-}inv\ (the\text{-}inv\ \pi))) \circ (the\text{-}inv\ \pi) = p$
    using *bij inv-inv-id*
    unfolding *bij-betw-def comp-def*
    by (*simp add*: *f-the-inv-into-f*)
  moreover have $\pi\ `\ (the\text{-}inv\ \pi)\ `\ V = V$
    using *bij the-inv-f-f bij-betw-def image-inv-into-cancel*
      surj-imp-inv-eq top-greatest
    by (*metis* (*no-types*, *opaque-lifting*))
  ultimately have *preimg*: $rename\ \pi\ ?inv = (A, V, p)$
    unfolding *Let-def*
    by *simp*
  moreover have $?inv \in \mathcal{K}_{\mathcal{E}}\ K\ w$
  proof –
    have *cons*:
      $(consensus\text{-}\mathcal{K}\ K)\ (A, V, p) \wedge finite\text{-}profile\ V\ A\ p \wedge elect\ (rule\text{-}\mathcal{K}\ K)\ V\ A$
$p = \{w\}$
      using *consensus*
      by *simp*
    moreover have *bij-inv*: $bij\ (the\text{-}inv\ \pi)$
      using *bij bij-betw-the-inv-into*
      by *metis*
     moreover have *fin-preimg*: $finite\text{-}profile\ (fst\ (snd\ ?inv))\ (fst\ ?inv)\ (snd$
$(snd\ ?inv))$
      using *bij-inv rename.simps fst-conv rename-finite cons*
      by *fastforce*
    ultimately have *cons-preimg*:
      $consensus\text{-}\mathcal{K}\ K\ ?inv\ \wedge$
        $(rule\text{-}\mathcal{K}\ K\ V\ A\ p = rule\text{-}\mathcal{K}\ K\ (fst\ (snd\ ?inv))\ (fst\ ?inv)\ (snd\ (snd$
$?inv)))$
      using *K-anon renamed bij cons*

```
        unfolding consensus-rule-anonymity-def Let-def
        by simp
      hence elect (rule-𝒦 K) (fst (snd ?inv)) (fst ?inv) (snd (snd ?inv)) = {w}
        using cons
        by simp
      thus ?thesis
        using cons-preimg fin-preimg
        by simp
      qed
      ultimately show (A, V, p) ∈ rename π ‘ 𝒦_ℰ K w
        using image-eqI
        by metis
  qed
  ultimately have ∀ w. (𝒦_ℰ K w) = rename π ‘ (𝒦_ℰ K w)
    by blast
  hence ∀ w. score d K (A, V, p) w = score d K (A', V', q) w
    using eq-range
    by simp
  hence arg-min-set (score d K (A, V, p)) (limit-set A UNIV)
        = arg-min-set (score d K (A', V', q)) (limit-set A' UNIV)
    using eq-univ
    by presburger
  thus ℛ_𝒲 d K V A p = ℛ_𝒲 d K V' A' q
    by simp
 qed
 thus distance-ℛ d K V A p = distance-ℛ d K V' A' q
  using eq-univ
  by simp
qed

end
```

5.5 Votewise Distance Rationalization

```
theory Votewise-Distance-Rationalization
 imports Distance-Rationalization
         Votewise-Distance
begin
```

A votewise distance rationalization of a voting rule is its distance rationalization with a distance function that depends on the submitted votes in a simple and a transparent manner by using a distance on individual orders and combining the components with a norm on R to n.

5.5.1 Common Rationalizations

```
fun swap-ℛ :: ('a, 'v::linorder, 'a Result) Consensus-Class ⇒
```

$('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
swap-$\mathcal{R}$ K = *$\mathcal{SCF}$-result.distance-$\mathcal{R}$* (*votewise-distance swap l-one*) K

5.5.2 Theorems

lemma *votewise-non-voters-irrelevant*:
fixes
d :: *'a Vote Distance* **and**
N :: *Norm*
shows *non-voters-irrelevant* (*votewise-distance d N*)
proof (*unfold non-voters-irrelevant-def*, *clarify*)
fix
A :: *'a set* **and**
V :: *'v::linorder set* **and**
p :: *('a, 'v) Profile* **and**
A' :: *'a set* **and**
V' :: *'v set* **and**
p' :: *('a, 'v) Profile* **and**
q :: *('a, 'v) Profile*
assume *coincide*: $\forall\ v \in V.\ p\ v = q\ v$
have $\forall\ i <$ *length* (*sorted-list-of-set* V). (*sorted-list-of-set* V)!$i \in V$
using *card-eq-0-iff not-less-zero nth-mem*
sorted-list-of-set.length-sorted-key-list-of-set
sorted-list-of-set.set-sorted-key-list-of-set
by *metis*
hence (*to-list* V p) = (*to-list* V q)
using *coincide length-map nth-equalityI to-list.simps*
by *auto*
thus *votewise-distance* d N (A, V, p) (A', V', p') =
votewise-distance d N (A, V, q) (A', V', p') $\wedge$
votewise-distance d N (A', V', p') (A, V, p) =
votewise-distance d N (A', V', p') (A, V, q)
unfolding *votewise-distance.simps*
by *presburger*
qed

lemma *swap-standard*: *standard* (*votewise-distance swap l-one*)
proof (*unfold standard-def*, *clarify*)
fix
A :: *'a set* **and**
V :: *'v::linorder set* **and**
p :: *('a, 'v) Profile* **and**
A' :: *'a set* **and**
V' :: *'v set* **and**
p' :: *('a, 'v) Profile*
assume *assms*: $V \neq V' \vee A \neq A'$
let *?l* = (λ *l1 l2*. (*map2* (λ q q'. *swap* (A, q) (A', q')) *l1 l2*))
have $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge$ *finite* $V \Longrightarrow \forall\ q\ q'.$ *swap* (A, q) (A', q') $= \infty$

by *simp*
hence $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge$ *finite* $V \Longrightarrow$
$\forall$ *l1 l2*. (*l1* $\neq$ [] $\wedge$ *l2* $\neq$ [] $\longrightarrow$ ($\forall$ $i <$ *length* (*?l l1 l2*). (*?l l1 l2*)!$i = \infty$))
by *simp*
moreover have $V = V' \wedge V \neq \{\} \wedge$ *finite* $V \Longrightarrow$ (*to-list* V p) $\neq$ [] $\wedge$ (*to-list* V' p') $\neq$ []
using *card-eq-0-iff length-map list.size*(*3*) *to-list.simps*
sorted-list-of-set.length-sorted-key-list-of-set
by *metis*
moreover have $\forall$ *l*. ($\exists$ $i <$ *length l*. *l*!$i = \infty$) $\longrightarrow$ *l-one l* $= \infty$
proof (*safe*)
fix
l :: *ereal list* **and**
i :: *nat*
assume
$i <$ *length l* **and**
l ! $i = \infty$
hence ($\sum$ $j <$ *length l*. $|l!j|$) $= \infty$
using *sum-Pinfty abs-ereal.simps*(*3*) *finite-lessThan lessThan-iff*
by *metis*
thus *l-one l* $= \infty$
by *auto*
qed
ultimately have $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge$ *finite* $V \Longrightarrow$
l-one (*?l* (*to-list* V p) (*to-list* V' p)) $= \infty$
using *length-greater-0-conv map-is-Nil-conv zip-eq-Nil-iff*
by *metis*
hence $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge$ *finite* $V \Longrightarrow$
votewise-distance swap l-one (A, V, p) (A', V', p') $= \infty$
by *simp*
moreover have $V \neq V' \Longrightarrow$ *votewise-distance swap l-one* (A, V, p) (A', V', p') $= \infty$
by *simp*
moreover have $A \neq A' \wedge V = \{\} \Longrightarrow$ *votewise-distance swap l-one* (A, V, p) (A', V', p') $= \infty$
by *simp*
moreover have *infinite* $V \Longrightarrow$ *votewise-distance swap l-one* (A, V, p) (A', V', p') $= \infty$
by *simp*
moreover have ($A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge$ *finite* V) $\vee$ *infinite* $V \vee$ ($A \neq A' \wedge V = \{\}$) $\vee$ $V \neq V'$
using *assms*
by *blast*
ultimately show *votewise-distance swap l-one* (A, V, p) (A', V', p') $= \infty$
by *fastforce*
qed

5.5.3 Equivalence Lemmas

```
type-synonym ('a, 'v) score-type = ('a, 'v) Election Distance ⇒
        ('a, 'v, 'a Result) Consensus-Class ⇒ ('a, 'v) Election ⇒ 'a ⇒ ereal

type-synonym ('a, 'v) dist-rat-type = ('a, 'v) Election Distance ⇒
        ('a, 'v, 'a Result) Consensus-Class ⇒ 'v set ⇒ 'a set ⇒ ('a, 'v) Profile
⇒ 'a set

type-synonym ('a, 'v) dist-rat-std-type = ('a, 'v) Election Distance ⇒
        ('a, 'v, 'a Result) Consensus-Class ⇒ ('a, 'v, 'a Result) Electoral-Module

type-synonym ('a, 'v) dist-type = ('a, 'v) Election Distance ⇒
        ('a, 'v, 'a Result) Consensus-Class ⇒ ('a, 'v, 'a Result) Electoral-Module

lemma equal-score-swap: (score::(('a, 'v::linorder) score-type)) (votewise-distance
swap l-one) =
        score-std (votewise-distance swap l-one)
  using votewise-non-voters-irrelevant swap-standard
      SCF-result.standard-distance-imp-equal-score
  by fast

lemma swap-R-code[code]: swap-R =
        (SCF-result.distance-R-std::(('a, 'v::linorder) dist-rat-std-type))
          (votewise-distance swap l-one)
proof −
  from equal-score-swap
  have
    ∀ K E a. (score::(('a, 'v::linorder) score-type))
                (votewise-distance swap l-one) K E a =
            score-std (votewise-distance swap l-one) K E a
    by metis
  hence ∀ K V A p. (SCF-result.R_W::(('a, 'v::linorder) dist-rat-type))
                    (votewise-distance swap l-one) K V A p =
                 SCF-result.R_W-std
                    (votewise-distance swap l-one) K V A p
    by (simp add: equal-score-swap)
  hence ∀ K V A p. (SCF-result.distance-R::(('a, 'v::linorder) dist-type))
                    (votewise-distance swap l-one) K V A p
                 = SCF-result.distance-R-std
                    (votewise-distance swap l-one) K V A p
    by fastforce
  thus ?thesis
    unfolding swap-R.simps
    by blast
qed

end
```

5.6 Symmetry in Distance-Rationalizable Rules

theory *Distance-Rationalization-Symmetry*
 imports *Distance-Rationalization*
begin

5.6.1 Minimizer function

fun *inf-dist* :: $'x$ *Distance* $\Rightarrow$ $'x$ *set* $\Rightarrow$ $'x \Rightarrow$ *ereal* **where**
 inf-dist d X a = *Inf* (*d a* ‘ *X*)

fun *closest-preimg-dist* :: ($'x \Rightarrow 'y$) $\Rightarrow$ $'x$ *set* $\Rightarrow$ $'x$ *Distance* $\Rightarrow$ $'x \Rightarrow 'y \Rightarrow$ *ereal* **where**
 closest-preimg-dist f $domain_f$ *d x y* = *inf-dist d* (*preimg f* $domain_f$ *y*) *x*

fun *minimizer* :: ($'x \Rightarrow 'y$) $\Rightarrow$ $'x$ *set* $\Rightarrow$ $'x$ *Distance* $\Rightarrow$ $'y$ *set* $\Rightarrow$ $'x \Rightarrow 'y$ *set* **where**
 minimizer f $domain_f$ *d Y x* = *arg-min-set* (*closest-preimg-dist f* $domain_f$ *d x*) *Y*

Auxiliary Lemmas

lemma *rewrite-arg-min-set*:
 fixes
  f :: $'x \Rightarrow 'y$::*linorder* **and**
  X :: $'x$ *set*
 shows *arg-min-set f X* = $\bigcup$ (*preimg f X* ‘ $\{y \in (f$ ‘ $X).\ \forall\ z \in f$ ‘ $X.\ y \leq z\}$)
proof (*safe*)
 fix *x* :: $'x$
 assume *arg-min*: $x \in$ *arg-min-set f X*
 hence *is-arg-min f* (λ *a*. $a \in X$) *x*
  by *simp*
 hence $\forall\ x' \in X.\ f\,x' \geq f\,x$
  by (*simp add*: *is-arg-min-linorder*)
 hence $\forall\ z \in f$ ‘ $X.\ f\,x \leq z$
  by *blast*
 moreover have $f\,x \in f$ ‘ X
  using *arg-min*
  by (*simp add*: *is-arg-min-linorder*)
 ultimately have $f\,x \in \{y \in f$ ‘ $X.\ \forall\ z \in f$ ‘ $X.\ y \leq z\}$
  by *blast*
 moreover have $x \in$ *preimg f X* (*f x*)
  using *arg-min*
  by (*simp add*: *is-arg-min-linorder*)
 ultimately show $x \in \bigcup$ (*preimg f X* ‘ $\{y \in (f$ ‘ $X).\ \forall\ z \in f$ ‘ $X.\ y \leq z\}$)
  by *blast*
next
 fix
  x :: $'x$ **and**
  x' :: $'x$ **and**
  b :: $'x$
 assume

same-img: $x \in$ *preimg* f X $(f\ x')$ **and**
min: $\forall\ z \in f$ ‘ X. $f\ x' \leq z$
hence $f\ x = f\ x'$
by *simp*
hence $\forall\ z \in f$ ‘ X. $f\ x \leq z$
using *min*
by *simp*
moreover have $x \in X$
using *same-img*
by *simp*
ultimately show $x \in$ *arg-min-set* f X
by (*simp add*: *is-arg-min-linorder*)
qed

Equivariance

lemma *restr-induced-rel*:
fixes
X :: *'x set* **and**
Y :: *'y set* **and**
Y' :: *'y set* **and**
φ :: (*'x*, *'y*) *binary-fun*
assumes $Y' \subseteq Y$
shows *Restr* (*rel-induced-by-action* X Y φ) Y' = *rel-induced-by-action* X Y' φ
using *assms*
by *auto*

theorem *group-act-invar-dist-and-equivar-f-imp-equivar-minimizer*:
fixes
f :: *'x* $\Rightarrow$ *'y* **and**
$domain_f$ :: *'x set* **and**
d :: *'x Distance* **and**
valid-img :: *'x* $\Rightarrow$ *'y set* **and**
X :: *'x set* **and**
G :: *'z monoid* **and**
φ :: (*'z*, *'x*) *binary-fun* **and**
ψ :: (*'z*, *'y*) *binary-fun*
defines *equivar-prop-set-valued* $\equiv$ *equivar-ind-by-act* (*carrier* G) X φ (*set-action* ψ)
assumes
action-φ: *group-action* G X φ **and**
group-act-res: *group-action* G *UNIV* ψ **and**
dom-in-X: $domain_f \subseteq X$ **and**
closed-domain:
closed-under-restr-rel (*rel-induced-by-action* (*carrier* G) X φ) X $domain_f$ **and**
equivar-img: *satisfies valid-img equivar-prop-set-valued* **and**
invar-d: *invariant-dist* d (*carrier* G) X φ **and**
equivar-f: *satisfies* f (*equivar-ind-by-act* (*carrier* G) $domain_f$ φ ψ)
shows *satisfies* (λ x. *minimizer* f $domain_f$ d (*valid-img* x) x) *equivar-prop-set-valued*

proof (*unfold equivar-ind-by-act-def equivar-prop-set-valued-def*,
simp del: *arg-min-set.simps*, *clarify*)
fix
x :: $'x$ **and**
g :: $'z$
assume
group-elem: $g \in$ *carrier* G **and**
x-in-X: $x \in X$ **and**
img-X: $\varphi\ g\ x \in X$
let $?x' = \varphi\ g\ x$
let $?c$ = *closest-preimg-dist f domain*$_f$ $d\ x$ **and**
$?c'$ = *closest-preimg-dist f domain*$_f$ $d\ ?x'$
have $\forall\ y.$ *preimg f domain*$_f$ $y \subseteq X$
using *dom-in-X*
by *fastforce*
hence *invar-dist-img*:
$\forall\ y.\ d\ x$ ‘ (*preimg f domain*$_f$ y) = $d\ ?x'$ ‘ ($\varphi\ g$ ‘ (*preimg f domain*$_f$ y))
using *x-in-X group-elem invar-dist-image invar-d action-*φ
by *metis*
have $\forall\ y.$ *preimg f domain*$_f$ ($\psi\ g\ y$) = ($\varphi\ g$) ‘ (*preimg f domain*$_f$ y)
using *group-act-equivar-f-imp-equivar-preimg*[*of* $G\ X\ \varphi\ \psi$ *domain*$_f$ $f\ g$] *assms group-elem*
by *blast*
hence $\forall\ y.\ d\ ?x'$ ‘ *preimg f domain*$_f$ ($\psi\ g\ y$) = $d\ ?x'$ ‘ ($\varphi\ g$) ‘ (*preimg f domain*$_f$ y)
by *presburger*
hence $\forall\ y.$ *Inf* ($d\ ?x'$ ‘ *preimg f domain*$_f$ ($\psi\ g\ y$)) = *Inf* ($d\ x$ ‘ *preimg f domain*$_f$ y)
using *invar-dist-img*
by *metis*
hence $\forall\ y.$ *inf-dist* d (*preimg f domain*$_f$ ($\psi\ g\ y$)) $?x'$ = *inf-dist* d (*preimg f domain*$_f$ y) x
by *simp*
hence $\forall\ y.$ *closest-preimg-dist f domain*$_f$ $d\ ?x'$ ($\psi\ g\ y$) =
closest-preimg-dist f domain$_f$ $d\ x\ y$
by *simp*
hence *comp*: *closest-preimg-dist f domain*$_f$ $d\ x$ = (*closest-preimg-dist f domain*$_f$ $d\ ?x'$) $\circ$ ($\psi\ g$)
by *auto*
hence $\forall\ Y\ \alpha.$ *preimg* $?c'$ ($\psi\ g$ ‘ Y) $\alpha = \psi\ g$ ‘ *preimg* $?c\ Y\ \alpha$
using *preimg-comp*
by *auto*
hence $\forall\ Y\ A.$ {*preimg* $?c'$ ($\psi\ g$ ‘ Y) α | $\alpha.\ \alpha \in A$} = {$\psi\ g$ ‘ *preimg* $?c\ Y\ \alpha$ | $\alpha.\ \alpha \in A$}
by *simp*
moreover have $\forall\ Y\ A.$ {$\psi\ g$ ‘ *preimg* $?c\ Y\ \alpha$ | $\alpha.\ \alpha \in A$} = {$\psi\ g$ ‘ β | $\beta.\ \beta \in$ *preimg* $?c\ Y$ ‘ A}
by *blast*
moreover have $\forall\ Y\ A.$ *preimg* $?c'$ ($\psi\ g$ ‘ Y) ‘ A = {*preimg* $?c'$ ($\psi\ g$ ‘ Y) α |

$\alpha.\ \alpha \in A\}$
  by *blast*
ultimately have
  $\forall\ Y\ A.\ preimg\ ?c'\ (\psi\ g\ `\ Y)\ `\ A = \{\psi\ g\ `\ \alpha \mid \alpha.\ \alpha \in preimg\ ?c\ Y\ `\ A\}$
  by *simp*
hence $\forall\ Y\ A.\ \bigcup\ (preimg\ ?c'\ (\psi\ g\ `\ Y)\ `\ A) = \bigcup\ \{\psi\ g\ `\ \alpha \mid \alpha.\ \alpha \in preimg\ ?c\ Y\ `\ A\}$
  by *simp*
moreover have $\forall\ Y\ A.\ \bigcup\ \{\psi\ g\ `\ \alpha \mid \alpha.\ \alpha \in preimg\ ?c\ Y\ `\ A\} = \psi\ g\ `\bigcup\ (preimg\ ?c\ Y\ `\ A)$
  by *blast*
ultimately have *eq-preimg-unions*:
  $\forall\ Y\ A.\ \bigcup\ (preimg\ ?c'\ (\psi\ g\ `\ Y)\ `\ A) = \psi\ g\ `\bigcup\ (preimg\ ?c\ Y\ `\ A)$
  by *simp*
have $\forall\ Y.\ ?c'\ `\ \psi\ g\ `\ Y = ?c\ `\ Y$
  using *comp*
  unfolding *image-comp*
  by *simp*
hence $\forall\ Y.\ \{\alpha \in ?c\ `\ Y.\ \forall\ \beta \in ?c\ `\ Y.\ \alpha \leq \beta\} =$
    $\{\alpha \in ?c'\ `\ \psi\ g\ `\ Y.\ \forall\ \beta \in ?c'\ `\ \psi\ g\ `\ Y.\ \alpha \leq \beta\}$
  by *simp*
hence
  $\forall\ Y.\ arg\text{-}min\text{-}set\ (closest\text{-}preimg\text{-}dist\ f\ domain_f\ d\ ?x')\ (\psi\ g\ `\ Y) =$
    $(\psi\ g)\ `\ (arg\text{-}min\text{-}set\ (closest\text{-}preimg\text{-}dist\ f\ domain_f\ d\ x)\ Y)$
  using *rewrite-arg-min-set*[*of ?c'*] *rewrite-arg-min-set*[*of ?c*] *eq-preimg-unions*
  by *presburger*
moreover have $valid\text{-}img\ (\varphi\ g\ x) = \psi\ g\ `\ valid\text{-}img\ x$
  using *equivar-img x-in-X group-elem img-X rewrite-equivar-ind-by-act*
  unfolding *equivar-prop-set-valued-def set-action.simps*
  by *metis*
ultimately show
  $arg\text{-}min\text{-}set\ (closest\text{-}preimg\text{-}dist\ f\ domain_f\ d\ (\varphi\ g\ x))\ (valid\text{-}img\ (\varphi\ g\ x)) =$
    $\psi\ g\ `\ arg\text{-}min\text{-}set\ (closest\text{-}preimg\text{-}dist\ f\ domain_f\ d\ x)\ (valid\text{-}img\ x)$
  by *presburger*
qed

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist*:
fixes
  $f :: 'x \Rightarrow 'y$ **and**
  $domain_f :: 'x\ set$ **and**
  $d :: 'x\ Distance$ **and**
  $rel :: 'x\ rel$
assumes
  r-refl: $refl\text{-}on\ domain_f\ (Restr\ rel\ domain_f)$ **and**
  tot-invar-d: *totally-invariant-dist d rel*
shows $satisfies\ (closest\text{-}preimg\text{-}dist\ f\ domain_f\ d)\ (Invariance\ rel)$
proof (*simp*, *safe*, *standard*)

fix
a :: *'x* **and**
b :: *'x* **and**
y :: *'y*
assume *rel*: $(a, b) \in rel$
have $\forall\ c \in domain_f.\ (c, c) \in rel$
using *r-refl*
unfolding *refl-on-def*
by *simp*
hence $\forall\ c \in domain_f.\ d\ a\ c = d\ b\ c$
using *rel tot-invar-d*
unfolding *rewrite-totally-invariant-dist*
by *blast*
thus *closest-preimg-dist f* $domain_f$ $d\ a\ y$ = *closest-preimg-dist f* $domain_f$ $d\ b\ y$
by *simp*
qed

lemma *refl-rel-and-tot-invar-dist-imp-invar-minimizer*:
fixes
f :: $'x \Rightarrow 'y$ **and**
$domain_f$ :: *'x set* **and**
d :: *'x Distance* **and**
rel :: *'x rel* **and**
img :: *'y set*
assumes
r-refl: *refl-on* $domain_f$ (*Restr rel* $domain_f$) **and**
tot-invar-d: *totally-invariant-dist d rel*
shows *satisfies* (*minimizer f* $domain_f$ *d img*) (*Invariance rel*)
proof –
have *satisfies* (*closest-preimg-dist f* $domain_f$ *d*) (*Invariance rel*)
using *r-refl tot-invar-d closest-dist-invar-under-refl-rel-and-tot-invar-dist*
by *simp*
moreover have *minimizer f* $domain_f$ *d img* =
(λ *x. arg-min-set x img*) $\circ$ (*closest-preimg-dist f* $domain_f$ *d*)
unfolding *comp-def*
by *auto*
ultimately show *?thesis*
using *invar-comp*
by *simp*
qed

theorem *group-act-invar-dist-and-invar-f-imp-invar-minimizer*:
fixes
f :: $'x \Rightarrow 'y$ **and**
$domain_f$ :: *'x set* **and**
d :: *'x Distance* **and**
img :: *'y set* **and**
X :: *'x set* **and**
G :: *'z monoid* **and**

φ :: ('z, 'x) *binary-fun*
defines
rel ≡ *rel-induced-by-action* (*carrier G*) *X* φ **and**
rel' ≡ *rel-induced-by-action* (*carrier G*) $domain_f$ φ
assumes
action-φ: *group-action G X* φ **and**
$domain_f \subseteq X$ **and**
closed-domain: *closed-under-restr-rel rel X* $domain_f$ **and**

invar-d: *invariant-dist d* (*carrier G*) *X* φ **and**
invar-f: *satisfies f* (*Invariance rel'*)
shows *satisfies* (*minimizer f* $domain_f$ *d img*) (*Invariance rel*)
proof −
let
?ψ = λ *g. id* **and**
?img = λ *x. img*
have *satisfies f* (*equivar-ind-by-act* (*carrier G*) $domain_f$ φ *?*ψ)
using *invar-f rewrite-invar-as-equivar*
unfolding *rel'-def*
by *blast*
moreover have *group-action G UNIV ?*ψ
using *const-id-is-group-act action-*φ
unfolding *group-action-def group-hom-def*
by *blast*
moreover have *satisfies ?img* (*equivar-ind-by-act* (*carrier G*) *X* φ (*set-action* *?*ψ))
unfolding *equivar-ind-by-act-def*
by *fastforce*
ultimately have
satisfies (λ *x. minimizer f* $domain_f$ *d* (*?img x*) *x*)
(*equivar-ind-by-act* (*carrier G*) *X* φ (*set-action ?*ψ))
using *assms group-act-invar-dist-and-equivar-f-imp-equivar-minimizer*[*of* *G X* φ *?*ψ $domain_f$ *?img d f*]
by *blast*
hence *satisfies* (*minimizer f* $domain_f$ *d img*)
(*equivar-ind-by-act* (*carrier G*) *X* φ (*set-action ?*ψ))
by *blast*
thus *?thesis*
unfolding *rel-def set-action.simps*
using *rewrite-invar-as-equivar image-id*
by *metis*
qed

5.6.2 Distance Rationalization as Minimizer

lemma $\mathcal{K}_\mathcal{E}$*-is-preimg*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r *Result*) *Consensus-Class* **and**

E :: ($'a$, $'v$) *Election* **and**
w :: $'r$
shows *preimg* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) $\{w\}$ = $\mathcal{K}_{\mathcal{E}}$ C w
proof −
have *preimg* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) $\{w\}$ =
$\{E \in$ *elections-*$\mathcal{K}$ C. (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) $E = \{w\}\}$
by *simp*
also have $\{E \in$ *elections-*$\mathcal{K}$ C. (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) $E = \{w\}\}$ =
$\{E \in$ *elections-*$\mathcal{K}$ C. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
by *simp*
also have
$\{E \in$ *elections-*$\mathcal{K}$ C. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$ =
elections-$\mathcal{K}$ C ∩ $\{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
by *blast*
also have
elections-$\mathcal{K}$ C ∩ $\{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
= $\mathcal{K}_{\mathcal{E}}$ C w
proof
show
elections-$\mathcal{K}$ C ∩ $\{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
⊆ $\mathcal{K}_{\mathcal{E}}$ C w
unfolding $\mathcal{K}_{\mathcal{E}}$.*simps*
by *force*
next
have
∀ $E \in \mathcal{K}_{\mathcal{E}}$ C w. $E \in \{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E)
(*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
unfolding $\mathcal{K}_{\mathcal{E}}$.*simps*
by *force*
hence ∀ $E \in \mathcal{K}_{\mathcal{E}}$ C w. $E \in$
elections-$\mathcal{K}$ C ∩ $\{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E) (*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
by *simp*
thus $\mathcal{K}_{\mathcal{E}}$ C w ⊆ *elections-*$\mathcal{K}$ C ∩ $\{E$. *elect* (*rule-*$\mathcal{K}$ C) (*voters-*$\mathcal{E}$ E)
(*alternatives-*$\mathcal{E}$ E) (*profile-*$\mathcal{E}$ E) = $\{w\}\}$
by *blast*
qed
finally show *preimg* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) $\{w\}$ = $\mathcal{K}_{\mathcal{E}}$ C w
by *simp*
qed

lemma *score-is-closest-preimg-dist*:
fixes
d :: ($'a$, $'v$) *Election Distance* **and**

C :: $('a, 'v, 'r$ *Result*) *Consensus-Class* **and**
E :: $('a, 'v)$ *Election* **and**
w :: $'r$
shows *score* d C E w = *closest-preimg-dist* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d E $\{w\}$
proof −
have *score* d C E w = *Inf* (d E ‘ ($\mathcal{K}_\mathcal{E}$ C w))
by *simp*
also have $\mathcal{K}_\mathcal{E}$ C w = *preimg* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) $\{w\}$
using $\mathcal{K}_\mathcal{E}$*-is-preimg*
by *metis*
also have *Inf* (d E ‘ (*preimg* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) $\{w\}$))
= *closest-preimg-dist* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d E $\{w\}$
by *simp*
finally show *?thesis*
by *simp*
qed

lemma (**in** *result*) $\mathcal{R}_\mathcal{W}$*-is-minimizer*:
fixes
d :: $('a, 'v)$ *Election Distance* **and**
C :: $('a, 'v, 'r$ *Result*) *Consensus-Class*
shows *fun*$_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ d C) =
(λ E. $\bigcup$ (*minimizer* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d
(*singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*)) E))
proof
fix E :: $('a, 'v)$ *Election*
let *?min* = (*minimizer* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d
(*singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*)) E)
have *?min* = *arg-min-set*
(*closest-preimg-dist* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d E)
(*singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*))
by *simp*
also have
... = *singleton-set-system* (*arg-min-set* (*score* d C E) (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*))
proof (*safe*)
fix R :: $'r$ *set*
assume
min: $R \in$ *arg-min-set*
(*closest-preimg-dist* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule*-$\mathcal{K}$ C)) (*elections*-$\mathcal{K}$ C) d E)
(*singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*))
hence $R \in$ *singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*)
using *arg-min-subset subsetD*
by (*metis* (*no-types*, *lifting*))
then obtain r :: $'r$ **where**
res-singleton: $R = \{r\}$ **and**
r-in-lim-set: $r \in$ *limit-set* (*alternatives*-$\mathcal{E}$ E) *UNIV*

by *auto*
have $\nexists$ R'. $R' \in$ *singleton-set-system* (*limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*) $\wedge$
closest-preimg-dist (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E R'
$<$ *closest-preimg-dist* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E R
using *min arg-min-set.simps is-arg-min-def CollectD*
by (*metis* (*mono-tags*, *lifting*))
hence $\nexists$ r'. $r' \in$ *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV* $\wedge$
closest-preimg-dist (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E $\{r'\}$
$<$ *closest-preimg-dist* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E
$\{r\}$
using *res-singleton*
by *auto*
hence $\nexists$ r'. $r' \in$ *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV* $\wedge$ *score* d C E r' $<$ *score*
d C E r
using *score-is-closest-preimg-dist*
by *metis*
hence $r \in$ *arg-min-set* (*score* d C E) (*limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*)
using *r-in-lim-set arg-min-set.simps is-arg-min-def CollectI*
by *metis*
thus $R \in$ *singleton-set-system* (*arg-min-set* (*score* d C E) (*limit-set* (*alternatives-$\mathcal{E}$*
E) *UNIV*))
using *res-singleton*
by *simp*
next
fix R :: *'r set*
assume
$R \in$ *singleton-set-system* (*arg-min-set* (*score* d C E) (*limit-set* (*alternatives-$\mathcal{E}$*
E) *UNIV*))
then obtain r :: *'r* **where**
res-singleton: $R = \{r\}$ **and**
r-min-lim-set: $r \in$ *arg-min-set* (*score* d C E) (*limit-set* (*alternatives-$\mathcal{E}$* E)
UNIV)
by *auto*
hence $\nexists$ r'. $r' \in$ *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV* $\wedge$ *score* d C E r' $<$ *score*
d C E r
using *CollectD arg-min-set.simps is-arg-min-def*
by *metis*
hence $\nexists$ r'. $r' \in$ *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV* $\wedge$
closest-preimg-dist (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E $\{r'\}$
$<$ *closest-preimg-dist* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d E
$\{r\}$
using *score-is-closest-preimg-dist*
by *metis*
moreover have $\forall$ $R' \in$ *singleton-set-system* (*limit-set* (*alternatives-$\mathcal{E}$* E)
UNIV).
$\exists$ $r' \in$ *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*. $R' = \{r'\}$
by *auto*
ultimately have $\nexists$ R'. $R' \in$ *singleton-set-system* (*limit-set* (*alternatives-$\mathcal{E}$* E)
UNIV) $\wedge$

closest-preimg-dist (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E R′*
< *closest-preimg-dist* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E R*
using *res-singleton*
by *auto*
moreover have *R* ∈ *singleton-set-system* (*limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)
using *r-min-lim-set res-singleton arg-min-subset*
by *fastforce*
ultimately show *R* ∈ *arg-min-set*
(*closest-preimg-dist* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E*)
(*singleton-set-system* (*limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*))
using *arg-min-set.simps is-arg-min-def CollectI*
by (*metis* (*mono-tags*, *lifting*))
qed
also have (*arg-min-set* (*score d C E*) (*limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)) =
$fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*
by *simp*
finally have ⋃ *?min* = ⋃ (*singleton-set-system* ($fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*))
by *presburger*
thus $fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E* = ⋃ *?min*
using *un-left-inv-singleton-set-system*
by *auto*
qed

Invariance

theorem (**in** *result*) *tot-invar-dist-imp-invar-dr-rule*:
fixes
d :: (*′a*, *′v*) *Election Distance* **and**
C :: (*′a*, *′v*, *′r Result*) *Consensus-Class* **and**
rel :: (*′a*, *′v*) *Election rel*
assumes
r-refl: *refl-on* (*elections-*$\mathcal{K}$ *C*) (*Restr rel* (*elections-*$\mathcal{K}$ *C*)) **and**
tot-invar-d: *totally-invariant-dist d rel* **and**
invar-res: *satisfies* (λ *E*. *limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) (*Invariance rel*)
shows *satisfies* ($fun_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ *d C*)) (*Invariance rel*)
proof −
let *?min* = λ *E*. ⋃ ∘ (*minimizer* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
(*singleton-set-system* (*limit-set* (*alternatives-*$\mathcal{E}$ *E*)
UNIV)))
have ∀ *E*. *satisfies* (*?min E*) (*Invariance rel*)
using *r-refl tot-invar-d invar-comp*
refl-rel-and-tot-invar-dist-imp-invar-minimizer[*of*
elections-$\mathcal{K}$ *C rel d elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)]
by *blast*
moreover have *satisfies ?min* (*Invariance rel*)
using *invar-res*
by *auto*
ultimately have *satisfies* (λ *E*. *?min E E*) (*Invariance rel*)
using *invar-parameterized-fun*[*of ?min rel*]

by *blast*
also have (λ *E*. *?min E E*) = $fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)
using $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*
unfolding *comp-def* $fun_{\mathcal{E}}$*.simps*
by *metis*
finally have *invar*-$\mathcal{R}_{\mathcal{W}}$: *satisfies* ($fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)) (*Invariance rel*)
by *simp*
hence *satisfies* (λ *E*. *limit-set* (*alternatives*-$\mathcal{E}$ *E*) *UNIV* − $fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*) (*Invariance rel*)
using *invar-res*
by *fastforce*
thus *satisfies* ($fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*)) (*Invariance rel*)
using *invar*-$\mathcal{R}_{\mathcal{W}}$
by *auto*
qed

theorem (**in** *result*) *invar-dist-cons-imp-invar-dr-rule*:
fixes
d :: (*'a*, *'v*) *Election Distance* **and**
C :: (*'a*, *'v*, *'r Result*) *Consensus-Class* **and**
G :: *'x monoid* **and**
φ :: (*'x*, (*'a*, *'v*) *Election*) *binary-fun* **and**
B :: (*'a*, *'v*) *Election set*
defines
rel ≡ *rel-induced-by-action* (*carrier G*) *B* φ **and**
rel' ≡ *rel-induced-by-action* (*carrier G*) (*elections*-$\mathcal{K}$ *C*) φ
assumes
action-φ: *group-action G B* φ **and**
consensus-C-in-B: *elections*-$\mathcal{K}$ *C* ⊆ *B* **and**
closed-domain:
closed-under-restr-rel rel B (*elections*-$\mathcal{K}$ *C*) **and**
invar-res: *satisfies* (λ *E*. *limit-set* (*alternatives*-$\mathcal{E}$ *E*) *UNIV*) (*Invariance rel*) **and**
invar-d: *invariant-dist d* (*carrier G*) *B* φ **and**
invar-C-winners: *satisfies* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule*-$\mathcal{K}$ *C*)) (*Invariance rel'*)
shows *satisfies* ($fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*)) (*Invariance rel*)
proof −
let *?min* = λ *E*. $\bigcup$ ∘ (*minimizer* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule*-$\mathcal{K}$ *C*)) (*elections*-$\mathcal{K}$ *C*) *d* (*singleton-set-system* (*limit-set* (*alternatives*-$\mathcal{E}$ *E*) *UNIV*)))
have ∀ *E*. *satisfies* (*?min E*) (*Invariance rel*)
using *action*-φ *closed-domain consensus-C-in-B invar-d invar-C-winners group-act-invar-dist-and-invar-f-imp-invar-minimizer rel-def rel'-def invar-comp*
by (*metis* (*no-types*, *lifting*))
moreover have *satisfies ?min* (*Invariance rel*)
using *invar-res*
by *auto*
ultimately have *satisfies* (λ *E*. *?min E E*) (*Invariance rel*)

using *invar-parameterized-fun*[*of ?min rel*]
by *blast*
also have $(\lambda\ E.\ ?min\ E\ E) = fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C)$
using $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*
unfolding *comp-def* $fun_{\mathcal{E}}$*.simps*
by *metis*
finally have *invar-*$\mathcal{R}_{\mathcal{W}}$: *satisfies* $(fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C))$ (*Invariance rel*)
by *simp*
hence *satisfies* (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV* −
$fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C)\ E)$ (*Invariance rel*)
using *invar-res*
by *fastforce*
thus *satisfies* ($fun_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ *d C*)) (*Invariance rel*)
using *invar-*$\mathcal{R}_{\mathcal{W}}$
by *simp*
qed

Equivariance

theorem (**in** *result*) *invar-dist-equivar-cons-imp-equivar-dr-rule*:
fixes
d :: (*'a*, *'v*) *Election Distance* **and**
C :: (*'a*, *'v*, *'r Result*) *Consensus-Class* **and**
G :: *'x monoid* **and**
φ :: (*'x*, (*'a*, *'v*) *Election*) *binary-fun* **and**
ψ :: (*'x*, *'r*) *binary-fun* **and**
B :: (*'a*, *'v*) *Election set*
defines
rel ≡ *rel-induced-by-action* (*carrier G*) *B* φ **and**
rel' ≡ *rel-induced-by-action* (*carrier G*) (*elections-*$\mathcal{K}$ *C*) φ **and**
equivar-prop ≡
equivar-ind-by-act (*carrier G*) (*elections-*$\mathcal{K}$ *C*) φ (*set-action* ψ) **and**
equivar-prop-global-set-valued ≡
equivar-ind-by-act (*carrier G*) *B* φ (*set-action* ψ) **and**
equivar-prop-global-result-valued ≡
equivar-ind-by-act (*carrier G*) *B* φ (*result-action* ψ)
assumes
action-φ: *group-action G B* φ **and**
group-act-res: *group-action G UNIV* ψ **and**
cons-elect-set: *elections-*$\mathcal{K}$ *C* ⊆ *B* **and**
closed-domain: *closed-under-restr-rel rel B* (*elections-*$\mathcal{K}$ *C*) **and**
equivar-res:
satisfies (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) *equivar-prop-global-set-valued*
and
invar-d: *invariant-dist d* (*carrier G*) *B* φ **and**
equivar-C-winners: *satisfies* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) *equivar-prop*
shows *satisfies* ($fun_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ *d C*)) *equivar-prop-global-result-valued*
proof −
let *?min-E* = λ *E. minimizer* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*

$(singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV))$
E
let $?min = \lambda\ E.\ \bigcup \circ (minimizer\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ d$
$(singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)$
$UNIV)))$
let $?\psi' = set\text{-}action\ (set\text{-}action\ \psi)$
let $?equivar\text{-}prop\text{-}global\text{-}set\text{-}valued' = equivar\text{-}ind\text{-}by\text{-}act\ (carrier\ G)\ B\ \varphi\ ?\psi'$
have $\forall\ E\ g.\ g \in carrier\ G \longrightarrow E \in B \longrightarrow$
$singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ (\varphi\ g\ E))\ UNIV) =$
$\{\{r\}\ |\ r.\ r \in limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ (\varphi\ g\ E))\ UNIV\}$
by *simp*
moreover have
$\forall\ E\ g.\ g \in carrier\ G \longrightarrow E \in B \longrightarrow$
$limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ (\varphi\ g\ E))\ UNIV = \psi\ g\ `\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}$
$E)\ UNIV)$
using *equivar-res action-φ group-action.element-image*
unfolding *equivar-prop-global-set-valued-def equivar-ind-by-act-def*
by *fastforce*
ultimately have $\forall\ E\ g.\ g \in carrier\ G \longrightarrow E \in B \longrightarrow$
$singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ (\varphi\ g\ E))\ UNIV) =$
$\{\{r\}\ |\ r.\ r \in \psi\ g\ `\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\}$
by *simp*
moreover have $\forall\ E\ g.\ \{\{r\}\ |\ r.\ r \in \psi\ g\ `\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\}$
$= \{\psi\ g\ `\ \{r\}\ |\ r.\ r \in limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV\}$
by *blast*
moreover have $\forall\ E\ g.\ \{\psi\ g\ `\ \{r\}\ |\ r.\ r \in limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV\}$
$=$
$?\psi'\ g\ \{\{r\}\ |\ r.\ r \in limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV\}$
unfolding *set-action.simps*
by *blast*
ultimately have $satisfies\ (\lambda\ E.\ singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}$
$E)\ UNIV))$
$?equivar\text{-}prop\text{-}global\text{-}set\text{-}valued'$
using *rewrite-equivar-ind-by-act*[*of*
$\lambda\ E.\ singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ carrier\ G$
$B\ \varphi\ ?\psi'$]
by *force*
moreover have $group\text{-}action\ G\ UNIV\ (set\text{-}action\ \psi)$
unfolding *set-action.simps*
using *group-act-induces-set-group-act*[*of G UNIV ψ*] *group-act-res*
by *simp*
ultimately have $satisfies\ ?min\text{-}E\ ?equivar\text{-}prop\text{-}global\text{-}set\text{-}valued'$
using *action-φ invar-d cons-elect-set closed-domain equivar-C-winners*
group-act-invar-dist-and-equivar-f-imp-equivar-minimizer[*of*
$G\ B\ \varphi\ set\text{-}action\ \psi\ elections\text{-}\mathcal{K}\ C$
$\lambda\ E.\ singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)$
$d\ elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)$]
unfolding *rel′-def rel-def equivar-prop-def*
by *metis*

moreover have *satisfies* $\bigcup$ (*equivar-ind-by-act* (*carrier G*) *UNIV ?*ψ' (*set-action* ψ))
  using *equivar-union-under-img-act*[*of carrier G* ψ]
  by *simp*
ultimately have *satisfies* ($\bigcup$ $\circ$ *?min-E*) *equivar-prop-global-set-valued*
  unfolding *equivar-prop-global-set-valued-def*
  using *equivar-ind-by-act-comp*[*of ?min-E B UNIV*]
  by *simp*
moreover have (λ *E. ?min E E*) = $\bigcup$ $\circ$ *?min-E*
  unfolding *comp-def*
  by *simp*
ultimately have *satisfies* (λ *E. ?min E E*) *equivar-prop-global-set-valued*
  by *simp*
moreover have (λ *E. ?min E E*) = *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)
  using $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*
  unfolding *comp-def fun*$_{\mathcal{E}}$*.simps*
  by *metis*
ultimately have *equivar-*$\mathcal{R}_{\mathcal{W}}$: *satisfies* (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)) *equivar-prop-global-set-valued*
  by *simp*
moreover have $\forall$ *g* $\in$ *carrier G. bij* (ψ *g*)
  using *group-act-res*
  unfolding *bij-betw-def*
  by (*simp add*: *group-action.inj-prop group-action.surj-prop*)
ultimately have
  satisfies (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV* $-$ *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*)
    equivar-prop-global-set-valued
  using *equivar-res equivar-set-minus*
  unfolding *equivar-prop-global-set-valued-def equivar-ind-by-act-def set-action.simps*
  by *blast*
thus *satisfies* (*fun*$_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ *d C*)) *equivar-prop-global-result-valued*
  using *equivar-*$\mathcal{R}_{\mathcal{W}}$
  unfolding *equivar-prop-global-result-valued-def equivar-prop-global-set-valued-def*
    rewrite-equivar-ind-by-act
  by *simp*
qed

5.6.3 Symmetry Property Inference Rules

theorem (**in** *result*) *anon-dist-and-cons-imp-anon-dr*:
  fixes
    d :: ('*a*, '*v*) *Election Distance* **and**
    C :: ('*a*, '*v*, '*r Result*) *Consensus-Class*
  assumes
    anon-d: *distance-anonymity*' *valid-elections d* **and**
    anon-C: *consensus-rule-anonymity*' (*elections-*$\mathcal{K}$ *C*) *C* **and**
    closed-C: *closed-under-restr-rel* (*anonymity*$_{\mathcal{R}}$ *valid-elections*) *valid-elections* (*elections-*$\mathcal{K}$ *C*)
  shows *anonymity*' *valid-elections* (*distance-*$\mathcal{R}$ *d C*)
proof –

have $\forall\ \pi.\ \forall\ E \in$ *elections-$\mathcal{K}$ C*. *φ-anon* (*elections-$\mathcal{K}$ C*) π E = *φ-anon valid-elections* π E
using *cons-domain-valid extensional-continuation-subset*
unfolding *φ-anon.simps*
by *metis*
hence *rel-induced-by-action* (*carrier anonymity$_\mathcal{G}$*) (*elections-$\mathcal{K}$ C*) (*φ-anon valid-elections*) =
rel-induced-by-action (*carrier anonymity$_\mathcal{G}$*) (*elections-$\mathcal{K}$ C*) (*φ-anon* (*elections-$\mathcal{K}$ C*))
using *coinciding-actions-ind-equal-rel*[*of carrier anonymity$_\mathcal{G}$ elections-$\mathcal{K}$ C*]
by *metis*
hence *satisfies* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*))
(*Invariance* (*rel-induced-by-action*
(*carrier anonymity$_\mathcal{G}$*) (*elections-$\mathcal{K}$ C*) (*φ-anon valid-elections*)))
using *anon-C*
unfolding *consensus-rule-anonymity'.simps anonymity$_\mathcal{R}$.simps*
by *presburger*
thus *?thesis*
using *cons-domain-valid*[*of C*] *assms anonymous-group-action.group-action-axioms*
well-formed-res-anon invar-dist-cons-imp-invar-dr-rule[*of anonymity$_\mathcal{G}$*]
unfolding *distance-anonymity'.simps anonymity$_\mathcal{R}$.simps anonymity'.simps*
consensus-rule-anonymity'.simps
by *blast*
qed

theorem (**in** *result-properties*) *neutr-dist-and-cons-imp-neutr-dr*:
fixes
d :: ('*a*, '*v*) *Election Distance* **and**
C :: ('*a*, '*v*, '*b Result*) *Consensus-Class*
assumes
neutr-d: *distance-neutrality valid-elections d* **and**
neutr-C: *consensus-rule-neutrality* (*elections-$\mathcal{K}$ C*) *C* **and**
closed-C:
closed-under-restr-rel (*neutrality$_\mathcal{R}$ valid-elections*) *valid-elections* (*elections-$\mathcal{K}$ C*)
shows *neutrality valid-elections* (*distance-$\mathcal{R}$ d C*)
proof –
have $\forall\ \pi.\ \forall\ E \in$ *elections-$\mathcal{K}$ C*. *φ-neutr valid-elections* π E = *φ-neutr* (*elections-$\mathcal{K}$ C*) π E
using *cons-domain-valid extensional-continuation-subset*
unfolding *φ-neutr.simps*
by *metis*
hence *satisfies* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*))
(*equivar-ind-by-act* (*carrier neutrality$_\mathcal{G}$*) (*elections-$\mathcal{K}$ C*)
(*φ-neutr valid-elections*) (*set-action ψ-neutr*))
using *neutr-C equivar-ind-by-act-coincide*[*of carrier neutrality$_\mathcal{G}$*]
unfolding *consensus-rule-neutrality.simps*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*

using *neutr-d closed-C* φ*-neutr-act.group-action-axioms well-formed-res-neutr act-neutr*
cons-domain-valid[*of C*] *invar-dist-equivar-cons-imp-equivar-dr-rule*[*of neutrality*$_{\mathcal{G}}$
valid-elections φ*-neutr valid-elections*]
by *simp*
qed

theorem *reversal-sym-dist-and-cons-imp-reversal-sym-dr*:
fixes
d :: (*'a*, *'c*) *Election Distance* **and**
C :: (*'a*, *'c*, *'a rel Result*) *Consensus-Class*
assumes
rev-sym-d: *distance-reversal-symmetry valid-elections d* **and**
rev-sym-C: *consensus-rule-reversal-symmetry* (*elections-*$\mathcal{K}$ *C*) *C* **and**
closed-C: *closed-under-restr-rel* (*reversal*$_{\mathcal{R}}$ *valid-elections*) *valid-elections* (*elections-*$\mathcal{K}$ *C*)
shows *reversal-symmetry valid-elections* ($\mathcal{SWF}$*-result.distance-*$\mathcal{R}$ *d C*)
proof −
have $\forall$ π. $\forall$ *E* $\in$ *elections-*$\mathcal{K}$ *C*. φ*-rev valid-elections* π *E* = φ*-rev* (*elections-*$\mathcal{K}$ *C*) π *E*
using *cons-domain-valid extensional-continuation-subset*
unfolding φ*-rev.simps*
by *metis*
hence *satisfies* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*))
(*equivar-ind-by-act* (*carrier reversal*$_{\mathcal{G}}$) (*elections-*$\mathcal{K}$ *C*)
(φ*-rev valid-elections*) (*set-action* ψ*-rev*))
using *rev-sym-C equivar-ind-by-act-coincide*[*of carrier reversal*$_{\mathcal{G}}$]
unfolding *consensus-rule-reversal-symmetry.simps*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
using *cons-domain-valid rev-sym-d closed-C* φ*-rev-act.group-action-axioms*
ψ*-rev-act.group-action-axioms* φ*-*ψ*-rev-well-formed*
$\mathcal{SWF}$*-result.invar-dist-equivar-cons-imp-equivar-dr-rule*[*of*
reversal$_{\mathcal{G}}$ *valid-elections* φ*-rev valid-elections* ψ*-rev C d*]
unfolding *distance-reversal-symmetry.simps reversal-symmetry-def reversal*$_{\mathcal{R}}$*.simps*
by *metis*
qed

theorem (**in** *result*) *tot-hom-dist-imp-hom-dr*:
fixes
d :: (*'a*, *nat*) *Election Distance* **and**
C :: (*'a*, *nat*, *'r Result*) *Consensus-Class*
assumes *distance-homogeneity finite-voter-elections d*
shows *homogeneity finite-voter-elections* (*distance-*$\mathcal{R}$ *d C*)
proof −
have *Restr* (*homogeneity*$_{\mathcal{R}}$ *finite-voter-elections*) (*elections-*$\mathcal{K}$ *C*) = *homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{K}$ *C*)
using *cons-domain-finite*[*of C*]

unfolding *homogeneity*$_{\mathcal{R}}$*.simps finite-voter-elections-def*
by *blast*
hence *refl-on* (*elections-*$\mathcal{K}$ *C*) (*Restr* (*homogeneity*$_{\mathcal{R}}$ *finite-voter-elections*) (*elections-*$\mathcal{K}$ *C*))
using *refl-homogeneity*$_{\mathcal{R}}$[*of elections-*$\mathcal{K}$ *C*] *cons-domain-finite*[*of C*]
by *presburger*
moreover have
satisfies (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)
(*Invariance* (*homogeneity*$_{\mathcal{R}}$ *finite-voter-elections*))
using *well-formed-res-homogeneity*
by *simp*
ultimately show *?thesis*
using *assms tot-invar-dist-imp-invar-dr-rule* [*of C homogeneity*$_{\mathcal{R}}$ *finite-voter-elections d*]
unfolding *distance-homogeneity-def homogeneity.simps*
by *metis*
qed

theorem (**in** *result*) *tot-hom-dist-imp-hom-dr'*:
fixes
d :: (*'a*, *'v*::*linorder*) *Election Distance* **and**
C :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
assumes *distance-homogeneity' finite-voter-elections d*
shows *homogeneity' finite-voter-elections* (*distance-*$\mathcal{R}$ *d C*)
proof −
have *Restr* (*homogeneity*$_{\mathcal{R}}$*' finite-voter-elections*) (*elections-*$\mathcal{K}$ *C*)
= *homogeneity*$_{\mathcal{R}}$*'* (*elections-*$\mathcal{K}$ *C*)
using *cons-domain-finite*
unfolding *homogeneity*$_{\mathcal{R}}$*'.simps finite-voter-elections-def*
by *blast*
hence *refl-on* (*elections-*$\mathcal{K}$ *C*) (*Restr* (*homogeneity*$_{\mathcal{R}}$*' finite-voter-elections*) (*elections-*$\mathcal{K}$ *C*))
using *refl-homogeneity*$_{\mathcal{R}}$*'*[*of elections-*$\mathcal{K}$ *C*] *cons-domain-finite*[*of C*]
by *presburger*
moreover have
satisfies (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)
(*Invariance* (*homogeneity*$_{\mathcal{R}}$*' finite-voter-elections*))
using *well-formed-res-homogeneity'*
by *simp*
ultimately show *?thesis*
using *assms tot-invar-dist-imp-invar-dr-rule*
unfolding *distance-homogeneity'-def homogeneity'.simps*
by *blast*
qed

