

Verified Construction of Fair Voting Rules

Michael Kirsten

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

November 17, 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

1	Social-Choice Types	9
1.1	Auxiliary Lemmas	9
1.2	Preference Relation	11
1.2.1	Definition	11
1.2.2	Ranking	12
1.2.3	Limited Preference	13
1.2.4	Auxiliary Lemmas	18
1.2.5	Lifting Property	27
1.3	Norm	36
1.3.1	Definition	36
1.3.2	Auxiliary Lemmas	37
1.3.3	Common Norms	38
1.3.4	Properties	38
1.3.5	Theorems	38
1.4	Electoral Result	39
1.4.1	Auxiliary Functions	39
1.4.2	Definition	40
1.5	Preference Profile	40
1.5.1	Definition	41
1.5.2	Vote Count	42
1.5.3	Voter Permutations	45
1.5.4	List Representation	48
1.5.5	Preference Counts	54
1.5.6	Condorcet Winner	57
1.5.7	Limited Profile	58
1.5.8	Lifting Property	59
1.6	Social Choice Result	62
1.6.1	Definition	62
1.6.2	Auxiliary Lemmas	62
1.7	Social Welfare Result	65
1.8	Electoral Result Types	65
1.9	Symmetry Properties of Functions	67
1.9.1	Functions	67

1.9.2	Relations for Symmetry Constructions	67
1.9.3	Invariance and Equivariance	68
1.9.4	Auxiliary Lemmas	68
1.9.5	Rewrite Rules	69
1.9.6	Group Actions	73
1.9.7	Invariance and Equivariance	76
1.9.8	Function Composition	81
1.10	Symmetry Properties of Voting Rules	84
1.10.1	Definitions	84
1.10.2	Auxiliary Lemmas	86
1.10.3	Anonymity Lemmas	96
1.10.4	Neutrality Lemmas	104
1.10.5	Homogeneity Lemmas	117
1.10.6	Reversal Symmetry Lemmas	118
1.11	Result-Dependent Voting Rule Properties	123
1.11.1	Property Definitions	123
1.11.2	Interpretations	123
2	Refined Types	124
2.1	Preference List	124
2.1.1	Well-Formedness	124
2.1.2	Auxiliary Lemmas About Lists	124
2.1.3	Ranking	129
2.1.4	Definition	129
2.1.5	Limited Preference	138
2.1.6	Auxiliary Definitions	142
2.1.7	Auxiliary Lemmas	142
2.1.8	First Occurrence Indices	145
2.2	Preference (List) Profile	147
2.2.1	Definition	147
2.2.2	Refinement Proof	148
2.3	Ordered Relation Type	149
2.4	Alternative Election Type	151
3	Quotient Rules	153
3.1	Quotients of Equivalence Relations	153
3.1.1	Definitions	153
3.1.2	Well-Definedness	153
3.1.3	Equivalence Relations	156
3.2	Quotients of Election Set Equivalences	158
3.2.1	Auxiliary Lemmas	158
3.2.2	Anonymity Quotient: Grid	161
3.2.3	Homogeneity Quotient: Simplex	170

4	Component Types	194
4.1	Distance	194
4.1.1	Definition	194
4.1.2	Conditions	195
4.1.3	Standard-Distance Property	195
4.1.4	Auxiliary Lemmas	195
4.1.5	Swap Distance	196
4.1.6	Spearman Distance	197
4.1.7	Properties	198
4.2	Votewise Distance	203
4.2.1	Definition	203
4.2.2	Inference Rules	203
4.3	Consensus	211
4.3.1	Definition	211
4.3.2	Consensus Conditions	211
4.3.3	Properties	212
4.3.4	Auxiliary Lemmas	212
4.3.5	Theorems	215
4.4	Electoral Module	219
4.4.1	Definition	219
4.4.2	Auxiliary Definitions	220
4.4.3	Properties	220
4.4.4	Social-Welfare Properties	222
4.4.5	Social-Choice Modules	222
4.4.6	Equivalence Definitions	223
4.4.7	Auxiliary Lemmas	224
4.4.8	Non-Blocking	234
4.4.9	Electing	234
4.4.10	Properties	235
4.4.11	Inference Rules	240
4.4.12	Social-Choice Properties	242
4.5	Electoral Module on Election Quotients	244
4.6	Evaluation Function	245
4.6.1	Definition	245
4.6.2	Property	245
4.6.3	Theorems	245
4.7	Elimination Module	247
4.7.1	General Definitions	247
4.7.2	Social-Choice Definitions	248
4.7.3	Social-Choice Eliminators	248
4.7.4	Soundness	248
4.7.5	Independence of Non-Voters	249
4.7.6	Non-Blocking	252
4.7.7	Non-Electing	254

4.7.8	Inference Rules	255
4.8	Aggregator	257
4.8.1	Definition	258
4.8.2	Properties	258
4.9	Maximum Aggregator	258
4.9.1	Definition	258
4.9.2	Auxiliary Lemma	259
4.9.3	Soundness	259
4.9.4	Properties	260
4.10	Termination Condition	261
4.11	Defer Equal Condition	261
5	Basic Modules	263
5.1	Defer Module	263
5.1.1	Definition	263
5.1.2	Soundness	263
5.1.3	Properties	263
5.2	Elect-First Module	264
5.2.1	Definition	264
5.2.2	Soundness	264
5.3	Consensus Class	265
5.3.1	Definition	265
5.3.2	Consensus Choice	265
5.3.3	Auxiliary Lemmas	266
5.3.4	Consensus Rules	269
5.3.5	Properties	269
5.3.6	Inference Rules	270
5.3.7	Theorems	273
5.4	Distance Rationalization	280
5.4.1	Definitions	280
5.4.2	Standard Definitions	281
5.4.3	Auxiliary Lemmas	282
5.4.4	Soundness	289
5.4.5	Inference Rules	289
5.5	Votewise Distance Rationalization	297
5.5.1	Common Rationalizations	297
5.5.2	Theorems	297
5.5.3	Equivalence Lemmas	299
5.6	Symmetry in Distance-Rationalizable Rules	300
5.6.1	Minimizer Function	300
5.6.2	Minimizer Translation	306
5.6.3	Inference Rules	314
5.6.4	Properties	317
5.7	Distance Rationalization on Election Quotients	317

5.7.1	Distances	317
5.7.2	Consensus and Results	328
5.7.3	Distance Rationalization	333
5.8	Code Generation Interpretations for Results and Properties	339
5.8.1	Code Lemmas	339
5.8.2	Interpretation Declarations and Constants	341
5.9	Drop Module	341
5.9.1	Definition	342
5.9.2	Soundness	342
5.9.3	Non-Electing	343
5.9.4	Properties	343
5.10	Pass Module	343
5.10.1	Definition	344
5.10.2	Soundness	344
5.10.3	Non-Blocking	345
5.10.4	Non-Electing	345
5.10.5	Properties	346
5.11	Elect Module	352
5.11.1	Definition	352
5.11.2	Soundness	352
5.11.3	Electing	352
5.12	Plurality Module	352
5.12.1	Definition	353
5.12.2	Soundness	355
5.12.3	Non-Blocking	356
5.12.4	Non-Electing	357
5.12.5	Property	357
5.13	Borda Module	362
5.13.1	Definition	362
5.13.2	Soundness	362
5.13.3	Non-Blocking	363
5.13.4	Non-Electing	363
5.14	Condorcet Module	363
5.14.1	Definition	363
5.14.2	Soundness	364
5.14.3	Property	364
5.15	Copeland Module	365
5.15.1	Definition	365
5.15.2	Soundness	365
5.15.3	Lemmas	366
5.15.4	Property	369
5.16	Minimax Module	371
5.16.1	Definition	371
5.16.2	Soundness	371

5.16.3	Lemma	371
5.16.4	Property	372
6	Compositional Structures	375
6.1	Drop- and Pass-Compatibility	375
6.2	Revision Composition	380
6.2.1	Definition	380
6.2.2	Soundness	380
6.2.3	Composition Rules	381
6.3	Sequential Composition	384
6.3.1	Definition	384
6.3.2	Soundness	390
6.3.3	Lemmas	391
6.3.4	Composition Rules	395
6.4	Parallel Composition	419
6.4.1	Definition	419
6.4.2	Soundness	420
6.4.3	Composition Rule	420
6.5	Loop Composition	422
6.5.1	Definition	423
6.5.2	Soundness	427
6.5.3	Lemmas	429
6.5.4	Composition Rules	440
6.6	Maximum Parallel Composition	442
6.6.1	Definition	443
6.6.2	Soundness	443
6.6.3	Lemmas	444
6.6.4	Composition Rules	454
6.7	Elect Composition	462
6.7.1	Definition	463
6.7.2	Auxiliary Lemmas	463
6.7.3	Soundness	463
6.7.4	Electing	463
6.7.5	Composition Rule	465
6.8	Defer-One Loop Composition	467
6.8.1	Soundness	468
7	Voting Rules	469
7.1	Plurality Rule	469
7.1.1	Definition	469
7.1.2	Soundness	472
7.1.3	Electing	473
7.1.4	Properties	476
7.2	Borda Rule	477

7.2.1	Definition	477
7.2.2	Soundness	478
7.2.3	Anonymity	478
7.3	Pairwise Majority Rule	478
7.3.1	Definition	479
7.3.2	Soundness	479
7.3.3	Condorcet Consistency	479
7.4	Copeland Rule	479
7.4.1	Definition	480
7.4.2	Soundness	480
7.4.3	Condorcet Consistency	480
7.5	Minimax Rule	480
7.5.1	Definition	480
7.5.2	Soundness	480
7.5.3	Condorcet Consistency	481
7.6	Black's Rule	481
7.6.1	Definition	481
7.6.2	Soundness	481
7.6.3	Condorcet Consistency	482
7.7	Nanson-Baldwin Rule	482
7.7.1	Definition	482
7.7.2	Soundness	482
7.8	Classic Nanson Rule	482
7.8.1	Definition	483
7.8.2	Soundness	483
7.9	Schwartz Rule	483
7.9.1	Definition	483
7.9.2	Soundness	484
7.10	Sequential Majority Comparison	484
7.10.1	Definition	484
7.10.2	Soundness	484
7.10.3	Electing	485
7.10.4	(Weak) Monotonicity	486
7.11	Kemeny Rule	489
7.11.1	Definition	489
7.11.2	Soundness	489
7.11.3	Anonymity	489
7.11.4	Neutrality	490

Chapter 1

Social-Choice Types

1.1 Auxiliary Lemmas

```
theory Auxiliary-Lemmas
  imports Main
begin

lemma sum-comp:
  fixes
     $f :: 'x \Rightarrow 'z :: \text{comm-monoid-add}$  and
     $g :: 'y \Rightarrow 'x$  and
     $X :: 'x \text{ set}$  and
     $Y :: 'y \text{ set}$ 
  assumes  $\text{bij-betw } g \ Y \ X$ 
  shows  $\text{sum } f \ X = \text{sum } (f \circ g) \ Y$ 
  using assms
proof (induction card X arbitrary: X Y f g)
  case 0
  hence  $\text{card } Y = 0$ 
    using bij-betw-same-card
    unfolding 0
    by simp
  hence
     $\text{sum } f \ X = 0$  and
     $\text{sum } (f \circ g) \ Y = 0$ 
    using 0 card-0-eq sum.empty sum.infinite
    by (metis, metis)
  thus ?case
    by simp
next
  case (Suc n)
  assume
     $\text{card-}X: \text{Suc } n = \text{card } X$  and
     $\text{bij-}g: \text{bij-betw } g \ Y \ X$ 
  obtain  $x :: 'x$ 
```

