Verified Construction of Fair Voting Rules

Michael Kirsten

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

November 28, 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 Auxiliary Lemmas

theory *Auxiliary-Lemmas*
  imports *Main*
begin

Summation function is invariant under application of a (bijective) permutation on the elements.

lemma *sum-comp*:
  fixes
    $f :: 'x \Rightarrow ('z :: \textit{comm-monoid-add})$ **and**
    $g :: 'y \Rightarrow 'x$ **and**
    $X :: 'x\ \textit{set}$ **and**
    $Y :: 'y\ \textit{set}$
  assumes *bij-betw* $g\ Y\ X$
  shows $(\sum\ x \in X.\ f\ x) = (\sum\ x \in Y.\ (f \circ g)\ x)$
  using *assms sum.reindex*
  unfolding *bij-betw-def*
  by (*metis* (*no-types*))

The inversion of a composition of injective functions is equivalent to the composition of the two individual inverted functions.

lemma *the-inv-comp*:
  fixes
    $X :: 'x\ \textit{set}$ **and**
    $Y :: 'y\ \textit{set}$ **and**
    $Z :: 'z\ \textit{set}$ **and**
    $f :: 'y \Rightarrow 'x$ **and**
    $g :: 'z \Rightarrow 'y$ **and**
    $x :: 'x$
  assumes
    bij-betw $f\ Y\ X$ **and**
    bij-betw $g\ Z\ Y$ **and**
    $x \in X$

shows *the-inv-into* Z $(f \circ g)$ $x = ((\textit{the-inv-into } Z\ g) \circ (\textit{the-inv-into } Y\ f))\ x$
using *assms the-inv-into-comp*
unfolding *bij-betw-def*
by *metis*

end

1.2 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.

1.2.1 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*

1.2.2 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

1.2.3 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 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 ⇒ 'a Preference-Relation ⇒ 'a Preference-Relation where
  limit A r = {(a, b) ∈ r. a ∈ A ∧ b ∈ A}

definition connex :: 'a set ⇒ 'a Preference-Relation ⇒ bool where
  connex A r ≡ limited A r ∧ (∀ a ∈ A. ∀ b ∈ A. a ≼r b ∨ b ≼r a)

lemma connex-imp-refl:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation
  assumes connex A r
  shows refl-on A r
  using assms
proof (unfold connex-def refl-on-def limited-def, elim conjE conjI, safe)
  fix a :: 'a
  assume a ∈ A
  hence a ≼r a
    using assms
    unfolding connex-def
    by metis
  thus (a, a) ∈ 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 b :: 'a
  assume (a, b) ∈ r
  moreover have refl-on A r
    using assms partial-order-onD
    unfolding linear-order-on-def
    by safe
  ultimately show
    a ∈ A and
    b ∈ A
    by (simp-all add: refl-on-domain)
next
  fix a b :: 'a
  assume
    a ∈ A and
    b ∈ A and
```

$\neg\ b \preceq_r a$
  moreover from *this*
  have $(b, a) \notin r$
    by *simp*
  moreover 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 $a\ b :: 'a$
  assume $(a, b) \in r$
  thus
    $a \in A$ **and**
    $b \in A$
    using *connex-r refl-on-domain connex-imp-refl*
    by (*metis*, *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
  show *trans r*
    using *trans-r*
    by *simp*
next
  show *antisym r*
    using *antisym-r*
    by *simp*
next

```
  fix a b :: 'a
  assume
    a ∈ A and
    b ∈ A and
    (b, a) ∉ r
  moreover with connex-r
  have a ≼_r b ∨ b ≼_r a
    unfolding connex-def
    by metis
  hence (a, b) ∈ r ∨ (b, a) ∈ r
    by simp
  ultimately show (a, b) ∈ r
    by metis
qed

lemma limit-to-limits:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation
  shows limited A (limit A r)
  unfolding limited-def
  by fastforce

lemma limit-presv-connex:
  fixes
    A B :: 'a set and
    r :: 'a Preference-Relation
  assumes
    connex: connex B r and
    subset: A ⊆ B
  shows connex A (limit A r)
proof (unfold connex-def limited-def limit.simps is-less-preferred-than.simps, safe)
  let ?s = {(a, b). (a, b) ∈ r ∧ a ∈ A ∧ b ∈ A}
  fix a b :: 'a
  assume
    a-in-A: a ∈ A and
    b-in-A: b ∈ A and
    not-b-pref-r-a: (b, a) ∉ r
  have b ≼_r a ∨ a ≼_r b
    using a-in-A b-in-A connex connex-def in-mono subset
    by metis
  hence a ≼_?s b ∨ b ≼_?s a
    using a-in-A b-in-A
    by auto
  thus (a, b) ∈ r
    using not-b-pref-r-a
    by simp
qed
```

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

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

lemma *limit-trans*:
fixes
$A\ B$ :: $'a$ *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

1.2.4 Auxiliary Lemmas

lemma *above-trans*:
fixes
r :: $'a$ *Preference-Relation* **and**
$a\ b$ :: $'a$
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 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**
$a\ b$ :: $'a$
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**
$a\ b$ :: $'a$
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
$A\ B$ :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
$a\ b$ :: $'a$
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* $A \land A \neq \{\} \land n + 1 = card\ A$
$\longrightarrow (\exists\ a \in A.\ above\ r\ a = \{a\})$
proof (*induction* n *arbitrary*: $A\ r$; *clarify*)
case *0*

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* :: *′a* **where**
A′ = {*a*}
using *card-1-singletonE add.left-neutral*
by *metis*
hence
a ∈ *A′* **and**
above r′ a = {*a*}
using *lin-ord-r connex-imp-refl above-refl lin-ord-imp-connex refl-on-domain*
unfolding *above-def*
by (*blast*, *fast*)
thus ∃ *a′* ∈ *A′*. *above r′ a′* = {*a′*}
by *metis*
next
case (*Suc n*)
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′* ≠ {} **and**
len-A-n-plus-one: *Suc n + 1 = card A′*
then obtain *B* :: *′a set* **where**
subset-B-card: *card B = n + 1* ∧ *B* ⊆ *A′*
using *Suc-inject add-Suc card.insert-remove finite.cases insert-Diff-single subset-insertI*
by (*metis* (*mono-tags*, *lifting*))
then obtain *a* :: *′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 ∃ *a′* ∈ *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* :: *′a* **where**
alt-b: *above* (*limit B r′*) *b* = {*b*}
by *blast*
hence *b-above*: {*a′*. (*b*, *a′*) ∈ *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 $\exists\ a' \in A'.\ above\ r'\ a' = \{a'\}$
proof (*cases*)
  assume *a-pref-r-b*: $a \preceq_{r'} b$
  have *refl-A*:
    $\forall\ A''\ r''\ a'\ a''.$
        $refl\text{-}on\ A''\ r'' \wedge (a' :: 'a,\ a'') \in r'' \longrightarrow a' \in A'' \wedge a'' \in A''$
    using *refl-on-domain*
    by *metis*
  have $\forall\ A''\ r''.\ linear\text{-}order\text{-}on\ (A'' :: 'a\ set)\ r'' \longrightarrow connex\ A''\ r''$
    by (*simp add*: *lin-ord-imp-connex*)
  hence *refl-A'*: $refl\text{-}on\ A'\ r'$
    using *connex-imp-refl lin-ord-r*
    by *metis*
  hence $a \in A' \wedge b \in A'$
    using *refl-on-domain 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*: $(b,\ b) \in limit\ B\ r'$
    using *alt-b mem-Collect-eq singletonI*
    unfolding *above-def*
    by *metis*
  have *b-wins*: $\{a'.\ (b,\ a') \in limit\ B\ r'\} = \{b\}$
    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*: $\forall\ b' \in B.\ b \in above\ r'\ b'$
  using *subset-B-card b-in-r b-wins b-refl CollectI Product-Type.Collect-case-prodD*
    unfolding *above-def*
    by *fastforce*
  moreover have $b \in above\ r'\ a$
    using *a-pref-r-b pref-imp-in-above*
    by *metis*
  ultimately have *b-wins*: $\forall\ a' \in A'.\ b \in above\ r'\ a'$
    using *Diff-iff a empty-iff insert-iff*
    by (*metis* (*no-types*))
  hence $\forall\ a' \in A'.\ a' \in above\ r'\ b \longrightarrow a' = b$
    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-order-on $(B'' :: {'a}\ set)\ r'' \wedge B' \subseteq B''$
$\longrightarrow$ *linear-order-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-order-on* B (*limit* $B\ r'$) $\wedge$ *total-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-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
$\forall\ a'\ a''.$
$a' \in B \longrightarrow a'' \in B$
$\longrightarrow a' = a'' \vee (a',\ a'') \in limit\ B\ r' \vee (a'',\ a') \in limit\ B\ r'$

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
hence $\exists\ a \in A.\ above\ r\ a = \{a\}$
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 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
$a \preceq_r a$ **and**
$b \preceq_r b$
using *above-a above-b singletonI pref-imp-in-above*
by (*metis*, *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 $a \in A$
    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 $a \in$ *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**
    $a\ b :: 'a$
  assumes
    lin-ord: *linear-order-on A r* **and**
    fin-A: *finite A* **and**
    a-in-A: $a \in A$ **and**
    b-in-A: $b \in A$ **and**
    a-neq-b: $a \neq b$
  shows *rank r a* $\neq$ *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*

1.2.5 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**
$r\ r'$ :: $'a$ *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**
$r\ r'$ :: $'a$ *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
a-pref-b-rel: $(a, b) \in r'$ **and**
b-pref-a-rel: $(b, a) \in r$
by *simp-all*
have *antisym r*
using *assms lifted-imp-equiv-rel-except-a lin-imp-antisym*
unfolding *equiv-rel-except-a-def*
by *metis*
hence *imp-b-eq-a*: $(b, a) \in r \longrightarrow (a, b) \in r \longrightarrow b = a$
unfolding *antisym-def*
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 $\forall\ a' \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((a', b') \in r) = ((a', b') \in r')$
unfolding *equiv-rel-except-a-def*
by *simp*
moreover 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*
ultimately have $(b, c) \in r'$
using *b-in-A b-neq-a b-pref-a-rel c-eq-r-s-exc-a equiv-r-s-exc-a 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 r' :: $'a$ *Preference-Relation* **and**
    a a' :: $'a$
  assumes
    lifted: *lifted* A r r' a **and**
    a'*-pref-a*: $a' \preceq_r a$
  shows $a' \preceq_{r'} a$
proof (*unfold is-less-preferred-than.simps*)
  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 *rest-eq*: $\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((b, b') \in r) = ((b, b') \in r')$
    using *lifted*
    unfolding *lifted-def equiv-rel-except-a-def*
    by *simp*
  have *ex-lifted*: $\exists\ b \in A - \{a\}.\ (a, b) \in r \land (b, a) \in r'$
    using *lifted*
    unfolding *lifted-def*
    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 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 *lifted-r*: $\exists\ b \in A - \{a\}.\ (a, b) \in r \wedge (b, a) \in r'$
    unfolding *lifted-def*
    by *simp*
  from *assms*
  have *rest-eq*: $\forall\ b \in A - \{a\}.\ \forall\ b' \in A - \{a\}.\ ((b, b') \in r) = ((b, b') \in r')$
    unfolding *lifted-def equiv-rel-except-a-def*
    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**
    $r\ r' :: 'a$ *Preference-Relation* **and**
    $a\ a' :: 'a$
  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*)
  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 $\forall$ $b' \in A - \{a\}$. ($b' \in$ *above* r a') = ($b' \in$ *above* r' a')
using *a′-in-A-sub-a lifted-a*
unfolding *lifted-def equiv-rel-except-a-def above-def*
by *simp*
thus $b = a$
using *lifted-a b-not-in-above-s limited-dest lin-ord-imp-connex*
member-remove pref-imp-in-above b-in-above-r
unfolding *lifted-def equiv-rel-except-a-def remove-def connex-def*
by *metis*
qed

lemma *limit-lifted-imp-eq-or-lifted*:
fixes
A A' :: *'a set* **and**
r 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 $\exists$ $a' \in A - \{a\}$. $a \preceq_r a' \wedge a' \preceq_{r'} a$
thus *?thesis*
using *DiffD1 limit-presv-prefs 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
$A\ A'$ :: *'a set* **and**
$r\ r'$ :: *'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**
$r\ r'$ :: *'a Preference-Relation* **and**
$a\ a'$ :: *'a*
assumes
lifted-a: *lifted* $A\ r\ r'\ a$ **and**
a'-above-a': *above* $r\ a' = \{a'\}$ **and**
fin-A: *finite* A
shows *above* $r'\ a' = \{a'\} \lor$ *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 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 r' :: $'a$ *Preference-Relation* **and**
a 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 :: $'a$ **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

1.3 Norm

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

A norm on R to n is a mapping N: $R \mapsto n$ on R that has the following properties for all mappings u (and v) in R to n:

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

1.3.1 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*)

1.3.2 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*
thus *?case*
using *bij-betw-same-card card-0-eq sum.empty sum.infinite*
by *metis*
next
case (*Suc x*)
fix

A :: $'a$ *set* **and**
A' :: $'b$ *set* **and**
x :: *nat*
assume
suc-x: *Suc* x = *card* A **and**
bij-A-A': *bij-betw* f A A'
hence *card-A'-from-x*: *card* A' = *Suc* x
using *bij-betw-same-card*
by *metis*
have *x-lt-card-A*: x < *card* A
using *suc-x*
by *presburger*
obtain a :: $'a$ **where**
a-in-A: $a \in A$
using *suc-x card-eq-SucD insertI1*
by *metis*
hence *a-compl-A*: *insert* a $(A - \{a\}) = A$
using *insert-absorb*
by *simp*
hence
inj-on-A: *inj-on* f A **and**
img-of-A: $A' = f$ ‘ A
using *bij-A-A'*
unfolding *bij-betw-def*
by (*simp*, *simp*)
hence *inj-on* f (*insert* a A)
using *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-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*))
moreover have *card-without-a*: *card* $(A - \{a\}) = x$
using *suc-x a-in-A*
by *simp*
ultimately have *card-A'-sub-f-eq-x*: *card* $(A' - \{f\ a\}) = x$
using *bij-betw-same-card*
by *metis*
have $(\sum a \in A.\ g\ a) = (\sum a \in A - \{a\}.\ g\ a) + g\ a$
using *x-lt-card-A add.commute card-Diff1-less-iff card-without-a insert-Diff insert-Diff-single sum.insert-remove*
by (*metis* (*no-types*))
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 *bij-without-a a-in-A bij-A-A' bij-betw-imp-inj-on the-inv-into-f-f A'-sub-fa DiffD1 sum.reindex-cong*

by (*metis* (*mono-tags*, *lifting*))
finally show $(\sum a \in A.\ g\ a) = (\sum a' \in A'.\ g\ (\textit{the-inv-into}\ A\ f\ a'))$
using *add.commute card-Diff1-less-iff insert-Diff insert-Diff-single lessI sum.insert-remove card-A′-from-x card-A′-sub-f-eq-x*
by *metis*
qed

1.3.3 Common Norms

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

1.3.4 Properties

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

1.3.5 Theorems

theorem *l-one-is-sym*: *symmetry l-one*
proof (*unfold symmetry-def*, *safe*)
fix $l\ l'$ :: *ereal list*
assume *perm*: $l <^{\sim\sim}> l'$
then obtain π :: $\textit{nat} \Rightarrow \textit{nat}$
where
$\textit{perm}_\pi$: π *permutes* $\{..< \textit{length}\ l\}$ **and**
l_π: $\textit{permute-list}\ \pi\ l = l'$
using *mset-eq-permutation*
by *metis*
hence $(\sum i < \textit{length}\ l.\ |l'!i|) = (\sum i < \textit{length}\ l.\ |l!(\pi\ i)|)$
using *permute-list-nth*
by *fastforce*
also have $\ldots = (\sum i = 0 ..< \textit{length}\ l.\ |l!(\pi\ i)|)$
using *lessThan-atLeast0*
by *presburger*
also have $(\lambda\ i.\ |l!(\pi\ i)|) = ((\lambda\ i.\ |l!i|) \circ \pi)$
by *fastforce*
also have $(\sum y = 0 ..< \textit{length}\ l.\ ((\lambda\ i.\ |l!i|) \circ \pi)\ y) =$
$(\sum i = 0 ..< \textit{length}\ l.\ |l!i|)$
using $\textit{perm}_\pi$ *atLeast-upt set-upt sum.permute*
by *metis*
also have $\ldots = (\sum i < \textit{length}\ l.\ |l!i|)$
using *atLeast0LessThan*
by *presburger*
finally have $(\sum i < \textit{length}\ l.\ |l'!i|) = (\sum i < \textit{length}\ l.\ |l!i|)$
by *metis*
moreover have $\textit{length}\ l = \textit{length}\ l'$
using *perm perm-length*
by *metis*
ultimately show $\textit{l-one}\ l = \textit{l-one}\ l'$

using *l-one.elims*
by *metis*
qed

end

1.4 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.

1.4.1 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* :: $'r\ Result \Rightarrow bool$ **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**
$set\text{-}equals\text{-}partition\ X\ (e,\ r,\ d) = (e \cup r \cup d = X)$

1.4.2 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* ⇒ (*'r Result*) ⇒ *bool* **and**
    limit :: *'a set* ⇒ *'r set* ⇒ *'r set*
  assumes ∀ (*A* :: *'a set*) (*r* :: *'r Result*).
    (*set-equals-partition* (*limit A UNIV*) *r* ∧ *disjoint3 r*) ⟶ *well-formed A r*

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

fun (**in** *result*) *limit*$_{\mathcal{R}}$ :: *'a set* ⇒ *'r Result* ⇒ *'r Result* **where**
  limit$_{\mathcal{R}}$ *A* (*e*, *r*, *d*) = (*limit A e*, *limit A r*, *limit A d*)

abbreviation *elect-r* :: *'r Result* ⇒ *'r set* **where**
  elect-r r ≡ *fst r*

abbreviation *reject-r* :: *'r Result* ⇒ *'r set* **where**
  reject-r r ≡ *fst* (*snd r*)

abbreviation *defer-r* :: *'r Result* ⇒ *'r set* **where**
  defer-r r ≡ *snd* (*snd r*)

end

1.5 Preference Profile

theory *Profile*
  imports *Preference-Relation*
    Auxiliary-Lemmas
    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.

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

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

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

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

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

definition *finite-elections* :: $('a, 'v)$ *Election set* **where**
finite-elections $\equiv \{E :: ('a, 'v)$ *Election. finite-election* $E\}$

definition *well-formed-elections* :: $('a, 'v)$ *Election set* **where**
well-formed-elections $\equiv \{E.$ *profile* (*voters-$\mathcal{E}$* E) (*alternatives-$\mathcal{E}$* E) (*profile-$\mathcal{E}$* E)$\}$

— This function subsumes elections with fixed alternatives, finite voters, and a default value for the profile value on non-voters.
fun *elections-$\mathcal{A}$* :: $'a$ *set* $\Rightarrow$ $('a, 'v)$ *Election set* **where**
elections-$\mathcal{A}$ $A =$
well-formed-elections

$\cap$ {*E*. *alternatives-$\mathcal{E}$ E* = *A* $\wedge$ *finite* (*voters-$\mathcal{E}$ E*)
  $\wedge$ ($\forall$ *v*. *v* $\notin$ *voters-$\mathcal{E}$ E* $\longrightarrow$ *profile-$\mathcal{E}$ E v* = {})}

— Here, we count the occurrences of a ballot in an election, i.e., how many voters specifically chose that exact ballot.

fun *vote-count* :: *'a Preference-Relation* $\Rightarrow$ (*'a*, *'v*) *Election* $\Rightarrow$ *nat* **where**
  vote-count p E = *card* {*v* $\in$ (*voters-$\mathcal{E}$ E*). (*profile-$\mathcal{E}$ E*) *v* = *p*}

1.5.2 Vote Count

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

by *blast*
ultimately have
$\mathit{sum\ card}\ \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \mid p.\ p \in \mathit{UNIV}\} =$
$\mathit{sum\ card}\ \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$+\ \mathit{sum\ card}\ \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} = \{\}\}$
using *sum.union-disjoint*[*of*
$\{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} = \{\}\}]$
by *simp*
moreover have
$\forall\ X \in \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} = \{\}\}.$
$\mathit{card}\ X = \mathit{0}$
using *card-eq-0-iff*
by *fastforce*
ultimately have *card-eq-sum*:
$\mathit{card}\ (\mathit{voters}\text{-}\mathcal{E}\ E) =$
$\mathit{sum\ card}\ \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
using *card-eq-sum′*
by *simp*
have
$\mathit{inj\text{-}on}\ (\lambda\ p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\})$
$\{p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
unfolding *inj-on-def*
by *blast*
moreover have
$(\lambda\ p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\})$
$`\ \{p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\subseteq \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
by *blast*
moreover have
$(\lambda\ p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\})$
$`\ \{p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\supseteq \{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
by *blast*
ultimately have
$\mathit{bij\text{-}betw}\ (\lambda\ p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\})$
$\{p.\ \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
$\{\{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\}$
$\mid p.\ p \in \mathit{UNIV} \land \{v \in \mathit{voters}\text{-}\mathcal{E}\ E.\ \mathit{profile}\text{-}\mathcal{E}\ E\ v = p\} \neq \{\}\}$
unfolding *bij-betw-def*
by *simp*

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

1.5.3 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 $\pi\ (A,\ V,\ p) = (A,\ \pi\ `\ V,\ p \circ (\textit{the-inv}\ \pi))$

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: $(A,\ V',\ q) = \mathit{rename}\ \pi\ (A,\ V,\ p)$ **and**
bij-perm: *bij* π
shows *profile* $V'\ A\ q$
proof (*unfold profile-def*, *safe*)
fix v' :: $'v$
assume $v' \in V'$
moreover have $V' = \pi$ ‘ V
using *renamed*
by *simp*
ultimately have $((\mathit{the\text{-}inv}\ \pi)\ v') \in V$
using *UNIV-I bij-perm bij-is-inj bij-is-surj*
f-the-inv-into-f inj-image-mem-iff
by *metis*
thus *linear-order-on* $A\ (q\ v')$
using *renamed bij-perm prof*
unfolding *profile-def*
by *simp*
qed

lemma *rename-prof*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a,\ 'v)$ *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
profile $V\ A\ p$ **and**
$(A,\ V',\ q) = \mathit{rename}\ \pi\ (A,\ V,\ p)$ **and**
bij π
shows *profile* $V'\ A\ q$
using *assms rename-sound*
by *metis*

lemma *rename-finite*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a,\ 'v)$ *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
finite V **and**
$(A,\ V',\ q) = \mathit{rename}\ \pi\ (A,\ V,\ p)$ **and**
bij π

shows *finite* V'
using *assms*
by *simp*

lemma *rename-inv*:
fixes
π :: $'v \Rightarrow 'v$ **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) $(A, V, p)) =$
$(A, \pi$ ‘ (*the-inv* π) ‘ V, $p \circ$ (*the-inv* (*the-inv* π)) $\circ$ (*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)) $= \pi$
using *assms surj-def inj-on-the-inv-into surj-imp-inv-eq the-inv-f-f*
unfolding *bij-betw-def*
by (*metis* (*mono-tags*, *opaque-lifting*))
moreover have $\pi \circ$ (*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 :: $'v \Rightarrow 'v$
assumes *bij* π
shows *inj* (*rename* π)
proof (*unfold inj-def split-paired-All rename.simps*, *safe*)
fix
$A\ A'$ :: $'a$ *set* **and**
$V\ V'$ :: $'v$ *set* **and**
$p\ p'$ :: $('a, 'v)$ *Profile* **and**
v :: $'v$
assume
$p \circ$ *the-inv* $\pi = p' \circ$ *the-inv* π **and**
π ‘ $V = \pi$ ‘ V'
thus
$v \in V \Longrightarrow v \in V'$ **and**
$v \in V' \Longrightarrow v \in V$ **and**
$p = p'$
using *assms*
by (*metis bij-betw-imp-inj-on inj-image-eq-iff*,

metis bij-betw-imp-inj-on inj-image-eq-iff,
metis bij-betw-the-inv-into bij-is-surj surj-fun-eq)
qed

lemma *rename-surj*:
fixes π :: $'v \Rightarrow 'v$
assumes *bij* π
shows
rename π ' *well-formed-elections* = *well-formed-elections* **and**
rename π ' *finite-elections* = *finite-elections*
proof (*safe*)
fix
A A' :: $'a$ *set* **and**
V V' :: $'v$ *set* **and**
p p' :: ($'a$, $'v$) *Profile*
assume *wf*: (A, V, p) $\in$ *well-formed-elections*
hence *rename* (*the-inv* π) (A, V, p) $\in$ *well-formed-elections*
using *assms bij-betw-the-inv-into rename-sound*
unfolding *well-formed-elections-def*
by *fastforce*
thus (A, V, p) $\in$ *rename* π ' *well-formed-elections*
using *assms image-eqI rename-inv*
by *metis*
assume (A', V', p') = *rename* π (A, V, p)
thus (A', V', p') $\in$ *well-formed-elections*
using *rename-sound wf assms*
unfolding *well-formed-elections-def*
by *fastforce*
next
fix
A A' :: $'b$ *set* **and**
V V' :: $'v$ *set* **and**
p p' :: ($'b$, $'v$) *Profile*
assume *finite*: (A, V, p) $\in$ *finite-elections*
hence *rename* (*the-inv* π) (A, V, p) $\in$ *finite-elections*
using *assms bij-betw-the-inv-into rename-prof rename-finite*
unfolding *finite-elections-def*
by *fastforce*
thus (A, V, p) $\in$ *rename* π ' *finite-elections*
using *assms image-eqI rename-inv*
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

1.5.4 List Representation

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 *map-helper*:
  fixes
    f :: $'x \Rightarrow 'y \Rightarrow 'z$ **and**
    g :: $'x \Rightarrow 'x$ **and**
    h :: $'y \Rightarrow 'y$ **and**
    l :: $'x$ *list* **and**
    l' :: $'y$ *list*
  shows *map2* f $($*map* g $l)$ $($*map* h $l')$ $=$ *map2* $(\lambda$ x $y.$ f $(g$ $x)$ $(h$ $y))$ l l'
proof $-$
  have *map2* f $($*map* g $l)$ $($*map* h $l')$ $=$
      map $(\lambda$ $(x,$ $y).$ f x $y)$ $($*map* $(\lambda$ $(x,$ $y).$ $(g$ $x,$ h $y))$ $($*zip* l $l'))$
    using *zip-map-map*
    by *metis*
  thus *?thesis*
    by *force*
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* $V)!i)$
  using *assms*
  by *force*

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 $p)$
  by *simp*

lemma *set-card-upper-bound*:
  fixes
    i :: *nat* **and**
    V :: *nat set*

assumes
fin-V: *finite* V **and**
bound-v: $\forall\ v \in V.\ v < i$
shows *card* $V \leq i$
proof (*cases* $V = \{\}$)
case *True*
thus *?thesis*
by *simp*
next
case *False*
hence *Max* $V \in V$
using *fin-V*
by *simp*
thus *?thesis*
using *assms Suc-leI card-le-Suc-Max order-trans*
by *metis*
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* $V!(\mathit{card}\ \{v \in V.\ v < x\}) = x$
proof −
let $?c = \mathit{card}\ \{v \in V.\ v < x\}$ **and**
$?set = \{v \in V.\ v < x\}$
have $\forall\ v \in V.\ \exists\ n.\ n < \mathit{card}\ V \wedge (\mathit{sorted\text{-}list\text{-}of\text{-}set}\ V!n) = v$
using *length-sorted-list-of-set sorted-list-of-set-unique in-set-conv-nth fin-V*
by *metis*
then obtain φ :: $'v \Rightarrow \mathit{nat}$ **where**
index-φ: $\forall\ v \in V.\ \varphi\ v < \mathit{card}\ V \wedge (\mathit{sorted\text{-}list\text{-}of\text{-}set}\ V!(\varphi\ v)) = v$
by *metis*
— $\varphi\ x = ?c$, i.e., $\varphi\ x \geq ?c$ and $\varphi\ x \leq ?c$
let $?i = \varphi\ x$
have *inj-φ*: *inj-on* $\varphi\ V$
using *inj-onI index-φ*
by *metis*
have $\forall\ v \in V.\ \forall\ v' \in V.\ v < v' \longrightarrow \varphi\ v < \varphi\ v'$
using *leD linorder-le-less-linear sorted-list-of-set-unique*
sorted-sorted-list-of-set sorted-nth-mono fin-V index-φ
by *metis*
hence $\forall\ j \in \{\varphi\ v \mid v.\ v \in ?set\}.\ j < ?i$
using *x-V*
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*
hence *geq*: $?c \leq\ ?i$
using *inj-φ*
by (*simp add*: *card-image inj-on-subset setcompr-eq-image*)
have *sorted-φ*:
$\forall\ i < card\ V.\ \forall\ j < card\ V.\ i < j$
$\longrightarrow (sorted\text{-}list\text{-}of\text{-}set\ V!i) < (sorted\text{-}list\text{-}of\text{-}set\ V!j)$
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 $?c <\ ?i$
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 geq index-φ x-V fin-V set-sorted-list-of-set length-sorted-list-of-set nth-mem order.strict-trans1*
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 $\ldots = (\bigcup\ j \in \{0\ ..< (?c + 1)\}.\ \{sorted\text{-}list\text{-}of\text{-}set\ V!j\})$
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 $\forall\ j \in \{0\ ..< (?c + 1)\}.\ card\ (?A\ j) = 1$
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*
hence $card\ (\bigcup\ j \in \{0\ ..< (?c + 1)\}.\ ?A\ j) =\ ?c + 1$

by *simp*
hence *?c + 1 ≤ ?c*
using *subset card-mono fin-img*
by (*metis* (*no-types*, *lifting*))
thus *False*
by *simp*
next
case *False*
thus *False*
using *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* ⇒ *′v* **and**
V :: *′v set* **and**
p :: (*′a*, *′v*) *Profile*
assumes *bij π*
shows
let $\varphi = \lambda\ i.\ card\ \{v \in \pi\ \text{`}\ V.\ v < \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i)\}$
in $(to\text{-}list\ V\ p) = permute\text{-}list\ \varphi\ (to\text{-}list\ (\pi\ \text{`}\ V)\ (\lambda\ x.\ p\ (the\text{-}inv\ \pi\ x)))$
proof (*cases finite V*)
case *False*
— If *V* is infinite, both lists are empty.
hence *to-list V p* = []
by *simp*
moreover have *infinite* (π ‘ *V*)
using *False assms bij-betw-finite bij-betw-subset top-greatest*
by *metis*
hence *to-list* (π ‘ *V*) (λ *x*. *p* (*the-inv* π *x*)) = []
by *simp*
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* ∈ π ‘ *V*. *v* < π ((*sorted-list-of-set V*)!*i*)}
— These are auxiliary statements equating everything with *?n*.
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* = . . .
  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*)
  — The lists have equal lengths.
  show
    length (*to-list V p*) =
      length (*map*
        (λ *i*. *to-list ?img ?q*!(*card* {*v* $\in$ *?img*.
          v < π (*sorted-list-of-set V*!*i*)}))
        [*0* ..< *length* (*to-list ?img ?q*)])
    using *eq-length*
    by *simp*
next
  — The *i*th entries of the lists coincide.
  fix *i* :: *nat*
  assume *in-bnds*: *i* < *?n*
  let *?c* = *card* {*v* $\in$ *?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 $\forall$ *v*. *v* $\in$ *?img* $\longrightarrow$ {*v*′ $\in$ *?img*. *v*′ < *v*} $\subseteq$ *?img* − {*v*}
      by *blast*
    moreover have *elem-of-img*: π (*sorted-list-of-set V*!*i*) $\in$ *?img*
      using *True in-bnds image-eqI nth-mem card-length-V*
        length-sorted-list-of-set set-sorted-list-of-set
      by *metis*
    ultimately have
      {*v* $\in$ *?img*. *v* < π (*sorted-list-of-set V*!*i*)}
    $\subseteq$ *?img* − {π (*sorted-list-of-set V*!*i*)}
      by *simp*
    hence {*v* $\in$ *?img*. *v* < π (*sorted-list-of-set V*!*i*)} $\subset$ *?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*:
. . . = *?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*:
. . . = *?q* (π (*sorted-list-of-set V*!*i*))
using *True elem-of-img*
by (*simp add*: *sorted-list-of-set-nth-equals-card*)
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* $\in$ *?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

1.5.5 Preference Counts

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* $\Rightarrow$ (*'a*, *'v*) *Profile* $\Rightarrow$ *'a* $\Rightarrow$ *enat* **where**
win-count V p a = (*if finite V*
then card {*v* $\in$ *V*. *above* (*p v*) *a* = {*a*}} *else* ∞)

fun *prefer-count* :: *'v set* $\Rightarrow$ (*'a*, *'v*) *Profile* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *enat* **where**
prefer-count V p x y = (*if finite V*
then card {*v* $\in$ *V*. *let r* = (*p v*) *in* ($y \preceq_r x$)} *else* ∞)

lemma *pref-count-voter-set-card*:
fixes
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
a b :: *'a*
assumes *finite V*
shows *prefer-count V p a b* $\leq$ *card V*
using *assms*
by (*simp add*: *card-mono*)

lemma *set-compr*:

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

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

lemma *pref-count*:
fixes
$A :: {}'a\ set$ **and**
$V :: {}'v\ set$ **and**
$p :: ({}'a, {}'v)\ \mathit{Profile}$ **and**
$a\ b :: {}'a$
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 $\mathit{prefer\text{-}count}\ V\ p\ a\ b = \mathit{card}\ V - (\mathit{prefer\text{-}count}\ V\ p\ b\ a)$
proof −
have $\forall\ v \in V.\ \mathit{let}\ r = (p\ v)\ \mathit{in}\ (\neg\ b \preceq_r a \longrightarrow a \preceq_r b)$
using *a-in-A b-in-A prof lin-ord-imp-connex*
unfolding *profile-def connex-def*
by *metis*
moreover 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*
ultimately have
$\{v \in V.\ (\mathit{let}\ r = (p\ v)\ \mathit{in}\ (b \preceq_r a))\} =$
$V - \{v \in V.\ (\mathit{let}\ r = (p\ v)\ \mathit{in}\ (a \preceq_r b))\}$
by *auto*
thus *?thesis*
by (*simp add*: *card-Diff-subset Collect-mono fin*)
qed

lemma *pref-count-sym*:
fixes
$p :: ({}'a, {}'v)\ \mathit{Profile}$ **and**

V :: $'v$ *set* **and**
$a\ b\ c$ :: $'a$
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 finite* V)
case *True*
moreover have
prefer-count $V\ p\ c\ a \in \mathbb{N}$ **and**
prefer-count $V\ p\ b\ c \in \mathbb{N}$
unfolding *Nats-def*
using *True of-nat-eq-enat*
by (*simp*, *simp*)
moreover have *prefer-count* $V\ p\ c\ a \leq$ *card* V
using *True prof pref-count-voter-set-card*
by *metis*
moreover have
prefer-count $V\ p\ a\ c =$ *card* $V -$ (*prefer-count* $V\ p\ c\ a$) **and**
prefer-count $V\ p\ c\ b =$ *card* $V -$ (*prefer-count* $V\ p\ b\ c$)
using *True pref-count prof c-in-A*
by (*metis* (*no-types*, *opaque-lifting*) *a-in-A a-neq-c*,
metis (*no-types*, *opaque-lifting*) *b-in-A c-neq-b*)
hence *card* $V -$ (*prefer-count* $V\ p\ b\ c$) + (*prefer-count* $V\ p\ c\ a$)
$\leq$ *card* $V -$ (*prefer-count* $V\ p\ c\ a$) + (*prefer-count* $V\ p\ c\ a$)
using *pref-count-ineq*
by *simp*
ultimately show *?thesis*
by *simp*
next
case *False*
thus *?thesis*
by *simp*
qed

lemma *empty-prof-imp-zero-pref-count*:
fixes
p :: $('a, 'v)$ *Profile* **and**
V :: $'v$ *set* **and**
$a\ b$ :: $'a$
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 b :: *'a* **and**
V :: *'v set*
assumes *infinite V*
shows ¬ *wins V b p a*
using *assms*
by *simp*

Having alternative a win against b implies that b does not win against a.

lemma *wins-antisym*:
fixes
p :: (*'a*, *'v*) *Profile* **and**
a b :: *'a* **and**
V :: *'v set*
assumes *wins V a p b* — This already implies that V is finite.
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*

1.5.6 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* ∈ *A* ∧ (∀ *x* ∈ *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 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*: $b \neq a$
hence *wins* V b p a
using *insert-Diff insert-iff assms*
by *simp*
hence $\neg$ *wins* V a p b
by (*simp add*: *wins-antisym*)
moreover have *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'*:
fixes
V :: $'v$ *set* **and**
A :: $'a$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a b :: $'a$
assumes

condorcet-winner V A p a **and**
b ≠ *a*
shows ¬ *condorcet-winner V A p b*
using *cond-winner-unique-eq assms*
by *metis*

1.5.7 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 B 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 B :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
profile V B p **and**
A ⊆ *B*
shows *profile V A* (*limit-profile A p*)
proof (*unfold profile-def*)
have ∀ *v* ∈ *V*. *linear-order-on A* (*limit A* (*p v*))
using *assms limit-presv-lin-ord*
unfolding *profile-def*
by *metis*
thus ∀ *v* ∈ *V*. *linear-order-on A* ((*limit-profile A p*) *v*)
by *simp*
qed

1.5.8 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* $\Rightarrow$ $('a,\ 'v)$ *Profile* $\Rightarrow$
$('a,\ 'v)$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
lifted $V\ A\ p\ p'\ a \equiv$
finite-profile $V\ A\ p$ $\wedge$ *finite-profile* $V\ A\ p'$ $\wedge\ a \in A$
$\wedge\ (\forall\ v \in V.\ \neg$ *Preference-Relation.lifted* $A\ (p\ v)\ (p'\ v)\ a \longrightarrow (p\ v) = (p'\ v))$
$\wedge\ (\exists\ v \in V.$ *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**
$p\ p'$ :: $('a,\ 'v)$ *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*)
show
profile $V\ A\ p$ **and**
profile $V\ A\ p'$ **and**
$a \in A$
using *assms*
unfolding *lifted-def*
by (*metis*, *metis*, *metis*)
next
fix v :: $'v$
assume $v \in V$
thus *equiv-rel-except-a* $A\ (p\ v)\ (p'\ v)\ a$
using *assms 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
$A\ A'$ :: $'a$ *set* **and**
V :: $'v$ *set* **and**
$p\ p'$ :: $('a,\ 'v)$ *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
— With the current definitions of *equiv-prof-except-a* and *limit-prof*, we can only conclude that the limited profiles coincide on the given voter set, since *limit-prof*

may change the profiles everywhere, while *equiv-prof-except-a* only makes statements about the voter set.

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*
  thus *limit-profile* A p v = *limit-profile* A q v
    using *subset not-in-A negl-diff-imp-eq-limit*
    by *simp*
qed

lemma *limit-prof-eq-or-lifted*:
  fixes
    A A' :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p 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* $a \in A$)
  case *True*
  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* $\forall$ $v \in V$. *limit* A (p v) = *limit* A (p' v))
    case *True*
    thus *?thesis*
      by *simp*
  next
    case *False*
    let *?p* = *limit-profile* A p
    let *?q* = *limit-profile* A p'
    have
      profile V A *?p* **and**
      profile V A *?q*
      using *lifted-a subset limit-profile-sound*

```
      unfolding lifted-def
      by (safe, safe)
    moreover have
      ∃ v ∈ V. Preference-Relation.lifted A (?p v) (?q v) a
      using False one
      unfolding limit-profile.simps
      by (metis (no-types, lifting))
    ultimately have lifted V A ?p ?q a
      using True lifted-a one rev-finite-subset subset
      unfolding lifted-def limit-profile.simps
      by (metis (no-types, lifting))
    thus ?thesis
      by simp
  qed
next
  case False
  thus ?thesis
    using lifted-a negl-diff-imp-eq-limit-prof subset lifted-imp-equiv-prof-except-a
    by metis
qed

end
```

1.6 Social Choice Result

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

1.6.1 Definition

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

```
fun well-formed-SCF :: 'a set ⇒ 'a Result ⇒ bool where
  well-formed-SCF A res = (disjoint3 res ∧ set-equals-partition A res)

fun limit-SCF :: 'a set ⇒ 'a set ⇒ 'a set where
  limit-SCF A r = A ∩ r
```

1.6.2 Auxiliary Lemmas

```
lemma result-imp-rej:
  fixes A e r d :: 'a set
  assumes well-formed-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 $A\ e\ r\ d :: 'a\ set$
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-r r* ⊆ *A*
proof (*safe*)
fix *a* :: *'a*
assume *a* ∈ *defer-r r*
moreover obtain
f :: *'a Result* ⇒ *'a set* ⇒ *'a set* **and**
g :: *'a Result* ⇒ *'a set* ⇒ *'a Result* **where**
A = *f r A* ∧ *r* = *g r A* ∧ *disjoint3* (*g r A*) ∧ *set-equals-partition* (*f r A*) (*g r A*)
using *assms*
by *simp*
moreover have
∀ *p*. ∃ *e r d*. *set-equals-partition A p* ⟶ (*e*, *r*, *d*) = *p* ∧ *e* ∪ *r* ∪ *d* = *A*
by *simp*
ultimately show *a* ∈ *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* ⊆ *A*
proof (*safe*)
fix *a* :: *'a*
assume *a* ∈ *elect-r r*
moreover obtain
f :: *'a Result* ⇒ *'a set* ⇒ *'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
$\forall\ p.\ \exists\ e\ r\ d.$ *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

1.7 Social Welfare Result

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

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

fun *limit-$\mathcal{SWF}$* :: *'a set* ⇒ (*'a Preference-Relation*) *set* ⇒
(*'a Preference-Relation*) *set* **where**
limit-$\mathcal{SWF}$ A res = {*limit A r* | *r*. *r* ∈ *res* ∧ *linear-order-on A* (*limit A r*)}

end

1.8 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-$\mathcal{SCF}$*
proof (*unfold-locales*, *safe*)
fix *A e r d* :: *'a set*
assume
set-equals-partition (*limit-$\mathcal{SCF}$ A UNIV*) (*e*, *r*, *d*) **and**
disjoint3 (*e*, *r*, *d*)
thus *well-formed-$\mathcal{SCF}$ A* (*e*, *r*, *d*)
by *simp*
qed

Results from committee functions, for the purpose of composability and modularity given as three sets of (potentially tied) sets of alternatives or committees. [[*Not actually used yet.*]]

global-interpretation *committee-result*: *result*
λ A r. *set-equals-partition* (*Pow A*) *r* ∧ *disjoint3 r*
λ A rs. {*r* ∩ *A* | *r*. *r* ∈ *rs*}
proof (*unfold-locales*, *safe*)
fix
A :: *'b set* **and**

```
    e r d :: 'b set set
  assume set-equals-partition {r ∩ A |r. r ∈ UNIV} (e, r, d)
  thus set-equals-partition (Pow A) (e, r, d)
    by force
qed
```

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

```
global-interpretation SWF-result: result well-formed-SWF limit-SWF
proof (unfold-locales, safe)
  fix
    A :: 'a set and
    e r d :: ('a Preference-Relation) set
  assume
    set-equals-partition (limit-SWF A UNIV) (e, r, d) and
    disjoint3 (e, r, d)
  moreover have
    limit-SWF A UNIV = {limit A r' | r'. linear-order-on A (limit A r')}
    by simp
  moreover have ... = {r'. linear-order-on A r'}
  proof (safe)
    fix r' :: 'a Preference-Relation
    assume lin-ord: linear-order-on A r'
    hence ∀ (a, b) ∈ r'. (a, b) ∈ limit A r'
      unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
      by force
    hence r' = limit A r'
      by force
    thus ∃ x. r' = limit A x ∧ linear-order-on A (limit A x)
      using lin-ord
      by metis
  qed
  ultimately show well-formed-SWF A (e, r, d)
    by simp
qed

setup Locale-Code.close-block

end
```

1.9 Symmetry Properties of Functions

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

begin

1.9.1 Functions

type-synonym $('x, 'y)$ *binary-fun* $= 'x \Rightarrow 'y \Rightarrow 'y$

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

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

1.9.2 Relations for Symmetry Constructions

fun *restricted-rel* $:: 'x\ rel \Rightarrow 'x\ set \Rightarrow 'x\ set \Rightarrow 'x\ rel$ **where**
restricted-rel $r\ S\ S' = r \cap (S \times S')$

fun *closed-restricted-rel* $:: 'x\ rel \Rightarrow 'x\ set \Rightarrow 'x\ set \Rightarrow bool$ **where**
closed-restricted-rel $r\ S\ T = ((restricted\text{-}rel\ r\ T\ S)\ ``\ T \subseteq T)$

fun *action-induced-rel* $:: 'x\ set \Rightarrow 'y\ set \Rightarrow ('x, 'y)\ binary\text{-}fun \Rightarrow 'y\ rel$ **where**
action-induced-rel $S\ T\ \varphi = \{(y, y').\ y \in T \wedge (\exists\ x \in S.\ \varphi\ x\ y = y')\}$

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

fun *equivariance* $:: 'x\ set \Rightarrow 'y\ set \Rightarrow ('x, 'y)\ binary\text{-}fun \Rightarrow ('y * 'y)\ rel$ **where**
equivariance $S\ T\ \varphi =$
$\{((u, v), (x, y)).\ (u, v) \in T \times T \wedge (\exists\ z \in S.\ x = \varphi\ z\ u \wedge y = \varphi\ z\ v)\}$

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

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

1.9.3 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 $('x, 'y)$ *symmetry* $=$
Invariance $'x\ rel \mid$
Equivariance $'x\ set\ (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ set$

fun *is-symmetry* $:: ('x \Rightarrow 'y) \Rightarrow ('x, 'y)\ symmetry \Rightarrow bool$ **where**
is-symmetry $f\ (Invariance\ r) = (\forall\ x.\ \forall\ y.\ (x, y) \in r \longrightarrow f\ x = f\ y) \mid$
is-symmetry $f\ (Equivariance\ s\ \tau) = (\forall\ (\varphi, \psi) \in \tau.\ \forall\ x \in s.\ f\ (\varphi\ x) = \psi\ (f\ x))$

definition *action-induced-equivariance* :: $'z$ *set* $\Rightarrow$ $'x$ *set* $\Rightarrow$ $('z, 'x)$ *binary-fun* $\Rightarrow$
    $('z, 'y)$ *binary-fun* $\Rightarrow$ $('x, 'y)$ *symmetry* **where**
  action-induced-equivariance $T\ S\ \varphi\ \psi \equiv$ *Equivariance* $S\ \{(\varphi\ z,\ \psi\ z) \mid z.\ z \in T\}$

1.9.4 Auxiliary Lemmas

lemma *un-left-inv-singleton-set-system*: $\bigcup \circ$ *singleton-set-system* $=$ *id*
proof
  fix s :: $'x$ *set*
  have $(\bigcup \circ$ *singleton-set-system*$)$ $s = \{x.\ \{x\} \in$ *singleton-set-system* $s\}$
    by *force*
  thus $(\bigcup \circ$ *singleton-set-system*$)$ $s =$ *id* s
    by *simp*
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

1.9.5 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 *is-symmetry* f (*Invariance* (*action-induced-rel* $T\ S\ \varphi$)) $=$
      is-symmetry f (*action-induced-equivariance* $T\ S\ \varphi$ $(\lambda\ g.\ id)$)
proof (*unfold action-induced-equivariance-def is-symmetry.simps*, *safe*)

fix
  x :: $'x$ **and**
  g :: $'z$
assume
  $x \in S$ **and**
  $g \in T$ **and**
  $\forall\ x\ y.\ (x,\ y) \in$ *action-induced-rel* $T\ S\ \varphi \longrightarrow f\ x = f\ y$
moreover with *this* **have** $(x,\ \varphi\ g\ x) \in$ *action-induced-rel* $T\ S\ \varphi$
  unfolding *action-induced-rel.simps*
  by *blast*
ultimately show $f\ (\varphi\ g\ x) = id\ (f\ x)$
  by *simp*
next
fix $x\ y$ :: $'x$
assume
  equivar:
    $\forall\ (\varphi,\ \psi) \in \{(\varphi\ g,\ id)\ |g.\ g \in T\}.\ \forall\ x \in S.\ f\ (\varphi\ x) = \psi\ (f\ x)$ **and**
  rel: $(x,\ y) \in$ *action-induced-rel* $T\ S\ \varphi$
then obtain g :: $'z$ **where**
  img: $\varphi\ g\ x = y$ **and**
  elt: $g \in T$
  unfolding *action-induced-rel.simps*
  by *blast*
moreover have $x \in S$
  using *rel*
  by *simp*
ultimately have $f\ (\varphi\ g\ x) = id\ (f\ x)$
  using *equivar elt*
  by *blast*
thus $f\ x = f\ y$
  using *img elt*
  by *simp*
qed

lemma *rewrite-invar-ind-by-act*:
fixes
  f :: $'x \Rightarrow\ 'y$ **and**
  S :: $'z$ *set* **and**
  T :: $'x$ *set* **and**
  φ :: $('z,\ 'x)$ *binary-fun*
shows *is-symmetry* f (*Invariance* (*action-induced-rel* $S\ T\ \varphi$)) $=$
    $(\forall\ x \in S.\ \forall\ y \in T.\ f\ y = f\ (\varphi\ x\ y))$
proof (*safe*)
fix
  y :: $'x$ **and**
  x :: $'z$
assume
  is-symmetry f (*Invariance* (*action-induced-rel* $S\ T\ \varphi$)) **and**
  $y \in T$ **and**

$x \in S$
moreover from *this* **have** $(y, \varphi\ x\ y) \in$ *action-induced-rel* $S\ T\ \varphi$
unfolding *action-induced-rel.simps*
by *blast*
ultimately show $f\ y = f\ (\varphi\ x\ y)$
by *simp*
next
assume $\forall\ x \in S.\ \forall\ y \in T.\ f\ y = f\ (\varphi\ x\ y)$
moreover have
$\forall\ (x, y) \in$ *action-induced-rel* $S\ T\ \varphi.\ x \in T \wedge (\exists\ z \in S.\ y = \varphi\ z\ x)$
by *auto*
ultimately show *is-symmetry* f (*Invariance* (*action-induced-rel* $S\ T\ \varphi$))
by *auto*
qed

lemma *rewrite-equivariance*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$S :: 'z$ *set* **and**
$T :: 'x$ *set* **and**
$\varphi :: ('z, 'x)$ *binary-fun* **and**
$\psi :: ('z, 'y)$ *binary-fun*
shows *is-symmetry* f (*action-induced-equivariance* $S\ T\ \varphi\ \psi$) =
$(\forall\ x \in S.\ \forall\ y \in T.\ f\ (\varphi\ x\ y) = \psi\ x\ (f\ y))$
unfolding *action-induced-equivariance-def*
by *auto*

lemma *rewrite-group-action-img*:
fixes
$m :: 'x$ *monoid* **and**
$S\ T :: 'y$ *set* **and**
$\varphi :: ('x, 'y)$ *binary-fun* **and**
$x\ y :: 'x$
assumes
$T \subseteq S$ **and**
$x \in$ *carrier* m **and**
$y \in$ *carrier* m **and**
group-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
$f\ g$ :: $'a \Rightarrow 'a$ **and**
S :: $'a$ *set*
assumes
$f \in$ *carrier* (*BijGroup S*) **and**
$g \in$ *carrier* (*BijGroup S*)
shows
rewrite-mult: $f \otimes_{BijGroup\ S} g$ = *extensional-continuation* ($f \circ g$) S **and**
rewrite-mult-univ: $S = UNIV \longrightarrow f \otimes_{BijGroup\ S} g = f \circ g$
using *assms*
unfolding *BijGroup-def compose-def comp-def restrict-def*
by (*simp*, *fastforce*)

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**
$S\ T$ :: $'a$ *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

φ ψ :: ('a, 'b) *binary-fun* **and**
S :: 'a *set* **and**
T U :: 'b *set*
assumes
$U \subseteq T$ **and**
$\forall\ x \in S.\ \forall\ y \in U.\ \psi\ x\ y = \varphi\ x\ y$
shows *action-induced-rel* S U ψ = *restricted-rel* (*action-induced-rel* S T φ) U *UNIV*
proof (*unfold action-induced-rel.simps restricted-rel.simps*, *safe*)
fix x :: 'b
assume $x \in U$
thus $x \in T$
using *assms*
by *blast*
next
fix
g :: 'a **and**
x :: 'b
assume
g-in-S: $g \in S$ **and**
x-in-U: $x \in U$
hence $\varphi\ g\ x = \psi\ g\ x$
using *assms*
by *simp*
thus $\exists\ g' \in S.\ \varphi\ g'\ x = \psi\ g\ x$
using *g-in-S*
by *blast*
next
fix
g :: 'a **and**
x :: 'b
show $\psi\ g\ x \in$ *UNIV*
by *blast*
next
fix
g :: 'a **and**
x :: 'b
assume
g-in-S: $g \in S$ **and**
x-in-U: $x \in U$
hence $\psi\ g\ x = \varphi\ g\ x$
using *assms*
by *simp*
thus $\exists\ g' \in S.\ \psi\ g'\ x = \varphi\ g\ x$
using *g-in-S*
by *blast*
qed

lemma *coinciding-actions-ind-equal-rel*:

fixes
  S :: *'x set* **and**
  T :: *'y set* **and**
  $\varphi\ \psi$ :: $('x,\ 'y)$ *binary-fun*
assumes $\forall\ x \in S.\ \forall\ y \in T.\ \varphi\ x\ y = \psi\ x\ y$
shows *action-induced-rel* $S\ T\ \varphi$ = *action-induced-rel* $S\ T\ \psi$
unfolding *extensional-continuation.simps*
using *assms*
by *auto*

1.9.6 Group Actions

lemma *const-id-is-group-action*:
  fixes m :: *'x monoid*
  assumes *group m*
  shows *group-action m UNIV* $(\lambda\ x.\ id)$
  using *assms*
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*, *safe*)
  show *group* (*BijGroup UNIV*)
    using *group-BijGroup*
    by *metis*
next
  show *id* $\in$ *carrier* (*BijGroup UNIV*)
    unfolding *BijGroup-def Bij-def*
    by *simp*
  thus $id = id \otimes_{\mathit{BijGroup\ UNIV}} id$
    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* $\equiv$ $(\lambda\ x.$ *extensional-continuation* (*image* $(\varphi\ x)$) (*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 *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 y* :: *'x*
note
‹*x* $\in$ *carrier m* $\Longrightarrow$ φ*-img x* $\in$ *carrier* (*BijGroup* (*Pow S*))› **and**
‹*y* $\in$ *carrier m* $\Longrightarrow$ φ*-img y* $\in$ *carrier* (*BijGroup* (*Pow S*))›
moreover assume
carrier-x: *x* $\in$ *carrier m* **and**
carrier-y: *y* $\in$ *carrier m*
ultimately have
carrier-election-x: φ*-img x* $\in$ *carrier* (*BijGroup* (*Pow S*)) **and**
carrier-election-y: φ*-img y* $\in$ *carrier* (*BijGroup* (*Pow S*))
by (*presburger*, *presburger*)
hence *h-closed*: $\forall$ *T* $\in$ *Pow S*. φ*-img y T* $\in$ *Pow S*
using *bij-betw-apply Int-Collect*
unfolding *BijGroup-def Bij-def partial-object.select-convs*
by (*metis* (*no-types*))
from *carrier-election-x carrier-election-y*
have φ*-img x* $\otimes_{\mathit{BijGroup\ (Pow\ S)}}$ φ*-img y* =
extensional-continuation (φ*-img x* $\circ$ φ*-img y*) (*Pow S*)
using *rewrite-mult*
by *blast*
moreover have
$\forall$ *T*. *T* $\notin$ *Pow S*
$\longrightarrow$ *extensional-continuation* (φ*-img x* $\circ$ φ*-img y*) (*Pow S*) *T* = *undefined*
by *simp*
moreover have
$\forall$ *T*. *T* $\notin$ *Pow S* $\longrightarrow$ φ*-img* (*x* $\otimes_{m}$ *y*) *T* = *undefined* **and**
$\forall$ *T* $\in$ *Pow S*.

extensional-continuation (φ*-img* $x \circ \varphi$*-img* y) (*Pow* S) $T = \varphi\ x$ ' $\varphi\ y$ ' T
using *h-closed*
unfolding φ*-img-def*
by (*simp*, *simp*)
moreover have $\forall\ T \in$ *Pow* S. φ*-img* ($x \otimes_m y$) $T = \varphi\ x$ ' $\varphi\ y$ ' T
unfolding φ*-img-def extensional-continuation.simps*
using *rewrite-group-action-img carrier-x carrier-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* ($x \otimes_m y$) = φ*-img* $x \otimes_{BijGroup\ (Pow\ S)}$ φ*-img* y
by *blast*
qed

1.9.7 Invariance and Equivariance

It suffices to show equivariance under the group action of a generating set of a group to show equivariance 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 *equivar-generators-imp-equivar-group*:
fixes
f :: $'x \Rightarrow 'y$ **and**
m :: $'z$ *monoid* **and**
S :: $'z$ *set* **and**
T :: $'x$ *set* **and**
φ :: ($'z$, $'x$) *binary-fun* **and**
ψ :: ($'z$, $'y$) *binary-fun*
assumes
equivar: *is-symmetry* f (*action-induced-equivariance* S T φ ψ) **and**
action-φ: *group-action* m T φ **and**
action-ψ: *group-action* m (f ' T) ψ **and**
gen: *carrier* m = *generate* m S
shows *is-symmetry* f (*action-induced-equivariance* (*carrier* m) T φ ψ)
proof (*unfold is-symmetry.simps action-induced-equivariance-def action-induced-rel.simps*, *safe*)
fix
g :: $'z$ **and**
x :: $'x$
assume
group-elem: $g \in$ *carrier* m **and**
x-in-t: $x \in T$
have $g \in$ *generate* m S
using *group-elem gen*
by *blast*
hence $\forall\ x \in T$. f ($\varphi\ g\ x$) = $\psi\ g$ ($f\ x$)
proof (*induct* g *rule*: *generate.induct*)

case *one*
hence $\forall\ x \in T.\ \varphi\ \mathbf{1}_{\ m}\ x = x$
using *action-φ group-action.id-eq-one restrict-apply*
by *metis*
moreover with *one* **have** $\forall\ y \in (f\ `\ T).\ \psi\ \mathbf{1}_{\ m}\ y = y$
using *action-ψ group-action.id-eq-one restrict-apply*
by *metis*
ultimately show *?case*
by *simp*
next
case (*incl g*)
hence $\forall\ x \in T.\ \varphi\ g\ x \in T$
using *action-φ gen generate.incl group-action.element-image*
by *metis*
thus *?case*
using *incl equivar rewrite-equivariance*
unfolding *is-symmetry.simps*
by *metis*
next
case (*inv g*)
hence *in-t*: $\forall\ x \in T.\ \varphi\ (inv_{\ m}\ g)\ x \in T$
using *action-φ gen generate.inv group-action.element-image*
by *metis*
hence $\forall\ x \in T.\ f\ (\varphi\ g\ (\varphi\ (inv_{\ m}\ g)\ x)) = \psi\ g\ (f\ (\varphi\ (inv_{\ m}\ g)\ x))$
using *gen action-φ equivar local.inv rewrite-equivariance*
by *metis*
moreover have $\forall\ x \in T.\ \varphi\ g\ (\varphi\ (inv_{\ m}\ g)\ x) = x$
using *action-φ gen generate.incl group.inv-closed group-action.orbit-sym-aux group.inv-inv group-action.group-hom local.inv*
unfolding *group-hom-def*
by (*metis* (*full-types*))
ultimately have $\forall\ x \in T.\ \psi\ g\ (f\ (\varphi\ (inv_{\ m}\ g)\ x)) = f\ x$
by *simp*
moreover have *in-img-t*: $\forall\ x \in T.\ f\ (\varphi\ (inv_{\ m}\ g)\ x) \in f\ `\ T$
using *in-t*
by *blast*
ultimately have
$\forall\ x \in T.\ \psi\ (inv_{\ m}\ g)\ (\psi\ g\ (f\ (\varphi\ (inv_{\ m}\ g)\ x))) = \psi\ (inv_{\ m}\ g)\ (f\ x)$
using *action-ψ gen*
by *metis*
moreover have
$\forall\ x \in T.\ \psi\ (inv_{\ m}\ g)\ (\psi\ g\ (f\ (\varphi\ (inv_{\ m}\ g)\ x))) = f\ (\varphi\ (inv_{\ m}\ g)\ x)$
using *in-img-t action-ψ gen generate.incl group-action.orbit-sym-aux local.inv*
by *metis*
ultimately show *?case*
by *simp*
next
case (*eng* g_1 g_2)
assume

equivar$_1$: $\forall\ x \in T.\ f\ (\varphi\ g_1\ x) = \psi\ g_1\ (f\, x)$ **and**
equivar$_2$: $\forall\ x \in T.\ f\ (\varphi\ g_2\ x) = \psi\ g_2\ (f\, x)$ **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 action-φ group-action.element-image*
by *metis*
hence $\forall\ x \in T.\ f\ (\varphi\ g_1\ (\varphi\ g_2\ x)) = \psi\ g_1\ (f\ (\varphi\ g_2\ x))$
using *equivar*$_1$
by *simp*
moreover have $\forall\ x \in T.\ f\ (\varphi\ g_2\ x) = \psi\ g_2\ (f\, x)$
using *equivar*$_2$
by *simp*
ultimately show *?case*
using *action-φ action-ψ gen gen*$_1$ *gen*$_2$ *group-action.composition-rule imageI*
by (*metis* (*no-types*, *lifting*))
qed
thus $f\ (\varphi\ g\ x) = \psi\ g\ (f\, x)$
using *x-in-t*
by *simp*
qed

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

lemma *invar-under-subset-rel*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$r\ s :: 'x$ *rel*
assumes
subset: $r \subseteq s$ **and**
invar: *is-symmetry* f (*Invariance* s)
shows *is-symmetry* 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**

$\varphi\ \varphi'$ :: $('x,\ 'y)$ *binary-fun* **and**
ψ :: $('x,\ 'z)$ *binary-fun*
assumes $\forall\ x \in S.\ \forall\ y \in T.\ \varphi\ x\ y = \varphi'\ x\ y$
shows *is-symmetry f* (*action-induced-equivariance* $S\ T\ \varphi\ \psi$) =
is-symmetry f (*action-induced-equivariance* $S\ T\ \varphi'\ \psi$)
using *assms*
unfolding *rewrite-equivariance*
by *simp*

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

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

theorem *group-action-equivar-f-imp-equivar-preimg*:
fixes
f :: $'x \Rightarrow\ 'y$ **and**
$\mathcal{D}_f\ S$ :: $'x$ *set* **and**
m :: $'z$ *monoid* **and**
φ :: $('z,\ 'x)$ *binary-fun* **and**
ψ :: $('z,\ 'y)$ *binary-fun* **and**
x :: $'z$
defines *equivar-prop* $\equiv$ *action-induced-equivariance* (*carrier m*) $\mathcal{D}_f\ \varphi\ \psi$
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-restricted-rel (*action-induced-rel* (*carrier m*) *S* φ) *S* $\mathcal{D}_f$ **and**
equivar-f: *is-symmetry 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 action-*φ*.group-hom group-elem-x*
unfolding *group-hom-def*
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*: $z \in \mathcal{D}_f \wedge z \in S$
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*) $\in$ (*action-induced-rel* (*carrier m*) *S* φ) $\cap$ ($\mathcal{D}_f \times S$)
using *img preimg-el domain group-elem-inv*
by *auto*
hence $a \in$ ((*action-induced-rel* (*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 action-induced-equivariance-def*
by *simp*
hence *f a* = *y*
using *preimg-el action-results.group-hom action-results.orbit-sym-aux group-elem-x*
by *simp*
hence $a \in$ *preimg f* $\mathcal{D}_f$ *y*

```
    using a-in-domain
    by simp
  moreover have z = φ x a
    using action-φ.group-hom action-φ.orbit-sym-aux img domain
          a-in-domain group-elem-x group-elem-inv group.inv-inv
    unfolding group-hom-def
    by metis
  ultimately show z ∈ (φ x) ‘ (preimg f 𝒟f y)
    by simp
next
  fix
    y :: 'y and
    z :: 'x
  assume z ∈ preimg f 𝒟f y
  hence domain: f z = y ∧ z ∈ 𝒟f ∧ z ∈ S
    using dom-in-s
    by auto
  hence φ x z ∈ S
    using group-elem-x group-action.element-image action-φ
    by metis
  hence (z, φ x z) ∈ (action-induced-rel (carrier m) S φ) ∩ (𝒟f × S) ∩ 𝒟f × S
    using group-elem-x domain
    by auto
  hence φ x z ∈ 𝒟f
    using closed-domain
    by auto
  moreover have (φ x, ψ x) ∈ {(φ a, ψ a) | a. a ∈ carrier m}
    using group-elem-x
    by blast
  ultimately show φ x z ∈ preimg f 𝒟f (ψ x y)
    using equivar-f domain
    unfolding equivar-prop-def action-induced-equivariance-def
    by simp
qed
```

1.9.8 Function Composition

```
lemma invar-comp:
  fixes
    f :: 'x ⇒ 'y and
    g :: 'y ⇒ 'z and
    r :: 'x rel
  assumes is-symmetry f (Invariance r)
  shows is-symmetry (g ∘ 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**
$\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set* **and**
$\upsilon :: (('y \Rightarrow 'y) \times ('z \Rightarrow 'z))$ *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**
is-symmetry f (*Equivariance S* τ) **and**
is-symmetry g (*Equivariance T* υ)
shows *is-symmetry* $(g \circ f)$ (*Equivariance S transitive-acts*)
proof (*unfold transitive-acts-def is-symmetry.simps comp-def*, *safe*)
fix
$\varphi :: 'x \Rightarrow 'x$ **and**
$\chi :: 'y \Rightarrow 'y$ **and**
$\psi :: 'z \Rightarrow 'z$ **and**
$x :: 'x$
assume
x-in-X: $x \in S$ **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*
by *fastforce*
finally show $g\ (f\ (\varphi\ x)) = \psi\ (g\ (f\,x))$
by *simp*
qed

lemma *equivar-ind-by-action-comp*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$g :: 'y \Rightarrow 'z$ **and**
$S :: 'w$ *set* **and**
$T :: 'x$ *set* **and**
$U :: 'y$ *set* **and**
$\varphi :: ('w, 'x)$ *binary-fun* **and**
$\chi :: ('w, 'y)$ *binary-fun* **and**
$\psi :: ('w, 'z)$ *binary-fun*
assumes

f ‘ $T \subseteq U$ **and**
$\forall\ x \in S.\ \chi\ x$ ‘ f ‘ $T \subseteq U$ **and**
is-symmetry f (*action-induced-equivariance* $S\ T\ \varphi\ \chi$) **and**
is-symmetry g (*action-induced-equivariance* $S\ U\ \chi\ \psi$)
shows *is-symmetry* $(g \circ f)$ (*action-induced-equivariance* $S\ T\ \varphi\ \psi$)
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 *is-symmetry* $(g \circ f)$ (*Equivariance* T $\{(\varphi\ a,\ \psi\ a) \mid a.\ a \in S\}$)
using *assms equivar-comp*[*of* - - - $?a_\varphi$ - $?a_\psi$] *equivar-under-subset′*
unfolding *action-induced-equivariance-def*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
unfolding *action-induced-equivariance-def*
by *blast*
qed

lemma *equivar-set-minus*:
fixes
$f\ g :: {'x} \Rightarrow {'y}$ *set* **and**
$S :: {'z}$ *set* **and**
$T :: {'x}$ *set* **and**
$\varphi :: ({'z},\ {'x})$ *binary-fun* **and**
$\psi :: ({'z},\ {'y})$ *binary-fun*
assumes
f-equivar: *is-symmetry* f (*action-induced-equivariance* $S\ T\ \varphi$ (*set-action* ψ))
and
g-equivar: *is-symmetry* g (*action-induced-equivariance* $S\ T\ \varphi$ (*set-action* ψ))
and
bij-a: $\forall\ a \in S.$ *bij* $(\psi\ a)$
shows
is-symmetry $(\lambda\ b.\ f\ b - g\ b)$ (*action-induced-equivariance* $S\ T\ \varphi$ (*set-action* ψ))
proof −
have
$\forall\ a \in S.\ \forall\ x \in T.\ f\ (\varphi\ a\ x) = \psi\ a$ ‘ $(f\ x)$ **and**
$\forall\ a \in S.\ \forall\ x \in T.\ g\ (\varphi\ a\ x) = \psi\ a$ ‘ $(g\ x)$
using *f-equivar g-equivar*
unfolding *rewrite-equivariance*
by (*simp*, *simp*)
hence $\forall\ a \in S.\ \forall\ b \in T.\ f\ (\varphi\ a\ b) - g\ (\varphi\ a\ b) = \psi\ a$ ‘ $(f\ b) - \psi\ a$ ‘ $(g\ b)$
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-equivariance
    by fastforce
qed

lemma equivar-union-under-image-action:
  fixes
    f :: 'x ⇒ 'y and
    S :: 'z set and
    φ :: ('z, 'x) binary-fun
  shows is-symmetry ⋃ (action-induced-equivariance S UNIV
            (set-action (set-action φ)) (set-action φ))
  unfolding action-induced-equivariance-def is-symmetry.simps set-action.simps
  by blast

end
```

1.10 Symmetry Properties of Voting Rules

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

1.10.1 Definitions

```
fun result-action :: ('x, 'r) binary-fun ⇒ ('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

Bijection group on the set of voters.