5.6.4 Further Properties

fun *decisiveness* :: (*'a*, *'v*) *Election set* ⇒ (*'a*, *'v*) *Election Distance* ⇒
(*'a*, *'v*, *'r Result*) *Electoral-Module* ⇒ *bool* **where**

$decisiveness\ X\ d\ m =$
$(\nexists\ E.\ E \in X \wedge (\exists\ \delta > 0.\ \forall\ E' \in X.\ d\ E\ E' < \delta \longrightarrow card\ (elect\text{-}r\ (fun_{\mathcal{E}}\ m\ E')) > 1))$

end

5.7 Distance Rationalization on Election Quotients

theory *Quotient-Distance-Rationalization*
imports *Quotient-Module*
Distance-Rationalization-Symmetry
begin

5.7.1 Quotient Distances

fun $dist_{\mathcal{Q}}$ :: $'x\ Distance \Rightarrow\ 'x\ set\ Distance$ **where**
$dist_{\mathcal{Q}}\ d\ A\ B = (if\ (A = \{\} \wedge B = \{\})\ then\ 0\ else$
$(if\ (A = \{\} \vee B = \{\})\ then\ \infty\ else$
$\pi_{\mathcal{Q}}\ (tup\ d)\ (A \times B)))$

fun *relation-paths* :: $'x\ rel \Rightarrow\ 'x\ list\ set$ **where**
$relation\text{-}paths\ r = \{p.\ \exists\ k.\ (length\ p = 2 * k \wedge (\forall\ i < k.\ (p!(2 * i),\ p!(2 * i + 1)) \in r))\}$

fun *admissible-paths* :: $'x\ rel \Rightarrow\ 'x\ set \Rightarrow\ 'x\ set \Rightarrow\ 'x\ list\ set$ **where**
$admissible\text{-}paths\ r\ X\ Y = \{x\#p@[y]\ |\ x\ y\ p.\ x \in X \wedge y \in Y \wedge p \in relation\text{-}paths\ r\}$

fun *path-length* :: $'x\ list \Rightarrow\ 'x\ Distance \Rightarrow ereal$ **where**
$path\text{-}length\ []\ d = 0\ |$
$path\text{-}length\ [x]\ d = 0\ |$
$path\text{-}length\ (x\#y\#xs)\ d = d\ x\ y + path\text{-}length\ xs\ d$

fun *quotient-dist* :: $'x\ rel \Rightarrow\ 'x\ Distance \Rightarrow\ 'x\ set\ Distance$ **where**
$quotient\text{-}dist\ r\ d\ A\ B = Inf\ (\bigcup\ \{\{path\text{-}length\ p\ d\ |\ p.\ p \in admissible\text{-}paths\ r\ A\ B\}\})$

fun $inf\text{-}dist_{\mathcal{Q}}$ :: $'x\ Distance \Rightarrow\ 'x\ set\ Distance$ **where**
$inf\text{-}dist_{\mathcal{Q}}\ d\ A\ B = Inf\ \{d\ a\ b\ |\ a\ b.\ a \in A \wedge b \in B\}$

fun *simple* :: $'x\ rel \Rightarrow\ 'x\ set \Rightarrow\ 'x\ Distance \Rightarrow bool$ **where**
$simple\ r\ X\ d =$
$(\forall\ A \in X\ //\ r.\ (\exists\ a \in A.\ \forall\ B \in X\ //\ r.\ inf\text{-}dist_{\mathcal{Q}}\ d\ A\ B = Inf\ \{d\ a\ b\ |\ b.\ b \in B\}))$
— We call a distance simple with respect to a relation if for all relation classes, there is an a in A minimizing the infimum distance between A and all B so that the infimum distance between these sets coincides with the infimum distance over

all b in B for fixed a.

fun *product-rel'* :: $'x$ *rel* $\Rightarrow$ $('x * 'x)$ *rel* **where**
product-rel' $r = \{(p_1, p_2).\ ((fst\ p_1, fst\ p_2) \in r \wedge snd\ p_1 = snd\ p_2) \vee$
$((snd\ p_1, snd\ p_2) \in r \wedge fst\ p_1 = fst\ p_2)\}$

Auxiliary Lemmas

lemma *tot-dist-invariance-is-congruence*:
fixes
d :: $'x$ *Distance* **and**
r :: $'x$ *rel*
shows $(totally\text{-}invariant\text{-}dist\ d\ r) = (tup\ d\ respects\ (product\text{-}rel\ r))$
unfolding *totally-invariant-dist.simps satisfies.simps congruent-def*
by *blast*

lemma *product-rel-helper*:
fixes
r :: $'x$ *rel* **and**
X :: $'x$ *set*
shows
trans-imp: *Relation.trans* $r \Longrightarrow$ *Relation.trans* $(product\text{-}rel\ r)$ **and**
refl-imp: *refl-on* $X\ r \Longrightarrow$ *refl-on* $(X \times X)\ (product\text{-}rel\ r)$ **and**
sym: *sym-on* $X\ r \Longrightarrow$ *sym-on* $(X \times X)\ (product\text{-}rel\ r)$
unfolding *Relation.trans-def refl-on-def sym-on-def product-rel.simps*
by *auto*

theorem *dist-pass-to-quotient*:
fixes
d :: $'x$ *Distance* **and**
r :: $'x$ *rel* **and**
X :: $'x$ *set*
assumes
equiv-X-r: *equiv* $X\ r$ **and**
tot-inv-dist-d-r: *totally-invariant-dist* $d\ r$
shows $\forall\ A\ B.\ A \in X\ //\ r \wedge B \in X\ //\ r \longrightarrow (\forall\ a\ b.\ a \in A \wedge b \in B \longrightarrow dist_{\mathcal{Q}}$
$d\ A\ B = d\ a\ b)$
proof (*safe*)
fix
A :: $'x$ *set* **and**
B :: $'x$ *set* **and**
a :: $'x$ **and**
b :: $'x$
assume
a-in-A: $a \in A$ **and**
$A \in X\ //\ r$
moreover with *equiv-X-r quotient-eq-iff*

have $(a,\ a) \in r$
  by *metis*
moreover with *equiv-X-r*
have *a-in-X*: $a \in X$
  using *equiv-class-eq-iff*
  by *metis*
ultimately have *A-eq-r-a*: $A = r \text{ `` } \{a\}$
  using *equiv-X-r quotient-eq-iff quotientI*
  by *fast*
assume
  b-in-B: $b \in B$ **and**
  $B \in X\ //\ r$
moreover with *equiv-X-r quotient-eq-iff*
have $(b,\ b) \in r$
  by *metis*
moreover with *equiv-X-r*
have *b-in-X*: $b \in X$
  using *equiv-class-eq-iff*
  by *metis*
ultimately have *B-eq-r-b*: $B = r \text{ `` } \{b\}$
  using *equiv-X-r quotient-eq-iff quotientI*
  by *fast*
from *A-eq-r-a B-eq-r-b a-in-X b-in-X*
have $A \times B \in (X \times X)\ //\ (\textit{product-rel}\ r)$
  unfolding *quotient-def*
  by *fastforce*
moreover have $\textit{equiv}\ (X \times X)\ (\textit{product-rel}\ r)$
  using *equiv-X-r product-rel-helper UNIV-Times-UNIV equivE equivI*
  by *metis*
moreover have $\textit{tup}\ d\ \textit{respects}\ (\textit{product-rel}\ r)$
  using *tot-inv-dist-d-r tot-dist-invariance-is-congruence*
  by *metis*
ultimately show $\textit{dist}_{\mathcal{Q}}\ d\ A\ B = d\ a\ b$
  unfolding $\textit{dist}_{\mathcal{Q}}\textit{.simps}$
  using *pass-to-quotient a-in-A b-in-B*
  by *fastforce*
qed

lemma *relation-paths-subset*:
  fixes
    $n :: \textit{nat}$ **and**
    $p :: \textit{'x list}$ **and**
    $r :: \textit{'x rel}$ **and**
    $X :: \textit{'x set}$
  assumes $r \subseteq X \times X$
  shows $\forall\ p.\ p \in \textit{relation-paths}\ r \longrightarrow (\forall\ i < \textit{length}\ p.\ p!i \in X)$
proof (*safe*)
  fix
    $p :: \textit{'x list}$ **and**

i :: *nat*
assume
p ∈ *relation-paths r*
then obtain *k* :: *nat* **where**
length p = *2* ∗ *k* **and**
rel: ∀ *i* < *k*. (*p*!(*2* ∗ *i*), *p*!(*2* ∗ *i* + *1*)) ∈ *r*
by *auto*
moreover obtain *k′* :: *nat* **where**
i-cases: *i* = *2* ∗ *k′* ∨ *i* = *2* ∗ *k′* + *1*
using *diff-Suc-1 even-Suc oddE odd-two-times-div-two-nat*
by *metis*
moreover assume *i* < *length p*
ultimately have *k′* < *k*
by *linarith*
thus *p*!*i* ∈ *X*
using *assms rel i-cases*
by *blast*
qed

lemma *admissible-path-len*:
fixes
d :: *′x Distance* **and**
r :: *′x rel* **and**
X :: *′x set* **and**
a :: *′x* **and**
b :: *′x* **and**
p :: *′x list*
assumes *refl-on X r*
shows *triangle-ineq X d* ∧ *p* ∈ *relation-paths r* ∧ *totally-invariant-dist d r* ∧
a ∈ *X* ∧ *b* ∈ *X* ⟶ *path-length* (*a*#*p*@[*b*]) *d* ≥ *d a b*
proof (*clarify*, *induction p d arbitrary*: *a b rule*: *path-length.induct*)
case (*1 d*)
show *d a b* ≤ *path-length* (*a*#[]@[*b*]) *d*
by *simp*
next
case (*2 x d*)
thus *d a b* ≤ *path-length* (*a*#[*x*]@[*b*]) *d*
by *simp*
next
case (*3 x y xs d*)
assume
ineq: *triangle-ineq X d* **and**
a-in-X: *a* ∈ *X* **and**
b-in-X: *b* ∈ *X* **and**
rel: *x*#*y*#*xs* ∈ *relation-paths r* **and**
invar: *totally-invariant-dist d r* **and**
hyp: ⋀ *a b*. *triangle-ineq X d* ⟹ *xs* ∈ *relation-paths r* ⟹ *totally-invariant-dist*
d r ⟹
a ∈ *X* ⟹ *b* ∈ *X* ⟹ *d a b* ≤ *path-length* (*a*#*xs*@[*b*]) *d*

then obtain *k* :: *nat* **where**
len: *length* (*x*#*y*#*xs*) = *2* ∗ *k*
by *auto*
moreover have ∀ *i* < *k* − *1*. (*xs*!(*2* ∗ *i*), *xs*!(*2* ∗ *i* + *1*)) =
((*x*#*y*#*xs*)!(*2* ∗ (*i* + *1*)), (*x*#*y*#*xs*)!(*2* ∗ (*i* + *1*) + *1*))
by *simp*
ultimately have ∀ *i* < *k* − *1*. (*xs*!(*2* ∗ *i*), *xs*!(*2* ∗ *i* + *1*)) ∈ *r*
using *rel less-diff-conv*
unfolding *relation-paths.simps*
by *fastforce*
moreover have *length xs* = *2* ∗ (*k* − *1*)
using *len*
by *simp*
ultimately have *xs* ∈ *relation-paths r*
by *simp*
hence ∀ *x y*. *x* ∈ *X* ∧ *y* ∈ *X* ⟶ *d x y* ≤ *path-length* (*x*#*xs*@[*y*]) *d*
using *ineq invar hyp*
by *blast*
moreover have *path-length* (*a*#(*x*#*y*#*xs*)@[*b*]) *d* = *d a x* + *path-length* (*y*#*xs*@[*b*])
d
by *simp*
moreover have *x-rel-y*: (*x*, *y*) ∈ *r*
using *rel*
unfolding *relation-paths.simps*
by *fastforce*
ultimately have *path-length* (*a*#(*x*#*y*#*xs*)@[*b*]) *d* ≥ *d a x* + *d y b*
using *assms add-left-mono assms refl-onD2 b-in-X*
unfolding *refl-on-def*
by *metis*
moreover have *d a x* + *d y b* = *d a x* + *d x b*
using *invar x-rel-y rewrite-totally-invariant-dist assms b-in-X*
unfolding *refl-on-def*
by *fastforce*
moreover have *d a x* + *d x b* ≥ *d a b*
using *a-in-X b-in-X x-rel-y assms ineq*
unfolding *refl-on-def triangle-ineq-def*
by *auto*
ultimately show *d a b* ≤ *path-length* (*a*#(*x*#*y*#*xs*)@[*b*]) *d*
by *simp*
qed

lemma *quotient-dist-coincides-with-dist*$_{\mathcal{Q}}$:
fixes
d :: *'x Distance* **and**
r :: *'x rel* **and**
X :: *'x set*
assumes
equiv: *equiv X r* **and**
tri: *triangle-ineq X d* **and**

invar: *totally-invariant-dist d r*
shows $\forall\ A \in X\ //\ r.\ \forall\ B \in X\ //\ r.$ *quotient-dist r d A B* = $dist_{\mathcal{Q}}$ *d A B*
proof (*clarify*)
fix
A :: *'x set* **and**
B :: *'x set*
assume
A-in-quot-X: $A \in X\ //\ r$ **and**
B-in-quot-X: $B \in X\ //\ r$
then obtain
a :: *'x* **and**
b :: *'x* **where**
el: $a \in A \wedge b \in B$ **and**
def-dist: $dist_{\mathcal{Q}}\ d\ A\ B = d\ a\ b$
using *dist-pass-to-quotient assms in-quotient-imp-non-empty ex-in-conv*
by (*metis* (*full-types*))
hence *equiv-class*: $A = r$ `` $\{a\} \wedge B = r$ `` $\{b\}$
using *A-in-quot-X B-in-quot-X assms equiv-class-eq-iff equiv-class-self quotientI quotient-eq-iff*
by *meson*
have *subset-X*: $r \subseteq X \times X \wedge A \subseteq X \wedge B \subseteq X$
using *assms A-in-quot-X B-in-quot-X equiv-def refl-on-def Union-quotient Union-upper*
by *metis*
have $\forall\ p \in$ *admissible-paths r A B*.
($\exists\ p'\ x\ y.\ x \in A \wedge y \in B \wedge p' \in$ *relation-paths r* $\wedge\ p = x\#p'@[y]$)
unfolding *admissible-paths.simps*
by *blast*
moreover have $\forall\ x\ y.\ x \in A \wedge y \in B \longrightarrow d\ x\ y = d\ a\ b$
using *invar equiv-class*
by *auto*
moreover have *refl-on X r*
using *equiv equiv-def*
by *blast*
ultimately have $\forall\ p.\ p \in$ *admissible-paths r A B* $\longrightarrow$ *path-length p d* $\geq d\ a\ b$
using *admissible-path-len*[*of X r d*] *tri subset-X el invar in-mono*
by *metis*
hence $\forall\ l.\ l \in \bigcup$ $\{\{$*path-length p d* $\mid p.\ p \in$ *admissible-paths r A B*$\}\}$ $\longrightarrow l \geq d\ a\ b$
by *blast*
hence *geq*: *quotient-dist r d A B* $\geq d\ a\ b$
unfolding *quotient-dist.simps*[*of r d A B*] *le-Inf-iff*
by *simp*
with *el def-dist*
have *geq*: *quotient-dist r d A B* $\geq dist_{\mathcal{Q}}\ d\ A\ B$
by *presburger*
have $[a,\ b] \in$ *admissible-paths r A B*
using *el*
by *simp*

moreover have *path-length* $[a, b]$ $d = d\ a\ b$
  by *simp*
ultimately have *quotient-dist* $r\ d\ A\ B \leq d\ a\ b$
  using *quotient-dist.simps*[*of* $r\ d\ A\ B$] *CollectI Inf-lower ccpo-Sup-singleton*
  by (*metis* (*mono-tags*, *lifting*))
thus *quotient-dist* $r\ d\ A\ B = dist_{\mathcal{Q}}\ d\ A\ B$
  using *geq def-dist nle-le*
  by *metis*
qed

lemma *inf-dist-coincides-with-dist*$_{\mathcal{Q}}$:
  fixes
    d :: $'x$ *Distance* **and**
    r :: $'x$ *rel* **and**
    X :: $'x$ *set*
  assumes
    equiv-X-r: *equiv* $X\ r$ **and**
    tot-inv-d-r: *totally-invariant-dist* $d\ r$
  shows $\forall\ A \in X\ //\ r.\ \forall\ B \in X\ //\ r.$ *inf-dist*$_{\mathcal{Q}}$ $d\ A\ B = dist_{\mathcal{Q}}\ d\ A\ B$
proof (*clarify*)
  fix
    A :: $'x$ *set* **and**
    B :: $'x$ *set*
  assume
    A-in-quot-X: $A \in X\ //\ r$ **and**
    B-in-quot-X: $B \in X\ //\ r$
  then obtain
    a :: $'x$ **and**
    b :: $'x$ **where**
      el: $a \in A \wedge b \in B$ **and**
      def-dist: $dist_{\mathcal{Q}}\ d\ A\ B = d\ a\ b$
      using *dist-pass-to-quotient equiv-X-r tot-inv-d-r in-quotient-imp-non-empty*
ex-in-conv
    by (*metis* (*full-types*))
  from *def-dist equiv-X-r tot-inv-d-r*
  have $\forall\ x\ y.\ x \in A \wedge y \in B \longrightarrow d\ x\ y = d\ a\ b$
    using *dist-pass-to-quotient A-in-quot-X B-in-quot-X*
    by *force*
  hence $\{d\ x\ y \mid x\ y.\ x \in A \wedge y \in B\} = \{d\ a\ b\}$
    using *el*
    by *blast*
  thus *inf-dist*$_{\mathcal{Q}}$ $d\ A\ B = dist_{\mathcal{Q}}\ d\ A\ B$
    unfolding *inf-dist*$_{\mathcal{Q}}$*.simps*
    using *def-dist*
    by *simp*
qed

lemma *inf-helper*:
  fixes

A :: $'x$ *set* **and**
B :: $'x$ *set* **and**
d :: $'x$ *Distance*
shows *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$ = *Inf* $\{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
proof −
have $\forall\ a\ b.\ a \in A \wedge b \in B \longrightarrow$ *Inf* $\{d\ a\ b \mid b.\ b \in B\} \leq d\ a\ b$
using *INF-lower Setcompr-eq-image*
by *metis*
hence $\forall\ \alpha \in \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}.\ \exists\ \beta \in \{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}.\ \beta \leq \alpha$
by *blast*
hence *Inf* $\{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\} \leq$ *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *Inf-mono*
by (*metis* (*no-types*, *lifting*))
moreover have $\neg$ (*Inf* $\{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\} <$ *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$)
proof (*rule ccontr*, *simp*)
assume *Inf* $\{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\} <$ *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
then obtain α :: *ereal* **where**
inf: $\alpha \in \{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$ **and**
less: $\alpha <$ *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *Inf-less-iff*
by (*metis* (*no-types*, *lifting*))
then obtain a :: $'x$ **where**
a-in-A: $a \in A$ **and**
$\alpha =$ *Inf* $\{d\ a\ b \mid b.\ b \in B\}$
by *blast*
with *less*
have *inf-less*: *Inf* $\{d\ a\ b \mid b.\ b \in B\} <$ *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
by *blast*
have $\{d\ a\ b \mid b.\ b \in B\} \subseteq \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *a-in-A*
by *blast*
hence *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\} \leq$ *Inf* $\{d\ a\ b \mid b.\ b \in B\}$
using *Inf-superset-mono*
by (*metis* (*no-types*, *lifting*))
with *inf-less*
show *False*
using *linorder-not-less*
by *simp*
qed
ultimately show *?thesis*
by *simp*
qed

lemma *invar-dist-simple*:

```
  fixes
    d :: 'y Distance and
    G :: 'x monoid and
    Y :: 'y set and
    φ :: ('x, 'y) binary-fun
  assumes
    action-φ: group-action G Y φ and
    invar: invariant-dist d (carrier G) Y φ
  shows simple (rel-induced-by-action (carrier G) Y φ) Y d
proof (unfold simple.simps, safe)
  fix A :: 'y set
  assume class_Y: A ∈ Y // rel-induced-by-action (carrier G) Y φ
  have equiv-rel: equiv Y (rel-induced-by-action (carrier G) Y φ)
    using assms rel-ind-by-group-act-equiv
    by blast
  with class_Y obtain a :: 'y where
    a-in-A: a ∈ A
    using equiv-Eps-in
    by blast
  have subset: ∀ B ∈ Y // rel-induced-by-action (carrier G) Y φ. B ⊆ Y
    using equiv-rel in-quotient-imp-subset
    by blast
  hence ∀ B ∈ Y // rel-induced-by-action (carrier G) Y φ.
        ∀ B' ∈ Y // rel-induced-by-action (carrier G) Y φ.
          ∀ b ∈ B. ∀ c ∈ B'. b ∈ Y ∧ c ∈ Y
    using class_Y
    by blast
  hence eq-dist:
    ∀ B ∈ Y // rel-induced-by-action (carrier G) Y φ.
      ∀ B' ∈ Y // rel-induced-by-action (carrier G) Y φ.
        ∀ b ∈ B. ∀ c ∈ B'. ∀ g ∈ carrier G.
          d (φ g c) (φ g b) = d c b
    using invar rewrite-invariant-dist class_Y
    by metis
  have ∀ b ∈ Y. ∀ g ∈ carrier G. (b, φ g b) ∈ rel-induced-by-action (carrier G)
Y φ
    unfolding rel-induced-by-action.simps
    using group-action.element-image action-φ
    by fastforce
  hence ∀ b ∈ Y. ∀ g ∈ carrier G. φ g b ∈ rel-induced-by-action (carrier G) Y φ
`` {b}
    unfolding Image-def
    by blast
  moreover have equiv-class:
    ∀ B. B ∈ Y // rel-induced-by-action (carrier G) Y φ ⟶
      (∀ b ∈ B. B = rel-induced-by-action (carrier G) Y φ `` {b})
    using Image-singleton-iff equiv-class-eq-iff equiv-rel quotientI quotient-eq-iff
    by meson
  ultimately have closed-class:
```

$\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.\ \forall\ b \in B.\ \forall\ g \in \textit{carrier}\ G.$
$\varphi\ g\ b \in B$
using *equiv-rel subset*
by *blast*
with *eq-dist* $\textit{class}_Y$
have *a-subset-A*:
$\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\{d\ a\ b \mid b.\ b \in B\} \subseteq \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *a-in-A*
by *blast*
have $\forall\ a' \in A.\ A = \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi\ ``\ \{a'\}$
using $\textit{class}_Y$ *equiv-rel equiv-class*
by *presburger*
hence $\forall\ a' \in A.\ (a',\ a) \in \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi$
using *a-in-A*
by *blast*
hence $\forall\ a' \in A.\ \exists\ g \in \textit{carrier}\ G.\ \varphi\ g\ a' = a$
by *simp*
hence $\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\forall\ a'\ b.\ a' \in A \wedge b \in B \longrightarrow (\exists\ g \in \textit{carrier}\ G.\ d\ a'\ b = d\ a\ (\varphi\ g\ b))$
using *eq-dist* $\textit{class}_Y$
by *metis*
hence $\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\forall\ a'\ b.\ a' \in A \wedge b \in B \longrightarrow d\ a'\ b \in \{d\ a\ b \mid b.\ b \in B\}$
using *closed-class mem-Collect-eq*
by *fastforce*
hence $\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\{d\ a\ b \mid b.\ b \in B\} \supseteq \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *closed-class*
by *blast*
with *a-subset-A*
have $\forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\textit{inf-dist}_{\mathcal{Q}}\ d\ A\ B = \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$
unfolding $\textit{inf-dist}_{\mathcal{Q}}.\textit{simps}$
by *fastforce*
thus $\exists\ a \in A.\ \forall\ B \in Y\ //\ \textit{rel-induced-by-action}\ (\textit{carrier}\ G)\ Y\ \varphi.$
$\textit{inf-dist}_{\mathcal{Q}}\ d\ A\ B = \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$
using *a-in-A*
by *blast*
qed

lemma *tot-invar-dist-simple*:
fixes
d :: *'x Distance* **and**
r :: *'x rel* **and**
X :: *'x set*
assumes
equiv-on-X: *equiv* X r **and**
invar: *totally-invariant-dist* d r

shows *simple r X d*
proof (*unfold simple.simps*, *safe*)
fix A :: $'x$ *set*
assume *A-quot-X*: $A \in X$ // r
then obtain a :: $'x$ **where**
a-in-A: $a \in A$
using *equiv-on-X equiv-Eps-in*
by *blast*
have $\forall\ a \in A.\ A = r$ ‘‘ $\{a\}$
using *A-quot-X equiv-on-X Image-singleton-iff equiv-class-eq-iff quotientI quotient-eq-iff*
by *meson*
hence $\forall\ a\ a'.\ a \in A \wedge a' \in A \longrightarrow (a, a') \in r$
by *blast*
moreover have $\forall\ B \in X$ // $r.\ \forall\ b \in B.\ (b, b) \in r$
using *equiv-on-X quotient-eq-iff*
by *metis*
ultimately have $\forall\ B \in X$ // $r.\ \forall\ a\ a'\ b.\ a \in A \wedge a' \in A \wedge b \in B \longrightarrow d\ a\ b = d\ a'\ b$
using *invar rewrite-totally-invariant-dist*
by *simp*
hence $\forall\ B \in X$ // $r.\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\} = \{d\ a\ b \mid a'\ b.\ a' \in A \wedge b \in B\}$
using *a-in-A*
by *blast*
moreover have $\forall\ B \in X$ // $r.\ \{d\ a\ b \mid a'\ b.\ a' \in A \wedge b \in B\} = \{d\ a\ b \mid b.\ b \in B\}$
using *a-in-A*
by *blast*
ultimately have $\forall\ B \in X$ // $r.\ Inf\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\} = Inf\ \{d\ a\ b \mid b.\ b \in B\}$
by *simp*
hence $\forall\ B \in X$ // $r.$ *inf-dist*$_{\mathcal{Q}}$ $d\ A\ B = Inf\ \{d\ a\ b \mid b.\ b \in B\}$
by *simp*
thus $\exists\ a \in A.\ \forall\ B \in X$ // $r.$ *inf-dist*$_{\mathcal{Q}}$ $d\ A\ B = Inf\ \{d\ a\ b \mid b.\ b \in B\}$
using *a-in-A*
by *blast*
qed

5.7.2 Quotient Consensus and Results

fun *elections-*$\mathcal{K}_{\mathcal{Q}}$ :: $('a, 'v)$ *Election rel* $\Rightarrow$ $('a, 'v, 'r$ *Result*$)$ *Consensus-Class* $\Rightarrow$ $('a, 'v)$ *Election set set* **where**
elections-$\mathcal{K}_{\mathcal{Q}}$ $r\ C$ = (*elections-*$\mathcal{K}$ C) // r

fun (**in** *result*) *limit-set*$_{\mathcal{Q}}$ :: $('a, 'v)$ *Election set* $\Rightarrow$ $'r$ *set* $\Rightarrow$ $'r$ *set* **where**
limit-set$_{\mathcal{Q}}$ X *res* = $\bigcap$ {*limit-set* (*alternatives-*$\mathcal{E}$ E) *res* | $E.\ E \in X$}

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:
  fixes
    X :: $'x$ *set* **and**
    Y :: $'x$ *set* **and**
    Z :: $'x$ *set* **and**
    r :: $'x$ *rel*
  assumes
    equiv X r **and**
    $Y \subseteq X$ **and**
    $Z \subseteq X$ **and**
    $Z \in Y \mathbin{//} r$ **and**
    closed-under-restr-rel r X Y
  shows $Z \subseteq Y$
proof (*safe*)
  fix z :: $'x$
  assume $z \in Z$
  then obtain y :: $'x$ **where**
    $y \in Y$ **and**
    $(y, z) \in r$
    using *assms*
    unfolding *quotient-def Image-def*
    by *blast*
  hence $(y, z) \in r \cap Y \times X$
    using *assms*
    unfolding *equiv-def refl-on-def*
    by *blast*
  hence $z \in \{z.\ \exists\ y \in Y.\ (y, z) \in r \cap Y \times X\}$
    by *blast*
  thus $z \in Y$
    using *assms*
    unfolding *closed-under-restr-rel.simps restr-rel.simps*
    by *blast*
qed

lemma (**in** *result*) *limit-set-invar*:
  fixes
    d :: $('a, 'v)$ *Election Distance* **and**
    r :: $('a, 'v)$ *Election rel* **and**
    C :: $('a, 'v, 'r$ *Result*$)$ *Consensus-Class* **and**
    X :: $('a, 'v)$ *Election set* **and**
    A :: $('a, 'v)$ *Election set*
  assumes
    quot-class: $A \in X \mathbin{//} r$ **and**
    equiv-rel: *equiv* X r **and**
    cons-subset: *elections-*$\mathcal{K}$ $C \subseteq X$ **and**
    invar-res: *satisfies* $(\lambda\ E.$ *limit-set* $($*alternatives-*$\mathcal{E}$ $E)$ *UNIV*$)$ $($*Invariance* $r)$
  shows $\forall\ a \in A.$ *limit-set* $($*alternatives-*$\mathcal{E}$ $a)$ *UNIV* $=$ *limit-set*$_{\mathcal{Q}}$ A *UNIV*
proof

fix a :: $('a, 'v)$ *Election*
assume *a-in-A*: $a \in A$
hence $\forall\ b \in A.\ (a, b) \in r$
using *quot-class equiv-rel quotient-eq-iff*
by *metis*
hence $\forall\ b \in A.$ *limit-set* (*alternatives-$\mathcal{E}$ b*) *UNIV* = *limit-set* (*alternatives-$\mathcal{E}$ a*) *UNIV*
using *invar-res*
unfolding *satisfies.simps*
by (*metis* (*mono-tags*, *lifting*))
hence *limit-set$_{\mathcal{Q}}$ A UNIV* = $\bigcap$ {*limit-set* (*alternatives-$\mathcal{E}$ a*) *UNIV*}
unfolding *limit-set$_{\mathcal{Q}}$.simps*
using *a-in-A*
by *blast*
thus *limit-set* (*alternatives-$\mathcal{E}$ a*) *UNIV* = *limit-set$_{\mathcal{Q}}$ A UNIV*
by *simp*
qed

lemma (**in** *result*) *preimg-invar*:
fixes
f :: $'x \Rightarrow 'y$ **and**
domain$_f$ :: $'x$ *set* **and**
d :: $'x$ *Distance* **and**
r :: $'x$ *rel* **and**
X :: $'x$ *set*
assumes
equiv-rel: *equiv X r* **and**
cons-subset: *domain$_f$* $\subseteq X$ **and**
closed-domain: *closed-under-restr-rel r X domain$_f$* **and**
invar-f: *satisfies f* (*Invariance* (*Restr r domain$_f$*))
shows $\forall\ y.$ (*preimg f domain$_f$ y*) // r = *preimg* ($\pi_{\mathcal{Q}}$ f) (*domain$_f$* // r) y
proof (*safe*)
fix
A :: $'x$ *set* **and**
y :: $'y$
assume *preimg-quot*: $A \in$ *preimg f domain$_f$ y* // r
hence *A-in-dom*: $A \in$ *domain$_f$* // r
unfolding *preimg.simps quotient-def*
by *blast*
obtain x :: $'x$ **where**
$x \in$ *preimg f domain$_f$ y* **and**
A-eq-img-singleton-r: $A = r$ `` $\{x\}$
using *equiv-rel preimg-quot quotientE*
unfolding *quotient-def*
by *blast*
hence *x-in-dom-and-f-x-y*: $x \in$ *domain$_f$* $\wedge$ $f\ x = y$
unfolding *preimg.simps*
by *blast*
moreover have r `` $\{x\} \subseteq X$

using *equiv-rel equiv-type*
by *fastforce*
ultimately have r `` $\{x\} \subseteq domain_f$
using *closed-domain A-eq-img-singleton-r A-in-dom*
by *fastforce*
hence $\forall\ x' \in r$ `` $\{x\}.\ (x, x') \in Restr\ r\ domain_f$
using *x-in-dom-and-f-x-y in-mono*
by *blast*
hence $\forall\ x' \in r$ `` $\{x\}.\ f\ x' = y$
using *invar-f x-in-dom-and-f-x-y*
unfolding *satisfies.simps*
by *metis*
moreover have $x \in A$
using *equiv-rel cons-subset equiv-class-self in-mono*
A-eq-img-singleton-r x-in-dom-and-f-x-y
by *metis*
ultimately have f ` $A = \{y\}$
using *A-eq-img-singleton-r*
by *auto*
hence $\pi_{\mathcal{Q}}\ f\ A = y$
unfolding $\pi_{\mathcal{Q}}$*.simps singleton-set.simps*
using *insert-absorb insert-iff insert-not-empty singleton-set-def-if-card-one*
is-singletonI is-singleton-altdef singleton-set.simps
by *metis*
thus $A \in preimg\ (\pi_{\mathcal{Q}}\ f)\ (domain_f\ //\ r)\ y$
using *A-in-dom*
unfolding *preimg.simps*
by *blast*
next
fix
A :: $'x\ set$ **and**
y :: $'y$
assume *quot-preimg*: $A \in preimg\ (\pi_{\mathcal{Q}}\ f)\ (domain_f\ //\ r)\ y$
hence *A-in-dom-rel-r*: $A \in domain_f\ //\ r$
using *cons-subset equiv-rel*
by *auto*
hence $A \subseteq X$
using *equiv-rel cons-subset Image-subset equiv-type quotientE*
by *metis*
hence *A-in-dom*: $A \subseteq domain_f$
using *closed-under-equiv-rel-subset*[*of* $X\ r\ domain_f\ A$]
closed-domain cons-subset A-in-dom-rel-r equiv-rel
by *blast*
moreover obtain x :: $'x$ **where**
x-in-A: $x \in A$ **and**
A-eq-r-img-single-x: $A = r$ `` $\{x\}$
using *A-in-dom-rel-r equiv-rel cons-subset equiv-class-self in-mono quotientE*
by *metis*
ultimately have $\forall\ x' \in A.\ (x, x') \in Restr\ r\ domain_f$

by *blast*
hence $\forall\ x' \in A.\ f\,x' = f\,x$
using *invar-f*
by *fastforce*
hence f ‘ $A = \{f\,x\}$
using *x-in-A*
by *blast*
hence $\pi_{\mathcal{Q}}\ f\ A = f\,x$
unfolding $\pi_{\mathcal{Q}}$*.simps singleton-set.simps*
using *is-singleton-altdef singleton-set-def-if-card-one*
by *fastforce*
also have $\pi_{\mathcal{Q}}\ f\ A = y$
using *quot-preimg*
unfolding *preimg.simps*
by *blast*
finally have $f\,x = y$
by *simp*
moreover have $x \in domain_f$
using *x-in-A A-in-dom*
by *blast*
ultimately have $x \in preimg\ f\ domain_f\ y$
by *simp*
thus $A \in preimg\ f\ domain_f\ y\ //\ r$
using *A-eq-r-img-single-x*
unfolding *quotient-def*
by *blast*
qed

lemma *minimizer-helper*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$domain_f :: 'x\ set$ **and**
$d :: 'x\ Distance$ **and**
$Y :: 'y\ set$ **and**
$x :: 'x$ **and**
$y :: 'y$
shows $y \in minimizer\ f\ domain_f\ d\ Y\ x =$
$(y \in Y \wedge (\forall\ y' \in Y.\ Inf\ (d\ x$ ‘ $(preimg\ f\ domain_f\ y)) \leq Inf\ (d\ x$ ‘ $(preimg\ f\ domain_f\ y'))))$
unfolding *is-arg-min-def minimizer.simps arg-min-set.simps*
by *auto*

lemma *rewr-singleton-set-system-union*:
fixes
$Y :: 'x\ set\ set$ **and**
$X :: 'x\ set$
assumes $Y \subseteq singleton\text{-}set\text{-}system\ X$
shows
singleton-set-union: $x \in \bigcup\ Y \longleftrightarrow \{x\} \in Y$ **and**

obtain-singleton: $A \in$ *singleton-set-system* $X \longleftrightarrow (\exists\ x \in X.\ A = \{x\})$
unfolding *singleton-set-system.simps*
using *assms*
by *auto*

lemma *union-inf*:
fixes X :: *ereal set set*
shows $Inf\ \{Inf\ A \mid A.\ A \in X\} = Inf\ (\bigcup\ X)$
proof −
let $?inf = Inf\ \{Inf\ A \mid A.\ A \in X\}$
have $\forall\ A \in X.\ \forall\ x \in A.\ ?inf \leq x$
using *INF-lower2 Inf-lower Setcompr-eq-image*
by *metis*
hence $\forall\ x \in \bigcup\ X.\ ?inf \leq x$
by *simp*
hence *le*: $?inf \leq Inf\ (\bigcup\ X)$
using *Inf-greatest*
by *blast*
have $\forall\ A \in X.\ Inf\ (\bigcup\ X) \leq Inf\ A$
using *Inf-superset-mono Union-upper*
by *metis*
hence $Inf\ (\bigcup\ X) \leq Inf\ \{Inf\ A \mid A.\ A \in X\}$
using *le-Inf-iff*
by *auto*
thus *?thesis*
using *le*
by *simp*
qed

5.7.3 Quotient Distance Rationalization

fun (**in** *result*) $\mathcal{R}_{\mathcal{Q}}$ :: ('a, 'v) *Election rel* ⇒ ('a, 'v) *Election Distance*
⇒ ('a, 'v, 'r *Result*) *Consensus-Class* ⇒ ('a, 'v) *Election set* ⇒ 'r *set* **where**
$\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A = \bigcup$ (*minimizer* ($\pi_{\mathcal{Q}}$ (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C))) (*elections-*$\mathcal{K}_{\mathcal{Q}}$ r C)
(*inf-dist*$_{\mathcal{Q}}$ d) (*singleton-set-system* (*limit-set*$_{\mathcal{Q}}$ A *UNIV*)) A)

fun (**in** *result*) *distance-*$\mathcal{R}_{\mathcal{Q}}$ :: ('a, 'v) *Election rel* ⇒ ('a, 'v) *Election Distance*
⇒ ('a, 'v, 'r *Result*) *Consensus-Class* ⇒ ('a, 'v) *Election set* ⇒ 'r *Result*
where
distance-$\mathcal{R}_{\mathcal{Q}}$ $r\ d\ C\ A$ =
($\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A$, $\pi_{\mathcal{Q}}$ (λ E. *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV*) $A - \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A$, {})

Hadjibeyli and Wilson 2016 4.17

theorem (**in** *result*) *invar-dr-simple-dist-imp-quotient-dr-winners*:
fixes
d :: ('a, 'v) *Election Distance* **and**

C :: ('a, 'v, 'r Result) *Consensus-Class* **and**
r :: ('a, 'v) *Election rel* **and**
X :: ('a, 'v) *Election set* **and**
A :: ('a, 'v) *Election set*
assumes
simple: *simple r X d* **and**
closed-domain: *closed-under-restr-rel r X* (*elections-$\mathcal{K}$ C*) **and**
invar-res: *satisfies* (λ *E*. *limit-set* (*alternatives-$\mathcal{E}$ E*) *UNIV*) (*Invariance r*) **and**
invar-C: *satisfies* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*Invariance* (*Restr r* (*elections-$\mathcal{K}$ C*))) **and**
invar-dr: *satisfies* (*fun$_\mathcal{E}$* ($\mathcal{R}_\mathcal{W}$ *d C*)) (*Invariance r*) **and**
quot-class: $A \in X$ // r **and**
equiv-rel: *equiv X r* **and**
cons-subset: *elections-$\mathcal{K}$ C* $\subseteq X$
shows $\pi_\mathcal{Q}$ (*fun$_\mathcal{E}$* ($\mathcal{R}_\mathcal{W}$ *d C*)) $A = \mathcal{R}_\mathcal{Q}$ *r d C A*
proof −
have *preimg-img-imp-cls*:
$\forall$ *y B*. $B \in$ *preimg* ($\pi_\mathcal{Q}$ (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*))) (*elections-$\mathcal{K}_\mathcal{Q}$ r C*) *y*
$\longrightarrow B \in$ (*elections-$\mathcal{K}$ C*) // r
by *simp*
have $\forall$ y'. $\forall$ $E \in$ *preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) y'. $E \in r$ `` $\{E\}$
using *equiv-rel cons-subset equiv-class-self equiv-rel in-mono*
unfolding *equiv-def preimg.simps*
by *fastforce*
hence $\forall$ y'.
$\bigcup$ (*preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) y' // r) $\supseteq$
preimg (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) y'
unfolding *quotient-def*
by *blast*
moreover have $\forall$ y'.
$\bigcup$ (*preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) y' // r) $\subseteq$
preimg (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) y'
proof (*standard*, *standard*)
fix
Y' :: 'r *set* **and**
E :: ('a, 'v) *Election*
assume $E \in \bigcup$ (*preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) Y' // r)
then obtain B :: ('a, 'v) *Election set* **where**
E-in-B: $E \in B$ **and**
$B \in$ *preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) Y' // r
by *blast*
then obtain E' :: ('a, 'v) *Election* **where**
$B = r$ `` $\{E'\}$ **and**
map-to-Y': $E' \in$ *preimg* (*elect-r* $\circ$ *fun$_\mathcal{E}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) Y'
using *quotientE*
by *blast*
hence *in-restr-rel*: $(E', E) \in r \cap$ (*elections-$\mathcal{K}$ C*) $\times X$
using *E-in-B equiv-rel*

unfolding *preimg.simps equiv-def refl-on-def*
by *blast*
hence $E \in$ *elections-*$\mathcal{K}$ C
using *closed-domain*
unfolding *closed-under-restr-rel.simps restr-rel.simps Image-def*
by *blast*
hence *rel-cons-els*: $(E', E) \in$ *Restr r* (*elections-*$\mathcal{K}$ C)
using *in-restr-rel*
by *blast*
hence (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) E = (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) E'
using *invar-C*
unfolding *satisfies.simps*
by *blast*
hence (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) $E = Y'$
using *map-to-*Y'
by *simp*
thus $E \in$ *preimg* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) Y'
unfolding *preimg.simps*
using *rel-cons-els*
by *blast*
qed
ultimately have *preimg-partition*: $\forall$ y'.
$\bigcup$ (*preimg* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) y' // r) =
preimg (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) y'
by *blast*
have *quot-classes-subset*: (*elections-*$\mathcal{K}$ C) // $r \subseteq X$ // r
using *cons-subset*
unfolding *quotient-def*
by *blast*
obtain a :: ($'a$, $'v$) *Election* **where**
a-in-A: $a \in A$ **and**
a-def-inf-dist: $\forall$ $B \in X$ // r. *inf-dist*$_{\mathcal{Q}}$ d A B = *Inf* $\{d\ a\ b \mid b.\ b \in B\}$
using *simple quot-class*
unfolding *simple.simps*
by *blast*
hence *inf-dist-preimg-sets*:
$\forall$ y' B. $B \in$ *preimg* ($\pi_{\mathcal{Q}}$ (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C))) (*elections-*$\mathcal{K}_{\mathcal{Q}}$ r C) y' $\longrightarrow$
inf-dist$_{\mathcal{Q}}$ d A B = *Inf* $\{d\ a\ b \mid b.\ b \in B\}$
using *preimg-img-imp-cls quot-classes-subset*
by *blast*
have *valid-res-eq*: *singleton-set-system* (*limit-set* (*alternatives-*$\mathcal{E}$ a) *UNIV*) =
singleton-set-system (*limit-set*$_{\mathcal{Q}}$ A *UNIV*)
using *invar-res a-in-A quot-class cons-subset equiv-rel limit-set-invar*
by *metis*
have *inf-le-iff*: $\forall$ x.
($\forall$ $y' \in$ *singleton-set-system* (*limit-set* (*alternatives-*$\mathcal{E}$ a) *UNIV*).
Inf (d a ` *preimg* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) $\{x\}$)
$\leq$ *Inf* (d a ` *preimg* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) y'))
= ($\forall$ $y' \in$ *singleton-set-system* (*limit-set*$_{\mathcal{Q}}$ A *UNIV*).