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

lemma the-inv-comp :

fixes
 $f :: 'y \Rightarrow 'z$ **and**
 $g :: 'x \Rightarrow 'y$ **and**
 $s :: 'x \text{ set}$ **and**
 $t :: 'y \text{ set}$ **and**
 $u :: 'z \text{ set}$ **and**
 $x :: 'z$
assumes
 $\text{bij-betw } f t u$ **and**
 $\text{bij-betw } g s t$ **and**
 $x \in u$
shows $\text{the-inv-into } s (f \circ g) x = ((\text{the-inv-into } s g) \circ (\text{the-inv-into } t f)) x$
proof (unfold comp-def)
have $\text{el-}Y$: $\text{the-inv-into } t f x \in t$
using $\text{assms } \text{bij-betw-apply } \text{bij-betw-the-inv-into}$
by metis

```

hence  $g (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)) = the\_inv\_into\ t\ f\ x$ 
using assms f-the-inv-into-f-bij-betw
by metis
moreover have  $f (the\_inv\_into\ t\ f\ x) = x$ 
using el-Y assms f-the-inv-into-f-bij-betw
by metis
ultimately have  $(f \circ g) (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)) = x$ 
by simp
hence  $the\_inv\_into\ s\ (f \circ g)\ x =$ 
 $the\_inv\_into\ s\ (f \circ g)\ ((f \circ g) (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)))$ 
by presburger
also have
 $the\_inv\_into\ s\ (f \circ g)\ ((f \circ g) (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x))) =$ 
 $the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)$ 
using assms bij-betw-apply bij-betw-imp-inj-on bij-betw-the-inv-into
 $bij\_betw\_trans\ the\_inv\_into\_f\_eq$ 
by (metis (no-types, lifting))
also have  $the\_inv\_into\ s\ (f \circ g)\ x = the\_inv\_into\ s\ (\lambda\ x.\ f\ (g\ x))\ x$ 
using o-apply
by metis
finally show  $the\_inv\_into\ s\ (\lambda\ x.\ f\ (g\ x))\ x = the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)$ 
by presburger
qed

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 × 'a Preference-Relation

```

```

fun is-less-preferred-than :: 'a ⇒ 'a Preference-Relation ⇒ 'a ⇒ bool

```

$(- \preceq_r - [50, 1000, 51] \ 50)$ **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* ≥ 1
proof (*unfold rank.simps above-def*)
have *a* $\in \{b \in \text{Field } r. (a, b) \in r\}$
using *FieldI2 refl*
by *fastforce*
hence $\{b \in \text{Field } r. (a, b) \in r\} \neq \{\}$
by *blast*
hence *card* $\{b \in \text{Field } r. (a, b) \in r\} \neq 0$
by (*simp add: fin finite-Field*)

```

    thus  $1 \leq \text{card } \{b. (a, b) \in r\}$ 
    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 \Rightarrow 'a Preference-Relation \Rightarrow 'a Preference-Relation **where**
limit A r = $\{(a, b) \in r. a \in A \wedge b \in A\}$

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

lemma *connex-imp-refl*:

```

fixes
  A :: 'a set and
  r :: 'a Preference-Relation
assumes connex A r
shows refl-on A r
using assms
proof (unfold connex-def refl-on-def limited-def, elim conjE conjI, safe)
  fix a :: 'a
  assume a  $\in A$ 
  hence a  $\preceq_r$  a
  using assms
  unfolding connex-def
  by metis
  thus (a, a)  $\in r$ 
  by simp
qed

```

lemma *lin-ord-imp-connex*:

```

fixes

```

```

    A :: 'a set and
    r :: 'a Preference-Relation
  assumes linear-order-on A r
  shows connex A r
proof (unfold connex-def limited-def, safe)
  fix a 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
    ¬ b ≤r a
  moreover from this
  have (b, a) ∉ 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) ∈ r
    using assms refl-onD
    unfolding linear-order-on-def total-on-def
    by metis
  thus a ≤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) ∈ 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
     $a \in A$  and
     $b \in A$  and
     $(b, a) \notin r$ 
  moreover with connex-r
  have  $a \preceq_r b \vee b \preceq_r a$ 
    unfolding connex-def
    by metis
  hence  $(a, b) \in r \vee (b, a) \in r$ 
    by simp
  ultimately show  $(a, b) \in r$ 
    by metis
qed

```

```

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

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

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

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

lemma *lin-ord-singleton*:
fixes $a :: 'a$
shows $\forall \ r. \text{linear-order-on } \{a\} \ r \longrightarrow r = \{(a, a)\}$
proof (*clarify*)
fix $r :: 'a \text{ Preference-Relation}$
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
 $r :: 'a \text{ Preference-Relation}$ and
 $a b :: 'a$
 assumes
 $\text{trans } r$ and
 $(a, b) \in r$
 shows $\text{above } r \ b \subseteq \text{above } r \ a$
 using *Collect-mono assms transE*
 unfolding *above-def*
 by *metis*

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

lemma *above-subset-geq-one*:
 fixes
 $A :: 'a \text{ set}$ and
 $r \ r' :: 'a \text{ Preference-Relation}$ and
 $a :: 'a$
 assumes
 $\text{linear-order-on } A \ r$ and
 $\text{linear-order-on } A \ r'$ and
 $\text{above } r \ a \subseteq \text{above } r' \ a$ and
 $\text{above } r' \ a = \{a\}$

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

lemma *pref-imp-in-above:*
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a \ b :: 'a$
shows $(a \preceq_r b) = (b \in \text{above } r \ a)$
unfolding *above-def*
by *simp*

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

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

lemma *above-one*:

fixes

$A :: 'a \text{ set}$ **and**

$r :: 'a \text{ Preference-Relation}$

assumes

lin-ord-r: *linear-order-on* A r **and**

fin-A: *finite* A **and**

non-empty-A: $A \neq \{\}$

shows $\exists a \in A. \text{above } r \ a = \{a\} \wedge (\forall a' \in A. \text{above } r \ a' = \{a'\} \longrightarrow a' = a)$

proof $-$

obtain $n :: \text{nat}$ **where**

len-n-plus-one: $n + 1 = \text{card } A$

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 $r \wedge \text{finite } A \wedge A \neq \{\} \wedge n + 1 = \text{card } A$

$\longrightarrow (\exists a \in A. \text{above } r \ a = \{a\})$

proof (*induction n arbitrary: A r; clarify*)

case 0

fix

$A' :: 'a \text{ set}$ **and**

$r' :: 'a \text{ Preference-Relation}$

assume

lin-ord-r: *linear-order-on* A' r' **and**

len-A-is-one: $0 + 1 = \text{card } A'$

then obtain $a :: 'a$ **where**

$A' = \{a\}$

using *card-1-singletonE add.left-neutral*

by *metis*

hence

$a \in A'$ **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 $\exists a' \in A'. \text{above } r' \ a' = \{a'\}$

by *metis*

next

case (*Suc n*)

fix

$A' :: 'a \text{ set}$ **and**

$r' :: 'a \text{ Preference-Relation}$

assume

lin-ord-r: *linear-order-on* A' r' **and**

fin-A: *finite* A' **and**

A-not-empty: $A' \neq \{\}$ **and**

len-A-n-plus-one: $\text{Suc } n + 1 = \text{card } A'$

then obtain $B :: 'a \text{ set}$ **where**

$\text{subset-}B\text{-card: } \text{card } B = n + 1 \wedge B \subseteq A'$
using *Suc-inject add-Suc card.insert-remove finite.cases insert-Diff-single*
 subset-insertI
by (*metis (mono-tags, lifting)*)
then obtain $a :: 'a$ **where**
 $a: A' - B = \{a\}$
using *Suc-eq-plus1 add-diff-cancel-left' fin-A len-A-n-plus-one card-1-singletonE*
 $\text{card-Diff-subset finite-subset}$
by *metis*
have $\exists a' \in B. \text{above } (\text{limit } B \ r') \ a' = \{a'\}$
using *subset-B-card Suc.IH add-diff-cancel-left' lin-ord-r card-eq-0-iff diff-le-self*
 $\text{leD lessI limit-presv-lin-ord}$
unfolding *One-nat-def*
by *metis*
then obtain $b :: 'a$ **where**
 $\text{alt-b: above } (\text{limit } B \ r') \ b = \{b\}$
by *blast*
hence $b\text{-above: } \{a'. (b, a') \in \text{limit } B \ r'\} = \{b\}$
unfolding *above-def*
by *metis*
hence $b\text{-pref-b: } b \preceq_{r'} b$
using *CollectD limit-rel-presv-prefs singletonI*
by (*metis (lifting)*)
show $\exists a' \in A'. \text{above } r' \ a' = \{a'\}$
proof (*cases*)
assume $a\text{-pref-r-b: } a \preceq_{r'} b$
have *refl-A:*
 $\forall A'' \ r'' \ a' \ a''. \text{refl-on } A'' \ r'' \wedge (a' :: 'a, a'') \in r'' \longrightarrow a' \in A'' \wedge a'' \in A''$
using *refl-on-domain*
by *metis*
have $\forall A'' \ r''. \text{linear-order-on } (A'' :: 'a \text{ set}) \ r'' \longrightarrow \text{connex } A'' \ r''$
by (*simp add: lin-ord-imp-connex*)
hence *refl-A': refl-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 $b\text{-in-r: } \forall a'. a' \in A' \longrightarrow b = a' \vee (b, a') \in r' \vee (a', b) \in r'$
using *lin-ord-r*
unfolding *linear-order-on-def total-on-def*
by *metis*
have $b\text{-in-lim-B-r: } (b, b) \in \text{limit } B \ r'$
using *alt-b mem-Collect-eq singletonI*
unfolding *above-def*
by *metis*
have $b\text{-wins: } \{a'. (b, a') \in \text{limit } B \ r'\} = \{b\}$
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:  $\forall b' \in B. b \in \text{above } r' b'$ 
using subset-B-card b-in-r b-wins b-refl CollectI Product-Type.Collect-case-prodD
    unfolding above-def
    by fastforce
moreover have  $b \in \text{above } r' a$ 
    using a-pref-r-b pref-imp-in-above
    by metis
ultimately have b-wins:  $\forall a' \in A'. b \in \text{above } r' a'$ 
    using Diff-iff a empty-iff insert-iff
    by (metis (no-types))
hence  $\forall a' \in A'. a' \in \text{above } r' b \longrightarrow a' = b$ 
    using CollectD lin-ord-r lin-imp-antisym
    unfolding above-def antisym-def
    by metis
hence  $\forall a' \in A'. (a' \in \text{above } r' b) = (a' = b)$ 
    using b-wins
    by blast
moreover have above-b-in-A:  $\text{above } r' b \subseteq A'$ 
    unfolding above-def
    using refl-A' refl-A
    by auto
ultimately have  $\text{above } r' b = \{b\}$ 
    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:
 $\forall B' B'' r''. \text{linear-order-on } (B'' :: 'a \text{ set}) r'' \wedge B' \subseteq B''$ 
 $\longrightarrow \text{linear-order-on } B' (\text{limit } B' r'')$ 
    using limit-presv-lin-ord
    by metis
have  $\{a'. (b, a') \in \text{limit } B r'\} = \{b\}$ 

```

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
 $\forall A'' r''.$
 $\text{total-on } A'' r'' =$
 $(\forall a'. (a' :: 'a) \notin A''$
 $\quad \vee (\forall a''. a'' \notin A'' \vee a' = a'' \vee (a', a'') \in r'' \vee (a'', a') \in r''))$
unfolding *total-on-def*
by *metis*
hence
 $\forall a' a''.$
 $a' \in B \longrightarrow a'' \in B$
 $\longrightarrow a' = a'' \vee (a', a'') \in \text{limit } B \ r' \vee (a'', a') \in \text{limit } B \ r'$
using *limit-B*
by *simp*
hence $\forall a' \in B. b \in \text{above } r' \ a'$
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 $\forall a' \in A'. (a' \in \text{above } r' \ a) = (a' = a)$
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*: $\text{above } r' \ a \subseteq A'$
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  $\exists a \in A. \text{above } r \ a = \{a\}$ 
  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:  $\text{above } r \ a = \{a\}$  and
    above-b:  $\text{above } r \ b = \{b\}$ 
  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
     $\exists a' \in A. \text{above } r \ a' = \{a'\} \wedge (\forall a'' \in A. \text{above } r \ a'' = \{a''\} \longrightarrow a'' = a')$ 
    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

$r :: 'a \text{ Preference-Relation}$ **and**

$a :: 'a$

assumes $\text{above } r \ a = \{a\}$

shows $\text{rank } r \ a = 1$

using *assms*

by *simp*

lemma *rank-one-imp-above-one*:

fixes

$A :: 'a \text{ set}$ **and**

$r :: 'a \text{ Preference-Relation}$ **and**

$a :: 'a$

assumes

$\text{lin-ord: linear-order-on } A \ r$ **and**

$\text{rank-one: rank } r \ a = 1$

shows $\text{above } r \ a = \{a\}$

proof –

from *lin-ord*

have $\text{refl-on } A \ r$

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 $a \in \text{above } r \ a$

using *above-refl*

by *fastforce*

with *rank-one*

show $\text{above } r \ a = \{a\}$

using *card-1-singletonE rank.simps singletonD*

by *metis*

qed

theorem *above-rank*:

fixes

$A :: 'a \text{ set}$ **and**

$r :: 'a \text{ Preference-Relation}$ **and**

$a :: 'a$

assumes $\text{linear-order-on } A \ r$

shows $(\text{above } r \ a = \{a\}) = (\text{rank } r \ a = 1)$

using *assms above-one-imp-rank-one rank-one-imp-above-one*

by *metis*

lemma *rank-unique*:

fixes

$A :: 'a \text{ set}$ **and**

$r :: 'a \text{ Preference-Relation}$ **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 $\text{rank } r \ a \neq \text{rank } r \ b$

proof (*unfold rank.simps above-def, clarify*)

assume card-eq : $\text{card } \{a'. (a, a') \in r\} = \text{card } \{a'. (b, a') \in r\}$

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 $\text{a-in-A refl-r refl-onD}$

by (*metis (full-types)*)

obtain $p :: 'a \Rightarrow \text{bool}$ **where**

rel-b : $\forall y. p \ y = ((b, y) \in r)$

using *is-less-preferred-than.simps*

by *metis*

hence *finite* (*Collect* p)

using $\text{refl-r refl-on-domain fin-A rev-finite-subset mem-Collect-eq subsetI}$

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 $\text{card-eq card-gt-0-iff rel-refl-b}$

by *force*

moreover have *trans* r

using $\text{lin-ord lin-imp-trans}$

by *metis*

moreover have $(a, b) \in r \vee (b, a) \in r$

using $\text{lin-ord a-in-A b-in-A a-neq-b}$

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 $\text{card-eq above-trans card-seteq order-refl}$

unfolding *above-def*