```
definition bijection_𝒱𝒢 :: ('v ⇒ 'v) monoid where
  bijection_𝒱𝒢 ≡ BijGroup (UNIV :: 'v set)
```

Permutation action on the set of voters. Invariance under this action implies anonymity.

```
fun φ-anon :: ('a, 'v) Election set ⇒ ('v ⇒ 'v) ⇒
        (('a, 'v) Election ⇒ ('a, 'v) Election) where
  φ-anon ℰ π = extensional-continuation (rename π) ℰ
```

fun $\mathit{anonymity}_{\mathcal{R}}$ :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$\mathit{anonymity}_{\mathcal{R}}\ \mathcal{E} = \mathit{action\text{-}induced\text{-}rel}\ (\mathit{carrier}\ \mathit{bijection}_{\mathcal{VG}})\ \mathcal{E}\ (\varphi\mathit{\text{-}anon}\ \mathcal{E})$

Neutrality

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

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

Bijection group on the set of alternatives. Invariance under this action implies neutrality.

definition $\mathit{bijection}_{\mathcal{AG}}$ :: $('a \Rightarrow 'a)$ *monoid* **where**
$\mathit{bijection}_{\mathcal{AG}} \equiv \mathit{BijGroup}\ (\mathit{UNIV} :: 'a\ \mathit{set})$

Permutation action on the set of alternatives.

fun φ*-neutral* :: $('a,\ 'v)$ *Election set* $\Rightarrow$
$('a \Rightarrow 'a,\ ('a,\ 'v)\ \mathit{Election})$ *binary-fun* **where**
$\varphi\mathit{\text{-}neutral}\ \mathcal{E}\ \pi = \mathit{extensional\text{-}continuation}\ (\mathit{alternatives\text{-}rename}\ \pi)\ \mathcal{E}$

fun $\mathit{neutrality}_{\mathcal{R}}$ :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$\mathit{neutrality}_{\mathcal{R}}\ \mathcal{E} = \mathit{action\text{-}induced\text{-}rel}\ (\mathit{carrier}\ \mathit{bijection}_{\mathcal{AG}})\ \mathcal{E}\ (\varphi\mathit{\text{-}neutral}\ \mathcal{E})$

fun $\psi\mathit{\text{-}neutral}_{\mathrm{c}}$ :: $('a \Rightarrow 'a,\ 'a)$ *binary-fun* **where**
$\psi\mathit{\text{-}neutral}_{\mathrm{c}}\ \pi\ r = \pi\ r$

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

Homogeneity

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

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

fun $\mathit{homogeneity}_{\mathcal{R}}'$ :: $('a,\ 'v :: \mathit{linorder})$ *Election set* $\Rightarrow$ $('a,\ 'v)$ *Election rel* **where**
$\mathit{homogeneity}_{\mathcal{R}}'\ \mathcal{E} =$
$\{(E,\ E').\ E \in \mathcal{E}$

$\wedge$ *alternatives-$\mathcal{E}$ E = alternatives-$\mathcal{E}$ E'*
$\wedge$ *finite* (*voters-$\mathcal{E}$ E*) $\wedge$ *finite* (*voters-$\mathcal{E}$ E'*)
$\wedge$ ($\exists$ *n* > *0*.
  to-list (*voters-$\mathcal{E}$ E'*) (*profile-$\mathcal{E}$ E'*) =
   copy-list n (*to-list* (*voters-$\mathcal{E}$ E*) (*profile-$\mathcal{E}$ E*)))}

Reversal Symmetry

fun *reverse-rel* :: *'a rel* $\Rightarrow$ *'a rel* **where**
 reverse-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-app f (*A*, *V*, *p*) = (*A*, *V*, *f* $\circ$ *p*)

definition *reversal*$_{\mathcal{G}}$ :: (*'a rel* $\Rightarrow$ *'a rel*) *monoid* **where**
 reversal$_{\mathcal{G}}$ $\equiv$ (|*carrier* = {*reverse-rel*, *id*}, *monoid.mult* = *comp*, *one* = *id*|)

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

fun ψ*-reverse* :: (*'a rel* $\Rightarrow$ *'a rel*, *'a rel*) *binary-fun* **where**
 ψ*-reverse* φ *r* = φ *r*

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

1.10.2 Auxiliary Lemmas

fun *n-app* :: *nat* $\Rightarrow$ (*'x* $\Rightarrow$ *'x*) $\Rightarrow$ (*'x* $\Rightarrow$ *'x*) **where**
 n-app-id: *n-app 0 f* = *id* |
 n-app-suc: *n-app* (*Suc n*) *f* = *f* $\circ$ *n-app n f*

lemma *n-app-rewrite*:
 fixes
  f :: *'x* $\Rightarrow$ *'x* **and**
  n :: *nat* **and**
  x :: *'x*
 shows (*f* $\circ$ *n-app n f*) *x* = (*n-app n f* $\circ$ *f*) *x*
proof (*unfold comp-def*, *induction n f arbitrary*: *x rule*: *n-app.induct*)
 case (*1 f*)
 fix
  f :: *'x* $\Rightarrow$ *'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* $\Rightarrow$ *'x* **and**

    n :: *nat* **and**
    x :: *'x*
  assume $\bigwedge$ *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 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-f: *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 −
  have *n-app-f-x-in-A*: *n-app 0 f x* ∈ *A*
    using *x-el*
    by *simp*
  moreover have *ex-A*:
    ∃ *n* > *0*. *n-app n f x* ∈ *B* − *A* ∧ (∀ *m* > *0*. *m* < *n* ⟶ *n-app m f x* ∈ *A*)
  proof (*rule ccontr*,
        unfold Diff-iff conj-assoc not-ex de-Morgan-conj not-gr-zero
            simp-thms not-all not-imp disj-not1 imp-disj2)
    assume *nex*:
      ∀ *n*. *n-app n f x* ∈ *B*
        ⟶ *n* = *0* ∨ *n-app n f x* ∈ *A* ∨ (∃ *m* > *0*. *m* < *n* ∧ *n-app m f x* ∉ *A*)
    hence ∀ *n* > *0*. *n-app n f x* ∈ *B*
          ⟶ *n-app n f x* ∈ *A* ∨ (∃ *m* > *0*. *m* < *n* ∧ *n-app m f x* ∉ *A*)
      by *blast*
    moreover have ¬ (∀ *n* > *0*. *n-app n f x* ∈ *B* ⟶ *n-app n f x* ∈ *A*)
    proof (*safe*)
      assume *in-A*: ∀ *n* > *0*. *n-app n f x* ∈ *B* ⟶ *n-app n f x* ∈ *A*
      hence ∀ *n* > *0*. *n-app n f x* ∈ *A* ⟶ *n-app* (*Suc n*) *f x* ∈ *A*
        using *n-app.simps bij-f*
        unfolding *bij-betw-def*
        by *force*
      hence *in-AB-imp-in-AB*:
        ∀ *n* > *0*. *n-app n f x* ∈ *A* ∩ *B* ⟶ *n-app* (*Suc n*) *f x* ∈ *A* ∩ *B*
        using *n-app.simps bij-f*
        unfolding *bij-betw-def*
        by *auto*
      have *in-int*: ∀ *n* > *0*. *n-app n f x* ∈ *A* ∩ *B*

proof (*clarify*)
  fix *n* :: *nat*
  assume *n* > *0*
  thus *n-app n f x* ∈ *A* ∩ *B*
  proof (*induction n*)
    case *0*
    thus *?case*
      by *safe*
  next
    case (*Suc n*)
    assume *0* < *n* ⟹ *n-app n f x* ∈ *A* ∩ *B*
    moreover have *n* = *0* ⟶ *n-app* (*Suc n*) *f x* = *f x*
      by *simp*
    ultimately show *n-app* (*Suc n*) *f x* ∈ *A* ∩ *B*
      using *x-el bij-f in-A in-AB-imp-in-AB*
      unfolding *bij-betw-def*
      by *blast*
  qed
qed
hence {*n-app n f x* | *n. n* > *0*} ⊆ *A* ∩ *B*
  by *blast*
hence *finite* {*n-app n f x* | *n. n* > *0*}
  using *fin-A fin-B rev-finite-subset*
  by *blast*
moreover have
  inj-on (λ *n. n-app n f x*) {*n. n* > *0*}
    ⟶ *infinite* ((λ *n. n-app n f x*) ‘ {*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 (λ *n. n-app n f x*) ‘ {*n. n* > *0*} = {*n-app n f x* | *n. n* > *0*}
  by *auto*
ultimately have ∃ *n* > *0* . ∃ *m* > *n. n-app n f x* = *n-app m f x*
  using *linorder-inj-onI′ mem-Collect-eq*
  by (*metis* (*no-types*, *lifting*))
hence ∃ *n-min* > *0*.
    (∃ *m* > *n-min. n-app n-min f x* = *n-app m f x*)
  ∧ (∀ *n* < *n-min.* ¬ (*0* < *n* ∧ (∃ *m* > *n. n-app n f x* = *n-app m f x*)))
  using *exists-least-iff*[*of*
          λ *n. n* > *0* ∧ (∃ *m* > *n. n-app n f x* = *n-app m f x*)]
  by *presburger*
then obtain *n-min* :: *nat* **where**
  n-min-pos: *n-min* > *0* **and**
  ex-gt-n-min: ∃ *m* > *n-min. n-app n-min f x* = *n-app m f x* **and**
  neq: ∀ *n* < *n-min.* ¬ (*n* > *0* ∧ (∃ *m* > *n. n-app n f x* = *n-app m f x*))
  by *blast*
then obtain *m* :: *nat* **where**
  m-gt-n-min: *m* > *n-min* **and**
  n-app-n-min-eq: *n-app n-min f x* = *f* (*n-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-app n-min f x* = *f* (*n-app* (*n-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-app* (*m* − *1*) *f x* ∈ *A* ∧ *n-app* (*n-min* − *1*) *f x* ∈ *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-id*
by *metis*
ultimately have *n-app* (*m* − *1*) *f x* = *n-app* (*n-min* − *1*) *f x*
using *bij-f*
unfolding *bij-betw-def inj-def inj-on-def*
by *simp*
moreover have *m* − *1* > *n-min* − *1*
using *m-gt-n-min n-min-pos*
by *simp*
moreover have *n-app* (*m* − *1*) *f x* ∈ *B*
using *in-int m-gt-n-min n-min-pos*
by *simp*
ultimately show *False*
using *x-el neq n-min-pos diff-less zero-less-one Diff-iff bot-nat-0.not-eq-extremum id-apply n-app-id*
by *metis*
qed
ultimately obtain *n* :: *nat* **where**
n-pos: *n* > *0* **and**
not-in-A: *n-app n f x* ∉ *A* **and**
less-in-A: ∀ *m*. (*0* < *m* ∧ *m* < *n*) ⟶ *n-app m f x* ∈ *A*
using *exists-least-iff*[*of* λ *n*. *n* > *0* ∧ *n-app n f x* ∉ *A*]
by *blast*
hence *n-app* (*n* − *1*) *f x* ∈ *A*
using *n-app-f-x-in-A bot-nat-0.not-eq-extremum diff-less zero-less-one*
by *metis*
moreover have *n-app n f x* = *f* (*n-app* (*n* − *1*) *f x*)
using *n-app-suc Suc-pred′ n-pos comp-eq-id-dest fun.map-id*
by (*metis* (*mono-tags*, *opaque-lifting*))
ultimately show *False*
using *bij-f nex not-in-A n-pos less-in-A*
unfolding *bij-betw-def*
by *blast*
qed
ultimately have
∀ *n*. (∀ *m* > *0*. *m* < *n* ⟶ *n-app m f x* ∈ *A*)
⟶ (∀ *m* > *0*. *m* < *n* ⟶ *n-app* (*m* − *1*) *f x* ∈ *A*)
using *bot-nat-0.not-eq-extremum less-imp-diff-less*
by *metis*
moreover have ∀ *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 show *?thesis*
using *that bij-f imageI IntI ex-A*
unfolding *bij-betw-def*
by *metis*
qed

lemma *n-app-rev*:
fixes
$A\ B$ :: $'x$ *set* **and**
f :: $'x \Rightarrow 'x$ **and**
$m\ n$ :: *nat* **and**
$x\ y$ :: $'x$
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**
$x\ y$ :: $'x$
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**
$m\ n$ :: *nat* **and**
$x\ y$ :: $'x$
assume
bij-f: *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-app* $n'\ f\ x \in A$ **and**
y-dom: $\forall\ m' < m$. *n-app* $m'\ f\ y \in A$ **and**

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

lemma *n-app-inv*:
fixes
A B :: *'x set* **and**
f :: *'x* ⇒ *'x* **and**
n :: *nat* **and**
x :: *'x*
assumes
$x \in B$ **and**
$\forall\ m \geq 0.\ m < n \longrightarrow$ *n-app m* (*the-inv-into A f*) $x \in B$ **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-f: 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-f
    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-f stays-in-B x-in-B
    by simp
qed

lemma bij-betw-finite-ind-global-bij:
  fixes
    A B :: 'x set and
    f :: 'x ⇒ 'x
  assumes
    fin-A: finite A and
    fin-B: finite B and
    bij-f: 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-f bij-betw-the-inv-into*
by *blast*
then obtain g' :: $'x \Rightarrow nat$ **where**
g'*-greater-zero*: $\forall\ x \in B - A.\ g'\ x > 0$ **and**
in-set-diff: $\forall\ x \in B - A.\ n\text{-}app\ (g'\ x)\ (the\text{-}inv\text{-}into\ A\ f)\ x \in A - B$ **and**
minimal: $\forall\ x \in B - A.\ \forall\ n > 0.$
$n < g'\ x \longrightarrow n\text{-}app\ n\ (the\text{-}inv\text{-}into\ A\ f)\ x \in B \cap A$
using *n-app-leaves-set fin-A fin-B*
by *metis*
obtain g :: $'x \Rightarrow 'x$ **where**
def-g:
$g = (\lambda\ x.\ if\ x \in A\ then\ f\ x\ else$
$if\ x \in B - A\ then\ n\text{-}app\ (g'\ x)\ (the\text{-}inv\text{-}into\ A\ f)\ x\ else\ x)$
by *simp*
hence *coincide*: $\forall\ a \in A.\ g\ a = f\ a$
by *simp*
have *id*: $\forall\ x \in UNIV - A - B.\ g\ x = x$
using *def-g*
by *simp*
have $\forall\ x \in B - A.\ n\text{-}app\ 0\ (the\text{-}inv\text{-}into\ A\ f)\ x \in B$
by *simp*
moreover have
$\forall\ x \in B - A.\ \forall\ n > 0.$
$n < g'\ x \longrightarrow n\text{-}app\ n\ (the\text{-}inv\text{-}into\ A\ f)\ x \in B$
using *minimal*
by *blast*
ultimately have
$\forall\ x \in B - A.\ n\text{-}app\ (g'\ x)\ f\ (n\text{-}app\ (g'\ x)\ (the\text{-}inv\text{-}into\ A\ f)\ x) = x$
using *n-app-inv bij-f DiffD1 antisym-conv2*
by *metis*
hence $\forall\ x \in B - A.\ n\text{-}app\ (g'\ x)\ f\ (g\ x) = x$
using *def-g*
by *simp*
with g'*-greater-zero in-set-diff*
have *reverse*: $\forall\ x \in B - A.\ g\ x \in A - B \wedge (\exists\ n > 0.\ n\text{-}app\ n\ f\ (g\ x) = x)$
using *def-g*
by *auto*
have $\forall\ x \in UNIV - A - B.\ g\ x = id\ x$
using *def-g*
by *simp*
hence g ‘ $(UNIV - A - B) = UNIV - A - B$
by *simp*
moreover have g ‘ $A = B$
using *def-g bij-f*
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*: 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-f*
  unfolding *bij-betw-def inj-on-def*
  by *simp*
hence *inj-cases*: *inj-on* g $(UNIV - (B - A))$
  unfolding *inj-on-def*
  using *DiffD2 DiffI bij-f bij-betwE def-g*
  by (*metis* (*no-types*, *lifting*))
have *card* $A =$ *card* B
  using *fin-A fin-B bij-f 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 $x\ y :: 'x$
  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-id*
    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 *g′-greater-zero 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-case: 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 Un-Diff-cancel2 image-Un sup-top-left
    unfolding bij-betw-def
    by metis
  moreover have inj g
    using inj-cases bij-case DiffD2 DiffI imageI surj-cases
    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 ⇒ ′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 ∀ x ∈ X. extensional-continuation f X x = f x
    by simp
  thus ?thesis
    using assms bij-betw-cong
    by metis
qed
```

1.10.3 Anonymity Lemmas

```
lemma anon-rel-vote-count:
  fixes
    ℰ :: (′a, ′v) Election set and
    E E′ :: (′a, ′v) Election
  assumes
    finite (voters-ℰ E) and
```

$(E, E') \in anonymity_{\mathcal{R}}\ \mathcal{E}$
shows *alternatives-*$\mathcal{E}$ E = *alternatives-*$\mathcal{E}$ $E' \wedge E \in \mathcal{E}$
$\wedge$ ($\forall$ *p*. *vote-count p E* = *vote-count p E'*)
proof −
have $E \in \mathcal{E}$
using *assms*
unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.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 bijection*$_{\mathcal{VG}}$*-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* $\in$ π ‘ {*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-*$\mathcal{E}$ E'. (*profile-*$\mathcal{E}$ *E*) (*the-inv* π *v*) = *p*} $\subseteq$
π ‘ {*v* $\in$ *voters-*$\mathcal{E}$ *E*. (*profile-*$\mathcal{E}$ *E*) *v* = *p*}
by *blast*
from *renamed* **have** $\forall$ *v* $\in$ *voters-*$\mathcal{E}$ *E*. π *v* $\in$ *voters-*$\mathcal{E}$ 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*. π ‘ {*v* $\in$ *voters-*$\mathcal{E}$ *E*. (*profile-*$\mathcal{E}$ *E*) *v* = *p*} $\subseteq$
{*v* $\in$ *voters-*$\mathcal{E}$ E'. (*profile-*$\mathcal{E}$ *E*) (*the-inv* π *v*) = *p*}
using *bijection-*π *bij-is-inj the-inv-f-f*
by *fastforce*

hence
$\forall\ p.\ \{v \in \textit{voters-}\mathcal{E}\ E'.\ (\textit{profile-}\mathcal{E}\ E')\ v = p\} =$
$\pi\ `\ \{v \in \textit{voters-}\mathcal{E}\ E.\ (\textit{profile-}\mathcal{E}\ E)\ v = p\}$
using *subset rewrite*
by (*simp add*: *subset-antisym*)
moreover have
$\forall\ p.\ \textit{card}\ (\pi\ `\ \{v \in \textit{voters-}\mathcal{E}\ E.\ (\textit{profile-}\mathcal{E}\ E)\ v = p\}) =$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ (\textit{profile-}\mathcal{E}\ E)\ v = p\}$
using *bijection-*π *bij-betw-same-card bij-betw-subset top-greatest*
by (*metis* (*no-types*, *lifting*))
ultimately show
$\textit{alternatives-}\mathcal{E}\ E =$
$\textit{alternatives-}\mathcal{E}\ E' \wedge E \in \mathcal{E} \wedge (\forall\ p.\ \textit{vote-count}\ p\ E = \textit{vote-count}\ p\ E')$
using *eq-alts assms*
by *simp*
qed

lemma *vote-count-anon-rel*:
fixes
$\mathcal{E}$ :: $('a,\ 'v)$ *Election set* **and**
$E\ E'$ :: $('a,\ 'v)$ *Election*
assumes
fin-voters-E: *finite* $(\textit{voters-}\mathcal{E}\ E)$ **and**
fin-voters-E': *finite* $(\textit{voters-}\mathcal{E}\ E')$ **and**
default-non-v: $\forall\ v.\ v \notin \textit{voters-}\mathcal{E}\ E \longrightarrow \textit{profile-}\mathcal{E}\ E\ v = \{\}$ **and**
default-non-v': $\forall\ v.\ v \notin \textit{voters-}\mathcal{E}\ E' \longrightarrow \textit{profile-}\mathcal{E}\ E'\ v = \{\}$ **and**
eq: $\textit{alternatives-}\mathcal{E}\ E = \textit{alternatives-}\mathcal{E}\ E' \wedge (E,\ E') \in \mathcal{E} \times \mathcal{E}$
$\wedge (\forall\ p.\ \textit{vote-count}\ p\ E = \textit{vote-count}\ p\ E')$
shows $(E,\ E') \in \textit{anonymity}_{\mathcal{R}}\ \mathcal{E}$
proof –
have $\forall\ p.\ \textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\} =$
$\textit{card}\ \{v \in \textit{voters-}\mathcal{E}\ E'.\ \textit{profile-}\mathcal{E}\ E'\ v = p\}$
using *eq*
unfolding *vote-count.simps*
by *blast*
moreover have
$\forall\ p.\ \textit{finite}\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\}$
$\wedge\ \textit{finite}\ \{v \in \textit{voters-}\mathcal{E}\ E'.\ \textit{profile-}\mathcal{E}\ E'\ v = p\}$
using *assms*
by *simp*
ultimately have
$\forall\ p.\ \exists\ \pi_p.\ \textit{bij-betw}\ \pi_p$
$\{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\}$
$\{v \in \textit{voters-}\mathcal{E}\ E'.\ \textit{profile-}\mathcal{E}\ E'\ v = p\}$
using *bij-betw-iff-card*
by *blast*
then obtain π :: $'a\ \textit{Preference-Relation} \Rightarrow ('v \Rightarrow 'v)$ **where**
bij-π: $\forall\ p.\ \textit{bij-betw}\ (\pi\ p)\ \{v \in \textit{voters-}\mathcal{E}\ E.\ \textit{profile-}\mathcal{E}\ E\ v = p\}$
$\{v \in \textit{voters-}\mathcal{E}\ E'.\ \textit{profile-}\mathcal{E}\ E'\ v = p\}$

by (*metis* (*no-types*))
obtain $\pi' :: 'v \Rightarrow 'v$ **where**
π'-perm: $\forall\ v \in$ *voters-$\mathcal{E}$ E*. $\pi'\ v = \pi$ (*profile-$\mathcal{E}$ E v*) v
by *fastforce*
hence $\forall\ v \in$ *voters-$\mathcal{E}$ E*. $\forall\ v' \in$ *voters-$\mathcal{E}$ E*.
$\pi'\ v = \pi'\ v' \longrightarrow \pi$ (*profile-$\mathcal{E}$ E v*) $v = \pi$ (*profile-$\mathcal{E}$ E v'*) v'
by *simp*
moreover have
$\forall\ w \in$ *voters-$\mathcal{E}$ E*. $\forall\ w' \in$ *voters-$\mathcal{E}$ E*.
π (*profile-$\mathcal{E}$ E w*) $w = \pi$ (*profile-$\mathcal{E}$ E w'*) w'
$\longrightarrow \{v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* = *profile-$\mathcal{E}$ E w*$\}$
$\cap\ \{v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* = *profile-$\mathcal{E}$ E w'*$\} \neq \{\}$
using *bij-π*
unfolding *bij-betw-def*
by *blast*
moreover have
$\forall\ w\ w'$.
$\{v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* = *profile-$\mathcal{E}$ E w*$\}$
$\cap\ \{v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* = *profile-$\mathcal{E}$ E w'*$\} \neq \{\}$
$\longrightarrow$ *profile-$\mathcal{E}$ E w* = *profile-$\mathcal{E}$ E w'*
by *blast*
ultimately have *eq-prof*:
$\forall\ v \in$ *voters-$\mathcal{E}$ E*. $\forall\ v' \in$ *voters-$\mathcal{E}$ E*.
$\pi'\ v = \pi'\ v' \longrightarrow$ *profile-$\mathcal{E}$ E v* = *profile-$\mathcal{E}$ E v'*
by *blast*
hence $\forall\ v \in$ *voters-$\mathcal{E}$ E*. $\forall\ v' \in$ *voters-$\mathcal{E}$ E*.
$\pi'\ v = \pi'\ v' \longrightarrow \pi$ (*profile-$\mathcal{E}$ E v*) $v = \pi$ (*profile-$\mathcal{E}$ E v*) v'
using *π'-perm*
by *metis*
hence $\forall\ v \in$ *voters-$\mathcal{E}$ E*. $\forall\ v' \in$ *voters-$\mathcal{E}$ E*. $\pi'\ v = \pi'\ v' \longrightarrow v = v'$
using *bij-π eq-prof mem-Collect-eq*
unfolding *bij-betw-def inj-on-def*
by (*metis* (*mono-tags*, *lifting*))
hence *inj*: *inj-on* π' (*voters-$\mathcal{E}$ E*)
unfolding *inj-on-def*
by *simp*
have π' ‘ *voters-$\mathcal{E}$ E* = $\{\pi$ (*profile-$\mathcal{E}$ E v*) $v \mid v.\ v \in$ *voters-$\mathcal{E}$ E*$\}$
using *π'-perm*
unfolding *Setcompr-eq-image*
by *simp*
also have
$\ldots = \bigcup\ \{\pi\ p$ ‘ $\{v \in$ *voters-$\mathcal{E}$ E*. *profile-$\mathcal{E}$ E v* $= p\} \mid p.\ p \in$ *UNIV*$\}$
unfolding *Union-eq*
by *blast*
also have
$\ldots = \bigcup\ \{\{v \in$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* $= p\} \mid p.\ p \in$ *UNIV*$\}$
using *bij-π*
unfolding *bij-betw-def*
by (*metis* (*mono-tags*, *lifting*))

finally have π' ` *voters-$\mathcal{E}$ E* = *voters-$\mathcal{E}$ E'*
  by *blast*
with *inj* **have** *bij'*: *bij-betw* π' (*voters-$\mathcal{E}$ E*) (*voters-$\mathcal{E}$ E'*)
  using *bij-π*
  unfolding *bij-betw-def*
  by *blast*
then obtain *π-global* :: *'v* $\Rightarrow$ *'v* **where**
  bijection-π_g: *bij π-global* **and**
  π-global-eq-π': $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E*. *π-global v* = π' *v* **and**
  π-global-eq-n-app-π':
    $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E'* $-$ *voters-$\mathcal{E}$ E*.
      π-global v $\in$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'* $\wedge$
      ($\exists$ *n* > *0*. *n-app n* π' (*π-global v*) = *v*) **and**
  π-global-non-voters: $\forall$ *v* $\in$ *UNIV* $-$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*. *π-global v* = *v*
  using *fin-voters-E fin-voters-E' bij-betw-finite-ind-global-bij*
  by *blast*
hence *inv*: $\forall$ *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
  $\forall$ *v* $\in$ *UNIV* $-$ (*voters-$\mathcal{E}$ E'* $-$ *voters-$\mathcal{E}$ E*).
      π-global v $\in$ *UNIV* $-$ (*voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*)
  using *π-global-eq-π' π-global-non-voters bij' bijection-π_g*
        DiffD1 DiffD2 DiffI bij-betwE
  by (*metis* (*no-types*, *lifting*))
ultimately have
  $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*.
      the-inv π-global v $\in$ *voters-$\mathcal{E}$ E'* $-$ *voters-$\mathcal{E}$ E*
  using *bijection-π_g π-global-eq-n-app-π' DiffD2 DiffI UNIV-I*
  by *metis*
hence $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*. $\forall$ *n* > *0*.
            profile-$\mathcal{E}$ E (*the-inv π-global v*) = {}
  using *default-non-v*
  by *simp*
moreover have $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E* $-$ *voters-$\mathcal{E}$ E'*. *profile-$\mathcal{E}$ E' v* = {}
  using *default-non-v'*
  by *simp*
ultimately have *comp-on-voters-diff*:
  $\forall$ *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* $\wedge$ π' *v'* = *v*
  using *bij' imageE π-global-eq-π'*
  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* $\wedge$ π' *v'* = *v*
  using *inv*
  by *metis*
hence $\forall$ *v* $\in$ *voters-$\mathcal{E}$ E'*.

$the\text{-}inv\ \pi\text{-}global\ v \in voters\text{-}\mathcal{E}\ E \land \pi'\ (the\text{-}inv\ \pi\text{-}global\ v) = v$
by *blast*
moreover have $\forall\ v' \in voters\text{-}\mathcal{E}\ E.\ profile\text{-}\mathcal{E}\ E'\ (\pi'\ v') = profile\text{-}\mathcal{E}\ E\ v'$
using π'*-perm bij-*π *bij-betwE mem-Collect-eq*
by *fastforce*
ultimately have *comp-on-E′-voters*:
$\forall\ v \in voters\text{-}\mathcal{E}\ E'.\ profile\text{-}\mathcal{E}\ E'\ v = (profile\text{-}\mathcal{E}\ E \circ the\text{-}inv\ \pi\text{-}global)\ v$
unfolding *comp-def*
by *metis*
have $\forall\ v \in UNIV - voters\text{-}\mathcal{E}\ E - voters\text{-}\mathcal{E}\ E'.$
$profile\text{-}\mathcal{E}\ E'\ v = (profile\text{-}\mathcal{E}\ E \circ the\text{-}inv\ \pi\text{-}global)\ v$
using π*-global-non-voters default-non-v default-non-v′ inv*
by *simp*
hence $profile\text{-}\mathcal{E}\ E' = profile\text{-}\mathcal{E}\ E \circ the\text{-}inv\ \pi\text{-}global$
using *comp-on-voters-diff comp-on-E′-voters*
by *blast*
moreover have $\pi\text{-}global$ ‘ $(voters\text{-}\mathcal{E}\ E) = voters\text{-}\mathcal{E}\ E'$
using π*-global-eq-*π' *bij′ bij-betw-imp-surj-on*
by *fastforce*
ultimately have $E' = rename\ \pi\text{-}global\ E$
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*
action-induced-rel.simps φ*-anon.simps bijection*$_{\mathcal{VG}}$*-def*
using *eq bijection-*π_g *case-prodI rewrite-carrier*
by *auto*
qed

lemma *rename-comp*:
fixes $\pi\ \pi' :: 'v \Rightarrow 'v$
assumes
bij π **and**
bij π'
shows $rename\ \pi \circ rename\ \pi' = rename\ (\pi \circ \pi')$
proof
fix $E :: ('a, 'v)\ Election$
have $rename\ \pi'\ E =$
$(alternatives\text{-}\mathcal{E}\ E, \pi'$ ‘ $(voters\text{-}\mathcal{E}\ E), (profile\text{-}\mathcal{E}\ E) \circ (the\text{-}inv\ \pi'))$
unfolding *alternatives-*$\mathcal{E}$*.simps voters-*$\mathcal{E}$*.simps profile-*$\mathcal{E}$*.simps*
using *prod.collapse rename.simps*
by *metis*
hence
$(rename\ \pi \circ rename\ \pi')\ E =$
$rename\ \pi\ (alternatives\text{-}\mathcal{E}\ E, \pi'$ ‘ $(voters\text{-}\mathcal{E}\ E), (profile\text{-}\mathcal{E}\ E) \circ (the\text{-}inv\ \pi'))$
unfolding *comp-def*
by *presburger*
also have

$\ldots = (\mathit{alternatives}\text{-}\mathcal{E}\ E,\ \pi\ `\ \pi'\ `\ (\mathit{voters}\text{-}\mathcal{E}\ E),$
$(\mathit{profile}\text{-}\mathcal{E}\ E) \circ (\mathit{the}\text{-}\mathit{inv}\ \pi') \circ (\mathit{the}\text{-}\mathit{inv}\ \pi))$
by *simp*
also have
$\ldots = (\mathit{alternatives}\text{-}\mathcal{E}\ E,\ (\pi \circ \pi')\ `\ (\mathit{voters}\text{-}\mathcal{E}\ E),$
$(\mathit{profile}\text{-}\mathcal{E}\ E) \circ \mathit{the}\text{-}\mathit{inv}\ (\pi \circ \pi'))$
using *assms the-inv-comp*[*of* π - - π']
unfolding *comp-def image-image*
by *simp*
finally show $(\mathit{rename}\ \pi \circ \mathit{rename}\ \pi')\ E = \mathit{rename}\ (\pi \circ \pi')\ E$
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 bijection$_{\mathcal{VG}}$ *well-formed-elections* φ*-anon well-formed-elections*
proof (*unfold group-action-def group-hom-def bijection*$_{\mathcal{VG}}$*-def*
group-hom-axioms-def hom-def, *intro conjI group-BijGroup*, *safe*)
fix π :: $'v \Rightarrow\ 'v$
assume *bij-carrier*: $\pi \in$ *carrier* (*BijGroup UNIV*)
hence *bij-*π: *bij* π
using *rewrite-carrier*
by *blast*
hence *rename* π ` *well-formed-elections* = *well-formed-elections*
using *rename-surj bij-*π
by *blast*
moreover have *inj-on* (*rename* π) *well-formed-elections*
using *rename-inj bij-*π *subset-inj-on*
by *blast*
ultimately have *bij-betw* (*rename* π) *well-formed-elections well-formed-elections*
unfolding *bij-betw-def*
by *blast*
hence *bij-betw* (φ*-anon well-formed-elections* π) *well-formed-elections well-formed-elections*
unfolding φ*-anon.simps extensional-continuation.simps*
using *bij-betw-ext*
by *simp*
moreover have φ*-anon well-formed-elections* $\pi \in$ *extensional well-formed-elections*
unfolding *extensional-def*
by *force*
ultimately show *bij-car-elect*:
φ*-anon well-formed-elections* $\pi \in$ *carrier* (*BijGroup well-formed-elections*)
unfolding *BijGroup-def Bij-def*
by *simp*
fix π' :: $'v \Rightarrow\ 'v$
assume *bij-carrier*: $\pi' \in$ *carrier* (*BijGroup UNIV*)
hence *bij-*π': *bij* π'
using *rewrite-carrier*
by *blast*

hence *rename* π' ‘ *well-formed-elections* = *well-formed-elections*
  using *rename-surj bij-*π
  by *blast*
moreover have *inj-on* (*rename* π') *well-formed-elections*
  using *rename-inj bij-*π' *subset-inj-on*
  by *blast*
ultimately have *bij-betw* (*rename* π') *well-formed-elections well-formed-elections*
  unfolding *bij-betw-def*
  by *blast*
hence *bij-betw* (φ*-anon well-formed-elections* π') *well-formed-elections well-formed-elections*
  unfolding φ*-anon.simps extensional-continuation.simps*
  using *bij-betw-ext*
  by *simp*
moreover from *this* **have** *wf-closed*$'$:
  φ*-anon well-formed-elections* π' ‘ *well-formed-elections* $\subseteq$ *well-formed-elections*
  using *bij-betw-imp-surj-on*
  by *blast*
moreover have φ*-anon well-formed-elections* π' $\in$ *extensional well-formed-elections*
  unfolding *extensional-def*
  by *force*
ultimately have *bij-car-elect*$'$:
  φ*-anon well-formed-elections* π' $\in$ *carrier* (*BijGroup well-formed-elections*)
  unfolding *BijGroup-def Bij-def*
  by *simp*
have
  φ*-anon well-formed-elections* π
    $\otimes_{\textit{BijGroup well-formed-elections}}$ (φ*-anon well-formed-elections*) π' =
   extensional-continuation
   (φ*-anon well-formed-elections* π $\circ$ φ*-anon well-formed-elections* π') *well-formed-elections*
  using *rewrite-mult bij-car-elect bij-car-elect*$'$
  by *blast*
moreover have
  $\forall$ $E \in$ *well-formed-elections.*
   extensional-continuation
    (φ*-anon well-formed-elections* π $\circ$ φ*-anon well-formed-elections* π')
     well-formed-elections E =
    (φ*-anon well-formed-elections* π $\circ$ φ*-anon well-formed-elections* π') E
  by *simp*
moreover have
  $\forall$ $E \in$ *well-formed-elections.*
     (φ*-anon well-formed-elections* π $\circ$ φ*-anon well-formed-elections* π') E =
     rename π (*rename* π' E)
  unfolding φ*-anon.simps*
  using *wf-closed*$'$
  by *auto*
moreover have
  $\forall$ $E \in$ *well-formed-elections. rename* π (*rename* π' E) = *rename* ($\pi \circ \pi'$) E
  using *rename-comp bij-*π *bij-*π' *comp-apply*
  by *metis*

moreover have
  $\forall\ E \in$ *well-formed-elections*. *rename* $(\pi \circ \pi')\ E =$
    φ*-anon well-formed-elections* $(\pi \otimes_{\ BijGroup\ UNIV}\ \pi')\ E$
  unfolding φ*-anon.simps*
  using *rewrite-mult-univ bij-*π *bij-*π' *rewrite-carrier mem-Collect-eq*
  by *fastforce*
moreover have
  $\forall\ E.\ E \notin$ *well-formed-elections*
    $\longrightarrow$ *extensional-continuation*
      $(\varphi$*-anon well-formed-elections* π
        $\circ\ \varphi$*-anon well-formed-elections* $\pi')$ *well-formed-elections* $E =$
    undefined
  by *simp*
moreover have
  $\forall\ E.\ E \notin$ *well-formed-elections*
      $\longrightarrow \varphi$*-anon well-formed-elections* $(\pi \otimes_{\ BijGroup\ UNIV}\ \pi')\ E =$
        undefined
  by *simp*
ultimately have
  $\forall\ E.\ \varphi$*-anon well-formed-elections* $(\pi \otimes_{\ BijGroup\ UNIV}\ \pi')\ E =$
    $(\varphi$*-anon well-formed-elections* π
      $\otimes_{\ BijGroup\ well\text{-}formed\text{-}elections}\ \varphi$*-anon well-formed-elections* $\pi')\ E$
  by *metis*
thus φ*-anon well-formed-elections* $(\pi \otimes_{\ BijGroup\ UNIV}\ \pi') =$
  φ*-anon well-formed-elections* π
    $\otimes_{\ BijGroup\ well\text{-}formed\text{-}elections}\ \varphi$*-anon well-formed-elections* π'
  by *blast*
qed

lemma (**in** *result*) *anonymity-action-presv-symmetry*: *is-symmetry* $(\lambda\ E.$ *limit*
  $($*alternatives-*$\mathcal{E}\ E)$ *UNIV*$)$ $($*Invariance* $($*anonymity*$_{\mathcal{R}}$ *well-formed-elections*$))$
  unfolding *anonymity*$_{\mathcal{R}}$*.simps*
  by *clarsimp*

1.10.4 Neutrality Lemmas

lemma *rel-rename-helper*:
  fixes
    $r :: 'a\ rel$ **and**
    $\pi :: 'a \Rightarrow 'a$ **and**
    $a\ b :: 'a$
  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*)
  fix $x\ y :: 'a$
  assume
    $(x, y) \in r$ **and**
    $\pi\ a = \pi\ x$ **and**

$\pi\ b = \pi\ y$
thus $\exists\ x\ y.\ (a,\ b) = (x,\ y) \wedge (x,\ y) \in r$
using *assms bij-is-inj the-inv-f-f*
by *metis*
next
fix $x\ y :: 'a$
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 $\pi\ \pi' :: 'a \Rightarrow 'a$
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 $\ldots = \{(\pi\ a,\ \pi\ b) \mid a\ b.\ (a,\ b) \in$ *rel-rename* $\pi'\ r\}$
unfolding *rel-rename.simps*
by *blast*
finally show *rel-rename* $(\pi \circ \pi')\ r$ = (*rel-rename* $\pi \circ$ *rel-rename* π') 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 $a\ b :: 'a$
assume
$(a,\ b) \in$ *rel-rename* $\pi\ r$ **and**
$(b,\ a) \in$ *rel-rename* $\pi\ r$
then obtain
$c\ c'\ d\ d' :: 'a$ **where**

c-rel-d: $(c,\ d) \in r$ **and**
d′-rel-c′: $(d',\ c') \in r$ **and**
π_c-eq-a: $\pi\ c = a$ **and**
$\pi_c{}'$-eq-a: $\pi\ c' = a$ **and**
π_d-eq-b: $\pi\ d = b$ **and**
$\pi_d{}'$-eq-b: $\pi\ d' = b$
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 $a\ b$ :: $'a$
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: $\pi\ a \neq \pi\ b$ **and**
π_b-not-rel-π_a: $(\pi\ b,\ \pi\ a) \notin$ *rel-rename* $\pi\ r$
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 $(\pi\ a,\ \pi\ b) \in$ *rel-rename* $\pi\ r$
unfolding *rel-rename.simps*
by *blast*
next
fix $a\ b\ c$ :: $'a$
assume
$(a,\ b) \in$ *rel-rename* $\pi\ r$ **and**
$(b,\ c) \in$ *rel-rename* $\pi\ r$
then obtain
$d\ e\ s\ t$ :: $'a$ **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 *rename-subset*:
  fixes
    $r\ s$ :: $'a$ *rel* **and**
    $a\ b$ :: $'a$ **and**
    π :: $'a \Rightarrow\ 'a$
  assumes
    bij-π: *bij* π **and**
    rel-rename $\pi\ r$ = *rel-rename* $\pi\ s$ **and**
    $(a,\ b) \in r$
  shows $(a,\ b) \in s$
proof −
  have $(\pi\ a,\ \pi\ b) \in \{(\pi\ a,\ \pi\ b)\ |\ a\ b.\ (a,\ b) \in s\}$
    using *assms*
    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

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*)
  fix
    $r\ s$ :: $'a$ *rel* **and**

```
    a b :: 'a
  assume rename: rel-rename π r = rel-rename π s
  {
    moreover assume (a, b) ∈ r
    ultimately have (π a, π b) ∈ {(π a, π b) | a b. (a, b) ∈ s}
      unfolding rel-rename.simps
      by blast
    hence ∃ c d. (c, d) ∈ s ∧ π c = π a ∧ π d = π b
      by fastforce
    moreover have ∀ c d. π c = π d ⟶ c = d
      using bij-π bij-pointE
      by metis
    ultimately show subset: (a, b) ∈ s
      by blast
  }
  moreover assume (a, b) ∈ s
  ultimately show (a, b) ∈ r
    using rename rename-subset bij-π
    by (metis (no-types))
next
  fix r :: 'a rel
  have rel-rename π {((the-inv π) a, (the-inv π) b) | a b. (a, b) ∈ r} =
          {(π ((the-inv π) a), π ((the-inv π) b)) | a b. (a, b) ∈ r}
    by auto
  also have ... = {(a, b) | a b. (a, b) ∈ 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 ∃ s. r = rel-rename π s
    by blast
qed

lemma alternatives-rename-comp:
  fixes π π' :: 'a ⇒ 'a
  shows alternatives-rename π ∘ alternatives-rename π' =
          alternatives-rename (π ∘ π')
proof
  fix ℰ :: ('a, 'v) Election
  have (alternatives-rename π ∘ alternatives-rename π') ℰ =
      (π ` π' ` (alternatives-ℰ ℰ), voters-ℰ ℰ,
        (rel-rename π) ∘ (rel-rename π') ∘ (profile-ℰ ℰ))
    by (simp add: fun.map-comp)
  also have
    ... = ((π ∘ π') ` (alternatives-ℰ ℰ), voters-ℰ ℰ,
            (rel-rename (π ∘ π')) ∘ (profile-ℰ ℰ))
    using rel-rename-comp image-comp
    by metis
  also have ... = alternatives-rename (π ∘ π') ℰ
```

by *simp*
finally show
$(alternatives\text{-}rename\ \pi \circ alternatives\text{-}rename\ \pi')\ \mathcal{E} =$
$alternatives\text{-}rename\ (\pi \circ \pi')\ \mathcal{E}$
by *blast*
qed

lemma *alternatives-rename-sound*:
fixes
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a,\ 'v)\ Profile$ **and**
$\pi :: 'a \Rightarrow 'a$
assumes
bij-π: $bij\ \pi$ **and**
wf-elects: $(A,\ V,\ p) \in well\text{-}formed\text{-}elections$ **and**
renamed: $(A',\ V',\ p') = alternatives\text{-}rename\ \pi\ (A,\ V,\ p)$
shows $(A',\ V',\ p') \in well\text{-}formed\text{-}elections$
proof −
have
$A' = \pi\ `\ A$ **and**
$V = V'$
using *renamed*
by (*simp*, *simp*)
moreover from *this* **have** $\forall\ v \in V'.\ linear\text{-}order\text{-}on\ A\ (p\ v)$
using *wf-elects*
unfolding *well-formed-elections-def profile-def*
by *simp*
moreover have $\forall\ v \in V'.\ p'\ v = rel\text{-}rename\ \pi\ (p\ v)$
using *renamed*
by *simp*
ultimately have $\forall\ v \in V'.\ linear\text{-}order\text{-}on\ A'\ (p'\ v)$
unfolding *linear-order-on-def partial-order-on-def preorder-on-def*
using *bij-π rel-rename-sound bij-is-inj*
by *metis*
thus $(A',\ V',\ p') \in well\text{-}formed\text{-}elections$
unfolding *well-formed-elections-def profile-def*
by *simp*
qed

lemma *alternatives-rename-bij*:
fixes $\pi :: ('a \Rightarrow 'a)$
assumes *bij-π*: $bij\ \pi$
shows *bij-betw* (*alternatives-rename* π) *well-formed-elections well-formed-elections*
proof (*unfold bij-betw-def*, *safe*, *intro inj-onI*, *clarify*)
fix
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a,\ 'v)\ Profile$

assume
renamed: *alternatives-rename* π $(A,\ V,\ p)$ = *alternatives-rename* π $(A',\ V',\ p')$
hence
π-*eq-img-A-A'*: π ‘ $A = \pi$ ‘ A' **and**
rel-rename-eq: *rel-rename* $\pi \circ p$ = *rel-rename* $\pi \circ p'$
by (*simp*, *simp*)
hence (*the-inv* (*rel-rename* π)) $\circ$ *rel-rename* $\pi \circ p$ =
(*the-inv* (*rel-rename* π)) $\circ$ *rel-rename* $\pi \circ p'$
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*
hence
$A = A'$ **and**
$p = p'$
using *bij-*π π*-eq-img-A-A'* *bij-betw-imp-inj-on inj-image-eq-iff*
by (*metis*, *safe*)
thus $A = A' \wedge (V,\ p) = (V',\ p')$
using *renamed*
by *simp*
next
fix
$A\ A'$ :: *'a set* **and**
$V\ V'$ :: *'v set* **and**
$p\ p'$ :: *('a, 'v) Profile*
assume *renamed*: $(A',\ V',\ p')$ = *alternatives-rename* π $(A,\ V,\ p)$
hence *rewr*: $V = V' \wedge A' = \pi$ ‘ A
by *simp*
moreover assume $(A,\ V,\ p) \in$ *well-formed-elections*
ultimately have $\forall\ v \in V'$. *linear-order-on* A $(p\ v)$
unfolding *well-formed-elections-def profile-def*
by *simp*
moreover have $\forall\ v \in V'$. $p'\ v$ = *rel-rename* π $(p\ v)$
using *renamed*
by *simp*
ultimately have $\forall\ v \in V'$. *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 assms*
by *metis*
thus $(A',\ V',\ p') \in$ *well-formed-elections*
unfolding *well-formed-elections-def profile-def*
by *simp*
next
fix
A :: *'a set* **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assume *wf-elects*: $(A, V, p) \in$ *well-formed-elections*
have *rename-inv*:
alternatives-rename (*the-inv* π) $(A, V, p) =$
((*the-inv* π) ‘ A, V, *rel-rename* (*the-inv* π) $\circ$ p)
by *simp*
also have
alternatives-rename π ((*the-inv* π) ‘ A, V, *rel-rename* (*the-inv* π) $\circ$ p) $=$
(π ‘ (*the-inv* π) ‘ A, V, *rel-rename* π $\circ$ *rel-rename* (*the-inv* π) $\circ$ p)
by *auto*
also have $\ldots = (A, V,$ *rel-rename* (π $\circ$ *the-inv* π) $\circ$ 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 $\circ$ *the-inv* π) $\circ$ $p) = (A, V,$ *rel-rename id* $\circ$ 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) $(A, V, p) \in$ *well-formed-elections*
using *rename-inv wf-elects alternatives-rename-sound bij-*π *bij-betw-the-inv-into*
by (*metis* (*no-types*))
ultimately show $(A, V, p) \in$ *alternatives-rename* π ‘ *well-formed-elections*
using *image-eqI*
by *metis*
qed

interpretation *φ-neutral-action*: *group-action bijection*$_{\mathcal{AG}}$ *well-formed-elections*
φ-neutral well-formed-elections
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*
bijection$_{\mathcal{AG}}$*-def*, *intro conjI group-BijGroup*, *safe*)
fix π :: $'a \Rightarrow 'a$
assume *bij-carrier*: $\pi \in$ *carrier* (*BijGroup UNIV*)
hence
bij-betw (*φ-neutral well-formed-elections* π) *well-formed-elections well-formed-elections*
using *universal-set-carrier-imp-bij-group alternatives-rename-bij bij-betw-ext*
unfolding *φ-neutral.simps*
by *metis*
thus *bij-carrier-elect*:
φ-neutral well-formed-elections $\pi \in$ *carrier* (*BijGroup well-formed-elections*)
unfolding *φ-neutral.simps BijGroup-def Bij-def extensional-def*
by *simp*
fix π' :: $'a \Rightarrow 'a$
assume *bij-carrier'*: $\pi' \in$ *carrier* (*BijGroup UNIV*)
hence
bij-betw (*φ-neutral well-formed-elections* π') *well-formed-elections well-formed-elections*

using *universal-set-carrier-imp-bij-group alternatives-rename-bij bij-betw-ext*
unfolding *φ-neutral.simps*
by *metis*
hence *bij-carrier-elect$'$*:
φ-neutral well-formed-elections π' $\in$ carrier (BijGroup well-formed-elections)
unfolding *φ-neutral.simps BijGroup-def Bij-def extensional-def*
by *simp*
hence *carrier-elects*:
φ-neutral well-formed-elections π $\in$ carrier (BijGroup well-formed-elections)
$\wedge$ φ-neutral well-formed-elections π' $\in$ carrier (BijGroup well-formed-elections)
using *bij-carrier-elect*
by *metis*
hence *bij-betw (φ-neutral well-formed-elections π') well-formed-elections well-formed-elections*
unfolding *BijGroup-def Bij-def extensional-def*
by *auto*
hence *wf-closed$'$*:
φ-neutral well-formed-elections π' ‘ well-formed-elections $\subseteq$ well-formed-elections
using *bij-betw-imp-surj-on*
by *blast*
have *φ-neutral well-formed-elections π*
$\otimes_{\textit{BijGroup well-formed-elections}}$ φ-neutral well-formed-elections π' =
extensional-continuation
(φ-neutral well-formed-elections π $\circ$ φ-neutral well-formed-elections π')
well-formed-elections
using *carrier-elects rewrite-mult*
by *auto*
moreover have
$\forall$ $\mathcal{E}$ $\in$ well-formed-elections. extensional-continuation
(φ-neutral well-formed-elections π $\circ$ φ-neutral well-formed-elections π')
well-formed-elections $\mathcal{E}$ =
(φ-neutral well-formed-elections π $\circ$ φ-neutral well-formed-elections π') $\mathcal{E}$
by *simp*
moreover have
$\forall$ $\mathcal{E}$ $\in$ well-formed-elections.
(φ-neutral well-formed-elections π $\circ$ φ-neutral well-formed-elections π') $\mathcal{E}$ =
alternatives-rename π (alternatives-rename π' $\mathcal{E}$)
unfolding *φ-neutral.simps*
using *wf-closed$'$*
by *auto*
moreover have
$\forall$ $\mathcal{E}$ $\in$ well-formed-elections.
alternatives-rename π (alternatives-rename π' $\mathcal{E}$) =
alternatives-rename (π $\circ$ π') $\mathcal{E}$
using *alternatives-rename-comp comp-apply*
by *metis*
moreover have
$\forall$ $\mathcal{E}$ $\in$ well-formed-elections. alternatives-rename (π $\circ$ π') $\mathcal{E}$ =
φ-neutral well-formed-elections (π $\otimes_{\textit{BijGroup UNIV}}$ π') $\mathcal{E}$
using *rewrite-mult-univ bij-carrier bij-carrier$'$*

unfolding φ*-anon.simps* φ*-neutral.simps extensional-continuation.simps*
by *metis*
moreover have
$\forall\ \mathcal{E}.\ \mathcal{E} \notin$ *well-formed-elections* $\longrightarrow$
extensional-continuation
(φ*-neutral well-formed-elections* $\pi \circ \varphi$*-neutral well-formed-elections* π')
well-formed-elections $\mathcal{E}$ = *undefined*
by *simp*
moreover have
$\forall\ \mathcal{E}.\ \mathcal{E} \notin$ *well-formed-elections*
$\longrightarrow \varphi$*-neutral well-formed-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')\ \mathcal{E}$ = *undefined*
by *simp*
ultimately have
$\forall\ \mathcal{E}.\ \varphi$*-neutral well-formed-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')\ \mathcal{E}$ =
(φ*-neutral well-formed-elections* π
$\otimes_{\mathit{BijGroup\ well\text{-}formed\text{-}elections}}$ φ*-neutral well-formed-elections* π') $\mathcal{E}$
by *metis*
thus
φ*-neutral well-formed-elections* $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ =
φ*-neutral well-formed-elections* π
$\otimes_{\mathit{BijGroup\ well\text{-}formed\text{-}elections}}$ φ*-neutral well-formed-elections* π'
by *blast*
qed

interpretation ψ*-neutral*$_{\mathrm{c}}$*-action*: *group-action bijection*$_{\mathcal{AG}}$ *UNIV* ψ*-neutral*$_{\mathrm{c}}$
proof (*unfold group-action-def group-hom-def hom-def bijection*$_{\mathcal{AG}}$*-def*
group-hom-axioms-def, *intro conjI group-BijGroup*, *safe*)
fix $\pi :: 'a \Rightarrow 'a$
assume $\pi \in$ *carrier* (*BijGroup UNIV*)
hence *bij* π
unfolding *BijGroup-def Bij-def*
by *simp*
thus ψ*-neutral*$_{\mathrm{c}}$ $\pi \in$ *carrier* (*BijGroup UNIV*)
unfolding ψ*-neutral*$_{\mathrm{c}}$*.simps*
using *rewrite-carrier*
by *blast*
fix $\pi' :: 'a \Rightarrow 'a$
show ψ*-neutral*$_{\mathrm{c}}$ $(\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi')$ =
ψ*-neutral*$_{\mathrm{c}}$ $\pi \otimes_{\mathit{BijGroup\ UNIV}}$ ψ*-neutral*$_{\mathrm{c}}$ π'
unfolding ψ*-neutral*$_{\mathrm{c}}$*.simps*
by *safe*
qed

interpretation ψ*-neutral*$_{\mathrm{w}}$*-action*: *group-action bijection*$_{\mathcal{AG}}$ *UNIV* ψ*-neutral*$_{\mathrm{w}}$
proof (*unfold group-action-def group-hom-def hom-def bijection*$_{\mathcal{AG}}$*-def*
group-hom-axioms-def, *intro conjI group-BijGroup*, *safe*)
fix $\pi :: 'a \Rightarrow 'a$
assume *bij-carrier*: $\pi \in$ *carrier* (*BijGroup UNIV*)
hence *bij* π

unfolding *bijection*$_{\mathcal{AG}}$*-def BijGroup-def Bij-def*
by *simp*
hence *bij* (ψ*-neutral*$_{\mathrm{w}}$ π)
unfolding *bijection*$_{\mathcal{AG}}$*-def BijGroup-def Bij-def* ψ*-neutral*$_{\mathrm{w}}$*.simps*
using *rel-rename-bij*
by *blast*
thus *group-elem*: ψ*-neutral*$_{\mathrm{w}}$ $\pi \in$ *carrier* (*BijGroup UNIV*)
using *rewrite-carrier*
by *blast*
moreover fix $\pi' :: 'a \Rightarrow 'a$
assume *bij-carrier'*: $\pi' \in$ *carrier* (*BijGroup UNIV*)
hence *bij* π'
unfolding *bijection*$_{\mathcal{AG}}$*-def BijGroup-def Bij-def*
by *simp*
hence *bij* (ψ*-neutral*$_{\mathrm{w}}$ π')
unfolding *bijection*$_{\mathcal{AG}}$*-def BijGroup-def Bij-def* ψ*-neutral*$_{\mathrm{w}}$*.simps*
using *rel-rename-bij*
by *blast*
hence *group-elem'*: ψ*-neutral*$_{\mathrm{w}}$ $\pi' \in$ *carrier* (*BijGroup UNIV*)
using *rewrite-carrier*
by *blast*
moreover have ψ*-neutral*$_{\mathrm{w}}$ ($\pi \otimes_{BijGroup\ UNIV} \pi'$) = ψ*-neutral*$_{\mathrm{w}}$ ($\pi \circ \pi'$)
using *bij-carrier bij-carrier' rewrite-mult-univ*
by *metis*
ultimately show
ψ*-neutral*$_{\mathrm{w}}$ ($\pi \otimes_{BijGroup\ UNIV} \pi'$) =
ψ*-neutral*$_{\mathrm{w}}$ $\pi \otimes_{BijGroup\ UNIV}$ ψ*-neutral*$_{\mathrm{w}}$ π'
using *rewrite-mult-univ*
by *fastforce*
qed

lemma *neutrality-action-presv-*$\mathcal{SCF}$*-symmetry*: *is-symmetry* ($\lambda\, \mathcal{E}$. *limit-*$\mathcal{SCF}$ (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$) *well-formed-elections*
(φ*-neutral well-formed-elections*) (*set-action* ψ*-neutral*$_{\mathrm{c}}$))
proof (*unfold rewrite-equivariance, safe*)
fix
$\pi :: 'a \Rightarrow 'a$ **and**
$A :: 'a$ *set* **and**
$V :: 'v$ *set* **and**
$p :: 'v \Rightarrow ('a \times 'a)$ *set* **and**
$r :: 'a$
assume
carrier-π: $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$ **and**
prof: $(A, V, p) \in$ *well-formed-elections*
{
moreover assume
$r \in$ *limit-*$\mathcal{SCF}$
(*alternatives-*$\mathcal{E}$ (φ*-neutral well-formed-elections* π (A, V, p))) *UNIV*

ultimately show
$r \in$ *set-action* ψ*-neutral*$_\mathrm{c}$ π (*limit-*$\mathcal{SCF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*)
by *auto*
}
{
moreover assume
$r \in$ *set-action* ψ*-neutral*$_\mathrm{c}$ π (*limit-*$\mathcal{SCF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*)
ultimately show
$r \in$ *limit-*$\mathcal{SCF}$
(*alternatives-*$\mathcal{E}$ (φ*-neutral well-formed-elections* π (A, V, p))) *UNIV*
using *prof*
by *simp*
}
qed

lemma *neutrality-action-presv-*$\mathcal{SWF}$*-symmetry*: *is-symmetry* (λ $\mathcal{E}$. *limit-*$\mathcal{SWF}$
(*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*) (*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$) *well-formed-elections*
(φ*-neutral well-formed-elections*) (*set-action* ψ*-neutral*$_\mathrm{w}$))
proof (*unfold rewrite-equivariance voters-*$\mathcal{E}$*.simps profile-*$\mathcal{E}$*.simps set-action.simps*,
safe)
show $\bigwedge$ π A V p r.
$\pi \in$ *carrier bijection*$_{\mathcal{AG}}$ $\Longrightarrow$ (A, V, p) $\in$ *well-formed-elections*
$\Longrightarrow$ $r \in$ *limit-*$\mathcal{SWF}$
(*alternatives-*$\mathcal{E}$ (φ*-neutral well-formed-elections* π (A, V , p))) *UNIV*
$\Longrightarrow$ $r \in$ ψ*-neutral*$_\mathrm{w}$ π ‘ *limit-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*
proof −
fix
π :: $'c \Rightarrow 'c$ **and**
A :: $'c$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'c$, $'v$) *Profile* **and**
r :: $'c$ *rel*
let *?r-inv* = ψ*-neutral*$_\mathrm{w}$ (*the-inv* π) r
assume
carrier-π: $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$ **and**
prof: (A, V, p) $\in$ *well-formed-elections*
have *inv-carrier*: *the-inv* $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$
using *carrier-*π *bij-betw-the-inv-into*
unfolding *bijection*$_{\mathcal{AG}}$*-def rewrite-carrier*
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 *bijection*$_{\mathcal{AG}}$*-def*
by *fastforce*
moreover have $\mathbf{1}$ $_{bijection_{\mathcal{AG}}}$ = *id*
unfolding *bijection*$_{\mathcal{AG}}$*-def BijGroup-def*
by *auto*
ultimately have *the-inv* $\pi \otimes_{bijection_{\mathcal{AG}}} \pi = \mathbf{1}_{bijection_{\mathcal{AG}}}$
using *carrier-*π *rewrite-mult-univ*

unfolding *bijection*$_{\mathcal{AG}}$*-def*
by *metis*
hence *inv* $_{bijection_{\mathcal{AG}}}$ π = *the-inv* π
using *carrier-*π *inv-carrier* ψ*-neutral*$_{c}$*-action.group-hom group.inv-closed group.inv-solve-right group.l-inv group-BijGroup group-hom.hom-one group-hom.one-closed*
unfolding *bijection*$_{\mathcal{AG}}$*-def*
by *metis*
hence *neutral-r*: r = ψ*-neutral*$_{w}$ π *?r-inv*
using *carrier-*π *inv-carrier iso-tuple-UNIV-I* ψ*-neutral*$_{w}$*-action.orbit-sym-aux*
by *metis*
have *bij-inv*: *bij* (*the-inv* π)
using *carrier-*π *bij-betw-the-inv-into universal-set-carrier-imp-bij-group*
unfolding *bijection*$_{\mathcal{AG}}$*-def*
by *blast*
hence *the-inv-*π: (*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 *bijection*$_{\mathcal{AG}}$*-def*
by *metis*
assume
r ∈ *limit-*$\mathcal{SWF}$
(*alternatives-*$\mathcal{E}$ (φ*-neutral well-formed-elections* π (A, V, p))) *UNIV*
hence r ∈ *limit-*$\mathcal{SWF}$ (π ‘ A) *UNIV*
unfolding φ*-neutral.simps*
using *prof*
by *simp*
hence *linear-order-on* (π ‘ A) r
by *auto*
hence *lin-inv*: *linear-order-on* A *?r-inv*
using *rel-rename-sound bij-inv bij-is-inj the-inv-*π
unfolding ψ*-neutral*$_{w}$*.simps linear-order-on-def preorder-on-def partial-order-on-def*
by *metis*
hence ∀ (a, b) ∈ *?r-inv*. $a \in A \wedge b \in A$
unfolding *linear-order-on-def partial-order-on-def preorder-on-def*
using *refl-on-def'*
by *metis*
hence *limit* A *?r-inv* = {(a, b). (a, b) ∈ *?r-inv*}
by *auto*
also have . . . = *?r-inv*
by *blast*
finally have . . . = *limit* A *?r-inv*
by *blast*
hence *?r-inv* ∈ *limit-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*
unfolding *limit-*$\mathcal{SWF}$*.simps alternatives-*$\mathcal{E}$*.simps*
using *lin-inv UNIV-I fst-conv mem-Collect-eq iso-tuple-UNIV-I CollectI*
by (*metis* (*mono-tags*, *lifting*))
thus *lim-el-*π:
r ∈ ψ*-neutral*$_{w}$ π ‘ *limit-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ (A, V, p)) *UNIV*

```
      using neutral-r
      by blast
  qed
  moreover
  fix
    π :: 'a ⇒ 'a and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    r :: 'a rel
  assume
    carrier-π: π ∈ carrier bijection_𝒜𝒢 and
    prof: (A, V, p) ∈ well-formed-elections
  hence prof-π:
    φ-neutral well-formed-elections π (A, V, p) ∈ well-formed-elections
    using φ-neutral-action.element-image
    by blast
  moreover have inv-group-elem: inv bijection_𝒜𝒢 π ∈ carrier bijection_𝒜𝒢
    using carrier-π ψ-neutral_c-action.group-hom group.inv-closed
    unfolding group-hom-def
    by metis
  moreover have φ-neutral well-formed-elections (inv bijection_𝒜𝒢 π)
        (φ-neutral well-formed-elections π (A, V, p)) ∈ well-formed-elections
    using prof φ-neutral-action.element-image inv-group-elem prof-π
    by metis
  moreover assume r ∈ limit-𝒮𝒲ℱ (alternatives-ℰ (A, V, p)) UNIV
  hence r ∈ limit-𝒮𝒲ℱ
      (alternatives-ℰ (φ-neutral well-formed-elections (inv bijection_𝒜𝒢 π)
        (φ-neutral well-formed-elections π (A, V, p)))) UNIV
    using φ-neutral-action.orbit-sym-aux carrier-π prof
    by metis
  ultimately have
    r ∈ ψ-neutral_w (inv bijection_𝒜𝒢 π) `
      limit-𝒮𝒲ℱ
        (alternatives-ℰ (φ-neutral well-formed-elections π (A, V, p))) UNIV
    using prod.collapse
    by metis
  thus ψ-neutral_w π r ∈ limit-𝒮𝒲ℱ
          (alternatives-ℰ (φ-neutral well-formed-elections π (A, V, p))) UNIV
    using carrier-π ψ-neutral_w-action.group-action-axioms
          ψ-neutral_w-action.inj-prop group-action.orbit-sym-aux
          inj-image-mem-iff inv-group-elem iso-tuple-UNIV-I
    by (metis (no-types, lifting))
qed
```

1.10.5 Homogeneity Lemmas

```
definition reflp-on' :: 'a set ⇒ 'a rel ⇒ bool where
    reflp-on' A r ≡ reflp-on A (λ x y. (x, y) ∈ r)
```

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

lemma (**in** *result*) *homogeneity-action-presv-symmetry*:
  is-symmetry (λ $\mathcal{E}$. *limit* (*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-elections-*$\mathcal{V}$
  shows *reflp-on'* $\mathcal{E}$ (*homogeneity*$_{\mathcal{R}}$*'* $\mathcal{E}$)
  using *assms*
  unfolding *homogeneity*$_{\mathcal{R}}$*'.simps reflp-on'-def reflp-on-def finite-elections-*$\mathcal{V}$*-def*
  by *auto*

lemma (**in** *result*) *homogeneity'-action-presv-symmetry*:
  is-symmetry (λ $\mathcal{E}$. *limit* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
    (*Invariance* (*homogeneity*$_{\mathcal{R}}$*'* *UNIV*))
  by *simp*