Inf (*inf-dist*$_\mathcal{Q}$ *d A* ' *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) {*x*})
≤ *Inf* (*inf-dist*$_\mathcal{Q}$ *d A* ' *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) *y'*))
proof −
have *preimg-partition-dist*: ∀ *y'*.
Inf {*d a b* | *b*. *b* ∈ ⋃ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*)} =
Inf (*d a* ' *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'*)
using *Setcompr-eq-image preimg-partition*
by *metis*
have ∀ *y'*.
{*Inf* {*d a b* | *b*. *b* ∈ *B*}
| *B*. *B* ∈ *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*}
= {*Inf E* | *E*. *E* ∈ {{*d a b* | *b*. *b* ∈ *B*}
| *B*. *B* ∈ *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*}}
by *blast*
hence ∀ *y'*.
Inf {*Inf* {*d a b* | *b*. *b* ∈ *B*} | *B*.
B ∈ *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*} =
Inf (⋃ {{*d a b* | *b*. *b* ∈ *B*} | *B*.
B ∈ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*)})
using *union-inf*
by *presburger*
moreover have
∀ *y'*. {*d a b* | *b*. *b* ∈ ⋃ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*)} =
⋃ {{*d a b* | *b*. *b* ∈ *B*} | *B*.
B ∈ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*)}
by *blast*
ultimately have *rewrite-inf-dist*:
∀ *y'*. *Inf* {*Inf* {*d a b* | *b*. *b* ∈ *B*}
| *B*. *B* ∈ *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*}
= *Inf* {*d a b* | *b*. *b* ∈ ⋃ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'* // *r*)}
by *presburger*
have ∀ *y'*. *inf-dist*$_\mathcal{Q}$ *d A* ' *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) *y'*
= {*Inf* {*d a b* | *b*. *b* ∈ *B*}
| *B*. *B* ∈ *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) *y'*}
using *inf-dist-preimg-sets*
unfolding *Image-def*
by *auto*
moreover have ∀ *y'*.
{*Inf* {*d a b* | *b*. *b* ∈ *B*} | *B*.
B ∈ *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) *y'*} =
{*Inf* {*d a b* | *b*. *b* ∈ *B*} | *B*.
B ∈ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y'*) // *r*}
unfolding *elections-*$\mathcal{K}_\mathcal{Q}$*.simps*

using *preimg-invar closed-domain cons-subset equiv-rel invar-C*
by *blast*
ultimately have
$\forall\ y'.\ Inf\ (inf\text{-}dist_{\mathcal{Q}}\ d\ A\ \text{‘}\ preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ y')$
$= Inf\ \{Inf\ \{d\ a\ b\ |\ b.\ b \in B\}$
$|\ B.\ B \in preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y'\ //\ r\}$
by *simp*
thus *?thesis*
using *valid-res-eq rewrite-inf-dist preimg-partition-dist*
by *presburger*
qed
from *a-in-A*
have $\pi_{\mathcal{Q}}\ (fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C))\ A = fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C)\ a$
using *invar-dr equiv-rel quot-class pass-to-quotient invariance-is-congruence*
by *blast*
moreover have $\forall\ x.\ x \in fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C)\ a \longleftrightarrow x \in \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A$
proof
fix $x :: 'r$
have $(x \in fun_{\mathcal{E}}\ (\mathcal{R}_{\mathcal{W}}\ d\ C)\ a) =$
$(x \in \bigcup\ (minimizer\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ d$
$(singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV))\ a))$
using $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*
by *metis*
also have ... $= (\{x\} \in minimizer\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)$
d
$(singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV))\ a)$
using *singleton-set-union*
unfolding *minimizer.simps arg-min-set.simps is-arg-min-def*
by *auto*
also have ... $= (\{x\} \in singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV)$
$\wedge$
$(\forall\ y' \in singleton\text{-}set\text{-}system\ (limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV).$
$Inf\ (d\ a\ \text{‘}\ preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ \{x\}) \leq$
$Inf\ (d\ a\ \text{‘}\ preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y')))$
using *minimizer-helper*
by (*metis* (*no-types*, *lifting*))
also have ... $= (\{x\} \in singleton\text{-}set\text{-}system\ (limit\text{-}set_{\mathcal{Q}}\ A\ UNIV) \wedge$
$(\forall\ y' \in singleton\text{-}set\text{-}system\ (limit\text{-}set_{\mathcal{Q}}\ A\ UNIV).$
$Inf\ (inf\text{-}dist_{\mathcal{Q}}\ d\ A\ \text{‘}\ preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ \{x\})$
$\leq Inf\ (inf\text{-}dist_{\mathcal{Q}}\ d\ A\ \text{‘}\ preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ y')))$
using *valid-res-eq inf-le-iff*
by *blast*
also have ... =
$(\{x\} \in minimizer\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)$
$(inf\text{-}dist_{\mathcal{Q}}\ d)\ (singleton\text{-}set\text{-}system\ (limit\text{-}set_{\mathcal{Q}}\ A\ UNIV))$
$A)$

using *minimizer-helper*
by (*metis* (*no-types*, *lifting*))
also have ... = ($x \in \bigcup$ (*minimizer* ($\pi_{\mathcal{Q}}$ (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C))) (*elections-*$\mathcal{K}_{\mathcal{Q}}$ r C)
(*inf-dist*$_{\mathcal{Q}}$ d) (*singleton-set-system* (*limit-set*$_{\mathcal{Q}}$ A *UNIV*)) A))
using *singleton-set-union*
unfolding *minimizer.simps arg-min-set.simps is-arg-min-def*
by *auto*
finally show ($x \in$ *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C) a) = ($x \in \mathcal{R}_{\mathcal{Q}}$ r d C A)
unfolding $\mathcal{R}_{\mathcal{Q}}$*.simps*
by *blast*
qed
ultimately show $\pi_{\mathcal{Q}}$ (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C)) $A = \mathcal{R}_{\mathcal{Q}}$ r d C A
by *blast*
qed

theorem (**in** *result*) *invar-dr-simple-dist-imp-quotient-dr*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r *Result*) *Consensus-Class* **and**
r :: ('a, 'v) *Election rel* **and**
X :: ('a, 'v) *Election set* **and**
A :: ('a, 'v) *Election set*
assumes
simple: *simple* r X d **and**
closed-domain: *closed-under-restr-rel* r X (*elections-*$\mathcal{K}$ C) **and**
invar-res: *satisfies* (λ E. *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV*) (*Invariance* r) **and**
invar-C: *satisfies* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*Invariance* (*Restr* r (*elections-*$\mathcal{K}$ C))) **and**
invar-dr: *satisfies* (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C)) (*Invariance* r) **and**
quot-class: $A \in X$ // r **and**
equiv-rel: *equiv* X r **and**
cons-subset: *elections-*$\mathcal{K}$ $C \subseteq X$
shows $\pi_{\mathcal{Q}}$ (*fun*$_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ d C)) A = *distance-*$\mathcal{R}_{\mathcal{Q}}$ r d C A
proof −
have $\forall$ E. *fun*$_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ d C) E =
(*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C) E, *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV* − *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C) E, {})
by *simp*
moreover have $\forall$ $E \in A$. *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C) $E = \pi_{\mathcal{Q}}$ (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C)) A
using *invar-dr invariance-is-congruence pass-to-quotient quot-class equiv-rel*
by *blast*
moreover have $\pi_{\mathcal{Q}}$ (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C)) $A = \mathcal{R}_{\mathcal{Q}}$ r d C A
using *invar-dr-simple-dist-imp-quotient-dr-winners assms*
by *blast*
moreover have
$\forall$ $E \in A$. *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV* = $\pi_{\mathcal{Q}}$ (λ E. *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV*) A

using *invar-res invariance-is-congruence′ pass-to-quotient quot-class equiv-rel*
by *blast*
ultimately have *all-eq*:
$\forall\ E \in A.\ fun_{\mathcal{E}}\ (distance\text{-}\mathcal{R}\ d\ C)\ E =$
$(\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \pi_{\mathcal{Q}}\ (\lambda\ E.\ limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ A - \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \{\})$
by *fastforce*
hence $\{(\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \pi_{\mathcal{Q}}\ (\lambda\ E.\ limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ A - \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \{\})\} \supseteq$
$fun_{\mathcal{E}}\ (distance\text{-}\mathcal{R}\ d\ C)$ ‘ A
by *blast*
moreover have $A \neq \{\}$
using *quot-class equiv-rel in-quotient-imp-non-empty*
by *metis*
ultimately have *single-img*:
$\{(\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \pi_{\mathcal{Q}}\ (\lambda\ E.\ limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ A - \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \{\})\} =$
$fun_{\mathcal{E}}\ (distance\text{-}\mathcal{R}\ d\ C)$ ‘ A
using *empty-is-image subset-singletonD*
by (*metis* (*no-types*, *lifting*))
moreover from *this*
have *card* ($fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*) ‘ *A*) = *1*
using *is-singleton-altdef is-singletonI*
by (*metis* (*no-types*, *lifting*))
moreover from *this single-img*
have *the-inv* (λ *x*. $\{x\}$) ($fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*) ‘ *A*) =
$(\mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \pi_{\mathcal{Q}}\ (\lambda\ E.\ limit\text{-}set\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ A - \mathcal{R}_{\mathcal{Q}}\ r\ d\ C\ A,\ \{\})$
using *singleton-insert-inj-eq singleton-set.elims singleton-set-def-if-card-one*
by (*metis* (*no-types*))
ultimately show *?thesis*
unfolding *distance*-$\mathcal{R}_{\mathcal{Q}}$.*simps*
using $\pi_{\mathcal{Q}}$.*simps*[*of* $fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*)] *singleton-set.simps*[*of* $fun_{\mathcal{E}}$ (*distance*-$\mathcal{R}$ *d C*) ‘ *A*]
by *presburger*
qed

end

5.8 Result and Property Locale Code Generation

theory *Interpretation-Code*
imports *Electoral-Module*
Distance-Rationalization
begin
setup *Locale-Code.open-block*

Lemmas stating the explicit instantiations of interpreted abstract functions from locales.

lemma *electoral-module-$\mathcal{SCF}$-code-lemma*:
fixes m :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module*
shows $\mathcal{SCF}$*-result.electoral-module m* = (∀ *A V p. profile V A p* ⟶ *well-formed-$\mathcal{SCF}$ A* (*m V A p*))
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *simp*

lemma $\mathcal{R}_\mathcal{W}$*-$\mathcal{SCF}$-code-lemma*:
fixes
d :: (′*a*, ′*v*) *Election Distance* **and**
K :: (′*a*, ′*v*, ′*a Result*) *Consensus-Class* **and**
V :: ′*v set* **and**
A :: ′*a set* **and**
p :: (′*a*, ′*v*) *Profile*
shows $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$ *d K V A p* = *arg-min-set* (*score d K* (*A*, *V*, *p*)) (*limit-set-$\mathcal{SCF}$ A UNIV*)
unfolding $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*.simps*
by *safe*

lemma *distance-$\mathcal{R}$-$\mathcal{SCF}$-code-lemma*:
fixes
d :: (′*a*, ′*v*) *Election Distance* **and**
K :: (′*a*, ′*v*, ′*a Result*) *Consensus-Class* **and**
V :: ′*v set* **and**
A :: ′*a set* **and**
p :: (′*a*, ′*v*) *Profile*
shows $\mathcal{SCF}$*-result.distance-$\mathcal{R}$ d K V A p* =
($\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$ *d K V A p*, (*limit-set-$\mathcal{SCF}$ A UNIV*) − $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$ *d K V A p*, {})
unfolding $\mathcal{SCF}$*-result.distance-$\mathcal{R}$.simps*
by *safe*

lemma $\mathcal{R}_\mathcal{W}$*-std-$\mathcal{SCF}$-code-lemma*:
fixes
d :: (′*a*, ′*v*) *Election Distance* **and**
K :: (′*a*, ′*v*, ′*a Result*) *Consensus-Class* **and**
V :: ′*v set* **and**
A :: ′*a set* **and**
p :: (′*a*, ′*v*) *Profile*
shows $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*-std d K V A p* =
arg-min-set (*score-std d K* (*A*, *V*, *p*)) (*limit-set-$\mathcal{SCF}$ A UNIV*)
unfolding $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*-std.simps*
by *safe*

lemma *distance-$\mathcal{R}$-std-$\mathcal{SCF}$-code-lemma*:
fixes
d :: (′*a*, ′*v*) *Election Distance* **and**

K :: ('a, 'v, 'a Result) *Consensus-Class* **and**
V :: 'v *set* **and**
A :: 'a *set* **and**
p :: ('a, 'v) *Profile*
shows *$\mathcal{SCF}$-result.distance-$\mathcal{R}$-std d K V A p* =
(*$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$-std d K V A p*, (*limit-set-$\mathcal{SCF}$ A UNIV*) − *$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$-std d K V A p*, {})
unfolding *$\mathcal{SCF}$-result.distance-$\mathcal{R}$-std.simps*
by *safe*

lemma *anonymity-$\mathcal{SCF}$-code-lemma*:
shows *$\mathcal{SCF}$-result.anonymity* =
(λ *m*::(('a, 'v, 'a Result) *Electoral-Module*).
$\mathcal{SCF}$-result.electoral-module m $\wedge$
($\forall$ *A V p* π::('v $\Rightarrow$ 'v).
bij π $\longrightarrow$ (*let* (A', V', q) = (*rename* π (A, V, p)) *in*
finite-profile V A p $\wedge$ *finite-profile V' A' q* $\longrightarrow$ *m V A p* = *m V' A' q*)))
unfolding *$\mathcal{SCF}$-result.anonymity-def*
by *simp*

Declarations for replacing interpreted abstract functions from locales by their explicit instantiations for code generation.

declare [[*lc-add $\mathcal{SCF}$-result.electoral-module electoral-module-$\mathcal{SCF}$-code-lemma*]]
declare [[*lc-add $\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$ $\mathcal{R}_\mathcal{W}$-$\mathcal{SCF}$-code-lemma*]]
declare [[*lc-add $\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$-std $\mathcal{R}_\mathcal{W}$-std-$\mathcal{SCF}$-code-lemma*]]
declare [[*lc-add $\mathcal{SCF}$-result.distance-$\mathcal{R}$ distance-$\mathcal{R}$-$\mathcal{SCF}$-code-lemma*]]
declare [[*lc-add $\mathcal{SCF}$-result.distance-$\mathcal{R}$-std distance-$\mathcal{R}$-std-$\mathcal{SCF}$-code-lemma*]]
declare [[*lc-add $\mathcal{SCF}$-result.anonymity anonymity-$\mathcal{SCF}$-code-lemma*]]

Constant aliases to use when exporting code instead of the interpreted functions

definition *$\mathcal{R}_\mathcal{W}$-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$*
definition *$\mathcal{R}_\mathcal{W}$-std-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$-std*
definition *distance-$\mathcal{R}$-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.distance-$\mathcal{R}$*
definition *distance-$\mathcal{R}$-std-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.distance-$\mathcal{R}$-std*
definition *electoral-module-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.electoral-module*
definition *anonymity-$\mathcal{SCF}$-code* = *$\mathcal{SCF}$-result.anonymity*

setup *Locale-Code.close-block*

end

5.9 Drop Module

theory *Drop-Module*
  imports *Component-Types/Electoral-Module*
    Component-Types/Social-Choice-Types/Result
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according drop module rejects the lexicographically first n alternatives (from A) and defers the rest. It is primarily used as counterpart to the pass module in a parallel composition, in order to segment the alternatives into two groups.

5.9.1 Definition

fun *drop-module* :: $nat \Rightarrow 'a$ *Preference-Relation* $\Rightarrow ('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
where
  $\textit{drop-module}\ n\ r\ V\ A\ p =$
    $(\{\},$
    $\{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a \leq n\},$
    $\{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a > n\})$

5.9.2 Soundness

theorem *drop-mod-sound*[*simp*]:
  fixes
    r :: $'a$ *Preference-Relation* **and**
    n :: *nat*
  shows $\mathcal{SCF}$*-result.electoral-module* (*drop-module* n r)
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile*
  assume *profile* V A p
  let *?mod* = *drop-module* n r
  have $\forall\ a \in A.\ a \in \{x \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ x \leq n\} \lor$
      $a \in \{x \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ x > n\}$
    by *auto*
  hence $\{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a \leq n\} \cup \{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a > n\} = A$
    by *blast*
  hence *set-partition*: *set-equals-partition* A (*drop-module* n r V A p)
    by *simp*
  have $\forall\ a \in A.$
      $\neg\ (a \in \{x \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ x \leq n\} \land$
        $a \in \{x \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ x > n\})$
    by *simp*
  hence $\{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a \leq n\} \cap \{a \in A.\ \textit{rank}\ (\textit{limit}\ A\ r)\ a > n\} = \{\}$
    by *blast*

thus *well-formed-$\mathcal{SCF}$ A (?mod V A p)*
using *set-partition*
by *simp*
qed

lemma *drop-mod-only-voters*:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
shows *only-voters-vote* (*drop-module n r*)
unfolding *only-voters-vote-def*
by *simp*

5.9.3 Non-Electing

The drop module is non-electing.

theorem *drop-mod-non-electing*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
shows *non-electing* (*drop-module n r*)
unfolding *non-electing-def*
by *simp*

5.9.4 Properties

The drop module is strictly defer-monotone.

theorem *drop-mod-def-lift-inv*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
shows *defer-lift-invariance* (*drop-module n r*)
unfolding *defer-lift-invariance-def*
by *simp*

end

5.10 Pass Module

theory *Pass-Module*
imports *Component-Types/Electoral-Module*
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according pass module defers the lexicographically first n alternatives (from A) and rejects the rest. It is primarily used as counterpart to the drop module in a parallel composition in order to segment the alternatives into two groups.

5.10.1 Definition

```
fun pass-module :: nat ⇒ 'a Preference-Relation ⇒ ('a, 'v, 'a Result) Electoral-Module
where
  pass-module n r V A p =
    ({},
    {a ∈ A. rank (limit A r) a > n},
    {a ∈ A. rank (limit A r) a ≤ n})
```

5.10.2 Soundness

```
theorem pass-mod-sound[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows SCF-result.electoral-module (pass-module n r)
proof (unfold SCF-result.electoral-module-def, safe)
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  let ?mod = pass-module n r
  have ∀ a ∈ A. a ∈ {x ∈ A. rank (limit A r) x > n} ∨
               a ∈ {x ∈ A. rank (limit A r) x ≤ n}
    using CollectI not-less
    by metis
  hence {a ∈ A. rank (limit A r) a > n} ∪ {a ∈ A. rank (limit A r) a ≤ n} = A
    by blast
  hence set-equals-partition A (pass-module n r V A p)
    by simp
  moreover have
    ∀ a ∈ A.
      ¬ (a ∈ {x ∈ A. rank (limit A r) x > n} ∧
        a ∈ {x ∈ A. rank (limit A r) x ≤ n})
    by simp
  hence {a ∈ A. rank (limit A r) a > n} ∩ {a ∈ A. rank (limit A r) a ≤ n} = {}
    by blast
  ultimately show well-formed-SCF A (?mod V A p)
    by simp
qed

lemma pass-mod-only-voters:
  fixes
```

r :: *'a Preference-Relation* **and**
n :: *nat*
shows *only-voters-vote* (*pass-module n r*)
unfolding *only-voters-vote-def pass-module.simps*
by *blast*

5.10.3 Non-Blocking

The pass module is non-blocking.

theorem *pass-mod-non-blocking*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes
order: *linear-order r* **and**
g0-n: $n > 0$
shows *non-blocking* (*pass-module n r*)
proof (*unfold non-blocking-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*pass-module n r*)
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assume
fin-A: *finite A* **and**
rej-pass-A: *reject* (*pass-module n r*) *V A p* = *A* **and**
a-in-A: $a \in A$
moreover have *lin*: *linear-order-on A* (*limit A r*)
using *limit-presv-lin-ord order top-greatest*
by *metis*
moreover have
$\exists$ $b \in A$. *above* (*limit A r*) *b* = $\{b\}$
$\wedge$ ($\forall$ $c \in A$. *above* (*limit A r*) *c* = $\{c\} \longrightarrow c = b$)
using *fin-A a-in-A lin above-one*
by *blast*
moreover have $\{b \in A$. *rank* (*limit A r*) $b > n\} \neq A$
using *Suc-leI g0-n leD mem-Collect-eq above-rank calculation*
unfolding *One-nat-def*
by (*metis* (*no-types*, *lifting*))
hence *reject* (*pass-module n r*) *V A p* $\neq$ *A*
by *simp*
thus $a \in \{\}$
using *rej-pass-A*
by *simp*
qed

5.10.4 Non-Electing

The pass module is non-electing.

theorem *pass-mod-non-electing*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *non-electing* (*pass-module n r*)
unfolding *non-electing-def*
using *assms*
by *simp*

5.10.5 Properties

The pass module is strictly defer-monotone.

theorem *pass-mod-dl-inv*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *defer-lift-invariance* (*pass-module n r*)
unfolding *defer-lift-invariance-def*
using *assms*
by *simp*

theorem *pass-zero-mod-def-zero*[*simp*]:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 0* (*pass-module 0 r*)
proof (*unfold defers-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*pass-module 0 r*)
using *pass-mod-sound assms*
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assume
card-pos: *0* ≤ *card A* **and**
finite-A: *finite A* **and**
prof-A: *profile V A p*
have *linear-order-on A* (*limit A r*)
using *assms limit-presv-lin-ord*
by *blast*
hence *limit-is-connex*: *connex A* (*limit A r*)
using *lin-ord-imp-connex*

by *simp*
have $\forall$ *n*. (*n*::*nat*) $\leq$ *0* $\longrightarrow$ *n* = *0*
by *blast*
hence $\forall$ *a A'*. *a* $\in$ *A'* $\wedge$ *a* $\in$ *A* $\longrightarrow$ *connex A'* (*limit A r*) $\longrightarrow$
$\neg$ *rank* (*limit A r*) *a* $\leq$ *0*
using *above-connex above-presv-limit card-eq-0-iff equals0D finite-A assms rev-finite-subset*
unfolding *rank.simps*
by (*metis* (*no-types*))
hence {*a* $\in$ *A*. *rank* (*limit A r*) *a* $\leq$ *0*} = {}
using *limit-is-connex*
by *simp*
hence *card* {*a* $\in$ *A*. *rank* (*limit A r*) *a* $\leq$ *0*} = *0*
using *card.empty*
by *metis*
thus *card* (*defer* (*pass-module 0 r*) *V A p*) = *0*
by *simp*
qed

For any natural number n and any linear order, the according pass module defers n alternatives (if there are n alternatives). NOTE: The induction proof is still missing. The following are the proofs for n=1 and n=2.

theorem *pass-one-mod-def-one*[*simp*]:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 1* (*pass-module 1 r*)
proof (*unfold defers-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*pass-module 1 r*)
using *pass-mod-sound assms*
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assume
card-pos: *1* $\leq$ *card A* **and**
finite-A: *finite A* **and**
prof-A: *profile V A p*
show *card* (*defer* (*pass-module 1 r*) *V A p*) = *1*
proof −
have *A* $\neq$ {}
using *card-pos*
by *auto*
moreover have *lin-ord-on-A*: *linear-order-on A* (*limit A r*)
using *assms limit-presv-lin-ord*
by *blast*
ultimately have *winner-exists*:
$\exists$ *a* $\in$ *A*. *above* (*limit A r*) *a* = {*a*} $\wedge$

$(\forall\ b \in A.\ above\ (limit\ A\ r)\ b = \{b\} \longrightarrow b = a)$
using *finite-A above-one*
by *simp*
then obtain w **where** *w-unique-top*:
$above\ (limit\ A\ r)\ w = \{w\} \wedge$
$(\forall\ a \in A.\ above\ (limit\ A\ r)\ a = \{a\} \longrightarrow a = w)$
using *above-one*
by *auto*
hence $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq 1\} = \{w\}$
proof
assume
w-top: $above\ (limit\ A\ r)\ w = \{w\}$ **and**
w-unique: $\forall\ a \in A.\ above\ (limit\ A\ r)\ a = \{a\} \longrightarrow a = w$
have $rank\ (limit\ A\ r)\ w \leq 1$
using *w-top*
by *auto*
hence $\{w\} \subseteq \{a \in A.\ rank\ (limit\ A\ r)\ a \leq 1\}$
using *winner-exists w-unique-top*
by *blast*
moreover have $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq 1\} \subseteq \{w\}$
proof
fix $a :: 'a$
assume *a-in-winner-set*: $a \in \{b \in A.\ rank\ (limit\ A\ r)\ b \leq 1\}$
hence *a-in-A*: $a \in A$
by *auto*
hence *connex-limit*: $connex\ A\ (limit\ A\ r)$
using *lin-ord-imp-connex lin-ord-on-A*
by *simp*
hence $let\ q = limit\ A\ r\ in\ a \preceq_q a$
using *connex-limit above-connex pref-imp-in-above a-in-A*
by *metis*
hence $(a, a) \in limit\ A\ r$
by *simp*
hence *a-above-a*: $a \in above\ (limit\ A\ r)\ a$
unfolding *above-def*
by *simp*
have $above\ (limit\ A\ r)\ a \subseteq A$
using *above-presv-limit assms*
by *fastforce*
hence *above-finite*: $finite\ (above\ (limit\ A\ r)\ a)$
using *finite-A finite-subset*
by *simp*
have $rank\ (limit\ A\ r)\ a \leq 1$
using *a-in-winner-set*
by *simp*
moreover have $rank\ (limit\ A\ r)\ a \geq 1$
using *Suc-leI above-finite card-eq-0-iff equals0D neq0-conv a-above-a*
unfolding *rank.simps One-nat-def*
by *metis*

ultimately have *rank* (*limit A r*) *a* = *1*
by *simp*
hence {*a*} = *above* (*limit A r*) *a*
using *a-above-a lin-ord-on-A rank-one-imp-above-one*
by *metis*
hence *a* = *w*
using *w-unique a-in-A*
by *simp*
thus $a \in \{w\}$
by *simp*
qed
ultimately have $\{w\} = \{a \in A.$ *rank* (*limit A r*) $a \leq 1\}$
by *auto*
thus *?thesis*
by *simp*
qed
thus *card* (*defer* (*pass-module 1 r*) *V A p*) = *1*
by *simp*
qed
qed

theorem *pass-two-mod-def-two*:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 2* (*pass-module 2 r*)
proof (*unfold defers-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*pass-module 2 r*)
using *assms*
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assume
min-card-two: $2 \leq$ *card A* **and**
fin-A: *finite A* **and**
prof-A: *profile V A p*
from *min-card-two*
have *not-empty-A*: $A \neq \{\}$
by *auto*
moreover have *limit-A-order*: *linear-order-on A* (*limit A r*)
using *limit-presv-lin-ord assms*
by *auto*
ultimately obtain *a* **where**
above (*limit A r*) *a* = {*a*}
using *above-one min-card-two fin-A prof-A*
by *blast*
hence $\forall\ b \in A.$ *let q* = *limit A r in* ($b \preceq_q a$)

using *limit-A-order pref-imp-in-above empty-iff lin-ord-imp-connex insert-iff insert-subset above-presv-limit assms*
unfolding *connex-def*
by *metis*
hence *a-best*: $\forall\ b \in A.\ (b,\ a) \in limit\ A\ r$
by *simp*
hence *a-above*: $\forall\ b \in A.\ a \in above\ (limit\ A\ r)\ b$
unfolding *above-def*
by *simp*
hence $a \in \{a \in A.\ rank\ (limit\ A\ r)\ a \leq 2\}$
using *CollectI not-empty-A empty-iff fin-A insert-iff limit-A-order above-one above-rank one-le-numeral*
by (*metis* (*no-types*, *lifting*))
hence *a-in-defer*: $a \in defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p$
by *simp*
have *finite* $(A - \{a\})$
using *fin-A*
by *simp*
moreover have *A-not-only-a*: $A - \{a\} \neq \{\}$
using *Diff-empty Diff-idemp Diff-insert0 not-empty-A insert-Diff finite.emptyI card.insert-remove card.empty min-card-two Suc-n-not-le-n numeral-2-eq-2*
by *metis*
moreover have *limit-A-without-a-order*:
linear-order-on $(A - \{a\})$ (*limit* $(A - \{a\})$ *r*)
using *limit-presv-lin-ord assms top-greatest*
by *blast*
ultimately obtain *b* **where**
b: *above* (*limit* $(A - \{a\})$ *r*) $b = \{b\}$
using *above-one*
by *metis*
hence $\forall\ c \in A - \{a\}.\ let\ q = limit\ (A - \{a\})\ r\ in\ (c \preceq_q b)$
using *limit-A-without-a-order pref-imp-in-above empty-iff lin-ord-imp-connex insert-iff insert-subset above-presv-limit assms*
unfolding *connex-def*
by *metis*
hence *b-in-limit*: $\forall\ c \in A - \{a\}.\ (c,\ b) \in limit\ (A - \{a\})\ r$
by *simp*
hence *b-best*: $\forall\ c \in A - \{a\}.\ (c,\ b) \in limit\ A\ r$
by *auto*
hence $\forall\ c \in A - \{a,\ b\}.\ c \notin above\ (limit\ A\ r)\ b$
using *b Diff-iff Diff-insert2 above-presv-limit insert-subset assms limit-presv-above limit-rel-presv-above*
by *metis*
moreover have *above-subset*: *above* (*limit* *A* *r*) $b \subseteq A$
using *above-presv-limit assms*
by *metis*
moreover have *b-above-b*: $b \in above\ (limit\ A\ r)\ b$
using *b b-best above-presv-limit mem-Collect-eq assms insert-subset*
unfolding *above-def*

by *metis*
ultimately have *above-b-eq-ab*: *above* (*limit* A r) $b = \{a,\ b\}$
using *a-above*
by *auto*
hence *card-above-b-eq-two*: *rank* (*limit* A r) $b = 2$
using *A-not-only-a b-in-limit*
by *auto*
hence *b-in-defer*: $b \in$ *defer* (*pass-module* 2 r) V A p
using *b-above-b above-subset*
by *auto*
have *b-above*: $\forall\ c \in A - \{a\}.\ b \in$ *above* (*limit* A r) c
using *b-best mem-Collect-eq*
unfolding *above-def*
by *metis*
have *connex* A (*limit* A r)
using *limit-A-order lin-ord-imp-connex*
by *auto*
hence $\forall\ c \in A.\ c \in$ *above* (*limit* A r) c
using *above-connex*
by *metis*
hence $\forall\ c \in A - \{a,\ b\}.\ \{a,\ b,\ c\} \subseteq$ *above* (*limit* A r) c
using *a-above b-above*
by *auto*
moreover have $\forall\ c \in A - \{a,\ b\}.$ *card* $\{a,\ b,\ c\} = 3$
using *DiffE Suc-1 above-b-eq-ab card-above-b-eq-two above-subset fin-A card-insert-disjoint finite-subset insert-commute numeral-3-eq-3*
unfolding *One-nat-def rank.simps*
by *metis*
ultimately have $\forall\ c \in A - \{a,\ b\}.$ *rank* (*limit* A r) $c \geq 3$
using *card-mono fin-A finite-subset above-presv-limit assms*
unfolding *rank.simps*
by *metis*
hence $\forall\ c \in A - \{a,\ b\}.$ *rank* (*limit* A r) $c > 2$
using *Suc-le-eq Suc-1 numeral-3-eq-3*
unfolding *One-nat-def*
by *metis*
hence $\forall\ c \in A - \{a,\ b\}.\ c \notin$ *defer* (*pass-module* 2 r) V A p
by (*simp add*: *not-le*)
moreover have *defer* (*pass-module* 2 r) V A $p \subseteq A$
by *auto*
ultimately have *defer* (*pass-module* 2 r) V A $p \subseteq \{a,\ b\}$
by *blast*
hence *defer* (*pass-module* 2 r) V A $p = \{a,\ b\}$
using *a-in-defer b-in-defer*
by *fastforce*
thus *card* (*defer* (*pass-module* 2 r) V A p) $= 2$
using *above-b-eq-ab card-above-b-eq-two*
unfolding *rank.simps*
by *presburger*

qed

end

5.11 Elect Module

theory *Elect-Module*
imports *Component-Types/Electoral-Module*
begin

The elect module is not concerned about the voter's ballots, and just elects all alternatives. It is primarily used in sequence after an electoral module that only defers alternatives to finalize the decision, thereby inducing a proper voting rule in the social choice sense.

5.11.1 Definition

fun *elect-module* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
elect-module V A p = (A, {}, {})

5.11.2 Soundness

theorem *elect-mod-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module elect-module*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *simp*

lemma *elect-mod-only-voters*: *only-voters-vote elect-module*
unfolding *only-voters-vote-def*
by *simp*

5.11.3 Electing

theorem *elect-mod-electing*[*simp*]: *electing elect-module*
unfolding *electing-def*
by *simp*

end

5.12 Plurality Module

theory *Plurality-Module*
  imports *Component-Types/Elimination-Module*
begin

The plurality module implements the plurality voting rule. The plurality rule elects all modules with the maximum amount of top preferences among all alternatives, and rejects all the other alternatives. It is electing and induces the classical plurality (voting) rule from social-choice theory.

5.12.1 Definition

fun *plurality-score* :: $('a, 'v)$ *Evaluation-Function* **where**
  plurality-score $V\ x\ A\ p$ = *win-count* $V\ p\ x$

fun *plurality* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **where**
  plurality $V\ A\ p$ = *max-eliminator plurality-score* $V\ A\ p$

fun *plurality'* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **where**
  plurality' $V\ A\ p$ =
    $(\{\},$
     $\{a \in A.\ \exists\ x \in A.$ *win-count* $V\ p\ x >$ *win-count* $V\ p\ a\},$
     $\{a \in A.\ \forall\ x \in A.$ *win-count* $V\ p\ x \leq$ *win-count* $V\ p\ a\})$

lemma *enat-leq-enat-set-max*:
  fixes
    x :: *enat* **and**
    X :: *enat set*
  assumes
    $x \in X$ **and**
    finite X
  shows $x \leq Max\ X$
  using *assms*
  by *simp*

lemma *plurality-mod-elim-equiv*:
  fixes
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: $('a, 'v)$ *Profile*
  assumes
    non-empty-A: $A \neq \{\}$ **and**
    fin-A: *finite* A **and**
    prof: *profile* $V\ A\ p$
  shows *plurality* $V\ A\ p$ = *plurality'* $V\ A\ p$
proof (*unfold plurality.simps plurality'.simps plurality-score.simps*, *standard*)
  have *fst* (*max-eliminator* $(\lambda\ V\ x\ A\ p.$ *win-count* $V\ p\ x)\ V\ A\ p) = \{\}$
    by *simp*

```
  also have ... = fst ({},
            {a ∈ A. ∃ b ∈ A. win-count V p a < win-count V p b},
            {a ∈ A. ∀ b ∈ A. win-count V p b ≤ win-count V p a})
    by simp
  finally show
    fst (max-eliminator (λ V x A p. win-count V p x) V A p) =
      fst ({},
           {a ∈ A. ∃ b ∈ A. win-count V p a < win-count V p b},
           {a ∈ A. ∀ b ∈ A. win-count V p b ≤ win-count V p a})
    by simp
next
  let ?no-max = {a ∈ A. win-count V p a < Max {win-count V p x | x. x ∈ A}}
= A
  have ?no-max ⟹ {win-count V p x | x. x ∈ A} ≠ {}
    using non-empty-A
    by blast
  moreover have finite {win-count V p x | x. x ∈ A}
    using fin-A
    by simp
  ultimately have exists-max: ?no-max ⟹ False
    using Max-in
    by fastforce
  have rej-eq:
    snd (max-eliminator (λ V b A p. win-count V p b) V A p) =
      snd ({},
           {a ∈ A. ∃ x ∈ A. win-count V p a < win-count V p x},
           {a ∈ A. ∀ x ∈ A. win-count V p x ≤ win-count V p a})
  proof (simp del: win-count.simps, safe)
    fix
      a :: 'a and
      b :: 'a
    assume
      b ∈ A and
      win-count V p a < Max {win-count V p a' | a'. a' ∈ A} and
      ¬ win-count V p b < Max {win-count V p a' | a'. a' ∈ A}
    thus ∃ b ∈ A. win-count V p a < win-count V p b
      using dual-order.strict-trans1 not-le-imp-less
      by blast
  next
    fix
      a :: 'a and
      b :: 'a
    assume
      a-in-A: a ∈ A and
      b-in-A: b ∈ A and
      wc-a-lt-wc-b: win-count V p a < win-count V p b
    moreover have ∀ t. t b ≤ Max {n. ∃ a'. (n::enat) = t a' ∧ a' ∈ A}
    proof (safe)
      fix
```

$t :: 'a \Rightarrow enat$
have $t\ b \in \{t\ a'\ |a'.\ a' \in A\}$
using *b-in-A*
by *auto*
thus $t\ b \leq Max\ \{t\ a'\ |a'.\ a' \in A\}$
using *enat-leq-enat-set-max fin-A*
by *auto*
qed
ultimately show $win\text{-}count\ V\ p\ a < Max\ \{win\text{-}count\ V\ p\ a'\ |\ a'.\ a' \in A\}$
using *dual-order.strict-trans1*
by *blast*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
wc-a-max: $\neg\ win\text{-}count\ V\ p\ a < Max\ \{win\text{-}count\ V\ p\ x\ |\ x.\ x \in A\}$
have $win\text{-}count\ V\ p\ b \in \{win\text{-}count\ V\ p\ x\ |\ x.\ x \in A\}$
using *b-in-A*
by *auto*
hence $win\text{-}count\ V\ p\ b \leq Max\ \{win\text{-}count\ V\ p\ x\ |\ x.\ x \in A\}$
using *b-in-A fin-A enat-leq-enat-set-max*
by *auto*
thus $win\text{-}count\ V\ p\ b \leq win\text{-}count\ V\ p\ a$
using *wc-a-max dual-order.strict-trans1 linorder-le-less-linear*
by *simp*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
wc-a-max: $\forall\ x \in A.\ win\text{-}count\ V\ p\ x \leq win\text{-}count\ V\ p\ a$ **and**
wc-a-not-max: $win\text{-}count\ V\ p\ a < Max\ \{win\text{-}count\ V\ p\ x\ |\ x.\ x \in A\}$
have $win\text{-}count\ V\ p\ b \leq win\text{-}count\ V\ p\ a$
using *b-in-A wc-a-max*
by *auto*
thus $win\text{-}count\ V\ p\ b < Max\ \{win\text{-}count\ V\ p\ x\ |\ x.\ x \in A\}$
using *wc-a-not-max*
by *simp*
next
assume *?no-max*
thus *False*
using *exists-max*
by *simp*
next

fix
a :: *'a* **and**
b :: *'a*
assume *?no-max*
thus *win-count V p a* $\leq$ *win-count V p b*
using *exists-max*
by *simp*
qed
thus *snd* (*max-eliminator* (λ *V b A p. win-count V p b*) *V A p*) =
snd ({},
{*a* $\in$ *A*. $\exists$ *b* $\in$ *A*. *win-count V p a* < *win-count V p b*},
{*a* $\in$ *A*. $\forall$ *b* $\in$ *A*. *win-count V p b* $\leq$ *win-count V p a*})
using *rej-eq snd-conv*
by *metis*
qed

5.12.2 Soundness

theorem *plurality-sound*[*simp*]: *$\mathcal{SCF}$-result.electoral-module plurality*
unfolding *plurality.simps*
using *max-elim-sound*
by *metis*

theorem *plurality'-sound*[*simp*]: *$\mathcal{SCF}$-result.electoral-module plurality'*
proof (*unfold $\mathcal{SCF}$-result.electoral-module-def*, *safe*)
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
have *disjoint3* (
{},
{*a* $\in$ *A*. $\exists$ *a'* $\in$ *A*. *win-count V p a* < *win-count V p a'*},
{*a* $\in$ *A*. $\forall$ *a'* $\in$ *A*. *win-count V p a'* $\leq$ *win-count V p a*})
by *auto*
moreover have
{*a* $\in$ *A*. $\exists$ *x* $\in$ *A*. *win-count V p a* < *win-count V p x*} $\cup$
{*a* $\in$ *A*. $\forall$ *x* $\in$ *A*. *win-count V p x* $\leq$ *win-count V p a*} = *A*
using *not-le-imp-less*
by *blast*
ultimately show *well-formed-$\mathcal{SCF}$ A* (*plurality' V A p*)
by *simp*
qed

lemma *plurality-score-only-voters*: *only-voters-count plurality-score*
proof (*unfold plurality-score.simps only-voters-count-def*, *safe*)
fix
A :: *'b set* **and**
V :: *'a set* **and**
p :: (*'b*, *'a*) *Profile* **and**

$p' :: ('b, 'a)$ *Profile* **and**
$a :: 'b$
assume
$\forall\ v \in V.\ p\ v = p'\ v$ **and**
$a \in A$
hence *finite* $V \longrightarrow$
card $\{v \in V.\ above\ (p\ v)\ a = \{a\}\} = card\ \{v \in V.\ above\ (p'\ v)\ a = \{a\}\}$
using *Collect-cong*
by (*metis* (*no-types*, *lifting*))
thus *win-count* $V\ p\ a =$ *win-count* $V\ p'\ a$
unfolding *win-count.simps*
by *presburger*
qed

lemma *plurality-only-voters*: *only-voters-vote plurality*
unfolding *plurality.simps*
using *max-elim-only-voters plurality-score-only-voters*
by *blast*

5.12.3 Non-Blocking

The plurality module is non-blocking.

theorem *plurality-mod-non-blocking*[*simp*]: *non-blocking plurality*
unfolding *plurality.simps*
using *max-elim-non-blocking*
by *metis*

5.12.4 Non-Electing

The plurality module is non-electing.

theorem *plurality-non-electing*[*simp*]: *non-electing plurality*
using *max-elim-non-electing*
unfolding *plurality.simps non-electing-def*
by *metis*

theorem *plurality′-non-electing*[*simp*]: *non-electing plurality′*
unfolding *non-electing-def*
by *simp*

5.12.5 Property

lemma *plurality-def-inv-mono-alts*:
fixes
$A :: 'a$ *set* **and**
$V :: 'v$ *set* **and**
$p :: ('a, 'v)$ *Profile* **and**
$q :: ('a, 'v)$ *Profile* **and**
$a :: 'a$

assumes
defer-a: $a \in$ *defer plurality* $V\ A\ p$ **and**
lift-a: *lifted* $V\ A\ p\ q\ a$
shows *defer plurality* $V\ A\ q$ = *defer plurality* $V\ A\ p \vee$ *defer plurality* $V\ A\ q = \{a\}$
proof −
have *set-disj*: $\forall\ b\ c.\ (b::'a) \notin \{c\} \vee b = c$
by *blast*
have *lifted-winner*: $\forall\ b \in A.\ \forall\ i \in V.$
above $(p\ i)\ b = \{b\} \longrightarrow$ (*above* $(q\ i)\ b = \{b\} \vee$ *above* $(q\ i)\ a = \{a\}$)
using *lift-a lifted-above-winner-alts*
unfolding *Profile.lifted-def*
by *metis*
hence $\forall\ i \in V.$ (*above* $(p\ i)\ a = \{a\} \longrightarrow$ *above* $(q\ i)\ a = \{a\}$)
using *defer-a lift-a*
unfolding *Profile.lifted-def*
by *metis*
hence *a-win-subset*: $\{i \in V.$ *above* $(p\ i)\ a = \{a\}\} \subseteq \{i \in V.$ *above* $(q\ i)\ a = \{a\}\}$
by *blast*
moreover have *lifted-prof*: *profile* $V\ A\ q$
using *lift-a*
unfolding *Profile.lifted-def*
by *metis*
ultimately have *win-count-a*: *win-count* $V\ p\ a \le$ *win-count* $V\ q\ a$
by (*simp add*: *card-mono*)
have *fin-A*: *finite* A
using *lift-a*
unfolding *Profile.lifted-def*
by *blast*
hence $\forall\ b \in A - \{a\}.$
$\forall\ i \in V.$ (*above* $(q\ i)\ a = \{a\} \longrightarrow$ *above* $(q\ i)\ b \neq \{b\}$)
using *DiffE above-one lift-a insertCI insert-absorb insert-not-empty*
unfolding *Profile.lifted-def profile-def*
by *metis*
with *lifted-winner*
have *above-QtoP*:
$\forall\ b \in A - \{a\}.$
$\forall\ i \in V.$ (*above* $(q\ i)\ b = \{b\} \longrightarrow$ *above* $(p\ i)\ b = \{b\}$)
using *lifted-above-winner-other lift-a*
unfolding *Profile.lifted-def*
by *metis*
hence $\forall\ b \in A - \{a\}.$
$\{i \in V.$ *above* $(q\ i)\ b = \{b\}\} \subseteq \{i \in V.$ *above* $(p\ i)\ b = \{b\}\}$
by (*simp add*: *Collect-mono*)
hence *win-count-other*: $\forall\ b \in A - \{a\}.$ *win-count* $V\ p\ b \ge$ *win-count* $V\ q\ b$
by (*simp add*: *card-mono*)
show *defer plurality* $V\ A\ q$ = *defer plurality* $V\ A\ p \vee$ *defer plurality* $V\ A\ q = \{a\}$

proof (*cases*)
  assume *win-count* V p a = *win-count* V q a
  hence *card* $\{i \in V.$ *above* $(p\ i)\ a = \{a\}\}$ = *card* $\{i \in V.$ *above* $(q\ i)\ a = \{a\}\}$
    using *win-count.simps Profile.lifted-def enat.inject lift-a*
    by (*metis* (*mono-tags*, *lifting*))
  moreover have *finite* $\{i \in V.$ *above* $(q\ i)\ a = \{a\}\}$
    using *Collect-mem-eq Profile.lifted-def finite-Collect-conjI lift-a*
    by (*metis* (*mono-tags*))
  ultimately have $\{i \in V.$ *above* $(p\ i)\ a = \{a\}\} = \{i \in V.$ *above* $(q\ i)\ a = \{a\}\}$
    using *a-win-subset*
    by (*simp add*: *card-subset-eq*)
  hence *above-pq*: $\forall\ i \in V.$ (*above* $(p\ i)\ a = \{a\}$) = (*above* $(q\ i)\ a = \{a\}$)
    by *blast*
  moreover have
    $\forall\ b \in A - \{a\}.$
      $\forall\ i \in V.$
        (*above* $(p\ i)\ b = \{b\} \longrightarrow$ (*above* $(q\ i)\ b = \{b\} \lor$ *above* $(q\ i)\ a = \{a\}$))
    using *lifted-winner*
    by *auto*
  moreover have
    $\forall\ b \in A - \{a\}.$
      $\forall\ i \in V.$ (*above* $(p\ i)\ b = \{b\} \longrightarrow$ *above* $(p\ i)\ a \neq \{a\}$)
  proof (*rule ccontr*, *simp*, *safe*, *simp*)
    fix
      $b :: 'a$ **and**
      $i :: 'v$
    assume
      b-in-A: $b \in A$ **and**
      i-is-voter: $i \in V$ **and**
      abv-b: *above* $(p\ i)\ b = \{b\}$ **and**
      abv-a: *above* $(p\ i)\ a = \{a\}$
    moreover from *b-in-A*
    have $A \neq \{\}$
      by *auto*
    moreover from *i-is-voter*
    have *linear-order-on* A $(p\ i)$
      using *lift-a*
      unfolding *Profile.lifted-def profile-def*
      by *simp*
    ultimately show $b = a$
      using *fin-A above-one-eq*
      by *metis*
  qed
  ultimately have *above-PtoQ*:
    $\forall\ b \in A - \{a\}.\ \forall\ i \in V.$ (*above* $(p\ i)\ b = \{b\} \longrightarrow$ *above* $(q\ i)\ b = \{b\}$)
    by *simp*
  hence $\forall\ b \in A.$
        card $\{i \in V.$ *above* $(p\ i)\ b = \{b\}\}$ =
          card $\{i \in V.$ *above* $(q\ i)\ b = \{b\}\}$

proof (*safe*)
fix b :: $'a$
assume
above-c: $\forall\ c \in A - \{a\}.\ \forall\ i \in V.\ above\ (p\ i)\ c = \{c\} \longrightarrow above\ (q\ i)\ c = \{c\}$ **and**
b-in-A: $b \in A$
show $card\ \{i \in V.\ above\ (p\ i)\ b = \{b\}\} =$
$card\ \{i \in V.\ above\ (q\ i)\ b = \{b\}\}$
using *DiffI b-in-A set-disj above-PtoQ above-QtoP above-pq*
by (*metis* (*no-types*, *lifting*))
qed
hence $\{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ p\ c \le win\text{-}count\ V\ p\ b\} =$
$\{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ q\ c \le win\text{-}count\ V\ q\ b\}$
by *auto*
hence $defer\ plurality'\ V\ A\ q = defer\ plurality'\ V\ A\ p \lor defer\ plurality'\ V\ A\ q = \{a\}$
by *simp*
hence $defer\ plurality\ V\ A\ q = defer\ plurality\ V\ A\ p \lor defer\ plurality\ V\ A\ q = \{a\}$
using *plurality-mod-elim-equiv empty-not-insert insert-absorb lift-a*
unfolding *Profile.lifted-def*
by (*metis* (*no-types*, *opaque-lifting*))
thus *?thesis*
by *simp*
next
assume $win\text{-}count\ V\ p\ a \ne win\text{-}count\ V\ q\ a$
hence *strict-less*: $win\text{-}count\ V\ p\ a < win\text{-}count\ V\ q\ a$
using *win-count-a*
by *simp*
have $a \in defer\ plurality\ V\ A\ p$
using *defer-a plurality.elims*
by (*metis* (*no-types*))
moreover have *non-empty-A*: $A \ne \{\}$
using *lift-a equals0D equiv-prof-except-a-def lifted-imp-equiv-prof-except-a*
by *metis*
moreover have *fin-A*: *finite-profile* $V\ A\ p$
using *lift-a*
unfolding *Profile.lifted-def*
by *simp*
ultimately have $a \in defer\ plurality'\ V\ A\ p$
using *plurality-mod-elim-equiv*
by *metis*
hence *a-in-win-p*: $a \in \{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ p\ c \le win\text{-}count\ V\ p\ b\}$
by *simp*
hence $\forall\ b \in A.\ win\text{-}count\ V\ p\ b \le win\text{-}count\ V\ p\ a$
by *simp*
hence *less*: $\forall\ b \in A - \{a\}.\ win\text{-}count\ V\ q\ b < win\text{-}count\ V\ q\ a$
using *DiffD1 antisym dual-order.trans not-le-imp-less win-count-a strict-less win-count-other*

by *metis*
hence $\forall\ b \in A - \{a\}.\ \neg\ (\forall\ c \in A.\ \textit{win-count}\ V\ q\ c \leq \textit{win-count}\ V\ q\ b)$
using *lift-a not-le*
unfolding *Profile.lifted-def*
by *metis*
hence $\forall\ b \in A - \{a\}.\ b \notin \{c \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ q\ b \leq \textit{win-count}\ V\ q\ c\}$
by *blast*
hence $\forall\ b \in A - \{a\}.\ b \notin \textit{defer plurality}'\ V\ A\ q$
by *simp*
hence $\forall\ b \in A - \{a\}.\ b \notin \textit{defer plurality}\ V\ A\ q$
using *lift-a non-empty-A plurality-mod-elim-equiv*
unfolding *Profile.lifted-def*
by (*metis* (*no-types*, *lifting*))
hence $\forall\ b \in A - \{a\}.\ b \notin \textit{defer plurality}\ V\ A\ q$
by *simp*
moreover have $a \in \textit{defer plurality}\ V\ A\ q$
proof −
have $\forall\ b \in A - \{a\}.\ \textit{win-count}\ V\ q\ b \leq \textit{win-count}\ V\ q\ a$
using *less less-imp-le*
by *metis*
moreover have $\textit{win-count}\ V\ q\ a \leq \textit{win-count}\ V\ q\ a$
by *simp*
ultimately have $\forall\ b \in A.\ \textit{win-count}\ V\ q\ b \leq \textit{win-count}\ V\ q\ a$
by *auto*
moreover have $a \in A$
using *a-in-win-p*
by *simp*
ultimately have $a \in \{b \in A.\ \forall\ c \in A.\ \textit{win-count}\ V\ q\ c \leq \textit{win-count}\ V\ q\ b\}$
by *simp*
hence $a \in \textit{defer plurality}'\ V\ A\ q$
by *simp*
hence $a \in \textit{defer plurality}\ V\ A\ q$
using *plurality-mod-elim-equiv non-empty-A fin-A lift-a non-empty-A*
unfolding *Profile.lifted-def*
by (*metis* (*no-types*))
thus *?thesis*
by *simp*
qed
moreover have $\textit{defer plurality}\ V\ A\ q \subseteq A$
by *simp*
ultimately show *?thesis*
by *blast*
qed
qed