```

    by metis
  hence (b, a) ∈ r
    using rel-refl-a sets-eq
  by blast
  hence (a, b) ∉ 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 ⊆ A
  unfolding above-def
  by auto

```

1.2.5 Lifting Property

```

definition equiv-rel-except-a :: 'a set ⇒ 'a Preference-Relation ⇒
  'a Preference-Relation ⇒ 'a ⇒ bool where
  equiv-rel-except-a A r r' a ≡
    linear-order-on A r ∧ linear-order-on A r' ∧ a ∈ A ∧
    (∀ a' ∈ A - {a}. ∀ b' ∈ A - {a}. (a' ≤r b') = (a' ≤r' b'))

```

```

definition lifted :: 'a set ⇒ 'a Preference-Relation ⇒ 'a Preference-Relation ⇒
  'a ⇒ bool where
  lifted A r r' a ≡
    equiv-rel-except-a A r r' a ∧ (∃ a' ∈ A - {a}. a ≤r a' ∧ a' ≤r' a)

```

```

lemma trivial-equiv-rel:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation
  assumes linear-order-on A r
  shows ∀ a ∈ 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:  $\exists b \in A - \{a\}. (a, b) \in r \wedge (b, a) \in r'$ 
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
 $A :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a \ a' :: 'a$
assumes
lifted-a: $\text{lifted } A \ r \ r' \ a$ **and**
a'-in-A-sub-a: $a' \in A - \{a\}$
shows $\text{above } r \ a' \subseteq \text{above } r' \ a' \cup \{a\}$
proof (*safe*)
fix $b :: 'a$
assume
b-in-above-r: $b \in \text{above } r \ a'$ **and**
b-not-in-above-s: $b \notin \text{above } r' \ a'$
have $\forall b' \in A - \{a\}. (b' \in \text{above } r \ a') = (b' \in \text{above } r' \ a')$
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
 $A \ A' :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes
lifted: $\text{lifted } A' \ r \ r' \ a$ **and**
subset: $A \subseteq A'$
shows $\text{limit } A \ r = \text{limit } A \ r' \vee \text{lifted } A \ (\text{limit } A \ r) \ (\text{limit } A \ r') \ a$
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*:
 $\forall a' \in A - \{a\}. \forall b' \in A - \{a\}.$

$((a', b') \in (\text{limit } A \ r)) = ((a', b') \in (\text{limit } A \ r'))$
using *DiffD1 limit-presv-prefs limit-rel-presv-prefs*
by *simp*
have *lin-ord-r-s*: *linear-order-on* A $(\text{limit } A \ r) \wedge \text{linear-order-on } A \ (\text{limit } A \ r')$
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 $(\text{limit } A \ r) \wedge \text{connex } A \ (\text{limit } A \ r')$
using *lifted subset limit-presv-lin-ord lin-ord-imp-connex*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
moreover have
 $\forall A'' \ r''. \text{connex } A'' \ r'' =$
 $(\text{limited } A'' \ r''$
 $\wedge (\forall b \ b'. (b :: 'a) \in A'' \longrightarrow b' \in A'' \longrightarrow (b \preceq_{r''} b' \vee b' \preceq_{r''} b)))$
unfolding *connex-def*
by (*simp add: Ball-def-raw*)
hence *limit-rel-r*:
 $\text{limited } A \ (\text{limit } A \ r)$
 $\wedge (\forall b \ b'. b \in A \wedge b' \in A \longrightarrow (b, b') \in \text{limit } A \ r \vee (b', b) \in \text{limit } A \ r)$
using *connex*
by *simp*
have *limit-imp-rel*: $\forall b \ b' \ A'' \ r''. (b :: 'a, b') \in \text{limit } A'' \ r'' \longrightarrow b \preceq_{r''} b'$
using *limit-rel-presv-prefs*
by *metis*
have *limit-rel-s*:
 $\text{limited } A \ (\text{limit } A \ r')$
 $\wedge (\forall b \ b'. b \in A \wedge b' \in A \longrightarrow (b, b') \in \text{limit } A \ r' \vee (b', b) \in \text{limit } A \ r')$
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  $\forall a' \in A - \{a\}. ((a, a') \in (\text{limit } A \ r)) = ((a, a') \in (\text{limit } A \ r'))$ 
    using DiffD1 limit-imp-rel limit-rel-r limit-rel-s a-in-A
    strict-pref-to-a not-worse
    by metis
  hence
     $\forall a' \in A - \{a\}. (let \ q = \text{limit } A \ r \ in \ a \preceq_q a') = (let \ q = \text{limit } A \ r' \ in \ a \preceq_q a')$ 
    by simp
  moreover have
     $\forall a' \in A - \{a\}. ((a', a) \in (\text{limit } A \ r)) = ((a', a) \in (\text{limit } A \ r'))$ 
    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  $(a, a) \in (\text{limit } A \ r) \wedge (a, a) \in (\text{limit } A \ r')$ 
    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
     $A \ A' :: 'a \text{ set}$  and
     $r \ r' :: 'a \text{ Preference-Relation}$  and
     $a :: 'a$ 
  assumes
    change:  $\text{equiv-rel-except-a } A' \ r \ r' \ a$  and
    subset:  $A \subseteq A'$  and
    not-in-A:  $a \notin A$ 
  shows  $\text{limit } A \ r = \text{limit } A \ r'$ 
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
     $A :: 'a \text{ set}$  and
     $r \ r' :: 'a \text{ Preference-Relation}$  and
     $a \ a' :: 'a$ 
  assumes
    lifted-a:  $\text{lifted } A \ r \ r' \ a$  and
    a'-above-a':  $\text{above } r \ a' = \{a'\}$  and
    fin-A:  $\text{finite } A$ 
  shows  $\text{above } r' \ a' = \{a'\} \vee \text{above } r' \ a = \{a\}$ 
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  $a \neq a'$ 
  thus ?thesis
  proof (cases)
    assume  $\text{above } r' \ a' = \{a'\}$ 
    thus ?thesis
      by simp
  next
    assume  $a' \text{-not-above-} a'$ :  $\text{above } r' \ a' \neq \{a'\}$ 
    have  $\forall a'' \in A. a'' \preceq_r a'$ 
    proof (safe)
      fix  $b :: 'a$ 
      assume  $y \text{-in-} A$ :  $b \in A$ 
      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  $y \text{-in-} A$   $a' \text{-above-} a'$  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 $\forall a'' \in A - \{a\}. \text{above } r' a'' \neq \{a''\}$
using *a'-not-above-a' empty-iff insert-iff pref-imp-in-above*
by *metis*
hence $\text{above } r' a = \{a\}$
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 $\text{above } r' a' = \{a'\} \vee \text{above } r' a = \{a\}$
by *simp*
qed
qed

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

theorem *lifted-above-winner-other:*
fixes
 $A :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a \ a' :: 'a$
assumes
lifted-a: *lifted* $A \ r \ r' \ a$ **and**
a'-above-a': $\text{above } r' \ a' = \{a'\}$ **and**
fin-A: *finite* A **and**
a-not-a': $a \neq a'$
shows $\text{above } r \ a' = \{a'\}$
proof (*rule ccontr*)
assume *not-above-x:* $\text{above } r \ a' \neq \{a'\}$
then obtain $b :: 'a$ **where**
 $\text{above } r \ b = \{b\}$

```

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} ∨ above r' a = {a}
using lifted-a fin-A lifted-above-winner-alts
by metis
moreover have  $\forall a''. \text{above } r' a'' = \{a''\} \longrightarrow a'' = a'$ 
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 $u = (0, 0, \dots, 0)$.
- triangle inequality: $N(u + v) \leq N(u) + N(v)$.

1.3.1 Definition

```

type-synonym Norm = ereal list  $\Rightarrow$  ereal

```

```

definition norm :: Norm  $\Rightarrow$  bool where
  norm  $n \equiv \forall (x::\text{ereal list}). n\ x \geq 0 \wedge (\forall\ i < \text{length } x. (x!i = 0) \longrightarrow n\ x = 0)$ 

```

1.3.2 Auxiliary Lemmas

```

lemma sum-over-image-of-bijection:
  fixes
     $A :: 'a \text{ set}$  and
     $A' :: 'b \text{ set}$  and
     $f :: 'a \Rightarrow 'b$  and
     $g :: 'a \Rightarrow \text{ereal}$ 
  assumes bij-betw  $f$   $A$   $A'$ 
  shows  $(\sum a \in A. g\ a) = (\sum a' \in A'. g\ (\text{the-inv-into } A\ f\ a'))$ 
  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
     $A :: 'a \text{ set}$  and
     $A' :: 'b \text{ set}$  and
     $x :: \text{nat}$ 
  assume
    suc-x:  $\text{Suc } x = \text{card } A$  and
    bij-A-A': bij-betw  $f$   $A$   $A'$ 
  hence card-A'-from-x:  $\text{card } A' = \text{Suc } x$ 
    using bij-betw-same-card
    by metis
  have x-lt-card-A:  $x < \text{card } A$ 
    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:  $\text{insert } a\ (A - \{a\}) = A$ 
    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*: $\text{card } (A - \{a\}) = x$
 using *suc-x a-in-A*
 by *simp*
 ultimately have *card-A'-sub-f-eq-x*: $\text{card } (A' - \{f\ a\}) = x$
 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 $\dots = (\sum a' \in (A' - \{f\ a\}).$
 $g\ (the_inv_into\ A\ f\ a')) + g\ (the_inv_into\ A\ f\ (f\ a))$
 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 $(\sum a \in A. g\ a) = (\sum a' \in A'. g\ (the_inv_into\ A\ f\ a'))$
 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
 l-one $x = (\sum i < \text{length } x. |x[i]|)$