1.10.6 Reversal Symmetry Lemmas

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

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

lemma *reverse-rel-lin-ord*:
  fixes
    A :: *'a set* **and**
    r :: *'a rel*
  assumes *linear-order-on A r*
  shows *linear-order-on A* (*reverse-rel r*)
  using *assms*
  unfolding *reverse-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 $reversal_{\mathcal{G}}$-*group*: *group* $reversal_{\mathcal{G}}$
proof
show $\mathbf{1}_{\ reversal_{\mathcal{G}}} \in carrier\ reversal_{\mathcal{G}}$
unfolding $reversal_{\mathcal{G}}$-*def*
by *simp*
next
show $carrier\ reversal_{\mathcal{G}} \subseteq Units\ reversal_{\mathcal{G}}$
unfolding $reversal_{\mathcal{G}}$-*def* *Units-def*
using *reverse-reverse-id*
by *auto*
next
fix $\alpha :: 'a\ rel \Rightarrow 'a\ rel$
show $\alpha \otimes_{\ reversal_{\mathcal{G}}} \mathbf{1}_{\ reversal_{\mathcal{G}}} = \alpha$
unfolding $reversal_{\mathcal{G}}$-*def*
by *auto*
assume α-*elem*: $\alpha \in carrier\ reversal_{\mathcal{G}}$
thus $\mathbf{1}_{\ reversal_{\mathcal{G}}} \otimes_{\ reversal_{\mathcal{G}}} \alpha = \alpha$
unfolding $reversal_{\mathcal{G}}$-*def*
by *auto*
fix $\alpha' :: 'a\ rel \Rightarrow 'a\ rel$
assume α'-*elem*: $\alpha' \in carrier\ reversal_{\mathcal{G}}$
thus $\alpha \otimes_{\ reversal_{\mathcal{G}}} \alpha' \in carrier\ reversal_{\mathcal{G}}$
using α-*elem* *reverse-reverse-id*
unfolding $reversal_{\mathcal{G}}$-*def*
by *auto*
fix $z :: 'a\ rel \Rightarrow 'a\ rel$
assume $z \in carrier\ reversal_{\mathcal{G}}$
thus $\alpha \otimes_{\ reversal_{\mathcal{G}}} \alpha' \otimes_{\ reversal_{\mathcal{G}}} z = \alpha \otimes_{\ reversal_{\mathcal{G}}} (\alpha' \otimes_{\ reversal_{\mathcal{G}}} z)$
using α-*elem* α'-*elem*
unfolding $reversal_{\mathcal{G}}$-*def*
by *auto*
qed

interpretation φ-*reverse-action*: *group-action* $reversal_{\mathcal{G}}$ *well-formed-elections* φ-*reverse* *well-formed-elections*
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def*, *intro conjI group-BijGroup CollectI ballI funcsetI*)
show *Group.group* $reversal_{\mathcal{G}}$
by *safe*
next
show *carrier-elect-gen*:
$\bigwedge \pi.\ \pi \in carrier\ reversal_{\mathcal{G}}$
$\Longrightarrow \varphi$-*reverse* *well-formed-elections* $\pi \in carrier\ (BijGroup\ well\text{-}formed\text{-}elections)$
proof $-$
fix $\pi :: 'c\ rel \Rightarrow 'c\ rel$
assume $\pi \in carrier\ reversal_{\mathcal{G}}$
hence π-*cases*: $\pi \in \{id,\ reverse\text{-}rel\}$

unfolding *reversal*$_\mathcal{G}$*-def*
by *auto*
hence [*simp*]: *rel-app* $\pi \circ$ *rel-app* π = *id*
using *reverse-reverse-id*
by *fastforce*
have $\forall$ $\mathcal{E}$. *rel-app* π (*rel-app* π $\mathcal{E}$) = $\mathcal{E}$
by (*simp add*: *pointfree-idE*)
moreover have $\forall$ $\mathcal{E}$ $\in$ *well-formed-elections*. *rel-app* π $\mathcal{E}$ $\in$ *well-formed-elections*
unfolding *well-formed-elections-def profile-def*
using π*-cases reverse-rel-lin-ord rel-app.simps fun.map-id*
by *fastforce*
hence *rel-app* π ' *well-formed-elections* $\subseteq$ *well-formed-elections*
by *blast*
ultimately have *bij-betw* (*rel-app* π) *well-formed-elections well-formed-elections*
using *bij-betw-byWitness*[*of well-formed-elections*]
by *blast*
hence *bij-betw* (φ*-reverse well-formed-elections* π)
well-formed-elections well-formed-elections
unfolding φ*-reverse.simps*
using *bij-betw-ext*
by *blast*
moreover have φ*-reverse well-formed-elections* π $\in$ *extensional well-formed-elections*
unfolding *extensional-def*
by *simp*
ultimately show
φ*-reverse well-formed-elections* π $\in$ *carrier* (*BijGroup well-formed-elections*)
unfolding *BijGroup-def Bij-def*
by *simp*
qed
moreover fix π π' :: *'a rel* $\Rightarrow$ *'a rel*
assume
rev: π $\in$ *carrier reversal*$_\mathcal{G}$ **and**
rev': π' $\in$ *carrier reversal*$_\mathcal{G}$
ultimately have *carrier-elect*:
φ*-reverse well-formed-elections* π $\in$ *carrier* (*BijGroup well-formed-elections*)
by *blast*
have φ*-reverse well-formed-elections* ($\pi \otimes_{reversal_\mathcal{G}} \pi'$) =
extensional-continuation (*rel-app* ($\pi \circ \pi'$)) *well-formed-elections*
unfolding *reversal*$_\mathcal{G}$*-def*
by *simp*
moreover have *rel-app* ($\pi \circ \pi'$) = *rel-app* $\pi \circ$ *rel-app* π'
using *rel-app.simps*
by *fastforce*
ultimately have
φ*-reverse well-formed-elections* ($\pi \otimes_{reversal_\mathcal{G}} \pi'$) =
extensional-continuation (*rel-app* $\pi \circ$ *rel-app* π') *well-formed-elections*
by *metis*
moreover have
$\forall$ *A V p*. $\forall$ *v* $\in$ *V*. *linear-order-on A* (*p v*) $\longrightarrow$ *linear-order-on A* (π' (*p v*))

```
    using empty-iff id-apply insert-iff rev' reverse-rel-lin-ord
    unfolding partial-object.simps reversal_G-def
    by metis
  hence extensional-continuation
      (φ-reverse well-formed-elections π ∘ φ-reverse well-formed-elections π')
        well-formed-elections =
          extensional-continuation (rel-app π ∘ rel-app π') well-formed-elections
    unfolding well-formed-elections-def profile-def
    by fastforce
  moreover have extensional-continuation
      (φ-reverse well-formed-elections π ∘ φ-reverse well-formed-elections π')
        well-formed-elections =
      φ-reverse well-formed-elections π
          ⊗ BijGroup well-formed-elections φ-reverse well-formed-elections π'
    using carrier-elect-gen carrier-elect rev' rewrite-mult
    by metis
  ultimately show
    φ-reverse well-formed-elections (π ⊗ reversal_G π') =
      φ-reverse well-formed-elections π
          ⊗ BijGroup well-formed-elections φ-reverse well-formed-elections π'
    by metis
qed

interpretation ψ-reverse-action: group-action reversal_G UNIV ψ-reverse
proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def ψ-reverse.simps,
      intro conjI group-BijGroup CollectI ballI funcsetI)
  show Group.group reversal_G
    by safe
next
  fix π :: 'a rel ⇒ 'a rel
  assume π ∈ carrier reversal_G
  hence π ∈ {id, reverse-rel}
    unfolding reversal_G-def
    by force
  hence bij π
    using reverse-reverse-id bij-id insertE o-bij singleton-iff
    by metis
  thus π ∈ carrier (BijGroup UNIV)
    using rewrite-carrier
    by blast
next
  fix π π' :: 'a rel ⇒ 'a rel
  assume
    π ∈ carrier reversal_G and
    π' ∈ carrier reversal_G
  hence bij π' ∧ bij π
    using singleton-iff comp-apply id-apply involuntory-imp-bij reverse-reverse-id
    unfolding bij-id insert-iff reversal_G-def partial-object.select-convs
    by (metis (mono-tags, opaque-lifting))
```

hence $\pi \otimes_{\mathit{BijGroup\ UNIV}} \pi' = \pi \circ \pi'$
using *rewrite-carrier rewrite-mult-univ*
by *blast*
also have $\ldots = \pi \otimes_{\mathit{reversal}_{\mathcal{G}}} \pi'$
unfolding *reversal*$_{\mathcal{G}}$*-def*
by *force*
finally show $\pi \otimes_{\mathit{reversal}_{\mathcal{G}}} \pi' = \pi \otimes_{\mathit{BijGroup\ UNIV}} \pi'$
by *presburger*
qed

lemma *reversal-symmetry-action-presv-symmetry*: *is-symmetry* (λ $\mathcal{E}$. *limit-*$\mathcal{SWF}$ (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*action-induced-equivariance* (*carrier reversal*$_{\mathcal{G}}$) *well-formed-elections*
(φ*-reverse well-formed-elections*) (*set-action* ψ*-reverse*))
proof (*unfold rewrite-equivariance*, *clarify*)
fix
π :: *'a rel* $\Rightarrow$ *'a rel* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assume $\pi \in$ *carrier reversal*$_{\mathcal{G}}$
hence *cases*: $\pi \in \{$*id*, *reverse-rel*$\}$
unfolding *reversal*$_{\mathcal{G}}$*-def*
by *force*
assume (*A*, *V*, *p*) $\in$ *well-formed-elections*
hence *eq-A*:
alternatives-$\mathcal{E}$ (φ*-reverse well-formed-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*. *reverse-rel r* = *limit A* (*reverse-rel r'*)
$\wedge$ *reverse-rel r'* $\in$ *UNIV* $\wedge$ *linear-order-on A* (*limit A* (*reverse-rel r'*))
using *reverse-rel-limit*[*of A*] *reverse-rel-lin-ord*
by *force*
hence
$\forall$ *r* $\in$ $\{$*limit A r* | *r*. *r* $\in$ *UNIV* $\wedge$ *linear-order-on A* (*limit A r*)$\}$.
reverse-rel r $\in$ $\{$*limit A* (*reverse-rel r'*)
| *r'*. *reverse-rel r'* $\in$ *UNIV*
$\wedge$ *linear-order-on A* (*limit A* (*reverse-rel r'*))$\}$
by *blast*
moreover have
$\{$*limit A* (*reverse-rel r'*) |
r'. *reverse-rel r'* $\in$ *UNIV* $\wedge$ *linear-order-on A* (*limit A* (*reverse-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-*$\mathcal{SWF}$ *A UNIV*. *reverse-rel r* $\in$ *limit-*$\mathcal{SWF}$ *A UNIV*
unfolding *limit-*$\mathcal{SWF}$*.simps*
by *blast*

hence *subset*:
  $\forall\ r \in$ *limit-$\mathcal{SWF}$ A UNIV*. $\pi\ r \in$ *limit-$\mathcal{SWF}$ A UNIV*
  using *cases*
  by *fastforce*
hence $\forall\ r \in$ *limit-$\mathcal{SWF}$ A UNIV*. $r \in \pi$ ` *limit-$\mathcal{SWF}$ A UNIV*
 using *reverse-reverse-id comp-apply empty-iff id-apply image-eqI insert-iff cases*
  by *metis*
hence π ` *limit-$\mathcal{SWF}$ A UNIV* = *limit-$\mathcal{SWF}$ A UNIV*
  using *subset*
  by *blast*
hence *set-action ψ-reverse* π (*limit-$\mathcal{SWF}$ A UNIV*) = *limit-$\mathcal{SWF}$ A UNIV*
  unfolding *set-action.simps*
  by *simp*
also have
  ... = *limit-$\mathcal{SWF}$*
      (*alternatives-$\mathcal{E}$* (*φ-reverse well-formed-elections* π (A, V, p))) *UNIV*
  using *eq-A*
  by *simp*
finally show
 limit-$\mathcal{SWF}$ (*alternatives-$\mathcal{E}$* (*φ-reverse well-formed-elections* π (A, V, p))) *UNIV*
=
    set-action ψ-reverse π (*limit-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* (A, V, p)) *UNIV*)
  by *simp*
qed

end

1.11 Result-Dependent Voting Rule Properties

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

1.11.1 Property Definitions

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 :: ($'a \Rightarrow 'a$, $'b$) *binary-fun* **and**
    ν :: $'v$ *itself*
 assumes

action-neutral: *group-action bijection*$_{\mathcal{AG}}$ *UNIV* ψ **and**
neutrality:
is-symmetry (λ $\mathcal{E}$ :: ('*a*, '*v*) *Election*. *limit* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$)
well-formed-elections
(φ*-neutral well-formed-elections*) (*set-action* ψ))

sublocale *result-properties* $\subseteq$ *result*
using *result-axioms*
by *safe*

1.11.2 Interpretations

global-interpretation $\mathcal{SCF}$*-properties*: *result-properties well-formed-*$\mathcal{SCF}$
limit-$\mathcal{SCF}$ ψ*-neutral*$_{\mathrm{c}}$
unfolding *result-properties-def result-properties-axioms-def*
using *neutrality-action-presv-*$\mathcal{SCF}$*-symmetry* ψ*-neutral*$_{\mathrm{c}}$*-action.group-action-axioms*
$\mathcal{SCF}$*-result.result-axioms*
by *blast*

global-interpretation $\mathcal{SWF}$*-properties*: *result-properties well-formed-*$\mathcal{SWF}$
limit-$\mathcal{SWF}$ ψ*-neutral*$_{\mathrm{w}}$
unfolding *result-properties-def result-properties-axioms-def*
using *neutrality-action-presv-*$\mathcal{SWF}$*-symmetry* ψ*-neutral*$_{\mathrm{w}}$*-action.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
    $f\ g :: {'a} \Rightarrow {'b} :: ord$ **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* $x \notin S$)
  case *True*
  thus $(x \in S \land (\nexists y.\ y \in S \land f\ y < f\ x)) = (x \in S \land (\nexists y.\ y \in S \land g\ y < g\ x))$
    by *safe*
next
  case *x-in-S*: *False*
  thus $(x \in S \land (\nexists\ y.\ y \in S \land f\ y < f\ x)) = (x \in S \land (\nexists\ y.\ y \in S \land g\ y < g\ x))$

```
  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 ∈ S and
        g y < g x
      moreover have ∀ a ∈ S. f a = g a
        using assms
        by simp
      moreover from this have g x = f x
        using x-in-S
        by metis
      ultimately show False
        using not-y
        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 | a l. a ∈ A ∧ l ∈ S}
proof −
  let ?P = λ A. finite {a#l | a l. a ∈ A ∧ l ∈ S}
  have ∀ a A'. finite A' ⟶ a ∉ A' ⟶ ?P A' ⟶ ?P (insert a A')
  proof (safe)
    fix
      a :: 'a and
```

```
      A' :: 'a set
    assume finite {a#l | a l. a ∈ A' ∧ l ∈ S}
    moreover have
      {a'#l | a' l. a' ∈ insert a A' ∧ l ∈ S} =
          {a#l | a l. a ∈ A' ∧ l ∈ S} ∪ {a#l | l. l ∈ S}
      by blast
    moreover have finite {a#l | l. l ∈ S}
      using fin-B
      by simp
    ultimately show ?P (insert a A')
      by simp
  qed
  thus ?P A
    using finite-induct[of - ?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)
  case Nil
  show finite (listset [])
    by simp
next
  case (Cons a l)
  fix
    a :: 'a set and
    l :: 'a set list
  assume ∀ i :: nat < length (a#l). finite ((a#l)!i)
  hence
    finite a and
    ∀ i < length l. finite (l!i)
    by auto
  moreover assume
    ∀ i :: nat < length l. finite (l!i) ⟹ finite (listset l)
  ultimately have
    finite (listset l) and
    finite {a'#l' | a' l'. a' ∈ a ∧ l' ∈ (listset l)}
    using list-cons-presv-finiteness
    by (blast, blast)
  thus finite (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 $(induct\ l,\ safe)$
case Nil
fix $l :: 'a\ list$
assume $l \in listset\ []$
thus $length\ l = length\ []$
by $simp$
next
case $(Cons\ a\ l)$
moreover fix
$a :: 'a\ set$ **and**
$l :: 'a\ set\ list$ **and**
$m :: 'a\ list$
assume
$\forall\ l'.\ l' \in listset\ l \longrightarrow length\ l' = length\ l$ **and**
$m \in listset\ (a\#l)$
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 $(simp\ add{:}\ set\text{-}Cons\text{-}def)$
ultimately show $length\ m = length\ (a\#l)$
by $force$
qed

lemma $all\text{-}ls\text{-}elems\text{-}in\text{-}ls\text{-}set$:
fixes $l :: 'a\ set\ list$
shows $\forall\ l' \in listset\ l.\ \forall\ i :: nat < length\ l'.\ l'!i \in l!i$
proof $(induct\ l,\ safe)$
case Nil
fix
$l' :: 'a\ list$ **and**
$i :: nat$
assume
$l' \in listset\ []$ **and**
$i < length\ l'$
thus $l'!i \in []!i$
by $simp$
next
case $(Cons\ a\ l)$
moreover fix
$a :: 'a\ set$ **and**
$l :: 'a\ set\ list$ **and**
$l' :: 'a\ list$ **and**
$i :: nat$
assume
$\forall\ l' \in listset\ l.\ \forall\ i :: nat < length\ l'.\ l'!i \in l!i$ **and**
$l' \in listset\ (a\#l)$ **and**
$i < length\ l'$
moreover from $this$ **have** $l' \in set\text{-}Cons\ a\ (listset\ l)$

```
    by simp
  hence ∃ b m. l' = b#m ∧ b ∈ a ∧ m ∈ (listset l)
    unfolding set-Cons-def
    by simp
  ultimately show l'!i ∈ (a#l)!i
    using 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)
  case Nil
  fix l' :: 'a list
  assume length l' = length []
  thus l' ∈ listset []
    by simp
next
  case (Cons a l)
  fix
    l :: 'a set list and
    l' :: 'a list and
    s :: 'a set
  assume length l' = length (s#l)
  moreover then obtain
    t :: 'a list and
    x :: 'a where
    l'-cons: l' = x#t
    using length-Suc-conv
    by metis
  moreover assume
    ∀ m. length m = length l ∧ (∀ i < length m. m!i ∈ l!i)
          ⟶ m ∈ listset l and
    ∀ i < length l'. l'!i ∈ (s#l)!i
  ultimately have
    x ∈ s and
    t ∈ listset l
    using diff-Suc-1 diff-Suc-eq-diff-pred zero-less-diff
          zero-less-Suc length-Cons
    by (metis nth-Cons-0, metis nth-Cons-Suc)
  thus 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* ∈ *set l* ∧ *b* ∈ *set l* ∧ *index l a* ≥ *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 *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 $r \in$ *pl-α* ' *permutations-of-set* A
shows *linear-order-on* A r
proof (*cases* $A = \{\}$, *unfold linear-order-on-def total-on-def partial-order-on-def antisym-def preorder-on-def*, *intro conjI impI allI ballI*)
case *True*
fix x y :: *'a*
show
refl-on A r **and**
trans r **and**
$(x, y) \in r \Longrightarrow x = y$ **and**
$x \in A \Longrightarrow (x, y) \in r \lor (y, x) \in r$
using *assms True*
unfolding *pl-α-def*
by (*simp*, *simp*, *simp*, *simp*)
next
case *False*
fix x y :: *'a*
show ((*refl-on* A r $\land$ *trans* r)
$\land$ ($\forall$ x y. $(x, y) \in r \longrightarrow (y, x) \in r \longrightarrow x = y$))
$\land$ ($\forall$ $x \in A$. $\forall$ $y \in A$. $x \neq y \longrightarrow (x, y) \in r \lor (y, x) \in r$)
proof (*intro conjI ballI allI impI*)
have $\forall$ $l \in$ *permutations-of-set* A. $l \neq []$
using *assms False permutations-of-setD*
by *force*
hence $\forall$ $a \in A$. $\forall$ $l \in$ *permutations-of-set* A. $(a, a) \in$ *pl-α* l
unfolding *is-less-preferred-than-l.simps permutations-of-set-def pl-α-def*
by *simp*
hence $\forall$ $a \in A$. $(a, a) \in r$
using *assms*
by *blast*
moreover have $r \subseteq A \times A$
using *assms*
unfolding *pl-α-def permutations-of-set-def*
by *auto*
ultimately show *refl-on* A r
unfolding *refl-on-def*
by *safe*
next
show *trans* r
using *assms rel-trans*

```
      by safe
  next
    fix x y :: 'a
    assume
      (x, y) ∈ r and
      (y, x) ∈ r
    moreover have
      ∀ x y. ∀ l ∈ permutations-of-set A. x ≲l y ∧ y ≲l x ⟶ x = y
      using is-less-preferred-than-l.simps index-eq-index-conv nle-le
      unfolding permutations-of-set-def
      by metis
    hence ∀ x y. ∀ l ∈ pl-α ‘ permutations-of-set A.
              (x, y) ∈ l ∧ (y, x) ∈ l ⟶ x = y
      unfolding pl-α-def permutations-of-set-def antisym-on-def
      by blast
    ultimately show x = y
      using assms
      by metis
  next
    fix x y :: 'a
    assume
      x ∈ A and
      y ∈ A and
      x ≠ y
    moreover have
      ∀ x ∈ A. ∀ y ∈ A. ∀ l ∈ permutations-of-set A.
            x ≠ y ∧ (¬ y ≲l x) ⟶ x ≲l y
      using is-less-preferred-than-l.simps
      unfolding permutations-of-set-def
      by auto
    hence ∀ x ∈ A. ∀ y ∈ A. ∀ l ∈ pl-α ‘ permutations-of-set A.
            x ≠ y ∧ (y, x) ∉ l ⟶ (x, y) ∈ l
      using is-less-preferred-than-l.simps
      unfolding permutations-of-set-def
      unfolding pl-α-def permutations-of-set-def
      by blast
    ultimately show (x, y) ∈ r ∨ (y, x) ∈ r
      using assms
      by metis
  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 $r \in \mathit{pl\text{-}\alpha}$ ‘ *permutations-of-set A*
proof –
let $?\varphi = \lambda\ a.\ \mathit{card}\ ((\mathit{underS}\ r\ a) \cap A)$
let $?\mathit{inv} = \mathit{the\text{-}inv\text{-}into}\ A\ ?\varphi$
let $?l = \mathit{map}\ (\lambda\ x.\ ?\mathit{inv}\ x)\ (\mathit{rev}\ [0\ ..{<}\ \mathit{card}\ A])$
have *antisym*:
$\forall\ a \in A.\ \forall\ b \in A.$
$a \in (\mathit{underS}\ r\ b) \wedge b \in (\mathit{underS}\ r\ a) \longrightarrow \mathit{False}$
using *lin-order*
unfolding *underS-def linear-order-on-def partial-order-on-def antisym-def*
by *blast*
hence $\forall\ a \in A.\ \forall\ b \in A.\ \forall\ c \in A.$
$a \in (\mathit{underS}\ r\ b) \longrightarrow b \in (\mathit{underS}\ r\ c) \longrightarrow a \in (\mathit{underS}\ r\ c)$
using *lin-order CollectD CollectI transD*
unfolding *underS-def linear-order-on-def*
partial-order-on-def preorder-on-def
by (*metis* (*mono-tags*, *lifting*))
hence *a-lt-b-imp*:
$\forall\ a \in A.\ \forall\ b \in A.\ a \in (\mathit{underS}\ r\ b) \longrightarrow (\mathit{underS}\ r\ a) \subset (\mathit{underS}\ r\ b)$
using *preorder-on-def partial-order-on-def linear-order-on-def*
antisym lin-order psubsetI underS-E underS-incr
by *metis*
hence *mon*: $\forall\ a \in A.\ \forall\ b \in A.\ a \in (\mathit{underS}\ r\ b) \longrightarrow ?\varphi\ a < ?\varphi\ b$
using *Int-iff Int-mono a-lt-b-imp card-mono card-subset-eq*
fin finite-Int order-le-imp-less-or-eq underS-E
subset-iff-psubset-eq
by *metis*
moreover have *total-underS*:
$\forall\ a \in A.\ \forall\ b \in A.\ a \neq b \longrightarrow a \in (\mathit{underS}\ r\ b) \vee b \in (\mathit{underS}\ r\ a)$
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 $\forall\ a \in A.\ \forall\ b \in A.\ a \neq b \longrightarrow ?\varphi\ a \neq ?\varphi\ b$
using *order-less-imp-not-eq2*
by *metis*
hence *inj*: *inj-on* $?\varphi$ *A*
using *inj-on-def*
by *blast*
have *in-bounds*: $\forall\ a \in A.\ ?\varphi\ a < \mathit{card}\ A$
using *CollectD IntD1 card-seteq fin inf-le2 linorder-le-less-linear*
unfolding *underS-def*
by (*metis* (*mono-tags*, *lifting*))
hence $?\varphi$ ‘ $A \subseteq \{0\ ..{<}\ \mathit{card}\ A\}$
using *atLeast0LessThan*
by *blast*
moreover have $\mathit{card}\ (?\varphi$ ‘ $A) = \mathit{card}\ A$
using *inj fin card-image*
by *blast*
ultimately have $?\varphi$ ‘ $A = \{0\ ..{<}\ \mathit{card}\ A\}$

```
  by (simp add: card-subset-eq)
hence bij-A: bij-betw ?φ 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-eq-A: 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 ∈ permutations-of-set A
  by auto
moreover have index-eq: ∀ a ∈ A. index ?l a = card A − 1 − ?φ a
proof
  fix a :: 'a
  assume a-in-A: a ∈ A
  have ∀ l. ∀ i < length l. (rev l)!i = l!(length l − 1 − i)
    using rev-nth
    by auto
  hence ∀ i < length [0 ..< card A]. (rev [0 ..< card A])!i =
            [0 ..< card A]!(length [0 ..< card A] − 1 − i)
    by blast
  moreover have ∀ 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 ∀ 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 ∀ i < card A. ?l!i = ?inv ((rev [0 ..< card A])!i)
    by simp
  ultimately have ∀ i < card A. ?l!i = ?inv (card A − 1 − i)
    by presburger
  moreover have
    card A − 1 − (card A − 1 − card (underS r a ∩ A)) =
        card (underS r a ∩ A)
    using in-bounds a-in-A
    by auto
  moreover have ?inv (card (underS r a ∩ 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 (*intro equalityI*, *unfold pl-α-def is-less-preferred-than-l.simps*, *safe*)
fix $a\ b :: 'a$
assume
in-bounds-a: $a \in set\ ?l$ **and**
in-bounds-b: $b \in set\ ?l$
moreover have *element-a*: $?inv\ (index\ ?l\ a) \in A$
using *bij-inv in-bounds-a atLeast0LessThan set-eq-A bij-inv cancel-comm-monoid-add-class.diff-cancel diff-Suc-eq-diff-pred diff-less in-bounds index-eq lessThan-iff less-imp-diff-less zero-less-Suc inj dist-l image-eqI image-eqI length-upt*
unfolding *bij-betw-def*
by (*metis* (*no-types*, *lifting*))
moreover have *el-b*: $?inv\ (index\ ?l\ b) \in A$
using *bij-inv in-bounds-b atLeast0LessThan set-eq-A bij-inv cancel-comm-monoid-add-class.diff-cancel diff-Suc-eq-diff-pred diff-less in-bounds index-eq lessThan-iff less-imp-diff-less zero-less-Suc inj dist-l image-eqI image-eqI length-upt*
unfolding *bij-betw-def*
by (*metis* (*no-types*, *lifting*))
moreover assume $index\ ?l\ b \leq index\ ?l\ a$
ultimately have $card\ A - 1 - (?\varphi\ b) \leq card\ A - 1 - (?\varphi\ a)$
using *index-eq set-eq-A*
by *metis*
moreover have $\forall\ a < card\ A.\ ?\varphi\ (?inv\ a) < card\ A$
using *fin bij-inv bij-A*
unfolding *bij-betw-def*
by *fastforce*
hence $?\varphi\ b \leq card\ A - 1 \wedge ?\varphi\ a \leq card\ A - 1$
using *in-bounds-a in-bounds-b fin*
by *fastforce*
ultimately have $?\varphi\ b \geq ?\varphi\ a$
using *fin le-diff-iff′*
by *blast*
hence $?\varphi\ a < ?\varphi\ b \vee ?\varphi\ a = ?\varphi\ b$
by *auto*
moreover have
$\forall\ a \in A.\ \forall\ b \in A.\ ?\varphi\ a < ?\varphi\ b \longrightarrow a \in underS\ r\ b$
using *mon total-underS antisym order-less-not-sym*

```
      by metis
    hence ?φ a < ?φ b ⟶ a ∈ underS r b
      using element-a el-b in-bounds-a in-bounds-b set-eq-A
      by blast
    hence ?φ a < ?φ b ⟶ (a, b) ∈ r
      unfolding underS-def
      by simp
    moreover have ∀ a ∈ A. ∀ b ∈ A. ?φ a = ?φ b ⟶ a = b
      using mon total-underS antisym order-less-not-sym
      by metis
    hence ?φ a = ?φ b ⟶ a = b
      using element-a el-b in-bounds-a in-bounds-b set-eq-A
      by blast
    hence ?φ a = ?φ b ⟶ (a, b) ∈ r
      using lin-order element-a el-b in-bounds-a
            in-bounds-b set-eq-A
      unfolding linear-order-on-def partial-order-on-def
                preorder-on-def refl-on-def
      by auto
    ultimately show (a, b) ∈ r
      by auto
  next
    fix a b :: 'a
    assume a-b-rel: (a, b) ∈ r
    hence
      a-in-A: a ∈ A and
      b-in-A: b ∈ A and
      a-under-b-or-eq: a ∈ underS r b ∨ a = b
      using lin-order
      unfolding linear-order-on-def partial-order-on-def
                preorder-on-def refl-on-def underS-def
      by auto
    thus
      a ∈ set ?l and
      b ∈ set ?l
      using bij-inv set-eq-A
      by (metis, metis)
    hence ?φ a ≤ ?φ b
      using mon le-eq-less-or-eq a-under-b-or-eq
            a-in-A b-in-A
      by auto
    thus index ?l b ≤ index ?l a
      using index-eq a-in-A b-in-A diff-le-mono2
      by metis
  qed
  ultimately show r ∈ pl-α ` permutations-of-set A
    by auto
qed
```

```
lemma index-helper:
  fixes
    l :: 'x list and
    x :: 'x
  assumes
    finite (set l) and
    distinct l and
    x ∈ set l
  shows index l x = card {y ∈ set l. index l y < index l x}
proof –
  have bij-l: bij-betw (index l) (set l) {0 ..< length l}
    using assms bij-betw-index
    by blast
  hence card {y ∈ set l. index l y < index l x} =
          card (index l ` {y ∈ set l. index l y < index l x})
    using CollectD bij-betw-same-card bij-betw-subset subsetI
    by (metis (no-types, lifting))
  also have index l ` {y ∈ set l. index l y < index l x} =
      {m | m. m ∈ index l ` (set l) ∧ m < index l x}
    by blast
  also have
    {m | m. m ∈ index l ` (set l) ∧ m < index l x} =
      {m | m. m < index l x}
    using bij-l 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 l x} = index l x
    by simp
  finally show ?thesis
    by simp
qed

lemma pl-α-eq-imp-list-eq:
  fixes l l' :: 'x list
  assumes
    fin-set-l: finite (set l) and
    set-eq: set l = set l' and
    dist-l: distinct l and
    dist-l': distinct l' and
    pl-α-eq: pl-α l = pl-α l'
  shows l = l'
proof (rule ccontr)
  assume l ≠ l'
  moreover with set-eq
  have l ≠ [] ∧ l' ≠ []
    by auto
  ultimately obtain
    i :: nat and
    x :: 'x where
```

$i < length\ l$ **and**
$l!i \neq l'!i$ **and**
$x = l!i$ **and**
x-in-l: $x \in set\ l$
using *dist-l dist-l' distinct-remdups-id*
length-remdups-card-conv nth-equalityI
nth-mem set-eq
by *metis*
moreover with *set-eq*
have *neq-ind*: $index\ l\ x \neq index\ l'\ x$
using *dist-l index-nth-id nth-index*
by *metis*
ultimately have
$card\ \{y \in set\ l.\ index\ l\ y < index\ l\ x\} \neq$
$card\ \{y \in set\ l.\ index\ l'\ y < index\ l'\ x\}$
using *dist-l dist-l' set-eq index-helper fin-set-l*
by (*metis* (*mono-tags*))
then obtain $y :: 'x$ **where**
y-in-set-l: $y \in set\ l$ **and**
y-neq-x: $y \neq x$ **and**
neq-indices:
$index\ l\ y < index\ l\ x \wedge index\ l'\ y > index\ l'\ x$
$\vee\ index\ l'\ y < index\ l'\ x \wedge index\ l\ y > index\ l\ x$
using *index-eq-index-conv not-less-iff-gr-or-eq set-eq*
by (*metis* (*mono-tags*, *lifting*))
hence
is-less-preferred-than-l $x\ l\ y\ \wedge$ *is-less-preferred-than-l* $y\ l'\ x$
$\vee$ *is-less-preferred-than-l* $x\ l'\ y\ \wedge$ *is-less-preferred-than-l* $y\ l\ x$
unfolding *is-less-preferred-than-l.simps*
using *y-in-set-l less-imp-le-nat set-eq x-in-l*
by *blast*
hence $(x, y) \in pl\text{-}\alpha\ l \wedge (x, y) \notin pl\text{-}\alpha\ l'$
$\vee\ (x, y) \in pl\text{-}\alpha\ l' \wedge (x, y) \notin pl\text{-}\alpha\ l$
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*) $\{r.\ linear\text{-}order\text{-}on\ X\ r\}$
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 *l* :: *'x list*
assume $l \in$ *permutations-of-set X*
thus *linear-order-on X* (*pl-α l*)
using *assms pl-α-lin-order*
by *blast*
next
fix *r* :: *'x rel*
assume *linear-order-on X r*
thus $r \in$ *pl-α* ‘ *permutations-of-set X*
using *assms lin-order-pl-α*
by *blast*
qed

2.1.5 Limited Preference

definition *limited* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
limited A r ≡ ∀ *a*. $a \in$ *set r* ⟶ $a \in A$

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

lemma *limited-dest*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List* **and**
a 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*
show *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 ($a\#l$))
  proof (*unfold limit-l.simps limit.simps*, *intro equalityI*, *safe*)
    fix *b c* :: $'a$
    assume *b-less-c*: (*b*, *c*) $\in$ *pl-*α (*filter* (λ *a*. $a \in A$) ($a\#l$))
    moreover 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*
    ultimately have
      $b \in$ *set* ($a\#l$) **and**
      $c \in$ *set* ($a\#l$)
      using *case-prodD filter-set mem-Collect-eq member-filter*
          is-less-preferred-than-l.simps
      unfolding *pl-*α*-def*
      by (*metis*, *metis*)
    thus (*b*, *c*) $\in$ *pl-*α ($a\#l$)
    proof (*unfold pl-*α*-def is-less-preferred-than-l.simps*, *safe*)
      have *idx-set-eq*:
        $\forall$ 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-l A l*) b'} =
        {(a', a''). $a' \in$ *set* (*limit-l A l*) $\wedge$ $a'' \in$ *set* (*limit-l A l*) $\wedge$
          index (*limit-l A l*) $a'' \leq$ *index* (*limit-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-l A l*) $\wedge$ $b' \in$ *set* (*limit-l A l*)
            $\wedge$ *index* (*limit-l A l*) $b' \leq$ *index* (*limit-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-l A l*) $\wedge$ $c \in$ *set* (*limit-l A l*)

$\wedge$ *index* (*limit-l A l*) $c \le$ *index* (*limit-l A l*) b
$\longrightarrow b \in$ *set* $l \wedge c \in$ *set* $l \wedge$ *index* $l\ c \le$ *index* $l\ b$
by *auto*
have $b \lesssim_($*filter* (λ a. $a \in A$) ($a\#l$)) c
using *b-less-c case-prodD mem-Collect-eq*
unfolding *pl-α-def*
by (*metis* (*no-types*))
moreover obtain
$f\ h :: 'a \Rightarrow 'a$ *list* $\Rightarrow 'a \Rightarrow 'a$ **and**
$g :: 'a \Rightarrow 'a$ *list* $\Rightarrow 'a \Rightarrow 'a$ *list* **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* ($g\ e\ s\ d$) $\wedge h\ e\ s\ d \in$ *set* ($g\ e\ s\ d$)
$\wedge$ *index* ($g\ e\ s\ d$) ($h\ e\ s\ d$) $\le$ *index* ($g\ e\ s\ d$) ($f\ e\ s\ d$)
by *fastforce*
ultimately have
$b = f\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b
$\wedge$ *filter* (λ a. $a \in A$) ($a\#l$) $=$
$g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b
$\wedge$ $c = h\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b
$\wedge$ $f\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b
$\in$ *set* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
$\wedge$ $h\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b
$\in$ *set* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
$\wedge$ *index* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
($h\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
$\le$ *index* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
($f\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
by *blast*
moreover have *filter* (λ a. $a \in A$) $l =$ *limit-l A l*
by *simp*
moreover have
index (*limit-l A l*) $c \ne$
index ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
($h\ c$ (*filter* (λ a. $a \in A$) ($a\ \#\ l$)) b)
$\vee$ *index* (*limit-l A l*) $b \ne$
index ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
($f\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
$\vee$ *index* (*limit-l A l*) $c \le$ *index* (*limit-l A l*) b
$\vee$ $\neg$ *index* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\ \#\ l$)) b)
($h\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
$\le$ *index* ($g\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
($f\ c$ (*filter* (λ a. $a \in A$) ($a\#l$)) b)
by *presburger*
ultimately have $a \ne c \longrightarrow$ *index* ($a\#l$) $c \le$ *index* ($a\#l$) b
using *add-le-cancel-right idx-imp index-Cons le-zero-eq*
nth-index set-ConsD wf-a-l
unfolding *filter.simps is-less-preferred-than-l.elims*
distinct.simps

```
        by metis
      thus index (a#l) c ≤ index (a#l) b
        by force
    qed
    show
      b ∈ A and
      c ∈ A
      using b-less-c case-prodD mem-Collect-eq set-filter
      unfolding pl-α-def is-less-preferred-than-l.simps
      by (metis (no-types, lifting),
          metis (no-types, lifting))
  next
    fix b c :: 'a
    assume
      b-less-c: (b, c) ∈ pl-α (a#l) and
      b-in-A: b ∈ A and
      c-in-A: c ∈ A
    have (b, c) ∈ pl-α (a#l)
      by (simp add: b-less-c)
    hence b ≲(a#l) c
      using case-prodD mem-Collect-eq
      unfolding pl-α-def
      by metis
    moreover have
      pl-α (filter (λ a. a ∈ A) l) =
          {(a, b). (a, b) ∈ pl-α l ∧ a ∈ A ∧ b ∈ A}
      using wf-a-l wf-imp-limit
      by simp
    ultimately have
      index (filter (λ a. a ∈ A) (a#l)) c
          ≤ index (filter (λ a. a ∈ A) (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 not-one-le-zero
            in-rel-Collect-case-prod-eq mem-Collect-eq
            linorder-le-cases
      by fastforce
    moreover have
      b ∈ set (filter (λ a. a ∈ A) (a#l)) and
      c ∈ set (filter (λ a. a ∈ A) (a#l))
      using b-less-c b-in-A c-in-A
      unfolding pl-α-def
      by (fastforce, fastforce)
    ultimately show (b, c) ∈ pl-α (filter (λ a. a ∈ A) (a#l))
      unfolding pl-α-def
      by simp
  qed
qed
```

2.1.6 Auxiliary Definitions

definition *total-on-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  total-on-l A l ≡ ∀ *a* ∈ *A*. *a* ∈ *set l*

definition *refl-on-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  refl-on-l A l ≡ (∀ *a*. *a* ∈ *set l* ⟶ *a* ∈ *A*) ∧ (∀ *a* ∈ *A*. $a \lesssim_l a$)

definition *trans* :: *'a Preference-List* ⇒ *bool* **where**
  trans l ≡ ∀ (*a*, *b*, *c*) ∈ *set l* × *set l* × *set l*. $a \lesssim_l b$ ∧ $b \lesssim_l c$ ⟶ $a \lesssim_l c$

definition *preorder-on-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  preorder-on-l A l ≡ *refl-on-l A l* ∧ *trans l*

definition *antisym-l* :: *'a list* ⇒ *bool* **where**
  antisym-l l ≡ ∀ *a b*. $a \lesssim_l b$ ∧ $b \lesssim_l a$ ⟶ *a* = *b*

definition *partial-order-on-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  partial-order-on-l A l ≡ *preorder-on-l A l* ∧ *antisym-l l*

definition *linear-order-on-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  linear-order-on-l A l ≡ *partial-order-on-l A l* ∧ *total-on-l A l*

definition *connex-l* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  connex-l A l ≡ *limited A l* ∧ (∀ *a* ∈ *A*. ∀ *b* ∈ *A*. $a \lesssim_l b$ ∨ $b \lesssim_l a$)

abbreviation *ballot-on* :: *'a set* ⇒ *'a Preference-List* ⇒ *bool* **where**
  ballot-on A l ≡ *well-formed-l l* ∧ *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 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 *Preference-List.is-less-preferred-than-l.simps above-l-def One-nat-def*
by *metis*

lemma *less-preferred-l-rel-equiv*:
fixes
l :: *'a Preference-List* **and**
a 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*: $b \in$ *set* (*above-l l a*)
hence *index l b* $\leq$ *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 $b \in$ *above* (*pl-*α *l*) *a*
  using *less-preferred-l-rel-equiv pref-imp-in-above*
  by *metis*
next
fix b :: $'a$
assume $b \in$ *above* (*pl-*α *l*) *a*
hence $a \lesssim_l b$
  using *pref-imp-in-above less-preferred-l-rel-equiv*
  by *metis*
thus $b \in$ *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 (*unfold rank-l.simps rank.simps*, *cases* $a \in$ *set l*)
case *True*
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
(*if* $a \in$ *set l then index l a* + *1 else 0*) =
card (*above* (*pl-*α *l*) *a*)
by *simp*
next
case *False*
hence *above* (*pl-*α *l*) *a* = {}
unfolding *above-def*
using *less-preferred-l-rel-equiv*
by *fastforce*
thus (*if* $a \in$ *set l then index l a* + *1 else 0*) =
card (*above* (*pl-*α *l*) *a*)
using *False*
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 *is-less-preferred-than-l.simps antisym-def total-on-def*
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
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*)

```
  case Nil
  thus ?case
    by simp
next
  fix
    l :: 'a Preference-List and
    a :: 'a
  case (Cons a l)
  thus ?case
    by simp
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 index-first member-def rank-l.simps
        assms add-diff-cancel-right'
  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)
  case Nil
  thus ?case
    by simp
next
  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* $(\lambda\ y\ z.\ y \lesssim_l z)$ $(set\ l) = pl\text{-}\alpha\ l$
proof (*unfold relation-of-def*, *safe*)
fix $a\ b$ :: $'a$
assume $a \lesssim_l b$
moreover have $a \lesssim_l b = (a \preceq_{(}pl\text{-}\alpha\ l)\ b)$
using *less-preferred-l-rel-equiv*
by (*metis* (*no-types*))
ultimately show $(a, b) \in pl\text{-}\alpha\ l$
by *simp*
next
fix $a\ b$ :: $'a$
assume $(a, b) \in pl\text{-}\alpha\ l$
thus $a \lesssim_l b$
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 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*

2.2.2 Refinement Proof

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 *profile-list-refines-profile*:
  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 −
  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
  moreover assume
    ⋀ ord. linear-order-on X ord ⟹ ?thesis
  ultimately show ?thesis
    by blast
qed

typedef 'a Ordered-Preference =
  {p :: 'a :: finite Preference-Relation. linear-order-on (UNIV :: 'a set) p}
  morphisms ord2pref pref2ord
proof (unfold mem-Collect-eq)
  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-Ordered-Preference type-definition.Rep-range*
  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 (safe)
  have election-equality
    (fst E, fst (snd E), snd (snd E)) (fst E, fst (snd E), snd (snd E))
    unfolding election-equality.simps
    by safe
  thus election-equality E E
    by clarsimp
next
  assume election-equality E E'
  hence election-equality
    (fst E, fst (snd E), snd (snd E)) (fst E', fst (snd E'), snd (snd E'))
    by simp
  hence election-equality
    (fst E', fst (snd E'), snd (snd E')) (fst E, fst (snd E), snd (snd E))
    unfolding election-equality.simps
    by (metis (mono-tags, lifting))
  thus election-equality E' E
    by clarsimp
next
  assume
    election-equality E E' and
    election-equality E' F
  hence
    election-equality
      (fst E, fst (snd E), snd (snd E)) (fst E', fst (snd E'), snd (snd E')) and
```

election-equality
  $(fst\ E', fst\ (snd\ E'), snd\ (snd\ E'))\ (fst\ F, fst\ (snd\ F), snd\ (snd\ F))$
 by (*simp*, *simp*)
hence *election-equality*
 $(fst\ E, fst\ (snd\ E), snd\ (snd\ E))\ (fst\ F, fst\ (snd\ F), snd\ (snd\ F))$
 unfolding *election-equality.simps*
 by (*metis* (*no-types*, *lifting*))
thus *election-equality E F*
 by *clarsimp*
qed

quotient-type $('a, 'v)$ $Election_{\mathcal{Q}}$ =
 $'a$ *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 $fst_{\mathcal{Q}}$ :: $('a, 'v)$ $Election_{\mathcal{Q}} \Rightarrow 'a$ *set* **where**
 $fst_{\mathcal{Q}}\ E = fst\ (rep\text{-}Election_{\mathcal{Q}}\ E)$

fun $snd_{\mathcal{Q}}$ :: $('a, 'v)$ $Election_{\mathcal{Q}} \Rightarrow 'v$ *set* × $('a, 'v)$ *Profile* **where**
 $snd_{\mathcal{Q}}\ E = snd\ (rep\text{-}Election_{\mathcal{Q}}\ E)$

abbreviation $alternatives\text{-}\mathcal{E}_{\mathcal{Q}}$ :: $('a, 'v)$ $Election_{\mathcal{Q}} \Rightarrow 'a$ *set* **where**
 $alternatives\text{-}\mathcal{E}_{\mathcal{Q}}\ E \equiv fst_{\mathcal{Q}}\ E$

abbreviation $voters\text{-}\mathcal{E}_{\mathcal{Q}}$ :: $('a, 'v)$ $Election_{\mathcal{Q}} \Rightarrow 'v$ *set* **where**
 $voters\text{-}\mathcal{E}_{\mathcal{Q}}\ E \equiv fst\ (snd_{\mathcal{Q}}\ E)$

abbreviation $profile\text{-}\mathcal{E}_{\mathcal{Q}}$ :: $('a, 'v)$ $Election_{\mathcal{Q}} \Rightarrow ('a, 'v)$ *Profile* **where**
 $profile\text{-}\mathcal{E}_{\mathcal{Q}}\ E \equiv snd\ (snd_{\mathcal{Q}}\ E)$

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**
  $singleton\text{-}set\ s = (if\ card\ s = 1\ then\ the\text{-}inv\ (\lambda\ x.\ \{x\})\ s\ else\ undefined)$
— This is undefined if $card\ s \neq 1$. 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**
  $\pi_{\mathcal{Q}}\ f\ s = singleton\text{-}set\ (f\ `\ s)$

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

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

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 *restr-equals-restricted-rel*:
  fixes
    s t :: *'a set* **and**
    r :: *'a rel*
  assumes
    closed-restricted-rel r s t **and**
    $t \subseteq s$
  shows *restricted-rel r t s* = *Restr r t*
proof (*unfold restricted-rel.simps*, *safe*)
  fix *a b* :: *'a*
  assume
    $(a, b) \in r$ **and**
    $a \in t$ **and**
    $b \in s$
  thus $b \in t$
    using *assms*
    unfolding *closed-restricted-rel.simps restricted-rel.simps*
    by *blast*
next
  fix *a b* :: *'a*
  assume $b \in t$
  thus $b \in s$
    using *assms*
    by *blast*
qed

lemma *equiv-rel-restr*:
  fixes
    s 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 (action-induced-rel (carrier m) s φ)
proof (unfold equiv-def refl-on-def sym-def Relation.trans-def
              action-induced-rel.simps, safe)
  fix y :: 'y
  assume y ∈ s
  hence φ 1 m y = y
    using assms group-action.id-eq-one restrict-apply'
    by metis
  thus ∃ g ∈ carrier m. φ 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 ∈ s and
    carrier-g: g ∈ carrier m
  hence y = φ (inv m g) (φ g y)
    using assms
    by (simp add: group-action.orbit-sym-aux)
  thus ∃ h ∈ carrier m. φ h (φ g y) = y
    using assms carrier-g group.inv-closed
          group-action.group-hom
    unfolding group-hom-def
    by metis
next
  fix
    y :: 'y and
    g h :: 'x
  assume
    y-in-s: y ∈ s and
    carrier-g: g ∈ carrier m and
    carrier-h: h ∈ carrier m
  hence φ (h ⊗ m g) y = φ h (φ 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 monoid.m-closed*
unfolding *group-def group-hom-def*
by *metis*
next
fix
y :: $'y$ **and**
g :: $'x$
assume
$y \in s$ **and**
$g \in$ *carrier m*
thus $\varphi\ g\ y \in s$
using *assms group-action.element-image*
by *metis*
next
fix
y :: $'y$ **and**
g :: $'x$
assume
$y \in s$ **and**
$g \in$ *carrier m*
thus $\varphi\ g\ y \in s$
using *assms group-action.element-image*
by *metis*
qed

end

3.2 Quotients of Election Set Equivalences

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
A :: $'a$ *set* **and**
B :: $'b$ *set* **and**
f :: $'b \Rightarrow nat$
assumes

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

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

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

3.2.2 Anonymity Quotient: Grid

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

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

fun *anonymity-class* :: (*′a* :: *finite*, *′v*) *Election set* ⇒
(*nat*, *′a Ordered-Preference*) *vec* **where**
anonymity-class X = (χ *p*. *vote-count*$_{\mathcal{Q}}$ (*ord2pref p*) *X*)

lemma *anon-rel-equiv*: *equiv* (*elections-$\mathcal{A}$ UNIV*) (*anonymity*$_{\mathcal{R}}$ (*elections-$\mathcal{A}$ UNIV*))
proof −
have *subset*: *elections-$\mathcal{A}$ UNIV* ⊆ *well-formed-elections*
by *simp*
have *equiv*: *equiv well-formed-elections* (*anonymity*$_{\mathcal{R}}$ *well-formed-elections*)
using *anonymous-group-action.group-action-axioms rel-ind-by-group-act-equiv*[*of bijection*$_{\mathcal{VG}}$ *well-formed-elections* φ*-anon well-formed-elections*]
rel-ind-by-coinciding-action-on-subset-eq-restr
by *fastforce*
have *closed*:

```
  closed-restricted-rel
   (anonymity_R well-formed-elections) well-formed-elections (elections-A UNIV)
proof (unfold closed-restricted-rel.simps restricted-rel.simps, safe)
  fix
    A A' :: 'c set and
    V V' :: 'd set and
    p p' :: ('c, 'd) Profile
  assume elt: (A, V, p) ∈ elections-A UNIV
  hence wf-elections: (A, V, p) ∈ well-formed-elections
    unfolding elections-A.simps
    by blast
  assume ((A, V, p), A', V', p') ∈ anonymity_R well-formed-elections
  then obtain π :: 'd ⇒ 'd where
    bij-π: bij π and
    img: (A', V', p') = rename π (A, V, p)
    unfolding anonymity_R.simps action-induced-rel.simps
              bijection_VG-def φ-anon.simps rewrite-carrier
              extensional-continuation.simps
    by auto
  hence (A', V', p') ∈ well-formed-elections
    using wf-elections rename-sound
    unfolding well-formed-elections-def
    by fastforce
  moreover have A' = A ∧ finite V
    using bij-π elt img rename.simps wf-elections well-formed-elections-def
    by auto
  moreover have ∀ v. v ∉ V' ⟶ (the-inv π v) ∉ V
    using elt Pair-inject UNIV-I bij-π rename.simps
        f-the-inv-into-f-bij-betw image-eqI img
    unfolding elections-A.simps
    by (metis (mono-tags, opaque-lifting))
  moreover have ∀ v. v ∉ V' ⟶ p' v = p (the-inv π v)
    using img
    by simp
  ultimately show (A', V', p') ∈ elections-A UNIV
    using elt img
    unfolding elections-A.simps rename.simps
    by auto
qed
have
  anonymity_R (elections-A UNIV) =
    restricted-rel (anonymity_R well-formed-elections) (elections-A UNIV)
        well-formed-elections
proof (unfold restricted-rel.simps, safe)
  fix
    A A' :: 'c set and
    V V' :: 'd set and
    p p' :: ('c, 'd) Profile
  assume rel: ((A, V, p), A', V', p') ∈ anonymity_R (elections-A UNIV)
```

hence $(A, V, p) \in$ *well-formed-elections*
  unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps elections-*$\mathcal{A}$*.simps*
  by *blast*
moreover obtain π :: $'d \Rightarrow 'd$ **where**
  bij π **and**
  (A', V', p') = *rename* π (A, V, p)
  using *rel*
  unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    bijection$_{\mathcal{VG}}$*-def* φ*-anon.simps rewrite-carrier*
    extensional-continuation.simps
  by *auto*
ultimately show $((A, V, p), A', V', p') \in$ *anonymity*$_{\mathcal{R}}$ *well-formed-elections*
  unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    bijection$_{\mathcal{VG}}$*-def* φ*-anon.simps rewrite-carrier*
  by *auto*
next
  fix
    A A' :: $'c$ *set* **and**
    V V' :: $'d$ *set* **and**
    p p' :: $('c, 'd)$ *Profile*
  assume $((A, V, p), A', V', p') \in$ *anonymity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
  thus $(A, V, p) \in$ *elections-*$\mathcal{A}$ *UNIV*
    unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    by *blast*
next
  fix
    A A' :: $'c$ *set* **and**
    V V' :: $'d$ *set* **and**
    p p' :: $('c, 'd)$ *Profile*
  assume
    rel: $((A, V, p), A', V', p') \in$ *anonymity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
  hence $((A, V, p), A', V', p') \in$ *anonymity*$_{\mathcal{R}}$ *well-formed-elections*
    unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    by *fastforce*
  moreover have *elt*: $(A, V, p) \in$ *elections-*$\mathcal{A}$ *UNIV*
    using *rel*
    unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    by *blast*
  ultimately have
    $((A, V, p), A', V', p') \in$ *restricted-rel*
    (*anonymity*$_{\mathcal{R}}$ *well-formed-elections*) (*elections-*$\mathcal{A}$ *UNIV*) *well-formed-elections*
    using *equiv*
    unfolding *restricted-rel.simps equiv-def refl-on-def*
    by *blast*
  hence $(A', V', p') \in$ *elections-*$\mathcal{A}$ *UNIV*
    using *closed elt*
    unfolding *closed-restricted-rel.simps*
    by *blast*
  thus $(A', V', p') \in$ *well-formed-elections*

```
      using subset
      by blast
  next
    fix
      A A' :: 'c set and
      V V' :: 'd set and
      p p' :: ('c, 'd) Profile
    assume
      (A, V, p) ∈ elections-𝒜 UNIV and
      ((A, V, p), A', V', p') ∈ anonymity_ℛ well-formed-elections
    moreover from this
    obtain π :: 'd ⇒ 'd where
      bij π and
      (A', V', p') = rename π (A, V, p)
      unfolding anonymity_ℛ.simps action-induced-rel.simps
                bijection_𝒱𝒢-def φ-anon.simps rewrite-carrier
                extensional-continuation.simps
      by auto
    ultimately show ((A, V, p), A', V', p') ∈ anonymity_ℛ (elections-𝒜 UNIV)
      unfolding anonymity_ℛ.simps action-induced-rel.simps
                bijection_𝒱𝒢-def φ-anon.simps rewrite-carrier
                extensional-continuation.simps
      by auto
  qed
  also have ... = Restr (anonymity_ℛ well-formed-elections) (elections-𝒜 UNIV)
    using restr-equals-restricted-rel closed subset
    by blast
  finally have
    anonymity_ℛ (elections-𝒜 UNIV) =
      Restr (anonymity_ℛ well-formed-elections) (elections-𝒜 UNIV)
    by simp
  thus ?thesis
    using equiv-rel-restr subset equiv
    by metis
qed
```

We assume that all elections 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 operate 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 anonymity_𝒬-isomorphism:
  assumes infinite (UNIV :: 'v set)
  shows bij-betw (anonymity-class :: ('a :: finite, 'v) Election set
          ⇒ nat^('a Ordered-Preference)) (anonymity_𝒬 (UNIV :: 'a set))
```

(*UNIV* :: (*nat* ⌢ (*'a Ordered-Preference*)) *set*)
proof (*unfold bij-betw-def inj-on-def*, *intro conjI ballI impI*)
fix *X Y* :: (*'a*, *'v*) *Election set*
assume
class-X: $X \in anonymity_{\mathcal{Q}}$ *UNIV* **and**
class-Y: $Y \in anonymity_{\mathcal{Q}}$ *UNIV* **and**
eq-vec: *anonymity-class X* = *anonymity-class Y*
have ∀ *E* ∈ *elections-$\mathcal{A}$ UNIV*. *finite* (*voters-$\mathcal{E}$ E*)
by *simp*
hence ∀ (*E*, *E'*) ∈ $anonymity_{\mathcal{R}}$ (*elections-$\mathcal{A}$ UNIV*). *finite* (*voters-$\mathcal{E}$ E*)
by *simp*
moreover have *subset*: *elections-$\mathcal{A}$ UNIV* ⊆ *well-formed-elections*
by *simp*
ultimately have
∀ (*E*, *E'*) ∈ $anonymity_{\mathcal{R}}$ (*elections-$\mathcal{A}$ 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* ($anonymity_{\mathcal{R}}$ (*elections-$\mathcal{A}$ UNIV*))
unfolding *congruent-def*
by *blast*
have *quotient-count*:
∀ *X* ∈ $anonymity_{\mathcal{Q}}$ *UNIV*. ∀ *p*. ∀ *E* ∈ *X*. $vote\text{-}count_{\mathcal{Q}}$ *p X* = *vote-count p E*
using *pass-to-quotient*[*of* $anonymity_{\mathcal{R}}$ (*elections-$\mathcal{A}$ UNIV*)]
vote-count-invar anon-rel-equiv
unfolding $anonymity_{\mathcal{Q}}$*.simps* $anonymity_{\mathcal{R}}$*.simps* $vote\text{-}count_{\mathcal{Q}}$*.simps*
by *metis*
moreover from *anon-rel-equiv*
obtain
E E' :: (*'a*, *'v*) *Election* **where**
E-in-X: *E* ∈ *X* **and**
E'-in-Y: *E'* ∈ *Y*
using *class-X class-Y equiv-Eps-in*
unfolding $anonymity_{\mathcal{Q}}$*.simps*
by *metis*
ultimately have
∀ *p*. $vote\text{-}count_{\mathcal{Q}}$ *p X* = *vote-count p E* ∧ $vote\text{-}count_{\mathcal{Q}}$ *p Y* = *vote-count p E'*
using *class-X class-Y*
by *blast*
moreover with *eq-vec* **have**
∀ *p*. $vote\text{-}count_{\mathcal{Q}}$ (*ord2pref p*) *X* = $vote\text{-}count_{\mathcal{Q}}$ (*ord2pref p*) *Y*
unfolding *anonymity-class.simps*
using *UNIV-I vec-lambda-inverse*
by *metis*
ultimately have ∀ *p*. *vote-count* (*ord2pref p*) *E* = *vote-count* (*ord2pref p*) *E'*
by *simp*
hence *eq*: ∀ *p* ∈ {*r*. *linear-order-on* (*UNIV* :: *'a set*) *r*}.
vote-count p E = *vote-count p E'*

using *pref2ord-inverse*
by *metis*
from *anon-rel-equiv class-X class-Y* **have** *subset-fixed-alts*:
$X \subseteq$ *elections-$\mathcal{A}$ UNIV* $\wedge$ $Y \subseteq$ *elections-$\mathcal{A}$ 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 *elections-$\mathcal{A}$.simps*
by *blast*
with *subset-fixed-alts* **have** *eq-complement*:
$\forall$ *p* $\in$ *UNIV* $-$ $\{$*r. linear-order-on (UNIV :: 'a set) r*$\}$.
$\{$*v* $\in$ *voters-$\mathcal{E}$ E. profile-$\mathcal{E}$ E v = p*$\}$ = $\{\}$
$\wedge$ $\{$*v* $\in$ *voters-$\mathcal{E}$ E$'$. profile-$\mathcal{E}$ E$'$ v = p*$\}$ = $\{\}$
using *E-in-X E$'$-in-Y*
unfolding *elections-$\mathcal{A}$.simps well-formed-elections-def profile-def*
by *auto*
hence $\forall$ *p* $\in$ *UNIV* $-$ $\{$*r. linear-order-on (UNIV :: 'a set) r*$\}$.
vote-count p E = 0 $\wedge$ *vote-count p E$'$ = 0*
unfolding *card-eq-0-iff vote-count.simps*
by *simp*
with *eq* **have** *eq-vote-count*: $\forall$ *p. vote-count p E = vote-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 *elections-$\mathcal{A}$.simps*
by *blast*
moreover from *subset-fixed-alts E-in-X E$'$-in-Y*
have *(E, E$'$)* $\in$ *(elections-$\mathcal{A}$ UNIV)* $\times$ *(elections-$\mathcal{A}$ UNIV)*
by *blast*
moreover from *this*
have ($\forall$ *v. v* $\notin$ *voters-$\mathcal{E}$ E* $\longrightarrow$ *profile-$\mathcal{E}$ E v* = $\{\}$)
$\wedge$ ($\forall$ *v. v* $\notin$ *voters-$\mathcal{E}$ E$'$* $\longrightarrow$ *profile-$\mathcal{E}$ E$'$ v* = $\{\}$)
by *simp*
ultimately have *(E, E$'$)* $\in$ *anonymity$_{\mathcal{R}}$ (elections-$\mathcal{A}$ UNIV)*
using *eq-alts vote-count-anon-rel*
by *metis*
hence *anonymity$_{\mathcal{R}}$ (elections-$\mathcal{A}$ UNIV)* `` $\{E\}$ =
anonymity$_{\mathcal{R}}$ (elections-$\mathcal{A}$ UNIV) `` $\{E'\}$
using *anon-rel-equiv equiv-class-eq*
by *metis*
also have *anonymity$_{\mathcal{R}}$ (elections-$\mathcal{A}$ UNIV)* `` $\{E\}$ = X
using *E-in-X class-X anon-rel-equiv Image-singleton-iff equiv-class-eq quotientE*
unfolding *anonymity$_{\mathcal{Q}}$.simps*
by (*metis* (*no-types*, *lifting*))
also have *anonymity$_{\mathcal{R}}$ (elections-$\mathcal{A}$ UNIV)* `` $\{E'\}$ = Y
using *E$'$-in-Y class-Y anon-rel-equiv Image-singleton-iff equiv-class-eq quotientE*