The plurality rule is invariant-monotone.

theorem *plurality-mod-def-inv-mono*[*simp*]: *defer-invariant-monotonicity plurality*
proof (*unfold defer-invariant-monotonicity-def*, *intro conjI impI allI*)

show *SCF-result.electoral-module plurality*
  by *simp*
next
  show *non-electing plurality*
  by *simp*
next
  fix
    A :: *'b set* **and**
    V :: *'a set* **and**
    p :: (*'b*, *'a*) *Profile* **and**
    q :: (*'b*, *'a*) *Profile* **and**
    a :: *'b*
  assume $a \in$ *defer plurality V A p* $\wedge$ *Profile.lifted V A p q a*
  hence *defer plurality V A q* = *defer plurality V A p* $\vee$ *defer plurality V A q* = $\{a\}$
    using *plurality-def-inv-mono-alts*
    by *metis*
  thus *defer plurality V A q* = *defer plurality V A p* $\vee$ *defer plurality V A q* = $\{a\}$
    by *simp*
qed

end

5.13 Borda Module

theory *Borda-Module*
  imports *Component-Types/Elimination-Module*
begin

This is the Borda module used by the Borda rule. The Borda rule is a voting rule, where on each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.13.1 Definition

fun *borda-score* :: (*'a*, *'v*) *Evaluation-Function* **where**
  borda-score V x A p = ($\sum$ $y \in A$. (*prefer-count V p x y*))

fun *borda* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
  borda V A p = *max-eliminator borda-score V A p*

5.13.2 Soundness

theorem *borda-sound*: *$\mathcal{SCF}$-result.electoral-module borda*
  unfolding *borda.simps*
  using *max-elim-sound*
  by *metis*

5.13.3 Non-Blocking

The Borda module is non-blocking.

theorem *borda-mod-non-blocking*[*simp*]: *non-blocking borda*
  unfolding *borda.simps*
  using *max-elim-non-blocking*
  by *metis*

5.13.4 Non-Electing

The Borda module is non-electing.

theorem *borda-mod-non-electing*[*simp*]: *non-electing borda*
  using *max-elim-non-electing*
  unfolding *borda.simps non-electing-def*
  by *metis*

end

5.14 Condorcet Module

theory *Condorcet-Module*
  imports *Component-Types/Elimination-Module*
begin

This is the Condorcet module used by the Condorcet (voting) rule. The Condorcet rule is a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.14.1 Definition

fun *condorcet-score* :: (*'a*, *'v*) *Evaluation-Function* **where**
  condorcet-score V x A p =
    (*if* (*condorcet-winner V A p x*) *then 1 else 0*)

fun *condorcet* :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **where**
condorcet V A p = (*max-eliminator condorcet-score*) *V A p*

5.14.2 Soundness

theorem *condorcet-sound*: *SCF-result.electoral-module condorcet*
unfolding *condorcet.simps*
using *max-elim-sound*
by *metis*

5.14.3 Property

theorem *condorcet-score-is-condorcet-rating*: *condorcet-rating condorcet-score*
proof (*unfold condorcet-rating-def*, *safe*)
fix

A :: ′*b set* **and**
V :: ′*a set* **and**
p :: (′*b*, ′*a*) *Profile* **and**
w :: ′*b* **and**
l :: ′*b*
assume
c-win: *condorcet-winner V A p w* **and**
l-neq-w: $l \neq w$
have ¬ *condorcet-winner V A p l*
using *cond-winner-unique-eq c-win l-neq-w*
by *metis*
thus *condorcet-score V l A p* < *condorcet-score V w A p*
using *c-win zero-less-one*
unfolding *condorcet-score.simps*
by (*metis* (*full-types*))
qed

theorem *condorcet-is-dcc*: *defer-condorcet-consistency condorcet*
proof (*unfold defer-condorcet-consistency-def SCF-result.electoral-module-def*, *safe*)
fix
A :: ′*b set* **and**
V :: ′*a set* **and**
p :: (′*b*, ′*a*) *Profile*
assume
profile V A p
hence *well-formed-SCF A* (*max-eliminator condorcet-score V A p*)
using *max-elim-sound*
unfolding *SCF-result.electoral-module-def*
by *metis*
thus *well-formed-SCF A* (*condorcet V A p*)
by *simp*
next
fix
A :: ′*b set* **and**

```
    V :: 'a set and
    p :: ('b, 'a) Profile and
    a :: 'b
  assume
    c-win-w: condorcet-winner V A p a
  let ?m = (max-eliminator condorcet-score)::(('b, 'a, 'b Result) Electoral-Module)
  have defer-condorcet-consistency ?m
    using cr-eval-imp-dcc-max-elim condorcet-score-is-condorcet-rating
    by metis
  hence ?m V A p =
          ({}, A − defer ?m V A p, {b ∈ A. condorcet-winner V A p b})
    using c-win-w
    unfolding defer-condorcet-consistency-def
    by (metis (no-types))
  thus condorcet V A p =
          ({},
          A − defer condorcet V A p,
          {d ∈ A. condorcet-winner V A p d})
    by simp
qed

end
```

5.15 Copeland Module

```
theory Copeland-Module
  imports Component-Types/Elimination-Module
begin
```

This is the Copeland module used by the Copeland voting rule. The Copeland rule elects the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.15.1 Definition

```
fun copeland-score :: ('a, 'v) Evaluation-Function where
  copeland-score V x A p =
    card {y ∈ A . wins V x p y} − card {y ∈ A . wins V y p x}

fun copeland :: ('a, 'v, 'a Result) Electoral-Module where
  copeland V A p = max-eliminator copeland-score V A p
```

5.15.2 Soundness

theorem *copeland-sound*: *SCF-result.electoral-module copeland*
  unfolding *copeland.simps*
  using *max-elim-sound*
  by *metis*

5.15.3 Only participating voters impact the result

lemma *copeland-score-only-voters-count*: *only-voters-count copeland-score*
proof (*unfold copeland-score.simps only-voters-count-def*, *safe*)
  fix
    A :: $'b$ *set* **and**
    V :: $'a$ *set* **and**
    p :: $('b, 'a)$ *Profile* **and**
    p' :: $('b, 'a)$ *Profile* **and**
    a :: $'b$
  assume
    $\forall\ v \in V.\ p\ v = p'\ v$ **and**
    $a \in A$
  hence $\forall\ x\ y.\ \{v \in V.\ (x, y) \in p\ v\} = \{v \in V.\ (x, y) \in p'\ v\}$
    by *blast*
  hence $\forall\ x\ y.\ card\ \{y \in A.\ wins\ V\ x\ p\ y\} = card\ \{y \in A.\ wins\ V\ x\ p'\ y\}\ \wedge$
        $card\ \{x \in A.\ wins\ V\ x\ p\ y\} = card\ \{x \in A.\ wins\ V\ x\ p'\ y\}$
    by *simp*
  thus $card\ \{y \in A.\ wins\ V\ a\ p\ y\} - card\ \{y \in A.\ wins\ V\ y\ p\ a\} =$
      $card\ \{y \in A.\ wins\ V\ a\ p'\ y\} - card\ \{y \in A.\ wins\ V\ y\ p'\ a\}$
    by *presburger*
qed

theorem *copeland-only-voters-vote*: *only-voters-vote copeland*
  unfolding *copeland.simps*
  using *max-elim-only-voters only-voters-vote-def*
      copeland-score-only-voters-count
  by *blast*

5.15.4 Lemmas

For a Condorcet winner w, we have: "$\{card\ y \in A\ .\ wins\ x\ p\ y\} = |A| - 1$".

lemma *cond-winner-imp-win-count*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile* **and**
    w :: $'a$
  assumes *condorcet-winner* $V\ A\ p\ w$
  shows $card\ \{a \in A.\ wins\ V\ w\ p\ a\} = card\ A - 1$
proof −
  have $\forall\ a \in A - \{w\}.\ wins\ V\ w\ p\ a$

using *assms*
by *auto*
hence $\{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} = A - \{w\}$
by *blast*
hence *winner-wins-against-all-others*:
$card\ \{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} = card\ (A - \{w\})$
by *simp*
have $w \in A$
using *assms*
by *simp*
hence $card\ (A - \{w\}) = card\ A - 1$
using *card-Diff-singleton assms*
by *metis*
hence *winner-amount-one*: $card\ \{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} = card\ (A) - 1$
using *winner-wins-against-all-others*
by *linarith*
have *win-for-winner-not-reflexive*: $\forall\ a \in \{w\}.\ \neg\ wins\ V\ a\ p\ a$
by (*simp add*: *wins-irreflex*)
hence $\{a \in \{w\}.\ wins\ V\ w\ p\ a\} = \{\}$
by *blast*
hence *winner-amount-zero*: $card\ \{a \in \{w\}.\ wins\ V\ w\ p\ a\} = 0$
by *simp*
have *union*:
$\{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} \cup \{x \in \{w\}.\ wins\ V\ w\ p\ x\} = \{a \in A.\ wins\ V\ w\ p\ a\}$
using *win-for-winner-not-reflexive*
by *blast*
have *finite-defeated*: *finite* $\{a \in A - \{w\}.\ wins\ V\ w\ p\ a\}$
using *assms*
by *simp*
have *finite* $\{a \in \{w\}.\ wins\ V\ w\ p\ a\}$
by *simp*
hence $card\ (\{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} \cup \{a \in \{w\}.\ wins\ V\ w\ p\ a\}) =$
$card\ \{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} + card\ \{a \in \{w\}.\ wins\ V\ w\ p\ a\}$
using *finite-defeated card-Un-disjoint*
by *blast*
hence $card\ \{a \in A.\ wins\ V\ w\ p\ a\} =$
$card\ \{a \in A - \{w\}.\ wins\ V\ w\ p\ a\} + card\ \{a \in \{w\}.\ wins\ V\ w\ p\ a\}$
using *union*
by *simp*
thus *?thesis*
using *winner-amount-one winner-amount-zero*
by *linarith*
qed

For a Condorcet winner w, we have: "$card\ \{y \in A\ .\ wins\ y\ p\ x = 0$".

lemma *cond-winner-imp-loss-count*:
fixes
$A :: 'a\ set$ **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$
assumes *condorcet-winner* $V\ A\ p\ w$
shows *card* $\{a \in A.\ \mathit{wins}\ V\ a\ p\ w\} = \mathit{0}$
using *Collect-empty-eq card-eq-0-iff insert-Diff insert-iff wins-antisym assms*
unfolding *condorcet-winner.simps*
by (*metis* (*no-types*, *lifting*))

Copeland score of a Condorcet winner.

lemma *cond-winner-imp-copeland-score*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$
assumes *condorcet-winner* $V\ A\ p\ w$
shows *copeland-score* $V\ w\ A\ p = \mathit{card}\ A - \mathit{1}$
proof (*unfold copeland-score.simps*)
have *card* $\{a \in A.\ \mathit{wins}\ V\ w\ p\ a\} = \mathit{card}\ A - \mathit{1}$
using *cond-winner-imp-win-count assms*
by *metis*
moreover have *card* $\{a \in A.\ \mathit{wins}\ V\ a\ p\ w\} = \mathit{0}$
using *cond-winner-imp-loss-count assms*
by (*metis* (*no-types*))
ultimately show
enat (*card* $\{a \in A.\ \mathit{wins}\ V\ w\ p\ a\} - \mathit{card}\ \{a \in A.\ \mathit{wins}\ V\ a\ p\ w\}) = \mathit{enat}\ (\mathit{card}$
$A - \mathit{1})$
by *simp*
qed

For a non-Condorcet winner l, we have: "*card* $\{y \in A\ .\ \mathit{wins}\ x\ p\ y\} = |A| - \mathit{2}$".

lemma *non-cond-winner-imp-win-count*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$ **and**
l :: $'a$
assumes
winner: *condorcet-winner* $V\ A\ p\ w$ **and**
loser: $l \neq w$ **and**
l-in-A: $l \in A$
shows *card* $\{a \in A\ .\ \mathit{wins}\ V\ l\ p\ a\} \leq \mathit{card}\ A - \mathit{2}$
proof –
have *wins* $V\ w\ p\ l$
using *assms*
by *auto*

hence ¬ *wins V l p w*
  using *wins-antisym*
  by *simp*
moreover have ¬ *wins V l p l*
  using *wins-irreflex*
  by *simp*
ultimately have *wins-of-loser-eq-without-winner*:
  $\{y \in A \;.\; wins\ V\ l\ p\ y\} = \{y \in A - \{l,\ w\} \;.\; wins\ V\ l\ p\ y\}$
  by *blast*
have $\forall\ M\, f.\ finite\ M \longrightarrow card\ \{x \in M \;.\; f\ x\} \leq card\ M$
  by (*simp add*: *card-mono*)
moreover have *finite* $(A - \{l,\ w\})$
  using *finite-Diff winner*
  by *simp*
ultimately have $card\ \{y \in A - \{l,\ w\} \;.\; wins\ V\ l\ p\ y\} \leq card\ (A - \{l,\ w\})$
  using *winner*
  by (*metis* (*full-types*))
thus *?thesis*
  using *assms wins-of-loser-eq-without-winner*
  by *simp*
qed

5.15.5 Property

The Copeland score is Condorcet rating.

theorem *copeland-score-is-cr*: *condorcet-rating copeland-score*
proof (*unfold condorcet-rating-def*, *unfold copeland-score.simps*, *safe*)
  fix
    A :: *'b set* **and**
    V :: *'v set* **and**
    p :: (*'b*, *'v*) *Profile* **and**
    w :: *'b* **and**
    l :: *'b*
  assume
    winner: *condorcet-winner V A p w* **and**
    l-in-A: $l \in A$ **and**
    l-neq-w: $l \neq w$
  hence $card\ \{y \in A.\ wins\ V\ l\ p\ y\} \leq card\ A - 2$
    using *non-cond-winner-imp-win-count*
    by (*metis* (*mono-tags*, *lifting*))
  hence $card\ \{y \in A.\ wins\ V\ l\ p\ y\} - card\ \{y \in A.\ wins\ V\ y\ p\ l\} \leq card\ A - 2$
    using *diff-le-self order.trans*
    by *simp*
  moreover have $card\ A - 2 < card\ A - 1$
    using *card-0-eq diff-less-mono2 empty-iff l-in-A l-neq-w neq0-conv less-one*
      Suc-1 zero-less-diff add-diff-cancel-left' diff-is-0-eq Suc-eq-plus1
      card-1-singleton-iff order-less-le singletonD le-zero-eq winner
    unfolding *condorcet-winner.simps*
    by *metis*

ultimately have
card $\{y \in A.\ wins\ V\ l\ p\ y\}$ − *card* $\{y \in A.\ wins\ V\ y\ p\ l\} <$ *card* $A - 1$
using *order-le-less-trans*
by *fastforce*
moreover have *card* $\{a \in A.\ wins\ V\ a\ p\ w\} = 0$
using *cond-winner-imp-loss-count winner*
by *metis*
moreover have *card* $A - 1 =$ *card* $\{a \in A.\ wins\ V\ w\ p\ a\}$
using *cond-winner-imp-win-count winner*
by (*metis* (*full-types*))
ultimately show
enat (*card* $\{y \in A.\ wins\ V\ l\ p\ y\}$ − *card* $\{y \in A.\ wins\ V\ y\ p\ l\}$) <
enat (*card* $\{y \in A.\ wins\ V\ w\ p\ y\}$ − *card* $\{y \in A.\ wins\ V\ y\ p\ w\}$)
using *enat-ord-simps*
by *simp*
qed

theorem *copeland-is-dcc*: *defer-condorcet-consistency copeland*
proof (*unfold defer-condorcet-consistency-def* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
fix
A :: *'b set* **and**
V :: *'a set* **and**
p :: (*'b*, *'a*) *Profile*
assume *profile V A p*
hence *well-formed-*$\mathcal{SCF}$ *A* (*max-eliminator copeland-score V A p*)
using *max-elim-sound*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *metis*
thus *well-formed-*$\mathcal{SCF}$ *A* (*copeland V A p*)
by *auto*
next
fix
A :: *'b set* **and**
V :: *'v set* **and**
p :: (*'b*, *'v*) *Profile* **and**
w :: *'b*
assume *condorcet-winner V A p w*
moreover have *defer-condorcet-consistency* (*max-eliminator copeland-score*)
by (*simp add*: *copeland-score-is-cr*)
ultimately have *max-eliminator copeland-score V A p* =
({}, *A* − *defer* (*max-eliminator copeland-score*) *V A p*, {*d* ∈ *A*. *condorcet-winner V A p d*})
unfolding *defer-condorcet-consistency-def*
by (*metis* (*no-types*))
moreover have *copeland V A p* = *max-eliminator copeland-score V A p*
by *simp*
ultimately show
copeland V A p = ({}, *A* − *defer copeland V A p*, {*d* ∈ *A*. *condorcet-winner V A p d*})

by *metis*
qed

end

5.16 Minimax Module

theory *Minimax-Module*
imports *Component-Types/Elimination-Module*
begin

This is the Minimax module used by the Minimax voting rule. The Minimax rule elects the alternatives with the highest Minimax score. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.16.1 Definition

fun *minimax-score* :: ('*a*, '*v*) *Evaluation-Function* **where**
minimax-score V x A p =
Min {*prefer-count V p x y* | *y* . *y* ∈ *A* − {*x*}}

fun *minimax* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
minimax A p = *max-eliminator minimax-score A p*

5.16.2 Soundness

theorem *minimax-sound*: *SCF-result.electoral-module minimax*
unfolding *minimax.simps*
using *max-elim-sound*
by *metis*

5.16.3 Lemma

lemma *non-cond-winner-minimax-score*:
fixes
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
w :: '*a* **and**
l :: '*a*
assumes
prof: *profile V A p* **and**
winner: *condorcet-winner V A p w* **and**

```
    l-in-A: l ∈ A and
    l-neq-w: l ≠ w
  shows minimax-score V l A p ≤ prefer-count V p l w
proof (simp, clarify)
  assume fin-V: finite V
  have w ∈ A
    using winner
    by simp
  hence el: card {v ∈ V. (w, l) ∈ p v} ∈ {(card {v ∈ V. (y, l) ∈ p v}) | y. y ∈
A ∧ y ≠ l}
    using l-neq-w
    by auto
  moreover have fin: finite {(card {v ∈ V. (y, l) ∈ p v}) | y. y ∈ A ∧ y ≠ l}
  proof –
    have ∀ y ∈ A. card {v ∈ V. (y, l) ∈ p v} ≤ card V
      using fin-V
      by (simp add: card-mono)
    hence ∀ y ∈ A. card {v ∈ V. (y, l) ∈ p v} ∈ {.. card V}
      unfolding less-Suc-eq-le
      by simp
    hence {(card {v ∈ V. (y, l) ∈ p v}) | y. y ∈ A ∧ y ≠ l} ⊆ {0..card V}
      by auto
    thus ?thesis
      by (simp add: finite-subset)
  qed
  ultimately have Min {(card {v ∈ V. (y, l) ∈ p v}) | y. y ∈ A ∧ y ≠ l}
        ≤ card {v ∈ V. (w, l) ∈ p v}
    using Min-le
    by blast
  hence enat-leq: enat (Min {(card {v ∈ V. (y, l) ∈ p v}) | y. y ∈ A ∧ y ≠ l})
                  ≤ enat (card {v ∈ V. (w, l) ∈ p v})
    using enat-ord-simps
    by simp
  have ∀ S::(nat set). finite S ⟶ (∀ m. (∀ x ∈ S. m ≤ x) ⟶ (∀ x ∈ S. enat
m ≤ enat x))
    using enat-ord-simps
    by simp
  hence ∀ S::(nat set). finite S ∧ S ≠ {} ⟶ (∀ x. x ∈ S ⟶ enat (Min S) ≤
enat x)
    by simp
  hence ∀ S::(nat set). finite S ∧ S ≠ {} ⟶
        (∀ x. x ∈ {enat x | x. x ∈ S} ⟶ enat (Min S) ≤ x)
    by auto
  moreover have ∀ S::(nat set). finite S ∧ S ≠ {} ⟶ enat (Min S) ∈ {enat x |
x. x ∈ S}
    by simp
  moreover have ∀ S::(nat set). finite S ∧ S ≠ {} ⟶ finite {enat x | x. x ∈ S}
                                                    ∧ {enat x | x. x ∈ S} ≠ {}
    by simp
```

ultimately have $\forall$ S::(*nat set*). *finite* $S \wedge S \neq \{\} \longrightarrow$
enat (*Min* S) = *Min* $\{$*enat* $x \mid x.\ x \in S\}$
using *Min-eqI*
by (*metis* (*no-types*, *lifting*))
moreover have $\{($*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y \neq l\} \neq \{\}$
using *el*
by *auto*
moreover have $\{$*enat* $x \mid x.\ x \in \{($*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y$
$\neq l\}\}$
$= \{$*enat* (*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y \neq l\}$
by *auto*
ultimately have *enat* (*Min* $\{($*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y \neq l\})$
$=$
Min $\{$*enat* (*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y \neq l\}$
using *fin*
by *presburger*
thus *Min* $\{$*enat* (*card* $\{v \in V.\ (y, l) \in p\ v\}) \mid y.\ y \in A \wedge y \neq l\}$
$\leq$ *enat* (*card* $\{v \in V.\ (w, l) \in p\ v\})$
using *enat-leq*
by *simp*
qed

5.16.4 Property

theorem *minimax-score-cond-rating*: *condorcet-rating minimax-score*
proof (*unfold condorcet-rating-def minimax-score.simps prefer-count.simps*,
safe, *rule ccontr*)
fix
A :: $'b$ *set* **and**
V :: $'a$ *set* **and**
p :: $('b, 'a)$ *Profile* **and**
w :: $'b$ **and**
l :: $'b$
assume
winner: *condorcet-winner* $V\ A\ p\ w$ **and**
l-in-A: $l \in A$ **and**
l-neq-w:$l \neq w$ **and**
min-leq:
$\neg$ *Min* $\{$*if finite* V *then enat* (*card* $\{v \in V.$ *let* $r = p\ v$ *in* $y \preceq_r l\})$ *else* $\infty \mid$
$y.\ y \in A - \{l\}\}$
$<$ *Min* $\{$*if finite* V *then*
enat (*card* $\{v \in V.$ *let* $r = p\ v$ *in* $y \preceq_r w\})$ *else*
$\infty \mid y.\ y \in A - \{w\}\}$
hence *min-count-ineq*:
Min $\{$*prefer-count* $V\ p\ l\ y \mid y.\ y \in A - \{l\}\} \geq$
Min $\{$*prefer-count* $V\ p\ w\ y \mid y.\ y \in A - \{w\}\}$
by *simp*
have *pref-count-gte-min*:
prefer-count $V\ p\ l\ w \geq$ *Min* $\{$*prefer-count* $V\ p\ l\ y \mid y\ .\ y \in A - \{l\}\}$

using *l-in-A l-neq-w condorcet-winner.simps winner non-cond-winner-minimax-score minimax-score.simps*
by *metis*
have *l-in-A-without-w*: $l \in A - \{w\}$
using *l-in-A l-neq-w*
by *simp*
hence *pref-counts-non-empty*: $\{$*prefer-count* $V\ p\ w\ y \mid y\ .\ y \in A - \{w\}\} \neq \{\}$
by *blast*
have *finite* $(A - \{w\})$
using *condorcet-winner.simps winner finite-Diff*
by *metis*
hence *finite* $\{$*prefer-count* $V\ p\ w\ y \mid y\ .\ y \in A - \{w\}\}$
by *simp*
hence $\exists\ n \in A - \{w\}$. *prefer-count* $V\ p\ w\ n =$
Min $\{$*prefer-count* $V\ p\ w\ y \mid y\ .\ y \in A - \{w\}\}$
using *pref-counts-non-empty Min-in*
by *fastforce*
then obtain n **where** *pref-count-eq-min*:
prefer-count $V\ p\ w\ n =$
Min $\{$*prefer-count* $V\ p\ w\ y \mid y\ .\ y \in A - \{w\}\}$ **and**
n-not-w: $n \in A - \{w\}$
by *metis*
hence *n-in-A*: $n \in A$
using *DiffE*
by *metis*
have *n-neq-w*: $n \neq w$
using *n-not-w*
by *simp*
have *w-in-A*: $w \in A$
using *winner*
by *simp*
have *pref-count-n-w-ineq*: *prefer-count* $V\ p\ w\ n >$ *prefer-count* $V\ p\ n\ w$
using *n-not-w winner*
by *auto*
have *pref-count-l-w-n-ineq*: *prefer-count* $V\ p\ l\ w \geq$ *prefer-count* $V\ p\ w\ n$
using *pref-count-gte-min min-count-ineq pref-count-eq-min*
by *auto*
hence *prefer-count* $V\ p\ n\ w \geq$ *prefer-count* $V\ p\ w\ l$
using *n-in-A w-in-A l-in-A n-neq-w l-neq-w pref-count-sym winner*
unfolding *condorcet-winner.simps*
by *metis*
hence *prefer-count* $V\ p\ l\ w >$ *prefer-count* $V\ p\ w\ l$
using *n-in-A w-in-A l-in-A n-neq-w l-neq-w pref-count-sym winner pref-count-n-w-ineq pref-count-l-w-n-ineq*
unfolding *condorcet-winner.simps*
by *auto*
hence *wins* $V\ l\ p\ w$
by *simp*
thus *False*

using *l-in-A-without-w wins-antisym winner*
unfolding *condorcet-winner.simps*
by *metis*
qed

theorem *minimax-is-dcc*: *defer-condorcet-consistency minimax*
proof (*unfold defer-condorcet-consistency-def* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
fix
A :: $'b$ *set* **and**
V :: $'a$ *set* **and**
p :: $('b, 'a)$ *Profile*
assume *profile V A p*
hence *well-formed-*$\mathcal{SCF}$ *A* (*max-eliminator minimax-score V A p*)
using *max-elim-sound par-comp-result-sound*
by *metis*
thus *well-formed-*$\mathcal{SCF}$ *A* (*minimax V A p*)
by *simp*
next
fix
A :: $'b$ *set* **and**
V :: $'a$ *set* **and**
p :: $('b, 'a)$ *Profile* **and**
w :: $'b$
assume *cwin-w*: *condorcet-winner V A p w*
have *max-mmaxscore-dcc*:
defer-condorcet-consistency ((*max-eliminator minimax-score*)
::($'b$, $'a$, $'b$ *Result*) *Electoral-Module*)
using *cr-eval-imp-dcc-max-elim minimax-score-cond-rating*
by *metis*
hence
max-eliminator minimax-score V A p =
({},
A − *defer* (*max-eliminator minimax-score*) *V A p*,
{$a \in A$. *condorcet-winner V A p a*})
using *cwin-w*
unfolding *defer-condorcet-consistency-def*
by *blast*
thus
minimax V A p =
({},
A − *defer minimax V A p*,
{$d \in A$. *condorcet-winner V A p d*})
by *simp*
qed

end

Chapter 6

Compositional Structures

6.1 Drop And Pass Compatibility

theory *Drop-And-Pass-Compatibility*
  imports *Basic-Modules/Drop-Module*
        Basic-Modules/Pass-Module
begin

This is a collection of properties about the interplay and compatibility of both the drop module and the pass module.

6.1.1 Properties

theorem *drop-zero-mod-rej-zero*[*simp*]:
  fixes *r* :: *'a Preference-Relation*
  assumes *linear-order r*
  shows *rejects 0* (*drop-module 0 r*)
proof (*unfold rejects-def*, *safe*)
  show *SCF-result.electoral-module* (*drop-module 0 r*)
    using *assms*
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile*
  assume
    fin-A: *finite A* **and**
    prof-A: *profile V A p*
  have *connex UNIV r*
    using *assms lin-ord-imp-connex*
    by *auto*
  hence *connex*: *connex A* (*limit A r*)
    using *limit-presv-connex subset-UNIV*
    by *metis*

have ∀ *B a*. *B* ≠ {} ∨ (*a*::′*a*) ∉ *B*
  by *simp*
hence ∀ *a B*. *a* ∈ *A* ∧ *a* ∈ *B* ⟶ *connex B* (*limit A r*) ⟶
    ¬ *card* (*above* (*limit A r*) *a*) ≤ *0*
  using *above-connex above-presv-limit card-eq-0-iff*
    fin-A finite-subset le-0-eq assms
  by (*metis* (*no-types*))
hence {*a* ∈ *A*. *card* (*above* (*limit A r*) *a*) ≤ *0*} = {}
  using *connex*
  by *auto*
hence *card* {*a* ∈ *A*. *card* (*above* (*limit A r*) *a*) ≤ *0*} = *0*
  using *card.empty*
  by (*metis* (*full-types*))
thus *card* (*reject* (*drop-module 0 r*) *V A p*) = *0*
  by *simp*
qed

The drop module rejects n alternatives (if there are at least n alternatives).

theorem *drop-two-mod-rej-n*[*simp*]:
  fixes *r* :: ′*a Preference-Relation*
  assumes *linear-order r*
  shows *rejects n* (*drop-module n r*)
proof (*unfold rejects-def*, *safe*)
  show *SCF-result.electoral-module* (*drop-module n r*)
    by *simp*
next
  fix
    A :: ′*a set* **and**
    V :: ′*v set* **and**
    p :: (′*a*, ′*v*) *Profile*
  assume
    card-n: *n* ≤ *card A* **and**
    fin-A: *finite A* **and**
    prof: *profile V A p*
  let *?inv-rank* = *the-inv-into A* (*rank* (*limit A r*))
  have *lin-ord-limit*: *linear-order-on A* (*limit A r*)
    using *assms limit-presv-lin-ord*
    by *auto*
  hence (*limit A r*) ⊆ *A* × *A*
    unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
    by *simp*
  hence ∀ *a* ∈ *A*. (*above* (*limit A r*) *a*) ⊆ *A*
    unfolding *above-def*
    by *auto*
  hence *leq*: ∀ *a* ∈ *A*. *rank* (*limit A r*) *a* ≤ *card A*
    using *fin-A*
    by (*simp add*: *card-mono*)
  have ∀ *a* ∈ *A*. {*a*} ⊆ (*above* (*limit A r*) *a*)
    using *lin-ord-limit*

unfolding *linear-order-on-def partial-order-on-def*
preorder-on-def refl-on-def above-def
by *auto*
hence $\forall\ a \in A.\ card\ \{a\} \leq card\ (above\ (limit\ A\ r)\ a)$
using *card-mono fin-A rev-finite-subset above-presv-limit*
by *metis*
hence *geq-1*: $\forall\ a \in A.\ 1 \leq rank\ (limit\ A\ r)\ a$
by *simp*
with *leq* **have** $\forall\ a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ card\ A\}$
by *simp*
hence $rank\ (limit\ A\ r)$ ` $A \subseteq \{1\ ..\ card\ A\}$
by *auto*
moreover have *inj*: $inj\text{-}on\ (rank\ (limit\ A\ r))\ A$
using *fin-A inj-onI rank-unique lin-ord-limit*
by *metis*
ultimately have *bij*: $bij\text{-}betw\ (rank\ (limit\ A\ r))\ A\ \{1\ ..\ card\ A\}$
using *bij-betw-def bij-betw-finite bij-betw-iff-card card-seteq*
dual-order.refl ex-bij-betw-nat-finite-1 fin-A
by *metis*
hence *bij-inv*: $bij\text{-}betw\ ?inv\text{-}rank\ \{1\ ..\ card\ A\}\ A$
using *bij-betw-the-inv-into*
by *blast*
hence $\forall\ S \subseteq \{1..card\ A\}.\ card\ (?inv\text{-}rank$ ` $S) = card\ S$
using *fin-A bij-betw-same-card bij-betw-subset*
by *metis*
moreover have *subset*: $\{1\ ..\ n\} \subseteq \{1\ ..\ card\ A\}$
using *card-n*
by *simp*
ultimately have $card\ (?inv\text{-}rank$ ` $\{1\ ..\ n\}) = n$
using *numeral-One numeral-eq-iff semiring-norm(85) card-atLeastAtMost*
by *presburger*
also have $?inv\text{-}rank$ ` $\{1..n\} = \{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\}$
proof
show $?inv\text{-}rank$ ` $\{1..n\} \subseteq \{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\}$
proof
fix $a :: 'a$
assume $a \in ?inv\text{-}rank$ ` $\{1..n\}$
then obtain b **where** *b-img*: $b \in \{1\ ..\ n\} \wedge ?inv\text{-}rank\ b = a$
by *auto*
hence $rank\ (limit\ A\ r)\ a = b$
using *subset f-the-inv-into-f-bij-betw subsetD bij*
by *metis*
hence $rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}$
using *b-img*
by *simp*
moreover have $a \in A$
using *b-img bij-inv bij-betwE subset*
by *blast*
ultimately show $a \in \{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\}$

by *blast*
qed
next
show $\{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\} \subseteq$ *the-inv-into* $A\ (rank\ (limit\ A\ r))$ ' $\{1\ ..\ n\}$
proof
fix a :: $'a$
assume *el*: $a \in \{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\}$
then obtain b **where** *b-img*: $b \in \{1..n\} \wedge rank\ (limit\ A\ r)\ a = b$
by *auto*
moreover have $a \in A$
using *el*
by *simp*
ultimately have *?inv-rank* $b = a$
using *inj the-inv-into-f-f*
by *metis*
thus $a \in$ *?inv-rank* ' $\{1\ ..\ n\}$
using *b-img*
by *auto*
qed
qed
finally have *card* $\{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1..n\}\} = n$
by *blast*
also have $\{a \in A.\ rank\ (limit\ A\ r)\ a \in \{1\ ..\ n\}\} = \{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\}$
using *geq-1*
by *auto*
also have ... = *reject* (*drop-module* n r) V A p
by *simp*
finally show *card* (*reject* (*drop-module* n r) V A p) = n
by *blast*
qed

The pass and drop module are (disjoint-)compatible.

theorem *drop-pass-disj-compat*[*simp*]:
fixes
r :: $'a$ *Preference-Relation* **and**
n :: *nat*
assumes *linear-order* r
shows *disjoint-compatibility* (*drop-module* n r) (*pass-module* n r)
proof (*unfold disjoint-compatibility-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*drop-module* n r)
using *assms*
by *simp*
next
show $\mathcal{SCF}$*-result.electoral-module* (*pass-module* n r)
using *assms*
by *simp*
next

fix
A :: *'a set* **and**
V :: *'b set*
have *linear-order-on A* (*limit A r*)
using *assms limit-presv-lin-ord*
by *blast*
hence *profile V A* (λ *v.* (*limit A r*))
using *profile-def*
by *blast*
then obtain *p* :: (*'a*, *'b*) *Profile* **where**
profile V A p
by *blast*
show $\exists$ $B \subseteq A$. ($\forall$ $a \in B$. *indep-of-alt* (*drop-module n r*) *V A a* $\wedge$
($\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*drop-module n r*) *V A p*)) $\wedge$
($\forall$ $a \in A - B$. *indep-of-alt* (*pass-module n r*) *V A a* $\wedge$
($\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*pass-module n r*) *V A p*))
proof
have *same-A*:
$\forall$ *p q.* (*profile V A p* $\wedge$ *profile V A q*) $\longrightarrow$
reject (*drop-module n r*) *V A p* = *reject* (*drop-module n r*) *V A q*
by *auto*
let *?A* = *reject* (*drop-module n r*) *V A p*
have *?A* $\subseteq$ *A*
by *auto*
moreover have $\forall$ $a \in$ *?A*. *indep-of-alt* (*drop-module n r*) *V A a*
using *assms*
unfolding *indep-of-alt-def*
by *simp*
moreover have $\forall$ $a \in$ *?A*. $\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*drop-module n r*) *V A p*
by *auto*
moreover have $\forall$ $a \in A -$ *?A*. *indep-of-alt* (*pass-module n r*) *V A a*
using *assms*
unfolding *indep-of-alt-def*
by *simp*
moreover have $\forall$ $a \in A -$ *?A*. $\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*pass-module n r*) *V A p*
by *auto*
ultimately show *?A* $\subseteq$ *A* $\wedge$
($\forall$ $a \in$ *?A*. *indep-of-alt* (*drop-module n r*) *V A a* $\wedge$
($\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*drop-module n r*) *V A p*)) $\wedge$
($\forall$ $a \in A -$ *?A*. *indep-of-alt* (*pass-module n r*) *V A a* $\wedge$
($\forall$ *p. profile V A p* $\longrightarrow$ $a \in$ *reject* (*pass-module n r*) *V A p*))
by *simp*
qed
qed

end

6.2 Revision Composition

theory *Revision-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

A revised electoral module rejects all originally rejected or deferred alternatives, and defers the originally elected alternatives. It does not elect any alternatives.

6.2.1 Definition

fun *revision-composition* :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
$\Rightarrow ('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
revision-composition $m\ V\ A\ p = (\{\}, A - elect\ m\ V\ A\ p, elect\ m\ V\ A\ p)$

abbreviation *rev* :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
$\Rightarrow ('a, 'v, 'a\ Result)$ *Electoral-Module* $(\text{-}\downarrow\ 50)$ **where**
$m\downarrow ==$ *revision-composition* m

6.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
fixes m :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
assumes $\mathcal{SCF}$*-result.electoral-module* m
shows $\mathcal{SCF}$*-result.electoral-module* (*revision-composition* m)
proof −
from *assms*
have $\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow elect\ m\ V\ A\ p \subseteq A$
using *elect-in-alts*
by *metis*
hence $\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow (A - elect\ m\ V\ A\ p) \cup elect\ m\ V\ A\ p = A$
by *blast*
hence *unity*:
$\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow$
set-equals-partition A (*revision-composition* $m\ V\ A\ p$)
by *simp*
have $\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow (A - elect\ m\ V\ A\ p) \cap elect\ m\ V\ A\ p = \{\}$
by *blast*
hence *disjoint*:
$\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow$ *disjoint3* (*revision-composition* $m\ V\ A\ p$)
by *simp*
from *unity disjoint*
show *?thesis*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*

by *simp*
qed

lemma *rev-comp-only-voters*:
fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes *only-voters-vote m*
shows *only-voters-vote* (*revision-composition m*)
using *assms*
unfolding *only-voters-vote-def revision-composition.simps*
by *presburger*

6.2.3 Composition Rules

An electoral module received by revision is never electing.

theorem *rev-comp-non-electing*[*simp*]:
fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes *$\mathcal{SCF}$-result.electoral-module m*
shows *non-electing* (*m*↓)
using *assms*
unfolding *non-electing-def*
by *simp*

Revising an electing electoral module results in a non-blocking electoral module.

theorem *rev-comp-non-blocking*[*simp*]:
fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes *electing m*
shows *non-blocking* (*m*↓)
proof (*unfold non-blocking-def*, *safe*, *simp-all*)
show *$\mathcal{SCF}$-result.electoral-module* (*m*↓)
using *assms rev-comp-sound*
unfolding *electing-def*
by (*metis* (*no-types*, *lifting*))
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
x :: *'a*
assume
fin-A: *finite A* **and**
prof-A: *profile V A p* **and**
no-elect: *A* − *elect m V A p* = *A* **and**
x-in-A: *x* ∈ *A*
from *no-elect* **have** *non-elect*:
non-electing m
using *assms prof-A x-in-A fin-A empty-iff*
Diff-disjoint Int-absorb2 elect-in-alts

unfolding *electing-def*
by (*metis* (*no-types*, *lifting*))
show *False*
using *non-elect assms empty-iff fin-A prof-A x-in-A*
unfolding *electing-def non-electing-def*
by (*metis* (*no-types*, *lifting*))
qed

Revising an invariant monotone electoral module results in a defer-invariant-monotone electoral module.

theorem *rev-comp-def-inv-mono*[*simp*]:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes *invariant-monotonicity* m
shows *defer-invariant-monotonicity* ($m\downarrow$)
proof (*unfold defer-invariant-monotonicity-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* ($m\downarrow$)
using *assms rev-comp-sound*
unfolding *invariant-monotonicity-def*
by *simp*
next
show *non-electing* ($m\downarrow$)
using *assms rev-comp-non-electing*
unfolding *invariant-monotonicity-def*
by *simp*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
q :: ($'a$, $'v$) *Profile* **and**
a :: $'a$ **and**
x :: $'a$ **and**
x' :: $'a$
assume
rev-p-defer-a: $a \in$ *defer* ($m\downarrow$) V A p **and**
a-lifted: *lifted* V A p q a **and**
rev-q-defer-x: $x \in$ *defer* ($m\downarrow$) V A q **and**
x-non-eq-a: $x \neq a$ **and**
rev-q-defer-x': $x' \in$ *defer* ($m\downarrow$) V A q
from *rev-p-defer-a*
have *elect-a-in-p*: $a \in$ *elect* m V A p
by *simp*
from *rev-q-defer-x x-non-eq-a*
have *elect-no-unique-a-in-q*: *elect* m V A $q \neq \{a\}$
by *force*
from *assms*
have *elect* m V A q = *elect* m V A p
using *a-lifted elect-a-in-p elect-no-unique-a-in-q*
unfolding *invariant-monotonicity-def*

by (*metis* (*no-types*))
thus $x' \in$ *defer* $(m\downarrow)$ V A p
using *rev-q-defer-x'*
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a* **and**
x :: *'a* **and**
x' :: *'a*
assume
rev-p-defer-a: $a \in$ *defer* $(m\downarrow)$ V A p **and**
a-lifted: *lifted* V A p q a **and**
rev-q-defer-x: $x \in$ *defer* $(m\downarrow)$ V A q **and**
x-non-eq-a: $x \neq a$ **and**
rev-p-defer-x': $x' \in$ *defer* $(m\downarrow)$ V A p
have *reject-and-defer*:
(A − *elect* m V A q, *elect* m V A q) = *snd* (($m\downarrow$) V A q)
by *force*
have *elect-p-eq-defer-rev-p*: *elect* m V A p = *defer* $(m\downarrow)$ V A p
by *simp*
hence *elect-a-in-p*: $a \in$ *elect* m V A p
using *rev-p-defer-a*
by *presburger*
have *elect* m V A $q \neq \{a\}$
using *rev-q-defer-x x-non-eq-a*
by *force*
with *assms*
show $x' \in$ *defer* $(m\downarrow)$ V A q
using *a-lifted rev-p-defer-x' snd-conv elect-a-in-p*
elect-p-eq-defer-rev-p reject-and-defer
unfolding *invariant-monotonicity-def*
by (*metis* (*no-types*))
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a* **and**
x :: *'a* **and**
x' :: *'a*
assume
$a \in$ *defer* $(m\downarrow)$ V A p **and**
lifted V A p q a **and**
$x' \in$ *defer* $(m\downarrow)$ V A q

with *assms*
show $x' \in defer\ (m\downarrow)\ V\ A\ p$
using *empty-iff insertE snd-conv revision-composition.elims*
unfolding *invariant-monotonicity-def*
by *metis*
next
fix
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ Profile$ **and**
$q :: ('a, 'v)\ Profile$ **and**
$a :: 'a$ **and**
$x :: 'a$ **and**
$x' :: 'a$
assume
rev-p-defer-a: $a \in defer\ (m\downarrow)\ V\ A\ p$ **and**
a-lifted: $lifted\ V\ A\ p\ q\ a$ **and**
rev-q-not-defer-a: $a \notin defer\ (m\downarrow)\ V\ A\ q$
from *assms*
have *lifted-inv*:
$\forall\ A\ V\ p\ q\ a.\ a \in elect\ m\ V\ A\ p \wedge lifted\ V\ A\ p\ q\ a \longrightarrow$
$elect\ m\ V\ A\ q = elect\ m\ V\ A\ p \vee elect\ m\ V\ A\ q = \{a\}$
unfolding *invariant-monotonicity-def*
by (*metis* (*no-types*))
have *p-defer-rev-eq-elect*: $defer\ (m\downarrow)\ V\ A\ p = elect\ m\ V\ A\ p$
by *simp*
have *q-defer-rev-eq-elect*: $defer\ (m\downarrow)\ V\ A\ q = elect\ m\ V\ A\ q$
by *simp*
thus $x' \in defer\ (m\downarrow)\ V\ A\ q$
using *p-defer-rev-eq-elect lifted-inv a-lifted rev-p-defer-a rev-q-not-defer-a*
by *blast*
qed

end

6.3 Sequential Composition

theory *Sequential-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

The sequential composition creates a new electoral module from two electoral modules. In a sequential composition, the second electoral module makes decisions over alternatives deferred by the first electoral module.