1.3.4 Properties

definition *symmetry* :: *Norm* \Rightarrow *bool* where
 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 $l\ l' :: \text{ereal list}$
 assume *perm*: $l <\sim\sim> l'$
 then obtain $\pi :: \text{nat} \Rightarrow \text{nat}$
 where
 perm $_{\pi}$: π permutes $\{.. < \text{length } l\}$ and
 l_{π} : *permute-list* $\pi\ l = l'$
 using *mset-eq-permutation*
 by *metis*
 hence $(\sum i < \text{length } l. |l[i]|) = (\sum i < \text{length } l. |l'(\pi\ i)|)$
 using *permute-list-nth*
 by *fastforce*

```

also have ... = sum ( $\lambda i. |l(\pi \ i)|$ ) {0 ..< length l}
  using lessThan-atLeast0
  by presburger
also have ( $\lambda i. |l(\pi \ i)|$ ) = (( $\lambda i. |l!i|$ )  $\circ \pi$ )
  by fastforce
also have sum (( $\lambda i. |l!i|$ )  $\circ \pi$ ) {0 ..< length l} =
  sum ( $\lambda i. |l!i|$ ) {0 ..< length l}
  using permπ atLeast-upt set-upt sum.permute
  by metis
also have ... = ( $\sum i < \text{length } l. |l!i|$ )
  using atLeast0LessThan
  by presburger
finally have ( $\sum i < \text{length } l. |l!i|$ ) = ( $\sum i < \text{length } l. |l!i|$ )
  by metis
moreover have length l = length l'
  using perm perm-length
  by metis
ultimately show l-one l = l-one l'
  using l-one.elims
  by metis
qed

end

```

1.4 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-equals-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  $\Rightarrow$  ('r Result)  $\Rightarrow$  bool and
    limit :: 'a set  $\Rightarrow$  'r set  $\Rightarrow$  'r set
  assumes  $\forall$  (A :: 'a set) (r :: 'r Result).
    (set-equals-partition (limit A UNIV) r  $\wedge$  disjoint3 r)  $\longrightarrow$  well-formed A r

```

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

```

fun (in result) limitR :: 'a set  $\Rightarrow$  'r Result  $\Rightarrow$  'r Result where
  limitR A (e, r, d) = (limit A e, limit A r, limit A d)

```

```

abbreviation elect-r :: 'r Result  $\Rightarrow$  'r set where
  elect-r r  $\equiv$  fst r

```

```

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

```

```

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

```

end

1.5 Preference Profile

```

theory Profile
  imports Preference-Relation

```


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 $(\text{'a}, \text{'v}) \text{ Profile} = \text{'v} \Rightarrow (\text{'a} \text{ Preference-Relation})$

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

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

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

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

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

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

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

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

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

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

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

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

— 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 $p \ E = \text{card } \{v \in (\text{voters-}\mathcal{E} \ E). (\text{profile-}\mathcal{E} \ E) \ v = p\}$

1.5.2 Vote Count

lemma *vote-count-sum*:

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

using *partition assms*
by (*simp add: finite-UnionD*)
moreover have

$$\begin{aligned} & \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\} = \\ & \quad \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \\ & \cup \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\} \end{aligned}$$
by *blast*
moreover have

$$\begin{aligned} & \{\} = \\ & \quad \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \\ & \cap \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\} \end{aligned}$$
by *blast*
ultimately have

$$\begin{aligned} & \text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\} = \\ & \quad \text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \\ & + \text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\} \end{aligned}$$
using *sum.union-disjoint[of*

$$\begin{aligned} & \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \\ & \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\} \end{aligned}$$
by *simp*
moreover have

$$\begin{aligned} & \forall X \in \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}. \\ & \text{card } X = 0 \end{aligned}$$
using *card-eq-0-iff*
by *fastforce*
ultimately have *card-eq-sum:*

$$\begin{aligned} & \text{card } (\text{voters-}\mathcal{E} \ E) = \\ & \quad \text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \\ & \quad \mid p. p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \end{aligned}$$
using *card-eq-sum'*
by *simp*
have

$$\begin{aligned} & \text{inj-on } (\lambda p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}) \\ & \quad \{p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \end{aligned}$$
unfolding *inj-on-def*
by *blast*
moreover have

$$\begin{aligned} & (\lambda p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}) \\ & \quad ' \{p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \\ & \subseteq \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \end{aligned}$$

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

```

      card {v ∈ voters- $\mathcal{E}$  E. profile- $\mathcal{E}$  E v = x} = 0
    using card-eq-0-iff
    by fastforce
  ultimately show
    (∑ p ∈ UNIV. card {v ∈ voters- $\mathcal{E}$  E. profile- $\mathcal{E}$  E v = p}) =
      card (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 ⇒ 'v) ⇒ ('a, 'v) Election ⇒ ('a, 'v) Election **where**
 rename π (A, V, p) = (A, π ' V, p ∘ (the-inv π))

lemma rename-sound:

```

fixes
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  π :: 'v ⇒ 'v
assumes
  prof: profile V A p and
  renamed: (A, V', q) = rename π (A, V, p) and
  bij-perm: bij π
shows profile V' A q
proof (unfold profile-def, safe)
  fix v' :: 'v
  assume v' ∈ V'
  moreover have V' = π ' V
    using renamed
    by simp
  ultimately have ((the-inv π) v') ∈ 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

```

```

     $\pi :: 'v \Rightarrow 'v$ 
assumes
  profile  $V\ A\ p$  and
   $(A, V', q) = \text{rename } \pi\ (A, V, p)$  and
  bij  $\pi$ 
shows profile  $V'\ A\ q$ 
using assms rename-sound
by metis

lemma rename-finite:
fixes
   $A :: 'a\ \text{set}$  and
   $V :: 'v\ \text{set}$  and
   $p :: ('a, 'v)\ \text{Profile}$  and
   $\pi :: 'v \Rightarrow 'v$ 
assumes
  finite  $V$  and
   $(A, V', q) = \text{rename } \pi\ (A, V, p)$  and
  bij  $\pi$ 
shows finite  $V'$ 
using assms
by simp

lemma rename-inv:
fixes
   $\pi :: 'v \Rightarrow 'v$  and
   $A :: 'a\ \text{set}$  and
   $V :: 'v\ \text{set}$  and
   $p :: ('a, 'v)\ \text{Profile}$ 
assumes bij  $\pi$ 
shows  $\text{rename } \pi\ (\text{rename } (\text{the-inv } \pi)\ (A, V, p)) = (A, V, p)$ 
proof –
have  $\text{rename } \pi\ (\text{rename } (\text{the-inv } \pi)\ (A, V, p)) =$ 
   $(A, \pi\ ‘\ (\text{the-inv } \pi)\ ‘\ V, p \circ (\text{the-inv } (\text{the-inv } \pi)) \circ (\text{the-inv } \pi))$ 
by simp
moreover have  $\pi\ ‘\ (\text{the-inv } \pi)\ ‘\ V = V$ 
using assms
by (simp add: f-the-inv-into-f-bij-betw image-comp)
moreover have  $(\text{the-inv } (\text{the-inv } \pi)) = \pi$ 
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  $\pi \circ (\text{the-inv } \pi) = \text{id}$ 
using assms f-the-inv-into-f-bij-betw
by fastforce
ultimately show  $\text{rename } \pi\ (\text{rename } (\text{the-inv } \pi)\ (A, V, p)) = (A, V, p)$ 
by (simp add: rewriteR-comp-comp)
qed

```

```

lemma rename-inj:
  fixes  $\pi :: 'v \Rightarrow 'v$ 
  assumes bij  $\pi$ 
  shows inj (rename  $\pi$ )
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
     $p \circ the\_inv\ \pi = p' \circ the\_inv\ \pi$  and
     $\pi\ 'V = \pi\ 'V'$ 
  thus
     $v \in V \implies v \in V'$  and
     $v \in V' \implies v \in V$  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  $\pi :: 'v \Rightarrow 'v$ 
  assumes bij  $\pi$ 
  shows
    rename  $\pi\ 'well-formed-elections = well-formed-elections$  and
    rename  $\pi\ 'finite-elections = finite-elections$ 
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  $\pi$ )  $(A, V, p) \in well-formed-elections$ 
    using assms bij-betw-the-inv-into rename-sound
    unfolding well-formed-elections-def
    by fastforce
  thus  $(A, V, p) \in rename\ \pi\ 'well-formed-elections$ 
    using assms image-eqI rename-inv
    by metis
  assume  $(A', V', p') = rename\ \pi\ (A, V, p)$ 
  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) ∈ finite-elections
hence rename (the-inv π) (A, V, p) ∈ finite-elections
  using assms bij-betw-the-inv-into rename-prof rename-finite
  unfolding finite-elections-def
  by fastforce
thus (A, V, p) ∈ 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') ∈ 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 ⇒ ('a, 'v) Profile ⇒
  ('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 ⇒ 'y ⇒ 'z and
  g :: 'x ⇒ 'x and
  h :: 'y ⇒ 'y and
  l :: 'x list and
  l' :: 'y list
shows map2 f (map g l) (map h l') = map2 (λ x y. f (g x) (h y)) l l'
proof -
  have map2 f (map g l) (map h l') =
    map (λ (x, y). f x y) (map (λ (x, y). (g x, h y)) (zip l l'))
  using zip-map-map
  by metis
also have ... = map2 (λ x y. f (g x) (h y)) l l'
  by auto
finally show ?thesis
  by presburger
qed

```

lemma *to-list-simp*:

```

fixes

```



```

    i :: nat and
    V :: 'v :: linorder set and
    p :: ('a, 'v) Profile
  assumes i < card V
  shows (to-list V p)!i = p ((sorted-list-of-set V)!i)
proof -
  have (to-list V p)!i = (map p (sorted-list-of-set V))!i
    by simp
  thus ?thesis
    using assms
    by simp
qed