```
    unfolding anonymity_Q.simps
    by (metis (no-types, lifting))
  finally show X = Y
    by simp
next
  have (UNIV :: (nat, 'a Ordered-Preference) vec set) ⊆
      (anonymity-class :: ('a, 'v) Election set ⇒ (nat, 'a Ordered-Preference) vec) `
      anonymity_Q UNIV
  proof (unfold anonymity-class.simps, safe)
    fix x :: (nat, 'a Ordered-Preference) vec
    have finite (UNIV :: 'a Ordered-Preference set)
      by simp
    hence finite {x$i | i. i ∈ UNIV}
      using finite-Atleast-Atmost-nat
      by blast
    hence (∑ i ∈ UNIV. x$i) < ∞
      using enat-ord-code
      by simp
    moreover have 0 ≤ (∑ i ∈ UNIV. x$i)
      by blast
    ultimately obtain V :: 'v set where
      fin-V: finite V and
      card V = (∑ i ∈ UNIV. x$i)
      using assms infinite-arbitrarily-large
      by metis
    then obtain X' :: 'a Ordered-Preference ⇒ 'v set where
      card': ∀ i. card (X' i) = x$i and
      partition': V = ⋃ {X' i | i. i ∈ UNIV} and
      disjoint': ∀ i j. i ≠ j ⟶ X' i ∩ X' j = {}
      using obtain-partition[of V UNIV ($) x]
      by auto
    obtain X :: 'a Preference-Relation ⇒ 'v set where
      def-X: X = (λ i. if i ∈ {r. linear-order r}
                  then X' (pref2ord i) else {})
      by simp
    hence {X i | i. i ∉ {r. linear-order r}} ⊆ {{}}
      by auto
    moreover have
      {X i | i. i ∈ {r. linear-order r}} =
        {X' (pref2ord i) | i. i ∈ {r. linear-order r}}
      using def-X
      by metis
    moreover have
      {X i | i. i ∈ UNIV} =
        {X i | i. i ∈ {r. linear-order r}}
        ∪ {X i | i. i ∈ UNIV − {r. linear-order r}}
      by blast
    ultimately have
      {X i | i. i ∈ UNIV} = {X' (pref2ord i) | i. i ∈ {r. linear-order r}}
```

$\vee \{X\ i \mid i.\ i \in UNIV\} =$
$\{X'\ (pref2ord\ i) \mid i.\ i \in \{r.\ linear\text{-}order\ r\}\} \cup \{\{\}\}$
by *auto*
also have
$\{X'\ (pref2ord\ i) \mid i.\ i \in \{r.\ linear\text{-}order\ r\}\} = \{X'\ i \mid i.\ i \in UNIV\}$
using *iso-tuple-UNIV-I pref2ord-cases*
by *metis*
finally have
$\{X\ i \mid i.\ i \in UNIV\} = \{X'\ i \mid i.\ i \in UNIV\} \vee$
$\{X\ i \mid i.\ i \in UNIV\} = \{X'\ i \mid i.\ i \in UNIV\} \cup \{\{\}\}$
by *simp*
hence $\bigcup\ \{X\ i \mid i.\ i \in UNIV\} = \bigcup\ \{X'\ i \mid i.\ i \in UNIV\}$
using *Sup-union-distrib ccpo-Sup-singleton sup-bot.right-neutral*
by (*metis* (*no-types*, *lifting*))
hence *partition*: $V = \bigcup\ \{X\ i \mid i.\ i \in UNIV\}$
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 $p' :: {}'v \Rightarrow {}'a$ *Preference-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 $p :: {}'v \Rightarrow {}'a$ *Preference-Relation* **where**
p-in-V-then-p': $p = (\lambda\ v.\ if\ v \in V\ then\ p'\ v\ else\ \{\})$
by *simp*
hence *lin-ord*: $\forall\ v \in V.\ linear\text{-}order\ (p\ v)$
using *def-X p-X p-disj*
by *fastforce*
hence *wf-elections*: $(UNIV,\ V,\ p) \in elections\text{-}\mathcal{A}\ UNIV$
using *fin-V*
unfolding *p-in-V-then-p' elections-$\mathcal{A}$.simps*
well-formed-elections-def profile-def
by *auto*
hence $\forall\ i.\ \forall\ E \in anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)$ `` $\{(UNIV,\ V,\ p)\}.$
$vote\text{-}count\ i\ E = vote\text{-}count\ i\ (UNIV,\ V,\ p)$
using *fin-V anon-rel-vote-count*[*of* $(UNIV,\ V,\ p)$ - *elections-$\mathcal{A}$ UNIV*]
by *simp*
moreover have
$(UNIV,\ V,\ p) \in anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)$ `` $\{(UNIV,\ V,\ p)\}$
using *anon-rel-equiv wf-elections*
unfolding *Image-def equiv-def refl-on-def*
by *blast*
ultimately have *eq-vote-count*:
$\forall\ i.\ vote\text{-}count\ i$ `
$(anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)$ `` $\{(UNIV,\ V,\ p)\}) =$

$\{vote\text{-}count\ i\ (UNIV,\ V,\ p)\}$
by *blast*
have $\forall\ i.\ \forall\ v \in V.\ p\ v = i \longleftrightarrow v \in X\ i$
using *p-X p-disj*
unfolding *p-in-V-then-p′*
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 $\forall\ i \in \{r.\ linear\text{-}order\ r\}.$
$vote\text{-}count\ i\ (UNIV,\ V,\ p) = x\$(pref2ord\ i)$
using *def-X card′*
by *simp*
hence $\forall\ i \in \{r.\ linear\text{-}order\ r\}.$
$vote\text{-}count\ i\ `\ (anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ p)\}) =$
$\{x\$(pref2ord\ i)\}$
using *eq-vote-count*
by *metis*
hence
$\forall\ i \in \{r.\ linear\text{-}order\ r\}.$
$vote\text{-}count_{\mathcal{Q}}\ i\ (anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ p)\}) =$
$x\$(pref2ord\ i)$
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 $\forall\ i.\ vote\text{-}count_{\mathcal{Q}}\ (ord2pref\ i)$
$(anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ p)\}) = x\$i$
using *ord2pref ord2pref-inverse*
by *metis*
hence *anonymity-class*
$(anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ p)\}) = x$
using *anonymity-class.simps vec-lambda-unique*
by (*metis* (*no-types*, *lifting*))
moreover have
$anonymity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ p)\} \in anonymity_{\mathcal{Q}}\ UNIV$
using *wf-elections*
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 *anonymity-class.elims*
by *blast*
qed
thus $(anonymity\text{-}class :: ('a,\ 'v)\ Election\ set$

$\Rightarrow$ (*nat*, *'a Ordered-Preference*) *vec*) ‘
anonymity$_{\mathcal{Q}}$ *UNIV* =
(*UNIV* :: (*nat*, *'a Ordered-Preference*) *vec set*)
by *blast*
qed

3.2.3 Homogeneity Quotient: Simplex

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

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

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

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

fun *anonymity-homogeneity-class* :: (*'a* :: *finite*, *'v*) *Election set* $\Rightarrow$
(*rat*, *'a Ordered-Preference*) *vec* **where**
anonymity-homogeneity-class $\mathcal{E}$ = (χ *p*. *vote-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-vector* :: *real*$^\frown$*'b* $\Rightarrow$ *rat*$^\frown$*'b* **where**
rat-vector v = (χ *p*. *the-inv of-rat* (*v*\$*p*))

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

definition *standard-basis* :: (*real*$^\frown$*'b*) *set* **where**
standard-basis $\equiv$ {*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 $\equiv$
insert 0 (*rat-vector-set* (*convex hull standard-basis* :: (*real*$^\frown$*'b*) *set*))

Auxiliary Lemmas

lemma *convex-combination-in-convex-hull*:

fixes
  $X :: (real^{\frown\prime}b)$ *set* **and**
  $x :: real^{\frown\prime}b$
assumes $\exists\ f :: real^{\frown\prime}b \Rightarrow real.$
  $(\sum\ y \in X.\ f\ y) = 1 \wedge (\forall\ x \in X.\ f\ x \geq 0)$
  $\wedge\ x = (\sum\ x \in X.\ f\ x *_R x)$
shows $x \in$ *convex hull* X
using *assms*
proof (*induction card X arbitrary*: $X\ x$)
case *0*
fix
  $X :: (real^{\frown\prime}b)$ *set* **and**
  $x :: real^{\frown\prime}b$
assume
  $0 =$ *card* X **and**
  $\exists\ f.\ (\sum\ y \in X.\ f\ y) = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x = (\sum\ x \in X.\ f\ x *_R x)$
hence $(\forall\ f.\ (\sum\ y \in X.\ f\ y) = 0) \wedge (\exists\ f.\ (\sum\ y \in X.\ f\ y) = 1)$
  using *card-0-eq empty-iff sum.infinite sum.neutral zero-neq-one*
  by *metis*
hence $\exists\ f.\ (\sum\ y \in X.\ f\ y) = 1 \wedge (\sum\ y \in X.\ f\ y) = 0$
  by *metis*
hence *False*
  using *zero-neq-one*
  by *metis*
thus *?case*
  by *simp*
next
case (*Suc n*)
fix
  $X :: (real^{\frown\prime}b)$ *set* **and**
  $x :: real^{\frown\prime}b$ **and**
  $n :: nat$
assume
  card: *Suc* $n =$ *card* X **and**
  $\exists\ f.\ (\sum\ y \in X.\ f\ y) = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x = (\sum\ x \in X.\ f\ x *_R x)$
and
  hyp: $\bigwedge (X :: (real^{\frown\prime}b)\ set)\ x.\ n = card\ X$
  $\Longrightarrow \exists\ f.\ (\sum\ y \in X.\ f\ y) = 1 \wedge (\forall\ x \in X.\ 0 \leq f\ x) \wedge x =$
  $(\sum\ x \in X.\ f\ x *_R x)$
  $\Longrightarrow x \in$ *convex hull* X
then obtain $f :: real^{\frown\prime}b \Rightarrow real$ **where**
  sum: $(\sum\ y \in X.\ f\ y) = 1$ **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-one*: $card\ X > 1$
using *card*
by *simp*
have $(\forall\ y \in X.\ f\ y \geq 1) \longrightarrow (\sum\ y \in X.\ f\ y) \geq (\sum\ x \in X.\ 1)$
using *fin sum-mono*
by *metis*
moreover have $(\sum\ x \in X.\ 1) = card\ X$
by *force*
ultimately have $(\forall\ y \in X.\ f\ y \geq 1) \longrightarrow card\ X \leq (\sum\ y \in X.\ f\ y)$
by *force*
hence $(\forall\ y \in X.\ f\ y \geq 1) \longrightarrow 1 < (\sum\ y \in X.\ f\ y)$
using *card-X-gt-one*
by *linarith*
then obtain $y :: real\hat{}'b$ **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 $f' :: real\hat{}'b \Rightarrow real$ **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
  $(\sum\ y \in X - \{y\}.\ f'\ y) = (\sum\ x \in X - \{y\}.\ f\ x)\ /\ (1 - f\ y)$
  unfolding *def′ sum-divide-distrib*
  by *simp*
moreover have
  $(\sum\ x \in X - \{y\}.\ f\ x)\ /\ (1 - f\ y) = (1 - f\ y)\ /\ (1 - f\ y)$
  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
  $(\sum\ y \in X - \{y\}.\ f'\ y) = 1 \wedge (\forall\ x \in X - \{y\}.\ 0 \leq f'\ x)$
    $\wedge (\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R\ x) =$
   $(\sum\ x \in X - \{y\}.\ f'\ x *_R\ x)$
  using *def′*
  by *metis*
hence $\exists\ f'.\ (\sum\ y \in X - \{y\}.\ f'\ y) = 1 \wedge (\forall\ x \in X - \{y\}.\ 0 \leq f'\ x)$
    $\wedge (\sum\ x \in X - \{y\}.\ (f\ x\ /\ (1 - f\ y)) *_R\ x) =$
   $(\sum\ x \in X - \{y\}.\ f'\ x *_R\ x)$
  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* =
$\{v :: real\,\hat{}\,'b.\ (\forall\ i.\ v\$i \geq 0) \wedge (\sum\ y \in UNIV.\ v\$y) = 1\}$
proof (*unfold convex-def hull-def*, *intro equalityI*)
let $?simplex = \{v :: real\,\hat{}\,'b.\ (\forall\ i.\ v\$i \geq 0) \wedge (\sum\ y \in UNIV.\ v\$y) = 1\}$
have $\forall\ v :: real\,\hat{}\,'b \in standard\text{-}basis.\ \exists\ b.$
$v\$b = 1 \wedge (\forall\ c.\ c \neq b \longrightarrow v\$c = 0)$
unfolding *standard-basis-def*
by *simp*
then obtain $one :: real\,\hat{}\,'b \Rightarrow 'b$ **where**
def-map: $\forall\ v \in standard\text{-}basis.\ v\$(one\ v) = 1 \wedge (\forall\ i \neq one\ v.\ v\$i = 0)$
by *metis*
hence $\forall\ v :: real\,\hat{}\,'b \in standard\text{-}basis.\ \forall\ b.\ v\$b \geq 0$
using *dual-order.refl zero-less-one-class.zero-le-one*
by *metis*
moreover have $\forall\ v :: real\,\hat{}\,'b \in standard\text{-}basis.$
$(\sum\ z \in UNIV.\ v\$z) = (\sum\ z \in UNIV - \{one\ v\}.\ v\$z) + v\$(one\ v)$
unfolding *def-map*
using *add.commute finite insert-UNIV sum.insert-remove*
by *metis*
moreover have $\forall\ v \in standard\text{-}basis.$
$(\sum\ z \in UNIV - \{one\ v\}.\ v\$z) + v\$(one\ v) = 1$
using *def-map*
by *simp*
ultimately have *standard-basis* $\subseteq$ *?simplex*
by *force*
moreover have $\forall\ x :: real\,\hat{}\,'b.\ \forall\ y.\ (\sum\ z \in UNIV.\ (x + y)\$z) =$
$(\sum\ z \in UNIV.\ x\$z) + (\sum\ z \in UNIV.\ y\$z)$
by (*simp add*: *sum.distrib*)
hence $\forall\ x :: real\,\hat{}\,'b.\ \forall\ y.\ \forall\ u\ v.$
$(\sum\ z \in UNIV.\ (u *_R x + v *_R y)\$z) =$
$u *_R (\sum\ z \in UNIV.\ x\$z) + v *_R (\sum\ z \in UNIV.\ y\$z)$
using *scaleR-right.sum sum.cong vector-scaleR-component*
by (*metis* (*no-types*))
hence $\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u\ v.$
$(\sum\ z \in UNIV.\ (u *_R x + v *_R y)\$z) = u *_R 1 + v *_R 1$
by *simp*
hence $\forall\ x \in ?simplex.\ \forall\ y \in ?simplex.\ \forall\ u \geq 0.\ \forall\ v \geq 0.$
$u + v = 1 \longrightarrow u *_R x + v *_R y \in ?simplex$
by *simp*
ultimately show
$\bigcap\ \{t.\ (\forall\ x \in t.\ \forall\ y \in t.\ \forall\ u \geq 0.\ \forall\ v \geq 0.$
$u + v = 1 \longrightarrow u *_R x + v *_R y \in t)$

$\wedge$ *standard-basis* $\subseteq t\} \subseteq$ *?simplex*
by *blast*
next
show $\{v.\ (\forall\ i.\ 0 \leq v\$i) \wedge (\sum\ y \in UNIV.\ v\$y) = 1\} \subseteq$
$\bigcap\ \{t.\ (\forall\ x \in t.\ \forall\ y \in t.\ \forall\ u \geq 0.\ \forall\ v \geq 0.$
$u + v = 1 \longrightarrow u *_R x + v *_R y \in t)$
$\wedge$ (*standard-basis* :: (*real*^*'b*) *set*) $\subseteq t\}$
proof (*intro subsetI*)
fix
x :: *real*^*'b* **and**
X :: *real*^*'b set*
assume *convex-comb*:
$x \in \{v.\ (\forall\ i.\ 0 \leq v\$i) \wedge (\sum\ y \in UNIV.\ v\$y) = 1\}$
have $\forall\ v \in$ *standard-basis*. $\exists\ b.\ v\$b = 1 \wedge (\forall\ b' \neq b.\ v\$b' = 0)$
unfolding *standard-basis-def*
by *simp*
then obtain *ind* :: *real*^*'b* $\Rightarrow$ *'b* **where**
ind-eq-one: $\forall\ v \in$ *standard-basis*. $v\$(ind\ v) = 1$ **and**
ind-eq-zero: $\forall\ v \in$ *standard-basis*. $\forall\ b \neq (ind\ v).\ v\$b = 0$
by *metis*
hence $\forall\ v \in$ *standard-basis*. $\forall\ v' \in$ *standard-basis*.
$ind\ v = ind\ v' \longrightarrow (\forall\ b.\ v\$b = v'\$b)$
by *metis*
hence *inj-ind*:
$\forall\ v \in$ *standard-basis*. $\forall\ v' \in$ *standard-basis*.
$ind\ v = ind\ v' \longrightarrow v = v'$
unfolding *vec-eq-iff*
by *blast*
hence *inj-on ind standard-basis*
unfolding *inj-on-def*
by *blast*
hence *bij-ind-std*: *bij-betw ind standard-basis* (*ind* ‘ *standard-basis*)
unfolding *bij-betw-def*
by *simp*
obtain *ind-inv* :: *'b* $\Rightarrow$ *real*^*'b* **where**
char-vec: *ind-inv* = ($\lambda\ b.\ \chi\ i.$ *if* $i = b$ *then 1 else 0*)
by *blast*
hence *in-basis*: $\forall\ b.$ *ind-inv* $b \in$ *standard-basis*
unfolding *standard-basis-def*
by *simp*
moreover from *this*
have *ind-inv-map*: $\forall\ b.$ *ind* (*ind-inv* b) $= b$
using *char-vec ind-eq-zero ind-eq-one axis-def axis-nth zero-neq-one*
by *metis*
ultimately have $\forall\ b.\ \exists\ v.\ v \in$ *standard-basis* $\wedge\ b =$ *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-ind-std bij-betw-byWitness*[*of - ind*] *in-basis inj-ind*
unfolding *image-subset-iff*
by *simp*
obtain $f :: real^{\frown\prime}b \Rightarrow real$ **where**
func: $f = (\lambda\ v.\ \mathit{if}\ v \in \mathit{standard\text{-}basis}\ \mathit{then}\ x\$(\mathit{ind}\ v)\ \mathit{else}\ 0)$
by *blast*
hence $(\sum\ y \in \mathit{standard\text{-}basis}.\ f\ y) = (\sum\ v \in \mathit{standard\text{-}basis}.\ x\$(\mathit{ind}\ v))$
by *simp*
also have $\ldots = (\sum\ y \in \mathit{ind}\ `\ \mathit{standard\text{-}basis}.\ x\$y)$
using *bij-ind-std sum-comp*[*of ind - -* (\$) *x*]
by *simp*
finally have *sum-eq-one*: $(\sum\ y \in \mathit{standard\text{-}basis}.\ f\ y) = 1$
using *univ convex-comb*
by *simp*
have *nonneg*: $\forall\ v \in \mathit{standard\text{-}basis}.\ f\ v \geq 0$
using *func convex-comb*
by *simp*
have $\forall\ v \in \mathit{standard\text{-}basis}.\ (\chi\ i.\ x\$(\mathit{ind}\ v) * v\$i)$
$= (\chi\ i.\ \mathit{if}\ i = \mathit{ind}\ v\ \mathit{then}\ x\$(\mathit{ind}\ v)\ \mathit{else}\ 0)$
using *ind-eq-one ind-eq-zero*
by *fastforce*
hence
$\forall\ v \in \mathit{standard\text{-}basis}.$
$x\$(\mathit{ind}\ v) *_R v = (\chi\ i.\ \mathit{if}\ i = \mathit{ind}\ v\ \mathit{then}\ x\$(\mathit{ind}\ v)\ \mathit{else}\ 0)$
unfolding *scaleR-vec-def*
by *simp*
moreover have $(\sum\ x \in \mathit{standard\text{-}basis}.\ f\ x *_R x) =$
$(\sum\ v \in \mathit{standard\text{-}basis}.\ x\$(\mathit{ind}\ v) *_R v)$
unfolding *func*
by *simp*
ultimately have $(\sum\ x \in \mathit{standard\text{-}basis}.\ f\ x *_R x)$
$= (\sum\ v \in \mathit{standard\text{-}basis}.\ \chi\ i.\ \mathit{if}\ i = \mathit{ind}\ v\ \mathit{then}\ x\$(\mathit{ind}\ v)\ \mathit{else}\ 0)$
by *force*
also have $\ldots = (\sum\ b \in \mathit{UNIV}.\ \chi\ i.\ \mathit{if}\ i = \mathit{ind}\ (\mathit{ind\text{-}inv}\ b)$
$\mathit{then}\ x\$(\mathit{ind}\ (\mathit{ind\text{-}inv}\ b))\ \mathit{else}\ 0)$
using *bij-inv sum-comp*
unfolding *comp-def*
by *blast*
also have $\ldots = (\sum\ b \in \mathit{UNIV}.\ \chi\ i.\ \mathit{if}\ i = b\ \mathit{then}\ x\$b\ \mathit{else}\ 0)$
using *ind-inv-map*
by *presburger*
finally have $(\sum\ x \in \mathit{standard\text{-}basis}.\ f\ x *_R x) =$
$(\sum\ b \in \mathit{UNIV}.\ \chi\ i.\ \mathit{if}\ i = b\ \mathit{then}\ x\$b\ \mathit{else}\ 0)$
by *simp*
hence $(\sum\ x \in \mathit{standard\text{-}basis}.\ f\ x *_R x) = x$
unfolding *vec-eq-iff*
by *simp*
hence $\exists\ f :: real^{\frown\prime}b \Rightarrow real.$
$(\sum\ y \in \mathit{standard\text{-}basis}.\ f\ y) = 1 \wedge (\forall\ x \in \mathit{standard\text{-}basis}.\ f\ x \geq 0)$

```
      ∧ x = (∑ x ∈ standard-basis. f x *R x)
    using sum-eq-one nonneg
    by blast
  thus x ∈ ⋂ {t. (∀ x ∈ t. ∀ y ∈ t. ∀ u ≥ 0. ∀ v ≥ 0.
                  u + v = 1 ⟶ u *R x + v *R y ∈ t)
          ∧ (standard-basis :: (real^'b) set) ⊆ t}
    using convex-combination-in-convex-hull
    unfolding convex-def hull-def
    by blast
 qed
qed

lemma fract-distr-helper:
 fixes a b c :: int
 assumes c ≠ 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 anonymity-homogeneity-is-equivalence:
 fixes X :: ('a, 'v) Election set
 assumes ∀ E ∈ X. finite (voters-ℰ E)
 shows equiv X (anonymity-homogeneityℛ X)
proof (unfold equiv-def, safe)
 show refl-on X (anonymity-homogeneityℛ X)
  unfolding refl-on-def anonymity-homogeneityℛ.simps
  by blast
next
 show sym (anonymity-homogeneityℛ X)
  unfolding sym-def anonymity-homogeneityℛ.simps
  using sup-commute
  by simp
next
 show Relation.trans (anonymity-homogeneityℛ X)
 proof
  fix E E' F :: ('a, 'v) Election
  assume
   rel: (E, E') ∈ anonymity-homogeneityℛ X and
   rel': (E', F) ∈ anonymity-homogeneityℛ X
  hence finite (voters-ℰ E')
   unfolding anonymity-homogeneityℛ.simps
   using assms
   by fastforce
  from rel rel' have eq-frac:
   (∀ r. vote-fraction r E = vote-fraction r E') ∧
    (∀ r. vote-fraction r E' = vote-fraction r F)
   unfolding anonymity-homogeneityℛ.simps
   by blast
  hence ∀ r. vote-fraction r E = vote-fraction r F
```

by *metis*
thus $(E, F) \in$ *anonymity-homogeneity*$_{\mathcal{R}}$ X
using *rel rel′ snd-conv*
unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.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 $(\sum\ a \in A.\ \mathit{Fract}\ (f\ a)\ b) = \mathit{Fract}\ (\sum\ x \in A.\ f\ x)\ b$
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 $(\sum\ a \in A.\ \mathit{Fract}\ (f\ a)\ b) = 0 \wedge (\sum\ y \in A.\ f\ y) = 0$
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 $A \Longrightarrow b \neq 0 \Longrightarrow (\sum\ a \in A.\ \mathit{Fract}\ (f\ a)\ b) = \mathit{Fract}\ (\sum\ x \in A.\ f$
$x)\ b$
hence $A \neq \{\}$

by *auto*
then obtain $c :: 'x$ **where**
c-in-A: $c \in A$
by *blast*
hence $(\sum\ a \in A.\ \mathit{Fract}\ (f\ a)\ b) =$
$(\sum\ a \in A - \{c\}.\ \mathit{Fract}\ (f\ a)\ b) + \mathit{Fract}\ (f\ c)\ b$
using *fin-A*
by (*simp add*: *sum-diff1*)
also have $\ldots = \mathit{Fract}\ (\sum\ x \in A - \{c\}.\ f\ x)\ b + \mathit{Fract}\ (f\ c)\ b$
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 $\ldots = \mathit{Fract}\ (\sum\ x \in A.\ f\ x)\ b$
using *c-in-A fin-A b-non-zero fract-distr-helper*
by (*simp add*: *sum-diff1*)
finally show $(\sum\ a \in A.\ \mathit{Fract}\ (f\ a)\ b) = \mathit{Fract}\ (\sum\ x \in A.\ f\ x)\ b$
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 *anonymity-homogeneity*$_{\mathcal{Q}}$*-isomorphism*:
assumes *infinite* (*UNIV* :: *'v set*)
shows
bij-betw (*anonymity-homogeneity-class* :: (*'a* :: *finite*, *'v*) *Election set* $\Rightarrow$
rat$^\frown$*'a Ordered-Preference*) (*anonymity-homogeneity*$_{\mathcal{Q}}$ (*UNIV* :: *'a set*))
(*vote-simplex* :: (*rat*$^\frown$*'a Ordered-Preference*) *set*)
proof (*unfold bij-betw-def inj-on-def*, *intro conjI ballI impI*)
fix $X\ Y$ :: (*'a*, *'v*) *Election set*
assume
class-X: $X \in$ *anonymity-homogeneity*$_{\mathcal{Q}}$ *UNIV* **and**
class-Y: $Y \in$ *anonymity-homogeneity*$_{\mathcal{Q}}$ *UNIV* **and**
eq-vec: *anonymity-homogeneity-class* $X =$ *anonymity-homogeneity-class* Y
have *equiv*:
equiv (*elections-$\mathcal{A}$ UNIV*) (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-$\mathcal{A}$ UNIV*))
using *anonymity-homogeneity-is-equivalence CollectD IntD1 inf-commute*
unfolding *elections-$\mathcal{A}$.simps*
by (*metis* (*no-types*, *lifting*))
hence *subset*:

$X \neq \{\} \wedge X \subseteq$ *elections-$\mathcal{A}$ UNIV* $\wedge$ $Y \neq \{\} \wedge Y \subseteq$ *elections-$\mathcal{A}$ UNIV*
using *class-X class-Y in-quotient-imp-non-empty in-quotient-imp-subset*
unfolding *anonymity-homogeneity$_{\mathcal{Q}}$.simps*
by *blast*
then obtain E E' :: $('a, 'v)$ *Election* **where**
E-in-X: $E \in X$ **and**
E′-in-Y: $E' \in Y$
by *blast*
hence *class-X-E*: *anonymity-homogeneity$_{\mathcal{R}}$* (*elections-$\mathcal{A}$ UNIV*) `` $\{E\} = X$
using *class-X equiv Image-singleton-iff equiv-class-eq quotientE*
unfolding *anonymity-homogeneity$_{\mathcal{Q}}$.simps*
by (*metis* (*no-types*, *opaque-lifting*))
hence $\forall$ $F \in X$. $(E, F) \in$ *anonymity-homogeneity$_{\mathcal{R}}$* (*elections-$\mathcal{A}$ UNIV*)
unfolding *Image-def*
by *blast*
hence $\forall$ $F \in X$. $\forall$ p. *vote-fraction* p F = *vote-fraction* p E
unfolding *anonymity-homogeneity$_{\mathcal{R}}$.simps*
by *fastforce*
hence $\forall$ p. *vote-fraction* p ` $X = \{$*vote-fraction* p $E\}$
using *E-in-X*
by *blast*
hence $\forall$ 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*:
$\forall$ p. (*anonymity-homogeneity-class* X)\$$p$ = *vote-fraction* (*ord2pref* p) E
unfolding *anonymity-homogeneity-class.simps*
using *vec-lambda-beta*
by *metis*
have *class-Y-E′*: *anonymity-homogeneity$_{\mathcal{R}}$* (*elections-$\mathcal{A}$ UNIV*) `` $\{E'\} = Y$
using *class-Y equiv E′-in-Y Image-singleton-iff equiv-class-eq quotientE*
unfolding *anonymity-homogeneity$_{\mathcal{Q}}$.simps*
by (*metis* (*no-types*, *opaque-lifting*))
hence $\forall$ $F \in Y$. $(E', F) \in$ *anonymity-homogeneity$_{\mathcal{R}}$* (*elections-$\mathcal{A}$ UNIV*)
unfolding *Image-def*
by *blast*
hence $\forall$ $F \in Y$. $\forall$ p. *vote-fraction* p E' = *vote-fraction* p F
unfolding *anonymity-homogeneity$_{\mathcal{R}}$.simps*
by *blast*
hence $\forall$ p. *vote-fraction* p ` $Y = \{$*vote-fraction* p $E'\}$
using *E′-in-Y*
by *fastforce*
hence $\forall$ p. *vote-fraction$_{\mathcal{Q}}$* p Y = *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-Y-E′*:
$\forall$ p. (*anonymity-homogeneity-class* Y)\$$p$ = *vote-fraction* (*ord2pref* p) E'

unfolding *anonymity-homogeneity-class.simps*
using *vec-lambda-beta*
by *metis*
hence $\forall$ *p. vote-fraction* (*ord2pref p*) E = *vote-fraction* (*ord2pref p*) E'
using *eq-X-E eq-vec*
by *metis*
hence *eq-ord*: $\forall$ *p. linear-order p* $\longrightarrow$ *vote-fraction p* E = *vote-fraction p* E'
using *mem-Collect-eq pref2ord-inverse*
by *metis*
have ($\forall$ *v. v* $\in$ *voters-*$\mathcal{E}$ $E \longrightarrow$ *linear-order* (*profile-*$\mathcal{E}$ E *v*)) $\wedge$
($\forall$ *v. v* $\in$ *voters-*$\mathcal{E}$ $E' \longrightarrow$ *linear-order* (*profile-*$\mathcal{E}$ E' *v*))
using *subset E-in-X E'-in-Y*
unfolding *elections-*$\mathcal{A}$*.simps well-formed-elections-def profile-def*
by *fastforce*
hence $\forall$ *p.* $\neg$ *linear-order p* $\longrightarrow$ *vote-count p* E = *0* $\wedge$ *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.* $\neg$ *linear-order p* $\longrightarrow$ *vote-fraction p* E = *0* $\wedge$ *vote-fraction p* E' = *0*
using *int-ops rat-number-collapse*
by *simp*
hence $\forall$ *p. vote-fraction p* E = *vote-fraction p* E'
using *eq-ord*
by *metis*
hence (E, E') $\in$ *anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
using *subset E-in-X E'-in-Y elections-*$\mathcal{A}$*.simps*
unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.simps*
by *blast*
thus $X = Y$
using *class-X-E class-Y-E' equiv equiv-class-eq*
by (*metis* (*no-types*, *lifting*))
next
show (*anonymity-homogeneity-class* :: ('*a*, '*v*) *Election set*
$\Rightarrow$ *rat*$\widehat{\ }$'*a Ordered-Preference*)
' *anonymity-homogeneity*$_{\mathcal{Q}}$ *UNIV* = *vote-simplex*
proof (*unfold vote-simplex-def*, *safe*)
fix X :: ('*a*, '*v*) *Election set*
assume
quot: $X \in$ *anonymity-homogeneity*$_{\mathcal{Q}}$ *UNIV* **and**
not-simplex:
anonymity-homogeneity-class $X \notin$ *rat-vector-set* (*convex hull standard-basis*)
have *equiv-rel*:
equiv (*elections-*$\mathcal{A}$ *UNIV*) (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*))
using *anonymity-homogeneity-is-equivalence elections-*$\mathcal{A}$*.simps*
by *blast*
then obtain E :: ('*a*, '*v*) *Election* **where**
E-in-X: $E \in X$ **and**
X = *anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*) '' $\{E\}$
using *quot anonymity-homogeneity*$_{\mathcal{Q}}$*.simps equiv-Eps-in proj-Eps*

unfolding *proj-def*
by *metis*
hence *rel*: $\forall\ E' \in X.\ (E,\ E') \in$ *anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
by *simp*
hence $\forall\ p.\ \forall\ E' \in X.$
vote-fraction (*ord2pref p*) E' = *vote-fraction* (*ord2pref p*) E
unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.simps*
by *fastforce*
hence $\forall\ p.$ *vote-fraction* (*ord2pref p*) ' X = {*vote-fraction* (*ord2pref p*) E}
using *E-in-X*
by *blast*
hence *repr*: $\forall\ p.$ *vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) X = *vote-fraction* (*ord2pref p*) E
using *is-singletonI singleton-set-def-if-card-one the-elem-eq*
unfolding *vote-fraction*$_{\mathcal{Q}}$*.simps* $\pi_{\mathcal{Q}}$*.simps is-singleton-altdef*
by *metis*
have $\forall\ p.$ *vote-count* (*ord2pref p*) $E \geq 0$
by *simp*
hence $\forall\ p.$ *card* (*voters-*$\mathcal{E}$ E) $> 0 \longrightarrow$
Fract (*int* (*vote-count* (*ord2pref p*) E)) (*int* (*card* (*voters-*$\mathcal{E}$ E))) ≥ 0
using *zero-le-Fract-iff*
by *simp*
hence $\forall\ p.$ *vote-fraction* (*ord2pref p*) $E \geq 0$
unfolding *vote-fraction.simps card-gt-0-iff*
by *simp*
hence $\forall\ p.$ *vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) $X \geq 0$
using *repr*
by *simp*
hence *geq-zero*: $\forall\ p.$ *real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) X) ≥ 0
using *zero-le-of-rat-iff*
by *blast*
have *voters-*$\mathcal{E}$ E = {} $\vee$ *infinite* (*voters-*$\mathcal{E}$ E) $\longrightarrow$
($\forall\ p.$ *real-of-rat* (*vote-fraction p E*) = *0*)
by *simp*
hence *zero-case*:
voters-$\mathcal{E}$ E = {} $\vee$ *infinite* (*voters-*$\mathcal{E}$ E) $\longrightarrow$
($\chi\ p.$ *real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) X)) = *0*
using *repr*
unfolding *zero-vec-def*
by *simp*
let *?vote-sum* = ($\sum\ p \in$ *UNIV*. *vote-count p E*)
have *finite* (*UNIV* :: ($'a \times 'a$) *set*)
by *simp*
hence *eq-card*: *finite* (*voters-*$\mathcal{E}$ E) $\longrightarrow$ *card* (*voters-*$\mathcal{E}$ E) = *?vote-sum*
using *vote-count-sum*
by *metis*
hence *finite* (*voters-*$\mathcal{E}$ E) $\wedge$ *voters-*$\mathcal{E}$ $E \neq$ {} $\longrightarrow$
($\sum\ p \in$ *UNIV*. *vote-fraction p E*) =
($\sum\ p \in$ *UNIV*. *Fract* (*vote-count p E*) *?vote-sum*)
unfolding *vote-fraction.simps*

by *presburger*
moreover have *fin-imp-sum-gt-zero*:
finite (*voters-$\mathcal{E}$ E*) $\wedge$ *voters-$\mathcal{E}$ E* $\neq$ {} $\longrightarrow$ *?vote-sum* > *0*
using *eq-card*
by *fastforce*
hence *finite* (*voters-$\mathcal{E}$ E*) $\wedge$ *voters-$\mathcal{E}$ E* $\neq$ {} $\longrightarrow$
($\sum$ *p* $\in$ *UNIV*. *Fract* (*vote-count p E*) *?vote-sum*) = *Fract ?vote-sum ?vote-sum*
using *fract-distr*[*of UNIV ?vote-sum*] *card-0-eq eq-card of-nat-eq-0-iff*
finite-class.finite-UNIV of-nat-sum sum.cong
by (*metis* (*no-types*, *lifting*))
moreover have
finite (*voters-$\mathcal{E}$ E*) $\wedge$ *voters-$\mathcal{E}$ E* $\neq$ {} $\longrightarrow$ *Fract ?vote-sum ?vote-sum* = *1*
using *fin-imp-sum-gt-zero Fract-le-one-iff Fract-less-one-iff*
of-nat-0-less-iff order-le-less order-less-irrefl
by *metis*
ultimately have *fin-imp-sum-eq-one*:
finite (*voters-$\mathcal{E}$ E*) $\wedge$ *voters-$\mathcal{E}$ E* $\neq$ {}
$\longrightarrow$ ($\sum$ *p* $\in$ *UNIV*. *vote-fraction p E*) = *1*
by *presburger*
have $\forall$ *p*. $\neg$ *linear-order p* $\longrightarrow$ *vote-count p E* = *0*
using *E-in-X rel*
unfolding *anonymity-homogeneity$_{\mathcal{Q}}$.simps quotient-def vote-count.simps*
elections-$\mathcal{A}$.simps well-formed-elections-def profile-def
by *fastforce*
hence $\forall$ *p*. $\neg$ *linear-order p* $\longrightarrow$ *vote-fraction p E* = *0*
using *rat-number-collapse*
by *simp*
moreover have ($\sum$ *p* $\in$ *UNIV*. *vote-fraction p E*) =
($\sum$ *p* $\in$ {*r*. *linear-order r*}. *vote-fraction p E*) +
($\sum$ *p* $\in$ *UNIV* − {*r*. *linear-order r*}. *vote-fraction p E*)
using *finite CollectD Collect-mono UNIV-I add.commute*
sum.subset-diff top-set-def
by *metis*
ultimately have ($\sum$ *p* $\in$ *UNIV*. *vote-fraction p E*) =
($\sum$ *p* $\in$ {*r*. *linear-order r*}. *vote-fraction p E*)
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
($\sum$ *p* $\in$ *UNIV*. *vote-fraction p E*) =
($\sum$ *p* $\in$ *UNIV*. *vote-fraction* (*ord2pref p*) *E*)
using *comp-def sum-comp*
by *auto*
hence *finite* (*voters-$\mathcal{E}$ E*) $\wedge$ *voters-$\mathcal{E}$ E* $\neq$ {}
$\longrightarrow$ ($\sum$ *p* $\in$ *UNIV*. *real-of-rat* (*vote-fraction* (*ord2pref p*) *E*)) = *1*
using *fin-imp-sum-eq-one of-rat-1 of-rat-sum*
by *metis*

hence (χ *p. real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*)) = *0* $\vee$
((∀ *p.* (χ *p. real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*))\$*p* ≥ *0*)
∧ ($\sum$ *x* ∈ *UNIV*. (χ *p. real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*))\$*x*) = *1*)
using *zero-case repr geq-zero*
by *force*
moreover have
∀ *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-vector* (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-vector.simps*
using *vec-lambda-beta*
by *blast*
moreover have
∀ *p. the-inv real-of-rat*
((χ *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 *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*
hence ∀ *p. the-inv real-of-rat* (*real-of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*)) =
vote-fraction$_{\mathcal{Q}}$ (*ord2pref p*) *X*
using *Rats-eq-range-nat-to-rat-surj surj-nat-to-rat-surj*
by *blast*
moreover have
∀ *p. vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X* = (*anonymity-homogeneity-class X*)\$*p*
by *simp*
ultimately have
∀ *p.* (*rat-vector* (χ *p. of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*)))\$*p* =
(*anonymity-homogeneity-class X*)\$*p*
by *metis*
hence *rat-vector* (χ *p. of-rat* (*vote-fraction*$_{\mathcal{Q}}$ (*ord2pref p*) *X*))
= *anonymity-homogeneity-class X*
by *simp*
hence ∃ *x* ∈ {*x* ∈ *insert 0* (*convex hull standard-basis*). ∀ *i. x*\$*i* ∈ ℚ}.
rat-vector x = *anonymity-homogeneity-class X*
using *simplex-el*
by *blast*
hence *rat-vector 0* = *anonymity-homogeneity-class X*
using *not-simplex image-iff insertE mem-Collect-eq*
unfolding *rat-vector-set.simps*

by (*metis* (*mono-tags*, *lifting*))
thus *anonymity-homogeneity-class X* = *0*
unfolding *rat-vector.simps*
using *Rats-0 inv-of-rat of-rat-0 vec-lambda-unique zero-index*
by (*metis* (*no-types*, *lifting*))
next
have ∀ *E* ∈ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*))
‘‘ {(*UNIV*, {}, (λ *v*. {}))}. ∀ *r*. *vote-fraction r E* = *0*
unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.simps*
by *force*
moreover have
∀ *E* ∈ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*))
‘‘ {(*UNIV*, {}, (λ *v*. {}))}. *finite* (*voters-*$\mathcal{E}$ *E*)
unfolding *Image-def anonymity-homogeneity*$_{\mathcal{R}}$*.simps*
by *fastforce*
ultimately have *all-zero*:
∀ *r*. ∀ *E* ∈ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*))
‘‘ {(*UNIV*, {}, (λ *v*. {}))}. *vote-fraction r E* = *0*
by *blast*
moreover have (*UNIV*, {}, λ *v*. {})
∈ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*) ‘‘ {(*UNIV*, {}, λ *v*. {})})
unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.simps Image-def elections-*$\mathcal{A}$*.simps*
well-formed-elections-def profile-def
by *simp*
ultimately have ∀ *r*. *0* ∈ *vote-fraction r*
‘ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*))
‘‘ {(*UNIV*, {}, (λ *v*. {}))}
using *image-eqI*
by (*metis* (*mono-tags*, *lifting*))
hence
∀ *r*. *vote-fraction r*
‘ (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
‘‘ {(*UNIV*, {}, λ *v*. {})}) = {*0*}
using *all-zero*
by *blast*
hence ∀ *r* :: ′*a Ordered-Preference*. *vote-fraction*$_{\mathcal{Q}}$ (*ord2pref r*)
(*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
‘‘ {(*UNIV*, {}, λ *v*. {})}) = *0*
using *is-singletonI singleton-insert-inj-eq′ singleton-set-def-if-card-one*
unfolding *vote-fraction*$_{\mathcal{Q}}$*.simps* $\pi_{\mathcal{Q}}$*.simps singleton-set.simps*
is-singleton-altdef singleton-set.simps
by *metis*
hence ∀ *r* :: ′*a Ordered-Preference*.
(*anonymity-homogeneity-class* (*anonymity-homogeneity*$_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
‘‘ {(*UNIV*, {}, λ *v*. {})}))$*r* = *0*
unfolding *anonymity-homogeneity-class.simps*
using *vec-lambda-beta*
by (*metis* (*no-types*))
moreover have ∀ *r* :: ′*a Ordered-Preference*. *0*$*r* = *0*

by *simp*
ultimately have *anonymity-homogeneity-class*
($anonymity\text{-}homogeneity_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*)
‘‘ {(*UNIV*, {}, λ *v*. {})}) = (*0* :: *rat*^'*a Ordered-Preference*)
using *vec-eq-iff*
by (*metis* (*no-types*))
moreover have (*UNIV*, {}, λ *v*. {}) $\in$ *elections-*$\mathcal{A}$ *UNIV*
unfolding *elections-*$\mathcal{A}$*.simps well-formed-elections-def profile-def*
by *simp*
hence ($anonymity\text{-}homogeneity_{\mathcal{R}}$ (*elections-*$\mathcal{A}$ *UNIV*) ‘‘ {(*UNIV*, {}, λ *v*. {})})
$\in$ $anonymity\text{-}homogeneity_{\mathcal{Q}}$ *UNIV*
unfolding $anonymity\text{-}homogeneity_{\mathcal{Q}}$*.simps quotient-def*
by *blast*
ultimately show *0* $\in$ *anonymity-homogeneity-class* ‘ $anonymity\text{-}homogeneity_{\mathcal{Q}}$
UNIV
using *image-eqI*
by (*metis* (*no-types*))
next
fix *x* :: *rat*^'*a Ordered-Preference*
assume *x* $\in$ *rat-vector-set* (*convex hull standard-basis*)
— The following converts a rational vector x to real vector x'.
then obtain *x'* :: *real*^'*a Ordered-Preference* **where**
conv: *x'* $\in$ *convex hull standard-basis* **and**
inv: $\forall$ *p*. *x*\$*p* = *the-inv real-of-rat* (*x'*\$*p*) **and**
rat: $\forall$ *p*. *x'*\$*p* $\in$ $\mathbb{Q}$
unfolding *rat-vector-set.simps rat-vector.simps*
by *force*
hence *convex*: ($\forall$ *p*. *0* $\leq$ *x'*\$*p*) $\wedge$ ($\sum$ *y* $\in$ *UNIV*. *x'*\$*y*) = *1*
using *standard-simplex-rewrite*
by *blast*
have *map*: $\forall$ *p*. *real-of-rat* (*x*\$*p*) = *x'*\$*p*
using *inv rat f-the-inv-into-f inj-onCI of-rat-eq-iff*
unfolding *Rats-def*
by (*metis* (*no-types*))
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-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*
hence *fract'*: $\forall$ *p*. *x'*\$*p* = (*fst* (*fraction'* *p*)) / (*snd* (*fraction'* *p*))
using *map div-by-0 divide-less-cancel of-int-0 of-int-pos of-rat-rat*
by *metis*
hence $\forall$ *p*. (*fst* (*fraction'* *p*)) / (*snd* (*fraction'* *p*)) $\geq$ *0*
using *convex*
by *fastforce*
hence $\forall$ *p*. *fst* (*fraction'* *p*) $\geq$ *0*
using *pos' not-less of-int-0-le-iff of-int-pos zero-le-divide-iff*

by *metis*
hence $\forall\ p.\ fst\ (fraction'\ p) \in \mathbb{N} \wedge snd\ (fraction'\ p) \in \mathbb{N}$
using *pos' nonneg-int-cases of-nat-in-Nats order-less-le*
by *metis*
hence $\forall\ p.\ \exists\ (n :: nat)\ (m :: nat).$
$fst\ (fraction'\ p) = n \wedge snd\ (fraction'\ p) = m$
using *Nats-cases*
by *metis*
hence $\forall\ p.\ \exists\ m :: nat \times nat.\ fst\ (fraction'\ p) = int\ (fst\ m)$
$\wedge\ snd\ (fraction'\ p) = int\ (snd\ m)$
by *simp*
then obtain *fraction* :: *'a Ordered-Preference* $\Rightarrow$ $(nat \times nat)$ **where**
eq: $\forall\ p.\ fst\ (fraction'\ p) = int\ (fst\ (fraction\ p)) \wedge$
$snd\ (fraction'\ p) = int\ (snd\ (fraction\ p))$
by *metis*
hence *fract*: $\forall\ p.\ x'\$p = (fst\ (fraction\ p))\ /\ (snd\ (fraction\ p))$
using *fract'*
by *simp*
hence *pos*: $\forall\ p.\ 0 < snd\ (fraction\ p)$
using *eq pos'*
by *simp*
let $?prod = \prod\ p \in UNIV.\ snd\ (fraction\ p)$
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 *finite UNIV-I mod-mod-trivial mod-prod-eq mod-self prod-zero*
by (*metis* (*no-types*, *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-amount:
$voter\text{-}amount = (\lambda\ p \in UNIV.\ (fst\ (fraction\ p)) * (?prod\ div\ (snd\ (fraction\ p))))$
by *blast*
let $?voter\text{-}sum = (\sum\ p \in UNIV.\ (fst\ (fraction\ p)) * (?prod\ div\ (snd\ (fraction\ p))))$
have *rewrite-div*: $\forall\ p.\ ?prod\ div\ (snd\ (fraction\ p)) = ?prod\ /\ (snd\ (fraction\ p))$
using *div less-imp-of-nat-less nonzero-mult-div-cancel-right*
of-nat-less-0-iff of-nat-mult pos
by *metis*
hence $?voter\text{-}sum = (\sum\ p \in UNIV.\ (fst\ (fraction\ p)) * (?prod\ /\ (snd\ (fraction\ p))))$

using *def-amount*
by *simp*
hence *?voter-sum* = *?prod* ∗ ($\sum$ *p* ∈ *UNIV*. (*fst* (*fraction p*)) / (*snd* (*fraction p*)))
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*: *?voter-sum* = *?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* = *?voter-sum*
using *assms infinite-arbitrarily-large*
by *blast*
then obtain *part* :: *'a Ordered-Preference* ⇒ *'v set* **where**
partition: *V* = $\bigcup$ {*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 *def-amount 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*) ∈ *elections-𝒜 UNIV*
unfolding *elections-𝒜.simps well-formed-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
∀ *p*. ∀ *v*. (*v* ∈ {*v* ∈ *V*. *prof' v* = *p*}) = (*v* ∈ {*v* ∈ *V*. *prof v* = *ord2pref p*})
using *prof*
by (*simp add*: *ord2pref-inject*)
ultimately have
∀ *p* :: *'a Ordered-Preference*.
vote-fraction (*ord2pref p*) (*UNIV*, *V*, *prof*) = *Fract* (*voter-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 *def-amount card-V-eq-sum rewrite-sum*
  by *force*
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 following are the percentages of voters voting for each linearly ordered profile in (UNIV, V, prof) that equals the entries of the given vector.
ultimately have
  $\forall\ p :: 'a\ Ordered\text{-}Preference.$
    $vote\text{-}fraction\ (ord2pref\ p)\ (UNIV,\ V,\ prof) = x'\$p$
  using *fract*
  by *presburger*
moreover have
  $\forall\ E \in anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ prof)\}.$
    $\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)\ E =$
      $vote\text{-}fraction\ (ord2pref\ p)\ (UNIV,\ V,\ prof)$
  unfolding *anonymity-homogeneity*$_{\mathcal{R}}$*.simps*
  by *fastforce*
ultimately have *all-eq-vec*:
  $\forall\ p.\ \forall\ E \in anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)$
    $``\ \{(UNIV,\ V,\ prof)\}.\ vote\text{-}fraction\ (ord2pref\ p)\ E = x\$p$
  using *Re-complex-of-real Re-divide-of-real of-rat.rep-eq of-real-of-int-eq*
    injI of-rat-eq-iff the-inv-f-f rat inv
  by (*metis* (*mono-tags*, *opaque-lifting*))
moreover have
  $(UNIV,\ V,\ prof) \in anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ prof)\}$
  using *anonymity-homogeneity*$_{\mathcal{R}}$*.simps election*
  by *blast*
ultimately have $\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)$ ‘
  $anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ prof)\} \supseteq \{x\$p\}$
  using *image-insert insert-iff mk-disjoint-insert singletonD subsetI*
  by (*metis* (*no-types*, *lifting*))
hence $\forall\ p.\ vote\text{-}fraction\ (ord2pref\ p)$ ‘
  $anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)\ ``\ \{(UNIV,\ V,\ prof)\} = \{x\$p\}$
  using *all-eq-vec*
  by *blast*

hence $x = $ *anonymity-homogeneity-class*
$(anonymity\text{-}homogeneity_{\mathcal{R}}\ (elections\text{-}\mathcal{A}\ UNIV)$ `` $\{(UNIV,\ V,\ prof)\})$
using *is-singletonI singleton-inject singleton-set-def-if-card-one vec-lambda-unique*
unfolding *anonymity-homogeneity-class.simps is-singleton-altdef*
$vote\text{-}fraction_{\mathcal{Q}}.simps\ \pi_{\mathcal{Q}}.simps$
by (*metis* (*no-types*, *lifting*))
thus $x \in$ (*anonymity-homogeneity-class*
$::\ ('a,\ 'v)\ Election\ set \Rightarrow rat^{\frown}'a\ Ordered\text{-}Preference)$
` $anonymity\text{-}homogeneity_{\mathcal{Q}}\ UNIV$
unfolding $anonymity\text{-}homogeneity_{\mathcal{Q}}.simps$ *quotient-def*
using *election*
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 $d = (\lambda\ pair.\ d\ (fst\ pair)\ (snd\ pair))$

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

4.1.2 Conditions

definition *symmetric* :: *'a set* ⇒ *'a Distance* ⇒ *bool* **where**
symmetric S d ≡ ∀ *x y*. *x* ∈ *S* ∧ *y* ∈ *S* ⟶ *d x y* = *d y x*

definition *triangle-ineq* :: *'a set* ⇒ *'a Distance* ⇒ *bool* **where**
triangle-ineq S d ≡ ∀ *x y z*. *x* ∈ *S* ∧ *y* ∈ *S* ∧ *z* ∈ *S* ⟶ *d x z* ≤ *d x y* + *d y z*

definition *eq-if-zero* :: *'a set* ⇒ *'a Distance* ⇒ *bool* **where**
eq-if-zero S d ≡ ∀ *x y*. *x* ∈ *S* ∧ *y* ∈ *S* ⟶ *d x y* = *0* ⟶ *x* = *y*

definition *vote-distance* :: (*'a Vote set* ⇒ *'a Vote Distance* ⇒ *bool*) ⇒
'a Vote Distance ⇒ *bool* **where**
vote-distance π d ≡ *π* {(*A*, *p*). *linear-order-on A p* ∧ *finite A*} *d*

definition *election-distance* :: ((*'a*, *'v*) *Election set* ⇒
(*'a*, *'v*) *Election Distance* ⇒ *bool*) ⇒
(*'a*, *'v*) *Election Distance* ⇒ *bool* **where**
election-distance π d ≡ *π* {(*A*, *V*, *p*). *finite-profile V A p*} *d*

4.1.3 Standard-Distance Property

definition *standard* :: (*'a*, *'v*) *Election Distance* ⇒ *bool* **where**
standard d ≡
∀ *A A' V V' p p'*. *A* ≠ *A'* ∨ *V* ≠ *V'* ⟶ *d* (*A*, *V*, *p*) (*A'*, *V'*, *p'*) = ∞

4.1.4 Auxiliary Lemmas

fun *arg-min-set* :: (*'b* ⇒ *'a* :: *ord*) ⇒ *'b set* ⇒ *'b set* **where**
arg-min-set f A = *Collect* (*is-arg-min f* (λ *a*. *a* ∈ *A*))

lemma *arg-min-subset*:
fixes
B :: *'b set* **and**
f :: *'b* ⇒ *'a* :: *ord*
shows *arg-min-set f B* ⊆ *B*
unfolding *arg-min-set.simps is-arg-min-def*
by *safe*

lemma *sum-monotone*:
fixes
A :: *'a set* **and**
f g :: *'a* ⇒ *int*
assumes ∀ *a* ∈ *A*. *f a* ≤ *g a*
shows (∑ *a* ∈ *A*. *f a*) ≤ (∑ *a* ∈ *A*. *g a*)
using *assms*
proof (*induction A rule*: *infinite-finite-induct*)
case (*infinite A*)
fix *A* :: *'a set*
show *?case*

using *infinite*
by *simp*
next
case *empty*
show *?case*
by *simp*
next
case (*insert x F*)
fix
x :: $'a$ **and**
F :: $'a$ *set*
show *?case*
using *insert*
by *simp*
qed

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

lemma *uneq-ereal*:
fixes $x\ y$ :: *int*
assumes $x \leq y$
shows *ereal* (*real-of-int* x) $\leq$ *ereal* (*real-of-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-ord $r\ s\ a\ b = ((a \preceq_r b \wedge b \preceq_s a) \vee (b \preceq_r a \wedge a \preceq_s b))$

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

fun *pairwise-disagreements'* :: *'a set* ⇒ *'a Preference-Relation* ⇒
'a Preference-Relation ⇒ (*'a* × *'a*) *set* **where**
pairwise-disagreements' A r s =
Set.filter (λ (*a*, *b*). *a* ≠ *b* ∧ *neq-ord r s a b*) (*A* × *A*)

lemma *set-eq-filter*:
fixes
X :: *'a set* **and**
P :: *'a* ⇒ *bool*
shows {*x* ∈ *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 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 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 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 $x\ y :: {'a}\ Vote$
assumes $alts\text{-}\mathcal{V}\ x = alts\text{-}\mathcal{V}\ y$
shows $spearman\ x\ y =$
$(\sum\ a \in alts\text{-}\mathcal{V}\ x.\ abs\ (int\ (rank\ (pref\text{-}\mathcal{V}\ x)\ a) - int\ (rank\ (pref\text{-}\mathcal{V}\ y)\ a)))$
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 $total\text{-}invariance_{\mathcal{D}} :: {'x}\ Distance \Rightarrow {'x}\ rel \Rightarrow bool$ **where**
$total\text{-}invariance_{\mathcal{D}}\ d\ rel = is\text{-}symmetry\ (tup\ d)\ (Invariance\ (product\ rel))$

fun $invariance_{\mathcal{D}} :: {'y}\ Distance \Rightarrow {'x}\ set \Rightarrow {'y}\ set \Rightarrow$
$('x,\ 'y)\ binary\text{-}fun \Rightarrow bool$ **where**
$invariance_{\mathcal{D}}\ d\ X\ Y\ \varphi = is\text{-}symmetry\ (tup\ d)\ (Invariance\ (equivariance\ X\ Y\ \varphi))$

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

fun $distance\text{-}anonymity' :: ('a,\ 'v)\ Election\ set \Rightarrow$
$('a,\ 'v)\ Election\ Distance \Rightarrow bool$ **where**
$distance\text{-}anonymity'\ X\ d = invariance_{\mathcal{D}}\ d\ (carrier\ bijection_{\mathcal{VG}})\ X\ (\varphi\text{-}anon\ X)$

fun $distance\text{-}neutrality :: ('a,\ 'v)\ Election\ set \Rightarrow$
$('a,\ 'v)\ Election\ Distance \Rightarrow bool$ **where**
$distance\text{-}neutrality\ X\ d = invariance_{\mathcal{D}}\ d\ (carrier\ bijection_{\mathcal{AG}})\ X\ (\varphi\text{-}neutral\ X)$

fun $distance\text{-}reversal\text{-}symmetry :: ('a,\ 'v)\ Election\ set \Rightarrow$
$('a,\ 'v)\ Election\ Distance \Rightarrow bool$ **where**
$distance\text{-}reversal\text{-}symmetry\ X\ d =$
$invariance_{\mathcal{D}}\ d\ (carrier\ reversal_{\mathcal{G}})\ X\ (\varphi\text{-}reverse\ X)$

definition $distance\text{-}homogeneity' :: ('a,\ 'v :: linorder)\ Election\ set \Rightarrow$
$('a,\ 'v)\ Election\ Distance \Rightarrow bool$ **where**
$distance\text{-}homogeneity'\ X\ d \equiv total\text{-}invariance_{\mathcal{D}}\ d\ (homogeneity_{\mathcal{R}}'\ X)$

```
definition distance-homogeneity :: ('a, 'v) Election set ⇒
      ('a, 'v) Election Distance ⇒ bool where
  distance-homogeneity X d ≡ total-invariance_𝒟 d (homogeneity_ℛ X)
```

Auxiliary Lemmas

```
lemma rewrite-total-invariance_𝒟:
  fixes
    d :: 'x Distance and
    r :: 'x rel
  shows total-invariance_𝒟 d r = (∀ (x, y) ∈ r. ∀ (a, b) ∈ r. d a x = d b y)
proof (unfold total-invariance_𝒟.simps is-symmetry.simps product.simps, safe)
  fix a b x y :: 'x
  assume
    ∀ x y. (x, y) ∈ {(p, p').
      (fst p, fst p') ∈ r ∧ (snd p, snd p') ∈ r}
        ⟶ tup d x = tup d y and
    (a, b) ∈ r and
    (x, y) ∈ r
  thus d a x = d b y
    unfolding total-invariance_𝒟.simps is-symmetry.simps
    by simp
next
  fix a b x y :: 'x
  assume
    ∀ (x, y) ∈ r. ∀ (a, b) ∈ r. d a x = d b y and
    (fst (x, a), fst (y, b)) ∈ r and
    (snd (x, a), snd (y, b)) ∈ r
  hence d x a = d y b
    by auto
  thus tup d (x, a) = tup d (y, b)
    by simp
qed

lemma rewrite-invariance_𝒟:
  fixes
    d :: 'y Distance and
    X :: 'x set and
    Y :: 'y set and
    φ :: ('x, 'y) binary-fun
  shows invariance_𝒟 d X Y φ =
          (∀ x ∈ X. ∀ y ∈ Y. ∀ z ∈ Y. d y z = d (φ x y) (φ x z))
proof (unfold invariance_𝒟.simps is-symmetry.simps equivariance.simps, safe)
  fix
    x :: 'x and
    y z :: 'y
  assume
    x ∈ X and
    y ∈ Y and
```

$z \in Y$ **and**
$\forall\ x\ y.\ (x,\ y) \in \{((u,\ v),\ x,\ y).\ (u,\ v) \in Y \times Y$
$\wedge (\exists\ z \in X.\ x = \varphi\ z\ u \wedge y = \varphi\ z\ v)\}$
$\longrightarrow tup\ d\ x = tup\ d\ y$
thus $d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z)$
by *fastforce*
next
fix
$x :: 'x$ **and**
$a\ b :: 'y$
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

lemma *invar-dist-image*:
fixes
$d :: 'y$ *Distance* **and**
$G :: 'x$ *monoid* **and**
$Y\ Y' :: 'y$ *set* **and**
$\varphi :: ('x,\ 'y)$ *binary-fun* **and**
$y :: 'y$ **and**
$g :: 'x$
assumes
invar-d: $invariance_{\mathcal{D}}\ d\ (carrier\ G)\ Y\ \varphi$ **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 $((y,\ y'),\ ((\varphi\ g\ y),\ (\varphi\ g\ y'))) \in equivariance\ (carrier\ G)\ Y\ \varphi$
using Y'*-in-*Y *y-in-*Y *g-carrier*
unfolding *equivariance.simps*
by *blast*
hence *eq-dist*: $tup\ d\ ((\varphi\ g\ y),\ (\varphi\ g\ y')) = tup\ d\ (y,\ y')$
using *invar-d*
unfolding $invariance_{\mathcal{D}}$*.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*: *invariance*$_{\mathcal{D}}$ *swap* (*carrier bijection*$_{\mathcal{AG}}$)
UNIV ($\lambda\ \pi\ (A,\ q).\ (\pi$ ‘ A, *rel-rename* $\pi\ q$))
proof (*unfold rewrite-invariance*$_{\mathcal{D}}$, *safe*)
fix
$\pi :: 'a \Rightarrow 'a$ **and**
$A\ A' :: 'a\ set$ **and**
$q\ q' :: 'a\ rel$
assume $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$
hence *bij*-π: *bij* π
unfolding *bijection*$_{\mathcal{AG}}$-*def*
using *rewrite-carrier*
by *blast*
show *swap* $(A,\ q)\ (A',\ q') =$
swap (π ‘ A, *rel-rename* $\pi\ q$) (π ‘ A', *rel-rename* $\pi\ q'$)
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* $\pi\ q$) (π ‘ A', *rel-rename* $\pi\ q'$) $=$ *card ?swap-set'*
by *simp*
moreover have *bij-betw ?f ?swap-set ?swap-set'*
proof (*unfold bij-betw-def inj-on-def*, *intro conjI impI ballI*)
fix $x\ y :: 'a \times 'a$
assume
$x \in$ *?swap-set* **and**
$y \in$ *?swap-set* **and**
?f $x =$ *?f* y
hence
π (*fst* x) $= \pi$ (*fst* y) **and**
π (*snd* x) $= \pi$ (*snd* y)
by *auto*
hence
fst $x =$ *fst* y **and**
snd $x =$ *snd* y

using *bij-π bij-pointE*
by (*metis*, *metis*)
thus $x = y$
using *prod.expand*
by *metis*
next
show *?f ‘ ?swap-set* = *?swap-set′*
proof
have $\forall\ a\ b.\ (a,\ b) \in A \times A \longrightarrow (\pi\ a,\ \pi\ b) \in \pi\ ‘\ A \times \pi\ ‘\ A$
by *simp*
moreover have $\forall\ a\ b.\ a \neq b \longrightarrow\ \pi\ a \neq \pi\ b$
using *bij-π 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* ⊆ *?swap-set′*
by *auto*
next
have ∀ *a b*. (*a*, *b*) ∈ (*rel-rename π q*) ⟶ (*the-inv π a*, *the-inv π b*) ∈ *q*
unfolding *rel-rename.simps*
using *bij-π bij-is-inj the-inv-f-f*
by *fastforce*
moreover have
∀ *a b*. (*a*, *b*) ∈ (*rel-rename π q′*) ⟶ (*the-inv π a*, *the-inv π b*) ∈ *q′*
unfolding *rel-rename.simps*
using *bij-π bij-is-inj the-inv-f-f*
by *fastforce*
ultimately have
∀ *a b*. *neq-ord* (*rel-rename π q*) (*rel-rename π q′*) *a b*
⟶ *neq-ord q q′* (*the-inv π a*) (*the-inv π b*)
by *simp*
moreover have
∀ *a b*. (*a*, *b*) ∈ π ‘ *A* × π ‘ *A* ⟶ (*the-inv π a*, *the-inv π b*) ∈ *A* × *A*
using *bij-π 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-π 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-π*
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-π bij-is-inj inj-image-eq-iff
      by metis
    thus ?thesis
      using False
      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-map-permute:
  fixes
    d :: 'a Vote Distance and
```

n :: *Norm* **and**
$A\ A'$ :: *'a set* **and**
φ :: *nat* $\Rightarrow$ *nat* **and**
$p\ p'$ :: *'a Preference-Relation list*
assumes
perm: φ *permutes* $\{0\ ..< \mathit{length}\ p\}$ **and**
len-eq: *length* p = *length* p' **and**
sym-n: *symmetry* n
shows n (*map2* ($\lambda\ q\ q'.\ d\ (A, q)\ (A', q')$) $p\ p'$) =
n (*map2* ($\lambda\ q\ q'.\ d\ (A, q)\ (A', q')$) (*permute-list* φ p) (*permute-list* φ p'))
proof −
have *length* (*map2* ($\lambda\ x\ y.\ d\ (A, x)\ (A', y)$) $p\ p'$) = *length* p
using *len-eq*
by *simp*
hence n (*map2* ($\lambda\ q\ q'.\ d\ (A, q)\ (A', q')$) $p\ p'$) =
n (*permute-list* φ (*map2* ($\lambda\ x\ y.\ d\ (A, x)\ (A', y)$) $p\ p'$))
using *perm sym-n mset-permute-list atLeast-upt*
unfolding *symmetry-def*
by *fastforce*
thus *?thesis*
using *perm len-eq atLeast-upt permute-list-map*[*of* - - $\lambda\ (q, q').\ d\ (A, q)\ (A',$
$q')$]
by (*simp add*: *permute-list-zip*)
qed

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

lemma *linorder-rank-injective*:
fixes
V :: *'v* :: *linorder set* **and**
$v\ v'$ :: *'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 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' :: $'a$ *set* **and**
V V' :: $'v$ :: *linorder set* **and**
p 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**
— Use a total permutation function in order to apply facts such as *mset-permute-list*.
?perm-total = λ i. *if* i < *?len*
then card {$v \in$ *?rn-V*. $v < \pi$ (*?sl-V*!i)}
else i
assume *bij-π*: *bij* π
show *votewise-distance* d n (A, V, p) (A', V', p') =
votewise-distance d n *?rn1* *?rn2*
proof (*cases finite* $V \land V = V' \land (V \neq \{\} \lor A = A')$)
case *False*
— Case: Both distances are infinite.
hence *votewise-distance* d n (A, V, p) (A', V', p') = ∞
by *auto*
moreover have *infinite* $V \longrightarrow$ *infinite* *?rn-V*

using *bij-π bij-betw-finite bij-betw-subset subset-UNIV*
by *metis*
moreover have $V \neq V' \longrightarrow ?rn\text{-}V \neq ?rn\text{-}V'$
using *bij-π inj-image-mem-iff subsetI subset-antisym*
unfolding *bij-def*
by *metis*
ultimately show *votewise-distance d n* $(A,\ V,\ p)\ (A',\ V',\ p') =$
votewise-distance d n ?rn1 ?rn2
using *False*
by *auto*
next
case *True*
— Case: Both distances are finite.
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-π 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*]
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 *to-list V' p'* = *permute-list ?perm* (*to-list ?rn-V' ?rn-p'*)
unfolding *comp-def*
using *True bij-π to-list-permutes-under-bij*
by (*metis* (*no-types*))
hence *rn-list-perm-list-V'*:
to-list V' p' = *permute-list ?perm-total* (*to-list ?rn-V' ?rn-p'*)
using *lengths-eq permute-invariant-under-coinciding-funs*[*of to-list ?rn-V'*
?rn-p']
by *fastforce*
have *?perm-total permutes* $\{0\ ..<\ ?len\}$
proof −
have $\forall\ i\ j.\ i < ?len \wedge j < ?len \wedge i \neq j$
$\longrightarrow \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i) \neq \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!j)$
using *True bij-π bij-pointE 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*:
$\forall\ i.\ i < ?len \longrightarrow \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i) \in \pi\ `\ V$
using *True image-eqI length-map nth-mem to-list.simps*

sorted-list-of-set.set-sorted-key-list-of-set
by (*metis* (*no-types*))
ultimately have
$\forall\ i < ?len.\ \forall\ j < ?len.\ ?perm\text{-}total\ i = ?perm\text{-}total\ j \longrightarrow i = j$
using *True linorder-rank-injective Collect-cong finite-imageI*
by (*metis* (*no-types*, *lifting*))
hence *inj*: *inj-on ?perm-total* $\{0\ ..< ?len\}$
unfolding *inj-on-def*
by *simp*
have $\forall\ v' \in \pi$ ‘ $V.\ card\ \{v \in \pi$ ‘ $V.\ v < v'\} < card\ (\pi$ ‘ $V)$
using *True card-seteq finite-imageI less-irrefl linorder-not-le*
mem-Collect-eq subsetI
by (*metis* (*no-types*, *lifting*))
moreover have $\forall\ i < ?len.\ \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i) \in \pi$ ‘ V
using *in-bnds-imp-img-el*
by *simp*
moreover have *card* (π ‘ V) = *card V*
using *bij-π bij-betw-same-card bij-betw-subset top-greatest*
by *metis*
moreover have *card V* = *?len*
by *simp*
ultimately have
$\forall\ i.\ i < ?len \longrightarrow ?perm\text{-}total\ i \in \{0\ ..< ?len\}$
using *atLeast0LessThan lessThan-iff*
by (*metis* (*full-types*))
hence *?perm-total* ‘ $\{0\ ..< ?len\} \subseteq \{0\ ..< ?len\}$
by *force*
hence *bij-betw ?perm-total* $\{0\ ..< ?len\}\ \{0\ ..< ?len\}$
using *inj card-image card-subset-eq atLeast0LessThan finite-atLeastLessThan*
unfolding *bij-betw-def*
by *blast*
thus *?thesis*
using *atLeast0LessThan bij-imp-permutes*
by *fastforce*
qed
hence *votewise-distance d n ?rn1 ?rn2* =
n (*map2* ($\lambda\ q\ q'.\ d\ (A,\ q)\ (A',\ q')$)
(*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-map-permute*[*of - to-list ?rn-V ?rn-p*]
True assms len-V-rn-V-eq
by *force*
also have … = *n* (*map2* ($\lambda\ q\ q'.\ d\ (A,\ q)\ (A',\ q')$) (*to-list V p*) (*to-list V′ p′*))
using *rn-list-perm-list-V rn-list-perm-list-V′*
by *presburger*
finally show
votewise-distance d n $(A,\ V,\ p)\ (A',\ V',\ p')$ = *votewise-distance d n ?rn1 ?rn2*
using *True*
by *force*

```
  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 invariance_𝒟 d (carrier bijection_𝒜𝒢) UNIV vote-action
  shows distance-neutrality well-formed-elections (votewise-distance d n)
proof (unfold distance-neutrality.simps rewrite-invariance_𝒟, safe)
  fix
    A A' :: 'a set and
    V V' :: 'v :: linorder set and
    p p' :: ('a, 'v) Profile and
    π :: 'a ⇒ 'a
  assume
    carrier: π ∈ carrier bijection_𝒜𝒢 and
    valid: (A, V, p) ∈ well-formed-elections and
    valid': (A', V', p') ∈ well-formed-elections
  hence bij-π: bij π
    unfolding bijection_𝒜𝒢-def
    using rewrite-carrier
    by blast
  thus votewise-distance d n (A, V, p) (A', V', p') =
          votewise-distance d n
            (φ-neutral well-formed-elections π (A, V, p))
              (φ-neutral well-formed-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-neutral well-formed-elections π (A, V, p))
                (φ-neutral well-formed-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
      ... =
      n (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
      ... = n (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'))
      unfolding map-helper
```

    by *simp*
  also have
    $\ldots = (n\ (map2\ (\lambda\ q\ q'.\ d\ (A,\ q)\ (A',\ q'))\ (to\text{-}list\ V\ p)\ (to\text{-}list\ V'\ p')))$
    using *rewrite-invariance*$_\mathcal{D}$[*of d - UNIV vote-action*] *assms carrier*
      UNIV-I case-prod-conv
    unfolding *vote-action-def*
    by (*metis* (*no-types*, *lifting*))
  finally have *votewise-distance d n*
    (φ-*neutral well-formed-elections* π $(A,\ V,\ p)$)
      (φ-*neutral well-formed-elections* π $(A',\ V',\ p')$) =
    $n\ (map2\ (\lambda\ q\ q'.\ d\ (A,\ q)\ (A',\ q'))\ (to\text{-}list\ V\ p)\ (to\text{-}list\ V'\ p'))$
    by *simp*
  thus *?thesis*
    using *True*
    by *auto*
next
  case *False*
  hence $\neg\ (finite\ V \land V = V' \land (V \neq \{\} \lor \pi\ `\ A = \pi\ `\ A'))$
    using *bij-*π *bij-is-inj inj-image-eq-iff*
    by *metis*
  thus *?thesis*
    using *False valid valid'*
    by *force*
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 *nonempty-set*$_\mathcal{C}$ :: $('a,\ 'v)$ *Consensus* **where**

$nonempty\text{-}set_{\mathcal{C}}\ (A,\ V,\ p) = (A \neq \{\})$

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

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

Equal top ranked alternatives.