6.3.1 Definition

```
fun sequential-composition :: ('a, 'v, 'a Result) Electoral-Module ⇒
                              ('a, 'v, 'a Result) Electoral-Module ⇒
                              ('a, 'v, 'a Result) Electoral-Module where
  sequential-composition m n V A p =
    (let new-A = defer m V A p;
        new-p = limit-profile new-A p in (
                (elect m V A p) ∪ (elect n V new-A new-p),
                (reject m V A p) ∪ (reject n V new-A new-p),
                defer n V new-A new-p))

abbreviation sequence ::
  ('a, 'v, 'a Result) Electoral-Module ⇒ ('a, 'v, 'a Result) Electoral-Module
    ⇒ ('a, 'v, 'a Result) Electoral-Module
     (infix ▷ 50) where
  m ▷ n == sequential-composition m n

fun sequential-composition' :: ('a, 'v, 'a Result) Electoral-Module ⇒
        ('a, 'v, 'a Result) Electoral-Module ⇒ ('a, 'v, 'a Result) Electoral-Module
where
  sequential-composition' m n V A p =
    (let (m-e, m-r, m-d) = m V A p; new-A = m-d;
        new-p = limit-profile new-A p;
        (n-e, n-r, n-d) = n V new-A new-p in
            (m-e ∪ n-e, m-r ∪ n-r, n-d))

lemma seq-comp-only-voters:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module
  assumes
    only-voters-vote m ∧ only-voters-vote n
  shows only-voters-vote (m ▷ n)
proof (unfold only-voters-vote-def, clarify)
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile
  assume coincide: ∀ v ∈ V. p v = p' v
  hence eq: m V A p = m V A p' ∧ n V A p = n V A p'
    using assms
    unfolding only-voters-vote-def
    by blast
  hence coincide-limit:
   ∀ v ∈ V. limit-profile (defer m V A p) p v = limit-profile (defer m V A p') p' v
    using coincide
    by simp
  moreover have
```

elect m V A p ∪ *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
  = *elect m V A p'* ∪ *elect n V* (*defer m V A p'*) (*limit-profile* (*defer m V A p'*) *p'*)
  using *assms eq coincide-limit*
  unfolding *only-voters-vote-def*
  by *metis*
moreover have
  reject m V A p ∪ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
  = *reject m V A p'* ∪ *reject n V* (*defer m V A p'*) (*limit-profile* (*defer m V A p'*) *p'*)
  using *assms eq coincide-limit*
  unfolding *only-voters-vote-def*
  by *metis*
moreover have
  defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
  = *defer n V* (*defer m V A p'*) (*limit-profile* (*defer m V A p'*) *p'*)
  using *assms eq coincide-limit*
  unfolding *only-voters-vote-def*
  by *metis*
ultimately show ($m \triangleright n$) *V A p* = ($m \triangleright n$) *V A p'*
  unfolding *sequential-composition.simps*
  by *metis*
qed

lemma *seq-comp-presv-disj*:
fixes
  m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
  n :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
  A :: '*a set* **and**
  V :: '*v set* **and**
  p :: ('*a*, '*v*) *Profile*
assumes *module-m*: $\mathcal{SCF}$*-result.electoral-module m* **and**
  module-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
  prof: *profile V A p*
shows *disjoint3* (($m \triangleright n$) *V A p*)
proof −
let *?new-A* = *defer m V A p*
let *?new-p* = *limit-profile ?new-A p*
have *prof-def-lim*: *profile V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
  using *def-presv-prof prof module-m*
  by *metis*
have *defer-in-A*:
  ∀ *A' V' p' m' a*.
  (*profile V' A' p'* ∧
  $\mathcal{SCF}$*-result.electoral-module m'* ∧
  (*a*::'*a*) ∈ *defer m' V' A' p'*) ⟶
  a ∈ *A'*
  using *UnCI result-presv-alts*
  by *fastforce*

from *module-m prof*
have *disjoint-m*: *disjoint3* (m V A p)
unfolding $\mathcal{SCF}$*-result.electoral-module-def well-formed-*$\mathcal{SCF}$*.simps*
by *blast*
from *module-m module-n def-presv-prof prof*
have *disjoint-n*: *disjoint3* (n V *?new-A* *?new-p*)
unfolding $\mathcal{SCF}$*-result.electoral-module-def well-formed-*$\mathcal{SCF}$*.simps*
by *metis*
have *disj-n*:
elect m V A p $\cap$ *reject* m V A p $= \{\} \wedge$
elect m V A p $\cap$ *defer* m V A p $= \{\} \wedge$
reject m V A p $\cap$ *defer* m V A p $= \{\}$
using *prof module-m*
by (*simp add*: *result-disj*)
have *reject* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p) $\subseteq$ *defer* m V
A p
using *def-presv-prof reject-in-alts prof module-m module-n*
by *metis*
with *disjoint-m module-m module-n prof*
have *elect-reject-diff*: *elect* m V A p $\cap$ *reject* n V *?new-A* *?new-p* $= \{\}$
using *disj-n*
by *blast*
from *prof module-m module-n*
have *elec-n-in-def-m*:
elect n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p) $\subseteq$ *defer* m V A p
using *def-presv-prof elect-in-alts*
by *metis*
have *elect-defer-diff*: *elect* m V A p $\cap$ *defer* n V *?new-A* *?new-p* $= \{\}$
proof $-$
obtain f :: $'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ **where**
$\forall$ B B'.
($\exists$ a b. $a \in B' \wedge b \in B \wedge a = b$) $=$
(f B $B' \in B' \wedge$ ($\exists$ a. $a \in B \wedge f$ B $B' = a$))
using *disjoint-iff*
by *metis*
then obtain g :: $'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ **where**
$\forall$ B B'.
($B \cap B' = \{\} \longrightarrow$ ($\forall$ a b. $a \in B \wedge b \in B' \longrightarrow a \neq b$)) $\wedge$
($B \cap B' \neq \{\} \longrightarrow f$ B $B' \in B \wedge g$ B $B' \in B' \wedge f$ B $B' = g$ B B')
by *auto*
thus *?thesis*
using *defer-in-A disj-n module-n prof-def-lim prof*
by *fastforce*
qed
have *rej-intersect-new-elect-empty*: *reject* m V A p $\cap$ *elect* n V *?new-A* *?new-p*
$= \{\}$
using *disj-n disjoint-m disjoint-n def-presv-prof prof*
module-m module-n elec-n-in-def-m
by *blast*

have (*elect m V A p* ∪ *elect n V ?new-A ?new-p*) ∩
    (*reject m V A p* ∪ *reject n V ?new-A ?new-p*) = {}
proof (*safe*)
  fix $x :: 'a$
  assume
    $x \in$ *elect m V A p* **and**
    $x \in$ *reject m V A p*
  hence $x \in$ *elect m V A p* ∩ *reject m V A p*
    by *simp*
  thus $x \in \{\}$
    using *disj-n*
    by *simp*
next
  fix $x :: 'a$
  assume
    $x \in$ *elect m V A p* **and**
    $x \in$ *reject n V* (*defer m V A p*)
      (*limit-profile* (*defer m V A p*) *p*)
  thus $x \in \{\}$
    using *elect-reject-diff*
    by *blast*
next
  fix $x :: 'a$
  assume
    $x \in$ *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) **and**
    $x \in$ *reject m V A p*
  thus $x \in \{\}$
    using *rej-intersect-new-elect-empty*
    by *blast*
next
  fix $x :: 'a$
  assume
    $x \in$ *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) **and**
    $x \in$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
  thus $x \in \{\}$
    using *disjoint-iff-not-equal module-n prof-def-lim result-disj prof*
    by *metis*
qed
moreover have
  (*elect m V A p* ∪ *elect n V ?new-A ?new-p*) ∩ (*defer n V ?new-A ?new-p*) = {}
  using *Int-Un-distrib2 Un-empty elect-defer-diff module-n*
      prof-def-lim result-disj prof
  by (*metis* (*no-types*))
moreover have
  (*reject m V A p* ∪ *reject n V ?new-A ?new-p*) ∩ (*defer n V ?new-A ?new-p*) =
{}
proof (*safe*)
  fix $x :: 'a$
  assume

x-in-def: $x \in$ *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) **and**
x-in-rej: $x \in$ *reject m V A p*
from *x-in-def*
have $x \in$ *defer m V A p*
using *defer-in-A module-n prof-def-lim prof*
by *blast*
with *x-in-rej*
have $x \in$ *reject m V A p* $\cap$ *defer m V A p*
by *fastforce*
thus $x \in \{\}$
using *disj-n*
by *blast*
next
fix *x* :: *'a*
assume
$x \in$ *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) **and**
$x \in$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
thus $x \in \{\}$
using *module-n prof-def-lim reject-not-elec-or-def*
by *fastforce*
qed
ultimately have
disjoint3 (*elect m V A p* $\cup$ *elect n V ?new-A ?new-p*,
reject m V A p $\cup$ *reject n V ?new-A ?new-p*,
defer n V ?new-A ?new-p)
by *simp*
thus *?thesis*
unfolding *sequential-composition.simps*
by *metis*
qed

lemma *seq-comp-presv-alts*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes *module-m*: $\mathcal{SCF}$*-result.electoral-module m* **and**
module-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
prof: *profile V A p*
shows *set-equals-partition A* ((*m* $\triangleright$ *n*) *V A p*)
proof −
let *?new-A* = *defer m V A p*
let *?new-p* = *limit-profile ?new-A p*
have *elect-reject-diff*: *elect m V A p* $\cup$ *reject m V A p* $\cup$ *?new-A* = *A*
using *module-m prof*
by (*simp add*: *result-presv-alts*)
have *elect n V ?new-A ?new-p* $\cup$

      reject n V ?new-A ?new-p ∪
        defer n V ?new-A ?new-p = *?new-A*
    using *module-m module-n prof def-presv-prof result-presv-alts*
    by *metis*
  hence (*elect m V A p* ∪ *elect n V ?new-A ?new-p*) ∪
      (*reject m V A p* ∪ *reject n V ?new-A ?new-p*) ∪
        defer n V ?new-A ?new-p = *A*
    using *elect-reject-diff*
    by *blast*
  hence *set-equals-partition A*
      (*elect m V A p* ∪ *elect n V ?new-A ?new-p*,
        reject m V A p ∪ *reject n V ?new-A ?new-p*,
          defer n V ?new-A ?new-p)
    by *simp*
  thus *?thesis*
    unfolding *sequential-composition.simps*
    by *metis*
qed

lemma *seq-comp-alt-eq*[*code*]: *sequential-composition* = *sequential-composition′*
proof (*unfold sequential-composition′.simps sequential-composition.simps*)
  have ∀ *m n V A E*.
      (*case m V A E of* (*e*, *r*, *d*) ⇒
        case n V d (*limit-profile d E*) *of* (*e′*, *r′*, *d′*) ⇒
        (*e* ∪ *e′*, *r* ∪ *r′*, *d′*)) =
          (*elect m V A E* ∪ *elect n V* (*defer m V A E*) (*limit-profile* (*defer m V A E*) *E*),
            reject m V A E ∪ *reject n V* (*defer m V A E*) (*limit-profile* (*defer m V A E*) *E*),
            defer n V (*defer m V A E*) (*limit-profile* (*defer m V A E*) *E*))
    using *case-prod-beta′*
    by (*metis* (*no-types*, *lifting*))
  thus
    (λ *m n V A p*.
        let A′ = *defer m V A p*; *p′* = *limit-profile A′ p in*
      (*elect m V A p* ∪ *elect n V A′ p′*, *reject m V A p* ∪ *reject n V A′ p′*, *defer n V A′ p′*)) =
      (λ *m n V A pr*.
        let (*e*, *r*, *d*) = *m V A pr*; *A′* = *d*; *p′* = *limit-profile A′ pr*;
          (*e′*, *r′*, *d′*) = *n V A′ p′ in*
      (*e* ∪ *e′*, *r* ∪ *r′*, *d′*))
    by *metis*
qed

6.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
  fixes
    m :: (*′a*, *′v*, *′a Result*) *Electoral-Module* **and**

n :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n
shows $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
fix
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a, 'v)$ *Profile*
assume
prof-A: *profile* $V\ A\ p$
have $\forall\ r.$ *well-formed-*$\mathcal{SCF}$ $(A::'a\ set)\ r =$
(*disjoint3* r $\wedge$ *set-equals-partition* $A\ r$)
by *simp*
thus *well-formed-*$\mathcal{SCF}$ $A\ ((m \triangleright n)\ V\ A\ p)$
using *assms seq-comp-presv-disj seq-comp-presv-alts prof-A*
by *metis*
qed

6.3.3 Lemmas

lemma *seq-comp-dec-only-def*:
fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a,'v)$ *Profile*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
prof: *profile* $V\ A\ p$ **and**
empty-defer: *defer* $m\ V\ A\ p = \{\}$
shows $(m \triangleright n)\ V\ A\ p = \ m\ V\ A\ p$
proof –
have $\forall\ m'\ A'\ V'\ p'.$
($\mathcal{SCF}$*-result.electoral-module* m' $\wedge$ *profile* $V'\ A'\ p'$) $\longrightarrow$
profile V' (*defer* $m'\ V'\ A'\ p'$) (*limit-profile* (*defer* $m'\ V'\ A'\ p'$) p')
using *def-presv-prof prof*
by *metis*
hence *prof-no-alt*: *profile* V $\{\}$ (*limit-profile* (*defer* $m\ V\ A\ p$) p)
using *empty-defer prof module-m*
by *metis*
show *?thesis*
proof
have
(*elect* $m\ V\ A\ p$) $\cup$ (*elect* $n\ V$ (*defer* $m\ V\ A\ p$) (*limit-profile* (*defer* $m\ V\ A\ p$) p)) =

```
        elect m V A p
      using elect-in-alts[of n V defer m V A p (limit-profile (defer m V A p) p)]
          empty-defer module-n prof prof-no-alt
      by auto
    thus elect (m ▷ n) V A p = elect m V A p
      using fst-conv
      unfolding sequential-composition.simps
      by metis
  next
    have rej-empty:
      ∀ m' V' p'.
        (𝒮𝒞ℱ-result.electoral-module m'
          ∧ profile V' ({}::'a set) p') ⟶ reject m' V' {} p' = {}
      using bot.extremum-uniqueI reject-in-alts
      by metis
    have (reject m V A p, defer n V {} (limit-profile {} p)) = snd (m V A p)
      using bot.extremum-uniqueI defer-in-alts empty-defer
          module-n prod.collapse prof-no-alt
      by (metis (no-types))
    thus snd ((m ▷ n) V A p) = snd (m V A p)
      using rej-empty empty-defer module-n prof-no-alt prof
      by fastforce
  qed
qed

lemma seq-comp-def-then-elect:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    n-electing-m: non-electing m and
    def-one-m: defers 1 m and
    electing-n: electing n and
    f-prof: finite-profile V A p
  shows elect (m ▷ n) V A p = defer m V A p
proof (cases)
  assume A = {}
  with electing-n n-electing-m f-prof
  show ?thesis
    using bot.extremum-uniqueI defer-in-alts elect-in-alts seq-comp-sound
    unfolding electing-def non-electing-def
    by metis
next
  assume non-empty-A: A ≠ {}
  from n-electing-m f-prof
  have ele: elect m V A p = {}
```

unfolding *non-electing-def*
by *simp*
from *non-empty-A def-one-m f-prof finite*
have *def-card*: *card* (*defer m V A p*) = *1*
unfolding *defers-def*
by (*simp add*: *Suc-leI card-gt-0-iff*)
with *n-electing-m f-prof*
have *def*: $\exists\ a \in A.\ defer\ m\ V\ A\ p = \{a\}$
using *card-1-singletonE defer-in-alts singletonI subsetCE*
unfolding *non-electing-def*
by *metis*
from *ele def n-electing-m*
have *rej*: $\exists\ a \in A.\ reject\ m\ V\ A\ p = A - \{a\}$
using *Diff-empty def-one-m f-prof reject-not-elec-or-def*
unfolding *defers-def*
by *metis*
from *ele rej def n-electing-m f-prof*
have *res-m*: $\exists\ a \in A.\ m\ V\ A\ p = (\{\},\ A - \{a\},\ \{a\})$
using *Diff-empty elect-rej-def-combination reject-not-elec-or-def*
unfolding *non-electing-def*
by *metis*
hence $\exists\ a \in A.\ elect\ (m \triangleright n)\ V\ A\ p = elect\ n\ V\ \{a\}\ (limit\text{-}profile\ \{a\}\ p)$
using *prod.sel sup-bot.left-neutral*
unfolding *sequential-composition.simps*
by *metis*
with *def-card def electing-n n-electing-m f-prof*
have $\exists\ a \in A.\ elect\ (m \triangleright n)\ V\ A\ p = \{a\}$
using *electing-for-only-alt fst-conv def-presv-prof sup-bot.left-neutral*
unfolding *non-electing-def sequential-composition.simps*
by *metis*
with *def def-card electing-n n-electing-m f-prof res-m*
show *?thesis*
using *def-presv-prof electing-for-only-alt fst-conv sup-bot.left-neutral*
unfolding *non-electing-def sequential-composition.simps*
by *metis*
qed

lemma *seq-comp-def-card-bounded*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
$\mathcal{SCF}$*-result.electoral-module n* **and**
finite-profile V A p
shows $card\ (defer\ (m \triangleright n)\ V\ A\ p) \leq card\ (defer\ m\ V\ A\ p)$

using *card-mono defer-in-alts assms def-presv-prof snd-conv finite-subset*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-set-bounded*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n **and**
profile V A p
shows *defer* ($m \triangleright n$) V A $p \subseteq$ *defer* m V A p
using *defer-in-alts assms snd-conv def-presv-prof*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-defers-def-set*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile*
shows *defer* ($m \triangleright n$) V A p = *defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
using *snd-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-then-elect-elec-set*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile*
shows *elect* ($m \triangleright n$) V A p =
elect n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p) $\cup$ (*elect* m V A p)
using *Un-commute fst-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-elim-one-red-def-set*:
fixes

m :: ('a, 'v, 'a Result) *Electoral-Module* **and**
n :: ('a, 'v, 'a Result) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
eliminates 1 n **and**
profile V A p **and**
card (*defer* m V A p) > 1
shows *defer* ($m \triangleright n$) V A $p \subset$ *defer* m V A p
using *assms snd-conv def-presv-prof single-elim-imp-red-def-set*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-set-trans*:
fixes
m :: ('a, 'v, 'a Result) *Electoral-Module* **and**
n :: ('a, 'v, 'a Result) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile* **and**
a :: 'a
assumes
$a \in$ (*defer* ($m \triangleright n$) V A p) **and**
$\mathcal{SCF}$*-result.electoral-module* m $\wedge$ $\mathcal{SCF}$*-result.electoral-module* n **and**
profile V A p
shows $a \in$ *defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p) $\wedge$
$a \in$ *defer* m V A p
using *seq-comp-def-set-bounded assms in-mono seq-comp-defers-def-set*
by (*metis* (*no-types*, *opaque-lifting*))

6.3.4 Composition Rules

The sequential composition preserves the non-blocking property.

theorem *seq-comp-presv-non-blocking*[*simp*]:
fixes
m :: ('a, 'v, 'a Result) *Electoral-Module* **and**
n :: ('a, 'v, 'a Result) *Electoral-Module*
assumes
non-blocking-m: *non-blocking* m **and**
non-blocking-n: *non-blocking* n
shows *non-blocking* ($m \triangleright n$)
proof −
fix
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile*
let *?input-sound* = $A \neq \{\}$ $\wedge$ *finite-profile* V A p

from *non-blocking-m*
have *?input-sound* $\longrightarrow$ *reject* m V A $p \neq A$
  unfolding *non-blocking-def*
  by *simp*
with *non-blocking-m*
have *A-reject-diff*: *?input-sound* $\longrightarrow$ A $-$ *reject* m V A $p \neq \{\}$
  using *Diff-eq-empty-iff reject-in-alts subset-antisym*
  unfolding *non-blocking-def*
  by *metis*
from *non-blocking-m*
have *?input-sound* $\longrightarrow$ *well-formed-$\mathcal{SCF}$* A (m V A p)
  unfolding *$\mathcal{SCF}$-result.electoral-module-def non-blocking-def*
  by *simp*
hence *?input-sound* $\longrightarrow$ *elect* m V A p $\cup$ *defer* m V A p $=$ A $-$ *reject* m V A p
  using *non-blocking-m elec-and-def-not-rej*
  unfolding *non-blocking-def*
  by *metis*
with *A-reject-diff*
have *?input-sound* $\longrightarrow$ *elect* m V A p $\cup$ *defer* m V A $p \neq \{\}$
  by *simp*
hence *?input-sound* $\longrightarrow$ (*elect* m V A $p \neq \{\}$ $\vee$ *defer* m V A $p \neq \{\}$)
  by *simp*
with *non-blocking-m non-blocking-n*
show *?thesis*
proof (*unfold non-blocking-def*)
  assume
    emod-reject-m:
    $\mathcal{SCF}$-result.electoral-module m $\wedge$
      ($\forall$ A V p. $A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p $\longrightarrow$ *reject* m V A $p \neq A$) **and**
    emod-reject-n:
    $\mathcal{SCF}$-result.electoral-module n $\wedge$
      ($\forall$ A V p. $A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p $\longrightarrow$ *reject* n V A $p \neq A$)
  show
    $\mathcal{SCF}$-result.electoral-module ($m \triangleright n$) $\wedge$
      ($\forall$ A V p. $A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p $\longrightarrow$ *reject* ($m \triangleright n$) V A $p \neq A$)
  proof (*safe*)
    show *$\mathcal{SCF}$-result.electoral-module* ($m \triangleright n$)
      using *emod-reject-m emod-reject-n*
      by *simp*
  next
    fix
      A :: *'a set* **and**
      V :: *'v set* **and**
      p :: (*'a*, *'v*) *Profile* **and**
      x :: *'a*
    assume
      fin-A: *finite* A **and**
      prof-A: *profile* V A p **and**
      rej-mn: *reject* ($m \triangleright n$) V A p $=$ A **and**

x-in-A: $x \in A$
from *emod-reject-m fin-A prof-A*
have *fin-defer*:
finite (*defer m V A p*) $\wedge$ *profile V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *def-presv-prof defer-in-alts finite-subset*
by (*metis* (*no-types*))
from *emod-reject-m emod-reject-n fin-A prof-A*
have *seq-elect*:
elect ($m \triangleright n$) *V A p* =
elect n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) $\cup$ *elect m V A p*
using *seq-comp-def-then-elect-elec-set*
by *metis*
from *emod-reject-n emod-reject-m fin-A prof-A*
have *def-limit*:
defer ($m \triangleright n$) *V A p* = *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *seq-comp-defers-def-set*
by *metis*
from *emod-reject-n emod-reject-m fin-A prof-A*
have *elect* ($m \triangleright n$) *V A p* $\cup$ *defer* ($m \triangleright n$) *V A p* = *A* $-$ *reject* ($m \triangleright n$) *V A p*
using *elec-and-def-not-rej seq-comp-sound*
by *metis*
hence *elect-def-disj*:
elect n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) $\cup$
elect m V A p $\cup$
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) = {}
using *def-limit seq-elect Diff-cancel rej-mn*
by *auto*
have *rej-def-eq-set*:
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) $-$
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) = {} $\longrightarrow$
reject n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) =
defer m V A p
using *elect-def-disj emod-reject-n fin-defer*
by (*simp add*: *reject-not-elec-or-def*)
have
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) $-$
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) = {} $\longrightarrow$
elect m V A p = *elect m V A p* $\cap$ *defer m V A p*
using *elect-def-disj*
by *blast*
thus $x \in \{\}$
using *rej-def-eq-set result-disj fin-defer Diff-cancel Diff-empty fin-A prof-A emod-reject-m emod-reject-n reject-not-elec-or-def x-in-A*
by *metis*
qed

  qed
qed

Sequential composition preserves the non-electing property.

theorem *seq-comp-presv-non-electing*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    non-electing m **and**
    non-electing n
  shows *non-electing* (*m* ▷ *n*)
proof (*unfold non-electing-def*, *safe*)
  have $\mathcal{SCF}$*-result.electoral-module m* ∧ $\mathcal{SCF}$*-result.electoral-module n*
    using *assms*
    unfolding *non-electing-def*
    by *blast*
  thus $\mathcal{SCF}$*-result.electoral-module* (*m* ▷ *n*)
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    x :: *'a*
  assume
    profile V A p **and**
    x ∈ *elect* (*m* ▷ *n*) *V A p*
  thus *x* ∈ {}
    using *assms*
    unfolding *non-electing-def*
    using *seq-comp-def-then-elect-elec-set def-presv-prof Diff-empty Diff-partition*
        empty-subsetI
    by *metis*
qed

Composing an electoral module that defers exactly 1 alternative in sequence after an electoral module that is electing results (still) in an electing electoral module.

theorem *seq-comp-electing*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    def-one-m: *defers 1 m* **and**
    electing-n: *electing n*
  shows *electing* (*m* ▷ *n*)
proof −
  have *defer-card-eq-one*:

$\forall$ A V p. (*card* $A \geq 1$ $\wedge$ *finite* A $\wedge$ *profile* V A p) $\longrightarrow$ *card* (*defer* m V A p) = *1*
using *def-one-m*
unfolding *defers-def*
by *metis*
hence *def-m1-not-empty*:
$\forall$ A V p. ($A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p) $\longrightarrow$ *defer* m V A $p \neq \{\}$
using *One-nat-def Suc-leI card-eq-0-iff card-gt-0-iff zero-neq-one*
by *metis*
thus *?thesis*
proof –
have $\forall$ m'.
($\neg$ *electing* m' $\vee$ $\mathcal{SCF}$*-result.electoral-module* m' $\wedge$
($\forall$ A' V' p'. ($A' \neq \{\}$ $\wedge$ *finite* A' $\wedge$ *profile* V' A' p') $\longrightarrow$ *elect* m' V' A' $p' \neq \{\}$))
$\wedge$ (*electing* m' $\vee$ $\neg$ $\mathcal{SCF}$*-result.electoral-module* m' $\vee$
($\exists$ A V p. ($A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p $\wedge$ *elect* m' V A $p = \{\}$)))
unfolding *electing-def*
by *blast*
hence $\forall$ m'.
($\neg$ *electing* m' $\vee$ $\mathcal{SCF}$*-result.electoral-module* m' $\wedge$
($\forall$ A' V' p'. ($A' \neq \{\}$ $\wedge$ *finite* A' $\wedge$ *profile* V' A' p') $\longrightarrow$ *elect* m' V' A' $p' \neq \{\}$))
$\wedge$ ($\exists$ A V p. (*electing* m' $\vee$ $\neg$ $\mathcal{SCF}$*-result.electoral-module* m' $\vee$ $A \neq \{\}$
$\wedge$ *finite* A $\wedge$ *profile* V A p $\wedge$ *elect* m' V A $p = \{\}$))
by *simp*
then obtain
A :: ('*a*, '*v*, '*a Result*) *Electoral-Module* $\Rightarrow$ '*a set* **and**
V :: ('*a*, '*v*, '*a Result*) *Electoral-Module* $\Rightarrow$ '*v set* **and**
p :: ('*a*, '*v*, '*a Result*) *Electoral-Module* $\Rightarrow$ ('*a*, '*v*) *Profile* **where**
f-mod:
$\forall$ m'::('*a*, '*v*, '*a Result*) *Electoral-Module*.
($\neg$ *electing* m' $\vee$ $\mathcal{SCF}$*-result.electoral-module* m' $\wedge$
($\forall$ A' V' p'. ($A' \neq \{\}$ $\wedge$ *finite* A' $\wedge$ *profile* V' A' p')
$\longrightarrow$ *elect* m' V' A' $p' \neq \{\}$)) $\wedge$
(*electing* m' $\vee$ $\neg$ $\mathcal{SCF}$*-result.electoral-module* m' $\vee$ A $m' \neq \{\}$ $\wedge$
finite (A m') $\wedge$ *profile* (V m') (A m') (p m') $\wedge$ *elect* m' (V m') (A m') (p m') = $\{\}$)
by *metis*
hence *f-elect*:
$\mathcal{SCF}$*-result.electoral-module* n $\wedge$
($\forall$ A V p. ($A \neq \{\}$ $\wedge$ *finite* A $\wedge$ *profile* V A p) $\longrightarrow$ *elect* n V A $p \neq \{\}$)
using *electing-n*
unfolding *electing-def*
by *metis*
have *def-card-one*:
$\mathcal{SCF}$*-result.electoral-module* m $\wedge$
($\forall$ A V p. (*1* $\leq$ *card* A $\wedge$ *finite* A $\wedge$ *profile* V A p) $\longrightarrow$ *card* (*defer* m V A p) = *1*)

using *def-one-m defer-card-eq-one*
unfolding *defers-def*
by *blast*
hence $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
using *f-elect seq-comp-sound*
by *metis*
with *f-mod f-elect def-card-one*
show *?thesis*
using *seq-comp-def-then-elect-elec-set def-presv-prof defer-in-alts def-m1-not-empty bot-eq-sup-iff finite-subset*
unfolding *electing-def*
by *metis*
qed
qed

lemma *def-lift-inv-seq-comp-help*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
monotone-m: *defer-lift-invariance* m **and**
monotone-n: *defer-lift-invariance* n **and**
only-voters-n: *only-voters-vote* n **and**
def-and-lifted: $a \in$ (*defer* $(m \triangleright n)$ V A p) $\wedge$ *lifted* V A p q a
shows $(m \triangleright n)$ V A p $=$ $(m \triangleright n)$ V A q
proof −
let *?new-Ap* = *defer* m V A p
let *?new-Aq* = *defer* m V A q
let *?new-p* = *limit-profile ?new-Ap* p
let *?new-q* = *limit-profile ?new-Aq* q
from *monotone-m monotone-n*
have *modules*: $\mathcal{SCF}$*-result.electoral-module* m $\wedge$ $\mathcal{SCF}$*-result.electoral-module* n
unfolding *defer-lift-invariance-def*
by *simp*
hence *profile* V A p $\longrightarrow$ *defer* $(m \triangleright n)$ V A $p \subseteq$ *defer* m V A p
using *seq-comp-def-set-bounded*
by *metis*
moreover have *profile-p*: *lifted* V A p q a $\longrightarrow$ *finite-profile* V A p
unfolding *lifted-def*
by *simp*
ultimately have *defer-subset*: *defer* $(m \triangleright n)$ V A $p \subseteq$ *defer* m V A p
using *def-and-lifted*
by *blast*
hence *mono-m*: m V A p $=$ m V A q

using *monotone-m def-and-lifted modules profile-p*
seq-comp-def-set-trans
unfolding *defer-lift-invariance-def*
by *metis*
hence *new-A-eq*: *?new-Ap* = *?new-Aq*
by *presburger*
have *defer-eq*: *defer* $(m \triangleright n)$ *V A p* = *defer n V ?new-Ap ?new-p*
using *snd-conv*
unfolding *sequential-composition.simps*
by *metis*
have *mono-n*: *n V ?new-Ap ?new-p* = *n V ?new-Aq ?new-q*
proof (*cases*)
assume *lifted V ?new-Ap ?new-p ?new-q a*
thus *?thesis*
using *defer-eq mono-m monotone-n def-and-lifted*
unfolding *defer-lift-invariance-def*
by (*metis* (*no-types*, *lifting*))
next
assume *unlifted-a*: ¬*lifted V ?new-Ap ?new-p ?new-q a*
from *def-and-lifted*
have *finite-profile V A q*
unfolding *lifted-def*
by *simp*
with *modules new-A-eq*
have *prof-p*: *profile V ?new-Ap ?new-q*
using *def-presv-prof*
by (*metis* (*no-types*))
moreover from *modules profile-p def-and-lifted*
have *prof-q*: *profile V ?new-Ap ?new-p*
using *def-presv-prof*
by (*metis* (*no-types*))
moreover from *defer-subset def-and-lifted*
have $a \in$ *?new-Ap*
by *blast*
ultimately have *lifted-stmt*:
(∃ $v \in V$.
Preference-Relation.lifted ?new-Ap (*?new-p v*) (*?new-q v*) *a*) ⟶
(∃ $v \in V$.
¬ *Preference-Relation.lifted ?new-Ap* (*?new-p v*) (*?new-q v*) *a* ∧
(*?new-p v*) ≠ (*?new-q v*))
using *unlifted-a def-and-lifted defer-in-alts infinite-super modules profile-p*
unfolding *lifted-def*
by *metis*
from *def-and-lifted modules*
have ∀ $v \in V$. (*Preference-Relation.lifted A* (*p v*) (*q v*) *a* ∨ (*p v*) = (*q v*))
unfolding *Profile.lifted-def*
by *metis*
with *def-and-lifted modules mono-m*
have ∀ $v \in V$.

(*Preference-Relation.lifted ?new-Ap* (*?new-p v*) (*?new-q v*) *a* ∨
(*?new-p v*) = (*?new-q v*))
using *limit-lifted-imp-eq-or-lifted defer-in-alts*
unfolding *Profile.lifted-def limit-profile.simps*
by (*metis* (*no-types*, *lifting*))
with *lifted-stmt*
have ∀ *v* ∈ *V*. (*?new-p v*) = (*?new-q v*)
by *blast*
with *mono-m*
show *?thesis*
using *leI not-less-zero nth-equalityI only-voters-n*
unfolding *only-voters-vote-def*
by *presburger*
qed
from *mono-m mono-n*
show *?thesis*
unfolding *sequential-composition.simps*
by (*metis* (*full-types*))
qed

Sequential composition preserves the property defer-lift-invariance.

theorem *seq-comp-presv-def-lift-inv*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
defer-lift-invariance m **and**
defer-lift-invariance n **and**
only-voters-vote n
shows *defer-lift-invariance* (*m* ⊳ *n*)
proof (*unfold defer-lift-invariance-def*, *safe*)
show *𝒮𝒞ℱ-result.electoral-module* (*m* ⊳ *n*)
using *assms seq-comp-sound*
unfolding *defer-lift-invariance-def*
by *blast*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assume
a ∈ *defer* (*m* ⊳ *n*) *V A p* **and**
Profile.lifted V A p q a
thus (*m* ⊳ *n*) *V A p* = (*m* ⊳ *n*) *V A q*
unfolding *defer-lift-invariance-def*
using *assms def-lift-inv-seq-comp-help*
by *metis*

qed

Composing a non-blocking, non-electing electoral module in sequence with an electoral module that defers exactly one alternative results in an electoral module that defers exactly one alternative.

theorem *seq-comp-def-one*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    non-blocking-m: *non-blocking m* **and**
    non-electing-m: *non-electing m* **and**
    def-one-n: *defers 1 n*
  shows *defers 1* (*m* ▷ *n*)
proof (*unfold defers-def*, *safe*)
  have $\mathcal{SCF}$*-result.electoral-module m*
    using *non-electing-m*
    unfolding *non-electing-def*
    by *simp*
  moreover have $\mathcal{SCF}$*-result.electoral-module n*
    using *def-one-n*
    unfolding *defers-def*
    by *simp*
  ultimately show $\mathcal{SCF}$*-result.electoral-module* (*m* ▷ *n*)
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile*
  assume
    pos-card: $1 \leq$ *card A* **and**
    fin-A: *finite A* **and**
    prof-A: *profile V A p*
  from *pos-card*
  have $A \neq \{\}$
    by *auto*
  with *fin-A prof-A*
  have *reject m V A p* $\neq$ *A*
    using *non-blocking-m*
    unfolding *non-blocking-def*
    by *simp*
  hence $\exists$ *a*. $a \in A \wedge a \notin$ *reject m V A p*
    using *non-electing-m reject-in-alts fin-A prof-A*
        card-seteq infinite-super subsetI upper-card-bound-for-reject
    unfolding *non-electing-def*
    by *metis*
  hence *defer m V A p* $\neq \{\}$
    using *electoral-mod-defer-elem empty-iff non-electing-m fin-A prof-A*

unfolding *non-electing-def*
by (*metis* (*no-types*))
hence *card* (*defer m V A p*) $\geq$ *1*
using *Suc-leI card-gt-0-iff fin-A prof-A non-blocking-m defer-in-alts infinite-super*
unfolding *One-nat-def non-blocking-def*
by *metis*
moreover have
$\forall\ i\ m'.\ defers\ i\ m' = (\mathcal{SCF}\text{-}result.electoral\text{-}module\ m' \wedge (\forall\ A'\ V'\ p'.\ (i \leq card\ A' \wedge finite\ A' \wedge profile\ V'\ A'\ p') \longrightarrow card\ (defer\ m'\ V'\ A'\ p') = i))$
unfolding *defers-def*
by *simp*
ultimately have
card (*defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)) = *1*
using *def-one-n fin-A prof-A non-blocking-m def-presv-prof card.infinite not-one-le-zero*
unfolding *non-blocking-def*
by *metis*
moreover have
defer ($m \triangleright n$) *V A p* = *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *seq-comp-defers-def-set*
by (*metis* (*no-types*, *opaque-lifting*))
ultimately show *card* (*defer* ($m \triangleright n$) *V A p*) = *1*
by *simp*
qed

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *disj-compat-seq*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
m' :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
compatible: *disjoint-compatibility m n* **and**
module-m': $\mathcal{SCF}$*-result.electoral-module m'* **and**
only-voters: *only-voters-vote m'*
shows *disjoint-compatibility* ($m \triangleright m'$) *n*
proof (*unfold disjoint-compatibility-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* ($m \triangleright m'$)
using *compatible module-m' seq-comp-sound*
unfolding *disjoint-compatibility-def*
by *metis*
next
show $\mathcal{SCF}$*-result.electoral-module n*

using *compatible*
unfolding *disjoint-compatibility-def*
by *metis*
next
fix
S :: $'a$ *set* **and**
V :: $'v$ *set*
have *modules*:
$\mathcal{SCF}$*-result.electoral-module* $(m \triangleright m') \wedge \mathcal{SCF}$*-result.electoral-module* n
using *compatible module-m′ seq-comp-sound*
unfolding *disjoint-compatibility-def*
by *metis*
obtain A :: $'a$ *set* **where** *rej-A*:
$A \subseteq S \wedge$
$(\forall\ a \in A.$
indep-of-alt $m\ V\ S\ a \wedge (\forall\ p.$ *profile* $V\ S\ p \longrightarrow a \in$ *reject* $m\ V\ S\ p)) \wedge$
$(\forall\ a \in S - A.$
indep-of-alt $n\ V\ S\ a \wedge (\forall\ p.$ *profile* $V\ S\ p \longrightarrow a \in$ *reject* $n\ V\ S\ p))$
using *compatible*
unfolding *disjoint-compatibility-def*
by (*metis* (*no-types*, *lifting*))
show
$\exists\ A \subseteq S.$
$(\forall\ a \in A.$ *indep-of-alt* $(m \triangleright m')\ V\ S\ a \wedge$
$(\forall\ p.$ *profile* $V\ S\ p \longrightarrow a \in$ *reject* $(m \triangleright m')\ V\ S\ p)) \wedge$
$(\forall\ a \in S - A.$
indep-of-alt $n\ V\ S\ a \wedge (\forall\ p.$ *profile* $V\ S\ p \longrightarrow a \in$ *reject* $n\ V\ S\ p))$
proof
have $\forall\ a\ p\ q.\ a \in A \wedge$ *equiv-prof-except-a* $V\ S\ p\ q\ a \longrightarrow$
$(m \triangleright m')\ V\ S\ p = (m \triangleright m')\ V\ S\ q$
proof (*safe*)
fix
a :: $'a$ **and**
p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile*
assume
a-in-A: $a \in A$ **and**
lifting-equiv-p-q: *equiv-prof-except-a* $V\ S\ p\ q\ a$
hence *eq-def*: *defer* $m\ V\ S\ p =$ *defer* $m\ V\ S\ q$
using *rej-A*
unfolding *indep-of-alt-def*
by *metis*
from *lifting-equiv-p-q*
have *profiles*: *profile* $V\ S\ p \wedge$ *profile* $V\ S\ q$
unfolding *equiv-prof-except-a-def*
by *simp*
hence (*defer* $m\ V\ S\ p) \subseteq S$
using *compatible defer-in-alts*
unfolding *disjoint-compatibility-def*

by *metis*
moreover have $a \notin$ *defer m V S q*
using *a-in-A compatible defer-not-elec-or-rej*[*of m V A p*]
profiles rej-A IntI emptyE result-disj
unfolding *disjoint-compatibility-def*
by *metis*
ultimately have
$\forall\ v \in V.$ *limit-profile* (*defer m V S p*) *p v* = *limit-profile* (*defer m V S q*) *q*
v
using *lifting-equiv-p-q negl-diff-imp-eq-limit-prof*[*of V S p q a defer m V S*
q]
unfolding *eq-def limit-profile.simps*
by *blast*
with *eq-def*
have m' *V* (*defer m V S p*) (*limit-profile* (*defer m V S p*) *p*) =
m' *V* (*defer m V S q*) (*limit-profile* (*defer m V S q*) *q*)
using *only-voters*
unfolding *only-voters-vote-def*
by *simp*
moreover have *m V S p* = *m V S q*
using *rej-A a-in-A lifting-equiv-p-q*
unfolding *indep-of-alt-def*
by *metis*
ultimately show $(m \triangleright m')$ *V S p* = $(m \triangleright m')$ *V S q*
unfolding *sequential-composition.simps*
by (*metis* (*full-types*))
qed
moreover have $\forall\ a' \in A.\ \forall\ p'.$ *profile V S* $p' \longrightarrow a' \in$ *reject* $(m \triangleright m')$ *V S* p'
using *rej-A UnI1 prod.sel*
unfolding *sequential-composition.simps*
by *metis*
ultimately show $A \subseteq S\ \wedge$
$(\forall\ a' \in A.$ *indep-of-alt* $(m \triangleright m')$ *V S* $a'\ \wedge$
$(\forall\ p'.$ *profile V S* $p' \longrightarrow a' \in$ *reject* $(m \triangleright m')$ *V S* $p'))\ \wedge$
$(\forall\ a' \in S - A.$ *indep-of-alt n V S* $a'\ \wedge$
$(\forall\ p'.$ *profile V S* $p' \longrightarrow a' \in$ *reject n V S* $p'))$
using *rej-A indep-of-alt-def modules*
by (*metis* (*no-types*, *lifting*))
qed
qed

theorem *seq-comp-cond-compat*[*simp*]:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
dcc-m: *defer-condorcet-consistency m* **and**
nb-n: *non-blocking n* **and**
ne-n: *non-electing n*

shows *condorcet-compatibility* $(m \triangleright n)$
proof (*unfold condorcet-compatibility-def*, *safe*)
have $\mathcal{SCF}$*-result.electoral-module m*
using *dcc-m*
unfolding *defer-condorcet-consistency-def*
by *presburger*
moreover have $\mathcal{SCF}$*-result.electoral-module n*
using *nb-n*
unfolding *non-blocking-def*
by *presburger*
ultimately have $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
by *simp*
thus $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
by *presburger*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assume
cw-a: *condorcet-winner V A p a* **and**
a-in-rej-seq-m-n: $a \in$ *reject* $(m \triangleright n)$ *V A p*
hence $\exists$ *a'*. *defer-condorcet-consistency m* $\wedge$ *condorcet-winner V A p a'*
using *dcc-m*
by *blast*
hence *m V A p* $= (\{\}, A - ($*defer m V A p*$), \{a\})$
using *defer-condorcet-consistency-def cw-a cond-winner-unique*
by (*metis* (*no-types*, *lifting*))
have *sound-m*: $\mathcal{SCF}$*-result.electoral-module m*
using *dcc-m*
unfolding *defer-condorcet-consistency-def*
by *presburger*
moreover have $\mathcal{SCF}$*-result.electoral-module n*
using *nb-n*
unfolding *non-blocking-def*
by *presburger*
ultimately have *sound-seq-m-n*: $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
by *simp*
have *def-m*: *defer m V A p* $= \{a\}$
using *cw-a cond-winner-unique dcc-m snd-conv*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
have *rej-m*: *reject m V A p* $= A - \{a\}$
using *cw-a cond-winner-unique dcc-m prod.sel*(*1*) *snd-conv*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
have *elect m V A p* $= \{\}$
using *cw-a def-m rej-m dcc-m prod.sel*(*1*)

unfolding *defer-condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
hence *diff-elect-m*: $A - elect\ m\ V\ A\ p = A$
using *Diff-empty*
by (*metis* (*full-types*))
have *cond-win*:
$finite\ A \wedge finite\ V \wedge profile\ V\ A\ p \wedge a \in A \wedge (\forall\ a'.\ a' \in A - \{a'\} \longrightarrow wins\ V\ a\ p\ a')$
using *cw-a condorcet-winner.simps DiffD2 singletonI*
by (*metis* (*no-types*))
have $\forall\ a'\ A'.\ (a'::'a) \in A' \longrightarrow insert\ a'\ (A' - \{a'\}) = A'$
by *blast*
have *nb-n-full*:
$\mathcal{SCF}$*-result.electoral-module n* $\wedge$
$(\forall\ A'\ V'\ p'.\ A' \neq \{\} \wedge finite\ A' \wedge finite\ V' \wedge profile\ V'\ A'\ p' \longrightarrow reject\ n\ V'\ A'\ p' \neq A')$
using *nb-n non-blocking-def*
by *metis*
have *def-seq-diff*: $defer\ (m \triangleright n)\ V\ A\ p = A - elect\ (m \triangleright n)\ V\ A\ p - reject\ (m \triangleright n)\ V\ A\ p$
using *defer-not-elec-or-rej cond-win sound-seq-m-n*
by *metis*
have *set-ins*: $\forall\ a'\ A'.\ (a'::'a) \in A' \longrightarrow insert\ a'\ (A' - \{a'\}) = A'$
by *fastforce*
have $\forall\ p'\ A'\ p''.\ p' = (A'::'a\ set,\ p''::'a\ set \times 'a\ set) \longrightarrow snd\ p' = p''$
by *simp*
hence $snd\ (elect\ m\ V\ A\ p \cup elect\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p),$
$reject\ m\ V\ A\ p \cup reject\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p),$
$defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p)) =$
$(reject\ m\ V\ A\ p \cup reject\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p),$
$defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p))$
by *blast*
hence *seq-snd-simplified*:
$snd\ ((m \triangleright n)\ V\ A\ p) =$
$(reject\ m\ V\ A\ p \cup reject\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p),$
$defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p))$
using *sequential-composition.simps*
by *metis*
hence *seq-rej-union-eq-rej*:
$reject\ m\ V\ A\ p \cup reject\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p) =$
$reject\ (m \triangleright n)\ V\ A\ p$
by *simp*
hence *seq-rej-union-subset-A*:
$reject\ m\ V\ A\ p \cup reject\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p)$

$\subseteq A$
  using *sound-seq-m-n cond-win reject-in-alts*
  by (*metis* (*no-types*))
hence $A - \{a\} = reject\ (m \triangleright n)\ V\ A\ p - \{a\}$
  using *seq-rej-union-eq-rej defer-not-elec-or-rej cond-win def-m diff-elect-m double-diff rej-m sound-m sup-ge1*
  by (*metis* (*no-types*))
hence $reject\ (m \triangleright n)\ V\ A\ p \subseteq A - \{a\}$
  using *seq-rej-union-subset-A seq-snd-simplified set-ins def-seq-diff nb-n-full cond-win fst-conv Diff-empty Diff-eq-empty-iff a-in-rej-seq-m-n def-m def-presv-prof sound-m ne-n diff-elect-m insert-not-empty defer-in-alts reject-not-elec-or-def seq-comp-def-then-elect-elec-set finite-subset seq-comp-defers-def-set sup-bot.left-neutral*
  unfolding *non-electing-def*
  by (*metis* (*no-types*, *lifting*))
thus *False*
  using *a-in-rej-seq-m-n*
  by *blast*
next
fix
  A :: $'a$ *set* **and**
  V :: $'v$ *set* **and**
  p :: $('a, 'v)$ *Profile* **and**
  a :: $'a$ **and**
  a' :: $'a$
assume
  cw-a: *condorcet-winner* $V\ A\ p\ a$ **and**
  not-cw-a': $\neg$ *condorcet-winner* $V\ A\ p\ a'$ **and**
  a'-in-elect-seq-m-n: $a' \in elect\ (m \triangleright n)\ V\ A\ p$
hence $\exists\ a''.$ *defer-condorcet-consistency* $m \wedge$ *condorcet-winner* $V\ A\ p\ a''$
  using *dcc-m*
  by *blast*
hence *result-m*: $m\ V\ A\ p = (\{\}, A - (defer\ m\ V\ A\ p), \{a\})$
  using *defer-condorcet-consistency-def cw-a cond-winner-unique*
  by (*metis* (*no-types*, *lifting*))
have *sound-m*: $\mathcal{SCF}$-*result.electoral-module* m
  using *dcc-m*
  unfolding *defer-condorcet-consistency-def*
  by *presburger*
moreover have $\mathcal{SCF}$-*result.electoral-module* n
  using *nb-n*
  unfolding *non-blocking-def*
  by *presburger*
ultimately have *sound-seq-m-n*: $\mathcal{SCF}$-*result.electoral-module* $(m \triangleright n)$
  by *simp*
have $reject\ m\ V\ A\ p = A - \{a\}$
  using *cw-a dcc-m prod.sel(1) snd-conv result-m*
  unfolding *defer-condorcet-consistency-def*
  by (*metis* (*mono-tags*, *lifting*))

hence *a′-in-rej*: $a' \in$ *reject m V A p*
using *Diff-iff cw-a not-cw-a′ a′-in-elect-seq-m-n condorcet-winner.elims*(*1*)
elect-in-alts singleton-iff sound-seq-m-n subset-iff
by (*metis* (*no-types*, *lifting*))
have $\forall$ *p′ A′ p″*. *p′* = (*A′*::*′a set*, *p″*::*′a set* $\times$ *′a set*) $\longrightarrow$ *snd p′* = *p″*
by *simp*
hence *m-seq-n*:
snd (*elect m V A p* $\cup$ *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*),
reject m V A p $\cup$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*),
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)) =
(*reject m V A p* $\cup$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*),
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*))
by *blast*
have $a' \in$ *elect m V A p*
using *a′-in-elect-seq-m-n condorcet-winner.simps cw-a def-presv-prof ne-n*
seq-comp-def-then-elect-elec-set sound-m sup-bot.left-neutral
unfolding *non-electing-def*
by (*metis* (*no-types*))
hence *a-in-rej-union*:
$a \in$ *reject m V A p* $\cup$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *Diff-iff a′-in-rej condorcet-winner.simps cw-a*
reject-not-elec-or-def sound-m
by (*metis* (*no-types*))
have *m-seq-n-full*:
($m \triangleright n$) *V A p* =
(*elect m V A p* $\cup$ *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*),
reject m V A p $\cup$ *reject n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*),
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*))
unfolding *sequential-composition.simps*
by *metis*
have $\forall$ *A′ A″*. (*A′*::*′a set*) = *fst* (*A′*, *A″*::*′a set*)
by *simp*
hence $a \in$ *reject* ($m \triangleright n$) *V A p*
using *a-in-rej-union m-seq-n m-seq-n-full*
by *presburger*
moreover have
finite A $\wedge$ *finite V* $\wedge$ *profile V A p* $\wedge$ $a \in A$ $\wedge$ ($\forall$ *a″*. $a'' \in A - \{a\} \longrightarrow$ *wins V a p a″*)
using *cw-a m-seq-n-full a′-in-elect-seq-m-n a′-in-rej ne-n sound-m*
unfolding *condorcet-winner.simps*
by *metis*
ultimately show *False*
using *a′-in-elect-seq-m-n IntI empty-iff result-disj sound-seq-m-n a′-in-rej def-presv-prof*
fst-conv m-seq-n-full ne-n non-electing-def sound-m sup-bot.right-neutral

by *metis*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$ **and**
a' :: $'a$
assume
cw-a: *condorcet-winner* V A p a **and**
a'*-in-A*: $a' \in A$ **and**
not-cw-a': $\neg$ *condorcet-winner* V A p a'
have *reject* m V A p $= A - \{a\}$
using *cw-a cond-winner-unique dcc-m prod.sel(1) snd-conv*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*mono-tags*, *lifting*))
moreover have $a \neq a'$
using *cw-a not-cw-a'*
by *safe*
ultimately have $a' \in$ *reject* m V A p
using *DiffI a'-in-A singletonD*
by (*metis* (*no-types*))
hence $a' \in$ *reject* m V A p $\cup$ *reject* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
by *blast*
moreover have
$(m \triangleright n)$ V A p $=$
(*elect* m V A p $\cup$ *elect* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p),
reject m V A p $\cup$ *reject* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p),
defer n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p))
unfolding *sequential-composition.simps*
by *metis*
moreover have
snd (*elect* m V A p $\cup$ *elect* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p),
reject m V A p $\cup$ *reject* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p),
defer n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)) $=$
(*reject* m V A p $\cup$ *reject* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p),
defer n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p))
using *snd-conv*
by *metis*
ultimately show $a' \in$ *reject* $(m \triangleright n)$ V A p
using *fst-eqD*
by (*metis* (*no-types*))
qed

Composing a defer-condorcet-consistent electoral module in sequence with a non-blocking and non-electing electoral module results in a defer-condorcet-consistent module.

theorem *seq-comp-dcc*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    dcc-m: *defer-condorcet-consistency m* **and**
    nb-n: *non-blocking n* **and**
    ne-n: *non-electing n*
  shows *defer-condorcet-consistency* ($m \triangleright n$)
proof (*unfold defer-condorcet-consistency-def*, *safe*)
  have $\mathcal{SCF}$*-result.electoral-module m*
    using *dcc-m*
    unfolding *defer-condorcet-consistency-def*
    by *metis*
  thus $\mathcal{SCF}$*-result.electoral-module* ($m \triangleright n$)
    using *ne-n*
    unfolding *non-electing-def*
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    a :: *'a*
  assume *cw-a*: *condorcet-winner V A p a*
  hence $\exists$ *a'*. *defer-condorcet-consistency m* $\wedge$ *condorcet-winner V A p a'*
    using *dcc-m*
    by *blast*
  hence *result-m*: *m V A p* = ({}, *A* − (*defer m V A p*), {*a*})
    using *defer-condorcet-consistency-def cw-a cond-winner-unique*
    by (*metis* (*no-types*, *lifting*))
  hence *elect-m-empty*: *elect m V A p* = {}
    using *eq-fst-iff*
    by *metis*
  have *sound-m*: $\mathcal{SCF}$*-result.electoral-module m*
    using *dcc-m*
    unfolding *defer-condorcet-consistency-def*
    by *metis*
  hence *sound-seq-m-n*: $\mathcal{SCF}$*-result.electoral-module* ($m \triangleright n$)
    using *ne-n*
    unfolding *non-electing-def*
    by *simp*
  have *defer-eq-a*: *defer* ($m \triangleright n$) *V A p* = {*a*}
  proof (*safe*)
    fix *a'* :: *'a*
    assume *a'-in-def-seq-m-n*: *a'* $\in$ *defer* ($m \triangleright n$) *V A p*
    have {*a*} = {*a* $\in$ *A*. *condorcet-winner V A p a*}

using *cond-winner-unique cw-a*
by *metis*
moreover have *defer-condorcet-consistency m* $\longrightarrow$
m V A p = ({}, *A* − *defer m V A p*, {*a* ∈ *A*. *condorcet-winner V A p a*})
using *cw-a defer-condorcet-consistency-def*
by (*metis* (*no-types*))
ultimately have *defer m V A p* = {*a*}
using *dcc-m snd-conv*
by (*metis* (*no-types*, *lifting*))
hence *defer* (*m* ▷ *n*) *V A p* = {*a*}
using *cw-a a′-in-def-seq-m-n condorcet-winner.elims*(*2*) *empty-iff*
seq-comp-def-set-bounded sound-m subset-singletonD nb-n
unfolding *non-blocking-def*
by *metis*
thus *a′* = *a*
using *a′-in-def-seq-m-n*
by *blast*
next
have ∃ *a′*. *defer-condorcet-consistency m* ∧ *condorcet-winner V A p a′*
using *cw-a dcc-m*
by *blast*
hence *m V A p* = ({}, *A* − (*defer m V A p*), {*a*})
using *defer-condorcet-consistency-def cw-a cond-winner-unique*
by (*metis* (*no-types*, *lifting*))
hence *elect-m-empty*: *elect m V A p* = {}
using *eq-fst-iff*
by *metis*
have *profile V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *condorcet-winner.simps cw-a def-presv-prof sound-m*
by (*metis* (*no-types*))
hence *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) = {}
using *ne-n non-electing-def*
by *metis*
hence *elect* (*m* ▷ *n*) *V A p* = {}
using *elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral*
by (*metis* (*no-types*))
moreover have *condorcet-compatibility* (*m* ▷ *n*)
using *dcc-m nb-n ne-n*
by *simp*
hence *a* ∉ *reject* (*m* ▷ *n*) *V A p*
unfolding *condorcet-compatibility-def*
using *cw-a*
by *metis*
ultimately show *a* ∈ *defer* (*m* ▷ *n*) *V A p*
using *cw-a electoral-mod-defer-elem empty-iff*
sound-seq-m-n condorcet-winner.simps
by *metis*
qed
have *profile V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)

using *condorcet-winner.simps cw-a def-presv-prof sound-m*
by (*metis* (*no-types*))
hence *elect n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*) = {}
using *ne-n*
unfolding *non-electing-def*
by *metis*
hence *elect* ($m \triangleright n$) *V A p* = {}
using *elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral*
by (*metis* (*no-types*))
moreover have *def-seq-m-n-eq-a*: *defer* ($m \triangleright n$) *V A p* = {*a*}
using *cw-a defer-eq-a*
by (*metis* (*no-types*))
ultimately have ($m \triangleright n$) *V A p* = ({}, *A* − {*a*}, {*a*})
using *Diff-empty cw-a elect-rej-def-combination*
reject-not-elec-or-def sound-seq-m-n condorcet-winner.simps
by (*metis* (*no-types*))
moreover have {*a'* ∈ *A*. *condorcet-winner V A p a'*} = {*a*}
using *cw-a cond-winner-unique*
by *metis*
ultimately show ($m \triangleright n$) *V A p*
= ({}, *A* − *defer* ($m \triangleright n$) *V A p*, {*a'* ∈ *A*. *condorcet-winner V A p a'*})
using *def-seq-m-n-eq-a*
by *metis*
qed