```

```

lemma to-list-comp:
  fixes
    V :: 'v :: linorder set and
    p :: ('a, 'v) Profile and
    f :: 'a rel  $\Rightarrow$  'a rel
  shows to-list V (f  $\circ$  p) = map f (to-list V p)
  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* $V!(\text{card } \{v \in V. v < x\}) = x$
proof –
let $?c = \text{card } \{v \in V. v < x\}$ **and**
 $?set = \{v \in V. v < x\}$
have $\forall v \in V. \exists n. n < \text{card } V \wedge (\text{sorted-list-of-set } V!n) = v$
using *length-sorted-list-of-set sorted-list-of-set-unique in-set-conv-nth fin-V*
by *metis*
then obtain $\varphi :: 'v \Rightarrow \text{nat}$ **where**
 $\text{index-}\varphi: \forall v \in V. \varphi v < \text{card } V \wedge (\text{sorted-list-of-set } V!(\varphi v)) = v$
by *metis*
– $\varphi x = ?c$, i.e., $\varphi x \geq ?c$ and $\varphi x \leq ?c$
let $?i = \varphi x$
have $\text{inj-}\varphi: \text{inj-on } \varphi V$
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 $?i \geq \text{card } \{\varphi v \mid v. v \in ?set\}$
using *set-card-upper-bound*
by *simp*
also have $\text{card } \{\varphi v \mid v. v \in ?set\} = ?c$
using *inj-φ*
by (*simp add: card-image inj-on-subset setcompr-eq-image*)
finally have *geq: ?c ≤ ?i*
by *simp*
have *sorted-φ:*
 $\forall i < \text{card } V. \forall j < \text{card } V. i < j$
 $\longrightarrow (\text{sorted-list-of-set } V!i) < (\text{sorted-list-of-set } V!j)$
by (*simp add: sorted-wrt-nth-less*)
have *leq: ?i ≤ ?c*
proof (*rule ccontr, cases ?c < card V*)
case *True*
let $?A = \lambda j. \{\text{sorted-list-of-set } V!j\}$
assume $\neg ?i \leq ?c$
hence $?c < ?i$
by *simp*
hence $\forall j \leq ?c. \text{sorted-list-of-set } V!j \in V \wedge \text{sorted-list-of-set } V!j < x$
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 $\{\text{sorted-list-of-set } V!j \mid j. j \leq ?c\} \subseteq \{v \in V. v < x\}$
 by *blast*
 also have $\{\text{sorted-list-of-set } V!j \mid j. j \leq ?c\} =$
 $\{\text{sorted-list-of-set } V!j \mid j. j \in \{0 \dots (?c + 1)\}\}$
 using *add.commute*
 by *auto*
 also have $\{\text{sorted-list-of-set } V!j \mid j. j \in \{0 \dots (?c + 1)\}\} =$
 $(\bigcup j \in \{0 \dots (?c + 1)\}. \{\text{sorted-list-of-set } V!j\})$
 by *blast*
 finally have *subset*: $(\bigcup j \in \{0 \dots (?c + 1)\}. ?A j) \subseteq \{v \in V. v < x\}$
 by *simp*
 have $\forall i \leq ?c. \forall j \leq ?c.$
 $i \neq j \longrightarrow \text{sorted-list-of-set } V!i \neq \text{sorted-list-of-set } V!j$
 using *True*
 by (*simp add: nth-eq-iff-index-eq*)
 hence $\forall i \in \{0 \dots (?c + 1)\}. \forall j \in \{0 \dots (?c + 1)\}.$
 $(i \neq j \longrightarrow \{\text{sorted-list-of-set } V!i\} \cap \{\text{sorted-list-of-set } V!j\} = \{\})$
 by *fastforce*
 hence *disjoint-family-on* $?A \{0 \dots (?c + 1)\}$
 unfolding *disjoint-family-on-def*
 by *simp*
 moreover have $\forall j \in \{0 \dots (?c + 1)\}. \text{card } (?A j) = 1$
 by *simp*
 ultimately have
 $\text{card } (\bigcup j \in \{0 \dots (?c + 1)\}. ?A j) = (\sum j \in \{0 \dots (?c + 1)\}. 1)$
 using *card-UN-disjoint'*
 by *fastforce*
 hence $\text{card } (\bigcup j \in \{0 \dots (?c + 1)\}. ?A j) = ?c + 1$
 by *simp*
 hence $?c + 1 \leq ?c$
 using *subset card-mono fin-img*
 by (*metis (no-types, lifting)*)
 thus *False*
 by *simp*
 next
 case *False*
 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
 $\pi :: 'v :: \text{linorder} \Rightarrow 'v$ and
 $V :: 'v \text{ set}$ and

```

    p :: ('a, 'v) Profile
  assumes bij  $\pi$ 
  shows
    let  $\varphi = (\lambda i. \text{card } \{v \in \pi \text{ ' } V. v < \pi ((\text{sorted-list-of-set } V)!i)\})$ 
    in (to-list V p) = permute-list  $\varphi$  (to-list ( $\pi \text{ ' } V$ ) ( $\lambda x. p (\text{the-inv } \pi x)$ ))
  proof (cases finite V)
    case False
    — If V is infinite, both lists are empty.
    hence to-list V p = []
    by simp
    moreover have infinite ( $\pi \text{ ' } V$ )
    using False assms bij-betw-finite bij-betw-subset top-greatest
    by metis
    hence to-list ( $\pi \text{ ' } V$ ) ( $\lambda x. p (\text{the-inv } \pi x)$ ) = []
    by simp
    ultimately show ?thesis
    by simp
  next
  case True
  let
    ?q =  $\lambda x. p (\text{the-inv } \pi x)$  and
    ?img =  $\pi \text{ ' } V$  and
    ?n = length (to-list V p) and
    ?perm =  $\lambda i. \text{card } \{v \in \pi \text{ ' } V. v < \pi ((\text{sorted-list-of-set } V)!i)\}$ 
    — 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 = card V
    by simp
  also have card-length-img: length (to-list ?img ?q) = card ?img
    using True
    by simp
  finally have eq-length: length (to-list ?img ?q) = ?n
    by simp
  show ?thesis
  proof (unfold Let-def permute-list-def, rule nth-equalityI)
    — The lists have equal lengths.
    show
      length (to-list V p) =
        length (map
          ( $\lambda i. \text{to-list ?img ?q}!(\text{card } \{v \in ?img. v < \pi (\text{sorted-list-of-set } V)!i\})$ )
          [0 ..< length (to-list ?img ?q)])
        using eq-length
        by simp
  next
  — The ith entries of the lists coincide.
  fix i :: nat

```

assume *in-bnds*: $i < ?n$
let $?c = \text{card } \{v \in ?img. v < \pi (\text{sorted-list-of-set } V!i)\}$
have $\text{map } (\lambda i. (\text{to-list } ?img ?q)! ?c) [0 ..< ?n]!i =$
 $p ((\text{sorted-list-of-set } V)!i)$
proof –
have $\forall v. v \in ?img \longrightarrow \{v' \in ?img. v' < v\} \subseteq ?img - \{v\}$
by *blast*
moreover have *elem-of-img*: $\pi (\text{sorted-list-of-set } V!i) \in ?img$
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
 $\{v \in ?img. v < \pi (\text{sorted-list-of-set } V!i)\}$
 $\subseteq ?img - \{\pi (\text{sorted-list-of-set } V!i)\}$
by *simp*
hence $\{v \in ?img. v < \pi (\text{sorted-list-of-set } V!i)\} \subset ?img$
using *elem-of-img*
by *blast*
moreover have *img-card-eq-V-length*: $\text{card } ?img = ?n$
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*:
 $\text{map } (\lambda i. \text{to-list } ?img ?q! ?c) [0 ..< ?n]!i = \text{to-list } ?img ?q! ?c$
using *in-bnds*
by *simp*
also have *img-list-card-eq-inv-img-list*:
 $\text{to-list } ?img ?q! ?c = ?q ((\text{sorted-list-of-set } ?img)! ?c)$
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*:
 $(\text{sorted-list-of-set } ?img)! ?c = \pi (\text{sorted-list-of-set } V!i)$
using *True elem-of-img sorted-list-of-set-nth-equals-card*
by *blast*
finally show *?thesis*
using *assms bij-betw-imp-inj-on the-inv-f-f*
img-list-map img-card-eq-img-list-i
img-list-card-eq-inv-img-list
by *metis*
qed
also have *to-list V p!i* = $p ((\text{sorted-list-of-set } V)!i)$
using *True in-bnds*
by *simp*
finally show *to-list V p!i* =
 $\text{map } (\lambda i. (\text{to-list } ?img ?q)! (\text{card } \{v \in ?img. v < \pi (\text{sorted-list-of-set } V!i)\}))$
 $[0 ..< \text{length } (\text{to-list } ?img ?q)]!i$
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* | *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*) *Profile* **and**
 a :: '*a*
shows {*prefer-count V p a a'* | *a'*. *a'* \in *A* - {*a*}} =
 (*prefer-count V p a*) '*(A - {a})*
by *blast*

lemma *pref-count*:

fixes
 A :: '*a set* **and**
 V :: '*v set* **and**
 p :: ('*a*, '*v*) *Profile* **and**

```

  a b :: 'a
assumes
  prof: profile V A p and
  fin: finite V and
  a-in-A: a ∈ A and
  b-in-A: b ∈ A and
  neg: a ≠ b
shows prefer-count V p a b = card V - (prefer-count V p b a)
proof -
have ∀ v ∈ V. ¬ (let r = (p v) in (b ≤r a)) ⟶ (let r = (p v) in (a ≤r b))
  using a-in-A b-in-A prof lin-ord-imp-connex
  unfolding profile-def connex-def
  by metis
moreover have ∀ v ∈ V. ((b, a) ∈ (p v) ⟶ (a, b) ∉ (p v))
  using antisymD neg lin-imp-antisym prof
  unfolding profile-def
  by metis
ultimately have
  {v ∈ V. (let r = (p v) in (b ≤r a))} =
  V - {v ∈ V. (let r = (p v) in (a ≤r b))}
  by auto
thus ?thesis
  by (simp add: card-Diff-subset Collect-mono fin)
qed

```

lemma pref-count-sym:

```

fixes
  p :: ('a, 'v) Profile and
  V :: 'v set and
  a b c :: 'a
assumes
  pref-count-ineq: prefer-count V p a c ≥ prefer-count V p c b and
  prof: profile V A p and
  a-in-A: a ∈ A and
  b-in-A: b ∈ A and
  c-in-A: c ∈ A and
  a-neq-c: a ≠ c and
  c-neq-b: c ≠ b
shows prefer-count V p b c ≥ prefer-count V p c a
proof (cases finite V)
case True
moreover have
  prefer-count V p c a ∈ ℕ and
  prefer-count V p b c ∈ ℕ
  unfolding Nats-def
  using True of-nat-eq-enat
  by (simp, simp)
moreover have prefer-count V p c a ≤ 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)
    ≤ 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

```


$V :: 'v \text{ set}$
assumes $\text{wins } V a p b$ — This already implies that V is finite.
shows $\neg \text{wins } V b p a$
using *assms*
by *simp*

lemma *wins-irreflex*:
fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$ **and**
 $V :: 'v \text{ set}$
shows $\neg \text{wins } V a p a$
using *wins-antisym*
by *metis*

1.5.6 Condorcet Winner

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

lemma *cond-winner-unique-eq*:
fixes
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a b :: 'a$
assumes
 $\text{condorcet-winner } V A p a$ **and**
 $\text{condorcet-winner } V A p b$
shows $b = a$
proof (*rule ccontr*)
assume $b \neq a$
hence $\text{wins } V b p a$
using *insert-Diff insert-iff assms*
by *simp*
hence $\neg \text{wins } V a p b$
by (*simp add: wins-antisym*)
moreover have $\text{wins } V a p b$
using *Diff-iff b-neq-a singletonD assms*
by *auto*
ultimately show *False*
by *simp*
qed

lemma *cond-winner-unique*:
fixes
 $A :: 'a \text{ set}$ **and**