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

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

Equal votes.

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

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

Unanimity condition.

fun $unanimity_{\mathcal{C}}$ :: $('a,\ 'v)$ *Consensus* **where**
$unanimity_{\mathcal{C}}\ c = (nonempty\text{-}set_{\mathcal{C}}\ c \land nonempty\text{-}profile_{\mathcal{C}}\ c \land equal\text{-}top_{\mathcal{C}}\ c)$

Strong unanimity condition.

fun $strong\text{-}unanimity_{\mathcal{C}}$ :: $('a,\ 'v)$ *Consensus* **where**
$strong\text{-}unanimity_{\mathcal{C}}\ c = (nonempty\text{-}set_{\mathcal{C}}\ c \land nonempty\text{-}profile_{\mathcal{C}}\ c \land equal\text{-}vote_{\mathcal{C}}\ c)$

4.3.3 Properties

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

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

4.3.4 Auxiliary Lemmas

lemma *cons-anon-conj*:
fixes $c\ c'$ :: $('a,\ 'v)$ *Consensus*
assumes
consensus-anonymity c **and**

consensus-anonymity c'
shows *consensus-anonymity* $(\lambda\ e.\ c\ e \wedge c'\ e)$
proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
fix
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ q :: ('a, 'v)\ Profile$ **and**
$\pi :: 'v \Rightarrow 'v$
assume
bij-π: *bij* π **and**
renamed: *rename* $\pi\ (A, V, p) = (A', V', q)$ **and**
prof: *profile* $V\ A\ p$
hence *profile* $V'\ A'\ q$
using *rename-sound fst-conv rename.simps*
by *metis*
moreover assume
$c\ (A, V, p)$ **and**
$c'\ (A, V, p)$
ultimately show $c\ (A', V', q) \wedge c'\ (A', V', q)$
using *bij-π renamed 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 $\forall\ C'.\ C' \in \mathfrak{C} \longrightarrow$ *is-symmetry* C' (*Invariance rel*)
shows *is-symmetry* C (*Invariance rel*)
proof (*unfold is-symmetry.simps*, *intro allI impI*)
fix $E\ E' :: ('a, 'v)\ Election$
assume $(E, E') \in rel$
hence $\forall\ C' \in \mathfrak{C}.\ C'\ E = C'\ E'$
using *assms*
unfolding *is-symmetry.simps*
by *blast*
thus $C\ E = C\ E'$
unfolding *C-def*
by *blast*
qed

lemma *cons-anon-invariant*:
fixes
$c :: ('a, 'v)\ Consensus$ **and**
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ q :: ('a, 'v)\ Profile$ **and**

$\pi :: 'v \Rightarrow 'v$
assumes
anon: *consensus-anonymity* c **and**
bij-π: *bij* π **and**
prof-p: *profile* V A p **and**
renamed: *rename* π $(A,\ V,\ p) = (A',\ V',\ q)$ **and**
cond-c: c $(A,\ V,\ p)$
shows c $(A',\ V',\ q)$
proof $-$
have *profile* V' A' q
using *rename-sound bij-π renamed prof-p*
by *fastforce*
thus *?thesis*
using *anon cond-c renamed rename-finite bij-π 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
$A\ A' :: 'a$ *set* **and**
$V\ V' :: 'v$ *set* **and**
$p\ q :: ('a,\ 'v)$ *Profile* **and**
$\pi :: 'v \Rightarrow 'v$
assume
bij-π: *bij* π **and**
cond-b: b $(A,\ V,\ p)$ **and**
prof-p: *profile* V A p **and**
renamed: *rename* π $(A,\ V,\ p) = (A',\ V',\ q)$
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-π rename-sound*
by *fastforce*

ultimately have $b'\ x\ (A',\ V',\ q)$
  using *all-cond-anon bij-π 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
    $A\ A'$ :: *'a set* **and**
    $V\ V'$ :: *'v set* **and**
    $p\ q$ :: (*'a*, *'v*) *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume
    bij-π: *bij* π **and**
    renamed: *rename* $\pi\ (A,\ V,\ p) = (A',\ V',\ q)$
  hence *card* V = *card* V'
    using *rename.simps Pair-inject bij-betw-same-card*
      bij-betw-subset top-greatest
    by (*metis* (*mono-tags*, *lifting*))
  moreover assume $\textit{nonempty-profile}_{\mathcal{C}}\ (A,\ V,\ p)$
  ultimately show $\textit{nonempty-profile}_{\mathcal{C}}\ (A',\ V',\ q)$
    using *length-0-conv renamed*
    unfolding $\textit{nonempty-profile}_{\mathcal{C}}\textit{.simps}$
    by *auto*
qed

lemma *equal-top-cons'-anonymous*:
  fixes a :: *'a*
  shows *consensus-anonymity* ($\textit{equal-top}_{\mathcal{C}}'\ a$)
proof (*unfold consensus-anonymity-def Let-def*, *clarify*)
  fix
    $A\ A'$ :: *'a set* **and**
    $V\ V'$ :: *'v set* **and**
    $p\ q$ :: (*'a*, *'v*) *Profile* **and**
    π :: $'v \Rightarrow 'v$
  assume

bij-π: *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 $\forall\ v' \in V'.\ q\ v' = p\ ((\textit{the-inv}\ \pi)\ v')$
using *renamed*
by *auto*
moreover have $\forall\ v' \in V'.\ (\textit{the-inv}\ \pi)\ v' \in V$
using *bij-π renamed rename.simps bij-is-inj*
f-the-inv-into-f-bij-betw inj-image-mem-iff
by *fastforce*
moreover have *winner*: $\forall\ v \in V.\ \textit{above}\ (p\ v)\ a = \{a\}$
using *top-cons-a*
by *simp*
ultimately have $\forall\ v' \in V'.\ \textit{above}\ (q\ v')\ a = \{a\}$
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
$A\ A'$ :: *'a set* **and**
$V\ V'$ :: *'v set* **and**
$p\ q$ :: (*'a*, *'v*) *Profile* **and**
π :: $'v \Rightarrow 'v$
assume
bij-π: *bij* π **and**
renamed: *rename* π $(A,\ V,\ p) = (A',\ V',\ q)$
have $\forall\ v' \in V'.\ (\textit{the-inv}\ \pi)\ v' \in V$
using *bij-π renamed bij-is-inj inj-image-mem-iff*
f-the-inv-into-f-bij-betw
by *fastforce*
moreover assume *equal-vote*$_{\mathcal{C}}$*'* r $(A,\ V,\ p)$
ultimately show *equal-vote*$_{\mathcal{C}}$*'* r $(A',\ V',\ q)$
using *renamed*

by *force*
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 well-formed-elections nonempty-set*$_{\mathcal{C}}$
unfolding *well-formed-elections-def*
by *auto*

lemma *nonempty-profile*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality well-formed-elections nonempty-profile*$_{\mathcal{C}}$
unfolding *well-formed-elections-def*
by *auto*

lemma *equal-vote*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality well-formed-elections equal-vote*$_{\mathcal{C}}$
proof (*unfold well-formed-elections-def consensus-neutrality.simps is-symmetry.simps*,
intro allI impI,
unfold split-paired-all neutrality$_{\mathcal{R}}$*.simps action-induced-rel.simps*
voters-$\mathcal{E}$*.simps alternatives-*$\mathcal{E}$*.simps profile-*$\mathcal{E}$*.simps* φ*-neutral.simps*
extensional-continuation.simps equal-vote$_{\mathcal{C}}$*.simps equal-vote*$_{\mathcal{C}}$*′.simps*
alternatives-rename.simps case-prod-unfold mem-Collect-eq fst-conv
snd-conv mem-Sigma-iff conj-assoc If-def simp-thms, *safe*)
fix
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a, 'v)\ Profile$ **and**
$\pi :: 'a \Rightarrow 'a$ **and**
$r :: 'a\ rel$
assume
profile $V\ A\ p$ **and**
$(THE\ z.$
$(profile\ V\ A\ p \longrightarrow z = (\pi\ `\ A,\ V,\ rel\text{-}rename\ \pi \circ p))$
$\wedge\ (\neg\ profile\ V\ A\ p \longrightarrow z = undefined)) = (A',\ V',\ p')$
hence
equal-voters: $V' = V$ **and**
perm-profile: $p' = (\lambda\ x.\ \{(\pi\ a,\ \pi\ b)\ |\ a\ b.\ (a,\ b) \in p\ x\})$
unfolding *comp-def*
by (*simp*, *simp*)
have
$(\forall\ v \in V.\ p\ v = r)$
$\longrightarrow (\exists\ r'.\ \forall\ v \in V.\ \{(\pi\ a,\ \pi\ b)\ |\ a\ b.\ (a,\ b) \in p\ v\} = r')$
by *simp*
{
moreover assume $\forall\ v' \in V.\ p\ v' = r$
ultimately show $\exists\ r.\ \forall\ v \in V'.\ p'\ v = r$

using *equal-voters perm-profile*
by *metis*
}
assume $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$
hence *bij* π
using *rewrite-carrier*
unfolding *bijection*$_{\mathcal{AG}}$*-def*
by *blast*
hence $\forall$ a. *the-inv* π (π a) = a
using *bij-is-inj the-inv-f-f*
by *metis*
moreover have
($\forall$ $v \in V$. {(π a, π b) | a b. (a, b) $\in$ p v} = r) $\longrightarrow$
($\forall$ $v \in V$. {(*the-inv* π (π a), *the-inv* π (π b)) | a b. (a, b) $\in$ p v} =
{(*the-inv* π a, *the-inv* π b) | a b. (a, b) $\in$ r})
by *fastforce*
ultimately have
($\forall$ $v \in V$. {(π a, π b) | a b. (a, b) $\in$ p v} = r) $\longrightarrow$
($\forall$ $v \in V$. {(a, b) | a b. (a, b) $\in$ p v} =
{(*the-inv* π a, *the-inv* π b) | a b. (a, b) $\in$ r})
by *auto*
hence
($\forall$ $v' \in V$. {(π a, π b) | a b. (a, b) $\in$ p v'} = r)
$\longrightarrow$ ($\exists$ r'. $\forall$ $v' \in V$. p v' = r')
by *simp*
moreover assume $\forall$ $v' \in V'$. p' v' = r
ultimately show $\exists$ r'. $\forall$ $v' \in V$. p v' = r'
using *equal-voters perm-profile*
by *metis*
qed

lemma *strong-unanimity*$_{\mathcal{C}}$*-neutral*: *consensus-neutrality*
well-formed-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$_{\mathcal{R}}$ *well-formed-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-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-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-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.

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

fun *voters-determine-election* :: $('a, 'v, ('r\ Result))$ *Electoral-Module* $\Rightarrow$ *bool* **where**
voters-determine-election $m =$
$(\forall\ A\ V\ p\ p'.\ (\forall\ v \in V.\ p\ v = p'\ v) \longrightarrow m\ V\ A\ p = m\ V\ A\ p')$

lemma (**in** *result*) *electoral-modI*:
fixes m :: $('a, 'v, ('r\ Result))$ *Electoral-Module*
assumes $\forall\ A\ V\ p.\ profile\ V\ A\ p \longrightarrow well\text{-}formed\ A\ (m\ V\ A\ p)$
shows *electoral-module m*
unfolding *electoral-module.simps*
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* $\Rightarrow$
bool **where**
anonymity $m \equiv$
electoral-module $m \wedge$
$(\forall\ A\ V\ p\ \pi :: ('v \Rightarrow 'v).$
bij $\pi \longrightarrow (let\ (A',\ V',\ q) = (rename\ \pi\ (A,\ V,\ p))\ in$
$profile\ V\ A\ p \wedge profile\ V'\ A'\ q \longrightarrow m\ V\ A\ p = m\ V'\ A'\ q))$

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

fun *anonymity-in* :: $('a, 'v)$ *Election set* $\Rightarrow$ $('a, 'v, 'r)$ *Electoral-Module* $\Rightarrow$

bool **where**
anonymity-in X m = *is-symmetry* ($fun_{\mathcal{E}}$ *m*) (*Invariance* ($anonymity_{\mathcal{R}}$ *X*))

fun *anonymity'* :: (*'a*, *'v*, *'r*) *Electoral-Module* ⇒ *bool* **where**
anonymity' m = *anonymity-in well-formed-elections m*

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

fun *homogeneity* :: (*'a*, *'v*, *'b Result*) *Electoral-Module* ⇒ *bool* **where**
homogeneity m = *homogeneity-in finite-elections-*$\mathcal{V}$ *m*

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

fun *homogeneity'* :: (*'a*, *'v* :: *linorder*, *'b Result*) *Electoral-Module* ⇒ *bool* **where**
homogeneity' m = *homogeneity'-in finite-elections-*$\mathcal{V}$ *m*

lemma *hom-imp-anon*:
fixes
X :: (*'a*, *'v*) *Election set* **and**
m :: (*'a*, *'v*, (*'r Result*)) *Electoral-Module*
assumes
homogeneity-in X m **and**
∀ *E* ∈ *X*. *finite* (*voters-*$\mathcal{E}$ *E*)
shows *anonymity-in X m*
proof (*unfold anonymity-in.simps is-symmetry.simps*, *intro allI impI*)
fix *E E'* :: (*'a*, *'v*) *Election*
assume *rel*: (*E*, *E'*) ∈ $anonymity_{\mathcal{R}}$ *X*
then obtain π :: *'v* ⇒ *'v* **where**
π ∈ *carrier* $bijection_{\mathcal{VG}}$ **and**
E' = φ*-anon X* π *E*
unfolding $anonymity_{\mathcal{R}}$*.simps action-induced-rel.simps*
by *blast*
moreover from *this* **have** *bij* π
unfolding $bijection_{\mathcal{VG}}$*-def rewrite-carrier*
by *simp*
moreover from *this* **have** *in-election-set*: *E* ∈ *X*

using *rel*
unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
by *blast*
ultimately have *finite* (*voters-*$\mathcal{E}$ *E*$'$)
using *assms rename.simps rename-finite split-pairs*
unfolding φ*-anon.simps extensional-continuation.simps voters-*$\mathcal{E}$*.simps*
by *metis*
moreover have *fin-E*: *finite* (*voters-*$\mathcal{E}$ *E*)
using *in-election-set assms*
unfolding *anonymity*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
by *blast*
moreover have $\forall$ *r. vote-count r E* = *1* $*$ (*vote-count r E*$'$)
using *fin-E anon-rel-vote-count rel mult-1*
by *metis*
moreover have *alternatives-*$\mathcal{E}$ *E* = *alternatives-*$\mathcal{E}$ *E*$'$
using *fin-E anon-rel-vote-count rel*
by *metis*
ultimately show *fun*$_{\mathcal{E}}$ *m E* = *fun*$_{\mathcal{E}}$ *m E*$'$
using *assms in-election-set*
unfolding *homogeneity-in.simps is-symmetry.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-in* :: ($'$*a*, $'$*v*) *Election set* $\Rightarrow$
($'$*a*, $'$*v*, $'$*b Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
neutrality-in X m =
is-symmetry (*fun*$_{\mathcal{E}}$ *m*) (*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$) *X*
(φ*-neutral X*) (*result-action* ψ))

fun (**in** *result-properties*) *neutrality* :: ($'$*a*, $'$*v*, $'$*b Result*) *Electoral-Module* $\Rightarrow$
bool **where**
neutrality m = *neutrality-in well-formed-elections m*

4.4.4 Social-Welfare Properties

Reversal Symmetry

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

definition *reversal-symmetry-in* :: ($'$*a*, $'$*v*) *Election set* $\Rightarrow$
($'$*a*, $'$*v*, $'$*a rel Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
reversal-symmetry-in X m $\equiv$
is-symmetry (*fun*$_{\mathcal{E}}$ *m*) (*action-induced-equivariance* (*carrier reversal*$_{\mathcal{G}}$) *X*
(φ*-reverse X*) (*result-action* ψ*-reverse*))

fun *reversal-symmetry* :: $('a, 'v, 'a\ rel\ Result)$ *Electoral-Module* $\Rightarrow$ *bool* **where**
reversal-symmetry m = *reversal-symmetry-in well-formed-elections m*

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*
$\wedge\ (\forall\ p\ q.\ equiv\text{-}prof\text{-}except\text{-}a\ V\ A\ p\ q\ a \longrightarrow m\ V\ A\ p = m\ V\ A\ q)$

definition *unique-winner-if-profile-non-empty* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ *bool* **where**
unique-winner-if-profile-non-empty m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p*. (*A* ≠ {} ∧ *V* ≠ {} ∧ *profile V A p*) ⟶
(∃ *a* ∈ *A*. *m V A p* = ({*a*}, *A* − {*a*}, {})))

4.4.6 Equivalence Definitions

definition *prof-contains-result* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a* ⇒ *bool* **where**
prof-contains-result m V A p q a ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
profile V A p ∧ *profile V A q* ∧ *a* ∈ *A* ∧
(*a* ∈ *elect m V A p* ⟶ *a* ∈ *elect m V A q*) ∧
(*a* ∈ *reject m V A p* ⟶ *a* ∈ *reject m V A q*) ∧
(*a* ∈ *defer m V A p* ⟶ *a* ∈ *defer m V A q*)

definition *prof-leq-result* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a* ⇒ *bool* **where**
prof-leq-result m V A p q a ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
profile V A p ∧ *profile V A q* ∧ *a* ∈ *A* ∧
(*a* ∈ *reject m V A p* ⟶ *a* ∈ *reject m V A q*) ∧
(*a* ∈ *defer m V A p* ⟶ *a* ∉ *elect m V A q*)

definition *prof-geq-result* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a* ⇒ *bool* **where**
prof-geq-result m V A p q a ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
profile V A p ∧ *profile V A q* ∧ *a* ∈ *A* ∧
(*a* ∈ *elect m V A p* ⟶ *a* ∈ *elect m V A q*) ∧
(*a* ∈ *defer m V A p* ⟶ *a* ∉ *reject m V A q*)

definition *mod-contains-result* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a* ⇒ *bool* **where**
mod-contains-result m n V A p a ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
$\mathcal{SCF}$*-result.electoral-module n* ∧
profile V A p ∧ *a* ∈ *A* ∧
(*a* ∈ *elect m V A p* ⟶ *a* ∈ *elect n V A p*) ∧
(*a* ∈ *reject m V A p* ⟶ *a* ∈ *reject n V A p*) ∧
(*a* ∈ *defer m V A p* ⟶ *a* ∈ *defer n V A p*)

definition *mod-contains-result-sym* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a* ⇒ *bool* **where**

mod-contains-result-sym m n V A p a ≡
  $\mathcal{SCF}$*-result.electoral-module m* ∧
  $\mathcal{SCF}$*-result.electoral-module n* ∧
  profile V A p ∧ *a* ∈ *A* ∧
  (*a* ∈ *elect m V A p* ⟷ *a* ∈ *elect n V A p*) ∧
  (*a* ∈ *reject m V A p* ⟷ *a* ∈ *reject n V A p*) ∧
  (*a* ∈ *defer m V A p* ⟷ *a* ∈ *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 r 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*
  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*) ∪ (*reject m V A p*) ∪ (*defer m V A p*) = *A*
proof (*safe*)
  fix *a* :: *'a*

have
partition-imp-exist:
$\forall\ p'.$ *set-equals-partition* $A\ p'$
$\longrightarrow (\exists\ E\ R\ D.\ p' = (E,\ R,\ D) \wedge E \cup R \cup D = A)$ **and**
partition-A:
set-equals-partition $A\ (m\ V\ A\ p)$
using *assms*
by (*simp*, *simp*)
{
assume $a \in$ *elect* $m\ V\ A\ p$
with *partition-imp-exist partition-A*
show $a \in A$
using *UnI1 fstI*
by (*metis* (*no-types*))
}
{
assume $a \in$ *reject* $m\ V\ A\ p$
with *partition-imp-exist partition-A*
show $a \in A$
using *UnI1 fstI sndI subsetD sup-ge2*
by *metis*
}
{
assume $a \in$ *defer* $m\ V\ A\ p$
with *partition-imp-exist partition-A*
show $a \in A$
using *sndI subsetD sup-ge2*
by *metis*
}
{
assume
$a \in A$ **and**
$a \notin$ *defer* $m\ V\ A\ p$ **and**
$a \notin$ *reject* $m\ V\ A\ p$
with *partition-imp-exist partition-A*
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
$\mathcal{SCF}$*-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
  have wf: well-formed-SCF A (m V A p)
    using assms
    unfolding SCF-result.electoral-module.simps
    by metis
  have disj: disjoint3 (m V A p)
    using assms
    by simp
  {
    assume
      a ∈ elect m V A p and
      a ∈ reject m V A p
    with wf disj
    show a ∈ {}
      using prod.exhaust-sel DiffE UnCI result-imp-rej
      by (metis (no-types))
  }
  {
    assume
      elect-a: a ∈ elect m V A p and
      defer-a: a ∈ defer m V A p
    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)) ∧
          e (m V A p) ∩ r (m V A p) = {} ∧
          e (m V A p) ∩ d (m V A p) = {} ∧
          r (m V A p) ∩ d (m V A p) = {}
      using IntI emptyE prod.collapse disj disjoint3.simps
      by metis
    hence ((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) = {})
      using eq-snd-iff fstI
      by metis
    thus a ∈ {}
      using elect-a defer-a disjoint-iff-not-equal
      by (metis (no-types))
  }
  {
```

```
    assume
      a ∈ reject m V A p and
      a ∈ defer m V A p
    with wf disj
    show a ∈ {}
      using prod.exhaust-sel DiffE UnCI result-imp-rej
      by (metis (no-types))
  }
qed

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

lemma reject-in-alts:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    SCF-result.electoral-module m and
    profile V A p
  shows reject m V A p ⊆ A
  using le-supI1 assms result-presv-alts sup-ge2
  by metis

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
    SCF-result.electoral-module m and
    profile V A p
  shows defer m V A p ⊆ 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*
 using *assms card-mono*
 by (*metis elect-in-alts*,
   metis reject-in-alts,
   metis defer-in-alts)

lemma *reject-not-elected-or-deferred*:
 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 −
 from *assms* **have** (*elect m V A p*) ∪ (*reject m V A p*) ∪ (*defer m V A p*) = *A*
  using *result-presv-alts*
  by *blast*
 with *assms* **show** *?thesis*
  using *result-disj*
  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* $\cup$ *defer m V A p* $=$ *A* $-$ (*reject m V A p*)
proof $-$
from *assms* **have** (*elect m V A p*) $\cup$ (*reject m V A p*) $\cup$ (*defer m V A p*) $=$ *A*
using *result-presv-alts*
by *blast*
with *assms* **show** *?thesis*
using *result-disj*
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 $-$
from *assms* **have** (*elect m V A p*) $\cup$ (*reject m V A p*) $\cup$ (*defer m V A p*) $=$ *A*
using *result-presv-alts*
by *simp*
with *assms* **show** *?thesis*
using *result-disj*
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-elected-or-deferred*
by *metis*

lemma *mod-contains-result-comm*:
fixes
m 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*)
show
$\mathcal{SCF}$*-result.electoral-module* n **and**
$\mathcal{SCF}$*-result.electoral-module* m **and**
profile V A p **and**
$a \in A$
using *assms*
unfolding *mod-contains-result-def*
by *safe*
next
show
$a \in$ *elect* n V A $p \Longrightarrow a \in$ *elect* m V A p **and**
$a \in$ *reject* n V A $p \Longrightarrow a \in$ *reject* m V A p **and**
$a \in$ *defer* n V A $p \Longrightarrow a \in$ *defer* m V A p
using *assms IntI electoral-mod-defer-elem empty-iff result-disj*
unfolding *mod-contains-result-def*
by (*metis* (*mono-tags*, *lifting*),
metis (*mono-tags*, *lifting*),
metis (*mono-tags*, *lifting*))
qed

lemma *not-rej-imp-elec-or-defer*:
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-elected-or-deferred 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 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* $\subseteq$ *A* **and**
rejected-in-A: *reject m V A q* $\subseteq$ *A* **and**
deferred-in-A: *defer m V A q* $\subseteq$ *A*
using *mod-m prof-q*
by (*metis elect-in-alts*, *metis reject-in-alts*, *metis defer-in-alts*)
have
$\forall$ $a \in$ *elect m V A p*. $a \in$ *elect m V A q* **and**
$\forall$ $a \in$ *reject m V A p*. $a \in$ *reject m V A q* **and**
$\forall$ $a \in$ *defer m V A p*. $a \in$ *defer m V A q*
using *eq mod-m prof-p in-mono*
unfolding *prof-contains-result-def*
by (*metis* (*no-types*, *lifting*) *elect-in-alts*,
metis (*no-types*, *lifting*) *reject-in-alts*,
metis (*no-types*, *lifting*) *defer-in-alts*)
moreover have
$\forall$ $a \in$ *elect m V A q*. $a \in$ *elect m V A p* **and**

$\forall\ a \in reject\ m\ V\ A\ q.\ a \in reject\ m\ V\ A\ p$ **and**
$\forall\ a \in defer\ m\ V\ A\ q.\ a \in defer\ m\ V\ A\ p$
proof (*safe*)
fix $a :: 'a$
assume *q-elect-a*: $a \in elect\ m\ V\ A\ q$
hence $a \in A$
using *elected-in-A*
by *blast*
moreover have
$a \notin defer\ m\ V\ A\ q$ **and**
$a \notin reject\ m\ V\ A\ q$
using *q-elect-a prof-q mod-m result-disj disjoint-iff-not-equal*
by (*metis*, *metis*)
ultimately show $a \in elect\ m\ V\ A\ p$
using *eq electoral-mod-defer-elem*
unfolding *prof-contains-result-def*
by *metis*
next
fix $a :: 'a$
assume *q-rejects-a*: $a \in reject\ m\ V\ A\ q$
hence $a \in A$
using *rejected-in-A*
by *blast*
moreover have
$a \notin defer\ m\ V\ A\ q$ **and**
$a \notin elect\ m\ V\ A\ q$
using *q-rejects-a prof-q mod-m result-disj disjoint-iff-not-equal*
by (*metis*, *metis*)
ultimately show $a \in reject\ m\ V\ A\ p$
using *eq electoral-mod-defer-elem*
unfolding *prof-contains-result-def*
by *metis*
next
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
$a \notin elect\ m\ V\ A\ q$ **and**
$a \notin reject\ m\ V\ A\ q$
using *q-defers-a prof-q mod-m result-disj disjoint-iff-not-equal*
by (*metis*, *metis*)
ultimately show $a \in defer\ m\ V\ A\ p$
using *eq electoral-mod-defer-elem*
unfolding *prof-contains-result-def*
by *metis*
qed
ultimately show *?thesis*

using *prod.collapse subsetI subset-antisym*
by (*metis* (*no-types*))
qed

lemma *eq-def-and-elect-imp-eq*:
fixes
m n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p 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)) **and**
reject n V A q = A − ((*elect* n V A q) ∪ (*defer* n V A q))
using *elect-rej-def-combination result-imp-rej mod-m mod-n fin-p fin-q*
unfolding $\mathcal{SCF}$*-result.electoral-module.simps*
by (*metis*, *metis*)
thus *?thesis*
using *prod-eqI elec-eq def-eq*
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. (($A \neq \{\}$ ∧ *finite* A ∧ *profile* V A p) ⟶ *reject* m V A $p \neq A$))

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. ($A \neq \{\}$ ∧ *finite* A ∧ *profile* V A p) ⟶ *elect* m V A $p \neq \{\}$)

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 (*intro equalityI*)
show *elect-in-A*: *elect m V A p* ⊆ *A*
using *electing prof elect-in-alts*
unfolding *electing-def*
by *metis*
show *A* ⊆ *elect m V A p*
proof (*intro subsetI*)
fix *a* :: *'a*
assume *a* ∈ *A*
thus *a* ∈ *elect m V A p*
using *one-alt electing prof elect-in-A IntD2 Int-absorb2 card-1-singletonE card-gt-0-iff equals0I zero-less-one singletonD*
unfolding *electing-def*
by (*metis* (*no-types*))
qed
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 ∈ *A*
moreover have
$\mathcal{SCF}$*-result.electoral-module m* ∧

(∀ A V q. A ≠ {} ∧ *finite* A ∧ *profile* V A q ⟶ *elect* m V A q ≠ {})
using *assms*
unfolding *electing-def*
by *metis*
ultimately show a ∈ {}
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* ⇒ *bool* **where**
non-electing m ≡
𝒮𝒞ℱ*-result.electoral-module* m
∧ (∀ A V p. *profile* V A p ⟶ *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*:
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'*:
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* ⊆ *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* ⇒ *bool* **where**
defer-deciding m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧ *non-electing m* ∧ *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* ⇒ *bool* **where**
decrementing m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p*. *profile V A p* ∧ *card A > 1* ⟶ *card* (*reject m V A p*) ≥ *1*)

definition *defer-condorcet-consistency* :: (*'a*, *'v*, *'a Result*)
Electoral-Module ⇒ *bool* **where**
defer-condorcet-consistency m ≡

$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p a. condorcet-winner V A p a* ⟶
(*m V A p* = ({}, *A* − (*defer m V A p*), {*d* ∈ *A. condorcet-winner V A p d*})))

definition *condorcet-compatibility* :: ('*a*, '*v*, '*a Result*)
Electoral-Module ⇒ *bool* **where**
condorcet-compatibility m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p a. condorcet-winner V A p a* ⟶
(*a* ∉ *reject m V A p* ∧
(∀ *b.* ¬ *condorcet-winner V A p b* ⟶ *b* ∉ *elect m V A p*) ∧
(*a* ∈ *elect m V A p* ⟶
(∀ *b* ∈ *A.* ¬ *condorcet-winner V A p b* ⟶ *b* ∈ *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* ⇒ *bool* **where**
defer-monotonicity m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p q a.*
(*a* ∈ *defer m V A p* ∧ *lifted V A p q a*) ⟶ *a* ∈ *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* ⇒ *bool* **where**
defer-lift-invariance m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p q a.* (*a* ∈ (*defer m V A p*) ∧ *lifted V A p q a*)
⟶ *m V A p* = *m V A q*)

fun *dli-rel* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ ('*a*, '*v*) *Election rel* **where**
dli-rel m = {((*A*, *V*, *p*), (*A*, *V*, *q*)) |*A V p q.* (∃ *a* ∈ *defer m V A p. lifted V A p q a*)}

lemma *rewrite-dli-as-invariance*:
fixes *m* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
shows
defer-lift-invariance m =
($\mathcal{SCF}$-*result.electoral-module m*
∧ (*is-symmetry* ($fun_{\mathcal{E}}$ *m*) (*Invariance* (*dli-rel m*))))
proof (*unfold is-symmetry.simps*, *safe*)
assume *defer-lift-invariance m*
thus $\mathcal{SCF}$-*result.electoral-module m*
unfolding *defer-lift-invariance-def*
by *blast*
next
fix
A A' :: '*a set* **and**
V V' :: '*v set* **and**

$p\ q$ :: $('a,\ 'v)$ *Profile*
assume
invar: *defer-lift-invariance* m **and**
rel: $((A,\ V,\ p),\ (A',\ V',\ q)) \in$ *dli-rel* m
then obtain a :: $'a$ **where**
$a \in$ *defer* $m\ V\ A\ p \wedge$ *lifted* $V\ A\ p\ q\ a$
unfolding *dli-rel.simps*
by *blast*
moreover with *rel* **have** $A = A' \wedge V = V'$
by *simp*
ultimately show $fun_{\mathcal{E}}\ m\ (A,\ V,\ p) = fun_{\mathcal{E}}\ m\ (A',\ V',\ q)$
using *invar fst-eqD snd-eqD profile-$\mathcal{E}$.simps*
unfolding *defer-lift-invariance-def fun$_{\mathcal{E}}$.simps alternatives-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps*
by *metis*
next
assume
$\mathcal{SCF}$-result.electoral-module m **and**
$\forall\ E\ E'.\ (E,\ E') \in$ *dli-rel* $m \longrightarrow fun_{\mathcal{E}}\ m\ E = fun_{\mathcal{E}}\ m\ E'$
hence *$\mathcal{SCF}$-result.electoral-module* $m \wedge (\forall\ A\ V\ p\ q.$
$((A,\ V,\ p),\ (A,\ V,\ q)) \in$ *dli-rel* $m \longrightarrow m\ V\ A\ p = m\ V\ A\ q)$
unfolding *fun$_{\mathcal{E}}$.simps alternatives-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps voters-$\mathcal{E}$.simps*
using *fst-conv snd-conv*
by *metis*
moreover have
$\forall\ A\ V\ p\ q\ a.\ (a \in (\textit{defer}\ m\ V\ A\ p) \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow$
$((A,\ V,\ p),\ (A,\ V,\ q)) \in$ *dli-rel* m
unfolding *dli-rel.simps*
by *blast*
ultimately show *defer-lift-invariance* m
unfolding *defer-lift-invariance-def*
by *blast*
qed

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* :: $('a,\ 'v,\ 'a\ \textit{Result})$ *Electoral-Module* $\Rightarrow$
$('a,\ 'v,\ 'a\ \textit{Result})$ *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 ⇒ *bool* **where**
invariant-monotonicity m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p q a*. (*a* ∈ *elect m V A p* ∧ *lifted V A p q a*) ⟶
(*elect m V A q* = *elect m V A p* ∨ *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 ⇒ *bool* **where**
defer-invariant-monotonicity m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧ *non-electing m* ∧
(∀ *A V p q a*. (*a* ∈ *defer m V A p* ∧ *lifted V A p q a*) ⟶
(*defer m V A q* = *defer m V A p* ∨ *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 *defer m V A p* ≠ {*a*}
moreover 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*
ultimately have ∃ *b* ∈ *A*. *b* ≠ *a* ∧ *defer m V A p* = {*b*}
using *Suc-leI card-gt-0-iff def-one equals0D card-1-singletonE*
defer-in-alts insert-subset
unfolding *defer-deciding-def One-nat-def defers-def*
by *metis*
hence *a* ∉ *defer m V A p*
by *force*
hence *a* ∈ *reject m V A p*

using *ccomp c-win electoral-mod-defer-elem dd equals0D*
unfolding *defer-deciding-def non-electing-def condorcet-compatibility-def*
by *metis*
moreover have $a \notin$ *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*, *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 *c-winner*: *condorcet-winner* $V\ A\ p\ a$
hence *elect* $m\ V\ A\ p = \{\}$
using *dd*
unfolding *defer-deciding-def non-electing-def*
by *simp*
moreover have *defer* $m\ V\ A\ p = \{a\}$
using *c-winner dd ccomp ccomp-and-dd-imp-def-only-winner*
by *simp*
ultimately have $m\ V\ A\ p = (\{\},\ A -$ *defer* $m\ V\ A\ p,\ \{a\})$
using *c-winner reject-not-elected-or-deferred*
elect-rej-def-combination Diff-empty dd
unfolding *defer-deciding-def condorcet-winner.simps*
by *metis*
moreover have $\{a\} = \{c \in A.$ *condorcet-winner* $V\ A\ p\ c\}$
using *c-winner cond-winner-unique*
by *metis*
ultimately show
$m\ V\ A\ p = (\{\},\ A -$ *defer* $m\ V\ A\ p,\ \{c \in A.$ *condorcet-winner* $V\ A\ p\ c\})$
by *simp*
qed

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

theorem *disj-compat-comm*[*simp*]:
  fixes m 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 **and**
    $\mathcal{SCF}$*-result.electoral-module* n
    using *assms*
    unfolding *disjoint-compatibility-def*
    by *safe*
next
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set*
  obtain B :: $'a$ *set* **where**
    $B \subseteq A \wedge$
      $(\forall\ a \in B.$
        indep-of-alt $m\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ p)) \wedge$
      $(\forall\ a \in A - B.$
        indep-of-alt $n\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $n\ V\ A\ p))$
    using *assms*
    unfolding *disjoint-compatibility-def*
    by *metis*
  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* $n\ V\ A\ p)) \wedge$
      $(\forall\ a \in B.$
        indep-of-alt $m\ V\ A\ a \wedge (\forall\ p.$ *profile* $V\ A\ p \longrightarrow a \in$ *reject* $m\ V\ A\ p))$
    by *blast*
  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* ⇒
    bool **where**
  condorcet-consistency m ≡
    $\mathcal{SCF}$*-result.electoral-module* m ∧
    (∀ A V p a. *condorcet-winner* V A p a ⟶
      (m V A p = ({e ∈ A. *condorcet-winner* V A p e}, A − (*elect* m V A p), {})))

lemma *condorcet-consistency-equiv*:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  shows *condorcet-consistency* m =
        ($\mathcal{SCF}$*-result.electoral-module* m ∧
          (∀ A V p a. *condorcet-winner* V A p a ⟶
            (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* (*mono-tags*, *lifting*))
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**
    ∀ A V p a. *condorcet-winner* V A p a
        ⟶ m V A p = ({a}, A − *elect* m V A p, {})
  thus *condorcet-consistency* m
    using *cond-winner-unique*
    unfolding *condorcet-consistency-def*
    by (*metis* (*mono-tags*, *lifting*))
qed

lemma *condorcet-consistency-equiv′*:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  shows *condorcet-consistency* m =
        ($\mathcal{SCF}$*-result.electoral-module* m ∧

$(\forall\ A\ V\ p\ a.$
$\quad condorcet\text{-}winner\ V\ A\ p\ a \longrightarrow m\ V\ A\ p = (\{a\},\ A - \{a\},\ \{\})))$

proof (*unfold condorcet-consistency-equiv*, *safe*)
fix
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a,\ 'v)\ Profile$ **and**
$a :: 'a$
assume *condorcet-winner V A p a*
{
moreover assume
$\forall\ A\ V\ p\ a'.\ condorcet\text{-}winner\ V\ A\ p\ a'$
$\quad \longrightarrow m\ V\ A\ p = (\{a'\},\ A - elect\ m\ V\ A\ p,\ \{\})$
ultimately show $m\ V\ A\ p = (\{a\},\ A - \{a\},\ \{\})$
using *fst-conv*
by *metis*
}
{
moreover assume
$\forall\ A\ V\ p\ a'.\ condorcet\text{-}winner\ V\ A\ p\ a'$
$\quad \longrightarrow m\ V\ A\ p = (\{a'\},\ A - \{a'\},\ \{\})$
ultimately show $m\ V\ A\ p = (\{a\},\ A - elect\ m\ V\ A\ p,\ \{\})$
using *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* :: $('a,\ 'v,\ 'a\ Result)\ Electoral\text{-}Module \Rightarrow bool$ **where**
$monotonicity\ m \equiv$
$\mathcal{SCF}\text{-}result.electoral\text{-}module\ m\ \wedge$
$(\forall\ A\ V\ p\ q\ a.\ a \in elect\ m\ V\ A\ p \wedge lifted\ V\ A\ p\ q\ a \longrightarrow a \in elect\ m\ V\ A\ q)$

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 *is-symmetry* ($fun_{\mathcal{E}}$ m) (*Invariance* r) = $fun_{\mathcal{E}}$ *m respects r*
unfolding *is-symmetry.simps congruent-def*
by *blast*

lemma *invariance-is-congruence'*:
fixes
  f :: $'x \Rightarrow 'y$ **and**
  r :: $'x$ *rel*
shows *is-symmetry f* (*Invariance r*) = *f respects r*
unfolding *is-symmetry.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**
  is-symmetry ($fun_{\mathcal{E}}$ 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* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *set* $\Rightarrow$ ($'a$, $'v$) *Profile* $\Rightarrow$ *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* $\Rightarrow$ *bool* **where**
  condorcet-rating $f \equiv$
    $\forall\ A\ V\ p\ w\ .$ *condorcet-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.

fun *voters-determine-evaluation* :: $('a, 'v)$ *Evaluation-Function* $\Rightarrow$ *bool* **where**
  voters-determine-evaluation $f =$
    $(\forall\ A\ V\ p\ p'.\ (\forall\ v \in V.\ p\ v = p'\ v) \longrightarrow (\forall\ a \in A.\ f\ V\ a\ A\ p = f\ V\ a\ A\ p'))$

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 $e\ V\ a\ A\ p = Max\ \{e\ V\ b\ A\ p \mid b.\ b \in A\}$
proof $-$
  let $?set = \{e\ V\ b\ A\ p \mid b.\ b \in A\}$ **and**
      $?eMax = Max\ \{e\ V\ b\ A\ p \mid b.\ b \in A\}$ **and**
      $?eW = e\ V\ a\ A\ p$
  have $?eW \in\ ?set$
    using *CollectI winner*
    unfolding *condorcet-winner.simps*
    by (*metis* (*mono-tags*, *lifting*))
  moreover have $\forall\ e \in\ ?set.\ e \leq\ ?eW$
  proof (*safe*)
    fix b :: $'a$
    assume $b \in A$
    thus $e\ V\ b\ A\ p \leq e\ V\ a\ A\ p$
      using *less-imp-le rating winner order-refl*
      unfolding *condorcet-rating-def*
      by *metis*
  qed

```
  moreover have finite ?set
    using f-prof
    by simp
  moreover have ?set ≠ {}
    using winner
    unfolding condorcet-winner.simps
    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 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 ∈ A and
    loser: a ≠ b
  shows e V b A p < Max {e V c A p | c. c ∈ 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 ... = Max {e V c A p | c. c ∈ 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 = (*if finite A then* $\{a \in A\ .\ r\ (e\ V\ a\ A\ p)\ t\}$ *else* {})

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

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* ⇒
('*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.simps*
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.simps*
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.simps*
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.simps*
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.simps*
  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.simps*
  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.simps*
  by *auto*

4.7.5 Independence of Non-Voters

lemma *voters-determine-elim-mod*[*simp*]:
  fixes
    e :: ('*a*, '*v*) *Evaluation-Function* **and**
    t :: *Threshold-Value* **and**
    r :: *Threshold-Relation*
  assumes *voters-determine-evaluation* e
  shows *voters-determine-election* (*elimination-module* e t r)
proof (*unfold voters-determine-election.simps elimination-module.simps*, *safe*)
  fix
    A :: '*a set* **and**
    V :: '*v set* **and**
    p 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 *voters-determine-election.simps*
    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 *voters-determine-less-elim*[*simp*]:
fixes
e :: ('*a*, '*v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
assumes *voters-determine-evaluation e*
shows *voters-determine-election* (*less-eliminator e t*)
using *assms voters-determine-elim-mod*
unfolding *less-eliminator.simps voters-determine-election.simps*
by (*metis* (*full-types*))

lemma *voters-determine-leq-elim*[*simp*]:
fixes
e :: ('*a*, '*v*) *Evaluation-Function* **and**
t :: *Threshold-Value*
assumes *voters-determine-evaluation e*
shows *voters-determine-election* (*leq-eliminator e t*)
using *assms voters-determine-elim-mod*
unfolding *leq-eliminator.simps voters-determine-election.simps*
by (*metis* (*full-types*))

lemma *voters-determine-max-elim*[*simp*]:
fixes *e* :: ('*a*, '*v*) *Evaluation-Function*
assumes *voters-determine-evaluation e*
shows *voters-determine-election* (*max-eliminator e*)
proof (*unfold max-eliminator.simps voters-determine-election.simps*, *safe*)
fix
A :: '*a set* **and**
V :: '*v set* **and**
p 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 *voters-determine-evaluation.simps*
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 voters-determine-less-elim*
unfolding *voters-determine-election.simps*
by (*metis* (*no-types*, *lifting*))
qed

lemma *voters-determine-min-elim*[*simp*]:
fixes *e* :: ('*a*, '*v*) *Evaluation-Function*
assumes *voters-determine-evaluation e*

shows *voters-determine-election* (*min-eliminator* e)
proof (*unfold min-eliminator.simps voters-determine-election.simps*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p 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 *voters-determine-election.simps*
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 voters-determine-leq-elim*
unfolding *voters-determine-election.simps*
by (*metis* (*no-types*, *lifting*))
qed

lemma *voters-determine-less-avg-elim*[*simp*]:
fixes e :: ($'a$, $'v$) *Evaluation-Function*
assumes *voters-determine-evaluation* e
shows *voters-determine-election* (*less-average-eliminator* e)
proof (*unfold less-average-eliminator.simps voters-determine-election.simps*, *safe*)
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p 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 *voters-determine-election.simps*
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 voters-determine-less-elim*
unfolding *voters-determine-election.simps*
by (*metis* (*no-types*, *lifting*))
qed

lemma *voters-determine-leq-avg-elim*[*simp*]:
fixes e :: ($'a$, $'v$) *Evaluation-Function*
assumes *voters-determine-evaluation* e
shows *voters-determine-election* (*leq-average-eliminator* e)
proof (*unfold leq-average-eliminator.simps voters-determine-election.simps*, *safe*)

fix
  A :: $'a$ *set* **and**
  V :: $'v$ *set* **and**
  p 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 *voters-determine-election.simps*
  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 voters-determine-leq-elim*
  unfolding *voters-determine-election.simps*
  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 *SCF-result.electoral-module.simps*
by *auto*

lemma *min-elim-non-blocking*:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
shows *non-blocking* (*min-eliminator e*)
unfolding *non-blocking-def*
using *SCF-result.electoral-module.simps*
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 *SCF-result.electoral-module.simps*
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 *SCF-result.electoral-module.simps*
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 *force*

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 *force*

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 *force*

lemma *max-elim-non-electing*:
fixes e :: ($'a$, $'v$) *Evaluation-Function*
shows *non-electing* (*max-eliminator e*)
unfolding *non-electing-def*
by *force*

lemma *min-elim-non-electing*:
fixes e :: ($'a$, $'v$) *Evaluation-Function*
shows *non-electing* (*min-eliminator e*)
unfolding *non-electing-def*
by *force*

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 *force*

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 *force*
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 ∈ reject (max-eliminator e) V A p
  have e V a A p = Max {e V b A p | b. b ∈ A}
    using c-win cond-winner-imp-max-eval-val assms
    by fastforce
  hence a ∉ 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 ∈ elect (max-eliminator e) V A p
  moreover have a ∉ 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' :: 'a
  assume
    condorcet-winner V A p a and
    a ∈ elect (max-eliminator e) V A p
  thus a' ∈ reject (max-eliminator e) V A p
    using empty-iff max-elim-non-electing
    unfolding condorcet-winner.simps 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)
  show SCF-result.electoral-module (max-eliminator e)
    using max-elim-sound
    by metis
next
  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* | *b*. *b* ∈ *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*
using *f-prof*
by *fastforce*
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* − *defer* (*max-eliminator e*) *V A p*, {*a*})
using *elim-set winner*
by *auto*
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 $\equiv$
$\forall$ $A\ e\ e'\ d\ d'\ r\ r'.$
(*well-formed-$\mathcal{SCF}$* A $(e,\ r,\ d)$ $\wedge$ *well-formed-$\mathcal{SCF}$* A $(e',\ r',\ d')$) $\longrightarrow$
well-formed-$\mathcal{SCF}$ A (*agg* A $(e,\ r,\ d)$ $(e',\ r',\ d')$)

4.8.2 Properties

definition *agg-commutative* :: $'a$ *Aggregator* $\Rightarrow$ *bool* **where**
agg-commutative agg $\equiv$
aggregator agg $\wedge$ ($\forall$ $A\ e\ e'\ d\ d'\ r\ r'.$
agg A $(e,\ r,\ d)$ $(e',\ r',\ d')$ $=$ *agg* A $(e',\ r',\ d')$ $(e,\ r,\ d)$)

definition *agg-conservative* :: $'a$ *Aggregator* $\Rightarrow$ *bool* **where**
agg-conservative agg $\equiv$
aggregator agg $\wedge$
($\forall$ $A\ e\ e'\ d\ d'\ r\ r'.$
((*well-formed-$\mathcal{SCF}$* A $(e,\ r,\ d)$ $\wedge$ *well-formed-$\mathcal{SCF}$* A $(e',\ r',\ d')$) $\longrightarrow$
elect-r (*agg* A $(e,\ r,\ d)$ $(e',\ r',\ d')$) $\subseteq$ $(e \cup e')$ $\wedge$

$reject\text{-}r\ (agg\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq (r \cup r') \wedge$
$defer\text{-}r\ (agg\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) \subseteq (d \cup d')))$

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**
$max\text{-}aggregator\ A\ (e,\ r,\ d)\ (e',\ r',\ d') =$
$(e \cup e',$
$A - (e \cup e' \cup d \cup d'),$
$(d \cup d') - (e \cup e'))$

4.9.2 Auxiliary Lemma

lemma *max-agg-rej-set*:
fixes
$A\ e\ e'\ d\ d'\ r\ r'$ :: *'a set* **and**
a :: *'a*
assumes
wf-first-mod: $well\text{-}formed\text{-}\mathcal{SCF}\ A\ (e,\ r,\ d)$ **and**
wf-second-mod: $well\text{-}formed\text{-}\mathcal{SCF}\ A\ (e',\ r',\ d')$
shows $reject\text{-}r\ (max\text{-}aggregator\ A\ (e,\ r,\ d)\ (e',\ r',\ d')) = r \cap r'$
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 max-aggregator.simps well-formed-$\mathcal{SCF}$.simps disjoint3.simps set-equals-partition.simps*, *safe*)
  fix
    A e e' d d' r 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 e e' d d' r 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 e e' d d' r 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 ∈ e
    using a-not-in-e'
    by simp
next
  fix
    A e e' d d' r r' :: 'a set and
    a :: 'a
  assume
    wf-result: well-formed-SCF A (e', r', d') and
    reject-a: a ∈ reject-r (max-aggregator A (e, r, d) (e', r', d')) and
    a-not-in-r': a ∉ r'
  have a ∈ r ∪ r'
    using wf-result reject-a
    by force
  thus a ∈ r
    using a-not-in-r'
    by simp
next
  fix
    A e e' d d' r r' :: 'a set and
    a :: 'a
  assume
    defer-a: a ∈ defer-r (max-aggregator A (e, r, d) (e', r', d')) and
    a-not-in-d': a ∉ d'
  have a ∈ d ∪ d'
    using defer-a
    by force
  thus a ∈ 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.

type-synonym *'r Termination-Condition* = *'r Result* ⇒ *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.

fun *defer-equal-condition* :: *nat* ⇒ *'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.simps*
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 allI impI*)
  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**
  $\mathcal{K}_{\mathcal{E}}\ K\ w =$
    $\{(A,\ V,\ p)\ |\ A\ V\ p.\ (consensus\text{-}\mathcal{K}\ K)\ (A,\ V,\ p) \land finite\text{-}profile\ V\ A\ p$
      $\land\ elect\ (rule\text{-}\mathcal{K}\ K)\ V\ A\ p = \{w\}\}$

fun *elections-*$\mathcal{K}$ :: $('a,\ 'v,\ 'r\ Result)$ *Consensus-Class* $\Rightarrow$ $('a,\ 'v)$ *Election set* **where**
  elections-$\mathcal{K}$ $K = \bigcup\ ((\mathcal{K}_{\mathcal{E}}\ K)\ `\ UNIV)$

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 $c\ m \equiv$
   $\forall\ A\ V\ V'\ p\ p'.$
    $profile\ V\ A\ p \land profile\ V'\ A\ p' \land c\ (A,\ V,\ p) \land c\ (A,\ V',\ p')$
     $\longrightarrow m\ V\ A\ p = m\ V'\ A\ p'$

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* $\Rightarrow$
    $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow$
    $('a, 'v, 'a$ *Result*$)$ *Consensus-Class* **where**
  consensus-choice c m =
   (*let*
    $w = (\lambda\ V\ A\ p.$ *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*
    $(\lambda\ c.$ *nonempty-set*$_{\mathcal{C}}$ $c \wedge$ *nonempty-profile*$_{\mathcal{C}}$ c
      $\wedge$ *equal-top*$_{\mathcal{C}}{}'$ $a\ c)$ *elect-first-module*
proof (*unfold well-formed-def*, *safe*)
  fix
   a :: $'a$ **and**
   A :: $'a$ *set* **and**
   $V\ V'$ :: $'v$ :: *wellorder set* **and**
   $p\ p'$ :: $('a, 'v)$ *Profile*
  let *?cond* $= \lambda\ c.$ *nonempty-set*$_{\mathcal{C}}$ $c \wedge$ *nonempty-profile*$_{\mathcal{C}}$ $c \wedge$ *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 $\forall\ a' \in A.$
   $(($*above* $(p$ $($*least* $V))$ $a' = \{a'\}) = ($*above* $(p'$ $($*least* $V'))$ $a' = \{a'\}))$
  proof
   fix a' :: $'a$
   assume *a'-in-A*: $a' \in A$
   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' ≠ a
      have non-empty: V ≠ {} ∧ V' ≠ {}
        using not-empty-p not-empty-p'
        by simp
      hence A ≠ {} ∧ linear-order-on A (p (least V))
              ∧ linear-order-on A (p' (least V'))
        using not-empty-A not-empty-A' prof-p prof-p' enumerate-0
              a'-in-A card.remove enumerate-in-set finite-enumerate-in-set
              least.elims all-not-in-conv zero-less-Suc
        unfolding profile-def
        by metis
      hence (a ∈ above (p (least V)) a' ∨ a' ∈ above (p (least V)) a)
          ∧ (a ∈ above (p' (least V')) a' ∨ a' ∈ 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} ∧ above (p (least V)) a' = {a'}
            ⟶ a = a')
        ∧ (above (p' (least V')) a = {a} ∧ above (p' (least V')) a' = {a'}
            ⟶ a = a')
        by auto
      thus ?thesis
        using bot-nat-0.not-eq-extremum card-0-eq cond-Ap cond-Ap'
              enumerate-0 enumerate-in-set equal-top𝒞'.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-set𝒞 c ∧ nonempty-profile𝒞 c ∧ equal-vote𝒞' r c) elect-first-module
proof (unfold well-formed-def, clarify)
 fix
    a :: 'a and
    A :: 'a set and
    V V' :: 'v :: wellorder set and
    p p' :: ('a, 'v) Profile
  let ?cond = λ c. nonempty-set𝒞 c ∧ nonempty-profile𝒞 c ∧ equal-vote𝒞' 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-0
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 $\subseteq$ *well-formed-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 $E \in$ *well-formed-elections*
unfolding *well-formed-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-elections-*$\mathcal{V}$
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-elections*-$\mathcal{V}$
unfolding *finite-elections-def finite-elections*-$\mathcal{V}$*-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 $\equiv$ *consensus-choice* $unanimity_{\mathcal{C}}$ *elect-first-module*

Strong unanimity condition.

definition *strong-unanimity* :: ($'a$, $'v$ :: *wellorder*, $'a$ *Result*) *Consensus-Class* **where**
strong-unanimity $\equiv$ *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 =
is-symmetry (*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 =
is-symmetry (*elect-r* $\circ$ $fun_{\mathcal{E}}$ (*rule*-$\mathcal{K}$ C))
(*action-induced-equivariance* (*carrier* $bijection_{\mathcal{AG}}$) X (φ-*neutral* X) (*set-action* ψ))

fun *consensus-rule-reversal-symmetry* :: ($'a$, $'v$) *Election set* $\Rightarrow$

('a, 'v, 'a rel Result) *Consensus-Class* ⇒ *bool* **where**
consensus-rule-reversal-symmetry X C = is-symmetry (elect-r ∘ fun$_\mathcal{E}$ *(rule-*$\mathcal{K}$ *C))*
(action-induced-equivariance (carrier reversal$_\mathcal{G}$*) X (φ-reverse X) (set-action ψ-reverse))*

5.3.6 Inference Rules

lemma *if-else-cons-equivar*:
fixes
m n :: ('a, 'v, 'a Result) *Electoral-Module* **and**
c :: ('a, 'v) *Consensus* **and**
G :: 'b *set* **and**
X :: ('a, 'v) *Election set* **and**
φ :: ('b, ('a, 'v) *Election*) *binary-fun* **and**
ψ :: ('b, 'a) *binary-fun* **and**
f :: 'a *Result* ⇒ 'a *set*
defines
equivar ≡ *action-induced-equivariance G X φ (set-action ψ)* **and**
if-else-cons ≡ *(c, (λ V A p. if c (A, V, p) then m V A p else n V A p))*
assumes
equivar-m: *is-symmetry (f ∘ fun*$_\mathcal{E}$ *m) equivar* **and**
equivar-n: *is-symmetry (f ∘ fun*$_\mathcal{E}$ *n) equivar* **and**
invar-cons: *is-symmetry c (Invariance (action-induced-rel G X φ))*
shows *is-symmetry (f ∘ fun*$_\mathcal{E}$ *(rule-*$\mathcal{K}$ *if-else-cons))*
(action-induced-equivariance G X φ (set-action ψ))
proof *(unfold rewrite-equivariance, intro ballI impI)*
fix
E :: ('a, 'v) *Election* **and**
g :: 'b
assume
g-in-G: *g* ∈ *G* **and**
E-in-X: *E* ∈ *X*
show *(f ∘ fun*$_\mathcal{E}$ *(rule-*$\mathcal{K}$ *if-else-cons)) (φ g E) =*
set-action ψ g ((f ∘ fun$_\mathcal{E}$ *(rule-*$\mathcal{K}$ *if-else-cons)) E)*
proof *(cases c E)*
case *True*
hence *c (φ g E)*
using *invar-cons rewrite-invar-ind-by-act g-in-G E-in-X*
by *metis*
hence *(f ∘ fun*$_\mathcal{E}$ *(rule-*$\mathcal{K}$ *if-else-cons)) (φ g E) =*
(f ∘ fun$_\mathcal{E}$ *m) (φ g E)*
unfolding *if-else-cons-def*
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 g-in-G rewrite-equivariance*
unfolding *equivar-def*
by *(metis (mono-tags, lifting))*
also have *(f ∘ fun*$_\mathcal{E}$ *m) E =*

$(f \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ if\text{-}else\text{-}cons))\ E$
using *True E-in-X g-in-G invar-cons if-else-cons-def*
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 E-in-X*
by *metis*
hence $(f \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ if\text{-}else\text{-}cons))\ (\varphi\ g\ E) =$
$(f \circ fun_{\mathcal{E}}\ n)\ (\varphi\ g\ E)$
unfolding *if-else-cons-def*
by *simp*
also have $(f \circ fun_{\mathcal{E}}\ n)\ (\varphi\ g\ E) =$
$set\text{-}action\ \psi\ g\ ((f \circ fun_{\mathcal{E}}\ n)\ E)$
using *equivar-n E-in-X g-in-G rewrite-equivariance*
unfolding *equivar-def*
by (*metis* (*mono-tags*, *lifting*))
also have $(f \circ fun_{\mathcal{E}}\ n)\ E =$
$(f \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ if\text{-}else\text{-}cons))\ E$
using *False E-in-X g-in-G invar-cons*
unfolding *if-else-cons-def*
by *simp*
finally show *?thesis*
by *simp*
qed
qed

lemma *consensus-choice-anonymous*:
fixes
$\alpha\ \beta$ :: ($'a$, $'v$) *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
$A\ A'$ :: $'a$ *set* **and**
$V\ V'$ :: $'v$ *set* **and**
$p\ q$ :: ($'a$, $'v$) *Profile* **and**
π :: $'v \Rightarrow\ 'v$
assume
bij-π: *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-*π *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'] *bij-*π
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-*π
prof-p prof-q
by *simp*
have $\exists\ x.\ \beta'\ x\ (A,\ V,\ p)$
using *beta-Ap beta-sat*
by *simp*
then obtain $x :: 'b$ **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-*π *prof-p renamed*
cons-anon-invariant prof-q
unfolding *consensus-anonymity-def*
by *blast*
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*
    unfolding *strong-unanimity*$_{\mathcal{C}}$*.simps*
    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-equivariant*:
fixes
G :: $'b$ *set* **and**
E :: $('a, 'v)$ *Election set* **and**
φ :: $('b, ('a, 'v)$ *Election*) *binary-fun* **and**
ψ :: $('b, 'a)$ *binary-fun*
shows *is-symmetry* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ *defer-module*)
(*action-induced-equivariance* G E φ (*set-action* ψ))
using *rewrite-equivariance*
by *fastforce*

lemma *elect-first-winners-neutral*: *is-symmetry* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ *elect-first-module*)
(*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$)
well-formed-elections (φ*-neutral well-formed-elections*)
(*set-action* ψ*-neutral*$_{\mathrm{c}}$))
proof (*unfold rewrite-equivariance*, *clarify*)
fix
A :: $'a$ *set* **and**
V :: $'v$ :: *wellorder set* **and**
p :: $('a, 'v)$ *Profile* **and**
π :: $'a \Rightarrow 'a$
assume
bij-carrier-π: $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$ **and**
valid: $(A, V, p) \in$ *well-formed-elections*
hence *bijective-*π: *bij* π
unfolding *bijection*$_{\mathcal{AG}}$*-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-carrier-*π *valid* **have**
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ *elect-first-module*)
(φ*-neutral well-formed-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. $\{b.\ (a, b) \in \{(\pi\ a, \pi\ b) \mid a\ b.\ (a, b) \in p$ (*least* V)$\}\} = \{a\}\}$
unfolding *above-def*
by *simp*
ultimately have *elect-simp*:
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ *elect-first-module*)
(φ*-neutral well-formed-elections* π (A, V, p)) =

{a ∈ π ‘ A. {b. (a, b) ∈ {(π a, π b) | a b. (a, b) ∈ p (*least* V)}} = {a}}
by *simp*
have ∀ a ∈ π ‘ A. {b. (a, b) ∈ {(π x, π y) | x y. (x, y) ∈ 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 *blast*
hence (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*)
(φ*-neutral well-formed-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 . . . = π ‘ {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*
finally have
(*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*)
(φ*-neutral well-formed-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 *bijective-*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-*neutral well-formed-elections* π (A, V, p)) =
      π ‘ {$a \in A$. *above* (p (*least* V)) a = {a}}
  by *presburger*
moreover have
  elect elect-first-module V A p = {$a \in A$. *above* (p (*least* V)) a = {a}}
  by *simp*
moreover have *set-action* ψ-*neutral*$_c$ 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-*neutral well-formed-elections* π (A, V, p)) =
   set-action ψ-*neutral*$_c$ π
      ((*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*) (A, V, p))
  by *blast*
qed

lemma *strong-unanimity-neutral*:
  defines *domain* ≡ *well-formed-elections* ∩ *Collect strong-unanimity*$_{\mathcal{C}}$
  — We want to show neutrality on a set as general as possible, as this implies subset neutrality.
  shows $\mathcal{SCF}$-*properties.consensus-rule-neutrality domain strong-unanimity*
proof –
  have *coincides*:
    ∀ π. ∀ $E \in$ *domain*. φ-*neutral domain* π E =
      φ-*neutral well-formed-elections* π E
    unfolding *domain-def* φ-*neutral.simps*
    by *auto*
  hence *neutrality*$_{\mathcal{R}}$ *domain* ⊆ *neutrality*$_{\mathcal{R}}$ *well-formed-elections*
    unfolding *neutrality*$_{\mathcal{R}}$*.simps action-induced-rel.simps*
    using *domain-def*
    by *auto*
  hence *consensus-neutrality domain strong-unanimity*$_{\mathcal{C}}$
    using *strong-unanimity*$_{\mathcal{C}}$*-neutral invar-under-subset-rel*
    unfolding *consensus-neutrality.simps*
    by *blast*
  hence *is-symmetry strong-unanimity*$_{\mathcal{C}}$
    (*Invariance* (*action-induced-rel* (*carrier bijection*$_{\mathcal{AG}}$)
          domain (φ-*neutral well-formed-elections*)))
    unfolding *consensus-neutrality.simps neutrality*$_{\mathcal{R}}$*.simps*
    using *coincides coinciding-actions-ind-equal-rel*
    by *metis*
  moreover have *is-symmetry* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ *elect-first-module*)
      (*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$)
        domain (φ-*neutral well-formed-elections*) (*set-action* ψ-*neutral*$_c$))
    using *elect-first-winners-neutral*

unfolding *domain-def action-induced-equivariance-def*
using *equivar-under-subset*
by *blast*
ultimately have *is-symmetry* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*))
(*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$) *domain*
(φ*-neutral well-formed-elections*) (*set-action* ψ*-neutral*$_{c}$))
using *defer-winners-equivariant*[*of - domain -* ψ*-neutral*$_{c}$]
if-else-cons-equivar[*of - - - domain -* ψ*-neutral*$_{c}$ *- strong-unanimity*$_{\mathcal{C}}$]
unfolding *strong-unanimity-def*
by *fastforce*
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* $\subseteq$ *well-formed-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$
unfolding *well-formed-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*-neutral* (*well-formed-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$) π E =
φ*-neutral* (*elections-*$\mathcal{K}$ *strong-unanimity*) π E
unfolding φ*-neutral.simps*
using *extensional-continuation-subset*
by (*metis* (*no-types*, *lifting*))
ultimately have
is-symmetry (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *strong-unanimity*))
(*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$) (*elections-*$\mathcal{K}$ *strong-unanimity*)
(φ*-neutral* (*well-formed-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$))
(*set-action* ψ*-neutral*$_{c}$))
using *strong-unanimity-neutral equivar-under-subset*
unfolding *action-induced-equivariance-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*
by (*metis* (*no-types*))
qed

lemma *strong-unanimity-closed-under-neutrality*: *closed-restricted-rel*
(*neutrality*$_{\mathcal{R}}$ *well-formed-elections*) *well-formed-elections*
(*elections-*$\mathcal{K}$ *strong-unanimity*)
proof (*unfold closed-restricted-rel.simps restricted-rel.simps neutrality*$_{\mathcal{R}}$*.simps*
action-induced-rel.simps elections-$\mathcal{K}$*.simps*, *safe*)
fix

$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'b\ set$ **and**
$p\ p' :: ('a,\ 'b)$ *Profile* **and**
$\pi :: 'a \Rightarrow 'a$ **and**
$a :: 'a$
assume
prof: $(A,\ V,\ p) \in$ *well-formed-elections* **and**
cons: $(A,\ V,\ p) \in \mathcal{K}_{\mathcal{E}}$ *strong-unanimity a* **and**
bij-carrier-π: $\pi \in$ *carrier bijection*$_{\mathcal{AG}}$ **and**
img: *φ-neutral well-formed-elections* $\pi\ (A,\ V,\ p) = (A',\ V',\ p')$
hence *fin*: $(A,\ V,\ p) \in$ *finite-elections*
unfolding *$\mathcal{K}_{\mathcal{E}}$.simps finite-elections-def*
by *simp*
hence *valid'*: $(A',\ V',\ p') \in$ *well-formed-elections*
using *bij-carrier-π img φ-neutral-action.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-neutral.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-carrier-π CollectD finite-imageI fst-eqD snd-eqD*
unfolding *finite-elections-def well-formed-elections-def alternatives-$\mathcal{E}$.simps*
voters-$\mathcal{E}$.simps profile-$\mathcal{E}$.simps
by (*metis* (*no-types*, *lifting*))
let *?domain* = *well-formed-elections* $\cap$ *Collect strong-unanimity*$_{\mathcal{C}}$
have $((A,\ V,\ p),\ (A',\ V',\ p')) \in$ *neutrality*$_{\mathcal{R}}$ *well-formed-elections*
using *bij-carrier-π img fin valid'*
unfolding *neutrality$_{\mathcal{R}}$.simps action-induced-rel.simps*
finite-elections-def well-formed-elections-def
by *blast*
moreover have *unanimous*: $(A,\ V,\ p) \in$ *?domain*
using *cons fin*
unfolding *$\mathcal{K}_{\mathcal{E}}$.simps strong-unanimity-def well-formed-elections-def*
by *simp*
ultimately have *unanimous'*: $(A',\ V',\ p') \in$ *?domain*
using *strong-unanimity$_{\mathcal{C}}$-neutral valid'*
unfolding *consensus-neutrality.simps*
by *force*
have *rewrite*: $\forall\ \pi \in$ *carrier bijection*$_{\mathcal{AG}}$.
φ-neutral ?domain $\pi\ (A,\ V,\ p) \in$ *?domain*
$\longrightarrow$ (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ strong-unanimity*))
(*φ-neutral ?domain* $\pi\ (A,\ V,\ p)$) =
set-action ψ-neutral$_{\mathrm{c}}$ π
((*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ strong-unanimity*)) $(A,\ V,\ p)$)
using *strong-unanimity-neutral unanimous*