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *seq-comp-mono*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
def-monotone-m: *defer-lift-invariance m* **and**
non-ele-m: *non-electing m* **and**
def-one-m: *defers 1 m* **and**
electing-n: *electing n*
shows *monotonicity* ($m \triangleright n$)
proof (*unfold monotonicity-def*, *safe*)
have $\mathcal{SCF}$*-result.electoral-module m*
using *non-ele-m*
unfolding *non-electing-def*
by *simp*
moreover have $\mathcal{SCF}$*-result.electoral-module n*
using *electing-n*
unfolding *electing-def*
by *simp*
ultimately show $\mathcal{SCF}$*-result.electoral-module* ($m \triangleright n$)
by *simp*

next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    q :: (*'a*, *'v*) *Profile* **and**
    w :: *'a*
  assume
    elect-w-in-p: *w* ∈ *elect* (*m* ▷ *n*) *V A p* **and**
    lifted-w: *Profile.lifted V A p q w*
  thus *w* ∈ *elect* (*m* ▷ *n*) *V A q*
    unfolding *lifted-def*
    using *seq-comp-def-then-elect lifted-w assms*
    unfolding *defer-lift-invariance-def*
    by *metis*
qed

Composing a defer-invariant-monotone electoral module in sequence before a non-electing, defer-monotone electoral module that defers exactly 1 alternative results in a defer-lift-invariant electoral module.

theorem *def-inv-mono-imp-def-lift-inv*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    strong-def-mon-m: *defer-invariant-monotonicity m* **and**
    non-electing-n: *non-electing n* **and**
    defers-one: *defers 1 n* **and**
    defer-monotone-n: *defer-monotonicity n* **and**
    only-voters: *only-voters-vote n*
  shows *defer-lift-invariance* (*m* ▷ *n*)
proof (*unfold defer-lift-invariance-def*, *safe*)
  have *𝒮𝒞ℱ-result.electoral-module m*
    using *strong-def-mon-m*
    unfolding *defer-invariant-monotonicity-def*
    by *metis*
  moreover have *𝒮𝒞ℱ-result.electoral-module n*
    using *defers-one*
    unfolding *defers-def*
    by *metis*
  ultimately show *𝒮𝒞ℱ-result.electoral-module* (*m* ▷ *n*)
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    q :: (*'a*, *'v*) *Profile* **and**
    a :: *'a*

assume
defer-a-p: $a \in$ *defer* $(m \triangleright n)$ V A p **and**
lifted-a: *Profile.lifted* V A p q a
have *non-electing-m*: *non-electing* m
using *strong-def-mon-m*
unfolding *defer-invariant-monotonicity-def*
by *simp*
have *electoral-mod-m*: $\mathcal{SCF}$*-result.electoral-module* m
using *strong-def-mon-m*
unfolding *defer-invariant-monotonicity-def*
by *metis*
have *electoral-mod-n*: $\mathcal{SCF}$*-result.electoral-module* n
using *defers-one*
unfolding *defers-def*
by *metis*
have *finite-profile-p*: *finite-profile* V A p
using *lifted-a*
unfolding *Profile.lifted-def*
by *simp*
have *finite-profile-q*: *finite-profile* V A q
using *lifted-a*
unfolding *Profile.lifted-def*
by *simp*
have $1 \leq$ *card* A
using *Profile.lifted-def card-eq-0-iff emptyE less-one lifted-a linorder-le-less-linear*
by *metis*
hence *n-defers-exactly-one-p*: *card* (*defer* n V A p) $= 1$
using *finite-profile-p defers-one*
unfolding *defers-def*
by (*metis* (*no-types*))
have *fin-prof-def-m-q*: *profile* V (*defer* m V A q) (*limit-profile* (*defer* m V A q) q)
using *def-presv-prof electoral-mod-m finite-profile-q*
by (*metis* (*no-types*))
have *def-seq-m-n-q*:
defer $(m \triangleright n)$ V A q $=$ *defer* n V (*defer* m V A q) (*limit-profile* (*defer* m V A q) q)
using *seq-comp-defers-def-set*
by *simp*
have *prof-def-m*: *profile* V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
using *def-presv-prof electoral-mod-m finite-profile-p*
by (*metis* (*no-types*))
hence *prof-seq-comp-m-n*:
profile V (*defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p))
(*limit-profile* (*defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p))
(*limit-profile* (*defer* m V A p) p))
using *def-presv-prof electoral-mod-n*
by (*metis* (*no-types*))
have *a-non-empty*: $a \notin \{\}$

by *simp*
have *def-seq-m-n*:
defer $(m \triangleright n)$ *V A p* = *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
using *seq-comp-defers-def-set*
by *simp*
have $1 \leq$ *card* (*defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*))
using *a-non-empty card-gt-0-iff defer-a-p electoral-mod-n prof-def-m seq-comp-defers-def-set One-nat-def Suc-leI defer-in-alts electoral-mod-m finite-profile-p finite-subset*
by (*metis* (*mono-tags*))
hence *card* (*defer n V* (*defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*))
(*limit-profile* (*defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*))
(*limit-profile* (*defer m V A p*) *p*)))) = *1*
using *n-defers-exactly-one-p prof-seq-comp-m-n defers-one defer-in-alts electoral-mod-m finite-profile-p finite-subset prof-def-m*
unfolding *defers-def*
by *metis*
hence *defer-seq-m-n-eq-one*: *card* (*defer* $(m \triangleright n)$ *V A p*) = *1*
using *One-nat-def Suc-leI a-non-empty card-gt-0-iff def-seq-m-n defer-a-p defers-one electoral-mod-m prof-def-m finite-profile-p seq-comp-def-set-trans defer-in-alts rev-finite-subset*
unfolding *defers-def*
by *metis*
hence *def-seq-m-n-eq-a*: *defer* $(m \triangleright n)$ *V A p* = $\{a\}$
using *defer-a-p is-singleton-altdef is-singleton-the-elem singletonD*
by (*metis* (*no-types*))
show $(m \triangleright n)$ *V A p* = $(m \triangleright n)$ *V A q*
proof (*cases*)
assume *defer m V A q* $\neq$ *defer m V A p*
hence *defer m V A q* = $\{a\}$
using *defer-a-p electoral-mod-n finite-profile-p lifted-a seq-comp-def-set-trans strong-def-mon-m*
unfolding *defer-invariant-monotonicity-def*
by (*metis* (*no-types*))
moreover from *this*
have ($a \in$ *defer m V A p*) $\longrightarrow$ *card* (*defer* $(m \triangleright n)$ *V A q*) = *1*
using *card-eq-0-iff card-insert-disjoint defers-one electoral-mod-m empty-iff order-refl finite.emptyI seq-comp-defers-def-set def-presv-prof finite-profile-q finite.insertI*
unfolding *One-nat-def defers-def*
by *metis*
moreover have $a \in$ *defer m V A p*
using *electoral-mod-m electoral-mod-n defer-a-p seq-comp-def-set-bounded finite-profile-p finite-profile-q*
by *blast*
ultimately have *defer* $(m \triangleright n)$ *V A q* = $\{a\}$
using *Collect-mem-eq card-1-singletonE empty-Collect-eq insertCI subset-singletonD*

def-seq-m-n-q defer-in-alts electoral-mod-n fin-prof-def-m-q
by (*metis* (*no-types*, *lifting*))
hence *defer* $(m \triangleright n)$ V A p = *defer* $(m \triangleright n)$ V A q
using *def-seq-m-n-eq-a*
by *presburger*
moreover have *elect* $(m \triangleright n)$ V A p = *elect* $(m \triangleright n)$ V A q
using *prof-def-m fin-prof-def-m-q finite-profile-p finite-profile-q non-electing-def non-electing-m non-electing-n seq-comp-def-then-elect-elec-set*
by *metis*
ultimately show *?thesis*
using *electoral-mod-m electoral-mod-n eq-def-and-elect-imp-eq finite-profile-p finite-profile-q seq-comp-sound*
by (*metis* (*no-types*))
next
assume $\neg$ (*defer* m V A q $\neq$ *defer* m V A p)
hence *def-eq*: *defer* m V A q = *defer* m V A p
by *presburger*
have *elect* m V A $p = \{\}$
using *finite-profile-p non-electing-m*
unfolding *non-electing-def*
by *simp*
moreover have *elect* m V A $q = \{\}$
using *finite-profile-q non-electing-m*
unfolding *non-electing-def*
by *simp*
ultimately have *elect-m-equal*: *elect* m V A p = *elect* m V A q
by *simp*
have ($\forall$ $v \in V$. (*limit-profile* (*defer* m V A p) p) v = (*limit-profile* (*defer* m V A p) q) v)
$\vee$ *lifted* V (*defer* m V A q) (*limit-profile* (*defer* m V A p) p) (*limit-profile* (*defer* m V A p) q) a
using *def-eq defer-in-alts electoral-mod-m lifted-a finite-profile-q limit-prof-eq-or-lifted*
by *metis*
moreover have
($\forall$ $v \in V$. (*limit-profile* (*defer* m V A p) p) v = (*limit-profile* (*defer* m V A p) q) v)
$\Longrightarrow$ n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
= n V (*defer* m V A q) (*limit-profile* (*defer* m V A q) q)
using *only-voters def-eq*
unfolding *only-voters-vote-def*
by *presburger*
moreover have
lifted V (*defer* m V A q) (*limit-profile* (*defer* m V A p) p)
(*limit-profile* (*defer* m V A p) q) a
$\Longrightarrow$ *defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
= *defer* n V (*defer* m V A q) (*limit-profile* (*defer* m V A q) q)
proof –
assume *lifted*:

Profile.lifted V (*defer m V A q*) (*limit-profile* (*defer m V A p*) *p*)
(*limit-profile* (*defer m V A p*) *q*) *a*
hence $a \in$ *defer n V* (*defer m V A q*) (*limit-profile* (*defer m V A q*) *q*)
using *lifted-a def-seq-m-n defer-a-p defer-monotone-n*
fin-prof-def-m-q def-eq
unfolding *defer-monotonicity-def*
by *metis*
hence $a \in$ *defer* ($m \triangleright n$) *V A q*
using *def-seq-m-n-q*
by *simp*
moreover have *card* (*defer* ($m \triangleright n$) *V A q*) = *1*
using *def-seq-m-n-q defers-one def-eq defer-seq-m-n-eq-one defers-def lifted electoral-mod-m fin-prof-def-m-q finite-profile-p seq-comp-def-card-bounded Profile.lifted-def*
by *metis*
ultimately have *defer* ($m \triangleright n$) *V A q* = $\{a\}$
using *a-non-empty card-1-singletonE insertE*
by *metis*
thus *defer n V* (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
= *defer n V* (*defer m V A q*) (*limit-profile* (*defer m V A q*) *q*)
using *def-seq-m-n-eq-a def-seq-m-n-q def-seq-m-n*
by *presburger*
qed
ultimately have *defer* ($m \triangleright n$) *V A p* = *defer* ($m \triangleright n$) *V A q*
using *def-seq-m-n def-seq-m-n-q*
by *presburger*
hence *defer* ($m \triangleright n$) *V A p* = *defer* ($m \triangleright n$) *V A q*
using *a-non-empty def-eq def-seq-m-n def-seq-m-n-q*
defer-a-p defer-monotone-n finite-profile-p
defer-seq-m-n-eq-one defers-one electoral-mod-m
fin-prof-def-m-q
unfolding *defers-def*
by (*metis* (*no-types*, *lifting*))
moreover from *this*
have *reject* ($m \triangleright n$) *V A p* = *reject* ($m \triangleright n$) *V A q*
using *electoral-mod-m electoral-mod-n finite-profile-p finite-profile-q non-electing-def non-electing-m non-electing-n eq-def-and-elect-imp-eq seq-comp-presv-non-electing*
by (*metis* (*no-types*))
ultimately have *snd* (($m \triangleright n$) *V A p*) = *snd* (($m \triangleright n$) *V A q*)
using *prod-eqI*
by *metis*
moreover have *elect* ($m \triangleright n$) *V A p* = *elect* ($m \triangleright n$) *V A q*
using *prof-def-m fin-prof-def-m-q non-electing-n finite-profile-p finite-profile-q non-electing-def def-eq elect-m-equal fst-conv*
unfolding *sequential-composition.simps*
by (*metis* (*no-types*))
ultimately show ($m \triangleright n$) *V A p* = ($m \triangleright n$) *V A q*
using *prod-eqI*
by *metis*

  qed
qed

end

6.4 Parallel Composition

theory *Parallel-Composition*
  imports *Basic-Modules/Component-Types/Aggregator*
    Basic-Modules/Component-Types/Electoral-Module
begin

The parallel composition composes a new electoral module from two electoral modules combined with an aggregator. Therein, the two modules each make a decision and the aggregator combines them to a single (aggregated) result.

6.4.1 Definition

fun *parallel-composition* :: $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
    $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
    $'a$ *Aggregator* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
  parallel-composition m n agg V A p = *agg A* $(m\ V\ A\ p)\ (n\ V\ A\ p)$

abbreviation *parallel* :: $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ $'a$ *Aggregator* $\Rightarrow$
    $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module*
    (- $\|$- - [*50*, *1000*, *51*] *50*) **where**
  $m \|_a n$ == *parallel-composition m n a*

6.4.2 Soundness

theorem *par-comp-sound*[*simp*]:
  fixes
    m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
    n :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
    a :: $'a$ *Aggregator*
  assumes
    $\mathcal{SCF}$*-result.electoral-module m* **and**
    $\mathcal{SCF}$*-result.electoral-module n* **and**
    aggregator a
  shows $\mathcal{SCF}$*-result.electoral-module* $(m \|_a n)$
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: $('a, 'v)$ *Profile*

```
  assume
    profile V A p
  moreover have
    ∀ a'. aggregator a' =
      (∀ A' e r d e' r' d'.
        (well-formed-SCF (A'::'a set) (e, r', d)
        ∧ well-formed-SCF A' (r, d', e'))
            ⟶ well-formed-SCF A' (a' A' (e, r', d) (r, d', e')))
    unfolding aggregator-def
    by blast
  moreover have
    ∀ m' V' A' p'.
      (SCF-result.electoral-module m' ∧ finite (A'::'a set)
        ∧ finite (V'::'v set) ∧ profile V' A' p') ⟶ well-formed-SCF A' (m' V' A'
p')
    using par-comp-result-sound
    by (metis (no-types))
  ultimately have well-formed-SCF A (a A (m V A p) (n V A p))
    using elect-rej-def-combination assms
    by (metis par-comp-result-sound)
  thus well-formed-SCF A ((m ∥a n) V A p)
    by simp
qed
```

6.4.3 Composition Rule

Using a conservative aggregator, the parallel composition preserves the property non-electing.

```
theorem conserv-agg-presv-non-electing[simp]:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    a :: 'a Aggregator
  assumes
    non-electing-m: non-electing m and
    non-electing-n: non-electing n and
    conservative: agg-conservative a
  shows non-electing (m ∥a n)
proof (unfold non-electing-def, safe)
  have SCF-result.electoral-module m
    using non-electing-m
    unfolding non-electing-def
    by simp
  moreover have SCF-result.electoral-module n
    using non-electing-n
    unfolding non-electing-def
    by simp
  moreover have aggregator a
    using conservative
```

unfolding *agg-conservative-def*
by *simp*
ultimately show $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_a n)$
using *par-comp-sound*
by *simp*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: *('a, 'v) Profile* **and**
w :: *'a*
assume
prof-A: *profile V A p* **and**
w-wins: $w \in$ *elect* $(m \parallel_a n)$ *V A p*
have *emod-m*: $\mathcal{SCF}$*-result.electoral-module m*
using *non-electing-m*
unfolding *non-electing-def*
by *simp*
have *emod-n*: $\mathcal{SCF}$*-result.electoral-module n*
using *non-electing-n*
unfolding *non-electing-def*
by *simp*
have $\forall$ $r\ r'\ d\ d'\ e\ e'\ A'\ f$.
((*well-formed-*$\mathcal{SCF}$ $(A'::'a\ set)$ (e', r', d') $\wedge$
well-formed-$\mathcal{SCF}$ A' (e, r, d)) $\longrightarrow$
elect-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e$ $\wedge$
reject-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r$ $\wedge$
defer-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d) =$
((*well-formed-*$\mathcal{SCF}$ A' (e', r', d') $\wedge$
well-formed-$\mathcal{SCF}$ A' (e, r, d)) $\longrightarrow$
elect-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e$ $\wedge$
reject-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r$ $\wedge$
defer-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d)$
by *linarith*
hence $\forall$ a'. *agg-conservative* a' $=$
(*aggregator* a' $\wedge$
($\forall$ $A'\ e\ e'\ d\ d'\ r\ r'$.
(*well-formed-*$\mathcal{SCF}$ $(A'::'a\ set)$ (e, r, d) $\wedge$
well-formed-$\mathcal{SCF}$ A' (e', r', d')) $\longrightarrow$
elect-r $(a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e'$ $\wedge$
reject-r $(a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq r \cup r'$ $\wedge$
defer-r $(a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq d \cup d'$))
unfolding *agg-conservative-def*
by *simp*
hence *aggregator* a $\wedge$
($\forall$ $A'\ e\ e'\ d\ d'\ r\ r'$.
(*well-formed-*$\mathcal{SCF}$ A' (e, r, d) $\wedge$
well-formed-$\mathcal{SCF}$ A' (e', r', d')) $\longrightarrow$
elect-r $(a\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e'$ $\wedge$

$reject\text{-}r\ (a\ A'\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq r \cup r' \wedge$
$defer\text{-}r\ (a\ A'\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq d \cup d')$
using *conservative*
by *presburger*
hence *let* $c = (a\ A\ (m\ V\ A\ p)\ (n\ V\ A\ p))$ *in*
$(elect\text{-}r\ c \subseteq ((elect\ m\ V\ A\ p) \cup (elect\ n\ V\ A\ p)))$
using *emod-m emod-n par-comp-result-sound*
prod.collapse prof-A
by *metis*
hence $w \in ((elect\ m\ V\ A\ p) \cup (elect\ n\ V\ A\ p))$
using *w-wins*
by *auto*
thus $w \in \{\}$
using *sup-bot-right prof-A*
non-electing-m non-electing-n
unfolding *non-electing-def*
by (*metis* (*no-types*, *lifting*))
qed

end

6.5 Loop Composition

theory *Loop-Composition*
imports *Basic-Modules/Component-Types/Termination-Condition*
Basic-Modules/Defer-Module
Sequential-Composition
begin

The loop composition uses the same module in sequence, combined with a termination condition, until either

- the termination condition is met or
- no new decisions are made (i.e., a fixed point is reached).

6.5.1 Definition

lemma *loop-termination-helper*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
$\neg\ t\ (acc\ V\ A\ p)$ **and**
$defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$ **and**
$finite\ (defer\ acc\ V\ A\ p)$
shows $((acc \triangleright m, m, t, V, A, p), (acc, m, t, V, A, p)) \in$
$measure\ (\lambda\ (acc, m, t, V, A, p).\ card\ (defer\ acc\ V\ A\ p))$
using *assms psubset-card-mono*
by *simp*

This function handles the accumulator for the following loop composition function.

function *loop-comp-helper* ::
$('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
$'a$ *Termination-Condition* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
$finite\ (defer\ acc\ V\ A\ p) \land (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
$\longrightarrow t\ (acc\ V\ A\ p) \Longrightarrow$
$loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = acc\ V\ A\ p\ |$
$\neg\ (finite\ (defer\ acc\ V\ A\ p) \land (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
$\longrightarrow t\ (acc\ V\ A\ p)) \Longrightarrow$
$loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ p$
proof −
fix
P :: *bool* **and**
accum ::
$('a, 'v, 'a\ Result)$ *Electoral-Module* $\times$ $('a, 'v, 'a\ Result)$ *Electoral-Module*
$\times$ $'a$ *Termination-Condition* $\times$ $'v$ *set* $\times$ $'a$ *set* $\times$ $('a, 'v)$ *Profile*
have *accum-exists*: $\exists\ m\ n\ t\ V\ A\ p.\ (m, n, t, V, A, p) = accum$
using *prod-cases5*
by *metis*
assume
$\bigwedge acc\ V\ A\ p\ m\ t.$
$finite\ (defer\ acc\ V\ A\ p) \land defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$
$\longrightarrow t\ (acc\ V\ A\ p) \Longrightarrow accum = (acc, m, t, V, A, p) \Longrightarrow P$ **and**
$\bigwedge acc\ V\ A\ p\ m\ t.$
$\neg\ (finite\ (defer\ acc\ V\ A\ p) \land defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$
$\longrightarrow t\ (acc\ V\ A\ p)) \Longrightarrow accum = (acc, m, t, V, A, p) \Longrightarrow P$
thus P
using *accum-exists*
by *metis*
next
fix
t :: $'a$ *Termination-Condition* **and**
acc :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**

t' :: $'a$ *Termination-Condition* **and**
acc' :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
A' :: $'a$ *set* **and**
V' :: $'v$ *set* **and**
p' :: $('a, 'v)$ *Profile* **and**
m' :: $('a, 'v, 'a$ *Result*) *Electoral-Module*
assume
finite (*defer acc* V A p) $\wedge$ *defer* (*acc* $\triangleright$ m) V A $p \subset$ *defer acc* V A p
$\longrightarrow$ t (*acc* V A p) **and**
finite (*defer* acc' V' A' p') $\wedge$ *defer* ($acc' \triangleright m'$) V' A' $p' \subset$ *defer* acc' V' A' p'
$\longrightarrow$ t' (acc' V' A' p') **and**
$(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$
thus $acc\ V\ A\ p = acc'\ V'\ A'\ p'$
by *fastforce*
next
fix
t :: $'a$ *Termination-Condition* **and**
acc :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
m :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
t' :: $'a$ *Termination-Condition* **and**
acc' :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
A' :: $'a$ *set* **and**
V' :: $'v$ *set* **and**
p' :: $('a, 'v)$ *Profile* **and**
m' :: $('a, 'v, 'a$ *Result*) *Electoral-Module*
assume
finite (*defer acc* V A p) $\wedge$ *defer* (*acc* $\triangleright$ m) V A $p \subset$ *defer acc* V A p
$\longrightarrow$ t (*acc* V A p) **and**
$\neg$ (*finite* (*defer* acc' V' A' p') $\wedge$ *defer* ($acc' \triangleright m'$) V' A' $p' \subset$ *defer* acc' V' A'
p'
$\longrightarrow$ t' (acc' V' A' p')) **and**
$(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$
thus $acc\ V\ A\ p =$ *loop-comp-helper-sumC* ($acc' \triangleright m'$, m', t', V', A', p')
by *force*
next
fix
t :: $'a$ *Termination-Condition* **and**
acc :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
m :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
t' :: $'a$ *Termination-Condition* **and**
acc' :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **and**
A' :: $'a$ *set* **and**
V' :: $'v$ *set* **and**

p' :: $('a, 'v)$ *Profile* **and**
m' :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
assume
$\neg$ (*finite* (*defer acc V A p*) $\wedge$ *defer* (*acc* $\triangleright$ *m*) *V A p* $\subset$ *defer acc V A p*
$\longrightarrow$ *t* (*acc V A p*)) **and**
$\neg$ (*finite* (*defer acc' V' A' p'*) $\wedge$ *defer* (*acc'* $\triangleright$ *m'*) *V' A' p'* $\subset$ *defer acc' V' A'* *p'*
$\longrightarrow$ *t'* (*acc' V' A' p'*)) **and**
$(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$
thus *loop-comp-helper-sumC* (*acc* $\triangleright$ *m*, *m*, *t*, *V*, *A*, *p*) =
loop-comp-helper-sumC (*acc'* $\triangleright$ *m'*, *m'*, *t'*, *V'*, *A'*, *p'*)
by *force*
qed
termination
proof (*safe*)
fix
m :: $('b, 'a, 'b\ Result)$ *Electoral-Module* **and**
n :: $('b, 'a, 'b\ Result)$ *Electoral-Module* **and**
t :: $'b$ *Termination-Condition* **and**
A :: $'b$ *set* **and**
V :: $'a$ *set* **and**
p :: $('b, 'a)$ *Profile*
have *term-rel*:
$\exists$ *R*. *wf R* $\wedge$
(*finite* (*defer m V A p*) $\wedge$ *defer* (*m* $\triangleright$ *n*) *V A p* $\subset$ *defer m V A p* $\longrightarrow$ *t* (*m* *V A p*) $\vee$
$((m \triangleright n, n, t, V, A, p), (m, n, t, V, A, p)) \in R$)
using *loop-termination-helper wf-measure termination*
by (*metis* (*no-types*))
obtain
R :: $((('b, 'a, 'b\ Result)\ Electoral\text{-}Module \times ('b, 'a, 'b\ Result)\ Electoral\text{-}Module \times$
$('b\ Termination\text{-}Condition) \times 'a\ set \times 'b\ set \times ('b, 'a)\ Profile) \times$
$('b, 'a, 'b\ Result)\ Electoral\text{-}Module \times ('b, 'a, 'b\ Result)\ Electoral\text{-}Module \times$
$('b\ Termination\text{-}Condition) \times 'a\ set \times 'b\ set \times ('b, 'a)\ Profile)$ *set* **where**
wf R $\wedge$
(*finite* (*defer m V A p*) $\wedge$ *defer* (*m* $\triangleright$ *n*) *V A p* $\subset$ *defer m V A p* $\longrightarrow$ *t* (*m V* *A p*) $\vee$
$((m \triangleright n, n, t, V, A, p), m, n, t, V, A, p) \in R$)
using *term-rel*
by *presburger*
have $\forall$ *R'*.
All (*loop-comp-helper-dom* ::
$('b, 'a, 'b\ Result)\ Electoral\text{-}Module \times ('b, 'a, 'b\ Result)\ Electoral\text{-}Module$
$\times\ 'b\ Termination\text{-}Condition \times 'a\ set \times 'b\ set \times ('b, 'a)\ Profile \Rightarrow bool$) $\vee$
($\exists$ *t' m' A' V' p' n'*. *wf R'* $\longrightarrow$
$((m' \triangleright n', n', t', V'::'a\ set, A'::'b\ set, p'), m', n', t', V', A', p') \notin R' \wedge$
finite (*defer m' V' A' p'*) $\wedge$ *defer* (*m'* $\triangleright$ *n'*) *V' A' p'* $\subset$ *defer m' V' A' p'*

$\wedge$
$\neg\ t'\ (m'\ V'\ A'\ p'))$
using *termination*
by *metis*
thus *loop-comp-helper-dom* $(m,\ n,\ t,\ V,\ A,\ p)$
using *loop-termination-helper wf-measure*
by *metis*
qed

lemma *loop-comp-code-helper*[*code*]:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
shows
loop-comp-helper acc m t V A p =
(*if* (*t* (*acc V A p*) $\vee \neg$ ((*defer* (*acc* $\triangleright$ *m*) *V A p*) $\subset$ (*defer acc V A p*)) $\vee$
infinite (*defer acc V A p*))
then (*acc V A p*) *else* (*loop-comp-helper* (*acc* $\triangleright$ *m*) *m t V A p*))
using *loop-comp-helper.simps*
by (*metis* (*no-types*))

function *loop-composition* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* $\Rightarrow$ '*a Termination-Condition*
$\Rightarrow$ ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
t ({}, {}, *A*) $\Longrightarrow$ *loop-composition m t V A p* = *defer-module V A p* |
$\neg$(*t* ({}, {}, *A*)) $\Longrightarrow$ *loop-composition m t V A p* = (*loop-comp-helper m m t*) *V A p*
by (*fastforce*, *simp-all*)
termination
using *termination wf-empty*
by *blast*

abbreviation *loop* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* $\Rightarrow$ '*a Termination-Condition*
$\Rightarrow$ ('*a*, '*v*, '*a Result*) *Electoral-Module* (- $\circlearrowleft$- *50*) **where**
$m \circlearrowleft_t \equiv$ *loop-composition m t*

lemma *loop-comp-code*[*code*]:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
shows *loop-composition m t V A p* =
(*if* (*t* ({},{},*A*))

then (*defer-module* V A p) *else* (*loop-comp-helper* m m t) V A p)
by *simp*

lemma *loop-comp-helper-imp-partit*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
acc :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
n :: *nat*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
profile: *profile* V A p **and**
module-acc: $\mathcal{SCF}$*-result.electoral-module* *acc* **and**
defer-card-n: n = *card* (*defer acc* V A p)
shows *well-formed-*$\mathcal{SCF}$ A (*loop-comp-helper acc* m t V A p)
using *assms*
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
have $\forall$ m' n'.
($\mathcal{SCF}$*-result.electoral-module* m' $\wedge$ $\mathcal{SCF}$*-result.electoral-module* n')
$\longrightarrow$ $\mathcal{SCF}$*-result.electoral-module* ($m' \triangleright n'$)
by *auto*
hence $\mathcal{SCF}$*-result.electoral-module* (*acc* $\triangleright$ m)
using *less.prems module-m*
by *blast*
hence $\neg$ t (*acc* V A p) $\wedge$ *defer* (*acc* $\triangleright$ m) V A p $\subset$ *defer acc* V A p $\wedge$
finite (*defer acc* V A p) $\longrightarrow$
well-formed-$\mathcal{SCF}$ A (*loop-comp-helper acc* m t V A p)
using *less.hyps less.prems loop-comp-helper.simps*(*2*)
psubset-card-mono
by *metis*
moreover have *well-formed-*$\mathcal{SCF}$ A (*acc* V A p)
using *less.prems profile*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *blast*
ultimately show *?case*
using *loop-comp-code-helper*
by (*metis* (*no-types*))
qed

6.5.2 Soundness

theorem *loop-comp-sound*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
t :: $'a$ *Termination-Condition*

assumes $\mathcal{SCF}$*-result.electoral-module m*
shows $\mathcal{SCF}$*-result.electoral-module* $(m \circlearrowleft_t)$
using *def-mod-sound loop-composition.simps*
loop-comp-helper-imp-partit assms
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
by *metis*

lemma *loop-comp-helper-imp-no-def-incr*:
fixes
m :: ('a, 'v, 'a Result) *Electoral-Module* **and**
t :: 'a *Termination-Condition* **and**
acc :: ('a, 'v, 'a Result) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile* **and**
n :: *nat*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module m* **and**
profile: *profile V A p* **and**
mod-acc: $\mathcal{SCF}$*-result.electoral-module acc* **and**
card-n-defer-acc: $n = card\ (defer\ acc\ V\ A\ p)$
shows $defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ V\ A\ p \subseteq defer\ acc\ V\ A\ p$
using *assms*
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
have *emod-acc-m*: $\mathcal{SCF}$*-result.electoral-module* $(acc \triangleright m)$
using *less.prems module-m seq-comp-sound*
by *blast*
have $\forall\ A\ A'.\ (finite\ A \wedge A' \subset A) \longrightarrow card\ A' < card\ A$
using *psubset-card-mono*
by *metis*
hence $\neg\ t\ (acc\ V\ A\ p) \wedge defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p \wedge$
$finite\ (defer\ acc\ V\ A\ p) \longrightarrow$
$defer\ (loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t)\ V\ A\ p \subseteq defer\ acc\ V\ A\ p$
using *emod-acc-m less.hyps less.prems*
by *blast*
hence $\neg\ t\ (acc\ V\ A\ p) \wedge defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p \wedge$
$finite\ (defer\ acc\ V\ A\ p) \longrightarrow$
$defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ V\ A\ p \subseteq defer\ acc\ V\ A\ p$
using *loop-comp-helper.simps*(*2*)
by *metis*
thus *?case*
using *eq-iff loop-comp-code-helper*
by (*metis* (*no-types*))
qed

6.5.3 Lemmas

lemma *loop-comp-helper-def-lift-inv-helper*:

fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
acc :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
n :: *nat*
assumes
monotone-m: *defer-lift-invariance m* **and**
prof: *profile V A p* **and**
dli-acc: *defer-lift-invariance acc* **and**
card-n-defer: n = *card* (*defer acc V A p*) **and**
defer-finite: *finite* (*defer acc V A p*) **and**
only-voters-m: *only-voters-vote m*
shows
$\forall$ *q a*. $a \in$ (*defer* (*loop-comp-helper acc m t*) *V A p*) $\wedge$ *lifted V A p q a* $\longrightarrow$
(*loop-comp-helper acc m t*) *V A p* = (*loop-comp-helper acc m t*) *V A q*
using *assms*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
case (*less n*)
have *defer-card-comp*:
defer-lift-invariance acc $\longrightarrow$
($\forall$ *q a*. $a \in$ (*defer* (*acc* $\triangleright$ *m*) *V A p*) $\wedge$ *lifted V A p q a* $\longrightarrow$
card (*defer* (*acc* $\triangleright$ *m*) *V A p*) = *card* (*defer* (*acc* $\triangleright$ *m*) *V A q*))
using *monotone-m def-lift-inv-seq-comp-help only-voters-m*
by *metis*
have *defer-lift-invariance acc* $\longrightarrow$
($\forall$ *q a*. $a \in$ (*defer acc V A p*) $\wedge$ *lifted V A p q a* $\longrightarrow$
card (*defer acc V A p*) = *card* (*defer acc V A q*))
unfolding *defer-lift-invariance-def*
by *simp*
hence *defer-card-acc*:
defer-lift-invariance acc $\longrightarrow$
($\forall$ *q a*. ($a \in$ (*defer* (*acc* $\triangleright$ *m*) *V A p*) $\wedge$ *lifted V A p q a*) $\longrightarrow$
card (*defer acc V A p*) = *card* (*defer acc V A q*))
using *assms seq-comp-def-set-trans*
unfolding *defer-lift-invariance-def*
by *metis*
thus *?case*
proof (*cases*)
assume *card-unchanged*: *card* (*defer* (*acc* $\triangleright$ *m*) *V A p*) = *card* (*defer acc V A*
p)
have *defer-lift-invariance acc* $\longrightarrow$
($\forall$ *q a*. $a \in$ (*defer acc V A p*) $\wedge$ *lifted V A p q a* $\longrightarrow$
(*loop-comp-helper acc m t*) *V A q* = *acc V A q*)
proof (*safe*)
fix
q :: $('a, 'v)$ *Profile* **and**

$a :: 'a$
assume
dli-acc: *defer-lift-invariance acc* **and**
a-in-def-acc: $a \in$ *defer acc V A p* **and**
lifted-A: *Profile.lifted V A p q a*
moreover have $\mathcal{SCF}$*-result.electoral-module m*
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have *emod-acc*: $\mathcal{SCF}$*-result.electoral-module acc*
using *dli-acc*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have *acc-eq-pq*: *acc V A q* = *acc V A p*
using *a-in-def-acc dli-acc lifted-A*
unfolding *defer-lift-invariance-def*
by (*metis* (*full-types*))
ultimately have *finite* (*defer acc V A p*)
$\longrightarrow$ *loop-comp-helper acc m t V A q* = *acc V A q*
using *card-unchanged defer-card-comp prof loop-comp-code-helper*
psubset-card-mono dual-order.strict-iff-order
seq-comp-def-set-bounded less
by (*metis* (*mono-tags*, *lifting*))
thus *loop-comp-helper acc m t V A q* = *acc V A q*
using *acc-eq-pq loop-comp-code-helper*
by (*metis* (*full-types*))
qed
moreover from *card-unchanged*
have (*loop-comp-helper acc m t*) *V A p* = *acc V A p*
using *loop-comp-code-helper order.strict-iff-order psubset-card-mono*
by *metis*
ultimately have
defer-lift-invariance (*acc* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance acc* $\longrightarrow$
($\forall$ *q a*. $a \in$ (*defer* (*loop-comp-helper acc m t*) *V A p*) $\wedge$ *lifted V A p q a*
$\longrightarrow$
(*loop-comp-helper acc m t*) *V A p* = (*loop-comp-helper acc m t*) *V*
A q)
unfolding *defer-lift-invariance-def*
by *metis*
moreover have *defer-lift-invariance* (*acc* $\triangleright$ *m*)
using *less monotone-m seq-comp-presv-def-lift-inv*
by *simp*
ultimately show *?thesis*
using *less monotone-m*
by *metis*
next
assume *card-changed*: $\neg$ (*card* (*defer* (*acc* $\triangleright$ *m*) *V A p*) = *card* (*defer acc V A*
p))
with *prof*

have *card-smaller-for-p*:
SCF-result.electoral-module acc ∧ *finite A* ⟶
card (*defer* (*acc* ▷ *m*) *V A p*) < *card* (*defer acc V A p*)
using *monotone-m order.not-eq-order-implies-strict*
card-mono less.prems seq-comp-def-set-bounded
unfolding *defer-lift-invariance-def*
by *metis*
with *defer-card-acc defer-card-comp*
have *card-changed-for-q*:
defer-lift-invariance acc ⟶
(∀ *q a*. *a* ∈ (*defer* (*acc* ▷ *m*) *V A p*) ∧ *lifted V A p q a* ⟶
card (*defer* (*acc* ▷ *m*) *V A q*) < *card* (*defer acc V A q*))
using *lifted-def less*
unfolding *defer-lift-invariance-def*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
proof (*cases*)
assume *t-not-satisfied-for-p*: ¬ *t* (*acc V A p*)
hence *t-not-satisfied-for-q*:
defer-lift-invariance acc ⟶
(∀ *q a*. *a* ∈ (*defer* (*acc* ▷ *m*) *V A p*) ∧ *lifted V A p q a* ⟶ ¬ *t* (*acc V*
A q))
using *monotone-m prof seq-comp-def-set-trans*
unfolding *defer-lift-invariance-def*
by *metis*
have *dli-card-def*:
defer-lift-invariance (*acc* ▷ *m*) ∧ *defer-lift-invariance acc* ⟶
(∀ *q a*. *a* ∈ (*defer* (*acc* ▷ *m*) *V A p*) ∧ *Profile.lifted V A p q a* ⟶
card (*defer* (*acc* ▷ *m*) *V A q*) ≠ (*card* (*defer acc V A q*)))
proof −
have
∀ *m'*.
(¬ *defer-lift-invariance m'* ∧ *SCF-result.electoral-module m'* ⟶
(∃ *V' A' p' q' a*.
m' V' A' p' ≠ *m' V' A' q'* ∧ *lifted V' A' p' q' a* ∧ *a* ∈ *defer m' V'*
A' p')) ∧
(*defer-lift-invariance m'* ⟶
SCF-result.electoral-module m' ∧
(∀ *V' A' p' q' a*.
m' V' A' p' ≠ *m' V' A' q'* ⟶ *lifted V' A' p' q' a* ⟶ *a* ∉ *defer*
m' V' A' p'))
unfolding *defer-lift-invariance-def*
by *blast*
thus *?thesis*
using *card-changed monotone-m prof seq-comp-def-set-trans*
by (*metis* (*no-types*, *opaque-lifting*))
qed
hence *dli-def-subset*:
defer-lift-invariance (*acc* ▷ *m*) ∧ *defer-lift-invariance acc* ⟶

$(\forall\ p'\ a.\ a \in (\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p) \land \mathit{lifted}\ V\ A\ p\ p'\ a \longrightarrow$
$\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p' \subset \mathit{defer}\ \mathit{acc}\ V\ A\ p')$
using *Profile.lifted-def dli-card-def defer-lift-invariance-def monotone-m psubsetI seq-comp-def-set-bounded*
by (*metis* (*no-types*, *opaque-lifting*))
with *t-not-satisfied-for-p*
have *rec-step-q*:
$\mathit{defer\text{-}lift\text{-}invariance}\ (\mathit{acc} \triangleright m) \land \mathit{defer\text{-}lift\text{-}invariance}\ \mathit{acc} \longrightarrow$
$(\forall\ q\ a.\ a \in (\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p) \land \mathit{lifted}\ V\ A\ p\ q\ a \longrightarrow$
$\mathit{loop\text{-}comp\text{-}helper}\ \mathit{acc}\ m\ t\ V\ A\ q = \mathit{loop\text{-}comp\text{-}helper}\ (\mathit{acc} \triangleright m)\ m\ t\ V\ A\ q)$
proof (*safe*)
fix
$q :: ('a, 'v)\ \mathit{Profile}$ **and**
$a :: 'a$
assume
a-in-def-impl-def-subset:
$\forall\ q'\ a'.\ a' \in \mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p \land \mathit{lifted}\ V\ A\ p\ q'\ a' \longrightarrow$
$\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ q' \subset \mathit{defer}\ \mathit{acc}\ V\ A\ q'$ **and**
dli-acc: *defer-lift-invariance acc* **and**
a-in-def-seq-acc-m: $a \in \mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p$ **and**
lifted-pq-a: *lifted V A p q a*
hence $\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ q \subset \mathit{defer}\ \mathit{acc}\ V\ A\ q$
by *metis*
moreover have $\mathcal{SCF}$*-result.electoral-module acc*
using *dli-acc*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have $\neg\ t\ (\mathit{acc}\ V\ A\ q)$
using *dli-acc a-in-def-seq-acc-m lifted-pq-a t-not-satisfied-for-q*
by *metis*
ultimately show $\mathit{loop\text{-}comp\text{-}helper}\ \mathit{acc}\ m\ t\ V\ A\ q$
$= \mathit{loop\text{-}comp\text{-}helper}\ (\mathit{acc} \triangleright m)\ m\ t\ V\ A\ q$
using *loop-comp-code-helper defer-in-alts finite-subset lifted-pq-a*
unfolding *lifted-def*
by (*metis* (*mono-tags*, *lifting*))
qed
have *rec-step-p*:
$\mathcal{SCF}$*-result.electoral-module acc* $\longrightarrow$
$\mathit{loop\text{-}comp\text{-}helper}\ \mathit{acc}\ m\ t\ V\ A\ p = \mathit{loop\text{-}comp\text{-}helper}\ (\mathit{acc} \triangleright m)\ m\ t\ V\ A\ p$
proof (*safe*)
assume *emod-acc*: $\mathcal{SCF}$*-result.electoral-module acc*
have *sound-imp-defer-subset*:
$\mathcal{SCF}$*-result.electoral-module m* $\longrightarrow \mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p \subseteq \mathit{defer}\ \mathit{acc}\ V\ A\ p$
using *emod-acc prof seq-comp-def-set-bounded*
by *blast*
hence *card-ineq*: $\mathit{card}\ (\mathit{defer}\ (\mathit{acc} \triangleright m)\ V\ A\ p) < \mathit{card}\ (\mathit{defer}\ \mathit{acc}\ V\ A\ p)$
using *card-changed card-mono less order-neq-le-trans*

unfolding *defer-lift-invariance-def*
by *metis*
have *def-limited-acc*:
profile V (*defer acc V A p*) (*limit-profile* (*defer acc V A p*) *p*)
using *def-presv-prof emod-acc prof*
by *metis*
have *defer* (*acc* ▷ *m*) *V A p* ⊆ *defer acc V A p*
using *sound-imp-defer-subset defer-lift-invariance-def monotone-m*
by *blast*
hence *defer* (*acc* ▷ *m*) *V A p* ⊂ *defer acc V A p*
using *def-limited-acc card-ineq card-psubset less*
by *metis*
with *def-limited-acc*
show *loop-comp-helper acc m t V A p* = *loop-comp-helper* (*acc* ▷ *m*) *m t V A p*
using *loop-comp-code-helper t-not-satisfied-for-p less*
by (*metis* (*no-types*))
qed
show *?thesis*
proof (*safe*)
fix
q :: (′*a*, ′*v*) *Profile* **and**
a :: ′*a*
assume
a-in-defer-lch: *a* ∈ *defer* (*loop-comp-helper acc m t*) *V A p* **and**
a-lifted: *Profile.lifted V A p q a*
have *mod-acc*: *SCF-result.electoral-module acc*
using *less.prems*
unfolding *defer-lift-invariance-def*
by *simp*
hence *loop-comp-equiv*:
loop-comp-helper acc m t V A p = *loop-comp-helper* (*acc* ▷ *m*) *m t V A p*
using *rec-step-p*
by *blast*
hence *a* ∈ *defer* (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *V A p*
using *a-in-defer-lch*
by *presburger*
moreover have *l-inv*: *defer-lift-invariance* (*acc* ▷ *m*)
using *less.prems monotone-m only-voters-m seq-comp-presv-def-lift-inv*[*of acc m*]
by *blast*
ultimately have *a* ∈ *defer* (*acc* ▷ *m*) *V A p*
using *prof monotone-m in-mono loop-comp-helper-imp-no-def-incr*
unfolding *defer-lift-invariance-def*
by (*metis* (*no-types*, *lifting*))
with *l-inv loop-comp-equiv* **show**
loop-comp-helper acc m t V A p = *loop-comp-helper acc m t V A q*
proof −
assume

dli-acc-seq-m: *defer-lift-invariance* (*acc* ▷ *m*) **and**
a-in-def-seq: $a \in$ *defer* (*acc* ▷ *m*) *V A p*
moreover from *this* **have** $\mathcal{SCF}$*-result.electoral-module* (*acc* ▷ *m*)
unfolding *defer-lift-invariance-def*
by *blast*
moreover have $a \in$ *defer* (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *V A p*
using *loop-comp-equiv a-in-defer-lch*
by *presburger*
ultimately have
loop-comp-helper (*acc* ▷ *m*) *m t V A p*
= *loop-comp-helper* (*acc* ▷ *m*) *m t V A q*
using *monotone-m mod-acc less a-lifted card-smaller-for-p*
defer-in-alts infinite-super less
unfolding *lifted-def*
by (*metis* (*no-types*))
moreover have *loop-comp-helper acc m t V A q*
= *loop-comp-helper* (*acc* ▷ *m*) *m t V A q*
using *dli-acc-seq-m a-in-def-seq less a-lifted rec-step-q*
by *blast*
ultimately show *?thesis*
using *loop-comp-equiv*
by *presburger*
qed
qed
next
assume $\neg \neg t$ (*acc V A p*)
thus *?thesis*
using *loop-comp-code-helper less*
unfolding *defer-lift-invariance-def*
by *metis*
qed
qed
qed

lemma *loop-comp-helper-def-lift-inv*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assumes
defer-lift-invariance m **and**
only-voters-vote m **and**
defer-lift-invariance acc **and**
profile V A p **and**

lifted V A p q a **and**
a ∈ defer (*loop-comp-helper acc m t*) *V A p*
shows (*loop-comp-helper acc m t*) *V A p* = (*loop-comp-helper acc m t*) *V A q*
using *assms loop-comp-helper-def-lift-inv-helper lifted-def*
defer-in-alts defer-lift-invariance-def finite-subset
by *metis*

lemma *lifted-imp-fin-prof*:
fixes
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assumes *lifted V A p q a*
shows *finite-profile V A p*
using *assms*
unfolding *lifted-def*
by *simp*

lemma *loop-comp-helper-presv-def-lift-inv*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
defer-lift-invariance m **and**
only-voters-vote m **and**
defer-lift-invariance acc
shows *defer-lift-invariance* (*loop-comp-helper acc m t*)
proof (*unfold defer-lift-invariance-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* (*loop-comp-helper acc m t*)
using *loop-comp-helper-imp-partit assms*
unfolding $\mathcal{SCF}$*-result.electoral-module-def*
defer-lift-invariance-def
by *metis*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
q :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assume
a ∈ defer (*loop-comp-helper acc m t*) *V A p* **and**
lifted V A p q a
thus *loop-comp-helper acc m t V A p* = *loop-comp-helper acc m t V A q*
using *lifted-imp-fin-prof loop-comp-helper-def-lift-inv assms*
by *metis*