```

    p :: ('a, 'v) Profile and
    a :: 'a
  assumes condorcet-winner V A p a
  shows {a' ∈ 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 ∈ 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  $\subseteq$  B
  shows profile V A (limit-profile A p)
proof (unfold profile-def)
  have  $\forall v \in V. \text{linear-order-on } A (\text{limit } A (p \ v))$ 
    using assms limit-presv-lin-ord
    unfolding profile-def
    by metis
  thus  $\forall v \in V. \text{linear-order-on } A ((\text{limit-profile } A \ p) \ v)$ 
    by simp
qed

```

1.5.8 Lifting Property

definition *equiv-prof-except-a* :: 'v set \Rightarrow 'a set \Rightarrow ('a, 'v) Profile \Rightarrow ('a, 'v) Profile \Rightarrow 'a \Rightarrow bool **where**

equiv-prof-except-a V A p p' a \equiv

profile V A p \wedge *profile* V A p' \wedge a \in A \wedge

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

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

$\wedge (\forall v \in V. \neg \text{Preference-Relation.lifted } A (p \ v) (p' \ v) \ a \longrightarrow (p \ v) = (p' \ v))$

$\wedge (\exists v \in V. \text{Preference-Relation.lifted } A (p \ v) (p' \ v) \ a)$

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 ∈ A
    using assms
    unfolding lifted-def
    by (metis, metis, metis)
next
fix v :: 'v
assume v ∈ 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 ⊆ A' and
  not-in-A: a ∉ A
shows ∀ v ∈ 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 ∈ 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  $(\forall v \in V. \text{limit-profile } A \ p \ v = \text{limit-profile } A \ p' \ v)$ 
     $\vee \text{lifted } V \ A \ (\text{limit-profile } A \ p) \ (\text{limit-profile } A \ p') \ a$ 
proof (cases  $a \in A$ )
  case True
  have  $\forall v \in V. \text{Preference-Relation.lifted } A' \ (p \ v) \ (p' \ v) \ a \vee (p \ v) = (p' \ v)$ 
    using lifted-a
    unfolding lifted-def
    by metis
  hence one:
     $\forall v \in V.$ 
       $\text{Preference-Relation.lifted } A \ (\text{limit } A \ (p \ v)) \ (\text{limit } A \ (p' \ v)) \ a \vee$ 
       $(\text{limit } A \ (p \ v)) = (\text{limit } A \ (p' \ v))$ 
    using limit-lifted-imp-eq-or-lifted subset
    by metis
  thus ?thesis
proof (cases  $\forall v \in V. \text{limit } A \ (p \ v) = \text{limit } A \ (p' \ v)$ )
  case True
  thus ?thesis
    by simp
next
  case False
  let ?p =  $\text{limit-profile } A \ p$ 
  let ?q =  $\text{limit-profile } A \ p'$ 
  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
     $\exists v \in V. \text{Preference-Relation.lifted } A \ (?p \ v) \ (?q \ v) \ a$ 
    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  $\Rightarrow$  'a Result  $\Rightarrow$  bool where
  well-formed-SCF A res = (disjoint3 res  $\wedge$  set-equals-partition A res)
```

```
fun limit-SCF :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set where
  limit-SCF A r = A  $\cap$  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 ∈ r and
    a ∈ e
  moreover have
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧ (r ∩ d = {}) ∧ (e ∪ r ∪ d = A)
  using assms
  by simp
  ultimately show False
  by auto
next
  fix a :: 'a
  assume
    a ∈ r and
    a ∈ d
  moreover have
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧ (r ∩ d = {}) ∧ (e ∪ r ∪ 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-SCF A (e, r, d) and
    fin-A: finite A
  shows card A = card e + card r + card d
proof -
  have e ∪ r ∪ d = A
  using wf-result
  by simp
  moreover have (e ∩ r = {}) ∧ (e ∩ d = {}) ∧ (r ∩ 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-SCF A r
  shows defer-r r ⊆ A

```

```

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

```

```

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

```

```

lemma reject-subset:
  fixes
     $A :: 'a\ \text{set}$  and
     $r :: 'a\ \text{Result}$ 
  assumes well-formed-SCF  $A\ r$ 
  shows  $\text{reject-}r\ r \subseteq A$ 
proof (safe)
  fix  $a :: 'a$ 
  assume  $a \in \text{reject-}r\ r$ 

```



```

moreover obtain
   $f :: 'a \text{ Result} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$  and
   $g :: 'a \text{ Result} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ Result}$  where
   $A = f \ r \ A \wedge r = g \ r \ A \wedge \text{disjoint3} \ (g \ r \ A) \wedge \text{set-equals-partition} \ (f \ r \ A) \ (g \ r \ A)$ 
using assms
by simp
moreover have
   $\forall \ p. \exists \ e \ r \ d. \text{set-equals-partition} \ A \ p \longrightarrow (e, r, d) = p \wedge e \cup r \cup d = A$ 
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-SWF ::  $'a \text{ set} \Rightarrow ('a \text{ Preference-Relation}) \text{ Result} \Rightarrow \text{bool}$  where
  well-formed-SWF  $A \text{ res} = (\text{disjoint3} \ \text{res} \wedge$ 
     $\text{set-equals-partition} \ \{r. \text{linear-order-on} \ A \ r\} \ \text{res})$ 

```

```

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

```

```

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 *SCF-result: result well-formed-SCF limit-SCF*

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

$\lambda\ A\ r.\ set\ equals\ partition\ (Pow\ A)\ r\ \wedge\ disjoint3\ r$
 $\lambda\ A\ rs.\ \{r \cap A \mid r.\ r \in rs\}$
proof (*unfold-locales, safe*)
fix
 $A :: 'b\ set\ \text{and}$
 $e\ r\ d :: 'b\ set\ set$
assume $set\ equals\ partition\ \{r \cap A \mid r.\ r \in UNIV\}\ (e,\ r,\ d)$
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
 $A :: 'a\ set\ \text{and}$
 $e\ r\ d :: ('a\ Preference\ Relation)\ set$
assume
 $set\ equals\ partition\ (limit\ SWF\ A\ UNIV)\ (e,\ r,\ d)\ \text{and}$
 $disjoint3\ (e,\ r,\ d)$
moreover have
 $limit\ SWF\ A\ UNIV = \{limit\ A\ r' \mid r'.\ linear\ order\ on\ A\ (limit\ A\ r')\}$
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  $\forall (a, b) \in r'. (a, b) \in \text{limit } A \ r'$ 
    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  $\exists x. r' = \text{limit } A \ x \wedge \text{linear-order-on } A \ (\text{limit } A \ x)$ 
    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 = ( $\lambda x. \text{if } (x \in s) \text{ then } (f \ x) \text{ else undefined}$ )

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

```

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-rel r t s) “ t  $\subseteq$  t)

fun action-induced-rel :: 'x set  $\Rightarrow$  'y set  $\Rightarrow$  ('x, 'y) binary-fun  $\Rightarrow$  'y rel where

```

$action\text{-}induced\text{-}rel\ s\ t\ \varphi = \{(y, y').\ y \in t \wedge (\exists\ x \in s.\ \varphi\ x\ y = y')\}$

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

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

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

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**
 $is\text{-}symmetry\ f\ (Invariance\ r) = (\forall\ x.\ \forall\ y.\ (x, y) \in r \longrightarrow f\ x = f\ y) \mid$
 $is\text{-}symmetry\ f\ (Equivariance\ s\ \tau) = (\forall\ (\varphi, \psi) \in \tau.\ \forall\ x \in s.\ f\ (\varphi\ x) = \psi\ (f\ x))$

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

1.9.4 Auxiliary Lemmas

lemma *un-left-inv-singleton-set-system*: $\bigcup \circ singleton\text{-}set\text{-}system = id$

proof

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

```

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

```

1.9.5 Rewrite Rules

```

theorem rewrite-invar-as-equivar:
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $s :: 'x \text{ set}$  and
     $t :: 'z \text{ set}$  and
     $\varphi :: ('z, 'x) \text{ binary-fun}$ 
  shows  $\text{is-symmetry } f (\text{Invariance } (\text{action-induced-rel } t \ s \ \varphi)) =$ 
     $\text{is-symmetry } f (\text{action-induced-equivariance } t \ s \ \varphi (\lambda \ g. \ \text{id}))$ 
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
     $\forall \ x \ y. (x, y) \in \text{action-induced-rel } t \ s \ \varphi \longrightarrow f \ x = f \ y$ 
  moreover with this have  $(x, \varphi \ g \ x) \in \text{action-induced-rel } t \ s \ \varphi$ 
  unfolding action-induced-rel.simps
  by blast
  ultimately show  $f (\varphi \ g \ x) = \text{id } (f \ x)$ 
  by simp
next

```

```

fix  $x\ y :: 'x$ 
assume
  equivar:
     $\forall (\varphi, \psi) \in \{(\varphi\ g, id) \mid g. g \in t\}. \forall x \in s. f\ (\varphi\ x) = \psi\ (f\ x)$  and
     $rel: (x, y) \in action\text{-}induced\text{-}rel\ t\ s\ \varphi$ 
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
     $\varphi :: ('z, 'x)\ binary\text{-}fun$ 
  shows is-symmetry  $f\ (Invariance\ (action\text{-}induced\text{-}rel\ s\ t\ \varphi)) =$ 
     $(\forall x \in s. \forall y \in t. f\ y = f\ (\varphi\ x\ y))$ 
proof (safe)
  fix
     $y :: 'x$  and
     $x :: 'z$ 
  assume
    is-symmetry  $f\ (Invariance\ (action\text{-}induced\text{-}rel\ s\ t\ \varphi))$  and
     $y \in t$  and
     $x \in s$ 
  moreover from this have  $(y, \varphi\ x\ y) \in action\text{-}induced\text{-}rel\ s\ t\ \varphi$ 
  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
     $\forall (x, y) \in action\text{-}induced\text{-}rel\ s\ t\ \varphi. x \in t \wedge (\exists z \in s. y = \varphi\ z\ x)$ 
  by auto
  ultimately show is-symmetry  $f\ (Invariance\ (action\text{-}induced\text{-}rel\ s\ t\ \varphi))$ 
  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 φ ψ) =
 $(\forall x \in s. \forall y \in t. f (\varphi x y) = \psi x (f y))$

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**

$x \in \text{carrier } m$ **and**

$y \in \text{carrier } m$ **and**

group-action m s φ

shows $\varphi (x \otimes_m y) \text{ ` } t = \varphi x \text{ ` } \varphi y \text{ ` } t$

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

$\varphi (x \otimes_m y) z \in \varphi x \text{ ` } \varphi y \text{ ` } t$ **and**

$\varphi x (\varphi y z) \in \varphi (x \otimes_m y) \text{ ` } t$

using *z-in-t*

by (*blast, force*)

qed

lemma *rewrite-carrier:* $\text{carrier } (\text{BijGroup } UNIV) = \{f'. \text{bij } f'\}$

unfolding *BijGroup-def Bij-def*

by *simp*

lemma *universal-set-carrier-imp-bij-group:*

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

assumes $f \in \text{carrier } (\text{BijGroup } UNIV)$

shows *bij* f

using *rewrite-carrier assms*

by *blast*

lemma *rewrite-sym-group*:

fixes

$f\ g :: 'a \Rightarrow 'a$ **and**

$s :: 'a\ set$

assumes

$f \in \text{carrier } (BijGroup\ s)$ **and**

$g \in \text{carrier } (BijGroup\ s)$

shows

rewrite-mult: $f \otimes BijGroup\ s\ g = \text{extensional-continuation } (f \circ g)\ s$ **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* $f\ s\ x = \text{extensional-continuation } f\ t\ x$

using *assms*

unfolding *subset-iff*

by *simp*

lemma *rel-ind-by-coinciding-action-on-subset-eq-restr*:

fixes

$\varphi\ \psi :: ('a, 'b)\ \text{binary-fun}$ **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* $s\ u\ \psi = \text{restricted-rel } (\text{action-induced-rel } s\ t\ \varphi)\ u\ UNIV$

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
     $g \in s$  and
     $x \in u$ 
  hence  $\varphi g x = \psi g x$ 
    using assms
    by simp
  thus  $\exists g' \in s. \varphi g' x = \psi g x$ 
    using  $\langle g \in s \rangle$ 
    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
     $g \in s$  and
     $x \in u$ 
  hence  $\psi g x = \varphi g x$ 
    using assms
    by simp
  thus  $\exists g' \in s. \psi g' x = \varphi g x$ 
    using  $\langle g \in s \rangle$ 
    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  $\text{action-induced-rel } s \ t \ \varphi = \text{action-induced-rel } s \ t \ \psi$ 
  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  $id = id \otimes BijGroup UNIV id$ 
    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
   $\varphi :: ('x, 'y) binary-fun$ 
defines  $\varphi\text{-img} \equiv (\lambda x. extensional\text{-continuation } (image (\varphi x)) (Pow s))$ 
assumes group-action m s  $\varphi$ 
shows group-action m (Pow s)  $\varphi\text{-img}$ 
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  $bij\text{-betw } (\varphi x) s s$ 
      using assms group-action.surj-prop
      unfolding bij-betw-def
      by (simp add: group-action.inj-prop)
    hence  $bij\text{-betw } (image (\varphi x)) (Pow s) (Pow s)$ 
      using bij-betw-Pow
      by metis
    moreover have  $\forall t \in Pow s. \varphi\text{-img } x t = image (\varphi x) t$ 
      unfolding  $\varphi\text{-img-def}$ 
      by simp
    ultimately have  $bij\text{-betw } (\varphi\text{-img } x) (Pow s) (Pow s)$ 
      using bij-betw-cong
  }

```