```
      rewrite-equivariance[of elect-r ∘ fun_𝓔 (rule-𝒦 strong-unanimity) carrier
bijection_𝒜𝒢]
    unfolding 𝒮𝒞ℱ-properties.consensus-rule-neutrality.simps
    by metis
  have img': φ-neutral ?domain π (A, V, p) = (A', V', p')
    using img unanimous
    by simp
  hence elect (rule-𝒦 strong-unanimity) V' A' p' =
        (elect-r ∘ fun_𝓔 (rule-𝒦 strong-unanimity))
          (φ-neutral ?domain π (A, V, p))
    by simp
  also have
    ... = set-action ψ-neutral_c π
          ((elect-r ∘ fun_𝓔 (rule-𝒦 strong-unanimity)) (A, V, p))
    using bij-carrier-π img' unanimous' rewrite
    by metis
  also have ... = set-action ψ-neutral_c π {a}
    using cons
    unfolding 𝒦_𝓔.simps
    by simp
  finally have (A', V', p') ∈ elections-𝒦 strong-unanimity
    unfolding 𝒦_𝓔.simps strong-unanimity-def consensus-choice.simps
    using unanimous' prof'
    by simp
  hence ((A, V, p), (A', V', p'))
        ∈ ⋃ (range (𝒦_𝓔 strong-unanimity)) × ⋃ (range (𝒦_𝓔 strong-unanimity))
    unfolding elections-𝒦.simps
    using cons
    by blast
  moreover have
    ∃ π ∈ carrier bijection_𝒜𝒢.
      φ-neutral well-formed-elections π (A, V, p) = (A', V', p')
    using img bij-carrier-π
    unfolding bijection_𝒜𝒢-def
    by blast
  ultimately show (A', V', p') ∈ ⋃ (range (𝒦_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 $\mathit{score} :: ('a,\ 'v)\ \mathit{Election\ Distance} \Rightarrow ('a,\ 'v,\ 'r\ \mathit{Result})\ \mathit{Consensus\text{-}Class} \Rightarrow$
$\quad ('a,\ 'v)\ \mathit{Election} \Rightarrow 'r \Rightarrow \mathit{ereal}$ **where**
$\quad \mathit{score}\ d\ K\ E\ w = \mathit{Inf}\ (d\ E\ `\ (\mathcal{K}_{\mathcal{E}}\ K\ w))$

fun (**in** result) $\mathcal{R}_{\mathcal{W}} :: ('a,\ 'v)\ \mathit{Election\ Distance} \Rightarrow$
$\quad ('a,\ 'v,\ 'r\ \mathit{Result})\ \mathit{Consensus\text{-}Class} \Rightarrow 'v\ \mathit{set} \Rightarrow 'a\ \mathit{set} \Rightarrow ('a,\ 'v)\ \mathit{Profile} \Rightarrow$
$\quad 'r\ \mathit{set}$ **where**
$\quad \mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p = \mathit{arg\text{-}min\text{-}set}\ (\mathit{score}\ d\ K\ (A,\ V,\ p))\ (\mathit{limit}\ A\ \mathit{UNIV})$

fun (**in** result) $\mathit{distance\text{-}}\mathcal{R} :: ('a,\ 'v)\ \mathit{Election\ Distance} \Rightarrow$
$\quad ('a,\ 'v,\ 'r\ \mathit{Result})\ \mathit{Consensus\text{-}Class} \Rightarrow$
$\quad ('a,\ 'v,\ 'r\ \mathit{Result})\ \mathit{Electoral\text{-}Module}$ **where**
$\quad \mathit{distance\text{-}}\mathcal{R}\ d\ K\ V\ A\ p =$
$\quad\quad (\mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p,\ (\mathit{limit}\ A\ \mathit{UNIV}) - \mathcal{R}_{\mathcal{W}}\ d\ K\ V\ A\ p,\ \{\})$

5.4.2 Standard Definitions

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

definition $\mathit{voters\text{-}determine\text{-}distance} :: ('a,\ 'v)\ \mathit{Election\ Distance} \Rightarrow \mathit{bool}$ **where**
$\quad \mathit{voters\text{-}determine\text{-}distance}\ d \equiv$
$\quad\quad \forall\ A\ A'\ V\ V'\ p\ q\ p'.$
$\quad\quad\quad (\forall\ v \in V.\ p\ v = q\ v)$
$\quad\quad\quad\quad \longrightarrow (d\ (A,\ V,\ p)\ (A',\ V',\ p') = d\ (A,\ V,\ q)\ (A',\ V',\ p')$
$\quad\quad\quad\quad\quad \land (d\ (A',\ V',\ p')\ (A,\ V,\ p) = d\ (A',\ V',\ p')\ (A,\ V,\ q)))$

Creates a set of all possible profiles on a finite alternative set that are empty everywhere outside of a given finite voter set.

fun $\mathit{profiles} :: 'v\ \mathit{set} \Rightarrow 'a\ \mathit{set} \Rightarrow (('a,\ 'v)\ \mathit{Profile})\ \mathit{set}$ **where**
$\quad \mathit{profiles}\ V\ A =$
$\quad\quad (\mathit{if\ infinite}\ A \lor \mathit{infinite}\ V$
$\quad\quad\quad \mathit{then}\ \{\}\ \mathit{else}\ \{p.\ p\ `\ V \subseteq \mathit{pl\text{-}}\alpha\ `\ \mathit{permutations\text{-}of\text{-}set}\ A\})$

fun $\mathcal{K}_{\mathcal{E}}\mathit{\text{-}std} :: ('a,\ 'v,\ 'r\ \mathit{Result})\ \mathit{Consensus\text{-}Class} \Rightarrow 'r \Rightarrow 'a\ \mathit{set} \Rightarrow 'v\ \mathit{set} \Rightarrow$

$('a,\ 'v)$ *Election set* **where**
$\mathcal{K}_\mathcal{E}$*-std K w A V* =
$(\lambda\ p.\ (A,\ V,\ p))$ ‘ *Set.filter*
$(\lambda\ p.$ *consensus-*$\mathcal{K}$ *K* $(A,\ V,\ p) \wedge$ *elect* (*rule-*$\mathcal{K}$ *K*) *V A p* $= \{w\})$
(*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 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 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 *listset-finiteness*:
fixes *l* :: $'a$ *set list*

assumes $\forall\ i :: nat.\ i < length\ l \longrightarrow finite\ (l!i)$
shows *finite* (*listset* l)
using *assms*
proof (*induct* l)
case *Nil*
show *finite* (*listset* [])
by *simp*
next
case (*Cons* a l)
fix
$a :: 'a\ set$ **and**
$l :: 'a\ set\ list$
assume $\forall\ i :: nat < length\ (a\#l).\ finite\ ((a\#l)!i)$
hence
finite a **and**
$\forall\ i < length\ l.\ finite\ (l!i)$
by *auto*
moreover assume
$\forall\ i :: nat < length\ l.\ finite\ (l!i) \Longrightarrow finite\ (listset\ l)$
ultimately have $finite\ \{a'\#l' \mid a'\ l'.\ a' \in a \wedge l' \in (listset\ l)\}$
using *list-cons-presv-finiteness*
by *blast*
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**
$\forall\ i :: nat.\ i < length\ l \longrightarrow l!i = \{\}$
shows $listset\ l = \{\}$
using *assms*
proof (*induct* l)
case *Nil*
thus $listset\ [] = \{\}$
by *simp*
next
case (*Cons* a l)
fix
$a :: 'a\ set$ **and**
$l :: 'a\ set\ list$ **and**
$l' :: 'a\ list$
assume *all-elems-empty*: $\forall\ i :: nat < length\ (a\#l).\ (a\#l)!i = \{\}$
hence $a = \{\}$
by *auto*
moreover from *all-elems-empty*
have $\forall\ i < length\ l.\ l!i = \{\}$
by *auto*

ultimately have $\{a'\#l' \mid a'\ l'.\ a' \in a \wedge l' \in (listset\ l)\} = \{\}$
  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, *safe*)
  case *Nil*
  fix l :: $'a$ *list*
  assume $l \in listset\ []$
  thus $length\ l = length\ []$
    by *simp*
next
  case (*Cons a l*)
  fix
    a :: $'a$ *set* **and**
    l :: $'a$ *set list* **and**
    l' :: $'a$ *list*
  assume
    $\forall\ l'.\ l' \in listset\ l \longrightarrow length\ l' = length\ l$ **and**
    $l' \in listset\ (a\#l)$
  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 $length\ l' = length\ (a\#l)$
    using *local.Cons*
    by *fastforce*
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* $(profiles\ V\ A \cap \{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = x\})$
proof (*cases* $A = \{\}$)
  let $?profs = profiles\ V\ A \cap \{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = x\}$
  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 ∀ p ∈ profiles V A. ∀ v. v ∈ V ⟶ p v = {}
    by (simp add: image-subset-iff)
  hence ∀ p ∈ ?profs. (∀ v. v ∈ V ⟶ p v = {}) ∧ (∀ v. v ∉ V ⟶ p v = x)
    by simp
  hence ∀ p ∈ ?profs. p = (λ v. if v ∈ V then {} else x)
    by (metis (no-types, lifting))
  hence ?profs ⊆ {λ v. if v ∈ V then {} else x}
    by blast
  thus finite ?profs
    using finite.emptyI finite-insert finite-subset
    by (metis (no-types, lifting))
next
  let ?profs = profiles V A ∩ {p. ∀ v. v ∉ V ⟶ p v = x}
  case False
  from fin-V obtain ord :: 'v rel where
    linear-order-on V ord
    using finite-list lin-ord-equiv lin-order-equiv-list-of-alts
    by metis
  then obtain list-V :: 'v list where
    len: length list-V = card V and
    pl: ord = pl-α list-V and
    perm: list-V ∈ 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 ∀ p ∈ profiles V A. ∀ v ∈ V. p v ∈ (pl-α ` permutations-of-set A)
    by (simp add: image-subset-iff)
  hence ∀ p ∈ profiles V A. (∀ v ∈ V. linear-order-on A (p v))
    using pl-α-lin-order fin-A False
    by metis
  moreover have ∀ p ∈ ?profs. ∀ i < length (?map p). (?map p)!i = p (list-V!i)
    by simp
  moreover have ∀ i < length list-V. list-V!i ∈ V
    using perm nth-mem
    unfolding permutations-of-set-def
    by safe
  moreover have lens-eq: ∀ p ∈ ?profs. length (?map p) = length list-V
    by simp
  ultimately have
    ∀ p ∈ ?profs. ∀ i < length (?map p). linear-order-on A ((?map p)!i)
    by simp
  hence subset-map-profs: ?map ` ?profs ⊆ {xs. length xs = card V ∧
                              (∀ i < length xs. linear-order-on A (xs!i))}
    using len lens-eq
    by fastforce
  have ∀ p1 p2.
     p1 ∈ ?profs ∧ p2 ∈ ?profs ∧ p1 ≠ p2 ⟶ (∃ v ∈ V. p1 v ≠ 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 *fin-supset*:
$\forall$ *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 *fin-supset 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-map-profs*
by *blast*
qed

lemma *profile-permutation-set*:
fixes
A :: *'a set* **and**
V :: *'v set*
shows *profiles V A* = {*p* :: (*'a*, *'v*) *Profile*. *finite-profile V A p*}

proof (*cases finite* $A \wedge$ *finite* $V \wedge A \neq \{\}$)
  case *True*
  assume *finite* $A \wedge$ *finite* $V \wedge A \neq \{\}$
  hence
    fin-A: *finite* A **and**
    fin-V: *finite* V **and**
    non-empty: $A \neq \{\}$
    by *safe*
  show *profiles* V A = $\{p'.$ *finite-profile* V A $p'\}$
  proof (*standard*, *clarify*)
    fix p :: $'v \Rightarrow$ $'a$ *Preference-Relation*
    assume $p \in$ *profiles* V A
    hence $\forall$ $v \in V$. p $v \in$ *pl-*α ‘ *permutations-of-set* A
      using *fin-A fin-V*
      by *auto*
    hence $\forall$ $v \in V$. *linear-order-on* A (p v)
      using *fin-A pl-*α*-lin-order non-empty*
      by *metis*
    thus *finite-profile* V A p
      unfolding *profile-def*
      using *fin-A fin-V*
      by *blast*
  next
    show $\{p.$ *finite-profile* V A $p\} \subseteq$ *profiles* V A
    proof (*standard*, *clarify*)
      fix p :: ($'a$, $'v$) *Profile*
      assume *prof*: *profile* V A p
      have $p \in \{p.$ p ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\}$
        using *fin-A lin-order-pl-*α *prof*
        unfolding *profile-def*
        by *blast*
      thus $p \in$ *profiles* V A
        using *fin-A fin-V*
        unfolding *profiles.simps*
        by *metis*
    qed
  qed
next
  case *False*
  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* $V \wedge A = \{\} \longrightarrow$
    ($\forall$ $\pi \in \{\pi.$ π ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\}$. $\forall$ $v \in V$. π 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* $V \wedge A = \{\}$
$\longrightarrow$ *profiles* $V\ A = \{\pi.\ \pi$ ‘ $V \subseteq$ (*pl-*α ‘ *permutations-of-set* A)$\}$
by *simp*
ultimately have *all-prof-eq*: *finite* $A \wedge$ *finite* $V \wedge A = \{\}$
$\longrightarrow$ *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 *non-voters*:
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 $(\forall\ p.$ *profile* $V\ \{\}\ p \wedge (\forall\ v.\ v \notin V \longrightarrow p\ v = \{\})$
$\longrightarrow (\forall\ v.\ p\ v = \{\})) \longrightarrow$ *finite* $V \longrightarrow A = \{\}$
$\longrightarrow \{p.$ *profile* $V\ \{\}\ p\} = \{p.\ \forall\ v \in V.\ p\ v = \{\}\}$
unfolding *profile-def*
using *lin-ord-not-empty*
by *auto*
hence *finite* $A \wedge$ *finite* $V \wedge A = \{\}$
$\longrightarrow \{p.$ *finite-profile* $V\ A\ p\} = \{p.\ \forall\ v \in V.\ p\ v = \{\}\}$
using *non-voters*
by *blast*
hence *finite* $A \wedge$ *finite* $V \wedge A = \{\}$
$\longrightarrow$ *profiles* $V\ A = \{p.$ *finite-profile* $V\ A\ p\}$
using *all-prof-eq*
by *simp*
moreover have *infinite* $A \vee$ *infinite* $V \longrightarrow$ *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* $A \vee$ *infinite* $V \vee A = \{\}$
using *not-fin-empty*
by *simp*
ultimately show *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.simps*, *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 A UNIV*)
    using $\mathcal{R}_{\mathcal{W}}$*.simps arg-min-subset*
    by *metis*
  hence *set-equals-partition* (*limit 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 Properties

fun *distance-decisiveness* :: ('*a*, '*v*) *Election set* $\Rightarrow$ ('*a*, '*v*) *Election Distance* $\Rightarrow$
    ('*a*, '*v*, '*r Result*) *Electoral-Module* $\Rightarrow$ *bool* **where**
  distance-decisiveness X d m =
    ($\nexists$ E. $E \in X$
    $\wedge$ ($\exists$ $\delta > 0$. $\forall$ $E' \in X$. $d\ E\ E' < \delta \longrightarrow$ *card* (*elect-r* (*fun*$_{\mathcal{E}}$ *m* E')) > *1*))

5.4.6 Inference Rules

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: *voters-determine-distance 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*:
    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* $K\ w\ A\ V =$
$\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ *Pair* A ‘ *Pair* V ‘ $\{p' :: ('a, 'v)$ *Profile. finite-profile* $V\ A\ p'\}$
by *force*
have $\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ *Pair* A ‘ *Pair* V ‘ $\{p' :: ('a, 'v)$ *Profile. finite-profile* $V\ A\ p'\}$
$\subseteq \mathcal{K}_{\mathcal{E}}\ K\ w$
by *simp*
hence *Inf* $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_{\mathcal{E}}\ K\ w)) \leq$
Inf $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_{\mathcal{E}}\ K\ w \cap$
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* $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_{\mathcal{E}}\ K\ w)) \geq$
Inf $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_{\mathcal{E}}\ K\ w \cap$
Pair A ‘ *Pair* V ‘ $\{p' :: ('a, 'v)$ *Profile. finite-profile* $V\ A\ p'\}))$
proof (*rule INF-greatest*)
let *?inf* = *Inf* $(d\ (A,\ V,\ p)$ ‘
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ *Pair* A ‘ *Pair* V ‘ $\{p'.$ *finite-profile* $V\ A\ p'\}))$
let *?compl* = $\mathcal{K}_{\mathcal{E}}\ K\ w\ -$
$\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ *Pair* A ‘ *Pair* V ‘ $\{p'.$ *finite-profile* $V\ A\ p'\}$
fix $i :: ('a, 'v)$ *Election*
assume *el*: $i \in \mathcal{K}_{\mathcal{E}}\ K\ w$
have *in-intersect*:
$i \in (\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ *Pair* A ‘ *Pair* V ‘ $\{p'.$ *finite-profile* $V\ A\ p'\})$
$\longrightarrow$ *?inf* $\leq d\ (A,\ V,\ p)\ i$
using *Complete-Lattices.complete-lattice-class.INF-lower*
by *metis*
have *compl-imp-neither-voter-nor-alt-nor-infinite-prof*:
$i \in$ *?compl* $\longrightarrow (V \neq$ *fst* $($*snd* $i)$
$\vee\ A \neq$ *fst* i
$\vee\ \neg$ *finite-profile* $V\ A\ ($*snd* $($*snd* $i)))$
by *fastforce*
moreover have *not-voters-imp-infty*: $V \neq$ *fst* $($*snd* $i) \longrightarrow d\ (A,\ V,\ p)\ i = \infty$
using *std prod.collapse*
unfolding *standard-def*
by *metis*
moreover have *not-alts-imp-infty*: $A \neq$ *fst* $i \longrightarrow d\ (A,\ V,\ p)\ i = \infty$
using *std prod.collapse*
unfolding *standard-def*
by *metis*
moreover have $V =$ *fst* $($*snd* $i) \wedge A =$ *fst* i
$\wedge\ \neg$ *finite-profile* $V\ A\ ($*snd* $($*snd* $i)) \longrightarrow$ *False*
using *el*
by *fastforce*
hence $i \in$ *?compl* $\longrightarrow d\ (A,\ V,\ p)\ i = \infty$
using *not-alts-imp-infty not-voters-imp-infty*
compl-imp-neither-voter-nor-alt-nor-infinite-prof

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-profile* $V\ A\ p'\}))$
$\leq d\ (A,\ V,\ p)\ i$
using *ereal-less-eq*
by (*metis* (*no-types*, *lifting*))
thus *Inf* $(d\ (A,\ V,\ p)$ `
$(\mathcal{K}_{\mathcal{E}}\ K\ w \cap$
Pair A ` *Pair* $V$ ` $\{p'.$ *finite-profile* $V\ A\ p'\}))$
$\leq d\ (A,\ V,\ p)\ i$
using *in-intersect el*
by *blast*
qed
ultimately have *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-profile* $V\ A\ p'\}))$
using *order-antisym*
by *simp*
also have *inf-eq-min-for-std-cons*: $\ldots =$ *score-std* $d\ K\ (A,\ V,\ p)\ w$
proof (*cases* $\mathcal{K}_{\mathcal{E}}$-*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-profile* $V\ A\ p'\})) = \infty$
using *eq-intersect top-ereal-def*
by *simp*
also have *score-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 $\mathcal{K}_{\mathcal{E}}$-*std* $K\ w\ A\ V =$
$(\mathcal{K}_{\mathcal{E}}\ K\ w) \cap \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$
using *eq-intersect*
by *blast*
hence *subset-dist-* $\mathcal{K}_{\mathcal{E}}$ *-std*:
$d\ (A,\ V,\ p)$ `$(\mathcal{K}_{\mathcal{E}}$-*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* $V\ A\ p' \longrightarrow$

$\quad\quad$ *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* $V\ A\ p'\}$.
$\quad (A',\ V',$ *?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 *voters-determine-distance-def*
by *simp*
ultimately have
$\forall\ (A',\ V',\ p') \in \{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$.
$\quad (\exists\ (X,\ Y,\ z) \in \{(A,\ V,\ p') \mid p'.$
$\quad\quad$ *finite-profile* $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$.
$\quad\quad d\ (A,\ V,\ p)\ (A',\ V',\ p') = d\ (A,\ V,\ p)\ (X,\ Y,\ z))$
by *fastforce*
hence
$\forall\ (A',\ V',\ p')$
$\quad \in \{(A',\ V',\ p').\ A' = A \wedge V' = V \wedge$ *finite-profile* $V\ A\ p'\}$.
$\quad\quad d\ (A,\ V,\ p)\ (A',\ V',\ p') \in$
$\quad\quad\quad d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$
$\quad\quad\quad\quad$ *finite-profile* $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
by *fastforce*
hence *subset-dist-restrict-non-voters*:
$d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\}$
$\quad \subseteq d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$
$\quad\quad$ *finite-profile* $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
by *fastforce*
have $\forall\ (A',\ V',\ p') \in \{(A,\ V,\ p') \mid p'.$
$\quad$ *finite-profile* $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$.
$\quad\quad (\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v))$
$\quad\quad\quad \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})$
using *fin*
unfolding *profile-def*
by *simp*
hence *subset-lin-ord*:
$\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
$\quad \subseteq \{(A,\ V,\ p') \mid p'.\ p' \in \{p'.$
$\quad\quad (\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v)) \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}\}$
by *blast*
have $\{p'.\ (\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v))$
$\quad\quad \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
$\quad \subseteq$ *profiles* $V\ A \cap \{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = \{\}\}$
using *lin-order-pl-*α *fin*
by *fastforce*

moreover have *finite* (*profiles* V A $\cap$ $\{p.\ \forall\ v.\ v \notin V \longrightarrow p\ v = \{\}\}$)
  using *fin fin-all-profs*
  by *blast*
ultimately have
  finite $\{p'.\ (\forall\ v \in V.$
    linear-order-on $A\ (p'\ v)) \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
  using *rev-finite-subset*
  by *blast*
hence *finite* $\{(A,\ V,\ p') \mid p'.\ p' \in \{p'.$
    $(\forall\ v \in V.$ *linear-order-on* $A\ (p'\ v)) \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}\}$
  by *simp*
hence *finite* $\{(A,\ V,\ p') \mid p'.$
    finite-profile $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\}$
  using *subset-lin-ord rev-finite-subset*
  by *simp*
hence *finite* $(d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$
    finite-profile $V\ A\ p' \wedge (\forall\ v.\ v \notin V \longrightarrow p'\ v = \{\})\})$
  by *simp*
hence *finite* $(d\ (A,\ V,\ p)$ ‘ $\{(A,\ V,\ p') \mid p'.$ *finite-profile* $V\ A\ p'\})$
  using *subset-dist-restrict-non-voters rev-finite-subset*
  by *simp*
hence *finite* $(d\ (A,\ V,\ p)$ ‘$(\mathcal{K}_\mathcal{E}$*-std* $K\ w\ A\ V))$
  using *subset-dist-*$\mathcal{K}_\mathcal{E}$*-std rev-finite-subset*
  by *blast*
moreover have $d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_\mathcal{E}$*-std* $K\ w\ A\ V) \neq \{\}$
  using *False*
  by *simp*
ultimately have
  Inf $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_\mathcal{E}$*-std* $K\ w\ A\ V)) =$
    Min $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_\mathcal{E}$*-std* $K\ w\ A\ V))$
  using *Min-Inf False*
  by *metis*
also have $\ldots =$ *score-std* $d\ K\ (A,\ V,\ p)\ w$
  using *False*
  by *simp*
also have *Inf* $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_\mathcal{E}$*-std* $K\ w\ A\ V)) =$
  Inf $(d\ (A,\ V,\ p)$ ‘ $(\mathcal{K}_\mathcal{E}\ K\ w \cap$
    Pair A ‘ *Pair* V ‘ $\{p'.$ *finite-profile* $V\ A\ p'\}))$
  using *eq-intersect*
  by *simp*
ultimately show *?thesis*
  by *simp*
qed
finally show *score* $d\ K\ (A,\ V,\ p)\ w =$ *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' :: 'a *set* **and**
V V' :: 'v *set* **and**
p q :: ('a, 'v) *Profile* **and**
π :: 'v $\Rightarrow$ 'v
assume
bijective: *bij* π **and**
renamed: *rename* π (A, V, p) = (A', V', q)
hence *eq-univ*: *limit* A *UNIV* = *limit* A' *UNIV*
by *simp*
have *dist-rename-inv*:
$\forall$ E :: ('a, 'v) *Election*. d (A, V, p) E = d (A', V', q) (*rename* π E)
using *d-anon bijective 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 X' :: 'a *set* **and**
Y Y' :: 'v *set* **and**
z z' :: ('a, 'v) *Profile*
assume (X', Y', z') = *rename* π (X, Y, z)
hence d (A', V', q) (X', Y', z') = d (A, V, p) (X, Y, z)
using *dist-rename-inv*
by *metis*
moreover assume (X, Y, z) $\in$ S
ultimately show d (A', V', q) (X', Y', z') $\in$ d (A, V, p) ‘ S
by *simp*
qed
ultimately have *eq-range*:
$\forall$ S :: ('a, 'v) *Election set*.
(d (A, V, p) ‘ S) = (d (A', V', q) ‘ (*rename* π ‘ S))

```
    by blast
  have ∀ w. rename π ‘ (𝒦_ℰ K w) ⊆ (𝒦_ℰ K w)
  proof (clarify)
    fix
      w :: 'r and
      A A' :: 'a set and
      V V' :: 'v set and
      p p' :: ('a, 'v) Profile
    assume (A, V, p) ∈ 𝒦_ℰ K w
    hence cons:
      (consensus-𝒦 K) (A, V, p) ∧ finite-profile V A p
        ∧ elect (rule-𝒦 K) V A p = {w}
      by simp
    moreover assume renamed: (A', V', p') = rename π (A, V, p)
    ultimately have finite-profile V' A' p'
      using bijective fst-conv rename-finite rename-prof
      unfolding rename.simps
      by metis
    moreover from this have cons-img:
      consensus-𝒦 K (A', V', p') ∧ (rule-𝒦 K V A p = rule-𝒦 K V' A' p')
      using K-anon renamed bijective cons
      unfolding consensus-rule-anonymity-def Let-def
      by simp
    ultimately show (A', V', p') ∈ 𝒦_ℰ K w
      using cons
      by simp
  qed
  moreover have ∀ w. (𝒦_ℰ K w) ⊆ rename π ‘ (𝒦_ℰ K w)
  proof (clarify)
    fix
      w :: 'r and
      A :: 'a set and
      V :: 'v set and
      p :: ('a, 'v) Profile
    assume (A, V, p) ∈ 𝒦_ℰ K w
    hence cons:
      (consensus-𝒦 K) (A, V, p) ∧ finite-profile V A p
           ∧ elect (rule-𝒦 K) V A p = {w}
      by simp
    let ?inv = rename (the-inv π) (A, V, p)
    have inv-inv-id: the-inv (the-inv π) = π
      using the-inv-f-f bijective 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-inv π) ‘ V), p ∘ (the-inv (the-inv π)))
      by simp
    moreover have p ∘ the-inv (the-inv π) ∘ the-inv π = p
      using bijective inv-inv-id
      unfolding bij-betw-def comp-def
```

by (*simp add*: *f-the-inv-into-f*)
moreover have π ‘ (*the-inv* π) ‘ $V = V$
using *bijective the-inv-f-f image-inv-into-cancel top-greatest surj-imp-inv-eq*
unfolding *bij-betw-def*
by (*metis* (*no-types*, *opaque-lifting*))
ultimately have *preimg*: *rename* π *?inv* = (A, V, p)
unfolding *Let-def*
by *simp*
have *bij* (*the-inv* π)
using *bijective bij-betw-the-inv-into*
by *metis*
moreover from *this* **have** *fin-preimg*:
finite-profile (*fst* (*snd ?inv*)) (*fst ?inv*) (*snd* (*snd ?inv*))
using *rename-prof cons*
by *fastforce*
ultimately have
consensus-$\mathcal{K}$ K *?inv* $\wedge$
(*rule-*$\mathcal{K}$ K V A p =
rule-$\mathcal{K}$ K (*fst* (*snd ?inv*)) (*fst ?inv*) (*snd* (*snd ?inv*)))
using *K-anon renamed bijective cons*
unfolding *consensus-rule-anonymity-def Let-def*
by *simp*
moreover from *this* **have**
elect (*rule-*$\mathcal{K}$ K) (*fst* (*snd ?inv*)) (*fst ?inv*) (*snd* (*snd ?inv*)) = $\{w\}$
using *cons*
by *simp*
ultimately have *?inv* $\in \mathcal{K}_{\mathcal{E}}$ K w
using *fin-preimg*
by *simp*
thus (A, V, p) $\in$ *rename* π ‘ $\mathcal{K}_{\mathcal{E}}$ K w
using *preimg image-eqI*
by *metis*
qed
ultimately have $\forall$ w. ($\mathcal{K}_{\mathcal{E}}$ K w) = *rename* π ‘ ($\mathcal{K}_{\mathcal{E}}$ K w)
by *blast*
hence $\forall$ 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* A *UNIV*) =
arg-min-set (*score* d K (A', V', q)) (*limit* A' *UNIV*)
using *eq-univ*
by *presburger*
hence $\mathcal{R}_{\mathcal{W}}$ d K V A p = $\mathcal{R}_{\mathcal{W}}$ d K V' A' q
by *simp*
thus *distance-*$\mathcal{R}$ d K V A p = *distance-*$\mathcal{R}$ 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-*$\mathcal{R}$ :: ($'a$, $'v$ :: *linorder*, $'a$ *Result*) *Consensus-Class* $\Rightarrow$
    ($'a$, $'v$, $'a$ *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 *voters-determine-distance* (*votewise-distance d N*)
proof (*unfold voters-determine-distance-def*, *clarify*)
  fix
    A A' :: $'a$ *set* **and**
    V V' :: $'v$ :: *linorder set* **and**
    p p' 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' :: $'a$ *set* **and**
V V' :: $'v$ :: *linorder set* **and**
p 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* $V \longrightarrow$
($\forall$ *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 *sorted-list-of-set.sorted-key-list-of-set-eq-Nil-iff*
to-list.simps Nil-is-map-conv
by (*metis* (*no-types*))
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 finite-lessThan lessThan-iff abs-ereal.simps*
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 *force*
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
$(A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \textit{finite } V)$
$\vee \textit{ infinite } V \vee (A \neq A' \wedge V = \{\}) \vee V \neq V'$
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* $\Rightarrow$
$('a, 'v, 'a$ *Result*$)$ *Consensus-Class* $\Rightarrow$ $('a, 'v)$ *Election* $\Rightarrow$ $'a \Rightarrow$ *ereal*

type-synonym $('a, 'v)$ *dist-rat-type* = $('a, 'v)$ *Election Distance* $\Rightarrow$
$('a, 'v, 'a$ *Result*$)$ *Consensus-Class* $\Rightarrow$ $'v$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $('a, 'v)$ *Profile* $\Rightarrow$ $'a$ *set*

type-synonym $('a, 'v)$ *dist-rat-std-type* = $('a, 'v)$ *Election Distance* $\Rightarrow$
$('a, 'v, 'a$ *Result*$)$ *Consensus-Class* $\Rightarrow$ $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*

type-synonym $('a, 'v)$ *dist-type* = $('a, 'v)$ *Election Distance* $\Rightarrow$
$('a, 'v, 'a$ *Result*$)$ *Consensus-Class* $\Rightarrow$ $('a, 'v, 'a$ *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*
$\mathcal{SCF}$*-result.standard-distance-imp-equal-score*
by *fast*

lemma *swap-*$\mathcal{R}$*-code*[*code*]: *swap-*$\mathcal{R}$ =
($\mathcal{SCF}$*-result.distance-*$\mathcal{R}$*-std* :: $('a, 'v ::$ *linorder*$)$ *dist-rat-std-type*)
(*votewise-distance swap l-one*)
unfolding *swap-*$\mathcal{R}$*.simps* $\mathcal{SCF}$*-result.distance-*$\mathcal{R}$*.simps* $\mathcal{SCF}$*-result.distance-*$\mathcal{R}$*-std.simps*
$\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*.simps* $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*-std.simps equal-score-swap*
by *safe*

end

5.6 Symmetry in Distance-Rationalizable Rules

theory *Distance-Rationalization-Symmetry*
imports *Distance-Rationalization*
begin

5.6.1 Minimizer Function

fun *distance-infimum* :: $'a$ *Distance* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a \Rightarrow$ *ereal* **where**

distance-infimum d A a = *Inf* (*d a* ‘ *A*)

fun *closest-preimg-distance* :: ($'a \Rightarrow 'b$) $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$
$'a \Rightarrow 'b \Rightarrow$ *ereal* **where**
closest-preimg-distance f $domain_f$ *d a b* = *distance-infimum d* (*preimg f* $domain_f$ *b*) *a*

fun *minimizer* :: ($'a \Rightarrow 'b$) $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Distance* $\Rightarrow$ $'b$ *set* $\Rightarrow$ $'a \Rightarrow 'b$ *set* **where**
minimizer f $domain_f$ *d A a* = *arg-min-set* (*closest-preimg-distance f* $domain_f$ *d a*) *A*

Auxiliary Lemmas

lemma *rewrite-arg-min-set*:
fixes
f :: $'a \Rightarrow 'b$ :: *linorder* **and**
A :: $'a$ *set*
shows *arg-min-set f A* = $\bigcup$ (*preimg f A* ‘ $\{y \in f$ ‘ $A.\ \forall\ z \in f$ ‘ $A.\ y \leq z\}$)
proof (*safe*)
fix *x* :: $'a$
assume $x \in$ *arg-min-set f A*
thus $x \in \bigcup$ (*preimg f A* ‘ $\{y \in f$ ‘ $A.\ \forall\ z \in f$ ‘ $A.\ y \leq z\}$)
by (*simp add*: *is-arg-min-linorder*)
next
fix *x y* :: $'a$
assume
$x \in$ *preimg f A* (*f y*) **and**
$\forall\ z \in f$ ‘ $A.\ f\ y \leq z$
thus $x \in$ *arg-min-set f A*
by (*simp add*: *is-arg-min-linorder*)
qed

Equivariance

abbreviation *Restrp* :: $'a$ *rel* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *rel* **where**
Restrp r A $\equiv$ *r Int* ($A \times$ *UNIV*)

lemma *restr-induced-rel*:
fixes
A :: $'a$ *set* **and**
B B' :: $'b$ *set* **and**
φ :: ($'a$, $'b$) *binary-fun*
assumes $B' \subseteq B$
shows *Restrp* (*action-induced-rel A B* φ) *B'* = *action-induced-rel A B'* φ
using *assms*
by *force*

theorem *group-action-invar-dist-and-equivar-f-imp-equivar-minimizer*:
fixes
f :: $'a \Rightarrow 'b$ **and**

$domain_f$ X :: $'a$ *set* **and**
d :: $'a$ *Distance* **and**
well-formed-img :: $'a \Rightarrow 'b$ *set* **and**
G :: $'c$ *monoid* **and**
φ :: $('c, 'a)$ *binary-fun* **and**
ψ :: $('c, 'b)$ *binary-fun*
defines *equivar-prop-set-valued* $\equiv$
action-induced-equivariance (*carrier* G) X φ (*set-action* ψ)
assumes
action-φ: *group-action* G X φ **and**
group-action-res: *group-action* G *UNIV* ψ **and**
dom-in-X: $domain_f \subseteq X$ **and**
closed-domain:
closed-restricted-rel (*action-induced-rel* (*carrier* G) X φ) X $domain_f$ **and**
equivar-img: *is-symmetry well-formed-img equivar-prop-set-valued* **and**
invar-d: $invariance_{\mathcal{D}}$ d (*carrier* G) X φ **and**
equivar-f:
is-symmetry f (*action-induced-equivariance* (*carrier* G) $domain_f$ φ ψ)
shows *is-symmetry* (λ *x. minimizer f* $domain_f$ d (*well-formed-img x*) x) *equivar-prop-set-valued*
proof (*unfold action-induced-equivariance-def equivar-prop-set-valued-def is-symmetry.simps set-action.simps minimizer.simps*, *clarify*)
fix
x :: $'a$ **and**
g :: $'c$
assume
group-elem: $g \in$ *carrier* G **and**
x-in-X: $x \in X$
hence *img-X*: φ g $x \in X$
using *action-φ group-action.element-image*
by *metis*
let $?x' = \varphi$ g x
let $?c$ = *closest-preimg-distance f* $domain_f$ d x **and**
$?c'$ = *closest-preimg-distance f* $domain_f$ d $?x'$
have $\forall$ *y. preimg f* $domain_f$ $y \subseteq X$
using *dom-in-X*
by *fastforce*
hence $\forall$ *y*. d x ‘ (*preimg f* $domain_f$ y) = d $?x'$ ‘ (φ g ‘ (*preimg f* $domain_f$ y))
using *x-in-X group-elem invar-dist-image invar-d action-φ*
by *metis*
hence $\forall$ *y*. *Inf* (d $?x'$ ‘ *preimg f* $domain_f$ (ψ g y)) =
Inf (d x ‘ *preimg f* $domain_f$ y)
using *assms group-action-equivar-f-imp-equivar-preimg*[*of* G] *group-elem*
by *metis*
hence *comp*:
closest-preimg-distance f $domain_f$ d x =
(*closest-preimg-distance f* $domain_f$ d $?x'$) $\circ$ (ψ g)
by *auto*
hence $\forall$ *Y A*. {*preimg* $?c'$ (ψ g ‘ Y) α | α. $\alpha \in A$} =
{ψ g ‘ *preimg* $?c$ Y α | α. $\alpha \in A$}

using *preimg-comp*
by *auto*
moreover have
$\forall\ Y\ A.\ \{\psi\ g\ \text{‘}\ preimg\ ?c\ Y\ \alpha \mid \alpha.\ \alpha \in A\} = \{\psi\ g\ \text{‘}\ \beta \mid \beta.\ \beta \in preimg\ ?c\ Y\ \text{‘}\ A\}$
by *blast*
moreover have
$\forall\ Y\ A.\ preimg\ ?c'\ (\psi\ g\ \text{‘}\ Y)\ \text{‘}\ A = \{preimg\ ?c'\ (\psi\ g\ \text{‘}\ Y)\ \alpha \mid \alpha.\ \alpha \in A\}$
by *blast*
ultimately have
$\forall\ Y\ A.\ \bigcup\ (preimg\ ?c'\ (\psi\ g\ \text{‘}\ Y)\ \text{‘}\ A) = \bigcup\ \{\psi\ g\ \text{‘}\ \alpha \mid \alpha.\ \alpha \in preimg\ ?c\ Y\ \text{‘}\ A\}$
by *simp*
moreover have
$\forall\ Y\ A.\ \bigcup\ \{\psi\ g\ \text{‘}\ \alpha \mid \alpha.\ \alpha \in preimg\ ?c\ Y\ \text{‘}\ A\} = \psi\ g\ \text{‘} \bigcup\ (preimg\ ?c\ Y\ \text{‘}\ A)$
by *blast*
ultimately have $\forall\ Y.$ *arg-min-set* (*closest-preimg-distance f domain*$_f$ *d ?x'*) (ψ g ‘ Y) =
(ψ g) ‘ (*arg-min-set* (*closest-preimg-distance f domain*$_f$ *d x*) Y)
using *rewrite-arg-min-set*[*of ?c'*] *rewrite-arg-min-set*[*of ?c*] *comp*
by *auto*
moreover have *well-formed-img* (φ g x) = ψ g ‘ *well-formed-img x*
using *equivar-img x-in-X group-elem img-X rewrite-equivariance*
unfolding *equivar-prop-set-valued-def set-action.simps*
by *metis*
ultimately show
arg-min-set (*closest-preimg-distance f domain*$_f$ *d* (φ g x))
(*well-formed-img* (φ g x)) =
ψ g ‘ *arg-min-set* (*closest-preimg-distance f domain*$_f$ *d x*)
(*well-formed-img x*)
by *presburger*
qed

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist*:
fixes
$f :: 'a \Rightarrow 'b$ **and**
domain$_f$ :: *'a set* **and**
d :: *'a Distance* **and**
rel :: *'a rel*
assumes
reflp-on' *domain*$_f$ (*Restrp rel domain*$_f$) **and**
total-invariance$_{\mathcal{D}}$ *d rel*
shows *is-symmetry* (*closest-preimg-distance f domain*$_f$ *d*) (*Invariance rel*)
proof (*unfold is-symmetry.simps*, *intro allI impI ext*)
fix
$a\ b :: 'a$ **and**
$y :: 'b$
assume $(a, b) \in rel$
hence $\forall\ c \in domain_f.\ d\ a\ c = d\ b\ c$

using *assms*
unfolding *reflp-on′-def reflp-on-def rewrite-total-invariance$_{\mathcal{D}}$*
by *blast*
thus *closest-preimg-distance f domain$_f$ d a y* =
closest-preimg-distance f domain$_f$ d b y
by *simp*
qed

lemma *refl-rel-and-tot-invar-dist-imp-invar-minimizer*:
fixes
f :: *′a* ⇒ *′b* **and**
domain$_f$ :: *′a set* **and**
d :: *′a Distance* **and**
rel :: *′a rel* **and**
img :: *′b set*
assumes
reflp-on′ domain$_f$ (*Restrp rel domain$_f$*) **and**
total-invariance$_{\mathcal{D}}$ d rel
shows *is-symmetry* (*minimizer f domain$_f$ d img*) (*Invariance rel*)
proof −
have *is-symmetry* (*closest-preimg-distance f domain$_f$ d*) (*Invariance rel*)
using *assms closest-dist-invar-under-refl-rel-and-tot-invar-dist*
by *metis*
thus *?thesis*
by *simp*
qed

theorem *group-act-invar-dist-and-invar-f-imp-invar-minimizer*:
fixes
f :: *′a* ⇒ *′b* **and**
domain$_f$ A :: *′a set* **and**
d :: *′a Distance* **and**
img :: *′b set* **and**
G :: *′c monoid* **and**
φ :: (*′c*, *′a*) *binary-fun*
defines
rel ≡ *action-induced-rel* (*carrier G*) *A φ* **and**
rel′ ≡ *action-induced-rel* (*carrier G*) *domain$_f$ φ*
assumes
action-φ: *group-action G A φ* **and**
dom-in-A: *domain$_f$* ⊆ *A* **and**
closed-domain: *closed-restricted-rel rel A domain$_f$* **and**
invar-d: *invariance$_{\mathcal{D}}$ d* (*carrier G*) *A φ* **and**
invar-f: *is-symmetry f* (*Invariance rel′*)
shows *is-symmetry* (*minimizer f domain$_f$ d img*) (*Invariance rel*)
proof −
let
?ψ = *λ g. id* **and**
?img = *λ x. img*

have *is-symmetry f* (*action-induced-equivariance* (*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-action action-*φ
unfolding *group-action-def group-hom-def*
by *blast*
moreover have
is-symmetry ?img (*action-induced-equivariance* (*carrier G*) *A* φ (*set-action ?*ψ))
unfolding *action-induced-equivariance-def*
by *fastforce*
ultimately have
is-symmetry (λ *x. minimizer f* $domain_f$ *d* (*?img x*) *x*)
(*action-induced-equivariance* (*carrier G*) *A* φ (*set-action ?*ψ))
using *group-action-invar-dist-and-equivar-f-imp-equivar-minimizer*[*of*
- - - - - *?img*] *assms*
by *blast*
thus *?thesis*
unfolding *rel-def set-action.simps*
using *rewrite-invar-as-equivar image-id*
by *metis*
qed

5.6.2 Minimizer Translation

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* $\circ$ $fun_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$ = $\mathcal{K}_\mathcal{E}$ *C w*
proof −
have *preimg* (*elect-r* $\circ$ $fun_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$ =
$\{$*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 . . . =
elections-$\mathcal{K}$ *C*
$\cap$ $\{$*E. elect* (*rule-*$\mathcal{K}$ *C*) (*voters-*$\mathcal{E}$ *E*) (*alternatives-*$\mathcal{E}$ *E*) (*profile-*$\mathcal{E}$ *E*) = $\{w\}\}$
by *blast*
finally show *preimg* (*elect-r* $\circ$ $fun_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$ = $\mathcal{K}_\mathcal{E}$ *C w*
by *force*
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-distance (*elect-r* ∘ $fun_{\mathcal{E}}$ (*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*
moreover have $\mathcal{K}_{\mathcal{E}}$ *C w* = *preimg* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$
using $\mathcal{K}_{\mathcal{E}}$*-is-preimg*
by *metis*
moreover have
Inf (*d E* ‘ (*preimg* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$)) =
closest-preimg-distance (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E* $\{w\}$
by *simp*
ultimately 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* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)) *E*))
proof
fix E :: ($'a$, $'v$) *Election*
let *?min* = (*minimizer* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)) *E*)
have *?min* =
arg-min-set
(*closest-preimg-distance* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E*)
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*))
by *simp*
also have
. . . = *singleton-set-system*
(*arg-min-set* (*score d C E*) (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*))
proof (*safe*)
fix R :: $'r$ *set*
assume
min: $R \in$ *arg-min-set*
(*closest-preimg-distance*
(*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E*)
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*))
hence $R \in$ *singleton-set-system* (*limit* (*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*
by *auto*
have $\nexists$ R'. $R' \in$ *singleton-set-system* (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
$\wedge$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E R'*
$<$ *closest-preimg-distance*
(*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*
$\wedge$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E* $\{r'\}$
$<$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E* $\{r\}$
using *res-singleton*
by *auto*
hence $r \in$ *arg-min-set* (*score d C E*) (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
using *score-is-closest-preimg-dist r-in-lim-set CollectI*
arg-min-set.simps is-arg-min-def
by *metis*
thus $R \in$ *singleton-set-system*
(*arg-min-set* (*score d C E*) (*limit* (*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* (*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
by *auto*
hence $\nexists$ r'. $r' \in$ *limit* (*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*
$\wedge$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E* $\{r'\}$
$<$ *closest-preimg-distance*
(*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*).
$\exists\ r' \in$ *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*. $R' = \{r'\}$
by *auto*
ultimately have
$\nexists\ R'.\ R' \in$ *singleton-set-system* (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
$\wedge$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E R'*
$<$ *closest-preimg-distance*
(*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E R*
using *res-singleton*
by *auto*
moreover have
R $\in$ *singleton-set-system* (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
using *r-min-lim-set res-singleton arg-min-subset*
by *fastforce*
ultimately show
R $\in$ *arg-min-set*
(*closest-preimg-distance*
(*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*)) (*elections-$\mathcal{K}$ C*) *d E*)
(*singleton-set-system* (*limit* (*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* (*alternatives-$\mathcal{E}$ E*) *UNIV*)) =
fun$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*
by *simp*
finally have $\bigcup$ *?min* = $\bigcup$ (*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* = $\bigcup$ *?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: *reflp-on'* (*elections-$\mathcal{K}$ C*) (*Restrp rel* (*elections-$\mathcal{K}$ C*)) **and**
tot-invar-d: *total-invariance$_{\mathcal{D}}$ d rel* **and**
invar-res:
is-symmetry (λ *E*. *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) (*Invariance rel*)
shows *is-symmetry* (*fun$_{\mathcal{E}}$* (*distance-$\mathcal{R}$ d C*)) (*Invariance rel*)
proof −
let *?min* =

λ *E*. $\bigcup \circ$ (*minimizer* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)))
have $\forall$ *E*. *is-symmetry* (*?min E*) (*Invariance rel*)
using *r-refl tot-invar-d invar-comp*
refl-rel-and-tot-invar-dist-imp-invar-minimizer
by *blast*
moreover have *is-symmetry ?min* (*Invariance rel*)
using *invar-res*
by *auto*
ultimately have *is-symmetry* (λ *E*. *?min E E*) (*Invariance rel*)
using *invar-parameterized-fun*[*of ?min*]
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-*$\mathcal{R}_\mathcal{W}$: *is-symmetry* (*fun*$_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*)) (*Invariance rel*)
by *simp*
hence
is-symmetry (λ *E*. *limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV* $-$ *fun*$_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*) *E*)
(*Invariance rel*)
using *invar-res*
by *fastforce*
thus *is-symmetry* (*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 :: *'b monoid* **and**
φ :: (*'b*, (*'a*, *'v*) *Election*) *binary-fun* **and**
B :: (*'a*, *'v*) *Election set*
defines
rel $\equiv$ *action-induced-rel* (*carrier G*) *B* φ **and**
rel' $\equiv$ *action-induced-rel* (*carrier G*) (*elections-*$\mathcal{K}$ *C*) φ
assumes
action-φ: *group-action G B* φ **and**
consensus-C-in-B: *elections-*$\mathcal{K}$ *C* $\subseteq$ *B* **and**
closed-domain:
closed-restricted-rel rel B (*elections-*$\mathcal{K}$ *C*) **and**
invar-res:
is-symmetry (λ *E*. *limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) (*Invariance rel*) **and**
invar-d: *invariance*$_\mathcal{D}$ *d* (*carrier G*) *B* φ **and**
invar-C-winners: *is-symmetry* (*elect-r* $\circ$ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*Invariance rel'*)
shows *is-symmetry* (*fun*$_\mathcal{E}$ (*distance-*$\mathcal{R}$ *d C*)) (*Invariance rel*)
proof $-$

let *?min* =
  λ *E*. $\bigcup$ $\circ$ (*minimizer* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
    (*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)))
have $\forall$ *E*. *is-symmetry* (*?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 *is-symmetry ?min* (*Invariance rel*)
  using *invar-res*
  by *auto*
ultimately have *is-symmetry* (λ *E*. *?min E E*) (*Invariance rel*)
  using *invar-parameterized-fun*[*of ?min*]
  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-*$\mathcal{R}_{\mathcal{W}}$:
  is-symmetry (*fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)) (*Invariance rel*)
  by *simp*
hence *is-symmetry* (λ *E*. *limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV* −
  fun$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) *E*) (*Invariance rel*)
  using *invar-res*
  by *fastforce*
thus *is-symmetry* (*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 :: *′b monoid* **and**
    φ :: (*′b*, (*′a*, *′v*) *Election*) *binary-fun* **and**
    ψ :: (*′b*, *′r*) *binary-fun* **and**
    B :: (*′a*, *′v*) *Election set*
  defines
    rel ≡ *action-induced-rel* (*carrier G*) *B* φ **and**
    rel′ ≡ *action-induced-rel* (*carrier G*) (*elections-*$\mathcal{K}$ *C*) φ **and**
    equivar-prop ≡
      action-induced-equivariance (*carrier G*) (*elections-*$\mathcal{K}$ *C*)
        φ (*set-action* ψ) **and**
    equivar-prop-global-set-valued ≡
        action-induced-equivariance (*carrier G*) *B* φ (*set-action* ψ) **and**
    equivar-prop-global-result-valued ≡

action-induced-equivariance (*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-*$\mathcal{K}$ $C \subseteq B$ **and**
closed-domain: *closed-restricted-rel rel* B (*elections-*$\mathcal{K}$ C) **and**
equivar-res:
is-symmetry (λ E. *limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)
equivar-prop-global-set-valued **and**
invar-d: *invariance*$_{\mathcal{D}}$ d (*carrier* G) B φ **and**
equivar-C-winners: *is-symmetry* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) *equivar-prop*

shows *is-symmetry* (*fun*$_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ d C)) *equivar-prop-global-result-valued*

proof −

let *?min-E* =
λ E. *minimizer* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) d
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)) E

let *?min* =
λ E. $\bigcup$ $\circ$ (*minimizer* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) (*elections-*$\mathcal{K}$ C) d
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)))

let *?*ψ' = *set-action* (*set-action* ψ)

let *?equivar-prop-global-set-valued*$'$ =
action-induced-equivariance (*carrier* G) B φ *?*ψ'

have $\forall$ E g. $g \in$ *carrier* G $\longrightarrow$ $E \in B$ $\longrightarrow$
singleton-set-system (*limit* (*alternatives-*$\mathcal{E}$ (φ g E)) *UNIV*) =
$\{\{r\}$ | r. $r \in$ *limit* (*alternatives-*$\mathcal{E}$ (φ g E)) *UNIV*$\}$
by *simp*

moreover have
$\forall$ E g. $g \in$ *carrier* G $\longrightarrow$ $E \in B$ $\longrightarrow$
limit (*alternatives-*$\mathcal{E}$ (φ g E)) *UNIV* =
ψ g ‘ (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)
using *equivar-res action-*φ *group-action.element-image*
unfolding *equivar-prop-global-set-valued-def action-induced-equivariance-def*
by *fastforce*

ultimately have $\forall$ E g. $g \in$ *carrier* G $\longrightarrow$ $E \in B$ $\longrightarrow$
singleton-set-system (*limit* (*alternatives-*$\mathcal{E}$ (φ g E)) *UNIV*) =
$\{\{r\}$ | r. $r \in \psi$ g ‘ (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)$\}$
by *simp*

moreover have
$\forall$ E g. $\{\{r\}$ | r. $r \in \psi$ g ‘ (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*)$\}$ =
$\{\psi$ g ‘ $\{r\}$ | r. $r \in$ *limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*$\}$
by *blast*

moreover have
$\forall$ E g. $\{\psi$ g ‘ $\{r\}$ | r. $r \in$ *limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*$\}$ =
?ψ' g $\{\{r\}$ | r. $r \in$ *limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*$\}$
unfolding *set-action.simps*
by *blast*

ultimately have
is-symmetry (λ E. *singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*))
?equivar-prop-global-set-valued$'$

using *rewrite-equivariance*[*of*
λ E. singleton-set-system (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
carrier G B φ ?ψ']
by *force*
moreover have *group-action G UNIV* (*set-action ψ*)
unfolding *set-action.simps*
using *group-act-induces-set-group-act*[*of - UNIV*] *group-act-res*
by *simp*
ultimately have *is-symmetry ?min-E ?equivar-prop-global-set-valued'*
using *action-φ invar-d cons-elect-set closed-domain equivar-C-winners*
group-action-invar-dist-and-equivar-f-imp-equivar-minimizer[*of*
G B φ set-action ψ elections-$\mathcal{K}$ C
λ E. singleton-set-system (*limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
d elect-r ∘ $fun_{\mathcal{E}}$ (*rule-$\mathcal{K}$ C*)]
unfolding *rel'-def rel-def equivar-prop-def*
by *metis*
moreover have
is-symmetry
⋃ (*action-induced-equivariance*
(*carrier G*) *UNIV ?ψ'* (*set-action ψ*))
using *equivar-union-under-image-action*[*of - ψ*]
by *simp*
ultimately have *is-symmetry* (⋃ ∘ *?min-E*) *equivar-prop-global-set-valued*
unfolding *equivar-prop-global-set-valued-def*
using *equivar-ind-by-action-comp*[*of - - UNIV*]
by *simp*
moreover have (*λ E. ?min E E*) = ⋃ ∘ *?min-E*
unfolding *comp-def*
by *simp*
ultimately have
is-symmetry (*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-*$\mathcal{R}_{\mathcal{W}}$:
is-symmetry ($fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*)) *equivar-prop-global-set-valued*
by *simp*
moreover have ∀ *g* ∈ *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
is-symmetry (*λ E. limit* (*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 *action-induced-equivariance-def set-action.simps*
equivar-prop-global-set-valued-def

by *blast*
thus *is-symmetry* ($fun_{\mathcal{E}}$ (*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-equivariance
by *simp*
qed

5.6.3 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' well-formed-elections d* **and**
anon-C: *consensus-rule-anonymity'* (*elections-*$\mathcal{K}$ *C*) *C* **and**
closed-C: *closed-restricted-rel* ($anonymity_{\mathcal{R}}$ *well-formed-elections*)
well-formed-elections (*elections-*$\mathcal{K}$ *C*)
shows *anonymity'* (*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 well-formed-elections* π E
using *cons-domain-valid extensional-continuation-subset*
unfolding *φ-anon.simps*
by *metis*
hence *action-induced-rel* (*carrier* $bijection_{\mathcal{VG}}$) (*elections-*$\mathcal{K}$ *C*)
(*φ-anon well-formed-elections*) =
action-induced-rel (*carrier* $bijection_{\mathcal{VG}}$) (*elections-*$\mathcal{K}$ *C*)
(*φ-anon* (*elections-*$\mathcal{K}$ *C*))
using *coinciding-actions-ind-equal-rel*
by *metis*
hence *is-symmetry* (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*))
(*Invariance* (*action-induced-rel*
(*carrier* $bijection_{\mathcal{VG}}$) (*elections-*$\mathcal{K}$ *C*) (*φ-anon well-formed-elections*)))
using *anon-C*
unfolding *consensus-rule-anonymity'.simps* $anonymity_{\mathcal{R}}$*.simps*
by *presburger*
thus *?thesis*
using *anon-d closed-C cons-domain-valid anonymous-group-action.group-action-axioms*
anonymity-action-presv-symmetry invar-dist-cons-imp-invar-dr-rule
unfolding *distance-anonymity'.simps* $anonymity_{\mathcal{R}}$*.simps anonymity'.simps*
anonymity-in.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
neutral-d: *distance-neutrality well-formed-elections d* **and**
neutral-C: *consensus-rule-neutrality* (*elections-$\mathcal{K}$ C*) *C* **and**
closed-C: *closed-restricted-rel* (*neutrality$_{\mathcal{R}}$ well-formed-elections*)
well-formed-elections (*elections-$\mathcal{K}$ C*)
shows *neutrality* (*distance-$\mathcal{R}$ d C*)
proof −
have $\forall\ \pi.\ \forall\ E \in$ *elections-$\mathcal{K}$ C*.
φ-neutral well-formed-elections π E = *φ-neutral* (*elections-$\mathcal{K}$ C*) π E
using *cons-domain-valid extensional-continuation-subset*
unfolding *φ-neutral.simps*
by *metis*
hence *is-symmetry* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*))
(*action-induced-equivariance* (*carrier bijection$_{\mathcal{AG}}$*) (*elections-$\mathcal{K}$ C*)
(*φ-neutral well-formed-elections*) (*set-action* ψ))
using *neutral-C equivar-ind-by-act-coincide*
unfolding *consensus-rule-neutrality.simps*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
using *neutral-d closed-C φ-neutral-action.group-action-axioms*
neutrality action-neutral cons-domain-valid[*of C*]
invar-dist-equivar-cons-imp-equivar-dr-rule[*of*
- - *φ-neutral well-formed-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
reverse-sym-d: *distance-reversal-symmetry well-formed-elections d* **and**
reverse-sym-C: *consensus-rule-reversal-symmetry* (*elections-$\mathcal{K}$ C*) *C* **and**
closed-C: *closed-restricted-rel* (*reversal$_{\mathcal{R}}$ well-formed-elections*)
well-formed-elections (*elections-$\mathcal{K}$ C*)
shows *reversal-symmetry* (*$\mathcal{SWF}$-result.distance-$\mathcal{R}$ d C*)
proof −
have $\forall\ \pi.\ \forall\ E \in$ *elections-$\mathcal{K}$ C*.
φ-reverse well-formed-elections π E = *φ-reverse* (*elections-$\mathcal{K}$ C*) π E
using *cons-domain-valid extensional-continuation-subset*
unfolding *φ-reverse.simps*
by *metis*
hence *is-symmetry* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*))
(*action-induced-equivariance* (*carrier reversal$_{\mathcal{G}}$*) (*elections-$\mathcal{K}$ C*)
(*φ-reverse well-formed-elections*) (*set-action ψ-reverse*))
using *reverse-sym-C equivar-ind-by-act-coincide*
unfolding *consensus-rule-reversal-symmetry.simps*

```
    by (metis (no-types, lifting))
  thus ?thesis
    using reverse-sym-d closed-C reversal-symmetry-action-presv-symmetry
          SWF-result.invar-dist-equivar-cons-imp-equivar-dr-rule
          φ-reverse-action.group-action-axioms cons-domain-valid
          ψ-reverse-action.group-action-axioms
    unfolding reversal-symmetry.simps reversal-symmetry-in-def
              reversal_R.simps distance-reversal-symmetry.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-elections-V d
  shows homogeneity (distance-R d C)
proof −
  have Restrp (homogeneity_R finite-elections-V) (elections-K C) =
          homogeneity_R (elections-K C)
    using cons-domain-finite
    unfolding homogeneity_R.simps finite-elections-V-def
    by blast
  hence reflp-on' (elections-K C)
      (Restrp (homogeneity_R finite-elections-V) (elections-K C))
    using refl-homogeneity_R[of elections-K C] cons-domain-finite[of C]
    by presburger
  moreover have
    is-symmetry (λ E. limit (alternatives-E E) UNIV)
        (Invariance (homogeneity_R finite-elections-V))
    using homogeneity-action-presv-symmetry
    by simp
  ultimately show ?thesis
    using assms tot-invar-dist-imp-invar-dr-rule
    unfolding distance-homogeneity-def homogeneity.simps homogeneity-in.simps
    by blast
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-elections-V d
  shows homogeneity' (distance-R d C)
proof (unfold homogeneity'.simps homogeneity'-in.simps)
  have Restrp (homogeneity_R' finite-elections-V) (elections-K C) =
          homogeneity_R' (elections-K C)
    using cons-domain-finite
    unfolding homogeneity_R'.simps finite-elections-V-def
```

by *blast*
hence *reflp-on'* (*elections-$\mathcal{K}$ C*)
(*Restrp* (*homogeneity$_\mathcal{R}$' finite-elections-$\mathcal{V}$*) (*elections-$\mathcal{K}$ C*))
using *refl-homogeneity$_\mathcal{R}$'*[*of elections-$\mathcal{K}$ C*] *cons-domain-finite*[*of C*]
by *presburger*
moreover have
is-symmetry (λ *E. limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*)
(*Invariance* (*homogeneity$_\mathcal{R}$' finite-elections-$\mathcal{V}$*))
using *homogeneity'-action-presv-symmetry*
by *simp*
ultimately show
is-symmetry (*fun$_\mathcal{E}$* (*distance-$\mathcal{R}$ d C*)) (*Invariance* (*homogeneity$_\mathcal{R}$' finite-elections-$\mathcal{V}$*))
using *assms tot-invar-dist-imp-invar-dr-rule*
unfolding *distance-homogeneity'-def*
by *blast*
qed

end

5.7 Distance Rationalization on Election Quotients

theory *Quotient-Distance-Rationalization*
imports *Quotient-Module*
Distance-Rationalization-Symmetry
begin

5.7.1 Distances

fun *distance$_\mathcal{Q}$* :: *'x Distance* $\Rightarrow$ *'x set Distance* **where**
distance$_\mathcal{Q}$ d A B = (*if A* = {} $\wedge$ *B* = {} *then 0 else*
(*if A* = {} $\vee$ *B* = {} *then* ∞ *else*
$\pi_\mathcal{Q}$ (*tup d*) (*A* $\times$ *B*)))

fun *relation-paths* :: *'x rel* $\Rightarrow$ *'x list set* **where**
relation-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-paths r X Y =
{*x#p@[y]* | *x y p. x* $\in$ *X* $\wedge$ *y* $\in$ *Y* $\wedge$ *p* $\in$ *relation-paths r*}

fun *path-length* :: *'x list* $\Rightarrow$ *'x Distance* $\Rightarrow$ *ereal* **where**
path-length [] *d* = *0* |
path-length [*x*] *d* = *0* |
path-length (*x#y#xs*) *d* = *d x y* + *path-length xs d*

fun *quotient-dist* :: *'x rel* $\Rightarrow$ *'x Distance* $\Rightarrow$ *'x set Distance* **where**

quotient-dist r d A B =
Inf ($\bigcup$ {{*path-length* p d | p. $p \in$ *admissible-paths* r A B}})

fun *distance-infimum*$_{\mathcal{Q}}$ :: $'x$ *Distance* $\Rightarrow$ $'x$ *set Distance* **where**
distance-infimum$_{\mathcal{Q}}$ 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.
distance-infimum$_{\mathcal{Q}}$ 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 that minimizes the infimum distance between A and all B such that the infimum distance between these sets coincides with the infimum distance over all b in B for a fixed a.

fun *product'* :: $'x$ *rel* $\Rightarrow$ ($'x * 'x$) *rel* **where**
product' 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 (*total-invariance*$_{\mathcal{D}}$ d r) = (*tup* d *respects* (*product* r))
unfolding *total-invariance*$_{\mathcal{D}}$*.simps is-symmetry.simps congruent-def*
by *blast*

lemma *product-helper*:
fixes
r :: $'x$ *rel* **and**
X :: $'x$ *set*
shows
trans-imp: *Relation.trans* r $\longrightarrow$ *Relation.trans* (*product* r) **and**
refl-imp: *refl-on* X r $\longrightarrow$ *refl-on* ($X \times X$) (*product* r) **and**
sym: *sym-on* X r $\longrightarrow$ *sym-on* ($X \times X$) (*product* r)
unfolding *Relation.trans-def refl-on-def sym-on-def product.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: *total-invariance*$_{\mathcal{D}}$ d r

shows $\forall\ A\ B.\ A \in X\ //\ r \wedge B \in X\ //\ r$
$\longrightarrow (\forall\ a\ b.\ a \in A \wedge b \in B \longrightarrow distance_{\mathcal{Q}}\ d\ A\ B = d\ a\ b)$
proof (*safe*)
fix
$A\ B :: 'x\ set$ **and**
$a\ b :: 'x$
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*: $A = r$ ‘‘ $\{a\}$
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*: $B = r$ ‘‘ $\{b\}$
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 $A \times B \in (X \times X)\ //\ (product\ r)$
unfolding *quotient-def*
by *fastforce*
moreover have $equiv\ (X \times X)\ (product\ r)$
using *equiv-X-r product-helper UNIV-Times-UNIV equivE equivI*
by *metis*
moreover have $tup\ d\ respects\ (product\ r)$
using *tot-inv-dist-d-r tot-dist-invariance-is-congruence*
by *metis*
ultimately show $distance_{\mathcal{Q}}\ d\ A\ B = d\ a\ b$
unfolding *distance*$_{\mathcal{Q}}$*.simps*
using *pass-to-quotient a-in-A b-in-B*
by *fastforce*
qed

lemma *relation-paths-subset*:

fixes
  n :: *nat* **and**
  p :: *'x list* **and**
  r :: *'x rel* **and**
  X :: *'x set*
assumes $r \subseteq X \times X$
shows $\forall\ p.\ p \in$ *relation-paths* $r \longrightarrow (\forall\ i <$ *length* $p.\ p!i \in X)$

proof (*safe*)
fix
  p :: *'x list* **and**
  i :: *nat*
assume $p \in$ *relation-paths* r
then obtain k :: *nat* **where**
  len-p: *length* $p = 2 * k$ **and**
  rel: $\forall\ i < k.\ (p!(2 * i),\ p!(2 * i + 1)) \in r$
  by *auto*
moreover obtain k' :: *nat* **where**
  i-cases: $i = 2 * k' \lor i = 2 * k' + 1$
  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 $p!i \in X$
  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**
  $a\ b$ :: *'x* **and**
  p :: *'x list*
assumes *refl-on* $X\ r$
shows *triangle-ineq* $X\ d \land p \in$ *relation-paths* $r \land$ *total-invariance*$_{\mathcal{D}}$ $d\ r$
  $\land\ a \in X \land b \in X \longrightarrow$ *path-length* $(a\#p@[b])\ d \geq d\ a\ b$
proof (*clarify*, *induction* $p\ d$ *arbitrary*: $a\ b$ *rule*: *path-length.induct*)
case (*1* d)
show $d\ a\ b \leq$ *path-length* $(a\#[]@[b])\ d$
  by *simp*
next
case (*2* $x\ d$)
thus $d\ a\ b \leq$ *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: $a \in X$ **and**
b-in-X: $b \in X$ **and**
rel: $x\#y\#xs \in$ *relation-paths r* **and**
invar: *total-invariance*$_{\mathcal{D}}$ *d r* **and**
hyp:
$\bigwedge$ *a b. triangle-ineq X d* $\Longrightarrow$ *xs* $\in$ *relation-paths r*
$\Longrightarrow$ *total-invariance*$_{\mathcal{D}}$ *d r* $\Longrightarrow a \in X \Longrightarrow b \in X$
$\Longrightarrow d\ a\ b \leq$ *path-length* $(a\#xs@[b])\ d$
then obtain k :: *nat* **where**
len: *length* $(x\#y\#xs) = 2 * k$
by *auto*
moreover have $\forall\ i < k - 1.\ (xs!(2 * i),\ xs!(2 * i + 1)) =$
$((x\#y\#xs)!(2 * (i + 1)),\ (x\#y\#xs)!(2 * (i + 1) + 1))$
by *simp*
ultimately have $\forall\ i < k - 1.\ (xs!(2 * i),\ xs!(2 * i + 1)) \in r$
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* $\in$ *relation-paths r*
by *simp*
hence $\forall\ x\ y.\ x \in X \land y \in X \longrightarrow d\ x\ y \leq$ *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*: $(x,\ y) \in r$
using *rel*
unfolding *relation-paths.simps*
by *fastforce*
ultimately have *path-length* $(a\#(x\#y\#xs)@[b])\ d \geq d\ a\ x + d\ y\ b$
using *assms add-left-mono assms refl-onD2 b-in-X*
unfolding *refl-on-def*
by *metis*
moreover have $d\ y\ b = d\ x\ b$
using *invar x-rel-y rewrite-total-invariance*$_{\mathcal{D}}$ *assms b-in-X*
unfolding *refl-on-def*
by *fastforce*
moreover have $d\ a\ x + d\ x\ b \geq d\ a\ b$
using *a-in-X b-in-X x-rel-y assms ineq*
unfolding *refl-on-def triangle-ineq-def*
by *auto*
ultimately show $d\ a\ b \leq$ *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: *total-invariance*$_{\mathcal{D}}$ d r
  shows $\forall\ A \in X\ //\ r.\ \forall\ B \in X\ //\ r.$ *quotient-dist* r d A B $=$ *distance*$_{\mathcal{Q}}$ d A B
proof (*clarify*)
  fix A B :: $'x$ *set*
  assume
    A-in-quot-X: $A \in X\ //\ r$ **and**
    B-in-quot-X: $B \in X\ //\ r$
  then obtain
    a b :: $'x$ **where**
      el: $a \in A \wedge b \in B$ **and**
      def-dist: *distance*$_{\mathcal{Q}}$ d A B $=$ d a b
    using *dist-pass-to-quotient assms in-quotient-imp-non-empty ex-in-conv*
    by (*metis* (*full-types*))
  have $b \in X$
    using *B-in-quot-X el equiv quotient-eq-iff equiv equiv-class-eq-iff*
    by *metis*
  hence $B = r$ ‘‘ $\{b\}$
    using *equiv-class-self B-in-quot-X el equiv quotientI quotient-eq-iff*
    by *metis*
  moreover have $a \in X$
    using *A-in-quot-X el equiv quotient-eq-iff equiv equiv-class-eq-iff*
    by *metis*
  ultimately have *equiv-class*: $A = r$ ‘‘ $\{a\} \wedge B = r$ ‘‘ $\{b\}$
    using *A-in-quot-X el equiv quotientI quotient-eq-iff*
    by *slow*
  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*
    unfolding *equiv-def*
    by *blast*
  moreover have $r \subseteq X \times X \wedge A \subseteq X \wedge B \subseteq X$
    using *assms A-in-quot-X B-in-quot-X Union-quotient Union-upper*
    unfolding *equiv-def refl-on-def*
    by *metis*