qed

lemma *loop-comp-presv-non-electing-helper*:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    t :: *'a Termination-Condition* **and**
    acc :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    n :: *nat*
  assumes
    non-electing-m: *non-electing m* **and**
    non-electing-acc: *non-electing acc* **and**
    prof: *profile V A p* **and**
    acc-defer-card: *n* = *card* (*defer acc V A p*)
  shows *elect* (*loop-comp-helper acc m t*) *V A p* = {}
  using *acc-defer-card non-electing-acc*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
  case (*less n*)
  thus *?case*
  proof (*safe*)
    fix *x* :: *'a*
    assume
      acc-no-elect:
      ($\bigwedge$ *i acc'*. *i* < *card* (*defer acc V A p*) $\Longrightarrow$
        i = *card* (*defer acc' V A p*) $\Longrightarrow$ *non-electing acc'* $\Longrightarrow$
          elect (*loop-comp-helper acc' m t*) *V A p* = {}) **and**
      acc-non-elect: *non-electing acc* **and**
      x-in-acc-elect: *x* ∈ *elect* (*loop-comp-helper acc m t*) *V A p*
    have ∀ *m' n'*. *non-electing m'* ∧ *non-electing n'* ⟶ *non-electing* (*m'* ⊳ *n'*)
      by *simp*
    hence *seq-acc-m-non-elect*: *non-electing* (*acc* ⊳ *m*)
      using *acc-non-elect non-electing-m*
      by *blast*
    have ∀ *i m'*.
          i < *card* (*defer acc V A p*) ∧ *i* = *card* (*defer m' V A p*) ∧
              non-electing m' ⟶
            elect (*loop-comp-helper m' m t*) *V A p* = {}
      using *acc-no-elect*
      by *blast*
    hence ∀ *m'*.
          finite (*defer acc V A p*) ∧ *defer m' V A p* ⊂ *defer acc V A p* ∧
              non-electing m' ⟶
            elect (*loop-comp-helper m' m t*) *V A p* = {}
      using *psubset-card-mono*
      by *metis*
    hence ¬ *t* (*acc V A p*) ∧ *defer* (*acc* ⊳ *m*) *V A p* ⊂ *defer acc V A p* ∧
              finite (*defer acc V A p*) ⟶

elect (*loop-comp-helper acc m t*) *V A p* = {}
using *loop-comp-code-helper seq-acc-m-non-elect*
by (*metis* (*no-types*))
moreover have *elect acc V A p* = {}
using *acc-non-elect prof non-electing-def*
by *blast*
ultimately show $x \in \{\}$
using *loop-comp-code-helper x-in-acc-elect*
by (*metis* (*no-types*))
qed
qed

lemma *loop-comp-helper-iter-elim-def-n-helper*:
fixes
m :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **and**
t :: ′*a Termination-Condition* **and**
acc :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **and**
A :: ′*a set* **and**
V :: ′*v set* **and**
p :: (′*a*, ′*v*) *Profile* **and**
n :: *nat* **and**
x :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: ∀ *r*. *t r* = (*card* (*defer-r r*) = *x*) **and**
x-greater-zero: $x > 0$ **and**
prof: *profile V A p* **and**
n-acc-defer-card: *n* = *card* (*defer acc V A p*) **and**
n-ge-x: $n \geq x$ **and**
def-card-gt-one: *card* (*defer acc V A p*) > *1* **and**
acc-nonelect: *non-electing acc*
shows *card* (*defer* (*loop-comp-helper acc m t*) *V A p*) = *x*
using *n-ge-x def-card-gt-one acc-nonelect n-acc-defer-card*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
case (*less n*)
have *mod-acc*: $\mathcal{SCF}$*-result.electoral-module acc*
using *less*
unfolding *non-electing-def*
by *metis*
hence *step-reduces-defer-set*: *defer* (*acc* ▷ *m*) *V A p* ⊂ *defer acc V A p*
using *seq-comp-elim-one-red-def-set single-elimination prof less*
by *metis*
thus *?case*
proof (*cases t* (*acc V A p*))
case *True*
assume *term-satisfied*: *t* (*acc V A p*)
thus *card* (*defer-r* (*loop-comp-helper acc m t V A p*)) = *x*

```
      using loop-comp-code-helper term-satisfied terminate-if-n-left
      by metis
  next
    case False
    hence card-not-eq-x: card (defer acc V A p) ≠ x
      using terminate-if-n-left
      by metis
    have fin-def-acc: finite (defer acc V A p)
      using prof mod-acc less card.infinite not-one-less-zero
      by metis
    hence rec-step:
      loop-comp-helper acc m t V A p = loop-comp-helper (acc ▷ m) m t V A p
      using False step-reduces-defer-set
      by simp
    have card-too-big: card (defer acc V A p) > x
      using card-not-eq-x dual-order.order-iff-strict less
      by simp
    hence enough-leftover: card (defer acc V A p) > 1
      using x-greater-zero
      by simp
    obtain k where
      new-card-k: k = card (defer (acc ▷ m) V A p)
      by metis
    have defer acc V A p ⊆ A
      using defer-in-alts prof mod-acc
      by metis
    hence step-profile: profile V (defer acc V A p) (limit-profile (defer acc V A p)
p)
      using prof limit-profile-sound
      by metis
    hence
      card (defer m V (defer acc V A p) (limit-profile (defer acc V A p) p)) =
        card (defer acc V A p) − 1
      using enough-leftover non-electing-m
            single-elimination single-elim-decr-def-card-2
      by blast
    hence k-card: k = card (defer acc V A p) − 1
      using mod-acc prof new-card-k non-electing-m seq-comp-defers-def-set
      by metis
    hence new-card-still-big-enough: x ≤ k
      using card-too-big
      by linarith
    show ?thesis
    proof (cases x < k)
      case True
      hence 1 < card (defer (acc ▷ m) V A p)
        using new-card-k x-greater-zero
        by linarith
      moreover have k < n
```

using *step-reduces-defer-set step-profile psubset-card-mono new-card-k less fin-def-acc*
by *metis*
moreover have $\mathcal{SCF}$*-result.electoral-module* (*acc* ▷ *m*)
using *mod-acc eliminates-def seq-comp-sound single-elimination*
by *metis*
moreover have *non-electing* (*acc* ▷ *m*)
using *less non-electing-m*
by *simp*
ultimately have *card* (*defer* (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *V A p*) = *x*
using *new-card-k new-card-still-big-enough less*
by *metis*
thus *?thesis*
using *rec-step*
by *presburger*
next
case *False*
thus *?thesis*
using *dual-order.strict-iff-order new-card-k new-card-still-big-enough rec-step terminate-if-n-left*
by *simp*
qed
qed
qed

lemma *loop-comp-helper-iter-elim-def-n*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
x :: *nat*
assumes
non-electing m **and**
eliminates 1 m **and**
∀ *r*. (*t r*) = (*card* (*defer-r r*) = *x*) **and**
x > *0* **and**
profile V A p **and**
card (*defer acc V A p*) ≥ *x* **and**
non-electing acc
shows *card* (*defer* (*loop-comp-helper acc m t*) *V A p*) = *x*
using *assms gr-implies-not0 le-neq-implies-less less-one linorder-neqE-nat nat-neq-iff less-le loop-comp-helper-iter-elim-def-n-helper loop-comp-code-helper*
by (*metis* (*no-types*, *lifting*))

lemma *iter-elim-def-n-helper*:

```
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    x :: nat
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left: ∀ r. (t r) = (card (defer-r r) = x) and
    x-greater-zero: x > 0 and
    prof: profile V A p and
    enough-alternatives: card A ≥ x
  shows card (defer (m ↺t) V A p) = x
proof (cases)
  assume card A = x
  thus ?thesis
    using terminate-if-n-left
    by simp
next
  assume card-not-x: ¬ card A = x
  thus ?thesis
  proof (cases)
    assume card A < x
    thus ?thesis
      using enough-alternatives not-le
      by blast
  next
    assume ¬ card A < x
    hence card A > x
      using card-not-x
      by linarith
    moreover from this
    have card (defer m V A p) = card A − 1
      using non-electing-m single-elimination single-elim-decr-def-card-2
          prof x-greater-zero
      by fastforce
    ultimately have card (defer m V A p) ≥ x
      by linarith
    moreover have (m ↺t) V A p = (loop-comp-helper m m t) V A p
      using card-not-x terminate-if-n-left
      by simp
    ultimately show ?thesis
      using non-electing-m prof single-elimination terminate-if-n-left x-greater-zero
          loop-comp-helper-iter-elim-def-n
      by metis
  qed
qed
```

6.5.4 Composition Rules

The loop composition preserves defer-lift-invariance.

theorem *loop-comp-presv-def-lift-inv*[*simp*]:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    t :: $'a$ *Termination-Condition*
  assumes *defer-lift-invariance* m **and** *only-voters-vote* m
  shows *defer-lift-invariance* $(m \circlearrowleft_t)$
proof (*unfold defer-lift-invariance-def*, *safe*)
  have $\mathcal{SCF}$*-result.electoral-module* m
    using *assms*
    unfolding *defer-lift-invariance-def*
    by *simp*
  thus $\mathcal{SCF}$*-result.electoral-module* $(m \circlearrowleft_t)$
    by (*simp add*: *loop-comp-sound*)
next
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    q :: ($'a$, $'v$) *Profile* **and**
    a :: $'a$
  assume
    $a \in$ *defer* $(m \circlearrowleft_t)$ V A p **and**
    lifted V A p q a
  moreover have
    $\forall\ p'\ q'\ a'.\ a' \in (\textit{defer}\ (m \circlearrowleft_t)\ V\ A\ p') \land \textit{lifted}\ V\ A\ p'\ q'\ a' \longrightarrow$
      $(m \circlearrowleft_t)\ V\ A\ p' = (m \circlearrowleft_t)\ V\ A\ q'$
    using *assms lifted-imp-fin-prof loop-comp-helper-def-lift-inv*
        loop-composition.simps defer-module.simps
    by (*metis* (*full-types*))
  ultimately show $(m \circlearrowleft_t)\ V\ A\ p = (m \circlearrowleft_t)\ V\ A\ q$
    by *metis*
qed

The loop composition preserves the property non-electing.

theorem *loop-comp-presv-non-electing*[*simp*]:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    t :: $'a$ *Termination-Condition*
  assumes *non-electing* m
  shows *non-electing* $(m \circlearrowleft_t)$
proof (*unfold non-electing-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* $(m \circlearrowleft_t)$
    using *loop-comp-sound assms*
    unfolding *non-electing-def*
    by *metis*
next

```
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    a :: 'a
  assume
    profile V A p and
    a ∈ elect (m ⟲t) V A p
  thus a ∈ {}
    using def-mod-non-electing loop-comp-presv-non-electing-helper
          assms empty-iff loop-comp-code
    unfolding non-electing-def
    by (metis (no-types))
qed

theorem iter-elim-def-n[simp]:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    n :: nat
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left: ∀ r. t r = (card (defer-r r) = n) and
    x-greater-zero: n > 0
  shows defers n (m ⟲t)
proof (unfold defers-def, safe)
  show SCF-result.electoral-module (m ⟲t)
    using loop-comp-sound non-electing-m
    unfolding non-electing-def
    by metis
next
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assume
    n ≤ card A and
    finite A and
    profile V A p
  thus card (defer (m ⟲t) V A p) = n
    using iter-elim-def-n-helper assms
    by metis
qed

end
```

6.6 Maximum Parallel Composition

theory *Maximum-Parallel-Composition*
imports *Basic-Modules/Component-Types/Maximum-Aggregator*
Parallel-Composition
begin

This is a family of parallel compositions. It composes a new electoral module from two electoral modules combined with the maximum aggregator. Therein, the two modules each make a decision and then a partition is returned where every alternative receives the maximum result of the two input partitions. This means that, if any alternative is elected by at least one of the modules, then it gets elected, if any non-elected alternative is deferred by at least one of the modules, then it gets deferred, only alternatives rejected by both modules get rejected.

6.6.1 Definition

fun *maximum-parallel-composition* :: ('a, 'v, 'a Result) Electoral-Module ⇒
('a, 'v, 'a Result) Electoral-Module ⇒ ('a, 'v, 'a Result) Electoral-Module
where
maximum-parallel-composition m n =
(*let a* = *max-aggregator in* ($m \parallel_a n$))

abbreviation *max-parallel* :: ('a, 'v, 'a Result) Electoral-Module
⇒ ('a, 'v, 'a Result) Electoral-Module ⇒('a, 'v, 'a Result) Electoral-Module
(**infix** $\parallel_\uparrow$ *50*) **where**
$m \parallel_\uparrow n$ == *maximum-parallel-composition m n*

6.6.2 Soundness

theorem *max-par-comp-sound*:
fixes
m :: ('a, 'v, 'a Result) Electoral-Module **and**
n :: ('a, 'v, 'a Result) Electoral-Module
assumes
$\mathcal{SCF}$-*result.electoral-module m* **and**
$\mathcal{SCF}$-*result.electoral-module n*
shows $\mathcal{SCF}$-*result.electoral-module* ($m \parallel_\uparrow n$)
using *assms*
by *simp*

lemma *max-par-comp-only-voters*:
fixes
m :: ('a, 'v, 'a Result) Electoral-Module **and**
n :: ('a, 'v, 'a Result) Electoral-Module

assumes
  only-voters-vote m **and**
  only-voters-vote n
shows *only-voters-vote* $(m \parallel_\uparrow n)$
using *max-aggregator.simps assms*
unfolding *Let-def maximum-parallel-composition.simps*
  parallel-composition.simps
  only-voters-vote-def
by *presburger*

6.6.3 Lemmas

lemma *max-agg-eq-result*:
fixes
  m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile* **and**
  a :: *'a*
assumes
  module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
  module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
  prof-p: *profile* $V\ A\ p$ **and**
  a-in-A: $a \in A$
shows *mod-contains-result* $(m \parallel_\uparrow n)\ m\ V\ A\ p\ a\ \lor$
  mod-contains-result $(m \parallel_\uparrow n)\ n\ V\ A\ p\ a$
proof (*cases*)
assume *a-elect*: $a \in$ *elect* $(m \parallel_\uparrow n)\ V\ A\ p$
hence *let* $(e,\ r,\ d) = m\ V\ A\ p;$
  $(e',\ r',\ d') = n\ V\ A\ p$ *in*
  $a \in e \cup e'$
  by *auto*
hence $a \in$ (*elect* $m\ V\ A\ p$) $\cup$ (*elect* $n\ V\ A\ p$)
  by *auto*
moreover have
  $\forall\ m'\ n'\ V'\ A'\ p'\ a'.$
  mod-contains-result $m'\ n'\ V'\ A'\ p'\ (a'::'a) =$
  ($\mathcal{SCF}$*-result.electoral-module* m'
  $\land$ $\mathcal{SCF}$*-result.electoral-module* n'
  $\land$ *profile* $V'\ A'\ p' \land a' \in A'$
  $\land$ ($a' \notin$ *elect* $m'\ V'\ A'\ p' \lor a' \in$ *elect* $n'\ V'\ A'\ p'$)
  $\land$ ($a' \notin$ *reject* $m'\ V'\ A'\ p' \lor a' \in$ *reject* $n'\ V'\ A'\ p'$)
  $\land$ ($a' \notin$ *defer* $m'\ V'\ A'\ p' \lor a' \in$ *defer* $n'\ V'\ A'\ p'$))
  unfolding *mod-contains-result-def*
  by *simp*
moreover have *module-mn*: $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_\uparrow n)$
  using *module-m module-n*
  by *simp*

moreover have $a \notin$ *defer* $(m \parallel_\uparrow n)$ V A p
  using *module-mn IntI a-elect empty-iff prof-p result-disj*
  by (*metis* (*no-types*))
moreover have $a \notin$ *reject* $(m \parallel_\uparrow n)$ V A p
  using *module-mn IntI a-elect empty-iff prof-p result-disj*
  by (*metis* (*no-types*))
ultimately show *?thesis*
  using *assms*
  by *blast*
next
assume *not-a-elect*: $a \notin$ *elect* $(m \parallel_\uparrow n)$ V A p
thus *?thesis*
proof (*cases*)
  assume *a-in-def*: $a \in$ *defer* $(m \parallel_\uparrow n)$ V A p
  thus *?thesis*
  proof (*safe*)
    assume *not-mod-cont-mn*: $\neg$ *mod-contains-result* $(m \parallel_\uparrow n)$ n V A p a
    have *par-emod*: $\forall$ m' n'.
      $\mathcal{SCF}$*-result.electoral-module* $m' \wedge$
      $\mathcal{SCF}$*-result.electoral-module* $n' \longrightarrow$
      $\mathcal{SCF}$*-result.electoral-module* $(m' \parallel_\uparrow n')$
      using *max-par-comp-sound*
      by *blast*
    have *set-intersect*: $\forall$ a' A' A''. $(a' \in A' \cap A'') = (a' \in A' \wedge a' \in A'')$
      by *blast*
    have *wf-n*: *well-formed-*$\mathcal{SCF}$ A $(n$ V A $p)$
      using *prof-p module-n*
      unfolding $\mathcal{SCF}$*-result.electoral-module-def*
      by *blast*
    have *wf-m*: *well-formed-*$\mathcal{SCF}$ A $(m$ V A $p)$
      using *prof-p module-m*
      unfolding $\mathcal{SCF}$*-result.electoral-module-def*
      by *blast*
    have *e-mod-par*: $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_\uparrow n)$
      using *par-emod module-m module-n*
      by *blast*
    hence $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_m$*ax-aggregator* $n)$
      by *simp*
    hence *result-disj-max*:
      elect $(m \parallel_m$*ax-aggregator* $n)$ V A $p \cap$
        reject $(m \parallel_m$*ax-aggregator* $n)$ V A $p = \{\} \wedge$
      elect $(m \parallel_m$*ax-aggregator* $n)$ V A $p \cap$
        defer $(m \parallel_m$*ax-aggregator* $n)$ V A $p = \{\} \wedge$
      reject $(m \parallel_m$*ax-aggregator* $n)$ V A $p \cap$
        defer $(m \parallel_m$*ax-aggregator* $n)$ V A $p = \{\}$
      using *prof-p result-disj*
      by *metis*
    have *a-not-elect*: $a \notin$ *elect* $(m \parallel_m$*ax-aggregator* $n)$ V A p
      using *result-disj-max a-in-def*

by *force*
have *result-m*: (*elect m V A p*, *reject m V A p*, *defer m V A p*) = *m V A p*
by *auto*
have *result-n*: (*elect n V A p*, *reject n V A p*, *defer n V A p*) = *n V A p*
by *auto*
have *max-pq*:
$\forall$ (A'::$'a$ *set*) m' n'.
elect-r (*max-aggregator* A' m' n') = *elect-r* m' $\cup$ *elect-r* n'
by *force*
have $a \notin$ *elect* (m $\|_m$*ax-aggregator* n) *V A p*
using *a-not-elect*
by *blast*
hence $a \notin$ *elect m V A p* $\cup$ *elect n V A p*
using *max-pq*
by *simp*
hence *b-not-elect-mn*: $a \notin$ *elect m V A p* $\wedge$ $a \notin$ *elect n V A p*
by *blast*
have *b-not-mpar-rej*: $a \notin$ *reject* (m $\|_m$*ax-aggregator* n) *V A p*
using *result-disj-max a-in-def*
by *fastforce*
have *mod-cont-res-fg*:
$\forall$ m' n' A' V' p' (a'::$'a$).
mod-contains-result m' n' V' A' p' a' =
($\mathcal{SCF}$-*result.electoral-module* m'
$\wedge$ $\mathcal{SCF}$-*result.electoral-module* n'
$\wedge$ *profile* V' A' p' $\wedge$ $a' \in A'$
$\wedge$ ($a' \in$ *elect* m' V' A' p' $\longrightarrow$ $a' \in$ *elect* n' V' A' p')
$\wedge$ ($a' \in$ *reject* m' V' A' p' $\longrightarrow$ $a' \in$ *reject* n' V' A' p')
$\wedge$ ($a' \in$ *defer* m' V' A' p' $\longrightarrow$ $a' \in$ *defer* n' V' A' p'))
unfolding *mod-contains-result-def*
by *simp*
have *max-agg-res*:
max-aggregator A (*elect m V A p*, *reject m V A p*, *defer m V A p*)
(*elect n V A p*, *reject n V A p*, *defer n V A p*) = (m $\|_m$*ax-aggregator* n)
V A p
by *simp*
have *well-f-max*:
$\forall$ r' r'' e' e'' d' d'' A'.
well-formed-$\mathcal{SCF}$ A' (e', r', d') $\wedge$
well-formed-$\mathcal{SCF}$ A' (e'', r'', d'') $\longrightarrow$
reject-r (*max-aggregator* A' (e', r', d') (e'', r'', d'')) = $r' \cap r''$
using *max-agg-rej-set*
by *metis*
have *e-mod-disj*:
$\forall$ m' (V'::$'v$ *set*) (A'::$'a$ *set*) p'.
$\mathcal{SCF}$-*result.electoral-module* m' $\wedge$ *profile* V' A' p'
$\longrightarrow$ *elect* m' V' A' p' $\cup$ *reject* m' V' A' p' $\cup$ *defer* m' V' A' p' = A'
using *result-presv-alts*
by *blast*

hence *e-mod-disj-n*: $elect\ n\ V\ A\ p \cup reject\ n\ V\ A\ p \cup defer\ n\ V\ A\ p = A$
  using *prof-p module-n*
  by *metis*
have $\forall\ m'\ n'\ A'\ V'\ p'\ (b::'a)$.
  $mod\text{-}contains\text{-}result\ m'\ n'\ V'\ A'\ p'\ b =$
  $(\mathcal{SCF}\text{-}result.electoral\text{-}module\ m'$
  $\wedge \mathcal{SCF}\text{-}result.electoral\text{-}module\ n'$
  $\wedge profile\ V'\ A'\ p' \wedge b \in A'$
  $\wedge (b \in elect\ m'\ V'\ A'\ p' \longrightarrow b \in elect\ n'\ V'\ A'\ p')$
  $\wedge (b \in reject\ m'\ V'\ A'\ p' \longrightarrow b \in reject\ n'\ V'\ A'\ p')$
  $\wedge (b \in defer\ m'\ V'\ A'\ p' \longrightarrow b \in defer\ n'\ V'\ A'\ p'))$
  unfolding *mod-contains-result-def*
  by *simp*
hence $a \in reject\ n\ V\ A\ p$
  using *e-mod-disj-n e-mod-par prof-p a-in-A module-n not-mod-cont-mn a-not-elect b-not-elect-mn b-not-mpar-rej*
  by *fastforce*
hence $a \notin reject\ m\ V\ A\ p$
  using *well-f-max max-agg-res result-m result-n set-intersect wf-m wf-n b-not-mpar-rej*
  by (*metis* (*no-types*))
hence $a \notin defer\ (m \parallel_\uparrow n)\ V\ A\ p \vee a \in defer\ m\ V\ A\ p$
  using *e-mod-disj prof-p a-in-A module-m b-not-elect-mn*
  by *blast*
thus $mod\text{-}contains\text{-}result\ (m \parallel_\uparrow n)\ m\ V\ A\ p\ a$
  using *b-not-mpar-rej mod-cont-res-fg e-mod-par prof-p a-in-A module-m a-not-elect*
  by *fastforce*
qed
next
assume *not-a-defer*: $a \notin defer\ (m \parallel_\uparrow n)\ V\ A\ p$
have *el-rej-defer*: $(elect\ m\ V\ A\ p,\ reject\ m\ V\ A\ p,\ defer\ m\ V\ A\ p) = m\ V\ A\ p$
  by *auto*
from *not-a-elect not-a-defer*
have *a-reject*: $a \in reject\ (m \parallel_\uparrow n)\ V\ A\ p$
  using *electoral-mod-defer-elem a-in-A module-m module-n prof-p max-par-comp-sound*
  by *metis*
hence $case\ snd\ (m\ V\ A\ p)\ of\ (r,\ d) \Rightarrow$
  $case\ n\ V\ A\ p\ of\ (e',\ r',\ d') \Rightarrow$
  $a \in reject\text{-}r\ (max\text{-}aggregator\ A\ (elect\ m\ V\ A\ p,\ r,\ d)\ (e',\ r',\ d'))$
  using *el-rej-defer*
  by *force*
hence $let\ (e,\ r,\ d) = m\ V\ A\ p;$
  $(e',\ r',\ d') = n\ V\ A\ p\ in$
  $a \in reject\text{-}r\ (max\text{-}aggregator\ A\ (e,\ r,\ d)\ (e',\ r',\ d'))$
  unfolding *case-prod-unfold*
  by *simp*
hence $let\ (e,\ r,\ d) = m\ V\ A\ p;$

$(e', r', d') = n\ V\ A\ p\ in$
$a \in A - (e \cup e' \cup d \cup d')$
by *simp*
hence $a \notin$ *elect* $m\ V\ A\ p \cup$ (*defer* $n\ V\ A\ p \cup$ *defer* $m\ V\ A\ p$)
by *force*
thus *?thesis*
using *mod-contains-result-comm mod-contains-result-def Un-iff*
a-reject prof-p a-in-A module-m module-n max-par-comp-sound
by (*metis* (*no-types*))
qed
qed

lemma *max-agg-rej-iff-both-reject*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a,'v) *Profile* **and**
a :: 'a
assumes
finite-profile $V\ A\ p$ **and**
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n
shows ($a \in$ *reject* ($m \parallel_\uparrow n$) $V\ A\ p$)
= ($a \in$ *reject* $m\ V\ A\ p \wedge a \in$ *reject* $n\ V\ A\ p$)
proof
assume *rej-a*: $a \in$ *reject* ($m \parallel_\uparrow n$) $V\ A\ p$
hence *case* $n\ V\ A\ p$ *of* $(e, r, d) \Rightarrow$
$a \in$ *reject-r* (*max-aggregator* A
(*elect* $m\ V\ A\ p$, *reject* $m\ V\ A\ p$, *defer* $m\ V\ A\ p$) (e, r, d))
by *auto*
hence *case snd* ($m\ V\ A\ p$) *of* $(r, d) \Rightarrow$
case $n\ V\ A\ p$ *of* $(e', r', d') \Rightarrow$
$a \in$ *reject-r* (*max-aggregator* A (*elect* $m\ V\ A\ p$, r, d) (e', r', d'))
by *force*
with *rej-a*
have *let* $(e, r, d) = m\ V\ A\ p$;
$(e', r', d') = n\ V\ A\ p\ in$
$a \in$ *reject-r* (*max-aggregator* A (e, r, d) (e', r', d'))
unfolding *prod.case-eq-if*
by *simp*
hence *let* $(e, r, d) = m\ V\ A\ p$;
$(e', r', d') = n\ V\ A\ p\ in$
$a \in A - (e \cup e' \cup d \cup d')$
by *simp*
hence
$a \in A -$ (*elect* $m\ V\ A\ p \cup$ *elect* $n\ V\ A\ p \cup$ *defer* $m\ V\ A\ p \cup$ *defer* $n\ V\ A\ p$)
by *auto*

thus $a \in$ *reject* m V A p $\wedge$ $a \in$ *reject* n V A p
  using *Diff-iff Un-iff electoral-mod-defer-elem assms*
  by *metis*
next
  assume $a \in$ *reject* m V A p $\wedge$ $a \in$ *reject* n V A p
  moreover from *this*
  have $a \notin$ *elect* m V A p $\wedge$ $a \notin$ *defer* m V A p
    $\wedge$ $a \notin$ *elect* n V A p $\wedge$ $a \notin$ *defer* n V A p
  using *IntI empty-iff assms result-disj*
  by *metis*
  ultimately show $a \in$ *reject* $(m \parallel_\uparrow n)$ V A p
  using *DiffD1 max-agg-eq-result mod-contains-result-comm mod-contains-result-def reject-not-elec-or-def assms*
  by (*metis* (*no-types*))
qed

lemma *max-agg-rej-fst-imp-seq-contained*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile* **and**
    a :: $'a$
  assumes
    f-prof: *finite-profile* V A p **and**
    module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
    module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
    rejected: $a \in$ *reject* n V A p
  shows *mod-contains-result* m $(m \parallel_\uparrow n)$ V A p a
  using *assms*
proof (*unfold mod-contains-result-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_\uparrow n)$
  using *module-m module-n*
  by *simp*
next
  show $a \in A$
  using *f-prof module-n rejected reject-in-alts*
  by *blast*
next
  assume *a-in-elect*: $a \in$ *elect* m V A p
  hence *a-not-reject*: $a \notin$ *reject* m V A p
  using *disjoint-iff-not-equal f-prof module-m result-disj*
  by *metis*
  have *reject* n V A $p \subseteq A$
  using *f-prof module-n*
  by (*simp add*: *reject-in-alts*)
  hence $a \in A$
  using *in-mono rejected*

by *metis*
with *a-in-elect a-not-reject*
show $a \in$ *elect* $(m \parallel_\uparrow n)$ V A p
using *f-prof max-agg-eq-result module-m module-n rejected max-agg-rej-iff-both-reject mod-contains-result-comm mod-contains-result-def*
by *metis*
next
assume $a \in$ *reject* m V A p
hence $a \in$ *reject* m V A p $\wedge$ $a \in$ *reject* n V A p
using *rejected*
by *simp*
thus $a \in$ *reject* $(m \parallel_\uparrow n)$ V A p
using *f-prof max-agg-rej-iff-both-reject module-m module-n*
by (*metis* (*no-types*))
next
assume *a-in-defer*: $a \in$ *defer* m V A p
then obtain d :: $'a$ **where**
defer-a: $a = d \wedge d \in$ *defer* m V A p
by *metis*
have *a-not-rej*: $a \notin$ *reject* m V A p
using *disjoint-iff-not-equal f-prof defer-a module-m result-disj*
by (*metis* (*no-types*))
have
$\forall$ m' A' V' p'.
$\mathcal{SCF}$*-result.electoral-module* m' $\wedge$ *finite* A' $\wedge$ *finite* V' $\wedge$ *profile* V' A' p'
$\longrightarrow$ *elect* m' V' A' p' $\cup$ *reject* m' V' A' p' $\cup$ *defer* m' V' A' p' $= A'$
using *result-presv-alts*
by *metis*
hence $a \in A$
using *a-in-defer f-prof module-m*
by *blast*
with *defer-a a-not-rej*
show $a \in$ *defer* $(m \parallel_\uparrow n)$ V A p
using *f-prof max-agg-eq-result max-agg-rej-iff-both-reject mod-contains-result-comm mod-contains-result-def module-m module-n rejected*
by *metis*
qed

lemma *max-agg-rej-fst-equiv-seq-contained*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes

finite-profile V A p **and**
SCF-result.electoral-module m **and**
SCF-result.electoral-module n **and**
$a \in$ *reject* n V A p
shows *mod-contains-result-sym* $(m \parallel_\uparrow n)$ m V A p a
using *assms*
proof (*unfold mod-contains-result-sym-def*, *safe*)
assume $a \in$ *reject* $(m \parallel_\uparrow n)$ V A p
thus $a \in$ *reject* m V A p
using *assms max-agg-rej-iff-both-reject*
by (*metis* (*no-types*))
next
have *mod-contains-result* m $(m \parallel_\uparrow n)$ V A p a
using *assms max-agg-rej-fst-imp-seq-contained*
by (*metis* (*full-types*))
thus
$a \in$ *elect* $(m \parallel_\uparrow n)$ V A $p \Longrightarrow a \in$ *elect* m V A p **and**
$a \in$ *defer* $(m \parallel_\uparrow n)$ V A $p \Longrightarrow a \in$ *defer* m V A p
using *mod-contains-result-comm*
unfolding *mod-contains-result-def*
by (*metis* (*full-types*), *metis* (*full-types*))
next
show
SCF-result.electoral-module $(m \parallel_\uparrow n)$ **and**
$a \in A$
using *assms max-agg-rej-fst-imp-seq-contained*
unfolding *mod-contains-result-def*
by (*metis* (*full-types*), *metis* (*full-types*))
next
show
$a \in$ *elect* m V A $p \Longrightarrow a \in$ *elect* $(m \parallel_\uparrow n)$ V A p **and**
$a \in$ *reject* m V A $p \Longrightarrow a \in$ *reject* $(m \parallel_\uparrow n)$ V A p **and**
$a \in$ *defer* m V A $p \Longrightarrow a \in$ *defer* $(m \parallel_\uparrow n)$ V A p
using *assms max-agg-rej-fst-imp-seq-contained*
unfolding *mod-contains-result-def*
by (*metis* (*no-types*), *metis* (*no-types*), *metis* (*no-types*))
qed

lemma *max-agg-rej-snd-imp-seq-contained*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
a :: '*a*
assumes
f-prof: *finite-profile* V A p **and**
module-m: *SCF-result.electoral-module* m **and**

module-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
rejected: $a \in$ *reject m V A p*
shows *mod-contains-result n* ($m \parallel_\uparrow n$) *V A p a*
using *assms*
proof (*unfold mod-contains-result-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* ($m \parallel_\uparrow n$)
using *module-m module-n*
by *simp*
next
show $a \in A$
using *f-prof in-mono module-m reject-in-alts rejected*
by (*metis* (*no-types*))
next
assume $a \in$ *elect n V A p*
thus $a \in$ *elect* ($m \parallel_\uparrow n$) *V A p*
using *max-aggregator.simps*[*of*
A elect m V A p reject m V A p defer m V A p
elect n V A p reject n V A p defer n V A p]
by *simp*
next
assume $a \in$ *reject n V A p*
thus $a \in$ *reject* ($m \parallel_\uparrow n$) *V A p*
using *f-prof max-agg-rej-iff-both-reject module-m module-n rejected*
by *metis*
next
assume $a \in$ *defer n V A p*
moreover have $a \in A$
using *f-prof max-agg-rej-fst-imp-seq-contained module-m rejected*
unfolding *mod-contains-result-def*
by *metis*
ultimately show $a \in$ *defer* ($m \parallel_\uparrow n$) *V A p*
using *disjoint-iff-not-equal max-agg-eq-result max-agg-rej-iff-both-reject*
f-prof mod-contains-result-comm mod-contains-result-def
module-m module-n rejected result-disj
by *metis*
qed

lemma *max-agg-rej-snd-equiv-seq-contained*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assumes
finite-profile V A p **and**
$\mathcal{SCF}$*-result.electoral-module m* **and**
$\mathcal{SCF}$*-result.electoral-module n* **and**

$a \in reject\ m\ V\ A\ p$
shows *mod-contains-result-sym* $(m \parallel_\uparrow n)\ n\ V\ A\ p\ a$
using *assms*
proof (*unfold mod-contains-result-sym-def*, *safe*)
assume $a \in reject\ (m \parallel_\uparrow n)\ V\ A\ p$
thus $a \in reject\ n\ V\ A\ p$
using *assms max-agg-rej-iff-both-reject*
by (*metis* (*no-types*))
next
have *mod-contains-result* $n\ (m \parallel_\uparrow n)\ V\ A\ p\ a$
using *assms max-agg-rej-snd-imp-seq-contained*
by (*metis* (*full-types*))
thus
$a \in elect\ (m \parallel_\uparrow n)\ V\ A\ p \Longrightarrow a \in elect\ n\ V\ A\ p$ **and**
$a \in defer\ (m \parallel_\uparrow n)\ V\ A\ p \Longrightarrow a \in defer\ n\ V\ A\ p$
using *mod-contains-result-comm*
unfolding *mod-contains-result-def*
by (*metis* (*full-types*), *metis* (*full-types*))
next
show
$\mathcal{SCF}$*-result.electoral-module* $(m \parallel_\uparrow n)$ **and**
$a \in A$
using *assms max-agg-rej-snd-imp-seq-contained*
unfolding *mod-contains-result-def*
by (*metis* (*full-types*), *metis* (*full-types*))
next
show
$a \in elect\ n\ V\ A\ p \Longrightarrow a \in elect\ (m \parallel_\uparrow n)\ V\ A\ p$ **and**
$a \in reject\ n\ V\ A\ p \Longrightarrow a \in reject\ (m \parallel_\uparrow n)\ V\ A\ p$ **and**
$a \in defer\ n\ V\ A\ p \Longrightarrow a \in defer\ (m \parallel_\uparrow n)\ V\ A\ p$
using *assms max-agg-rej-snd-imp-seq-contained*
unfolding *mod-contains-result-def*
by (*metis* (*no-types*), *metis* (*no-types*), *metis* (*no-types*))
qed

lemma *max-agg-rej-intersect*:
fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a, 'v)$ *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n **and**
profile $V\ A\ p$ **and**
finite A
shows $reject\ (m \parallel_\uparrow n)\ V\ A\ p = (reject\ m\ V\ A\ p) \cap (reject\ n\ V\ A\ p)$
proof −

have $A = (elect\ m\ V\ A\ p) \cup (reject\ m\ V\ A\ p) \cup (defer\ m\ V\ A\ p)$
$\wedge A = (elect\ n\ V\ A\ p) \cup (reject\ n\ V\ A\ p) \cup (defer\ n\ V\ A\ p)$
using *assms result-presv-alts*
by *metis*
hence $A - ((elect\ m\ V\ A\ p) \cup (defer\ m\ V\ A\ p)) = (reject\ m\ V\ A\ p)$
$\wedge A - ((elect\ n\ V\ A\ p) \cup (defer\ n\ V\ A\ p)) = (reject\ n\ V\ A\ p)$
using *assms reject-not-elec-or-def*
by *fastforce*
hence
$A - ((elect\ m\ V\ A\ p) \cup (elect\ n\ V\ A\ p)$
$\cup (defer\ m\ V\ A\ p) \cup (defer\ n\ V\ A\ p))$
$= (reject\ m\ V\ A\ p) \cap (reject\ n\ V\ A\ p)$
by *blast*
hence *let* $(e, r, d) = m\ V\ A\ p;$
$(e', r', d') = n\ V\ A\ p$ *in*
$A - (e \cup e' \cup d \cup d') = r \cap r'$
by *fastforce*
thus *?thesis*
by *auto*
qed

lemma *dcompat-dec-by-one-mod*:
fixes
m :: *('a, 'v, 'a Result) Electoral-Module* **and**
n :: *('a, 'v, 'a Result) Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
a :: *'a*
assumes
disjoint-compatibility $m\ n$ **and**
$a \in A$
shows
$(\forall\ p.$ *finite-profile* $V\ A\ p \longrightarrow$ *mod-contains-result* $m\ (m \parallel_\uparrow n)\ V\ A\ p\ a)$
$\vee (\forall\ p.$ *finite-profile* $V\ A\ p \longrightarrow$ *mod-contains-result* $n\ (m \parallel_\uparrow n)\ V\ A\ p\ a)$
using *DiffI assms max-agg-rej-fst-imp-seq-contained max-agg-rej-snd-imp-seq-contained*
unfolding *disjoint-compatibility-def*
by *metis*

6.6.4 Composition Rules

Using a conservative aggregator, the parallel composition preserves the property non-electing.

theorem *conserv-max-agg-presv-non-electing*[*simp*]:
fixes
m :: *('a, 'v, 'a Result) Electoral-Module* **and**
n :: *('a, 'v, 'a Result) Electoral-Module*
assumes
non-electing m **and**
non-electing n

shows *non-electing* $(m \parallel_\uparrow n)$
using *assms*
by *simp*

Using the max aggregator, composing two compatible electoral modules in parallel preserves defer-lift-invariance.

theorem *par-comp-def-lift-inv*[*simp*]:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes
compatible: *disjoint-compatibility* m n **and**
monotone-m: *defer-lift-invariance* m **and**
monotone-n: *defer-lift-invariance* n
shows *defer-lift-invariance* $(m \parallel_\uparrow n)$
proof (*unfold defer-lift-invariance-def*, *safe*)
have *mod-m*: $\mathcal{SCF}$*-result.electoral-module* m
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have *mod-n*: $\mathcal{SCF}$*-result.electoral-module* n
using *monotone-n*
unfolding *defer-lift-invariance-def*
by *simp*
ultimately show $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_\uparrow n)$
by *simp*
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
q :: ($'a$, $'v$) *Profile* **and**
a :: $'a$
assume
defer-a: $a \in$ *defer* $(m \parallel_\uparrow n)$ V A p **and**
lifted-a: *Profile.lifted* V A p q a
hence *f-profs*: *finite-profile* V A p $\wedge$ *finite-profile* V A q
unfolding *lifted-def*
by *simp*
from *compatible*
obtain B :: $'a$ *set* **where**
alts: $B \subseteq A$
$\wedge$ ($\forall$ $b \in B$. *indep-of-alt* m V A b $\wedge$
($\forall$ p'. *finite-profile* V A p' $\longrightarrow$ $b \in$ *reject* m V A p'))
$\wedge$ ($\forall$ $b \in A - B$. *indep-of-alt* n V A b $\wedge$
($\forall$ p'. *finite-profile* V A p' $\longrightarrow$ $b \in$ *reject* n V A p'))
using *f-profs*
unfolding *disjoint-compatibility-def*
by (*metis* (*no-types*, *lifting*))
have $\forall$ $b \in A$. *prof-contains-result* $(m \parallel_\uparrow n)$ V A p q b

proof (*cases*)
  assume *a-in-B*: $a \in B$
  hence $a \in$ *reject* $m\ V\ A\ p$
    using *alts f-profs*
    by *blast*
  with *defer-a*
  have *defer-n*: $a \in$ *defer* $n\ V\ A\ p$
    using *compatible f-profs max-agg-rej-snd-equiv-seq-contained*
    unfolding *disjoint-compatibility-def mod-contains-result-sym-def*
    by *metis*
  have $\forall\ b \in B$. *mod-contains-result-sym* $(m \parallel_{\uparrow} n)\ n\ V\ A\ p\ b$
    using *alts compatible max-agg-rej-snd-equiv-seq-contained f-profs*
    unfolding *disjoint-compatibility-def*
    by *metis*
  moreover have $\forall\ b \in A$. *prof-contains-result* $n\ V\ A\ p\ q\ b$
  proof (*unfold prof-contains-result-def*, *clarify*)
    fix $b :: 'a$
    assume *b-in-A*: $b \in A$
    show $\mathcal{SCF}$*-result.electoral-module* $n \wedge$ *profile* $V\ A\ p$
        $\wedge$ *profile* $V\ A\ q \wedge b \in A \wedge$
        $(b \in$ *elect* $n\ V\ A\ p \longrightarrow b \in$ *elect* $n\ V\ A\ q) \wedge$
        $(b \in$ *reject* $n\ V\ A\ p \longrightarrow b \in$ *reject* $n\ V\ A\ q) \wedge$
        $(b \in$ *defer* $n\ V\ A\ p \longrightarrow b \in$ *defer* $n\ V\ A\ q)$
    proof (*safe*)
      show $\mathcal{SCF}$*-result.electoral-module* n
        using *monotone-n*
        unfolding *defer-lift-invariance-def*
        by *metis*
    next
      show
        profile $V\ A\ p$ **and**
        profile $V\ A\ q$ **and**
        $b \in A$
        using *f-profs b-in-A*
        by (*simp*, *simp*, *simp*)
    next
      show
        $b \in$ *elect* $n\ V\ A\ p \Longrightarrow b \in$ *elect* $n\ V\ A\ q$ **and**
        $b \in$ *reject* $n\ V\ A\ p \Longrightarrow b \in$ *reject* $n\ V\ A\ q$ **and**
        $b \in$ *defer* $n\ V\ A\ p \Longrightarrow b \in$ *defer* $n\ V\ A\ q$
        using *defer-n lifted-a monotone-n f-profs*
        unfolding *defer-lift-invariance-def*
        by (*metis*, *metis*, *metis*)
    qed
  qed
  moreover have $\forall\ b \in B$. *mod-contains-result* $n\ (m \parallel_{\uparrow} n)\ V\ A\ q\ b$
    using *alts compatible max-agg-rej-snd-imp-seq-contained f-profs*
    unfolding *disjoint-compatibility-def*
    by *metis*

ultimately have *prof-contains-result-of-comps-for-elems-in-B*:
$\forall\ b \in B.$ *prof-contains-result* $(m \parallel_\uparrow n)\ V\ A\ p\ q\ b$
unfolding *mod-contains-result-def mod-contains-result-sym-def*
prof-contains-result-def
by *simp*
have $\forall\ b \in A - B.$ *mod-contains-result-sym* $(m \parallel_\uparrow n)\ m\ V\ A\ p\ b$
using *alts max-agg-rej-fst-equiv-seq-contained monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*
moreover have $\forall\ b \in A.$ *prof-contains-result* $m\ V\ A\ p\ q\ b$
proof (*unfold prof-contains-result-def*, *clarify*)
fix $b :: 'a$
assume *b-in-A*: $b \in A$
show $\mathcal{SCF}$*-result.electoral-module* $m \land$ *profile* $V\ A\ p \land$
profile $V\ A\ q \land b \in A \land$
$(b \in$ *elect* $m\ V\ A\ p \longrightarrow b \in$ *elect* $m\ V\ A\ q) \land$
$(b \in$ *reject* $m\ V\ A\ p \longrightarrow b \in$ *reject* $m\ V\ A\ q) \land$
$(b \in$ *defer* $m\ V\ A\ p \longrightarrow b \in$ *defer* $m\ V\ A\ q)$
proof (*safe*)
show $\mathcal{SCF}$*-result.electoral-module* m
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *metis*
next
show
profile $V\ A\ p$ **and**
profile $V\ A\ q$ **and**
$b \in A$
using *f-profs b-in-A*
by (*simp*, *simp*, *simp*)
next
show
$b \in$ *elect* $m\ V\ A\ p \Longrightarrow b \in$ *elect* $m\ V\ A\ q$ **and**
$b \in$ *reject* $m\ V\ A\ p \Longrightarrow b \in$ *reject* $m\ V\ A\ q$ **and**
$b \in$ *defer* $m\ V\ A\ p \Longrightarrow b \in$ *defer* $m\ V\ A\ q$
using *alts a-in-B lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by (*metis*, *metis*, *metis*)
qed
qed
moreover have $\forall\ b \in A - B.$ *mod-contains-result* $m\ (m \parallel_\uparrow n)\ V\ A\ q\ b$
using *alts max-agg-rej-fst-imp-seq-contained monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*
ultimately have $\forall\ b \in A - B.$ *prof-contains-result* $(m \parallel_\uparrow n)\ V\ A\ p\ q\ b$
unfolding *mod-contains-result-def mod-contains-result-sym-def*
prof-contains-result-def
by *simp*
thus *?thesis*

using *prof-contains-result-of-comps-for-elems-in-B*
by *blast*
next
assume $a \notin B$
hence *a-in-set-diff*: $a \in A - B$
using *DiffI lifted-a compatible f-profs*
unfolding *Profile.lifted-def*
by (*metis* (*no-types*, *lifting*))
hence *reject-n*: $a \in$ *reject* n V A p
using *alts f-profs*
by *blast*
hence *defer-m*: $a \in$ *defer* m V A p
using *mod-m mod-n defer-a f-profs max-agg-rej-fst-equiv-seq-contained*
unfolding *mod-contains-result-sym-def*
by (*metis* (*no-types*))
have $\forall$ $b \in B$. *mod-contains-result* $(m \parallel_\uparrow n)$ n V A p b
using *alts compatible f-profs max-agg-rej-snd-imp-seq-contained mod-contains-result-comm*
unfolding *disjoint-compatibility-def*
by *metis*
have $\forall$ $b \in B$. *mod-contains-result-sym* $(m \parallel_\uparrow n)$ n V A p b
using *alts max-agg-rej-snd-equiv-seq-contained monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*
moreover have $\forall$ $b \in A$. *prof-contains-result* n V A p q b
proof (*unfold prof-contains-result-def*, *clarify*)
fix b :: $'a$
assume *b-in-A*: $b \in A$
show $\mathcal{SCF}$*-result.electoral-module* n $\wedge$ *profile* V A p $\wedge$
profile V A q $\wedge$ $b \in A$ $\wedge$
($b \in$ *elect* n V A p $\longrightarrow$ $b \in$ *elect* n V A q) $\wedge$
($b \in$ *reject* n V A p $\longrightarrow$ $b \in$ *reject* n V A q) $\wedge$
($b \in$ *defer* n V A p $\longrightarrow$ $b \in$ *defer* n V A q)
proof (*safe*)
show $\mathcal{SCF}$*-result.electoral-module* n
using *monotone-n*
unfolding *defer-lift-invariance-def*
by *metis*
next
show
profile V A p **and**
profile V A q **and**
$b \in A$
using *f-profs b-in-A*
by (*simp*, *simp*, *simp*)
next
show
$b \in$ *elect* n V A $p \Longrightarrow b \in$ *elect* n V A q **and**
$b \in$ *reject* n V A $p \Longrightarrow b \in$ *reject* n V A q **and**
$b \in$ *defer* n V A $p \Longrightarrow b \in$ *defer* n V A q

using *alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by (*metis*, *metis*, *metis*)
qed
qed
moreover have $\forall\ b \in B.$ *mod-contains-result* n $(m \parallel_{\uparrow} n)$ $V\ A\ q\ b$
using *alts compatible max-agg-rej-snd-imp-seq-contained f-profs*
unfolding *disjoint-compatibility-def*
by *metis*
ultimately have *prof-contains-result-of-comps-for-elems-in-B*:
$\forall\ b \in B.$ *prof-contains-result* $(m \parallel_{\uparrow} n)$ $V\ A\ p\ q\ b$
unfolding *mod-contains-result-def mod-contains-result-sym-def*
prof-contains-result-def
by *simp*
have $\forall\ b \in A - B.$ *mod-contains-result-sym* $(m \parallel_{\uparrow} n)$ $m\ V\ A\ p\ b$
using *alts max-agg-rej-fst-equiv-seq-contained monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*
moreover have $\forall\ b \in A.$ *prof-contains-result* $m\ V\ A\ p\ q\ b$
proof (*unfold prof-contains-result-def*, *clarify*)
fix $b :: 'a$
assume *b-in-A*: $b \in A$
show $\mathcal{SCF}$*-result.electoral-module* $m \wedge$ *profile* $V\ A\ p$
$\wedge$ *profile* $V\ A\ q \wedge b \in A$
$\wedge$ $(b \in$ *elect* $m\ V\ A\ p \longrightarrow b \in$ *elect* $m\ V\ A\ q)$
$\wedge$ $(b \in$ *reject* $m\ V\ A\ p \longrightarrow b \in$ *reject* $m\ V\ A\ q)$
$\wedge$ $(b \in$ *defer* $m\ V\ A\ p \longrightarrow b \in$ *defer* $m\ V\ A\ q)$
proof (*safe*)
show $\mathcal{SCF}$*-result.electoral-module* m
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *simp*
next
show
profile $V\ A\ p$ **and**
profile $V\ A\ q$ **and**
$b \in A$
using *f-profs b-in-A*
by (*simp*, *simp*, *simp*)
next
show
$b \in$ *elect* $m\ V\ A\ p \Longrightarrow b \in$ *elect* $m\ V\ A\ q$ **and**
$b \in$ *reject* $m\ V\ A\ p \Longrightarrow b \in$ *reject* $m\ V\ A\ q$ **and**
$b \in$ *defer* $m\ V\ A\ p \Longrightarrow b \in$ *defer* $m\ V\ A\ q$
using *defer-m lifted-a monotone-m*
unfolding *defer-lift-invariance-def*
by (*metis*, *metis*, *metis*)
qed
qed

moreover have ∀ $x \in A - B$. *mod-contains-result m* $(m \parallel_\uparrow n)$ *V A q x*
  using *alts max-agg-rej-fst-imp-seq-contained monotone-m monotone-n f-profs*
  unfolding *defer-lift-invariance-def*
  by *metis*
ultimately have ∀ $x \in A - B$. *prof-contains-result* $(m \parallel_\uparrow n)$ *V A p q x*
  unfolding *mod-contains-result-def mod-contains-result-sym-def*
      prof-contains-result-def
  by *simp*
thus *?thesis*
  using *prof-contains-result-of-comps-for-elems-in-B*
  by *blast*
qed
thus $(m \parallel_\uparrow n)$ *V A p* = $(m \parallel_\uparrow n)$ *V A q*
  using *compatible f-profs eq-alts-in-profs-imp-eq-results max-par-comp-sound*
  unfolding *disjoint-compatibility-def*
  by *metis*
qed

lemma *par-comp-rej-card*:
 fixes
  m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile* **and**
  c :: *nat*
 assumes
  compatible: *disjoint-compatibility m n* **and**
  prof: *profile V A p* **and**
  fin-A: *finite A* **and**
  reject-sum: *card* (*reject m V A p*) + *card* (*reject n V A p*) = *card A* + *c*
 shows *card* (*reject* $(m \parallel_\uparrow n)$ *V A p*) = *c*
proof −
 obtain *B* :: *'a set* **where**
  alt-set: $B \subseteq A$
   ∧ (∀ $a \in B$. *indep-of-alt m V A a* ∧
     (∀ *q*. *profile V A q* ⟶ $a \in$ *reject m V A q*))
   ∧ (∀ $a \in A - B$. *indep-of-alt n V A a* ∧
     (∀ *q*. *profile V A q* ⟶ $a \in$ *reject n V A q*))
  using *compatible prof*
  unfolding *disjoint-compatibility-def*
  by *metis*
 have *reject-representation*:
  reject $(m \parallel_\uparrow n)$ *V A p* = (*reject m V A p*) ∩ (*reject n V A p*)
  using *prof fin-A compatible max-agg-rej-intersect*
  unfolding *disjoint-compatibility-def*
  by *metis*
 have $\mathcal{SCF}$*-result.electoral-module m* ∧ $\mathcal{SCF}$*-result.electoral-module n*
  using *compatible*

unfolding *disjoint-compatibility-def*
by *simp*
hence *subsets*: $(reject\ m\ V\ A\ p) \subseteq A \wedge (reject\ n\ V\ A\ p) \subseteq A$
using *prof*
by (*simp add*: *reject-in-alts*)
hence *finite* $(reject\ m\ V\ A\ p) \wedge$ *finite* $(reject\ n\ V\ A\ p)$
using *rev-finite-subset prof fin-A*
by *metis*
hence *card-difference*:
$card\ (reject\ (m \parallel_{\uparrow} n)\ V\ A\ p)$
$= card\ A + c - card\ ((reject\ m\ V\ A\ p) \cup (reject\ n\ V\ A\ p))$
using *card-Un-Int reject-representation reject-sum*
by *fastforce*
have $\forall\ a \in A.\ a \in (reject\ m\ V\ A\ p) \vee a \in (reject\ n\ V\ A\ p)$
using *alt-set prof fin-A*
by *blast*
hence $A = reject\ m\ V\ A\ p \cup reject\ n\ V\ A\ p$
using *subsets*
by *force*
thus $card\ (reject\ (m \parallel_{\uparrow} n)\ V\ A\ p) = c$
using *card-difference*
by *simp*
qed

Using the max-aggregator for composing two compatible modules in parallel, whereof the first one is non-electing and defers exactly one alternative, and the second one rejects exactly two alternatives, the composition results in an electoral module that eliminates exactly one alternative.

theorem *par-comp-elim-one*[*simp*]:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
n :: ('a, 'v, 'a *Result*) *Electoral-Module*
assumes
defers-m-one: *defers 1 m* **and**
non-elec-m: *non-electing m* **and**
rejec-n-two: *rejects 2 n* **and**
disj-comp: *disjoint-compatibility m n*
shows *eliminates 1* $(m \parallel_{\uparrow} n)$
proof (*unfold eliminates-def*, *safe*)
have $\mathcal{SCF}$*-result.electoral-module m*
using *non-elec-m*
unfolding *non-electing-def*
by *simp*
moreover have $\mathcal{SCF}$*-result.electoral-module n*
using *rejec-n-two*
unfolding *rejects-def*
by *simp*
ultimately show $\mathcal{SCF}$*-result.electoral-module* $(m \parallel_{\uparrow} n)$
by *simp*

next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile*
  assume
    min-card-two: *1* < *card A* **and**
    prof: *profile V A p*
  hence *card-geq-one*: *card A* ≥ *1*
    by *presburger*
  have *fin-A*: *finite A*
    using *min-card-two card.infinite not-one-less-zero*
    by *metis*
  have *module*: *SCF-result.electoral-module m*
    using *non-elec-m*
    unfolding *non-electing-def*
    by *simp*
  have *elect-card-zero*: *card* (*elect m V A p*) = *0*
    using *prof non-elec-m card-eq-0-iff*
    unfolding *non-electing-def*
    by *simp*
  moreover from *card-geq-one*
  have *def-card-one*: *card* (*defer m V A p*) = *1*
    using *defers-m-one module prof fin-A*
    unfolding *defers-def*
    by *blast*
  ultimately have *card-reject-m*: *card* (*reject m V A p*) = *card A* − *1*
  proof −
    have *well-formed-SCF A* (*elect m V A p*, *reject m V A p*, *defer m V A p*)
      using *prof module*
      unfolding *SCF-result.electoral-module-def*
      by *simp*
    hence *card A* =
        card (*elect m V A p*) + *card* (*reject m V A p*) + *card* (*defer m V A p*)
      using *result-count fin-A*
      by *blast*
    thus *?thesis*
      using *def-card-one elect-card-zero*
      by *simp*
  qed
  have *card A* ≥ *2*
    using *min-card-two*
    by *simp*
  hence *card* (*reject n V A p*) = *2*
    using *prof rejec-n-two fin-A*
    unfolding *rejects-def*
    by *blast*
  moreover from *this*
  have *card* (*reject m V A p*) + *card* (*reject n V A p*) = *card A* + *1*

```
    using card-reject-m card-geq-one
    by linarith
  ultimately show card (reject (m ∥↑ n) V A p) = 1
    using disj-comp prof card-reject-m par-comp-rej-card fin-A
    by blast
qed

end
```

6.7 Elect Composition

```
theory Elect-Composition
  imports Basic-Modules/Elect-Module
          Sequential-Composition
begin
```

The elect composition sequences an electoral module and the elect module. It finalizes the module's decision as it simply elects all their non-rejected alternatives. Thereby, any such elect-composed module induces a proper voting rule in the social choice sense, as all alternatives are either rejected or elected.