```

    by fastforce
  moreover have  $\varphi\text{-img } x \in \text{extensional } (\text{Pow } s)$ 
    unfolding  $\varphi\text{-img-def extensional-def}$ 
    by simp
  ultimately show  $\varphi\text{-img } x \in \text{carrier } (\text{BijGroup } (\text{Pow } s))$ 
    unfolding  $\text{BijGroup-def Bij-def}$ 
    by simp
}
fix x y :: 'x
note
   $\langle x \in \text{carrier } m \implies \varphi\text{-img } x \in \text{carrier } (\text{BijGroup } (\text{Pow } s)) \rangle$  and
   $\langle y \in \text{carrier } m \implies \varphi\text{-img } y \in \text{carrier } (\text{BijGroup } (\text{Pow } s)) \rangle$ 
moreover assume
  carrier-x:  $x \in \text{carrier } m$  and
  carrier-y:  $y \in \text{carrier } m$ 
ultimately have
  carrier-election-x:  $\varphi\text{-img } x \in \text{carrier } (\text{BijGroup } (\text{Pow } s))$  and
  carrier-election-y:  $\varphi\text{-img } y \in \text{carrier } (\text{BijGroup } (\text{Pow } s))$ 
  by (presburger, presburger)
hence h-closed:  $\forall t \in \text{Pow } s. \varphi\text{-img } y \ t \in \text{Pow } s$ 
  using bij-betw-apply Int-Collect
  unfolding  $\text{BijGroup-def Bij-def partial-object.select-convs}$ 
  by (metis (no-types))
from carrier-election-x carrier-election-y
have  $\varphi\text{-img } x \otimes \text{BijGroup } (\text{Pow } s) \ \varphi\text{-img } y =$ 
  extensional-continuation  $(\varphi\text{-img } x \circ \varphi\text{-img } y) (\text{Pow } s)$ 
  using rewrite-mult
  by blast
moreover have
   $\forall t. t \notin \text{Pow } s \longrightarrow \text{extensional-continuation } (\varphi\text{-img } x \circ \varphi\text{-img } y) (\text{Pow } s) \ t = \text{undefined}$ 
  by simp
moreover have
   $\forall t. t \notin \text{Pow } s \longrightarrow \varphi\text{-img } (x \otimes m \ y) \ t = \text{undefined}$  and
   $\forall t \in \text{Pow } s.$ 
    extensional-continuation  $(\varphi\text{-img } x \circ \varphi\text{-img } y) (\text{Pow } s) \ t = \varphi \ x \ ' \ \varphi \ y \ ' \ t$ 
  using h-closed
  unfolding  $\varphi\text{-img-def}$ 
  by (simp, simp)
moreover have  $\forall t \in \text{Pow } s. \varphi\text{-img } (x \otimes m \ y) \ t = \varphi \ x \ ' \ \varphi \ y \ ' \ t$ 
  unfolding  $\varphi\text{-img-def extensional-continuation.simps}$ 
  using rewrite-group-action-img carrier-x carrier-y assms PowD
  by metis
ultimately have
   $\forall t. \varphi\text{-img } (x \otimes m \ y) \ t = (\varphi\text{-img } x \otimes \text{BijGroup } (\text{Pow } s) \ \varphi\text{-img } y) \ t$ 
  by metis
thus  $\varphi\text{-img } (x \otimes m \ y) = \varphi\text{-img } x \otimes \text{BijGroup } (\text{Pow } s) \ \varphi\text{-img } y$ 
  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**
 $\varphi :: ('z, 'x)$ *binary-fun* **and**
 $\psi :: ('z, 'y)$ *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: $g \in \text{carrier } m$ **and**
x-in-t: $x \in t$

have $g \in \text{generate } m s$

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*: $\forall x \in t. \varphi (\text{inv } m \ g) \ x \in t$
 using *action- φ gen generate.inv group-action.element-image*
 by *metis*
 hence $\forall x \in t. f (\varphi \ g (\varphi (\text{inv } m \ g) \ x)) = \psi \ g (f (\varphi (\text{inv } m \ g) \ x))$
 using *gen action- φ equivar local.inv rewrite-equivariance*
 by *metis*
 moreover have $\forall x \in t. \varphi \ g (\varphi (\text{inv } m \ g) \ x) = x$
 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 $\forall x \in t. \psi \ g (f (\varphi (\text{inv } m \ g) \ x)) = f \ x$
 by *simp*
 moreover have *in-img-t*: $\forall x \in t. f (\varphi (\text{inv } m \ g) \ x) \in f \text{ `` } t$
 using *in-t*
 by *blast*
 ultimately have
 $\forall x \in t. \psi (\text{inv } m \ g) (\psi \ g (f (\varphi (\text{inv } m \ g) \ x))) = \psi (\text{inv } m \ g) (f \ x)$
 using *action- ψ gen*
 by *metis*
 moreover have
 $\forall x \in t. \psi (\text{inv } m \ g) (\psi \ g (f (\varphi (\text{inv } m \ g) \ x))) = f (\varphi (\text{inv } m \ g) \ x)$
 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 g₁ g₂*)
 assume
 equivar₁: $\forall x \in t. f (\varphi \ g_1 \ x) = \psi \ g_1 (f \ x)$ and
 equivar₂: $\forall x \in t. f (\varphi \ g_2 \ x) = \psi \ g_2 (f \ x)$ and
 gen₁: $g_1 \in \text{generate } m \ s$ and
 gen₂: $g_2 \in \text{generate } m \ s$
 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 *equivar₁*
 by *simp*
 moreover have $\forall x \in t. f (\varphi \ g_2 \ x) = \psi \ g_2 (f \ x)$
 using *equivar₂*
 by *simp*
 ultimately show ?*case*
 using *action- φ action- ψ gen gen₁ gen₂ 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
     $r :: 'x \text{ rel}$ 
  assumes
     $\forall x. \text{is-symmetry } (f \ x) \ (\text{Invariance } r)$  and
     $\text{is-symmetry } f \ (\text{Invariance } r)$ 
  shows  $\text{is-symmetry } (\lambda x. f \ x \ x) \ (\text{Invariance } r)$ 
  using assms
  by simp

```

```

lemma invar-under-subset-rel:
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $r \ s :: 'x \text{ rel}$ 
  assumes
     $\text{subset: } r \subseteq s$  and
     $\text{invar: is-symmetry } f \ (\text{Invariance } s)$ 
  shows  $\text{is-symmetry } f \ (\text{Invariance } r)$ 
  using assms
  by auto

```

```

lemma equivar-ind-by-act-coincide:
  fixes
     $s :: 'x \text{ set}$  and
     $t :: 'y \text{ set}$  and
     $f :: 'y \Rightarrow 'z$  and
     $\varphi \ \varphi' :: ('x, 'y) \text{ binary-fun}$  and
     $\psi :: ('x, 'z) \text{ binary-fun}$ 
  assumes  $\forall x \in s. \forall y \in t. \varphi \ x \ y = \varphi' \ x \ y$ 
  shows  $\text{is-symmetry } f \ (\text{action-induced-equivariance } s \ t \ \varphi \ \psi) =$ 
     $\text{is-symmetry } f \ (\text{action-induced-equivariance } s \ t \ \varphi' \ \psi)$ 
  using assms
  unfolding rewrite-equivariance
  by simp

```

```

lemma equivar-under-subset:
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $s \ t :: 'x \text{ set}$  and
     $\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y)) \text{ set}$ 
  assumes
     $\text{is-symmetry } f \ (\text{Equivariance } s \ \tau)$  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
   $\tau v :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$  set
assumes
  is-symmetry  $f$  (Equivariance  $s$   $\tau$ ) and
   $v \subseteq \tau$ 
shows is-symmetry  $f$  (Equivariance  $s$   $v$ )
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
   $\varphi :: ('z, 'x)$  binary-fun and
   $\psi :: ('z, 'y)$  binary-fun and
   $x :: 'z$ 
defines equivar-prop  $\equiv$  action-induced-equivariance (carrier  $m$ )  $\mathcal{D}_f$   $\varphi$   $\psi$ 
assumes
  action- $\varphi$ : group-action  $m$   $s$   $\varphi$  and
  action-res: group-action  $m$  UNIV  $\psi$  and
  dom-in-s:  $\mathcal{D}_f \subseteq s$  and
  closed-domain:
    closed-restricted-rel (action-induced-rel (carrier  $m$ )  $s$   $\varphi$ )  $s$   $\mathcal{D}_f$  and
    equivar-f: is-symmetry  $f$  equivar-prop and
    group-elem-x:  $x \in \text{carrier } m$ 
shows  $\forall y. \text{preimg } f \mathcal{D}_f (\psi \ x \ y) = (\varphi \ x) \text{ ` } (\text{preimg } f \mathcal{D}_f \ y)$ 
proof (safe)
interpret action- $\varphi$ : group-action  $m$   $s$   $\varphi$ 
using action- $\varphi$ 
by simp
interpret action-results: group-action  $m$  UNIV  $\psi$ 
using action-res
by simp
have group-elem-inv:  $(\text{inv } m \ x) \in \text{carrier } m$ 
using group.inv-closed action- $\varphi$ .group-hom group-elem-x
unfolding group-hom-def
by metis
fix

```

```

  y :: 'y and
  z :: 'x
assume preimg-el: z ∈ preimg f  $\mathcal{D}_f$  ( $\psi$  x y)
obtain a :: 'x where
  img: a =  $\varphi$  (inv m x) z
  by simp
have domain: z ∈  $\mathcal{D}_f$  ∧ z ∈ s
  using preimg-el dom-in-s
  by auto
hence a ∈ s
  using dom-in-s action- $\varphi$  group-elem-inv preimg-el img action- $\varphi$ .element-image
  by auto
hence (z, a) ∈ (action-induced-rel (carrier m) s  $\varphi$ ) ∩ ( $\mathcal{D}_f$  × s)
  using img preimg-el domain group-elem-inv
  by auto
hence a ∈ ((action-induced-rel (carrier m) s  $\varphi$ ) ∩ ( $\mathcal{D}_f$  × s)) “  $\mathcal{D}_f$ 
  using img preimg-el domain group-elem-inv
  by auto
hence a-in-domain: a ∈  $\mathcal{D}_f$ 
  using closed-domain
  by auto
moreover have ( $\varphi$  (inv m x),  $\psi$  (inv m x)) ∈ {( $\varphi$  g,  $\psi$  g) | g. g ∈ carrier m}
  using group-elem-inv
  by auto
ultimately have f a =  $\psi$  (inv m x) (f z)
  using domain equivar-f img
  unfolding equivar-prop-def action-induced-equivariance-def
  by simp
also have f z =  $\psi$  x y
  using preimg-el
  by simp
also have  $\psi$  (inv m x) ( $\psi$  x y) = y
  using action-results.group-hom action-results.orbit-sym-aux group-elem-x
  by simp
finally have f a = y
  by simp
hence a ∈ preimg f  $\mathcal{D}_f$  y
  using a-in-domain
  by simp
moreover have z =  $\varphi$  x a
  using action- $\varphi$ .group-hom action- $\varphi$ .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 ∈ ( $\varphi$  x) “ (preimg f  $\mathcal{D}_f$  y)
  by simp
next
fix
  y :: 'y and

```