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 el invar in-mono*
  by *metis*
hence $\forall$ *l*. *l* $\in \bigcup$ {{*path-length p d* | *p*. *p* $\in$ *admissible-paths r A B*}}
  $\longrightarrow$ *l* $\geq$ *d a b*
  by *blast*
hence *geq*: *quotient-dist r d A B* $\geq$ $distance_{\mathcal{Q}}$ *d A B*
  unfolding *quotient-dist.simps le-Inf-iff*
  using *def-dist*
  by *simp*
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*
  unfolding *quotient-dist.simps*
  using *CollectI Inf-lower ccpo-Sup-singleton*
  by (*metis* (*mono-tags*, *lifting*))
thus *quotient-dist r d A B* = $distance_{\mathcal{Q}}$ *d A B*
  using *geq def-dist nle-le*
  by *metis*
qed

lemma $inf\text{-}dist\text{-}coincides\text{-}with\text{-}dist_{\mathcal{Q}}$:
fixes
  d :: *'x Distance* **and**
  r :: *'x rel* **and**
  X :: *'x set*
assumes
  equiv-X-r: *equiv X r* **and**
  tot-inv-d-r: $total\text{-}invariance_{\mathcal{D}}$ *d r*
shows $\forall$ *A* $\in$ *X* // *r*. $\forall$ *B* $\in$ *X* // *r*.
  $distance\text{-}infimum_{\mathcal{Q}}$ *d A B* = $distance_{\mathcal{Q}}$ *d A B*
proof (*clarify*)
fix *A B* :: *'x set*
assume
  A-in-quot-X: *A* $\in$ *X* // *r* **and**
  B-in-quot-X: *B* $\in$ *X* // *r*
then obtain
  a b :: *'x* **where**
    el: *a* $\in$ *A* $\wedge$ *b* $\in$ *B* **and**
    def-dist: $distance_{\mathcal{Q}}$ *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 $\textit{distance-infimum}_{\mathcal{Q}}\ d\ A\ B = \textit{distance}_{\mathcal{Q}}\ d\ A\ B$
unfolding $\textit{distance-infimum}_{\mathcal{Q}}.\textit{simps}$
using *def-dist*
by *simp*
qed

lemma *inf-helper*:
fixes
$A\ B :: {'x}\ \textit{set}$ **and**
$d :: {'x}\ \textit{Distance}$
shows $\textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\} =$
$\textit{Inf}\ \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
proof $-$
have $\forall\ a\ b.\ a \in A \wedge b \in B \longrightarrow \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \leq d\ a\ b$
using *INF-lower Setcompr-eq-image*
by *metis*
hence $\forall\ \alpha \in \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}.$
$\exists\ \beta \in \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}.\ \beta \leq \alpha$
by *blast*
hence $\textit{Inf}\ \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
$\leq \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *Inf-mono*
by (*metis* (*no-types*, *lifting*))
moreover have
$\neg\ \textit{Inf}\ \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
$< \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
proof (*rule ccontr*, *safe*)
assume $\textit{Inf}\ \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
$< \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
then obtain $\alpha :: \textit{ereal}$ **where**
inf: $\alpha \in \{\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$ **and**
less: $\alpha < \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
using *Inf-less-iff*
by (*metis* (*no-types*, *lifting*))
then obtain $a :: {'x}$ **where**
a-in-A: $a \in A$ **and**
$\alpha = \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$
by *blast*
with *less*
have *inf-less*: $\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\} < \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$
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 $\textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\} \leq \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$

```
      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: invariance_D d (carrier G) Y φ
  shows simple (action-induced-rel (carrier G) Y φ) Y d
proof (unfold simple.simps, safe)
  fix A :: 'y set
  assume class_Y: A ∈ Y // action-induced-rel (carrier G) Y φ
  moreover have equiv-rel:
    equiv Y (action-induced-rel (carrier G) Y φ)
    using assms rel-ind-by-group-act-equiv
    by blast
  ultimately obtain a :: 'y where
    a-in-A: a ∈ A
    using equiv-Eps-in
    by blast
  have subset: ∀ B ∈ Y // action-induced-rel (carrier G) Y φ. B ⊆ Y
    using equiv-rel in-quotient-imp-subset
    by blast
  hence ∀ B ∈ Y // action-induced-rel (carrier G) Y φ.
        ∀ B' ∈ Y // action-induced-rel (carrier G) Y φ.
          ∀ b ∈ B. ∀ c ∈ B'. b ∈ Y ∧ c ∈ Y
    using class_Y
    by blast
  hence eq-dist:
    ∀ B ∈ Y // action-induced-rel (carrier G) Y φ.
      ∀ B' ∈ Y // action-induced-rel (carrier G) Y φ.
        ∀ b ∈ B. ∀ c ∈ B'. ∀ g ∈ carrier G.
          d (φ g c) (φ g b) = d c b
    using invar rewrite-invariance_D class_Y
    by metis
  have ∀ b ∈ Y. ∀ g ∈ carrier G.
        (b, φ g b) ∈ action-induced-rel (carrier G) Y φ
```

unfolding *action-induced-rel.simps*
using *group-action.element-image action-*φ
by *fastforce*
hence $\forall\ b \in Y.\ \forall\ g \in carrier\ G.$
$\varphi\ g\ b \in action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi\ `` \{b\}$
unfolding *Image-def*
by *blast*
moreover have *equiv-class*:
$\forall\ B.\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi \longrightarrow$
$(\forall\ b \in B.\ B = action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi\ `` \{b\})$
using *Image-singleton-iff equiv-class-eq-iff equiv-rel*
quotientI quotient-eq-iff
by *meson*
ultimately have *closed-class*:
$\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$\forall\ b \in B.\ \forall\ g \in carrier\ G.\ \varphi\ g\ b \in B$
using *equiv-rel subset*
by *blast*
with *eq-dist* $class_Y$
have *a-subset-A*:
$\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$\{d\ a\ b \mid b.\ b \in B\} \subseteq \{d\ a\ b \mid a\ b.\ a \in A \land b \in B\}$
using *a-in-A*
by *blast*
have $\forall\ a' \in A.\ A = action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi\ `` \{a'\}$
using $class_Y$ *equiv-rel equiv-class*
by *presburger*
hence $\forall\ a' \in A.\ (a',\ a) \in action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi$
using *a-in-A*
by *blast*
hence $\forall\ a' \in A.\ \exists\ g \in carrier\ G.\ \varphi\ g\ a' = a$
by *simp*
hence $\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$\forall\ a'\ b.\ a' \in A \land b \in B \longrightarrow (\exists\ g \in carrier\ G.\ d\ a'\ b = d\ a\ (\varphi\ g\ b))$
using *eq-dist* $class_Y$
by *metis*
hence $\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$\forall\ a'\ b.\ a' \in A \land b \in B \longrightarrow d\ a'\ b \in \{d\ a\ b \mid b.\ b \in B\}$
using *closed-class mem-Collect-eq*
by *fastforce*
hence $\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$\{d\ a\ b \mid b.\ b \in B\} \supseteq \{d\ a\ b \mid a\ b.\ a \in A \land b \in B\}$
using *closed-class*
by *blast*
with *a-subset-A*
have $\forall\ B \in Y\ //\ action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi.$
$distance\text{-}infimum_{\mathcal{Q}}\ d\ A\ B = Inf\ \{d\ a\ b \mid b.\ b \in B\}$
unfolding $distance\text{-}infimum_{\mathcal{Q}}.simps$
by *fastforce*

thus $\exists\ a \in A.\ \forall\ B \in Y\ //\ \textit{action-induced-rel}\ (\textit{carrier}\ G)\ Y\ \varphi.$
    $\textit{distance-infimum}_{\mathcal{Q}}\ d\ A\ B = \textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$
  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: $\textit{total-invariance}_{\mathcal{D}}\ d\ r$
 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 Image-singleton-iff equiv-class-eq equiv-on-X quotientE*
  by *metis*
 hence $\forall\ a\ a'.\ a \in A \land a' \in A \longrightarrow (a, a') \in r$
  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
  $\forall\ B \in X\ //\ r.\ \forall\ a\ a'\ b.\ a \in A \land a' \in A \land b \in B \longrightarrow d\ a\ b = d\ a'\ b$
  using *invar* $\textit{rewrite-total-invariance}_{\mathcal{D}}$
  by *simp*
 hence $\forall\ B \in X\ //\ r.$
  $\{d\ a\ b \mid a\ b.\ a \in A \land b \in B\} = \{d\ a\ b \mid a'\ b.\ a' \in A \land b \in B\}$
  using *a-in-A*
  by *blast*
 moreover have
  $\forall\ B \in X\ //\ r.\ \{d\ a\ b \mid a'\ b.\ a' \in A \land b \in B\} =$
    $\{d\ a\ b \mid b.\ b \in B\}$
  using *a-in-A*
  by *blast*
 ultimately have
  $\forall\ B \in X\ //\ r.\ \textit{Inf}\ \{d\ a\ b \mid a\ b.\ a \in A \land b \in B\} =$
    $\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$
  by *simp*
 hence $\forall\ B \in X\ //\ r.\ \textit{distance-infimum}_{\mathcal{Q}}\ d\ A\ B =$
    $\textit{Inf}\ \{d\ a\ b \mid b.\ b \in B\}$

by *simp*
thus $\exists\ a \in A.\ \forall\ B \in X\ //\ r.$
distance-infimum$_{\mathcal{Q}}$ $d\ A\ B = Inf\ \{d\ a\ b\ |\ b.\ b \in B\}$
using *a-in-A*
by *blast*
qed

5.7.2 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*$_{\mathcal{Q}}$ :: $('a,\ 'v)$ *Election set* $\Rightarrow$ $'r$ *set* $\Rightarrow$ $'r$ *set* **where**
limit$_{\mathcal{Q}}$ $X\ res = \bigcap\ \{limit\ (alternatives\text{-}\mathcal{E}\ E)\ res\ |\ E.\ E \in X\}$

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:
fixes
$X\ Y\ Z$ :: $'x$ *set* **and**
r :: $'x$ *rel*
assumes
equiv $X\ r$ **and**
$Y \subseteq X$ **and**
$Z \subseteq X$ **and**
$Z \in Y\ //\ r$ **and**
closed-restricted-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-restricted-rel.simps restricted-rel.simps*
by *blast*
qed

lemma (**in** *result*) *limit-invar*:
  fixes
    d :: ($'a$, $'v$) *Election Distance* **and**
    r :: ($'a$, $'v$) *Election rel* **and**
    C :: ($'a$, $'v$, $'r$ *Result*) *Consensus-Class* **and**
    X A :: ($'a$, $'v$) *Election set*
  assumes
    quot-class: $A \in X$ // r **and**
    equiv-rel: *equiv* X r **and**
    cons-subset: *elections-*$\mathcal{K}$ $C \subseteq X$ **and**
    invar-res: *is-symmetry* (λ E. *limit* (*alternatives-*$\mathcal{E}$ E) *UNIV*) (*Invariance* r)
  shows $\forall$ $a \in A$. *limit* (*alternatives-*$\mathcal{E}$ a) *UNIV* = *limit*$_{\mathcal{Q}}$ 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 (*alternatives-*$\mathcal{E}$ b) *UNIV* = *limit* (*alternatives-*$\mathcal{E}$ a) *UNIV*
    using *invar-res*
    unfolding *is-symmetry.simps*
    by (*metis* (*mono-tags*, *lifting*))
  hence *limit*$_{\mathcal{Q}}$ A *UNIV* = $\bigcap$ {*limit* (*alternatives-*$\mathcal{E}$ a) *UNIV*}
    unfolding *limit*$_{\mathcal{Q}}$*.simps*
    using *a-in-A*
    by *blast*
  thus *limit* (*alternatives-*$\mathcal{E}$ a) *UNIV* = *limit*$_{\mathcal{Q}}$ A *UNIV*
    by *simp*
qed

lemma (**in** *result*) *preimg-invar*:
  fixes
    f :: $'x \Rightarrow 'y$ **and**
    domain$_f$ X :: $'x$ *set* **and**
    d :: $'x$ *Distance* **and**
    r :: $'x$ *rel*
  assumes
    equiv-rel: *equiv* X r **and**
    cons-subset: *domain*$_f$ $\subseteq X$ **and**
    closed-domain: *closed-restricted-rel* r X *domain*$_f$ **and**
    invar-f: *is-symmetry* 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**
$x \in \mathit{preimg}\ f\ \mathit{domain}_f\ y$ **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*: $x \in \mathit{domain}_f \wedge f\ x = y$
unfolding *preimg.simps*
by *blast*
moreover have r `` $\{x\} \subseteq X$
using *equiv-rel equiv-type*
by *fastforce*
ultimately have r `` $\{x\} \subseteq \mathit{domain}_f$
using *closed-domain A-eq-img-singleton-r A-in-dom*
by *fastforce*
hence $\forall\ x' \in r$ `` $\{x\}.\ (x,\ x') \in \mathit{Restr}\ r\ \mathit{domain}_f$
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 *is-symmetry.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 $A \in \mathit{preimg}\ (\pi_{\mathcal{Q}}\ f)\ (\mathit{domain}_f\ //\ r)\ y$
using *A-in-dom*
unfolding *preimg.simps*
by *blast*
next
fix
$A :: {'x}\ \mathit{set}$ **and**
$y :: {'y}$
assume *quot-preimg*: $A \in \mathit{preimg}\ (\pi_{\mathcal{Q}}\ f)\ (\mathit{domain}_f\ //\ r)\ y$
hence *A-in-dom-rel-r*: $A \in \mathit{domain}_f\ //\ r$
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*
        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: $A = r \text{ `` } \{x\}$
    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 $f \text{ ` } A = \{f\ x\}$
    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 $y \in minimizer\ f\ domain_f\ d\ Y\ x =$

$(y \in Y \wedge (\forall\ y' \in Y.$
$\quad Inf\ (d\ x\ `\ (preimg\ f\ domain_f\ y)) \leq Inf\ (d\ x\ `\ (preimg\ f\ domain_f\ y'))))$

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-set-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 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**
$\mathcal{R}_\mathcal{Q}\ r\ d\ C\ A =$
$\bigcup\ (minimizer\ (\pi_\mathcal{Q}\ (elect\text{-}r \circ fun_\mathcal{E}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_\mathcal{Q}\ r\ C)$
$\quad (distance\text{-}infimum_\mathcal{Q}\ d)\ (singleton\text{-}set\text{-}system\ (limit_\mathcal{Q}\ A\ UNIV))\ A)$

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* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) *A* − $\mathcal{R}_\mathcal{Q}$ *r d C A*,
{})

Proposition 4.17 by Hadjibeyli and Wilson [3].

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 :: (*'a*, *'v*) *Election set*
assumes
simple: *simple r X d* **and**
closed-domain: *closed-restricted-rel r X* (*elections-*$\mathcal{K}$ *C*) **and**
invar-res:
is-symmetry (λ *E*. *limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) (*Invariance r*) **and**
invar-C: *is-symmetry* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))
(*Invariance* (*Restr r* (*elections-*$\mathcal{K}$ *C*))) **and**
invar-dr: *is-symmetry* (*fun*$_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*)) (*Invariance r*) **and**
quot-class: *A* ∈ *X* // *r* **and**
equiv-rel: *equiv X r* **and**
cons-subset: *elections-*$\mathcal{K}$ *C* ⊆ *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*:
∀ *y B*. *B* ∈ *preimg* ($\pi_\mathcal{Q}$ (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_\mathcal{Q}$ *r C*) *y*
⟶ *B* ∈ (*elections-*$\mathcal{K}$ *C*) // *r*
by *simp*
have ∀ *y*. ∀ *E*
∈ *preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y*. *E* ∈ *r* `` {*E*}
using *equiv-rel cons-subset equiv-class-self equiv-rel in-mono*
unfolding *equiv-def preimg.simps*
by *fastforce*
hence ∀ *y*.
⋃ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y* // *r*) ⊇
preimg (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y*
unfolding *quotient-def*
by *blast*
moreover have ∀ *y*.
⋃ (*preimg* (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y* // *r*) ⊆
preimg (*elect-r* ∘ *fun*$_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *y*
proof (*intro allI subsetI*)
fix
Y :: *'r set* **and**
E :: (*'a*, *'v*) *Election*
assume *E* ∈ ⋃ (*preimg* (*elect-r* ∘ *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-restricted-rel.simps restricted-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 *is-symmetry.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.
distance-infimum$_{\mathcal{Q}}$ 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*:

$\forall\ y\ B.\ B \in preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))\ (elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ y$
$\quad\longrightarrow distance\text{-}infimum_{\mathcal{Q}}\ d\ A\ B = Inf\ \{d\ a\ b \mid b.\ b \in B\}$
using *preimg-img-imp-cls quot-classes-subset*
by *blast*
have *wf-res-eq*: $singleton\text{-}set\text{-}system\ (limit\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV) =$
$singleton\text{-}set\text{-}system\ (limit_{\mathcal{Q}}\ A\ UNIV)$
using *invar-res a-in-A quot-class cons-subset equiv-rel limit-invar*
by *metis*
have *inf-le-iff*: $\forall\ x.$
$(\forall\ y \in singleton\text{-}set\text{-}system\ (limit\ (alternatives\text{-}\mathcal{E}\ a)\ UNIV).$
$Inf\ (d\ a\ \text{`}\ preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ \{x\})$
$\leq Inf\ (d\ a\ \text{`}\ preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y))$
$= (\forall\ y \in singleton\text{-}set\text{-}system\ (limit_{\mathcal{Q}}\ A\ UNIV).$
$Inf\ (distance\text{-}infimum_{\mathcal{Q}}\ d\ A\ \text{`}\ preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))$
$\quad(elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ \{x\})$
$\leq Inf\ (distance\text{-}infimum_{\mathcal{Q}}\ d\ A\ \text{`}\ preimg\ (\pi_{\mathcal{Q}}\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C)))$
$\quad(elections\text{-}\mathcal{K}_{\mathcal{Q}}\ r\ C)\ y))$
proof –
have *preimg-partition-dist*: $\forall\ y.$
$Inf\ \{d\ a\ b \mid b.\ b \in$
$\bigcup\ (preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y\ //\ r)\} =$
$Inf\ (d\ a\ \text{`}\ preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y)$
using *Setcompr-eq-image preimg-partition*
by *metis*
have $\forall\ y.$
$\{Inf\ \{d\ a\ b \mid b.\ b \in B\}$
$\mid B.\ B \in preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y\ //\ r\}$
$= \{Inf\ E \mid E.\ E \in \{\{d\ a\ b \mid b.\ b \in B\}$
$\mid B.\ B \in preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y\ //\ r\}\}$
by *blast*
hence $\forall\ y.$
$Inf\ \{Inf\ \{d\ a\ b \mid b.\ b \in B\} \mid B.$
$B \in preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y\ //\ r\} =$
$Inf\ (\bigcup\ \{\{d\ a\ b \mid b.\ b \in B\} \mid B.$
$B \in (preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (elections\text{-}\mathcal{K}\ C)\ y\ //\ r)\})$
using *union-inf*
by *presburger*
moreover have
$\forall\ y.$
$\{d\ a\ b \mid b.\ b \in \bigcup$
$(preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))$
$(elections\text{-}\mathcal{K}\ C)\ y\ //\ r)\} =$
$\bigcup\ \{\{d\ a\ b \mid b.\ b \in B\} \mid B.$
$B \in (preimg\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))$
$(elections\text{-}\mathcal{K}\ C)\ y\ //\ r)\}$
by *blast*
ultimately have *rewrite-inf-dist*:
$\forall\ y.\ Inf\ \{Inf\ \{d\ a\ b \mid b.\ b \in B\}$
$\mid B.\ B \in preimg$

(elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)) (elections-$\mathcal{K}$ C) y // r} =
Inf {d a b
| b. b $\in \bigcup$ (preimg
(elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)) (elections-$\mathcal{K}$ C) y // r)}
by presburger
have $\forall$ y. distance-infimum$_{\mathcal{Q}}$ d A ' preimg ($\pi_{\mathcal{Q}}$ (elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)))
(elections-$\mathcal{K}_{\mathcal{Q}}$ r C) y =
{Inf {d a b | b. b $\in$ B}
| 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}
using inf-dist-preimg-sets
unfolding Image-def
by auto
moreover have $\forall$ y.
{Inf {d a b | b. b $\in$ B} | 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} =
{Inf {d a b | b. b $\in$ B} | B.
B $\in$ (preimg (elect-r $\circ$ 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 (distance-infimum$_{\mathcal{Q}}$ d A ' preimg ($\pi_{\mathcal{Q}}$ (elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)))
(elections-$\mathcal{K}_{\mathcal{Q}}$ r C) y) =
Inf {Inf {d a b | b. b $\in$ B}
| B. B $\in$ preimg (elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)) (elections-$\mathcal{K}$ C) y // r}
by simp
thus ?thesis
using wf-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-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)) (elections-$\mathcal{K}$ C) d
(singleton-set-system (limit (alternatives-$\mathcal{E}$ a) UNIV)) a))
using $\mathcal{R}_{\mathcal{W}}$-is-minimizer
by metis
also have ... =
({x} $\in$ minimizer (elect-r $\circ$ fun$_{\mathcal{E}}$ (rule-$\mathcal{K}$ C)) (elections-$\mathcal{K}$ C) d
(singleton-set-system (limit (alternatives-$\mathcal{E}$ 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-set-system (limit (alternatives-$\mathcal{E}$ a) UNIV)

$\wedge$ ($\forall$ $y \in$ *singleton-set-system* (*limit* (*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*)))
using *minimizer-helper*
by (*metis* (*no-types*, *lifting*))
also have ... = ($\{x\} \in$ *singleton-set-system* (*limit$_{\mathcal{Q}}$ A UNIV*)
$\wedge$ ($\forall$ $y \in$ *singleton-set-system* (*limit$_{\mathcal{Q}}$ A UNIV*).
Inf (*distance-infimum$_{\mathcal{Q}}$ d A* ‘ *preimg* ($\pi_{\mathcal{Q}}$ (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*)))
(*elections-$\mathcal{K}_{\mathcal{Q}}$ r C*) $\{x\}$)
$\leq$ *Inf* (*distance-infimum$_{\mathcal{Q}}$ d A* ‘ *preimg* ($\pi_{\mathcal{Q}}$ (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*)))
(*elections-$\mathcal{K}_{\mathcal{Q}}$ r C*) *y*)))
using *wf-res-eq inf-le-iff*
by *blast*
also have ... =
($\{x\} \in$ *minimizer*
($\pi_{\mathcal{Q}}$ (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*))) (*elections-$\mathcal{K}_{\mathcal{Q}}$ r C*)
(*distance-infimum$_{\mathcal{Q}}$ d*)
(*singleton-set-system* (*limit$_{\mathcal{Q}}$ 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*)
(*distance-infimum$_{\mathcal{Q}}$ d*)
(*singleton-set-system* (*limit$_{\mathcal{Q}}$ 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 *safe*
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 :: (*'a*, *'v*) *Election set*
assumes
simple: *simple r X d* **and**
closed-domain: *closed-restricted-rel r X* (*elections-$\mathcal{K}$ C*) **and**
invar-res:
is-symmetry (λ *E. limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) (*Invariance r*) **and**
invar-C: *is-symmetry* (*elect-r* $\circ$ *fun$_{\mathcal{E}}$* (*rule-$\mathcal{K}$ C*))
(*Invariance* (*Restr r* (*elections-$\mathcal{K}$ C*))) **and**

invar-dr: *is-symmetry* ($fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *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 (*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* (*alternatives-$\mathcal{E}$ E*) *UNIV* =
$\pi_{\mathcal{Q}}$ (λ *E*. *limit* (*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-$\mathcal{R}$ d C*) *E* =
($\mathcal{R}_{\mathcal{Q}}$ *r d C A*,
$\pi_{\mathcal{Q}}$ (λ *E*. *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) *A* − $\mathcal{R}_{\mathcal{Q}}$ *r d C A*,
{})
by *fastforce*
hence
$fun_{\mathcal{E}}$ (*distance-$\mathcal{R}$ d C*) ‘ *A* $\subseteq$
{($\mathcal{R}_{\mathcal{Q}}$ *r d C A*,
$\pi_{\mathcal{Q}}$ (λ *E*. *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) *A* − $\mathcal{R}_{\mathcal{Q}}$ *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*,
$\pi_{\mathcal{Q}}$ (λ *E*. *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) *A* − $\mathcal{R}_{\mathcal{Q}}$ *r d C A*,
{})} =
$fun_{\mathcal{E}}$ (*distance-$\mathcal{R}$ d C*) ‘ *A*
using *empty-is-image subset-singletonD*
by (*metis* (*no-types*, *lifting*))
hence *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*
have *the-inv* (λ *x*. {*x*}) ($fun_{\mathcal{E}}$ (*distance-$\mathcal{R}$ d C*) ‘ *A*) =

($\mathcal{R}_\mathcal{Q}$ *r d C A*,
$\pi_\mathcal{Q}$ (λ *E*. *limit* (*alternatives-$\mathcal{E}$ E*) *UNIV*) *A* $-$ $\mathcal{R}_\mathcal{Q}$ *r d C A*,
{})
using *single-img 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 $\pi_\mathcal{Q}$.simps singleton-set.simps*
by *presburger*
qed

end

5.8 Code Generation Interpretations for Results and Properties

theory *Interpretation-Code*
imports *Electoral-Module*
Distance-Rationalization
begin
setup *Locale-Code.open-block*

5.8.1 Code Lemmas

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* =
($\forall$ *A V p*. *profile V A p* $\longrightarrow$ *well-formed-$\mathcal{SCF}$ A* (*m V A p*))
unfolding *$\mathcal{SCF}$-result.electoral-module.simps*
by *safe*

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-$\mathcal{SCF}$ 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-*$\mathcal{SCF}$ 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-*$\mathcal{SCF}$ 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-*$\mathcal{SCF}$ 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*: $\mathcal{SCF}$*-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 $\pi \longrightarrow$ (*let* (A', V', q) = (*rename* π (A, V, p)) *in*
     profile V A p $\wedge$ *profile* V' A' $q \longrightarrow m$ V A p = m V' A' q)))
unfolding $\mathcal{SCF}$*-result.anonymity-def*
by *simp*

5.8.2 Interpretation Declarations and Constants

Declarations for replacing interpreted abstract functions from locales by their explicit instantiations.

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 instead of the interpreted functions.

definition $\mathcal{R}_\mathcal{W}$*-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$
definition $\mathcal{R}_\mathcal{W}$*-std-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-result.*$\mathcal{R}_\mathcal{W}$*-std*
definition *distance-*$\mathcal{R}$*-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-result.distance-*$\mathcal{R}$
definition *distance-*$\mathcal{R}$*-std-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-result.distance-*$\mathcal{R}$*-std*
definition *electoral-module-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-result.electoral-module*
definition *anonymity-*$\mathcal{SCF}$*-code* $\equiv$ $\mathcal{SCF}$*-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**
  drop-module n r V A p =
    ({},
    {$a \in A$. *rank* (*limit A r*) $a \le n$},
    {$a \in A$. *rank* (*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.simps*, *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 $\forall\ a \in A.\ a \in \{x \in A.\ rank\ (limit\ A\ r)\ x \leq n\} \vee$
             $a \in \{x \in A.\ rank\ (limit\ A\ r)\ x > n\}$
    by *auto*
  hence $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\} \cup \{a \in A.\ rank\ (limit\ A\ r)\ a > n\} = A$
    by *blast*
  hence *set-partition*: *set-equals-partition* A (*drop-module* n r V A p)
    by *simp*
  have $\forall\ a \in A.$
         $\neg\ (a \in \{x \in A.\ rank\ (limit\ A\ r)\ x \leq n\} \wedge$
           $a \in \{x \in A.\ rank\ (limit\ A\ r)\ x > n\})$
    by *simp*
  hence $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\} \cap \{a \in A.\ rank\ (limit\ A\ r)\ a > n\} = \{\}$
    by *blast*
  thus *well-formed-*$\mathcal{SCF}$ A (*?mod* V A p)
    using *set-partition*
    by *simp*
qed

lemma *voters-determine-drop-mod*:
  fixes
    r :: $'a$ *Preference-Relation* **and**
    n :: *nat*
  shows *voters-determine-election* (*drop-module* n r)
  unfolding *voters-determine-election.simps*
  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 *auto*

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 *force*

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 =
    ({},
    $\{a \in A.\ rank\ (limit\ A\ r)\ a > n\}$,
    $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\}$)

5.10.2 Soundness

theorem *pass-mod-sound*[*simp*]:
  fixes
    r :: *'a Preference-Relation* **and**
    n :: *nat*
  shows $\mathcal{SCF}$*-result.electoral-module* (*pass-module n r*)
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module.simps*, *safe*)

fix
  A :: $'a$ *set* **and**
  V :: $'v$ *set* **and**
  p :: $('a, 'v)$ *Profile*
let *?mod* = *pass-module n r*
have $\forall\ a \in A.\ a \in \{x \in A.\ rank\ (limit\ A\ r)\ x > n\} \lor$
  $a \in \{x \in A.\ rank\ (limit\ A\ r)\ x \leq n\}$
  using *CollectI not-less*
  by *metis*
hence $\{a \in A.\ rank\ (limit\ A\ r)\ a > n\} \cup \{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\} = A$
  by *blast*
hence *set-equals-partition A* (*pass-module n r V A p*)
  by *simp*
moreover have
  $\forall\ a \in A.$
  $\neg\ (a \in \{x \in A.\ rank\ (limit\ A\ r)\ x > n\} \land$
  $a \in \{x \in A.\ rank\ (limit\ A\ r)\ x \leq n\})$
  by *simp*
hence $\{a \in A.\ rank\ (limit\ A\ r)\ a > n\} \cap \{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\} = \{\}$
  by *blast*
ultimately show *well-formed-$\mathcal{SCF}$ A* (*?mod V A p*)
  by *simp*
qed

lemma *voters-determine-pass-mod*:
fixes
  r :: $'a$ *Preference-Relation* **and**
  n :: *nat*
shows *voters-determine-election* (*pass-module n r*)
unfolding *voters-determine-election.simps 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**
  greater-zero: $n > 0$
shows *non-blocking* (*pass-module n r*)
proof (*unfold non-blocking-def*, *safe*)
show *$\mathcal{SCF}$-result.electoral-module* (*pass-module n r*)
  using *pass-mod-sound*
  by *metis*
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 greater-zero 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 *force*

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 pass-mod-sound*
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 *metis*
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 ∀ *n*. (*n* :: *nat*) ≤ *0* ⟶ *n* = *0*
by *blast*
hence ∀ *a A'*. *a* ∈ *A'* ∧ *a* ∈ *A* ⟶ *connex A'* (*limit A r*) ⟶
¬ *rank* (*limit A r*) *a* ≤ *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* ∈ *A*. *rank* (*limit A r*) *a* ≤ *0*} = {}
using *limit-is-connex*
by *simp*
hence *card* {*a* ∈ *A*. *rank* (*limit A r*) *a* ≤ *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 *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* ≤ *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* ≠ {}
      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*:
      ∃ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ∧
          (∀ *b* ∈ *A*. *above* (*limit A r*) *b* = {*b*} ⟶ *b* = *a*)
      using *finite-A above-one*
      by *simp*
    then obtain *w* :: *'a* **where**
      w-unique-top:
      above (*limit A r*) *w* = {*w*} ∧
        (∀ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ⟶ *a* = *w*)
      using *above-one*
      by *auto*
    hence {*a* ∈ *A*. *rank* (*limit A r*) *a* ≤ *1*} = {*w*}
    proof
      assume
        w-top: *above* (*limit A r*) *w* = {*w*} **and**
        w-unique: ∀ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ⟶ *a* = *w*
      have *rank* (*limit A r*) *w* ≤ *1*
        using *w-top*
        by *auto*
      hence {*w*} ⊆ {*a* ∈ *A*. *rank* (*limit A r*) *a* ≤ *1*}
        using *winner-exists w-unique-top*
        by *blast*
      moreover have {*a* ∈ *A*. *rank* (*limit A r*) *a* ≤ *1*} ⊆ {*w*}
      proof

```
      fix a :: 'a
      assume a-in-winner-set: a ∈ {b ∈ A. rank (limit A r) b ≤ 1}
      hence a-in-A: a ∈ 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 ≼_q a
        using connex-limit above-connex pref-imp-in-above a-in-A
        by metis
      hence (a, a) ∈ limit A r
        by simp
      hence a-above-a: a ∈ above (limit A r) a
        unfolding above-def
        by simp
      have above (limit A r) a ⊆ 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 ≤ 1
        using a-in-winner-set
        by simp
      moreover have rank (limit A r) a ≥ 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 ∈ {w}
        by simp
    qed
    ultimately have {w} = {a ∈ A. rank (limit A r) a ≤ 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 pass-mod-sound*
    by *metis*
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* :: *'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-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 $b :: 'a$ **where**
top-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 *top-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 *top-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*: $b \in defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p$
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 $\forall\ c \in A - \{a,\ b\}.\ c \notin defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p$
by (*simp add*: *not-le*)
moreover have $defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p \subseteq A$
by *auto*
ultimately have $defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p \subseteq \{a,\ b\}$
by *blast*
hence $defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p = \{a,\ b\}$
using *a-in-defer b-in-defer*
by *fastforce*
thus $card\ (defer\ (pass\text{-}module\ 2\ r)\ V\ A\ p) = 2$
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*]: *𝒮𝒞ℱ-result.electoral-module elect-module*
by *simp*

lemma *elect-mod-only-voters*: *voters-determine-election elect-module*
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 =
(*if finite A*
then ({},
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\}$,

$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\})$
else $(\{\}, \{\}, A))$

lemma *enat-leq-enat-set-max*:
fixes
x :: *enat* **and**
X :: *enat set*
assumes
$x \in X$ **and**
finite X
shows $x \leq \textit{Max}\ X$
using *assms*
by *simp*

lemma *plurality-mod-equiv*:
fixes
A :: *'a set* **and**
V :: *'v set* **and**
p :: *('a, 'v) Profile*
shows *plurality* $V\ A\ p$ = *plurality'* $V\ A\ p$
proof (*unfold plurality'.simps*)
have *no-winners-or-in-domain*:
finite $\{\textit{win-count}\ V\ p\ a \mid a.\ a \in A\} \longrightarrow$
$\{\textit{win-count}\ V\ p\ a \mid a.\ a \in A\} = \{\} \vee$
$\textit{Max}\ \{\textit{win-count}\ V\ p\ a \mid a.\ a \in A\} \in \{\textit{win-count}\ V\ p\ a \mid a.\ a \in A\}$
using *Max-in*
by *blast*
moreover have *only-one-max*:
finite $A \longrightarrow$
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\} =$
$\{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\}$
proof
assume *fin-A*: *finite* A
hence *finite* $\{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}$
by *simp*
hence
$\forall\ a \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ b \longrightarrow$
$\textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}$
using *CollectI Max-gr-iff empty-Collect-eq*
by (*metis* (*mono-tags*, *lifting*))
hence
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\}$
$\subseteq \{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\}$
by *blast*
moreover have
$\{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\}$
$\subseteq \{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\}$
using *fin-A no-winners-or-in-domain*
by *fastforce*

ultimately show
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\} =$
$\{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\}$
by *fastforce*
qed
ultimately have
finite $A \longrightarrow$
reject plurality $V\ A\ p = \{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p$
$x\}$
by *force*
moreover have
finite $A \longrightarrow$
defer plurality $V\ A\ p = \{a \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ p\ b \leq \textit{win-count}\ V\ p$
$a\}$
proof
assume *fin-A*: *finite A*
have $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\}$
$\cap\ \{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\} = \{\}$
by *force*
moreover have
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\}$
$\cup\ \{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\} = A$
by *force*
ultimately have
$\{a \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ p\ b \leq \textit{win-count}\ V\ p\ a\} =$
$A - \{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\}$
using *fin-A only-one-max Diff-cancel Int-Diff-Un Un-Diff inf-commute*
by (*metis* (*no-types*, *lifting*))
moreover have
$\{a \in A.\ \textit{win-count}\ V\ p\ a < \textit{Max}\ \{\textit{win-count}\ V\ p\ x \mid x.\ x \in A\}\} = A \longrightarrow$
$\{a \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ p\ b \leq \textit{win-count}\ V\ p\ a\} = A$
using *fin-A no-winners-or-in-domain*
by *fastforce*
ultimately show
defer plurality $V\ A\ p = \{a \in A.\ \forall\ b \in A.\ \textit{win-count}\ V\ p\ b \leq \textit{win-count}\ V\ p$
$a\}$
using *fin-A*
by *force*
qed
moreover have *elect plurality* $V\ A\ p = \{\}$
unfolding *max-eliminator.simps less-eliminator.simps elimination-module.simps*
by *force*
ultimately have
finite $A \longrightarrow$
plurality $V\ A\ p =$
$(\{\}, \{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\},$
$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\})$
using *split-pairs*
by (*metis* (*no-types*, *lifting*))

thus *plurality* $V\ A\ p =$
$(\textit{if finite}\ A$
$\textit{then}\ (\{\}, \{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\},$
$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\})$
$\textit{else}\ (\{\}, \{\}, A))$
by *force*
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.simps*, *safe*)
fix
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ \textit{Profile}$
have *disjoint3* (
$\{\},$
$\{a \in A.\ \exists\ a' \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ a'\},$
$\{a \in A.\ \forall\ a' \in A.\ \textit{win-count}\ V\ p\ a' \leq \textit{win-count}\ V\ p\ a\})$
by *auto*
moreover have
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\} \cup$
$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\} = A$
using *not-le-imp-less*
by *blast*
ultimately show *well-formed-*$\mathcal{SCF}$ A (*plurality′* $V\ A\ p$)
by *simp*
qed

lemma *voters-determine-plurality-score*: *voters-determine-evaluation plurality-score*
proof (*unfold plurality-score.simps voters-determine-evaluation.simps*, *safe*)
fix
$A :: 'b\ set$ **and**
$V :: 'a\ set$ **and**
$p\ p' :: ('b, 'a)\ \textit{Profile}$ **and**
$a :: 'b$
assume
$\forall\ v \in V.\ p\ v = p'\ v$ **and**
$a \in A$
hence *finite* $V \longrightarrow$
$\textit{card}\ \{v \in V.\ \textit{above}\ (p\ v)\ a = \{a\}\} = \textit{card}\ \{v \in V.\ \textit{above}\ (p'\ v)\ a = \{a\}\}$
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 *voters-determine-plurality*: *voters-determine-election plurality*
unfolding *plurality.simps*
using *voters-determine-max-elim voters-determine-plurality-score*
by *blast*

lemma *voters-determine-plurality′*: *voters-determine-election plurality′*
proof (*unfold voters-determine-election.simps*, *safe*)
fix
A :: ′*k set* **and**
V :: ′*v set* **and**
p p′ :: (′*k*, ′*v*) *Profile*
assume ∀ *v* ∈ *V*. *p v* = *p′ v*
thus *plurality′ V A p* = *plurality′ V A p′*
using *voters-determine-plurality plurality-mod-equiv*
unfolding *voters-determine-election.simps*
by (*metis* (*full-types*))
qed

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*

theorem *plurality′-mod-non-blocking*[*simp*]: *non-blocking plurality′*
using *plurality-mod-non-blocking plurality-mod-equiv plurality′-sound*
unfolding *non-blocking-def*
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*
using *plurality′-sound*
by *simp*

5.12.5 Property

lemma *plurality-def-inv-mono-alts*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p 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. $b \notin \{c\} \vee b = c$
    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 $a \leq$ *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*: $\forall\ b \in A - \{a\}.\ win\text{-}count\ V\ p\ b \geq win\text{-}count\ V\ q\ b$
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
$\forall\ b \in A - \{a\}.\ \forall\ i \in V.$
$(above\ (p\ i)\ b = \{b\} \longrightarrow (above\ (q\ i)\ b = \{b\} \vee above\ (q\ i)\ a = \{a\}))$
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 (*intro ballI impI*, *safe*)
fix
$b :: 'a$ **and**
$i :: 'v$
assume
$b \in A$ **and**
$i \in V$
moreover from *this* **have** *A-not-empty*: $A \neq \{\}$
by *blast*
ultimately have *linear-order-on* $A\ (p\ i)$
using *lift-a*
unfolding *lifted-def profile-def*
by *metis*
moreover assume
b-neq-a: $b \neq a$ **and**
abv-b: $above\ (p\ i)\ b = \{b\}$ **and**
abv-a: $above\ (p\ i)\ a = \{a\}$
ultimately show *False*
using *above-one-eq A-not-empty fin-A*
by (*metis* (*no-types*))
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 $b \in A$
thus $card\ \{i \in V.\ above\ (p\ i)\ b = \{b\}\} =$
$card\ \{i \in V.\ above\ (q\ i)\ b = \{b\}\}$
using *DiffI set-disj above-PtoQ above-QtoP above-pq*
by (*metis* (*no-types*, *lifting*))
qed
hence $\{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ p\ c \leq win\text{-}count\ V\ p\ b\} =$
$\{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ q\ c \leq win\text{-}count\ V\ q\ b\}$
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-equiv lift-a*
unfolding *Profile.lifted-def*
by (*metis* (*no-types*, *opaque-lifting*))
thus *?thesis*
by *simp*
next
assume $win\text{-}count\ V\ p\ a \neq win\text{-}count\ V\ q\ a$
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*: $A \neq \{\}$
using *lift-a equals0D equiv-prof-except-a-def*
lifted-imp-equiv-prof-except-a
by *metis*
moreover have *fin-A*: $finite\text{-}profile\ V\ A\ p$
using *lift-a*
unfolding *Profile.lifted-def*
by *simp*
ultimately have $a \in defer\ plurality'\ V\ A\ p$
using *plurality-mod-equiv*
by *metis*
hence *a-in-win-p*:
$a \in \{b \in A.\ \forall\ c \in A.\ win\text{-}count\ V\ p\ c \leq win\text{-}count\ V\ p\ b\}$
using *fin-A*
by *simp*

hence $\forall\ b \in A.$ *win-count* $V\ p\ b \leq$ *win-count* $V\ p\ a$
  by *simp*
hence *less*: $\forall\ b \in A - \{a\}.$ *win-count* $V\ q\ b <$ *win-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 $\forall\ b \in A - \{a\}.\ \neg\ (\forall\ c \in A.$ *win-count* $V\ q\ c \leq$ *win-count* $V\ q\ b)$
  using *lift-a not-le*
  unfolding *Profile.lifted-def*
  by *metis*
hence $\forall\ b \in A - \{a\}.$
      $b \notin \{c \in A.\ \forall\ b \in A.$ *win-count* $V\ q\ b \leq$ *win-count* $V\ q\ c\}$
  by *blast*
hence $\forall\ b \in A - \{a\}.\ b \notin$ *defer plurality'* $V\ A\ q$
  using *fin-A*
  by *simp*
hence $\forall\ b \in A - \{a\}.\ b \notin$ *defer plurality* $V\ A\ q$
  using *lift-a non-empty-A plurality-mod-equiv*
  unfolding *Profile.lifted-def*
  by (*metis* (*no-types*, *lifting*))
hence $\forall\ b \in A - \{a\}.\ b \notin$ *defer plurality* $V\ A\ q$
  by *simp*
moreover have $a \in$ *defer plurality* $V\ A\ q$
proof −
  have $\forall\ b \in A - \{a\}.$ *win-count* $V\ q\ b \leq$ *win-count* $V\ q\ a$
    using *less less-imp-le*
    by *metis*
  moreover have *win-count* $V\ q\ a \leq$ *win-count* $V\ q\ a$
    by *simp*
  ultimately have $\forall\ b \in A.$ *win-count* $V\ q\ b \leq$ *win-count* $V\ q\ a$
    by *auto*
  moreover have $a \in A$
    using *a-in-win-p*
    by *simp*
  ultimately have
    $a \in \{b \in A.\ \forall\ c \in A.$ *win-count* $V\ q\ c \leq$ *win-count* $V\ q\ b\}$
    by *simp*
  hence $a \in$ *defer plurality'* $V\ A\ q$
    by *simp*
  hence $a \in$ *defer plurality* $V\ A\ q$
    using *plurality-mod-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 *defer plurality* $V\ A\ q \subseteq A$
  by *simp*
ultimately show *?thesis*

by *blast*
qed
qed

lemma *plurality′-def-inv-mono-alts*:
fixes
A :: *′a set* **and**
V :: *′v set* **and**
p q :: (*′a*, *′v*) *Profile* **and**
a :: *′a*
assumes
$a \in$ *defer plurality′* V A p **and**
lifted V A p q a
shows *defer plurality′* V A q = *defer plurality′* V A p
$\vee$ *defer plurality′* V A $q = \{a\}$
using *assms plurality-def-inv-mono-alts plurality-mod-equiv*
by (*metis* (*no-types*))

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 $\mathcal{SCF}$*-result.electoral-module plurality*
using *plurality-sound*
by *metis*
next
show *non-electing plurality*
by *simp*
next
fix
A :: *′b set* **and**
V :: *′a set* **and**
p 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

theorem *plurality′-mod-def-inv-mono*[*simp*]: *defer-invariant-monotonicity plurality′*
using *plurality-mod-def-inv-mono plurality-mod-equiv*
plurality′-non-electing plurality′-sound
unfolding *defer-invariant-monotonicity-def*
by *metis*

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 $V\ x\ A\ p = (\sum\ y \in A.\ (\mathit{prefer\text{-}count}\ V\ p\ x\ y))$

fun *borda* :: $('a, 'v, 'a\ \mathit{Result})$ *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*: $\mathcal{SCF}$*-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**
$w\ l$ :: $'b$
assume
c-win: *condorcet-winner V A p w* **and**

l-neq-w: $l \neq w$
have $\neg$ *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* $\mathcal{SCF}$*-result.electoral-module.simps*, *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 condorcet-score V A p*)
using *max-elim-sound*
unfolding $\mathcal{SCF}$*-result.electoral-module.simps*
by *metis*
thus *well-formed-*$\mathcal{SCF}$ *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* $\in$ *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* $\in$ *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 {$y \in A$. *wins V x p y*} − *card* {$y \in A$. *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*: $\mathcal{SCF}$*-result.electoral-module copeland*
unfolding *copeland.simps*
using *max-elim-sound*
by *metis*

5.15.3 Lemmas

lemma *voters-determine-copeland-score*: *voters-determine-evaluation copeland-score*
proof (*unfold copeland-score.simps voters-determine-evaluation.simps*, *safe*)
fix
A :: 'b *set* **and**
V :: 'a *set* **and**
$p\ p'$ :: ('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 *voters-determine-copeland*: *voters-determine-election copeland*
unfolding *copeland.simps*
using *voters-determine-max-elim voters-determine-election.simps*
voters-determine-copeland-score
by *blast*

For a Condorcet winner w, we have: "$|\{y \in A\ .\ wins\ V\ w\ p\ y\}| = |A| - 1$".

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: "$|\{y \in A\ .\ wins\ V\ y\ p\ w\}| = 0$".

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* $\{a \in A.\ wins\ V\ a\ p\ w\} = 0$
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* = *card A* − *1*
proof (*unfold copeland-score.simps*)
have *card* $\{a \in A.\ wins\ V\ w\ p\ a\}$ = *card A* − *1*
using *cond-winner-imp-win-count assms*
by *metis*
moreover have *card* $\{a \in A.\ wins\ V\ a\ p\ w\} = 0$
using *cond-winner-imp-loss-count assms*
by (*metis* (*no-types*))

ultimately show
  enat (*card* $\{a \in A.\ wins\ V\ w\ p\ a\}$
    $-$ *card* $\{a \in A.\ wins\ V\ a\ p\ w\}) = enat\ (card\ A - 1)$
  by *simp*
qed

For a non-Condorcet winner l, we have: "$|\{y \in A\ .\ wins\ V\ l\ p\ y\}| = |A| - 2$".

lemma *non-cond-winner-imp-win-count*:
  fixes
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: *('a, 'v) Profile* **and**
    $w\ l$ :: *'a*
  assumes
    winner: *condorcet-winner* $V\ A\ p\ w$ **and**
    loser: $l \neq w$ **and**
    l-in-A: $l \in A$
  shows *card* $\{a \in A\ .\ wins\ V\ l\ p\ a\} \leq card\ A - 2$
proof $-$
  have *wins V w p l*
    using *assms*
    by *auto*
  hence $\neg$ *wins V l p w*
    using *wins-antisym*
    by *simp*
  moreover have $\neg$ *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

5.15.4 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**
$w\ l$ :: $'b$
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 diff-zero*
by (*metis* (*no-types*, *lifting*))
qed

theorem *copeland-is-dcc*: *defer-condorcet-consistency copeland*
proof (*unfold defer-condorcet-consistency-def* $\mathcal{SCF}$*-result.electoral-module.simps*, *safe*)
fix
A :: $'b$ *set* **and**
V :: $'a$ *set* **and**
p :: $('b, 'a)$ *Profile*
assume *profile* $V\ A\ p$
moreover from *this*
have *well-formed-*$\mathcal{SCF}$ A (*max-eliminator copeland-score* $V\ A\ p$)

```
    using max-elim-sound
    unfolding 𝒮𝒞ℱ-result.electoral-module.simps
    by metis
  ultimately show well-formed-𝒮𝒞ℱ A (copeland V A p)
    using copeland-sound
    unfolding 𝒮𝒞ℱ-result.electoral-module.simps
    by metis
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
    unfolding copeland.simps
    by safe
  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 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 (*unfold minimax-score.simps*, *intro Min-le*)
  have *finite V*
    using *winner*
    by *simp*
  moreover have ∀ *E n*. *infinite E* ⟶ (∃ *e*. ¬ *e* ≤ *enat n* ∧ *e* ∈ *E*)
    using *finite-enat-bounded*
    by *blast*
  ultimately show *finite* {*prefer-count V p l y* | *y*. *y* ∈ *A* − {*l*}}
    using *pref-count-voter-set-card*
    by *fastforce*
next
  have *w* ∈ *A*
    using *winner*
    by *simp*
  thus *prefer-count V p l w* ∈ {*prefer-count V p l y* | *y*. *y* ∈ *A* − {*l*}}
    using *l-neq-w*
    by *blast*
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**
    $w\ l$ :: $'b$
  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 :: $'b$ **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.simps*, *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 ∈ 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 ∈ 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.

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 $\mathcal{SCF}$*-result.electoral-module* (*drop-module 0 r*)
    using *assms drop-mod-sound*
    by *metis*
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 $\forall\ B\ a.\ B \neq \{\} \lor (a :: 'a) \notin B$
    by *simp*
  hence $\forall\ a\ B.\ a \in A \land a \in B \longrightarrow$ *connex B* (*limit A r*) $\longrightarrow$

¬ *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 *$\mathcal{SCF}$-result.electoral-module* (*drop-module n r*)
using *drop-mod-sound*
by *metis*
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 ∀ a ∈ A. card {a} ≤ card (above (limit A r) a)
  using card-mono fin-A rev-finite-subset above-presv-limit
  by metis
hence rank-geq-one: ∀ a ∈ A. 1 ≤ rank (limit A r) a
  by simp
with leq have ∀ a ∈ A. rank (limit A r) a ∈ {1 .. card A}
  by simp
hence rank (limit A r) ‘ A ⊆ {1 .. card A}
  by auto
moreover have inj: inj-on (rank (limit A r)) A
  using fin-A inj-onI rank-unique lin-ord-limit
  by metis
ultimately have bij-A: bij-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-betw ?inv-rank {1 .. card A} A
  using bij-betw-the-inv-into
  by blast
hence ∀ S ⊆ {1..card A}. card (?inv-rank ‘ S) = card S
  using fin-A bij-betw-same-card bij-betw-subset
  by metis
moreover have subset: {1 .. n} ⊆ {1 .. card A}
  using card-n
  by simp
ultimately have card (?inv-rank ‘ {1 .. n}) = n
  using numeral-One numeral-eq-iff card-atLeastAtMost diff-Suc-1
  by presburger
also have ?inv-rank ‘ {1..n} = {a ∈ A. rank (limit A r) a ∈ {1 .. n}}
proof
  show ?inv-rank ‘ {1..n} ⊆ {a ∈ A. rank (limit A r) a ∈ {1 .. n}}
  proof
    fix a :: 'a
    assume a ∈ ?inv-rank ‘ {1..n}
    then obtain b :: nat where
      b-img: b ∈ {1 .. n} ∧ ?inv-rank b = a
      by auto
    hence rank (limit A r) a = b
      using subset f-the-inv-into-f-bij-betw subsetD bij-A
      by metis
    hence rank (limit A r) a ∈ {1 .. n}
      using b-img
      by simp
    moreover have a ∈ A
      using b-img bij-inv bij-betwE subset
      by blast
    ultimately show a ∈ {a ∈ A. rank (limit A r) a ∈ {1 .. n}}
      by blast
```

```
    qed
  next
    show {a ∈ A. rank (limit A r) a ∈ {1 .. n}}
            ⊆ the-inv-into A (rank (limit A r)) ` {1 .. n}
    proof
      fix a :: 'a
      assume el: a ∈ {a ∈ A. rank (limit A r) a ∈ {1 .. n}}
      then obtain b :: nat where
        b-img: b ∈ {1..n} ∧ rank (limit A r) a = b
        by auto
      moreover have a ∈ A
        using el
        by simp
      ultimately have ?inv-rank b = a
        using inj the-inv-into-f-f
        by metis
      thus a ∈ ?inv-rank ` {1 .. n}
        using b-img
        by auto
    qed
  qed
  finally have card {a ∈ A. rank (limit A r) a ∈ {1 .. n}} = n
    by blast
  also have {a ∈ A. rank (limit A r) a ∈ {1 .. n}} =
              {a ∈ A. rank (limit A r) a ≤ n}
    using rank-geq-one
    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 SCF-result.electoral-module (drop-module n r)
    using assms drop-mod-sound
    by simp
next
  show SCF-result.electoral-module (pass-module n r)
    using assms pass-mod-sound
    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 ∃ B ⊆ A. (∀ a ∈ B. indep-of-alt (drop-module n r) V A a ∧
                      (∀ p. profile V A p ⟶ a ∈ reject (drop-module n r) V A p)) ∧
           (∀ a ∈ A − B. indep-of-alt (pass-module n r) V A a ∧
                      (∀ p. profile V A p ⟶ a ∈ reject (pass-module n r) V A p))
  proof
    have same-A:
      ∀ p q. (profile V A p ∧ profile V A q) ⟶
        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 ⊆ A
      by auto
    moreover have ∀ a ∈ ?A. indep-of-alt (drop-module n r) V A a
      using assms drop-mod-sound
      unfolding drop-module.simps indep-of-alt-def
      by (metis (mono-tags, lifting))
    moreover have
      ∀ a ∈ ?A. ∀ p. profile V A p
          ⟶ a ∈ reject (drop-module n r) V A p
      by auto
    moreover have ∀ a ∈ A − ?A. indep-of-alt (pass-module n r) V A a
      using assms pass-mod-sound
      unfolding pass-module.simps indep-of-alt-def
      by metis
    moreover have
      ∀ a ∈ A − ?A. ∀ p.
        profile V A p ⟶ a ∈ reject (pass-module n r) V A p
      by auto
    ultimately show ?A ⊆ A ∧
        (∀ a ∈ ?A. indep-of-alt (drop-module n r) V A a ∧
          (∀ p. profile V A p ⟶ a ∈ reject (drop-module n r) V A p)) ∧
        (∀ a ∈ A − ?A. indep-of-alt (pass-module n r) V A a ∧
          (∀ p. profile V A p ⟶ a ∈ 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 ⇒
('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 ⇒
('a, 'v, 'a Result) Electoral-Module (-↓ 50) **where**
m↓ ≡ *revision-composition m*

6.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
fixes *m* :: ('a, 'v, 'a Result) Electoral-Module
assumes *𝒮𝒞ℱ-result.electoral-module m*
shows *𝒮𝒞ℱ-result.electoral-module* (*revision-composition m*)
proof −
from *assms*
have ∀ A V p. *profile V A p* ⟶ *elect m V A p* ⊆ A
using *elect-in-alts*
by *metis*
hence ∀ A V p. *profile V A p* ⟶ (A − *elect m V A p*) ∪ *elect m V A p* = A
by *blast*
hence ∀ A V p. *profile V A p* ⟶
set-equals-partition A (*revision-composition m V A p*)
by *simp*
moreover have ∀ A V p. *profile V A p* ⟶ (A − *elect m V A p*) ∩ *elect m V A p* = {}
by *blast*
hence ∀ A V p. *profile V A p* ⟶ *disjoint3* (*revision-composition m V A p*)
by *simp*
ultimately show *?thesis*
by *simp*

qed

lemma *voters-determine-rev-comp*:
  fixes m :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes *voters-determine-election m*
  shows *voters-determine-election* (*revision-composition m*)
  using *assms*
  unfolding *voters-determine-election.simps 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* ($m\downarrow$)
  using *assms fstI rev-comp-sound revision-composition.simps*
  using *non-electing-def*
  by *metis*

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* ($m\downarrow$)
proof (*unfold non-blocking-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* ($m\downarrow$)
    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**
    reject-A: *reject* ($m\downarrow$) $V\ A\ p = A$ **and**
    x-in-A: $x \in A$
  hence *non-electing m*
    using *assms empty-iff Diff-disjoint Int-absorb2*
        elect-in-alts prod.collapse prod.inject
    unfolding *electing-def revision-composition.simps*
    by (*metis* (*no-types*, *lifting*))

thus $x \in \{\}$
  using *assms 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*↓)
proof (*unfold defer-invariant-monotonicity-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* (*m*↓)
    using *assms rev-comp-sound*
    unfolding *invariant-monotonicity-def*
    by *metis*
next
  show *non-electing* (*m*↓)
    using *assms rev-comp-non-electing*
    unfolding *invariant-monotonicity-def*
    by *simp*
next
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p q :: (*'a*, *'v*) *Profile* **and**
    a x x' :: *'a*
  assume
    rev-p-defer-a: $a \in$ *defer* (*m*↓) *V A p* **and**
    a-lifted: *lifted V A p q a* **and**
    rev-q-defer-x: $x \in$ *defer* (*m*↓) *V A q* **and**
    x-non-eq-a: $x \neq a$ **and**
    rev-q-defer-x': $x' \in$ *defer* (*m*↓) *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*↓) *V A p*
    using *rev-q-defer-x'*
    by *simp*
next

fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
$p\ q$ :: $('a, 'v)$ *Profile* **and**
$a\ x\ x'$ :: $'a$
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**
$p\ q$ :: $('a, 'v)$ *Profile* **and**
$a\ x\ x'$ :: $'a$
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**
$p\ q$ :: $('a, 'v)$ *Profile* **and**
$a\ x\ x'$ :: $'a$

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
moreover 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*))
moreover have *p-defer-rev-eq-elect*: *defer* $(m\downarrow)$ V A p $=$ *elect* m V A p
by *simp*
moreover have *defer* $(m\downarrow)$ V A q $=$ *elect* m V A q
by *simp*
ultimately show $x' \in$ *defer* $(m\downarrow)$ V A q
using *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* $\Rightarrow$
($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$
($'a$, $'v$, $'a$ *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) $\cup$ (*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*))

abbreviation *sequence* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* $\Rightarrow$

('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 *voters-determine-seq-comp*:
fixes *m n* :: ('a, 'v, 'a Result) Electoral-Module
assumes *voters-determine-election m* ∧ *voters-determine-election n*
shows *voters-determine-election* (*m* ▷ *n*)
proof (*unfold voters-determine-election.simps*, *clarify*)
fix
A :: 'a set **and**
V :: 'v set **and**
p 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 *voters-determine-election.simps*
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 *voters-determine-election.simps*
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 *voters-determine-election.simps*
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 voters-determine-election.simps
    by metis
  ultimately show (m ▷ n) V A p = (m ▷ n) V A p′
    unfolding sequential-composition.simps
    by metis
qed

lemma seq-comp-presv-disj:
  fixes
    m n :: (′a, ′v, ′a Result) Electoral-Module and
    A :: ′a set and
    V :: ′v set and
    p :: (′a, ′v) Profile
  assumes
    module-m: 𝒮𝒞ℱ-result.electoral-module m and
    module-n: 𝒮𝒞ℱ-result.electoral-module n and
    prof: profile V A p
  shows disjoint3 ((m ▷ 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′ ∧
       𝒮𝒞ℱ-result.electoral-module m′ ∧
       a ∈ defer m′ V′ A′ p′) ⟶
      a ∈ A′
    using UnCI result-presv-alts
    by (metis (mono-tags))
  from module-m prof
  have disjoint-m: disjoint3 (m V A p)
    unfolding 𝒮𝒞ℱ-result.electoral-module.simps well-formed-𝒮𝒞ℱ.simps
    by blast
  from module-m module-n def-presv-prof prof
  have disjoint-n: disjoint3 (n V ?new-A ?new-p)
    unfolding 𝒮𝒞ℱ-result.electoral-module.simps well-formed-𝒮𝒞ℱ.simps
    by metis
  have disj-n:
    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 = {}
    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 (*metis* (*no-types*, *opaque-lifting*))
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* $\cup$ *elect n V ?new-A ?new-p*) $\cap$
(*reject m V A p* $\cup$ *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* $\cap$ *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 $\cup$ *elect* n V *?new-A ?new-p*)
$\cap$ (*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 $\cup$ *reject* n V *?new-A ?new-p*)
$\cap$ (*defer* n V *?new-A ?new-p*) = $\{\}$
proof (*safe*)
fix x :: $'a$
assume $x \in$ *defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p)
hence $x \in$ *defer* m V A p
using *defer-in-A module-n prof-def-lim prof*
by *metis*
moreover assume $x \in$ *reject* m V A p
ultimately 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 ∈ defer n V (defer m V A p) (limit-profile (defer m V A p) p) and
      x ∈ reject n V (defer m V A p) (limit-profile (defer m V A p) p)
    thus x ∈ {}
      using module-n prof-def-lim reject-not-elected-or-deferred
      by blast
  qed
  ultimately have
    disjoint3 (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-presv-alts:
  fixes
    m n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    module-m: 𝒮𝒞ℱ-result.electoral-module m and
    module-n: 𝒮𝒞ℱ-result.electoral-module n and
    prof: profile V A p
  shows set-equals-partition A ((m ▷ 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 ∪ reject m V A p ∪ ?new-A = A
    using module-m prof
    by (simp add: result-presv-alts)
  have elect n V ?new-A ?new-p ∪
         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*[*fundef-cong*, *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') ⇒
      ($e \cup e'$, $r \cup r'$, 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*
      ($e \cup e'$, $r \cup r'$, d'))
    by *metis*
qed

6.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
  fixes m 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.simps*, *safe*)
  fix
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile*
  assume *profile* V A p

moreover have $\forall$ *r. well-formed-$\mathcal{SCF}$* (*A* :: *'a set*) *r* =
(*disjoint3 r* $\wedge$ *set-equals-partition A r*)
by *simp*
ultimately show *well-formed-$\mathcal{SCF}$ A* ((*m* $\triangleright$ *n*) *V A p*)
using *assms seq-comp-presv-disj seq-comp-presv-alts*
by *metis*
qed