6.7.1 Definition

```
fun elector :: ('a, 'v, 'a Result) Electoral-Module
                ⇒ ('a, 'v, 'a Result) Electoral-Module where
  elector m = (m ▷ elect-module)
```

6.7.2 Auxiliary Lemmas

```
lemma elector-seqcomp-assoc:
  fixes
    a :: ('a, 'v, 'a Result) Electoral-Module and
    b :: ('a, 'v, 'a Result) Electoral-Module
  shows (a ▷ (elector b)) = (elector (a ▷ b))
  unfolding elector.simps elect-module.simps sequential-composition.simps
  using boolean-algebra-cancel.sup2 fst-eqD snd-eqD sup-commute
  by (metis (no-types, opaque-lifting))
```

6.7.3 Soundness

```
theorem elector-sound[simp]:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes SCℱ-result.electoral-module m
```

shows $\mathcal{SCF}$*-result.electoral-module* (*elector m*)
using *assms*
by *simp*

lemma *elector-only-voters*:
fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes *only-voters-vote m*
shows *only-voters-vote* (*elector m*)
using *assms*
by (*simp add*: *elect-mod-only-voters seq-comp-only-voters*)

6.7.4 Electing

theorem *elector-electing*[*simp*]:
fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module m* **and**
non-block-m: *non-blocking m*
shows *electing* (*elector m*)
proof −
have $\forall$ *m'*.
$(\neg$ *electing m'* $\vee$ $\mathcal{SCF}$*-result.electoral-module m'* $\wedge$
$(\forall$ *A' V' p'*. $(A' \neq \{\} \wedge$ *finite A'* $\wedge$ *profile V' A' p'*$)$
$\longrightarrow$ *elect m' V' A' p'* $\neq \{\})) \wedge$
$($*electing m'* $\vee \neg$ $\mathcal{SCF}$*-result.electoral-module m'*
$\vee$ $(\exists$ *A V p*. $(A \neq \{\} \wedge$ *finite A* $\wedge$ *profile V A p* $\wedge$ *elect m' V A p* $= \{\})))$
unfolding *electing-def*
by *blast*
hence $\forall$ *m'*.
$(\neg$ *electing m'* $\vee$ $\mathcal{SCF}$*-result.electoral-module m'* $\wedge$
$(\forall$ *A' V' p'*. $(A' \neq \{\} \wedge$ *finite A'* $\wedge$ *profile V' A' p'*$)$
$\longrightarrow$ *elect m' V' A' p'* $\neq \{\})) \wedge$
$(\exists$ *A V p*. $($*electing m'* $\vee \neg$ $\mathcal{SCF}$*-result.electoral-module m'* $\vee A \neq \{\}$
$\wedge$ *finite A* $\wedge$ *profile V A p* $\wedge$ *elect m' V A p* $= \{\}))$
by *simp*
then obtain
A :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *'a set* **and**
V :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ *'v set* **and**
p :: (*'a*, *'v*, *'a Result*) *Electoral-Module* $\Rightarrow$ (*'a*, *'v*) *Profile* **where**
electing-mod:
$\forall$ *m'*::(*'a*, *'v*, *'a Result*) *Electoral-Module*.
$(\neg$ *electing m'* $\vee$ $\mathcal{SCF}$*-result.electoral-module m'* $\wedge$
$(\forall$ *A' V' p'*. $(A' \neq \{\} \wedge$ *finite A'* $\wedge$ *profile V' A' p'*$)$
$\longrightarrow$ *elect m' V' A' p'* $\neq \{\})) \wedge$
$($*electing m'* $\vee \neg$ $\mathcal{SCF}$*-result.electoral-module m'*
$\vee$ *A m'* $\neq \{\} \wedge$ *finite* (*A m'*) $\wedge$ *profile* (*V m'*) (*A m'*) (*p m'*)
$\wedge$ *elect m'* (*V m'*) (*A m'*) (*p m'*) $= \{\})$
by *metis*
moreover have *non-block*:

non-blocking (*elect-module*::′*v set* ⇒ ′*a set* ⇒ (′*a*, ′*v*) *Profile* ⇒ ′*a Result*)
by (*simp add*: *electing-imp-non-blocking*)
moreover obtain
e :: ′*a Result* ⇒ ′*a set* **and**
r :: ′*a Result* ⇒ ′*a set* **and**
d :: ′*a Result* ⇒ ′*a set* **where**
result: ∀ *s*. (*e s*, *r s*, *d s*) = *s*
using *disjoint3.cases*
by (*metis* (*no-types*))
moreover from *this*
have ∀ *s*. (*elect-r s*, *r s*, *d s*) = *s*
by *simp*
moreover from *this*
have
profile (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*)) ∧ *finite* (*A* (*elector m*))
⟶ *d* (*elector m* (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*))) = {}
by *simp*
moreover have *SCF-result.electoral-module* (*elector m*)
using *elector-sound module-m*
by *simp*
moreover from *electing-mod result*
have *finite* (*A* (*elector m*)) ∧
profile (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*)) ∧
elect (*elector m*) (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*)) = {} ∧
d (*elector m* (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*))) = {} ∧
reject (*elector m*) (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*)) =
r (*elector m* (*V* (*elector m*)) (*A* (*elector m*)) (*p* (*elector m*))) ⟶
electing (*elector m*)
using *Diff-empty elector.simps non-block-m snd-conv non-blocking-def reject-not-elec-or-def non-block seq-comp-presv-non-blocking*
by (*metis* (*mono-tags*, *opaque-lifting*))
ultimately show *?thesis*
using *non-block-m*
unfolding *elector.simps*
by *auto*
qed

6.7.5 Composition Rule

If m is defer-Condorcet-consistent, then elector(m) is Condorcet consistent.

lemma *dcc-imp-cc-elector*:
fixes *m* :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module*
assumes *defer-condorcet-consistency m*
shows *condorcet-consistency* (*elector m*)
proof (*unfold defer-condorcet-consistency-def condorcet-consistency-def*, *safe*)
show *SCF-result.electoral-module* (*elector m*)
using *assms elector-sound*
unfolding *defer-condorcet-consistency-def*
by *metis*

```
next
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    w :: 'a
  assume c-win: condorcet-winner V A p w
  have fin-A: finite A
    using condorcet-winner.simps c-win
    by metis
  have fin-V: finite V
    using condorcet-winner.simps c-win
    by metis
  have prof-A: profile V A p
    using c-win
    by simp
  have max-card-w: ∀ y ∈ A − {w}.
          card {i ∈ V. (w, y) ∈ (p i)}
            < card {i ∈ V. (y, w) ∈ (p i)}
    using c-win fin-V
    by simp
  have rej-is-complement:
    reject m V A p = A − (elect m V A p ∪ defer m V A p)
    using double-diff sup-bot.left-neutral Un-upper2 assms fin-A prof-A fin-V
          defer-condorcet-consistency-def elec-and-def-not-rej reject-in-alts
    by (metis (no-types, opaque-lifting))
  have subset-in-win-set: elect m V A p ∪ defer m V A p ⊆
      {e ∈ A. e ∈ A ∧ (∀ x ∈ A − {e}.
        card {i ∈ V. (e, x) ∈ p i} < card {i ∈ V. (x, e) ∈ p i})}
  proof (safe-step)
    fix x :: 'a
    assume x-in-elect-or-defer: x ∈ elect m V A p ∪ defer m V A p
    hence x-eq-w: x = w
     using Diff-empty Diff-iff assms cond-winner-unique c-win fin-A fin-V insert-iff
            snd-conv prod.sel(1) sup-bot.left-neutral
      unfolding defer-condorcet-consistency-def
      by (metis (mono-tags, lifting))
    have ⋀ x. x ∈ elect m V A p ⟹ x ∈ A
      using fin-A prof-A fin-V assms elect-in-alts in-mono
      unfolding defer-condorcet-consistency-def
      by metis
    moreover have ⋀ x. x ∈ defer m V A p ⟹ x ∈ A
      using fin-A prof-A fin-V assms defer-in-alts in-mono
      unfolding defer-condorcet-consistency-def
      by metis
    ultimately have x ∈ A
      using x-in-elect-or-defer
      by auto
    thus x ∈ {e ∈ A. e ∈ A ∧
```

```
        (∀ x ∈ A − {e}.
          card {i ∈ V. (e, x) ∈ p i}
            < card {i ∈ V. (x, e) ∈ p i})}
    using x-eq-w max-card-w
    by auto
qed
moreover have
  {e ∈ A. e ∈ A ∧
      (∀ x ∈ A − {e}.
          card {i ∈ V. (e, x) ∈ p i} <
            card {i ∈ V. (x, e) ∈ p i})}
        ⊆ elect m V A p ∪ defer m V A p
proof (safe)
  fix x :: 'a
  assume
    x-not-in-defer: x ∉ defer m V A p and
    x ∈ A and
    ∀ x' ∈ A − {x}.
      card {i ∈ V. (x, x') ∈ p i}
        < card {i ∈ V. (x', x) ∈ p i}
  hence c-win-x: condorcet-winner V A p x
    using fin-A prof-A fin-V
    by simp
  have (SCF-result.electoral-module m ∧ ¬ defer-condorcet-consistency m ⟶
        (∃ A V rs a. condorcet-winner V A rs a ∧
          m V A rs ≠ ({}, A − defer m V A rs,
          {a ∈ A. condorcet-winner V A rs a})))
      ∧ (defer-condorcet-consistency m ⟶
        (∀ A V rs a. finite A ⟶ finite V ⟶ condorcet-winner V A rs a ⟶
          m V A rs =
    ({}, A − defer m V A rs, {a ∈ A. condorcet-winner V A rs a})))
    unfolding defer-condorcet-consistency-def
    by blast
  hence
    m V A p = ({}, A − defer m V A p, {a ∈ A. condorcet-winner V A p a})
    using c-win-x assms fin-A fin-V
    by blast
  thus x ∈ elect m V A p
    using assms x-not-in-defer fin-A fin-V cond-winner-unique
          defer-condorcet-consistency-def insertCI prod.sel(2) c-win-x
    by (metis (no-types, lifting))
qed
ultimately have
  elect m V A p ∪ defer m V A p =
    {e ∈ A. e ∈ A ∧
      (∀ x ∈ A − {e}.
        card {i ∈ V. (e, x) ∈ p i} <
          card {i ∈ V. (x, e) ∈ p i})}
  by blast
```

thus *elector m V A p* =
({*e* ∈ *A*. *condorcet-winner V A p e*}, *A* − *elect* (*elector m*) *V A p*, {})
using *fin-A prof-A fin-V rej-is-complement*
by *simp*
qed

end

6.8 Defer One Loop Composition

theory *Defer-One-Loop-Composition*
imports *Basic-Modules/Component-Types/Defer-Equal-Condition*
Loop-Composition
Elect-Composition
begin

This is a family of loop compositions. It uses the same module in sequence until either no new decisions are made or only one alternative is remaining in the defer-set. The second family herein uses the above family and subsequently elects the remaining alternative.

6.8.1 Definition

fun *iter* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
iter m =
(*let t* = *defer-equal-condition 1 in*
($m \circlearrowleft_t$))

abbreviation *defer-one-loop* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* (-$\circlearrowleft_{\exists !d}$ *50*) **where**
$m \circlearrowleft_{\exists !d}$ ≡ *iter m*

fun *iterelect* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
iterelect m = *elector* ($m \circlearrowleft_{\exists !d}$)

end

Chapter 7

Voting Rules

7.1 Plurality Rule

theory *Plurality-Rule*
  imports *Compositional-Structures/Basic-Modules/Plurality-Module*
    Compositional-Structures/Revision-Composition
    Compositional-Structures/Elect-Composition
begin

This is a definition of the plurality voting rule as elimination module as well as directly. In the former one, the max operator of the set of the scores of all alternatives is evaluated and is used as the threshold value.

7.1.1 Definition

fun *plurality-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
  plurality-rule V A p = *elector plurality V A p*

fun *plurality-rule'* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
  plurality-rule' V A p =
    $(\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\}$,
    $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\}$,
    $\{\})$

lemma *plurality-revision-equiv*:
  fixes
    A :: '*a set* **and**
    V :: '*v set* **and**
    p :: ('*a*, '*v*) *Profile*
  shows *plurality' V A p* = (*plurality-rule'*$\downarrow$) *V A p*
proof (*unfold plurality-rule'.simps plurality'.simps revision-composition.simps*,
    standard, *clarsimp*, *standard*, *safe*)
  fix
    a :: '*a* **and**
    b :: '*a*

assume
finite V **and**
$b \in A$ **and**
card $\{i.\ i \in V \wedge above\ (p\ i)\ a = \{a\}\}$
$< card\ \{i.\ i \in V \wedge above\ (p\ i)\ b = \{b\}\}$ **and**
$\forall\ a' \in A.\ card\ \{i.\ i \in V \wedge above\ (p\ i)\ a' = \{a'\}\}$
$\leq card\ \{i.\ i \in V \wedge above\ (p\ i)\ a = \{a\}\}$
thus *False*
using *leD*
by *blast*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume
finite V **and**
$b \in A$ **and**
$\neg\ card\ \{i.\ i \in V \wedge above\ (p\ i)\ b = \{b\}\}$
$\leq card\ \{i.\ i \in V \wedge above\ (p\ i)\ a = \{a\}\}$
thus $\exists\ x \in A.$
$card\ \{i.\ i \in V \wedge above\ (p\ i)\ a = \{a\}\}$
$< card\ \{i.\ i \in V \wedge above\ (p\ i)\ x = \{x\}\}$
using *linorder-not-less*
by *blast*
next
fix
$a :: 'a$ **and**
$b :: 'a$
assume
finite V **and**
$b \in A$ **and**
$a \in A$ **and**
$card\ \{v \in V.\ above\ (p\ v)\ a = \{a\}\} < card\ \{v \in V.\ above\ (p\ v)\ b = \{b\}\}$ **and**
$\forall\ c \in A.\ card\ \{v \in V.\ above\ (p\ v)\ c = \{c\}\}$
$\leq card\ \{v \in V.\ above\ (p\ v)\ a = \{a\}\}$
thus *False*
by *auto*
qed

lemma *plurality-elim-equiv*:
fixes
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ Profile$
assumes
$A \neq \{\}$ **and**
finite A **and**
profile $V\ A\ p$
shows $plurality\ V\ A\ p = (plurality\text{-}rule'\downarrow)\ V\ A\ p$

using *assms plurality-mod-elim-equiv plurality-revision-equiv*
by (*metis* (*full-types*))

7.1.2 Soundness

theorem *plurality-rule-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module plurality-rule*
unfolding *plurality-rule.simps*
using *elector-sound plurality-sound*
by *metis*

theorem *plurality-rule′-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module plurality-rule′*
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module-def*, *safe*)
fix
A :: *′a set* **and**
V :: *′v set* **and**
p :: (*′a*, *′v*) *Profile*
have *disjoint3* (
$\{a \in A.\ \forall\ a' \in A.\ \textit{win-count}\ V\ p\ a' \le \textit{win-count}\ V\ p\ a\}$,
$\{a \in A.\ \exists\ a' \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ a'\}$,
$\{\}$)
by *auto*
moreover have
$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \le \textit{win-count}\ V\ p\ a\}\ \cup$
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\} = A$
using *not-le-imp-less*
by *auto*
ultimately show *well-formed-*$\mathcal{SCF}$ A (*plurality-rule′* V A p)
by *simp*
qed

lemma *plurality-rule-only-voters*:
only-voters-vote plurality-rule
unfolding *plurality-rule.simps*
using *elector-only-voters plurality-only-voters*
by *blast*

7.1.3 Electing

lemma *plurality-rule-elect-non-empty*:
fixes
A :: *′a set* **and**
V :: *′v set* **and**
p :: (*′a*, *′v*) *Profile*
assumes
A-non-empty: $A \neq \{\}$ **and**
prof-A: *profile* V A p **and**
fin-A: *finite* A
shows *elect plurality-rule* V A $p \neq \{\}$
proof
assume *plurality-elect-none*: *elect plurality-rule* V A $p = \{\}$

obtain *max* **where**
max: $max = Max$ (*win-count* V p ‘ A)
by *simp*
then obtain a **where**
max-a: *win-count* V p $a = max \wedge a \in A$
using *Max-in A-non-empty fin-A prof-A empty-is-image finite-imageI imageE*
by (*metis* (*no-types*, *lifting*))
hence $\forall$ $a' \in A$. *win-count* V p $a' \leq$ *win-count* V p a
using *fin-A prof-A max*
by *simp*
moreover have $a \in A$
using *max-a*
by *simp*
ultimately have $a \in \{a' \in A.\ \forall\ c \in A.$ *win-count* V p $c \leq$ *win-count* V p $a'\}$
by *blast*
hence $a \in$ *elect plurality-rule'* V A p
by *simp*
moreover have *elect plurality-rule'* V A p = *defer plurality* V A p
using *plurality-elim-equiv fin-A prof-A A-non-empty snd-conv*
unfolding *revision-composition.simps*
by *metis*
ultimately have $a \in$ *defer plurality* V A p
by *blast*
hence $a \in$ *elect plurality-rule* V A p
by *simp*
thus *False*
using *plurality-elect-none all-not-in-conv*
by *metis*
qed

The plurality module is electing.

theorem *plurality-rule-electing*[*simp*]: *electing plurality-rule*
proof (*unfold electing-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module plurality-rule*
using *plurality-rule-sound*
by *simp*
next
fix
A :: *'b set* **and**
V :: *'a set* **and**
p :: (*'b*, *'a*) *Profile* **and**
a :: *'b*
assume
fin-A: *finite* A **and**
prof-p: *profile* V A p **and**
elect-none: *elect plurality-rule* V A $p = \{\}$ **and**
a-in-A: $a \in A$
have $\forall$ A V p. $A \neq \{\} \wedge$ *finite* $A \wedge$ *profile* V A p
$\longrightarrow$ *elect plurality-rule* V A $p \neq \{\}$

```
    using plurality-rule-elect-non-empty
    by (metis (no-types))
  hence empty-A: A = {}
    using fin-A prof-p elect-none
    by (metis (no-types))
  thus a ∈ {}
    using a-in-A
    by simp
qed
```

7.1.4 Property

```
lemma plurality-rule-inv-mono-eq:
  fixes
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and
    a :: 'a
  assumes
    elect-a: a ∈ elect plurality-rule V A p and
    lift-a: lifted V A p q a
  shows elect plurality-rule V A q = elect plurality-rule V A p
        ∨ elect plurality-rule V A q = {a}
proof −
  have a ∈ elect (elector plurality) V A p
    using elect-a
    by simp
  moreover have eq-p: elect (elector plurality) V A p = defer plurality V A p
    by simp
  ultimately have a ∈ defer plurality V A p
    by blast
  hence defer plurality V A q = defer plurality V A p
        ∨ defer plurality V A q = {a}
    using lift-a plurality-def-inv-mono-alts
    by metis
  moreover have elect (elector plurality) V A q = defer plurality V A q
    by simp
  ultimately show
    elect plurality-rule V A q = elect plurality-rule V A p
      ∨ elect plurality-rule V A q = {a}
    using eq-p
    by simp
qed
```

The plurality rule is invariant-monotone.

```
theorem plurality-rule-inv-mono[simp]: invariant-monotonicity plurality-rule
proof (unfold invariant-monotonicity-def, intro conjI impI allI)
  show SCF-result.electoral-module plurality-rule
```

by *simp*
next
fix
A :: *'b set* **and**
V :: *'a set* **and**
p :: (*'b*, *'a*) *Profile* **and**
q :: (*'b*, *'a*) *Profile* **and**
a :: *'b*
assume $a \in$ *elect plurality-rule V A p* $\wedge$ *Profile.lifted V A p q a*
thus *elect plurality-rule V A q* = *elect plurality-rule V A p*
$\vee$ *elect plurality-rule V A q* = $\{a\}$
using *plurality-rule-inv-mono-eq*
by *metis*
qed

end

7.2 Borda Rule

theory *Borda-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Basic-Modules/Component-Types/Votewise-Distance-Rationalization
Compositional-Structures/Elect-Composition
begin

This is the Borda rule. On each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected.

7.2.1 Definition

fun *borda-rule* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
borda-rule V A p = *elector borda V A p*

fun *borda-rule*$_{\mathcal{R}}$ :: (*'a*, *'v*::*wellorder*, *'a Result*) *Electoral-Module* **where**
borda-rule$_{\mathcal{R}}$ *V A p* = *swap-*$\mathcal{R}$ *unanimity V A p*

7.2.2 Soundness

theorem *borda-rule-sound*: $\mathcal{SCF}$*-result.electoral-module borda-rule*
unfolding *borda-rule.simps*
using *elector-sound borda-sound*
by *metis*

theorem *borda-rule*$_{\mathcal{R}}$*-sound*: $\mathcal{SCF}$*-result.electoral-module borda-rule*$_{\mathcal{R}}$
unfolding *borda-rule*$_{\mathcal{R}}$*.simps swap-*$\mathcal{R}$*.simps*
using $\mathcal{SCF}$*-result.*$\mathcal{R}$*-sound*
by *metis*

7.2.3 Anonymity Property

theorem *borda-rule*$_{\mathcal{R}}$*-anonymous*: $\mathcal{SCF}$*-result.anonymity borda-rule*$_{\mathcal{R}}$
proof (*unfold borda-rule*$_{\mathcal{R}}$*.simps swap-*$\mathcal{R}$*.simps*)
let *?swap-dist* = *votewise-distance swap l-one*
from *l-one-is-sym*
have *distance-anonymity ?swap-dist*
using *symmetric-norm-imp-distance-anonymous*[*of l-one*]
by *simp*
with *unanimity-anonymous*
show $\mathcal{SCF}$*-result.anonymity* ($\mathcal{SCF}$*-result.distance-*$\mathcal{R}$ *?swap-dist unanimity*)
using $\mathcal{SCF}$*-result.anonymous-distance-and-consensus-imp-rule-anonymity*
by *metis*
qed

end

7.3 Pairwise Majority Rule

theory *Pairwise-Majority-Rule*
imports *Compositional-Structures/Basic-Modules/Condorcet-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the pairwise majority rule, a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives.

7.3.1 Definition

fun *pairwise-majority-rule* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
pairwise-majority-rule V A p = *elector condorcet V A p*

fun *condorcet'* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
condorcet' V A p = ((*min-eliminator condorcet-score*) $\circlearrowleft_{\exists!d}$) *V A p*

fun *pairwise-majority-rule'* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
pairwise-majority-rule' V A p = *iterelect condorcet' V A p*

7.3.2 Soundness

theorem *pairwise-majority-rule-sound*: $\mathcal{SCF}$*-result.electoral-module pairwise-majority-rule*
 unfolding *pairwise-majority-rule.simps*
 using *condorcet-sound elector-sound*
 by *metis*

theorem *condorcet′-rule-sound*: $\mathcal{SCF}$*-result.electoral-module condorcet′*
 unfolding *condorcet′.simps*
 by (*simp add*: *loop-comp-sound*)

theorem *pairwise-majority-rule′-sound*: $\mathcal{SCF}$*-result.electoral-module pairwise-majority-rule′*
 unfolding *pairwise-majority-rule′.simps*
 using *condorcet′-rule-sound elector-sound iter.simps iterelect.simps loop-comp-sound*
 by *metis*

7.3.3 Condorcet Consistency Property

theorem *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*
proof (*unfold pairwise-majority-rule.simps*)
 show *condorcet-consistency* (*elector condorcet*)
  using *condorcet-is-dcc dcc-imp-cc-elector*
  by *metis*
qed

end

7.4 Copeland Rule

theory *Copeland-Rule*
 imports *Compositional-Structures/Basic-Modules/Copeland-Module*
  Compositional-Structures/Elect-Composition
begin

This is the Copeland voting rule. The idea is to elect the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses.

7.4.1 Definition

fun *copeland-rule* :: (*′a*, *′v*, *′a Result*) *Electoral-Module* **where**
 copeland-rule V A p = *elector copeland V A p*

7.4.2 Soundness

theorem *copeland-rule-sound*: $\mathcal{SCF}$*-result.electoral-module copeland-rule*

unfolding *copeland-rule.simps*
using *elector-sound copeland-sound*
by *metis*

7.4.3 Condorcet Consistency Property

theorem *copeland-condorcet*: *condorcet-consistency copeland-rule*
proof (*unfold copeland-rule.simps*)
show *condorcet-consistency* (*elector copeland*)
using *copeland-is-dcc dcc-imp-cc-elector*
by *metis*
qed

end

7.5 Minimax Rule

theory *Minimax-Rule*
imports *Compositional-Structures/Basic-Modules/Minimax-Module*
Compositional-Structures/Elect-Composition
begin

This is the Minimax voting rule. It elects the alternatives with the highest Minimax score.

7.5.1 Definition

fun *minimax-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
minimax-rule V A p = *elector minimax V A p*

7.5.2 Soundness

theorem *minimax-rule-sound*: $\mathcal{SCF}$*-result.electoral-module minimax-rule*
unfolding *minimax-rule.simps*
using *elector-sound minimax-sound*
by *metis*

7.5.3 Condorcet Consistency Property

theorem *minimax-condorcet*: *condorcet-consistency minimax-rule*
proof (*unfold minimax-rule.simps*)
show *condorcet-consistency* (*elector minimax*)
using *minimax-is-dcc dcc-imp-cc-elector*
by *metis*
qed

end

7.6 Black's Rule

theory *Blacks-Rule*
imports *Pairwise-Majority-Rule*
Borda-Rule
begin

This is Black's voting rule. It is composed of a function that determines the Condorcet winner, i.e., the Pairwise Majority rule, and the Borda rule. Whenever there exists no Condorcet winner, it elects the choice made by the Borda rule, otherwise the Condorcet winner is elected.

7.6.1 Definition

declare *seq-comp-alt-eq*[*simp*]

fun *black* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
black A p = (*condorcet* ▷ *borda*) *A p*

fun *blacks-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
blacks-rule A p = *elector black A p*

declare *seq-comp-alt-eq*[*simp del*]

7.6.2 Soundness

theorem *blacks-sound*: *SCF-result.electoral-module black*
unfolding *black.simps*
using *seq-comp-sound condorcet-sound borda-sound*
by *metis*

theorem *blacks-rule-sound*: *SCF-result.electoral-module blacks-rule*
unfolding *blacks-rule.simps*
using *blacks-sound elector-sound*
by *metis*

7.6.3 Condorcet Consistency Property

theorem *black-is-dcc*: *defer-condorcet-consistency black*
unfolding *black.simps*
using *condorcet-is-dcc borda-mod-non-blocking borda-mod-non-electing seq-comp-dcc*
by *metis*

theorem *black-condorcet*: *condorcet-consistency blacks-rule*

unfolding *blacks-rule.simps*
using *black-is-dcc dcc-imp-cc-elector*
by *metis*

end

7.7 Nanson-Baldwin Rule

theory *Nanson-Baldwin-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the Nanson-Baldwin voting rule. It excludes alternatives with the lowest Borda score from the set of possible winners and then adjusts the Borda score to the new (remaining) set of still eligible alternatives.

7.7.1 Definition

fun *nanson-baldwin-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
nanson-baldwin-rule A p =
((*min-eliminator borda-score*) $\circlearrowleft_{\exists !d}$) *A p*

7.7.2 Soundness

theorem *nanson-baldwin-rule-sound*: $\mathcal{SCF}$*-result.electoral-module nanson-baldwin-rule*
unfolding *nanson-baldwin-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

7.8 Classic Nanson Rule

theory *Classic-Nanson-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the classic Nanson's voting rule, i.e., the rule that was originally invented by Nanson, but not the Nanson-Baldwin rule. The idea is similar, however, as alternatives with a Borda score less or equal than the average

Borda score are excluded. The Borda scores of the remaining alternatives are hence adjusted to the new set of (still) eligible alternatives.

7.8.1 Definition

fun *classic-nanson-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
classic-nanson-rule V A p =
((*leq-average-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) V A p

7.8.2 Soundness

theorem *classic-nanson-rule-sound*: $\mathcal{SCF}$*-result.electoral-module classic-nanson-rule*
unfolding *classic-nanson-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

7.9 Schwartz Rule

theory *Schwartz-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the Schwartz voting rule. Confusingly, it is sometimes also referred as Nanson's rule. The Schwartz rule proceeds as in the classic Nanson's rule, but excludes alternatives with a Borda score that is strictly less than the average Borda score.

7.9.1 Definition

fun *schwartz-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
schwartz-rule V A p =
((*less-average-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) V A p

7.9.2 Soundness

theorem *schwartz-rule-sound*: $\mathcal{SCF}$*-result.electoral-module schwartz-rule*
unfolding *schwartz-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

7.10 Sequential Majority Comparison

theory *Sequential-Majority-Comparison*
  imports *Plurality-Rule*
    Compositional-Structures/Drop-And-Pass-Compatibility
    Compositional-Structures/Revision-Composition
    Compositional-Structures/Maximum-Parallel-Composition
    Compositional-Structures/Defer-One-Loop-Composition
begin

Sequential majority comparison compares two alternatives by plurality voting. The loser gets rejected, and the winner is compared to the next alternative. This process is repeated until only a single alternative is left, which is then elected.

7.10.1 Definition

fun $smc :: 'a\ Preference\text{-}Relation \Rightarrow ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **where**
  $smc\ x\ V\ A\ p =$
    $((elector\ ((((pass\text{-}module\ 2\ x) \triangleright ((plurality\text{-}rule{\downarrow}) \triangleright (pass\text{-}module\ 1\ x)))\ \|_{\uparrow}$
    $(drop\text{-}module\ 2\ x))\ \circlearrowleft_{\exists\,!d}))\ V\ A\ p)$

7.10.2 Soundness

As all base components are electoral modules (, aggregators, or termination conditions), and all used compositional structures create electoral modules, sequential majority comparison unsurprisingly is an electoral module.

theorem *smc-sound*:
  fixes $x :: 'a\ Preference\text{-}Relation$
  assumes *linear-order x*
  shows $\mathcal{SCF}$*-result.electoral-module* (*smc x*)
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module-def*, *simp*, *safe*, *simp-all*)
  fix
    $A :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a, 'v)\ Profile$ **and**
    $x' :: 'a$
  let $?a = max\text{-}aggregator$
  let $?t = defer\text{-}equal\text{-}condition$
  let $?smc =$
    $pass\text{-}module\ 2\ x \triangleright$
      $((plurality\text{-}rule{\downarrow}) \triangleright pass\text{-}module\ (Suc\ 0)\ x)\ \|_{?a}$
        $drop\text{-}module\ 2\ x\ \circlearrowleft_{?t}\ (Suc\ 0)$
  assume
    profile V A p **and**
    $x' \in reject\ (?smc)\ V\ A\ p$ **and**
    $x' \in elect\ (?smc)\ V\ A\ p$
  thus *False*

using *IntI drop-mod-sound emptyE loop-comp-sound max-agg-sound assms par-comp-sound pass-mod-sound plurality-rule-sound rev-comp-sound result-disj seq-comp-sound*
by *metis*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: ('a, 'v) *Profile* **and**
x' :: 'a
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality-rule*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ (*Suc 0*)
assume
profile V A p **and**
$x' \in$ *reject* (*?smc*) *V A p* **and**
$x' \in$ *defer* (*?smc*) *V A p*
thus *False*
using *IntI assms result-disj emptyE drop-mod-sound loop-comp-sound max-agg-sound par-comp-sound pass-mod-sound plurality-rule-sound rev-comp-sound seq-comp-sound*
by *metis*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p :: ('a, 'v) *Profile* **and**
x' :: 'a
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality-rule*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ (*Suc 0*)
assume
prof: *profile V A p* **and**
elect-x': $x' \in$ *elect* (*?smc*) *V A p*
have $\mathcal{SCF}$*-result.electoral-module ?smc*
by (*simp add*: *loop-comp-sound*)
thus $x' \in A$
using *prof elect-x' elect-in-alts*
by *blast*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**

p :: $('a, 'v)$ *Profile* **and**
x' :: $'a$
let $?a$ = *max-aggregator*
let $?t$ = *defer-equal-condition*
let $?smc$ =
pass-module 2 x $\triangleright$
$((plurality\text{-}rule{\downarrow}) \triangleright pass\text{-}module\ (Suc\ 0)\ x) \parallel_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ $(Suc\ 0)$
assume
prof: *profile V A p* **and**
defer-x': $x' \in$ *defer* $(?smc)$ *V A p*
have $\mathcal{SCF}$*-result.electoral-module ?smc*
by (*simp add*: *loop-comp-sound*)
thus $x' \in A$
using *prof defer-x' defer-in-alts*
by *blast*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
x' :: $'a$
let $?a$ = *max-aggregator*
let $?t$ = *defer-equal-condition*
let $?smc$ =
pass-module 2 x $\triangleright$
$((plurality\text{-}rule{\downarrow}) \triangleright pass\text{-}module\ (Suc\ 0)\ x) \parallel_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ $(Suc\ 0)$
assume
prof: *profile V A p* **and**
reject-x': $x' \in$ *reject* $(?smc)$ *V A p*
have $\mathcal{SCF}$*-result.electoral-module ?smc*
by (*simp add*: *loop-comp-sound*)
thus $x' \in A$
using *prof reject-x' reject-in-alts*
by *blast*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
x' :: $'a$
let $?a$ = *max-aggregator*
let $?t$ = *defer-equal-condition*
let $?smc$ =
pass-module 2 x $\triangleright$
$((plurality\text{-}rule{\downarrow}) \triangleright pass\text{-}module\ (Suc\ 0)\ x) \parallel_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ $(Suc\ 0)$
assume

profile V A p **and**
$x' \in A$ **and**
$x' \notin$ *defer* (*?smc*) V A p **and**
$x' \notin$ *reject* (*?smc*) V A p
thus $x' \in$ *elect* (*?smc*) V A p
using *assms electoral-mod-defer-elem drop-mod-sound loop-comp-sound max-agg-sound par-comp-sound pass-mod-sound plurality-rule-sound rev-comp-sound seq-comp-sound*
by *metis*
qed

7.10.3 Electing

The sequential majority comparison electoral module is electing. This property is needed to convert electoral modules to a social choice function. Apart from the very last proof step, it is a part of the monotonicity proof below.

theorem *smc-electing*:
fixes x :: *'a Preference-Relation*
assumes *linear-order* x
shows *electing* (*smc* x)
proof −
let *?pass2* = *pass-module 2 x*
let *?tie-breaker* = (*pass-module 1 x*)
let *?plurality-defer* = (*plurality-rule*↓) ▷ *?tie-breaker*
let *?compare-two* = *?pass2* ▷ *?plurality-defer*
let *?drop2* = *drop-module 2 x*
let *?eliminator* = *?compare-two* $\|_\uparrow$ *?drop2*
let *?loop* =
let t = defer-equal-condition 1 in (*?eliminator* $\circlearrowleft_t$)

have *00011*: *non-electing* (*plurality-rule*↓)
by *simp*
have *00012*: *non-electing ?tie-breaker*
using *assms*
by *simp*
have *00013*: *defers 1 ?tie-breaker*
using *assms pass-one-mod-def-one*
by *simp*
have *20000*: *non-blocking* (*plurality-rule*↓)
by *simp*
have *0020*: *disjoint-compatibility ?pass2 ?drop2*
using *assms*
by *simp*
have *1000*: *non-electing ?pass2*
using *assms*
by *simp*
have *1001*: *non-electing ?plurality-defer*
using *00011 00012 seq-comp-presv-non-electing*
by *blast*

```
  have 2000: non-blocking ?pass2
    using assms
    by simp
  have 2001: defers 1 ?plurality-defer
    using 20000 00011 00013 seq-comp-def-one
    by blast
  have 002: disjoint-compatibility ?compare-two ?drop2
    using assms 0020 disj-compat-seq pass-mod-sound plurality-rule-sound
          rev-comp-sound seq-comp-sound pass-mod-only-voters
          plurality-rule-only-voters seq-comp-only-voters
          rev-comp-only-voters
    by metis
  have 100: non-electing ?compare-two
    using 1000 1001 seq-comp-presv-non-electing
    by simp
  have 101: non-electing ?drop2
    using assms
    by simp
  have 102: agg-conservative max-aggregator
    by simp
  have 200: defers 1 ?compare-two
    using 2000 1000 2001 seq-comp-def-one
    by simp
  have 201: rejects 2 ?drop2
    using assms
    by simp
  have 10: non-electing ?eliminator
    using 100 101 102 conserv-max-agg-presv-non-electing
    by blast
  have 20: eliminates 1 ?eliminator
    using 200 100 201 002 par-comp-elim-one
    by simp
  have 2: defers 1 ?loop
    using 10 20 iter-elim-def-n zero-less-one prod.exhaust-sel
          defer-equal-condition.simps
    by metis
  have 3: electing elect-module
    by simp
  show ?thesis
    using 2 3 assms seq-comp-electing smc-sound
    unfolding Defer-One-Loop-Composition.iter.simps
              smc.simps elector.simps electing-def
    by metis
qed
```

7.10.4 (Weak) Monotonicity Property

The following proof is a fully modular proof for weak monotonicity of sequential majority comparison. It is composed of many small steps.

```
theorem smc-monotone:
  fixes x :: 'a Preference-Relation
  assumes linear-order x
  shows monotonicity (smc x)
proof −
  let ?pass2 = pass-module 2 x
  let ?tie-breaker = pass-module 1 x
  let ?plurality-defer = (plurality-rule↓) ▷ ?tie-breaker
  let ?compare-two = ?pass2 ▷ ?plurality-defer
  let ?drop2 = drop-module 2 x
  let ?eliminator = ?compare-two ∥↑ ?drop2
  let ?loop =
    let t = defer-equal-condition 1 in (?eliminator ↺t)

  have 00010: defer-invariant-monotonicity (plurality-rule↓)
    by simp
  have 00011: non-electing (plurality-rule↓)
    by simp
  have 00012: non-electing ?tie-breaker
    using assms
    by simp
  have 00013: defers 1 ?tie-breaker
    using assms pass-one-mod-def-one
    by simp
  have 00014: defer-monotonicity ?tie-breaker
    using assms
    by simp
  have 20000: non-blocking (plurality-rule↓)
    by simp
  have 0000: defer-lift-invariance ?pass2
    using assms
    by simp
  have 0001: defer-lift-invariance ?plurality-defer
    using 00010 00012 00013 00014 def-inv-mono-imp-def-lift-inv
    unfolding pass-module.simps only-voters-vote-def
    by blast
  have 0020: disjoint-compatibility ?pass2 ?drop2
    using assms
    by simp
  have 1000: non-electing ?pass2
    using assms
    by simp
  have 1001: non-electing ?plurality-defer
    using 00011 00012 seq-comp-presv-non-electing
    by blast
  have 2000: non-blocking ?pass2
    using assms
    by simp
  have 2001: defers 1 ?plurality-defer
```

```
    using 20000 00011 00013 seq-comp-def-one
    by blast
  have 000: defer-lift-invariance ?compare-two
    using 0000 0001 seq-comp-presv-def-lift-inv
          plurality-rule-only-voters pass-mod-only-voters
          rev-comp-only-voters seq-comp-only-voters
    by blast
  have 001: defer-lift-invariance ?drop2
    using assms
    by simp
  have 002: disjoint-compatibility ?compare-two ?drop2
    using assms 0020 disj-compat-seq pass-mod-sound pass-mod-only-voters
          plurality-rule-sound rev-comp-sound seq-comp-sound
          plurality-rule-only-voters pass-mod-only-voters
          rev-comp-only-voters seq-comp-only-voters
    by metis
  have 100: non-electing ?compare-two
    using 1000 1001 seq-comp-presv-non-electing
    by simp
  have 101: non-electing ?drop2
    using assms
    by simp
  have 102: agg-conservative max-aggregator
    by simp
  have 200: defers 1 ?compare-two
    using 2000 1000 2001 seq-comp-def-one
    by simp
  have 201: rejects 2 ?drop2
    using assms
    by simp
  have 00: defer-lift-invariance ?eliminator
    using 000 001 002 par-comp-def-lift-inv
    by blast
  have 10: non-electing ?eliminator
    using 100 101 conserv-max-agg-presv-non-electing
    by blast
  have 20: eliminates 1 ?eliminator
    using 200 100 201 002 par-comp-elim-one
    by simp
  have 0: defer-lift-invariance ?loop
    using 00 loop-comp-presv-def-lift-inv
          plurality-rule-only-voters pass-mod-only-voters drop-mod-only-voters
          rev-comp-only-voters seq-comp-only-voters max-par-comp-only-voters
    by metis
  have 1: non-electing ?loop
    using 10 loop-comp-presv-non-electing
    by simp
  have 2: defers 1 ?loop
   using 10 20 iter-elim-def-n prod.exhaust-sel zero-less-one defer-equal-condition.simps
```

```
    by metis
  have 3: electing elect-module
    by simp
  show ?thesis
    using 0 1 2 3 assms seq-comp-mono
    unfolding Electoral-Module.monotonicity-def elector.simps
              Defer-One-Loop-Composition.iter.simps
              smc-sound smc.simps
    by (metis (full-types))
qed

end
```

7.11 Kemeny Rule

```
theory Kemeny-Rule
  imports
   Compositional-Structures/Basic-Modules/Component-Types/Votewise-Distance-Rationalization
   Compositional-Structures/Basic-Modules/Component-Types/Distance-Rationalization-Symmetry
begin
```

This is the Kemeny rule. It creates a complete ordering of alternatives and evaluates each ordering of the alternatives in terms of the sum of preference reversals on each ballot that would have to be performed in order to produce that transitive ordering. The complete ordering which requires the fewest preference reversals is the final result of the method.

7.11.1 Definition

```
fun kemeny-rule :: ('a, 'v::wellorder, 'a Result) Electoral-Module where
  kemeny-rule V A p = swap-ℛ strong-unanimity V A p
```

7.11.2 Soundness

```
theorem kemeny-rule-sound: 𝒮𝒞ℱ-result.electoral-module kemeny-rule
  unfolding kemeny-rule.simps swap-ℛ.simps
  using 𝒮𝒞ℱ-result.ℛ-sound
  by metis
```

7.11.3 Anonymity Property

```
theorem kemeny-rule-anonymous: 𝒮𝒞ℱ-result.anonymity kemeny-rule
proof (unfold kemeny-rule.simps swap-ℛ.simps)
  let ?swap-dist = votewise-distance swap l-one
  have distance-anonymity ?swap-dist
    using l-one-is-sym symmetric-norm-imp-distance-anonymous[of l-one]
```

by *simp*
thus *SCF-result.anonymity*
(*SCF-result.distance-R ?swap-dist strong-unanimity*)
using *strong-unanimity-anonymous*
SCF-result.anonymous-distance-and-consensus-imp-rule-anonymity
by *metis*
qed

7.11.4 Neutrality Property

lemma *swap-dist-neutral*: *distance-neutrality valid-elections*
(*votewise-distance swap l-one*)
using *neutral-dist-imp-neutral-votewise-dist swap-neutral*
by *blast*

theorem *kemeny-rule-neutral*: *SCF-properties.neutrality valid-elections kemeny-rule*
using *strong-unanimity-neutral′ swap-dist-neutral strong-unanimity-closed-under-neutrality*
SCF-properties.neutr-dist-and-cons-imp-neutr-dr
unfolding *kemeny-rule.simps swap-R.simps*
by *blast*

end

Bibliography

[1] K. Diekhoff, M. Kirsten, and J. Krämer. Formal property-oriented design of voting rules using composable modules. In S. Pekeč and K. Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019.

[2] K. Diekhoff, M. Kirsten, and J. Krämer. Verified construction of fair voting rules. In M. Gabbrielli, editor, *29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019), Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2020.