6.3.3 Lemmas

lemma *seq-comp-decrease-only-defer*:
fixes
m 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 - - - limit-profile* (*defer m V A p*) *p*]
empty-defer module-n prof prof-no-alt
by *auto*
thus *elect* (*m* $\triangleright$ *n*) *V A p* = *elect m V A p*
using *fst-conv*
unfolding *sequential-composition.simps*
by *metis*
next
have *rej-empty*:
$\forall$ *m' V' p'*.
(*$\mathcal{SCF}$-result.electoral-module m'*
$\wedge$ *profile V'* {} *p'*) $\longrightarrow$ *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)
      unfolding sequential-composition.simps
      using rej-empty empty-defer module-n prof-no-alt prof sndI sup-bot-right
      by metis
  qed
qed

lemma seq-comp-def-then-elect:
  fixes
    m 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: ∃ a ∈ 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: ∃ a ∈ A. reject m V A p = A − {a}
    using Diff-empty def-one-m f-prof reject-not-elected-or-deferred
```

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-elected-or-deferred*
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
$m\ n :: ('a,\ 'v,\ 'a\ Result)\ Electoral\text{-}Module$ **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
$m\ n :: ('a,\ 'v,\ 'a\ Result)\ Electoral\text{-}Module$ **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 n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  shows defer (m ▷ 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 n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  shows elect (m ▷ n) V A p =
            elect n V (defer m V A p)
              (limit-profile (defer m V A p) p) ∪ (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 n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    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 ▷ n) V A p ⊂ 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 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 $a \in defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p) \wedge$
$a \in defer\ m\ V\ A\ p$
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 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$
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 $?input\text{-}sound = A \neq \{\} \wedge finite\text{-}profile\ V\ A\ p$
from *non-blocking-m*
have $?input\text{-}sound \longrightarrow reject\ m\ V\ A\ p \neq A$
unfolding *non-blocking-def*
by *simp*
with *non-blocking-m*
have *A-reject-diff*: $?input\text{-}sound \longrightarrow A - reject\ m\ V\ A\ p \neq \{\}$
using *Diff-eq-empty-iff reject-in-alts subset-antisym*
unfolding *non-blocking-def*
by *metis*
from *non-blocking-m*
have $?input\text{-}sound \longrightarrow well\text{-}formed\text{-}\mathcal{SCF}\ A\ (m\ V\ A\ p)$
unfolding $\mathcal{SCF}$*-result.electoral-module.simps non-blocking-def*
by *simp*
hence $?input\text{-}sound \longrightarrow elect\ m\ V\ A\ p \cup defer\ m\ V\ A\ p = A - reject\ m\ V\ A\ p$
using *non-blocking-m elec-and-def-not-rej*
unfolding *non-blocking-def*
by *metis*
with *A-reject-diff*
have $?input\text{-}sound \longrightarrow elect\ m\ V\ A\ p \cup defer\ m\ V\ A\ p \neq \{\}$
by *simp*
hence $?input\text{-}sound \longrightarrow (elect\ m\ V\ A\ p \neq \{\} \vee defer\ m\ V\ A\ p \neq \{\})$
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 seq-comp-sound*
      by *metis*
  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-elected-or-deferred*)
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-elected-or-deferred x-in-A
by *metis*
qed
qed
qed

Sequential composition preserves the non-electing property.

theorem *seq-comp-presv-non-electing*[*simp*]:
fixes *m n* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
non-electing m **and**
non-electing n
shows *non-electing* $(m \triangleright n)$
proof (*unfold non-electing-def*, *safe*)
have $\mathcal{SCF}$*-result.electoral-module m* $\wedge$ $\mathcal{SCF}$*-result.electoral-module n*
using *assms*
unfolding *non-electing-def*
by *blast*
thus $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
using *seq-comp-sound*
by *metis*
next

fix
  A :: $'a$ *set* **and**
  V :: $'v$ *set* **and**
  p :: $('a, 'v)$ *Profile* **and**
  x :: $'a$
assume
  profile V A p **and**
  $x \in$ *elect* $(m \triangleright n)$ V A p
thus $x \in \{\}$
  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 n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
  assumes
    def-one-m: *defers 1 m* **and**
    electing-n: *electing n*
  shows *electing* $(m \triangleright n)$
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-m-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-m-not-empty bot-eq-sup-iff finite-subset
unfolding *electing-def*
by *metis*
qed
qed

lemma *def-lift-inv-seq-comp-help*:
fixes
m n :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p q :: ('a, 'v) *Profile* **and**
a :: 'a

assumes
  monotone-m: *defer-lift-invariance m* **and**
  monotone-n: *defer-lift-invariance n* **and**
  voters-determine-n: *voters-determine-election 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* ∧ $\mathcal{SCF}$*-result.electoral-module n*
    unfolding *defer-lift-invariance-def*
    by *simp*
  hence *profile V A p* ⟶ *defer* $(m \triangleright n)$ *V A p* ⊆ *defer m V A p*
    using *seq-comp-def-set-bounded*
    by *metis*
  moreover have *profile-p*: *lifted V A p q a* ⟶ *finite-profile V A p*
    unfolding *lifted-def*
    by *simp*
  ultimately have *defer-subset*: *defer* $(m \triangleright n)$ *V A p* ⊆ *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*:
($\exists$ $v \in V$.
Preference-Relation.lifted ?new-Ap (*?new-p v*) (*?new-q v*) *a*) $\longrightarrow$
($\exists$ $v \in V$.
$\neg$ *Preference-Relation.lifted ?new-Ap* (*?new-p v*) (*?new-q v*) *a* $\wedge$
(*?new-p v*) $\neq$ (*?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 $\forall$ $v \in V$. (*Preference-Relation.lifted A* (*p v*) (*q v*) *a* $\vee$ (*p v*) = (*q v*))
unfolding *Profile.lifted-def*
by *metis*
with *def-and-lifted modules mono-m*
have $\forall$ $v \in V$.
(*Preference-Relation.lifted ?new-Ap* (*?new-p v*) (*?new-q v*) *a* $\vee$
(*?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 $\forall$ $v \in V$. (*?new-p v*) = (*?new-q v*)
by *blast*
with *mono-m*
show *?thesis*
using *leI not-less-zero nth-equalityI voters-determine-n*
unfolding *voters-determine-election.simps*
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 n* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes

defer-lift-invariance m **and**
defer-lift-invariance n **and**
voters-determine-election n
shows *defer-lift-invariance* ($m \triangleright n$)
proof (*unfold defer-lift-invariance-def*, *safe*)
show $\mathcal{SCF}$*-result.electoral-module* ($m \triangleright n$)
using *assms seq-comp-sound*
unfolding *defer-lift-invariance-def*
by *blast*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
p q :: (*'a*, *'v*) *Profile* **and**
a :: *'a*
assume
$a \in$ *defer* ($m \triangleright n$) V A p **and**
Profile.lifted V A p q a
thus ($m \triangleright n$) V A p = ($m \triangleright n$) 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 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* ($m \triangleright n$)
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* ($m \triangleright n$)
using *seq-comp-sound*
by *metis*
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) ≥ 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' =
($\mathcal{SCF}$-*result.electoral-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 m' n :: ('a, 'v, 'a Result) Electoral-Module
  assumes
    compatible: disjoint-compatibility m n and
    module-m': SCF-result.electoral-module m' and
    voters-determine-m': voters-determine-election m'
  shows disjoint-compatibility (m ▷ m') n
proof (unfold disjoint-compatibility-def, safe)
  show SCF-result.electoral-module (m ▷ m')
    using compatible module-m' seq-comp-sound
    unfolding disjoint-compatibility-def
    by metis
next
  show SCF-result.electoral-module n
    using compatible
    unfolding disjoint-compatibility-def
    by metis
next
  fix
    S :: 'a set and
    V :: 'v set
  have modules:
    SCF-result.electoral-module (m ▷ m') ∧ 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 ⊆ S ∧
      (∀ a ∈ A.
        indep-of-alt m V S a ∧ (∀ p. profile V S p ⟶ a ∈ reject m V S p)) ∧
      (∀ a ∈ S − A.
        indep-of-alt n V S a ∧ (∀ p. profile V S p ⟶ a ∈ reject n V S p))
    using compatible
    unfolding disjoint-compatibility-def
    by (metis (no-types, lifting))
  show
    ∃ A ⊆ S.
      (∀ a ∈ A. indep-of-alt (m ▷ m') V S a ∧
        (∀ p. profile V S p ⟶ a ∈ reject (m ▷ m') V S p)) ∧
      (∀ a ∈ S − A.
        indep-of-alt n V S a ∧ (∀ p. profile V S p ⟶ a ∈ reject n V S p))
  proof
    have ∀ a p q. a ∈ A ∧ equiv-prof-except-a V S p q a ⟶
            (m ▷ m') V S p = (m ▷ m') V S q
    proof (safe)
```

```
    fix
      a :: 'a and
      p q :: ('a, 'v) Profile
    assume
      a-in-A: a ∈ A and
      lifting-equiv-p-q: equiv-prof-except-a V S p q a
    hence eq-defer: defer m V S p = defer m V S q
      using rej-A
      unfolding indep-of-alt-def
      by metis
    have profiles: profile V S p ∧ profile V S q
      using lifting-equiv-p-q
      unfolding equiv-prof-except-a-def
      by simp
    hence (defer m V S p) ⊆ S
      using compatible defer-in-alts
      unfolding disjoint-compatibility-def
      by metis
    moreover have a ∉ defer m V S q
      using a-in-A compatible profiles rej-A IntI emptyE result-disj
      unfolding disjoint-compatibility-def
      by metis
    ultimately have
      ∀ v ∈ 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 - S]
      unfolding eq-defer limit-profile.simps
      by blast
    hence 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 eq-defer voters-determine-m'
      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 ▷ m') V S p = (m ▷ m') V S q
      unfolding sequential-composition.simps
      by (metis (full-types))
  qed
  moreover have ∀ a' ∈ A. ∀ p'. profile V S p' ⟶ a' ∈ reject (m ▷ m') V S p'
    using rej-A UnI1 prod.sel
    unfolding sequential-composition.simps
    by metis
  ultimately show A ⊆ S ∧
      (∀ a' ∈ A. indep-of-alt (m ▷ m') V S a' ∧
        (∀ p'. profile V S p' ⟶ a' ∈ reject (m ▷ m') V S p')) ∧
      (∀ a' ∈ S − A. indep-of-alt n V S a' ∧
        (∀ p'. profile V S p' ⟶ a' ∈ 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 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* ($m \triangleright n$)
using *seq-comp-sound*
by *metis*
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* $(m \triangleright n)$
  using *seq-comp-sound*
  by *metis*
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*
  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 fst-conv*
  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-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
    hence seq-snd-simplified:
      snd ((m ▷ n) V A p) =
        (reject m V A p
        ∪ 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 sequential-composition.simps
      by metis
    hence seq-rej-union-eq-rej:
      reject m V A p
      ∪ reject n V (defer m V A p) (limit-profile (defer m V A p) p) =
          reject (m ▷ n) V A p
      by simp
    hence seq-rej-union-subset-A:
      reject m V A p
      ∪ reject n V (defer m V A p) (limit-profile (defer m V A p) p) ⊆ A
      using sound-seq-m-n cond-win reject-in-alts
      by (metis (no-types))
    hence A − {a} = reject (m ▷ 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 ▷ n) V A p ⊆ 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-elected-or-deferred 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' :: 'a
    assume
      cw-a: condorcet-winner V A p a and
      not-cw-a': ¬ condorcet-winner V A p a' and
      a'-in-elect-seq-m-n: a' ∈ elect (m ▷ n) V A p
    hence ∃ a''. defer-condorcet-consistency m ∧ 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* $(m \triangleright n)$
  using *seq-comp-sound*
  by *metis*
have *reject m V A p* $= A - \{a\}$
  using *cw-a dcc-m prod.sel 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 subset-iff*
    elect-in-alts singleton-iff sound-seq-m-n
  unfolding *condorcet-winner.simps*
  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*
    ∪ *elect 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*),
    defer 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*),
        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*
  ∪ *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-elected-or-deferred sound-m
  by (*metis* (*no-types*))
have *m-seq-n-full*:
  $(m \triangleright n)$ *V A p* =
    (*elect m V A p*
    ∪ *elect 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*),

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′ :: *′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*
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
      ∪ elect 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),
      defer 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),
        defer n V (defer m V A p) (limit-profile (defer m V A p) p))
    using snd-conv
    by metis
  ultimately show a' ∈ reject (m ▷ 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 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 ▷ n)
proof (unfold defer-condorcet-consistency-def, safe)
  have 𝒮𝒞ℱ-result.electoral-module m
    using dcc-m
    unfolding defer-condorcet-consistency-def
    by metis
  thus 𝒮𝒞ℱ-result.electoral-module (m ▷ n)
    using ne-n seq-comp-sound
    unfolding non-electing-def
    by metis
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 ∃ a'. defer-condorcet-consistency m ∧ 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 seq-comp-sound*
  unfolding *non-electing-def*
  by *metis*
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* $m\ V\ A\ p, \{a \in A.$ *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* $(m \triangleright n)\ V\ A\ p = \{a\}$
    using *cw-a* a'*-in-def-seq-m-n empty-iff sound-m nb-n*
      seq-comp-def-set-bounded subset-singletonD
    unfolding *condorcet-winner.simps non-blocking-def*
    by *metis*
  thus $a' = a$
    using a'*-in-def-seq-m-n*
    by *blast*
next
  have $\exists\ a'.$ *defer-condorcet-consistency* $m \land$ *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* $(m \triangleright n)$ V A $p = \{\}$
using *elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral*
by (*metis* (*no-types*))
moreover have *condorcet-compatibility* $(m \triangleright n)$
using *dcc-m nb-n ne-n*
by *simp*
hence $a \notin$ *reject* $(m \triangleright n)$ V A p
unfolding *condorcet-compatibility-def*
using *cw-a*
by *metis*
ultimately show $a \in$ *defer* $(m \triangleright n)$ 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-elected-or-deferred sound-seq-m-n condorcet-winner.simps*
by (*metis* (*no-types*))
moreover have $\{a' \in A.$ *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, $\{a' \in A.$ *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 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 ▷ n)
proof (unfold monotonicity-def, safe)
  have 𝒮𝒞ℱ-result.electoral-module m
    using non-ele-m
    unfolding non-electing-def
    by simp
  moreover have 𝒮𝒞ℱ-result.electoral-module n
    using electing-n
    unfolding electing-def
    by simp
  ultimately show 𝒮𝒞ℱ-result.electoral-module (m ▷ n)
    using seq-comp-sound
    by metis
next
  fix
    A :: 'a set and
    V :: 'v set and
    p 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 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
    voters-determine-n: voters-determine-election 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)
    using seq-comp-sound
    by metis
next
  fix
    A :: 'a set and
    V :: 'v set and
    p q :: ('a, 'v) Profile and
    a :: 'a
  assume
    defer-a-p: a ∈ defer (m ▷ 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: SCF-result.electoral-module m
    using strong-def-mon-m
    unfolding defer-invariant-monotonicity-def
    by metis
  have electoral-mod-n: 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 ≤ 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 ▷ 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 $(a \in \mathit{defer}\ m\ V\ A\ p) \longrightarrow \mathit{card}\ (\mathit{defer}\ (m \triangleright n)\ V\ A\ q) = \mathit{1}$
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 $a \in \mathit{defer}\ m\ V\ A\ p$
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 $\mathit{defer}\ (m \triangleright n)\ V\ A\ q = \{a\}$
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 $\mathit{defer}\ (m \triangleright n)\ V\ A\ p = \mathit{defer}\ (m \triangleright n)\ V\ A\ q$
using *def-seq-m-n-eq-a*
by *presburger*
moreover have $\mathit{elect}\ (m \triangleright n)\ V\ A\ p = \mathit{elect}\ (m \triangleright n)\ V\ A\ q$
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 $\neg\ (\mathit{defer}\ m\ V\ A\ q \neq \mathit{defer}\ m\ V\ A\ p)$
hence *def-eq*: $\mathit{defer}\ m\ V\ A\ q = \mathit{defer}\ m\ V\ A\ p$
by *presburger*
have $\mathit{elect}\ m\ V\ A\ p = \{\}$
using *finite-profile-p non-electing-m*
unfolding *non-electing-def*
by *simp*
moreover have $\mathit{elect}\ m\ V\ A\ q = \{\}$
using *finite-profile-q non-electing-m*
unfolding *non-electing-def*
by *simp*
ultimately have *elect-m-equal*:
$\mathit{elect}\ m\ V\ A\ p = \mathit{elect}\ m\ V\ A\ q$
by *simp*
have $(\forall\ v \in V.\ (\mathit{limit\text{-}profile}\ (\mathit{defer}\ m\ V\ A\ p)\ p)\ v = (\mathit{limit\text{-}profile}\ (\mathit{defer}\ m\ V\ A\ p)\ q)\ v)$
$\vee\ \mathit{lifted}\ V\ (\mathit{defer}\ m\ V\ A\ q)\ (\mathit{limit\text{-}profile}\ (\mathit{defer}\ m\ V\ A\ p)\ p)\ (\mathit{limit\text{-}profile}\ (\mathit{defer}\ m\ V\ A\ p)\ q)\ a$
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 *voters-determine-n def-eq*
unfolding *voters-determine-election.simps*
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 (*intro impI*)
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* (*no-types*, *lifting*))
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* ⇒
('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*a Aggregator* ⇒
('*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* ⇒ '*a Aggregator* ⇒
('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module*
(- ‖- - [*50*, *1000*, *51*] *50*) **where**
$m \parallel_a n \equiv$ *parallel-composition* $m\ n\ a$

6.4.2 Soundness

theorem *par-comp-sound*[*simp*]:

fixes
  $m\ n$ :: ($'a$, $'v$, $'a$ *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* $(m \parallel_a n)$
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module.simps*, *safe*)
fix
  A :: $'a$ *set* **and**
  V :: $'v$ *set* **and**
  p :: ($'a$, $'v$) *Profile*
assume *profile* $V\ A\ p$
moreover have
  $\forall\ a'.$ *aggregator* $a' =$
    $(\forall\ A'\ e\ r\ d\ e'\ r'\ d'.$
      (*well-formed-*$\mathcal{SCF}$ $(A'$ :: $'a$ *set*$)$ $(e,\ r',\ d)$
      $\wedge$ *well-formed-*$\mathcal{SCF}$ A' $(r,\ d',\ e'))$
        $\longrightarrow$ *well-formed-*$\mathcal{SCF}$ A' $(a'\ A'\ (e,\ r',\ d)\ (r,\ d',\ e')))$
  unfolding *aggregator-def*
  by *blast*
moreover have
  $\forall\ m'\ V'\ A'\ p'.$
    ($\mathcal{SCF}$*-result.electoral-module* m' $\wedge$ *finite* $(A'$ :: $'a$ *set*$)$
      $\wedge$ *finite* $(V'$ :: $'v$ *set*$)$ $\wedge$ *profile* $V'\ A'\ p')$
    $\longrightarrow$ *well-formed-*$\mathcal{SCF}$ A' $(m'\ V'\ A'\ p')$
  using *par-comp-result-sound*
  by (*metis* (*no-types*))
ultimately have *well-formed-*$\mathcal{SCF}$ 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-*$\mathcal{SCF}$ A $((m \parallel_a n)\ V\ A\ p)$
  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
  $m\ n$ :: ($'a$, $'v$, $'a$ *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* $(m \parallel_a n)$

proof (*unfold non-electing-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 *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 $\forall\ r\ r'\ d\ d'\ e\ e'\ A'\ f.$
        $((well\text{-}formed\text{-}\mathcal{SCF}\ (A' :: {'a}\ set)\ (e', r', d') \land$
          $well\text{-}formed\text{-}\mathcal{SCF}\ A'\ (e, r, d)) \longrightarrow$
          $elect\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e \land$
            $reject\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r \land$
            $defer\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d) =$
              $((well\text{-}formed\text{-}\mathcal{SCF}\ A'\ (e', r', d') \land$
                $well\text{-}formed\text{-}\mathcal{SCF}\ A'\ (e, r, d)) \longrightarrow$
                $elect\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e \land$
                  $reject\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r \land$
                  $defer\text{-}r\ (f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d)$
    by *linarith*
  hence $\forall\ a'.\ agg\text{-}conservative\ a' =$
        $(aggregator\ a' \land$
          $(\forall\ A'\ e\ e'\ d\ d'\ r\ r'.$
            $(well\text{-}formed\text{-}\mathcal{SCF}\ (A' :: {'a}\ set)\ (e, r, d) \land$

$\mathit{well\text{-}formed\text{-}}\mathcal{SCF}\ A'\ (e', r', d')) \longrightarrow$
$\mathit{elect\text{-}r}\ (a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e' \wedge$
$\mathit{reject\text{-}r}\ (a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq r \cup r' \wedge$
$\mathit{defer\text{-}r}\ (a'\ A'\ (e, r, d)\ (e', r', d')) \subseteq d \cup d'))$
unfolding *agg-conservative-def*
by *simp*
hence $\mathit{aggregator}\ a \wedge$
$(\forall\ A'\ e\ e'\ d\ d'\ r\ r'.$
$(\mathit{well\text{-}formed\text{-}}\mathcal{SCF}\ A'\ (e, r, d) \wedge$
$\mathit{well\text{-}formed\text{-}}\mathcal{SCF}\ A'\ (e', r', d')) \longrightarrow$
$\mathit{elect\text{-}r}\ (a\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e' \wedge$
$\mathit{reject\text{-}r}\ (a\ A'\ (e, r, d)\ (e', r', d')) \subseteq r \cup r' \wedge$
$\mathit{defer\text{-}r}\ (a\ A'\ (e, r, d)\ (e', r', d')) \subseteq d \cup d')$
using *conservative*
by *presburger*
hence $\mathit{let}\ c = (a\ A\ (m\ V\ A\ p)\ (n\ V\ A\ p))\ \mathit{in}$
$(\mathit{elect\text{-}r}\ c \subseteq ((\mathit{elect}\ m\ V\ A\ p) \cup (\mathit{elect}\ n\ V\ A\ p)))$
using *emod-m emod-n par-comp-result-sound*
prod.collapse prof-A
by *metis*
hence $w \in ((\mathit{elect}\ m\ V\ A\ p) \cup (\mathit{elect}\ n\ V\ A\ p))$
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 acc :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
    t :: $'a$ *Termination-Condition* **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**
  loop-comp-helper-finite:
  $finite\ (defer\ acc\ V\ A\ p) \wedge (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
    $\longrightarrow t\ (acc\ V\ A\ p) \Longrightarrow$
  $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = acc\ V\ A\ p\ |$
  loop-comp-helper-infinite:
  $\neg\ (finite\ (defer\ acc\ V\ A\ p) \wedge (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
    $\longrightarrow t\ (acc\ V\ A\ p)) \Longrightarrow$
  $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ p$
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
    $\bigwedge\ acc\ V\ A\ p\ m\ t.$
      $finite\ (defer\ acc\ V\ A\ p) \wedge defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$
        $\longrightarrow t\ (acc\ V\ A\ p) \Longrightarrow accum = (acc, m, t, V, A, p) \Longrightarrow P$ **and**
    $\bigwedge\ acc\ V\ A\ p\ m\ t.$
      $\neg\ (finite\ (defer\ acc\ V\ A\ p) \wedge defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$

$\longrightarrow t\ (acc\ V\ A\ p)) \Longrightarrow accum = (acc,\ m,\ t,\ V,\ A,\ p) \Longrightarrow P$

thus *P*
using *accum-exists*
by *metis*
next
fix
$t\ t' :: 'a$ *Termination-Condition* **and**
$acc\ acc' :: ('a,\ 'v,\ 'a\ Result)$ *Electoral-Module* **and**
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a,\ 'v)$ *Profile* **and**
$m\ m' :: ('a,\ 'v,\ 'a\ Result)$ *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
$t\ t' :: 'a$ *Termination-Condition* **and**
$acc\ acc' :: ('a,\ 'v,\ 'a\ Result)$ *Electoral-Module* **and**
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a,\ 'v)$ *Profile* **and**
$m\ m' :: ('a,\ 'v,\ 'a\ Result)$ *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 $acc\ V\ A\ p = loop\text{-}comp\text{-}helper\text{-}sumC\ (acc' \triangleright m',\ m',\ t',\ V',\ A',\ p')$
by *force*
next
fix
$t\ t' :: 'a$ *Termination-Condition* **and**
$acc\ acc' :: ('a,\ 'v,\ 'a\ Result)$ *Electoral-Module* **and**
$A\ A' :: 'a\ set$ **and**
$V\ V' :: 'v\ set$ **and**
$p\ p' :: ('a,\ 'v)$ *Profile* **and**
$m\ m' :: ('a,\ 'v,\ 'a\ Result)$ *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 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-Module*
$\times$ (*′b*, *′a*, *′b Result*) *Electoral-Module*
$\times$ (*′b Termination-Condition*) $\times$ *′a set* $\times$ *′b set*
$\times$ (*′b*, *′a*) *Profile*)
$\times$ (*′b*, *′a*, *′b Result*) *Electoral-Module*
$\times$ (*′b*, *′a*, *′b Result*) *Electoral-Module*
$\times$ (*′b Termination-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-Module* $\times$ (*′b*, *′a*, *′b Result*) *Electoral-Module*
$\times$ *′b Termination-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 acc :: ('*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-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 acc :: (*'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**
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 ∀ *m' n'*.
(*$\mathcal{SCF}$-result.electoral-module m'* ∧ *$\mathcal{SCF}$-result.electoral-module n'*)
⟶ *$\mathcal{SCF}$-result.electoral-module* (*m'* ▷ *n'*)
using *seq-comp-sound*
by *metis*
hence *$\mathcal{SCF}$-result.electoral-module* (*acc* ▷ *m*)
using *less.prems module-m*
by *blast*
hence ¬ *t* (*acc V A p*) ∧ *defer* (*acc* ▷ *m*) *V A p* ⊂ *defer acc V A p* ∧
finite (*defer acc V A p*) ⟶
well-formed-$\mathcal{SCF}$ A (*loop-comp-helper acc m t V A p*)
using *less.hyps less.prems loop-comp-helper-infinite*
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.simps*
by *metis*
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.simps*
by *metis*

lemma *loop-comp-helper-imp-no-def-incr*:
fixes
m acc :: (*'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**
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-comp-helper acc m t*) *V A p* ⊆ *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* ▷ *m*)
using *less.prems module-m seq-comp-sound*
by *blast*
have ∀ *A A'*. (*finite A* ∧ *A'* ⊂ *A*) ⟶ *card A'* < *card A*
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*) ⟶
defer (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *V A p* ⊆ *defer acc V A p*
using *emod-acc-m less.hyps less.prems*
by *blast*
hence ¬ *t* (*acc V A p*) ∧ *defer* (*acc* ▷ *m*) *V A p* ⊂ *defer acc V A p* ∧
finite (*defer acc V A p*) ⟶
defer (*loop-comp-helper acc m t*) *V A p* ⊆ *defer acc V A p*
using *loop-comp-helper-infinite*
by (*metis* (*no-types*))
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 acc :: (*'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**
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**
voters-determine-m: *voters-determine-election m*
shows
∀ *q a*. *a* ∈ (*defer* (*loop-comp-helper acc m t*) *V A p*) ∧ *lifted V A p q a* ⟶
(*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 ⟶
(∀ *q a*. *a* ∈ (*defer* (*acc* ⊳ *m*) *V A p*) ∧ *lifted V A p q a* ⟶
card (*defer* (*acc* ⊳ *m*) *V A p*) = *card* (*defer* (*acc* ⊳ *m*) *V A q*))
using *monotone-m def-lift-inv-seq-comp-help voters-determine-m*
by *metis*
have *defer-lift-invariance acc* ⟶
(∀ *q a*. *a* ∈ (*defer acc V A p*) ∧ *lifted V A p q a* ⟶
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 ⟶
(∀ *q a*. (*a* ∈ (*defer* (*acc* ⊳ *m*) *V A p*) ∧ *lifted V A p q a*) ⟶
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* ⊳ *m*) *V A p*) = *card* (*defer acc V A p*)
have *defer-lift-invariance acc* ⟶
(∀ *q a*. *a* ∈ (*defer acc V A p*) ∧ *lifted V A p q a* ⟶
(*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 *SCF-result.electoral-module m*
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have *emod-acc*: *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* ▷ *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* ▷ *m*)
using *less monotone-m seq-comp-presv-def-lift-inv*
by *safe*
ultimately show *?thesis*
using *less monotone-m*
by *metis*
next
assume *card-changed*:
$\neg$ (*card* (*defer* (*acc* ▷ *m*) *V A p*) = *card* (*defer acc V A p*))
with *prof*
have *card-smaller-for-p*:
SCF-result.electoral-module acc $\wedge$ *finite A* $\longrightarrow$

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-defer*:
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′* ∧ $\mathcal{SCF}$*-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′*
⟶ $\mathcal{SCF}$*-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*
⟶ (∀ *p′ a. a* ∈ (*defer* (*acc* ▷ *m*) *V A p*) ∧ *lifted V A p p′ a*
⟶ *defer* (*acc* ▷ *m*) *V A p′* ⊂ *defer acc V A p′*)

  using *Profile.lifted-def dli-card-defer 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*:
  defer-lift-invariance (*acc* ▷ *m*) ∧ *defer-lift-invariance acc*
    ⟶ (∀ *q a*. *a* ∈ (*defer* (*acc* ▷ *m*) *V A p*) ∧ *lifted V A p q a*
      ⟶ *loop-comp-helper acc m t V A q* =
          loop-comp-helper (*acc* ▷ *m*) *m t V A q*)
proof (*safe*)
  fix
    q :: (′*a*, ′*v*) *Profile* **and**
    a :: ′*a*
  assume
    a-in-def-imp-def-subset:
    ∀ *q*′ *a*′. *a*′ ∈ *defer* (*acc* ▷ *m*) *V A p* ∧ *lifted V A p q*′ *a*′ ⟶
      defer (*acc* ▷ *m*) *V A q*′ ⊂ *defer acc V A q*′ **and**
    dli-acc: *defer-lift-invariance acc* **and**
    a-in-def-seq-acc-m: *a* ∈ *defer* (*acc* ▷ *m*) *V A p* **and**
    lifted-pq-a: *lifted V A p q a*
  hence *defer* (*acc* ▷ *m*) *V A q* ⊂ *defer acc V A q*
    by *metis*
  moreover have *𝒮𝒞ℱ-result.electoral-module acc*
    using *dli-acc*
    unfolding *defer-lift-invariance-def*
    by *simp*
  moreover have ¬ *t* (*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 *loop-comp-helper acc m t V A q*
              = *loop-comp-helper* (*acc* ▷ *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*:
  𝒮𝒞ℱ-result.electoral-module acc ⟶
    loop-comp-helper acc m t V A p = *loop-comp-helper* (*acc* ▷ *m*) *m t V A p*
proof (*safe*)
  assume *emod-acc*: *𝒮𝒞ℱ-result.electoral-module acc*
  have *sound-imp-defer-subset*:
    𝒮𝒞ℱ-result.electoral-module m
      ⟶ *defer* (*acc* ▷ *m*) *V A p* ⊆ *defer acc V A p*
    using *emod-acc prof seq-comp-def-set-bounded*
    by *blast*
  hence *card-ineq*: *card* (*defer* (*acc* ▷ *m*) *V A p*) < *card* (*defer 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 voters-determine-m
            seq-comp-presv-def-lift-inv
      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 ∈ defer (acc ▷ m) V A p
```

```
        moreover from this have SCF-result.electoral-module (acc ▷ m)
          unfolding defer-lift-invariance-def
          by blast
        moreover have a ∈ 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 ¬ ¬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 acc :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    A :: 'a set and
    V :: 'v set and
    p q :: ('a, 'v) Profile and
    a :: 'a
  assumes
    defer-lift-invariance m and
    voters-determine-election 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**
$p\ q$ :: $('a, 'v)$ *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
$m\ acc$ :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition*
assumes
defer-lift-invariance m **and**
voters-determine-election 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.simps*
defer-lift-invariance-def
by *metis*
next
fix
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
$p\ q$ :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assume
$a \in$ *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
$m\ acc$ :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
A :: $'a$ *set* **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
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: $x \in$ *elect* (*loop-comp-helper acc m t*) *V A p*
have $\forall$ *m' n'*. *non-electing m'* $\wedge$ *non-electing n'* $\longrightarrow$ *non-electing* ($m' \triangleright n'$)
by *simp*
hence *seq-acc-m-non-elect*: *non-electing* (*acc* $\triangleright$ *m*)
using *acc-non-elect non-electing-m*
by *blast*
have $\forall$ *i m'*.
$i <$ *card* (*defer acc V A p*) $\wedge$ i = *card* (*defer m' V A p*) $\wedge$
non-electing m' $\longrightarrow$
elect (*loop-comp-helper m' m t*) *V A p* = {}
using *acc-no-elect*
by *blast*
hence $\forall$ *m'*.
finite (*defer acc V A p*) $\wedge$ *defer m' V A p* $\subset$ *defer acc V A p* $\wedge$
non-electing m' $\longrightarrow$
elect (*loop-comp-helper m' m t*) *V A p* = {}
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*) $\longrightarrow$
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 acc :: (*'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**
n x :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall$ *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* $\triangleright$ *m*) *V A p* $\subset$ *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*) $\neq$ *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* :: *nat* **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'
  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 *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 acc :: ('*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 **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* $\geq x$
shows *card* (*defer* ($m \circlearrowleft_t$) *V A p*) $= x$
proof (*cases*)
assume *card A* $= x$
thus *?thesis*
using *terminate-if-n-left*
by *simp*
next
assume *card-not-x*: $\neg$ *card A* $= x$
thus *?thesis*
proof (*cases*)
assume *card A* $< x$
thus *?thesis*
using *enough-alternatives not-le*
by *blast*
next
assume $\neg$ *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′*
prof x-greater-zero
by *fastforce*
ultimately have *card* (*defer m V A p*) $\geq x$
by *linarith*
moreover have ($m \circlearrowleft_t$) *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**
voters-determine-election 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* $(m \circlearrowleft_t)$
using *loop-comp-sound*
by *blast*
next
fix
A :: *'a set* **and**
V :: *'v set* **and**
$p\ q$ :: (*'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
$\forall\ p'\ q'\ a'.\ a' \in$ (*defer* $(m \circlearrowleft_t)\ V\ A\ p') \land$ *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**
$a \in$ *elect* $(m \circlearrowleft_t)\ V\ A\ p$

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*, *lifting*))
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* ($m \circlearrowleft_t$)
proof (*unfold defers-def*, *safe*)
 show $\mathcal{SCF}$*-result.electoral-module* ($m \circlearrowleft_t$)
  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**
  profile V A p
 thus *card* (*defer* ($m \circlearrowleft_t$) *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* $\Rightarrow$
$('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
$('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
maximum-parallel-composition $m\ n =$
$(let\ a = max\text{-}aggregator\ in\ (m \parallel_a n))$

abbreviation *max-parallel* :: $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
$('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
$('a, 'v, 'a\ Result)$ *Electoral-Module* (**infix** $\parallel_\uparrow$ *50*) **where**
$m \parallel_\uparrow n \equiv$ *maximum-parallel-composition* $m\ n$

6.6.2 Soundness

theorem *max-par-comp-sound*:
fixes $m\ n$ :: $('a, 'v, 'a\ Result)$ *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 max-agg-sound par-comp-sound*
unfolding *maximum-parallel-composition.simps*
by *metis*

lemma *voters-determine-max-par-comp*:
fixes $m\ n$ :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
assumes
voters-determine-election m **and**
voters-determine-election n
shows *voters-determine-election* $(m \parallel_\uparrow n)$
using *max-aggregator.simps assms*
unfolding *Let-def maximum-parallel-composition.simps*
parallel-composition.simps
voters-determine-election.simps
by *presburger*

6.6.3 Lemmas

lemma *max-agg-eq-result*:

fixes
  $m\ n$ :: $('a,\ 'v,\ 'a\ Result)$ *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 max-par-comp-sound*
    by *metis*
  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-defer*: $a \in$ *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.simps*
    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.simps*
    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-defer*
    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* ($m \parallel_m$*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 *a-not-elect-mn*: $a \notin \mathit{elect}\ m\ V\ A\ p \wedge a \notin \mathit{elect}\ n\ V\ A\ p$
by *blast*
have *a-not-mpar-rej*: $a \notin \mathit{reject}\ (m \parallel_{\uparrow} n)\ V\ A\ p$
using *result-disj-max a-in-defer*
by *fastforce*
have *mod-cont-res-fg*:
$\forall\ m'\ n'\ A'\ V'\ p'\ (a' :: 'a).$
$\mathit{mod\text{-}contains\text{-}result}\ m'\ n'\ V'\ A'\ p'\ a' =$
$(\mathcal{SCF}\text{-}\mathit{result.electoral\text{-}module}\ m'$
$\wedge\ \mathcal{SCF}\text{-}\mathit{result.electoral\text{-}module}\ n'$
$\wedge\ \mathit{profile}\ V'\ A'\ p' \wedge a' \in A'$
$\wedge\ (a' \in \mathit{elect}\ m'\ V'\ A'\ p' \longrightarrow a' \in \mathit{elect}\ n'\ V'\ A'\ p')$
$\wedge\ (a' \in \mathit{reject}\ m'\ V'\ A'\ p' \longrightarrow a' \in \mathit{reject}\ n'\ V'\ A'\ p')$
$\wedge\ (a' \in \mathit{defer}\ m'\ V'\ A'\ p' \longrightarrow a' \in \mathit{defer}\ n'\ V'\ A'\ p'))$
unfolding *mod-contains-result-def*
by *simp*
have *max-agg-res*:
$\mathit{max\text{-}aggregator}\ A\ (\mathit{elect}\ m\ V\ A\ p,\ \mathit{reject}\ m\ V\ A\ p,\ \mathit{defer}\ m\ V\ A\ p)$
$(\mathit{elect}\ n\ V\ A\ p,\ \mathit{reject}\ n\ V\ A\ p,\ \mathit{defer}\ n\ V\ A\ p) =$
$(m \parallel_m \mathit{ax\text{-}aggregator}\ n)\ V\ A\ p$
by *simp*
have *well-f-max*:
$\forall\ r'\ r''\ e'\ e''\ d'\ d''\ A'.$
$\mathit{well\text{-}formed}\text{-}\mathcal{SCF}\ A'\ (e',\ r',\ d') \wedge$
$\mathit{well\text{-}formed}\text{-}\mathcal{SCF}\ A'\ (e'',\ r'',\ d'') \longrightarrow$
$\mathit{reject\text{-}r}\ (\mathit{max\text{-}aggregator}\ A'\ (e',\ r',\ d')\ (e'',\ r'',\ d'')) =$
$r' \cap r''$
using *max-agg-rej-set*
by *metis*
have *e-mod-disj*:
$\forall\ m'\ (V' :: 'v\ set)\ (A' :: 'a\ set)\ p'.$
$\mathcal{SCF}\text{-}\mathit{result.electoral\text{-}module}\ m' \wedge \mathit{profile}\ V'\ A'\ p'$
$\longrightarrow \mathit{elect}\ m'\ V'\ A'\ p' \cup \mathit{reject}\ m'\ V'\ A'\ p' \cup \mathit{defer}\ m'\ V'\ A'\ p' = A'$
using *result-presv-alts*
by *blast*
hence *e-mod-disj-n*: $\mathit{elect}\ n\ V\ A\ p \cup \mathit{reject}\ n\ V\ A\ p \cup \mathit{defer}\ n\ V\ A\ p = A$
using *prof-p module-n*
by *metis*
have $\forall\ m'\ n'\ A'\ V'\ p'\ (b :: 'a).$
$\mathit{mod\text{-}contains\text{-}result}\ m'\ n'\ V'\ A'\ p'\ b =$
$(\mathcal{SCF}\text{-}\mathit{result.electoral\text{-}module}\ m'$
$\wedge\ \mathcal{SCF}\text{-}\mathit{result.electoral\text{-}module}\ n'$
$\wedge\ \mathit{profile}\ V'\ A'\ p' \wedge b \in A'$
$\wedge\ (b \in \mathit{elect}\ m'\ V'\ A'\ p' \longrightarrow b \in \mathit{elect}\ n'\ V'\ A'\ p')$
$\wedge\ (b \in \mathit{reject}\ m'\ V'\ A'\ p' \longrightarrow b \in \mathit{reject}\ n'\ V'\ A'\ p')$
$\wedge\ (b \in \mathit{defer}\ m'\ V'\ A'\ p' \longrightarrow b \in \mathit{defer}\ n'\ V'\ A'\ p'))$
unfolding *mod-contains-result-def*
by *simp*

hence $a \notin$ *defer* $n\ V\ A\ p$
using *a-not-mpar-rej a-in-A e-mod-par module-n not-a-elect not-mod-cont-mn prof-p*
by *blast*
hence $a \in$ *reject* $n\ V\ A\ p$
using *a-in-A a-not-elect-mn module-n not-rej-imp-elec-or-defer prof-p*
by *metis*
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 a-not-mpar-rej*
unfolding *maximum-parallel-composition.simps*
by (*metis* (*no-types*))
hence $a \notin$ *defer* $(m \parallel_\uparrow n)\ V\ A\ p \lor a \in$ *defer* $m\ V\ A\ p$
using *e-mod-disj prof-p a-in-A module-m a-not-elect-mn*
by *blast*
thus *mod-contains-result* $(m \parallel_\uparrow n)\ m\ V\ A\ p\ a$
using *a-not-mpar-rej mod-cont-res-fg e-mod-par prof-p a-in-A module-m a-not-elect*
unfolding *maximum-parallel-composition.simps*
by *metis*
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-r* (*max-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-r* (*max-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
    $m\ n$ :: (*'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 $case\ n\ V\ A\ p\ of\ (e, r, d) \Rightarrow$
        $a \in reject\text{-}r\ (max\text{-}aggregator\ A$
            $(elect\ m\ V\ A\ p,\ reject\ m\ V\ A\ p,\ defer\ m\ V\ A\ p)\ (e, r, d))$
    by *auto*
  hence $case\ snd\ (m\ V\ A\ p)\ of\ (r, d) \Rightarrow$
        $case\ n\ V\ A\ p\ of\ (e', r', d') \Rightarrow$
          $a \in reject\text{-}r\ (max\text{-}aggregator\ A\ (elect\ m\ V\ A\ p, r, d)\ (e', r', d'))$
    by *force*
  with *rej-a*
  have $let\ (e, r, d) = m\ V\ A\ p;$
        $(e', r', d') = n\ V\ A\ p\ in$
          $a \in reject\text{-}r\ (max\text{-}aggregator\ A\ (e, r, d)\ (e', r', d'))$
    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-elected-or-deferred assms*
by (*metis* (*no-types*))
qed

lemma *max-agg-rej-fst-imp-seq-contained*:
fixes
m 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 max-par-comp-sound*
by *metis*
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: $a = d \land d \in defer\ m\ V\ A\ p$
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* $m' \land$ *finite* $A' \land$ *finite* $V' \land$ *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
$m\ n :: ('a, 'v, 'a\ Result)$ *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**
$a \in reject\ n\ V\ A\ p$
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 $\|_\uparrow$ n) V A p a
    using *assms max-agg-rej-fst-imp-seq-contained*
    by (*metis* (*full-types*))
  thus
    $a \in$ *elect* (m $\|_\uparrow$ n) V A $p \Longrightarrow a \in$ *elect* m V A p **and**
    $a \in$ *defer* (m $\|_\uparrow$ 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
    $\mathcal{SCF}$*-result.electoral-module* (m $\|_\uparrow$ 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 $\|_\uparrow$ n) V A p **and**
    $a \in$ *reject* m V A $p \Longrightarrow a \in$ *reject* (m $\|_\uparrow$ n) V A p **and**
    $a \in$ *defer* m V A $p \Longrightarrow a \in$ *defer* (m $\|_\uparrow$ 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 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* m V A p
  shows *mod-contains-result* n (m $\|_\uparrow$ n) V A p a
  using *assms*
proof (*unfold mod-contains-result-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module* (m $\|_\uparrow$ n)
    using *module-m module-n max-par-comp-sound*
    by *metis*
next
  show $a \in A$
    using *f-prof in-mono module-m reject-in-alts rejected*
    by (*metis* (*no-types*))

```
next
  assume a ∈ elect n V A p
  thus a ∈ elect (m ∥↑ n) V A p
    using max-aggregator.simps[of
          - 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 ∈ reject n V A p
  thus a ∈ reject (m ∥↑ n) V A p
    using f-prof max-agg-rej-iff-both-reject module-m module-n rejected
    by metis
next
  assume a ∈ defer n V A p
  moreover have a ∈ A
    using f-prof max-agg-rej-fst-imp-seq-contained module-m rejected
    unfolding mod-contains-result-def
    by metis
  ultimately show a ∈ defer (m ∥↑ 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 (no-types, opaque-lifting))
qed

lemma max-agg-rej-snd-equiv-seq-contained:
  fixes
    m 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 ∈ reject m V A p
  shows mod-contains-result-sym (m ∥↑ n) n V A p a
  using assms
proof (unfold mod-contains-result-sym-def, safe)
  assume a ∈ reject (m ∥↑ n) V A p
  thus a ∈ reject n V A p
    using assms max-agg-rej-iff-both-reject
    by (metis (no-types))
next
  have mod-contains-result n (m ∥↑ 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
$m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **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-elected-or-deferred*
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
$m\ n$ :: ($'a$, $'v$, $'a$ *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 $m\ n$ :: ($'a$, $'v$, $'a$ *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 $m\ n$ :: ($'a$, $'v$, $'a$ *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* $(m \parallel_{\uparrow} n)$
using *max-par-comp-sound*
by *metis*
fix
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p\ q :: ('a, 'v)\ Profile$ **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: $B \subseteq A$
$\wedge\ (\forall\ b \in B.\ indep\text{-}of\text{-}alt\ m\ V\ A\ b\ \wedge$
$(\forall\ p'.\ finite\text{-}profile\ V\ A\ p' \longrightarrow b \in reject\ m\ V\ A\ p'))$
$\wedge\ (\forall\ b \in A - B.\ indep\text{-}of\text{-}alt\ n\ V\ A\ b\ \wedge$
$(\forall\ p'.\ finite\text{-}profile\ V\ A\ p' \longrightarrow b \in reject\ n\ V\ A\ p'))$
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* $\wedge$ *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 *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 ∀ b ∈ B. mod-contains-result-sym (m ∥↑ 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 ∀ b ∈ 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 ∈ A
      show SCF-result.electoral-module n ∧ profile V A p ∧
              profile V A q ∧ b ∈ A ∧
              (b ∈ elect n V A p ⟶ b ∈ elect n V A q) ∧
              (b ∈ reject n V A p ⟶ b ∈ reject n V A q) ∧
              (b ∈ defer n V A p ⟶ b ∈ defer n V A q)
      proof (safe)
        show 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 ∈ A
          using f-profs b-in-A
          by (simp, simp, simp)
      next
        show
          b ∈ elect n V A p ⟹ b ∈ elect n V A q and
          b ∈ reject n V A p ⟹ b ∈ reject n V A q and
          b ∈ defer n V A p ⟹ b ∈ 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 ∀ b ∈ B. mod-contains-result n (m ∥↑ 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:
    ∀ b ∈ B. prof-contains-result (m ∥↑ n) V A p q b
      unfolding mod-contains-result-def mod-contains-result-sym-def
              prof-contains-result-def
    by simp
  have ∀ b ∈ A − B. mod-contains-result-sym (m ∥↑ 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 ∀ b ∈ 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 ∈ A
    show 𝒮𝒞ℱ-result.electoral-module m ∧ profile V A p
        ∧ profile V A q ∧ b ∈ A
        ∧ (b ∈ elect m V A p ⟶ b ∈ elect m V A q)
        ∧ (b ∈ reject m V A p ⟶ b ∈ reject m V A q)
        ∧ (b ∈ defer m V A p ⟶ b ∈ defer m V A q)
    proof (safe)
      show 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 ∈ A
        using f-profs b-in-A
        by (simp, simp, simp)
    next
      show
        b ∈ elect m V A p ⟹ b ∈ elect m V A q and
        b ∈ reject m V A p ⟹ b ∈ reject m V A q and
        b ∈ defer m V A p ⟹ b ∈ 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 ∈ A − B. mod-contains-result m (m ∥↑ 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 ∈ A − B. prof-contains-result (m ∥↑ 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 ∥↑ n) V A p = (m ∥↑ 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 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* $\|_\uparrow$ *n*) *V A p*) = *c*
proof −
obtain *B* :: ′*a set* **where**
alt-set: *B* ⊆ *A*
∧ (∀ *a* ∈ *B*. *indep-of-alt m V A a* ∧
(∀ *q*. *profile V A q* ⟶ *a* ∈ *reject m V A q*))
∧ (∀ *a* ∈ *A* − *B*. *indep-of-alt n V A a* ∧
(∀ *q*. *profile V A q* ⟶ *a* ∈ *reject n V A q*))
using *compatible prof*
unfolding *disjoint-compatibility-def*
by *metis*
have *reject-representation*:
reject (*m* $\|_\uparrow$ *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*) ⊆ *A* ∧ (*reject n V A p*) ⊆ *A*
using *prof*
by (*simp add*: *reject-in-alts*)
hence *finite* (*reject m V A p*) ∧ *finite* (*reject n V A p*)
using *rev-finite-subset prof fin-A*
by *metis*
hence *card-difference*:
card (*reject* (*m* $\|_\uparrow$ *n*) *V A p*)
= *card A* + *c* − *card* ((*reject m V A p*) ∪ (*reject n V A p*))
using *card-Un-Int reject-representation reject-sum*
by *fastforce*
have ∀ *a* ∈ *A*. *a* ∈ (*reject m V A p*) ∨ *a* ∈ (*reject n V A p*)
using *alt-set prof fin-A*
by *blast*
hence *A* = *reject m V A p* ∪ *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 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* ($m \parallel_\uparrow n$)
    using *max-par-comp-sound*
    by *metis*
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*: $\mathcal{SCF}$*-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.simps
      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 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 sup-commute fst-conv snd-conv*
  by (*metis* (*no-types*, *opaque-lifting*))

6.7.3 Soundness

theorem *elector-sound*[*simp*]:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  assumes $\mathcal{SCF}$*-result.electoral-module* m
  shows $\mathcal{SCF}$*-result.electoral-module* (*elector* m)
  using *assms elect-mod-sound seq-comp-sound*
  unfolding *elector.simps*
  by *metis*

lemma *voters-determine-elector*:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  assumes *voters-determine-election* m
  shows *voters-determine-election* (*elector* m)
  using *assms elect-mod-only-voters voters-determine-seq-comp*
  unfolding *elector.simps*
  by *metis*

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 ∀ m'.

(¬ *electing* m′ ∨ 𝒮𝒞ℱ-*result.electoral-module* m′ ∧
(∀ A′ V′ p′. (A′ ≠ {} ∧ *finite* A′ ∧ *profile* V′ A′ p′)
⟶ *elect* m′ V′ A′ p′ ≠ {})) ∧
(*electing* m′ ∨ ¬ 𝒮𝒞ℱ-*result.electoral-module* m′
∨ (∃ A V p. (A ≠ {} ∧ *finite* A ∧ *profile* V A p ∧ *elect* m′ V A p = {})))
unfolding *electing-def*
by *blast*
hence ∀ m′.
(¬ *electing* m′ ∨ 𝒮𝒞ℱ-*result.electoral-module* m′ ∧
(∀ A′ V′ p′. (A′ ≠ {} ∧ *finite* A′ ∧ *profile* V′ A′ p′)
⟶ *elect* m′ V′ A′ p′ ≠ {})) ∧
(∃ A V p. (*electing* m′ ∨ ¬ 𝒮𝒞ℱ-*result.electoral-module* m′ ∨ A ≠ {}
∧ *finite* A ∧ *profile* V A p ∧ *elect* m′ V A p = {}))
by *simp*
then obtain
A :: ('a, 'v, 'a *Result*) *Electoral-Module* ⇒ 'a *set* **and**
V :: ('a, 'v, 'a *Result*) *Electoral-Module* ⇒ 'v *set* **and**
p :: ('a, 'v, 'a *Result*) *Electoral-Module* ⇒ ('a, 'v) *Profile* **where**
electing-mod:
∀ m′ :: ('a, 'v, 'a *Result*) *Electoral-Module*.
(¬ *electing* m′ ∨ 𝒮𝒞ℱ-*result.electoral-module* m′ ∧
(∀ A′ V′ p′. (A′ ≠ {} ∧ *finite* A′ ∧ *profile* V′ A′ p′)
⟶ *elect* m′ V′ A′ p′ ≠ {})) ∧
(*electing* m′ ∨ ¬ 𝒮𝒞ℱ-*result.electoral-module* m′
∨ A m′ ≠ {} ∧ *finite* (A m′) ∧ *profile* (V m′) (A m′) (p m′)
∧ *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-elected-or-deferred 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 sup-bot.left-neutral fst-eqD
    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 ($\mathcal{SCF}$*-result.electoral-module* $m \wedge \neg$ *defer-condorcet-consistency* $m \longrightarrow$
($\exists$ A V rs a. *condorcet-winner* V A rs a $\wedge$
m V A $rs \neq (\{\}, A -$ *defer* m V A rs,
$\{a \in A.$ *condorcet-winner* V A rs $a\})))$
$\wedge$ (*defer-condorcet-consistency* $m \longrightarrow$
($\forall$ A V rs a. *finite* $A \longrightarrow$ *finite* $V \longrightarrow$ *condorcet-winner* V A rs $a \longrightarrow$
m V A rs $=$
$(\{\}, A -$ *defer* m V A rs, $\{a \in A.$ *condorcet-winner* V A rs $a\})))$
unfolding *defer-condorcet-consistency-def*
by *blast*
hence
m V A $p = (\{\}, A -$ *defer* m V A p, $\{a \in A.$ *condorcet-winner* V A p $a\})$
using *c-win-x assms fin-A fin-V*
by *blast*
thus $x \in$ *elect* m V A p
using *assms x-not-in-defer fin-A fin-V cond-winner-unique*
defer-condorcet-consistency-def insertCI snd-conv c-win-x
by (*metis* (*no-types*, *lifting*))
qed
ultimately have
elect m V A $p \cup$ *defer* m V A p $=$
$\{e \in A.$ $e \in A \wedge$
$(\forall$ $x \in A - \{e\}$.
card $\{i \in V.$ $(e, x) \in p$ $i\}$ $<$
card $\{i \in V.$ $(x, e) \in p$ $i\})\}$
by *blast*
thus *elector* m V A p $=$
$(\{e \in A.$ *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.

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 *iter-elect* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* ⇒
    (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
  iter-elect m = *elector* (*m* $\circlearrowleft_{\exists!d}$)

6.8.1 Soundness

theorem *defer-one-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* $\circlearrowleft_{\exists!d}$)
  using *assms loop-comp-sound*
  unfolding *Defer-One-Loop-Composition.iter.simps*
  by *metis*

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 =
 (*if finite A*
 then ($\{a \in A.\ \forall\ x \in A.$ *win-count V p x* $\leq$ *win-count V p a*$\}$,
 $\{a \in A.\ \exists\ x \in A.$ *win-count V p x* $>$ *win-count V p a*$\}$,
 $\{\}$)
 else (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*
 by *fastforce*

lemma *plurality'-revision-equiv*:

```
  fixes
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  shows plurality' V A p = (plurality-rule'↓) V A p
proof (unfold plurality'.simps revision-composition.simps,
        intro prod-eqI equalityI subsetI prod.sel)
  fix b :: 'a
  assume
    assm: b ∈ fst
            (if finite A
             then ({}, {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})
             else ({}, {}, A))
  have finite A ⟶ b ∈ {}
    using assm
    by force
  moreover have infinite A ⟶ b ∈ {}
    using assm
    by fastforce
  ultimately show
    b ∈ fst ({}, A − elect plurality-rule' V A p, elect plurality-rule' V A p)
    by safe
next
  fix b :: 'a
  assume b ∈ fst ({}, A − elect plurality-rule' V A p, elect plurality-rule' V A p)
  thus b ∈ fst
        (if finite A
         then ({}, {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})
         else ({}, {}, A))
    by force
next
  fix b :: 'a
  assume
    assm: b ∈ fst (snd
            (if finite A
             then ({}, {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})
             else ({}, {}, A)))
  have finite A ⟶
    b ∈ A − {a ∈ A. ∀ x ∈ A. win-count V p x ≤ win-count V p a}
    using assm
    by fastforce
  moreover have infinite A ⟶ b ∈ {}
    using assm
    by fastforce
  ultimately show
    b ∈ fst (snd ({}, A − elect plurality-rule' V A p, elect plurality-rule' V A p))
```

by *force*
next
fix *b* :: *'a*
assume
assm: *b* ∈
fst (*snd* ({}, *A* − *elect plurality-rule' V A p*, *elect plurality-rule' V A p*))
have *finite A* ⟶
b ∈ *A* − {*a* ∈ *A*. ∀ *x* ∈ *A*. *win-count V p x* ≤ *win-count V p a*}
using *assm*
by *fastforce*
moreover have *infinite A* ⟶ *b* ∈ {}
using *assm*
by *fastforce*
ultimately show
b ∈ *fst* (*snd*
(*if finite A*
then ({}, {*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*})
else ({}, {}, *A*)))
using *linorder-not-less*
by *force*
next
fix *b* :: *'a*
assume
assm: *b* ∈ *snd* (*snd*
(*if finite A*
then ({}, {*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*})
else ({}, {}, *A*)))
have *finite A* ⟶
b ∈ *A* − {*a* ∈ *A*. ∃ *x* ∈ *A*. *win-count V p a* < *win-count V p x*}
using *assm*
by *fastforce*
moreover have *infinite A* ⟶ *b* ∈ *A*
using *assm*
by *fastforce*
ultimately have *b* ∈ *elect plurality-rule' V A p*
by *force*
thus *b* ∈ *snd* (*snd*
({}, *A* − *elect plurality-rule' V A p*, *elect plurality-rule' V A p*))
by *simp*
next
fix *b* :: *'a*
assume *assm*:
b ∈ *snd* (*snd* ({}, *A* − *elect plurality-rule' V A p*, *elect plurality-rule' V A p*))
have *finite A* ⟶
b ∈ *A* − {*a* ∈ *A*. ∃ *x* ∈ *A*. *win-count V p a* < *win-count V p x*}
using *assm*
by *fastforce*

moreover have *infinite* $A \longrightarrow b \in A$
  using *assm*
  by *fastforce*
ultimately show $b \in$ *snd* (*snd*
      (*if finite* A
       then $(\{\}, \{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ a < \textit{win-count}\ V\ p\ x\},$
         $\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\})$
       else $(\{\}, \{\}, A)))$
  by *force*
qed

lemma *plurality-rule-equiv*:
fixes
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: ('a, 'v) *Profile*
shows *plurality-rule* $V\ A\ p$ = *plurality-rule'* $V\ A\ p$
proof (*unfold plurality-rule.simps*)
have *plurality-rule'-rev-equiv-plurality*:
  plurality $V\ A\ p$ = (*plurality-rule'*↓) $V\ A\ p$
  using *plurality-mod-equiv plurality'-revision-equiv*
  by *metis*
have *defer* (*elector plurality*) $V\ A\ p$ = *defer plurality-rule'* $V\ A\ p$
  by *force*
moreover have *reject* (*elector plurality*) $V\ A\ p$ = *reject plurality-rule'* $V\ A\ p$
  using *plurality-rule'-rev-equiv-plurality*
  by *force*
moreover have *elect* (*elector plurality*) $V\ A\ p$ = *elect plurality-rule'* $V\ A\ p$
  using *plurality-rule'-rev-equiv-plurality*
  by *force*
ultimately show *elector plurality* $V\ A\ p$ = *plurality-rule'* $V\ A\ p$
  using *prod-eqI*
  by (*metis* (*mono-tags*, *lifting*))
qed

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.simps*, *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 *voters-determine-plurality-rule*: *voters-determine-election plurality-rule*
unfolding *plurality-rule.simps*
using *voters-determine-elector voters-determine-plurality*
by *blast*

lemma *voters-determine-plurality-rule'*: *voters-determine-election plurality-rule'*
proof (*unfold voters-determine-election.simps*, *safe*)
fix
A :: *'k set* **and**
V :: *'v set* **and**
$p\ p'$ :: (*'k*, *'v*) *Profile*
assume $\forall\ v \in V.\ p\ v = p'\ v$
thus *plurality-rule'* $V\ A\ p$ = *plurality-rule'* $V\ A\ p'$
using *voters-determine-plurality-rule plurality-rule-equiv*
unfolding *voters-determine-election.simps*
by (*metis* (*full-types*))
qed

7.1.3 Electing

lemma *plurality-rule-elect-non-empty*:
fixes
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$A \neq \{\}$ **and**
finite A
shows *elect plurality-rule* $V\ A\ p \neq \{\}$
proof
assume *plurality-elect-none*: *elect plurality-rule* $V\ A\ p = \{\}$
obtain *max* :: *enat* **where**
max: *max* = *Max* (*win-count* $V\ p$ ` A)
by *simp*
then obtain a :: *'a* **where**
max-a: *win-count* $V\ p\ a$ = *max* $\wedge\ a \in A$

using *Max-in assms 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 *assms 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-revision-equiv plurality-rule-equiv snd-conv*
unfolding *revision-composition.simps*
by (*metis* (*no-types*, *opaque-lifting*))
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

lemma *plurality-rule'-elect-non-empty*:
fixes
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$A \neq \{\}$ **and**
profile $V\ A\ p$ **and**
finite A
shows *elect plurality-rule'* $V\ A\ p \neq \{\}$
using *assms plurality-rule-elect-non-empty plurality-rule-equiv*
by *metis*

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 $\forall$ B W q. $B \neq \{\} \wedge$ *finite* $B \wedge$ *profile* W B q
$\longrightarrow$ *elect plurality-rule* W B $q \neq \{\}$
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 $a \in \{\}$
using *a-in-A*
by *simp*
qed

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 *metis*
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**
no-elect': *elect plurality-rule'* V A $p = \{\}$ **and**
a-in-A: $a \in A$
have *A-nonempty*: $A \neq \{\}$
using *a-in-A*
by *blast*
have *elect plurality-rule* V A $p = \{\}$
using *no-elect' plurality-rule-equiv*
by *metis*
moreover have *elect plurality-rule* V A $p \neq \{\}$
using *fin-A prof-p A-nonempty plurality-rule'-elect-non-empty plurality-rule-equiv*
by *metis*
ultimately show $a \in \{\}$
by *force*
qed

7.1.4 Properties

lemma *plurality-rule-inv-mono-eq*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p q :: $('a,\ 'v)$ *Profile* **and**
    a :: $'a$
  assumes
    elect-a: $a \in$ *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
        $\vee$ *elect plurality-rule* V A q = $\{a\}$
proof −
  have $a \in$ *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 $a \in$ *defer plurality* V A p
    by *blast*
  hence *defer plurality* V A q = *defer plurality* V A p
        $\vee$ *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
      $\vee$ *elect plurality-rule* V A q = $\{a\}$
    using *eq-p*
    by *simp*
qed

lemma *plurality-rule'-inv-mono-eq*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p q :: $('a,\ 'v)$ *Profile* **and**
    a :: $'a$
  assumes
    $a \in$ *elect plurality-rule'* V A p **and**
    lifted V A p q a
  shows *elect plurality-rule'* V A q = *elect plurality-rule'* V A p
        $\vee$ *elect plurality-rule'* V A q = $\{a\}$
  using *assms plurality-rule-equiv plurality-rule-inv-mono-eq*
  by (*metis* (*no-types*))

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 $\mathcal{SCF}$*-result.electoral-module plurality-rule*
    using *plurality-rule-sound*
    by *metis*
next
  fix
    A :: $'b$ *set* **and**
    V :: $'a$ *set* **and**
    p 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

theorem *plurality-rule′-inv-mono*[*simp*]: *invariant-monotonicity plurality-rule′*
proof −
  have (*plurality-rule*::($'k$, $'v$, $'k$ *Result*) *Electoral-Module*) = *plurality-rule′*
    using *plurality-rule-equiv*
    by *blast*
  thus *?thesis*
    using *plurality-rule-inv-mono*
    by (*metis* (*full-types*))
qed

(Weak) Monotonicity

theorem *plurality-rule-monotone*: *monotonicity plurality-rule*
proof (*unfold monotonicity-def*, *safe*)
  show $\mathcal{SCF}$*-result.electoral-module plurality-rule*
    using *plurality-rule-sound*
    by (*metis* (*no-types*))
next
  fix
    A :: $'b$ *set* **and**
    V :: $'a$ *set* **and**
    p q :: $('b, 'a)$ *Profile* **and**
    a :: $'b$
  assume
    $a \in$ *elect plurality-rule* V A p **and**
    Profile.lifted V A p q a
  thus $a \in$ *elect plurality-rule* V A q
    using *insertI1 plurality-rule-inv-mono-eq*
    by (*metis* (*no-types*))
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

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 = iter-elect 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'-sound*: $\mathcal{SCF}$*-result.electoral-module condorcet'*
using *Defer-One-Loop-Composition.iter.elims loop-comp-sound min-elim-sound*
unfolding *condorcet'.simps loop-comp-sound*
by *metis*

theorem *pairwise-majority-rule'-sound*: $\mathcal{SCF}$*-result.electoral-module pairwise-majority-rule'*
unfolding *pairwise-majority-rule'.simps*
using *condorcet'-sound elector-sound iter.simps iter-elect.simps loop-comp-sound*
by *metis*

7.3.3 Condorcet Consistency

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

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

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

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*

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

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*
using *min-elim-sound loop-comp-sound*
unfolding *nanson-baldwin-rule.simps Defer-One-Loop-Composition.iter.simps*
by *metis*

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*
using *leq-avg-elim-sound loop-comp-sound*
unfolding *classic-nanson-rule.simps Defer-One-Loop-Composition.iter.simps*
by *metis*

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*
  using *less-avg-elim-sound loop-comp-sound*
  unfolding *schwartz-rule.simps Defer-One-Loop-Composition.iter.simps*
  by *metis*

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-Relation* ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**

smc x V A p =
  ((*elector* ((((*pass-module 2 x*) ▹ ((*plurality-rule*↓) ▹ (*pass-module 1 x*))) $\|_\uparrow$
  (*drop-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-Relation*
  shows $\mathcal{SCF}$*-result.electoral-module* (*smc x*)
proof (*unfold* $\mathcal{SCF}$*-result.electoral-module.simps well-formed-*$\mathcal{SCF}$*.simps*, *safe*)
  fix
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile*
  assume *profile V A p*
  thus
    disjoint3 (*smc x V A p*) **and**
    set-equals-partition A (*smc x V A p*)
    unfolding *iter.simps smc.simps elector.simps*
    using *drop-mod-sound elect-mod-sound loop-comp-sound max-par-comp-sound*
        pass-mod-sound plurality-rule-sound rev-comp-sound seq-comp-sound
    by (*metis* (*no-types*) *seq-comp-presv-disj*, *metis* (*no-types*) *seq-comp-presv-alts*)
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*↓)
    using *plurality-rule-sound rev-comp-non-electing*

by *metis*
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 voters-determine-pass-mod voters-determine-plurality-rule voters-determine-seq-comp voters-determine-rev-comp*
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

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*$\downarrow$) $\triangleright$ *?tie-breaker*
let *?compare-two* = *?pass2* $\triangleright$ *?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 *00010*: *defer-invariant-monotonicity* (*plurality-rule*$\downarrow$)
by *simp*
have *00011*: *non-electing* (*plurality-rule*$\downarrow$)
using *rev-comp-non-electing plurality-rule-sound*
by *blast*
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*$\downarrow$)
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 voters-determine-election.simps
    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
          voters-determine-plurality-rule voters-determine-pass-mod
          voters-determine-rev-comp voters-determine-seq-comp
    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 plurality-rule-sound
      voters-determine-pass-mod rev-comp-sound seq-comp-sound voters-determine-seq-comp
      voters-determine-plurality-rule voters-determine-pass-mod voters-determine-rev-comp
    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
      voters-determine-plurality-rule voters-determine-pass-mod voters-determine-drop-mod
      voters-determine-rev-comp voters-determine-seq-comp voters-determine-max-par-comp
    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-$\mathcal{R}$ strong-unanimity V A p*

7.11.2 Soundness

theorem *kemeny-rule-sound*: *$\mathcal{SCF}$-result.electoral-module kemeny-rule*
unfolding *kemeny-rule.simps swap-$\mathcal{R}$.simps*
using *$\mathcal{SCF}$-result.$\mathcal{R}$-sound*
by *metis*

7.11.3 Anonymity

theorem *kemeny-rule-anonymous*: *$\mathcal{SCF}$-result.anonymity kemeny-rule*
proof (*unfold kemeny-rule.simps swap-$\mathcal{R}$.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 *$\mathcal{SCF}$-result.anonymity*
(*$\mathcal{SCF}$-result.distance-$\mathcal{R}$?swap-dist strong-unanimity*)
using *strong-unanimity-anonymous*
$\mathcal{SCF}$-result.anonymous-distance-and-consensus-imp-rule-anonymity
by *metis*
qed

7.11.4 Neutrality

lemma *swap-dist-neutral*: *distance-neutrality well-formed-elections*
(*votewise-distance swap l-one*)
using *neutral-dist-imp-neutral-votewise-dist swap-neutral*
by *blast*

theorem *kemeny-rule-neutral*: *$\mathcal{SCF}$-properties.neutrality kemeny-rule*
using *strong-unanimity-neutral' swap-dist-neutral strong-unanimity-closed-under-neutrality*
$\mathcal{SCF}$-properties.neutr-dist-and-cons-imp-neutr-dr
unfolding *kemeny-rule.simps swap-$\mathcal{R}$.simps*
by *blast*

end

Bibliography

[1] Karsten Diekhoff, Michael Kirsten, and Jonas Krämer. Formal property-oriented design of voting rules using composable modules. In Saša Pekeč and Kristen Brent Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019. doi:10.1007/978-3-030-31489-7.

[2] Karsten Diekhoff, Michael Kirsten, and Jonas Krämer. Verified construction of fair voting rules. In Maurizio 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. doi:10.1007/978-3-030-45260-5_6.

[3] Benjamin Hadjibeyli and Mark C. Wilson. Distance rationalization of social rules. *Computing Research Repository (CoRR)*, abs/1610.01902, 2016. arXiv:1610.01902.