```

  z :: 'x
assume z ∈ preimg f  $\mathcal{D}_f$  y
hence domain: f z = y ∧ z ∈  $\mathcal{D}_f$  ∧ z ∈ s
  using dom-in-s
  by auto
hence  $\varphi$  x z ∈ s
  using group-elem-x group-action.element-image action- $\varphi$ 
  by metis
hence (z,  $\varphi$  x z) ∈ (action-induced-rel (carrier m) s  $\varphi$ ) ∩ ( $\mathcal{D}_f$  × s) ∩  $\mathcal{D}_f$  × s
  using group-elem-x domain
  by auto
hence  $\varphi$  x z ∈  $\mathcal{D}_f$ 
  using closed-domain
  by auto
moreover have ( $\varphi$  x,  $\psi$  x) ∈ {( $\varphi$  a,  $\psi$  a) | a. a ∈ carrier m}
  using group-elem-x
  by blast
ultimately show  $\varphi$  x z ∈ preimg f  $\mathcal{D}_f$  ( $\psi$  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 ⇒ 'y and
    g :: 'y ⇒ 'z and
    s :: 'x set and
    t :: 'y set and
     $\tau$  :: (('x ⇒ 'x) × ('y ⇒ 'y)) set and
    v :: (('y ⇒ 'y) × ('z ⇒ 'z)) set
  defines
    transitive-acts ≡
      {( $\varphi$ ,  $\psi$ ). ∃  $\chi$  :: 'y ⇒ 'y. ( $\varphi$ ,  $\chi$ ) ∈  $\tau$  ∧ ( $\chi$ ,  $\psi$ ) ∈ v ∧  $\chi$  ' f ' s ⊆ t}
  assumes
    f ' s ⊆ t and
    is-symmetry f (Equivariance s  $\tau$ ) and

```

$is-symmetry\ g\ (Equivariance\ t\ v)$
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**
 $\chi-img_f-img_s-in-t: \chi\ 'f\ 's \subseteq t$ **and**
 $act-f: (\varphi, \chi) \in \tau$ **and**
 $act-g: (\chi, \psi) \in v$
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 $\{(\varphi \ a, \psi \ a) \mid a. a \in s\}$
 $\subseteq \{(\varphi, \psi). \exists v. (\varphi, v) \in ?a_\varphi \wedge (v, \psi) \in ?a_\psi \wedge v \text{ ' } f \text{ ' } t \subseteq u\}$
by *blast*
hence *is-symmetry* $(g \circ f)$ (*Equivariance* $t \{(\varphi \ a, \psi \ a) \mid a. a \in s\}$)
using *assms* *equivar-comp*[*of f t u ?a_φ g ?a_ψ*] *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
 $f \ g :: 'x \Rightarrow 'y \text{ set}$ **and**
 $s :: 'z \text{ set}$ **and**
 $t :: 'x \text{ set}$ **and**
 $\varphi :: ('z, 'x) \text{ binary-fun}$ **and**
 $\psi :: ('z, 'y) \text{ binary-fun}$
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: $\forall a \in s. \text{bij}(\psi \ a)$
shows
is-symmetry $(\lambda b. f \ b - g \ b)$ (*action-induced-equivariance* $s \ t \ \varphi$ (*set-action* ψ))
proof –
have
 $\forall a \in s. \forall x \in t. f(\varphi \ a \ x) = \psi \ a \text{ ' } (f \ x)$ **and**
 $\forall a \in s. \forall x \in t. g(\varphi \ a \ x) = \psi \ a \text{ ' } (g \ x)$
using *f-equivar g-equivar*
unfolding *rewrite-equivariance*
by (*simp*, *simp*)
hence $\forall a \in s. \forall b \in t. f(\varphi \ a \ b) - g(\varphi \ a \ b) = \psi \ a \text{ ' } (f \ b) - \psi \ a \text{ ' } (g \ b)$
by *blast*
moreover have $\forall a \in s. \forall u \ v. \psi \ a \text{ ' } u - \psi \ a \text{ ' } v = \psi \ a \text{ ' } (u - v)$
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
 $f :: 'x \Rightarrow 'y$ **and**
 $s :: 'z \text{ set}$ **and**
 $\varphi :: ('z, 'x) \text{ binary-fun}$

```

shows is-symmetry  $\bigcup$  (action-induced-equivariance s UNIV
  (set-action (set-action  $\varphi$ )) (set-action  $\varphi$ ))
proof (unfold action-induced-equivariance-def is-symmetry.simps set-action.simps,
  safe)
fix
  x :: 'z and
  ts :: 'x set set and
  t :: 'x set and
  y :: 'x
assume
  y  $\in$  t and
  t  $\in$  ts
thus
   $\varphi$  x y  $\in$   $\varphi$  x '  $\bigcup$  ts and
   $\varphi$  x y  $\in$   $\bigcup$  (( $\cdot$ ) ( $\varphi$  x) ' ts)
  by (blast, blast)
qed
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 (in result) closed-elections :: ('a, 'v) Election rel  $\Rightarrow$  bool where
  closed-elections r =
    ( $\forall$  (e, e')  $\in$  r.
      limit (alternatives- $\mathcal{E}$  e) UNIV = limit (alternatives- $\mathcal{E}$  e') UNIV)

fun result-action :: ('x, 'r) binary-fun  $\Rightarrow$  ('x, 'r) Result binary-fun where
  result-action  $\psi$  x = ( $\lambda$  r. ( $\psi$  x ' elect-r r,  $\psi$  x ' reject-r r,  $\psi$  x ' defer-r r))

```

Anonymity

```

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

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

```

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

Neutrality

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

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

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

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

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

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

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

Homogeneity

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

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

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

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_G* :: (*'a rel* \Rightarrow *'a rel*) *monoid* **where**
reversal_G \equiv (\langle carrier = {*reverse-rel*, *id*}, *monoid.mult* = *comp*, *one* = *id* \rangle)

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_R* :: (*'a*, *'v*) *Election* *set* \Rightarrow (*'a*, *'v*) *Election* *rel* **where**
reversal_R \mathcal{E} = *action-induced-rel* (*carrier reversal_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**

$f :: 'x \Rightarrow 'x$ **and**

$x :: 'x$

assumes

fin-A: *finite A* **and**

fin-B: *finite B* **and**

x-el: $x \in A - B$ **and**

bij-f: *bij-betw f A B*

obtains $n :: nat$ **where**

$n > 0$ **and**

$n\text{-app}\ n\ f\ x \in B - A$ **and**

$\forall\ m > 0. m < n \longrightarrow n\text{-app}\ m\ f\ x \in A \cap B$

proof –

have *n-app-f-x-in-A*: $n\text{-app}\ 0\ f\ x \in A$

using *x-el*

by *simp*

moreover have *ex-A*:

$\exists\ n > 0. n\text{-app}\ n\ f\ x \in B - A \wedge (\forall\ m > 0. m < n \longrightarrow n\text{-app}\ m\ f\ x \in A)$

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*:

$\forall\ n. n\text{-app}\ n\ f\ x \in B$

$\longrightarrow n = 0 \vee n\text{-app}\ n\ f\ x \in A \vee (\exists\ m > 0. m < n \wedge n\text{-app}\ m\ f\ x \notin A)$

hence $\forall\ n > 0. n\text{-app}\ n\ f\ x \in B$

$\longrightarrow n\text{-app}\ n\ f\ x \in A \vee (\exists\ m > 0. m < n \wedge n\text{-app}\ m\ f\ x \notin A)$

by *blast*

moreover have $\neg (\forall\ n > 0. n\text{-app}\ n\ f\ x \in B \longrightarrow n\text{-app}\ n\ f\ x \in A)$

proof (*safe*)

assume *in-A*: $\forall\ n > 0. n\text{-app}\ n\ f\ x \in B \longrightarrow n\text{-app}\ n\ f\ x \in A$

hence $\forall\ n > 0. n\text{-app}\ n\ f\ x \in A \longrightarrow n\text{-app}\ (Suc\ n)\ f\ x \in A$

using *n-app.simps bij-f*

unfolding *bij-betw-def*

by *force*

hence *in-AB-imp-in-AB*:

$\forall\ n > 0. n\text{-app}\ n\ f\ x \in A \cap B \longrightarrow n\text{-app}\ (Suc\ n)\ f\ x \in A \cap B$

using *n-app.simps bij-f*

unfolding *bij-betw-def*

by *auto*

have *in-int*: $\forall\ n > 0. n\text{-app}\ n\ f\ x \in A \cap B$

proof (*clarify*)

fix $n :: nat$

assume $n > 0$

thus $n\text{-app}\ n\ f\ x \in A \cap B$

```

proof (induction n)
  case 0
  thus ?case
    by safe
next
  case (Suc n)
  assume  $0 < n \implies n\text{-app } n \ f \ x \in A \cap B$ 
  moreover have  $n = 0 \longrightarrow n\text{-app } (Suc \ n) \ f \ x = f \ x$ 
    by simp
  ultimately show  $n\text{-app } (Suc \ n) \ f \ x \in A \cap B$ 
    using x-el bij-f in-A in-AB-imp-in-AB
    unfolding bij-betw-def
    by blast
qed
qed
hence  $\{n\text{-app } n \ f \ x \mid n. \ n > 0\} \subseteq A \cap B$ 
  by blast
hence finite  $\{n\text{-app } n \ f \ x \mid n. \ n > 0\}$ 
  using fin-A fin-B rev-finite-subset
  by blast
moreover have
  inj-on  $(\lambda \ n. \ n\text{-app } n \ f \ x) \ \{n. \ n > 0\}$ 
     $\longrightarrow$  infinite  $((\lambda \ n. \ n\text{-app } n \ f \ x) \ ' \ \{n. \ n > 0\})$ 
  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  $(\lambda \ n. \ n\text{-app } n \ f \ x) \ ' \ \{n. \ n > 0\} = \{n\text{-app } n \ f \ x \mid n. \ n > 0\}$ 
  by auto
ultimately have  $\neg \text{inj-on } (\lambda \ n. \ n\text{-app } n \ f \ x) \ \{n. \ n > 0\}$ 
  by metis
hence  $\exists \ n > 0. \ \exists \ m > n. \ n\text{-app } n \ f \ x = n\text{-app } m \ f \ x$ 
  using linorder-inj-onI' mem-Collect-eq
  by metis
hence  $\exists \ n\text{-min} > 0.$ 
   $(\exists \ m > n\text{-min}. \ n\text{-app } n\text{-min} \ f \ x = n\text{-app } m \ f \ x)$ 
   $\wedge (\forall \ n < n\text{-min}. \ \neg (0 < n \wedge (\exists \ m > n. \ n\text{-app } n \ f \ x = n\text{-app } m \ f \ x)))$ 
  using exists-least-iff[of
     $\lambda \ n. \ n > 0 \wedge (\exists \ m > n. \ n\text{-app } n \ f \ x = n\text{-app } m \ f \ x)]$ 
  by presburger
then obtain n-min :: nat where
  n-min-pos: n-min > 0 and
   $\exists \ m > n\text{-min}. \ n\text{-app } n\text{-min} \ f \ x = n\text{-app } m \ f \ x$  and
  neq:  $\forall \ n < n\text{-min}. \ \neg (n > 0 \wedge (\exists \ m > n. \ n\text{-app } n \ f \ x = n\text{-app } m \ f \ x))$ 
  by blast
then obtain m :: nat where
  m-gt-n-min: m > n-min and
   $n\text{-app } n\text{-min} \ f \ x = f \ (n\text{-app } (m - 1) \ f \ x)$ 
  using comp-apply diff-Suc-1 less-nat-zero-code n-app.elims
  by (metis (mono-tags, lifting))

```



```

moreover have  $n\text{-app } n\text{-min } f x = f (n\text{-app } (n\text{-min} - 1) f x)$ 
  using Suc-pred' n-min-pos comp-eq-id-dest id-comp diff-Suc-1
    less-nat-zero-code n-app.elims
  by (metis (mono-tags, opaque-lifting))
moreover have  $n\text{-app } (m - 1) f x \in A \wedge n\text{-app } (n\text{-min} - 1) f x \in A$ 
  using in-int x-el n-min-pos m-gt-n-min Diff-iff IntD1 diff-le-self id-apply
    nless-le cancel-comm-monoid-add-class.diff-cancel n-app-id
  by metis
ultimately have  $eq: n\text{-app } (m - 1) f x = n\text{-app } (n\text{-min} - 1) f x$ 
  using bij-f
  unfolding bij-betw-def inj-def inj-on-def
  by simp
moreover have  $m - 1 > n\text{-min} - 1$ 
  using m-gt-n-min n-min-pos
  by simp
ultimately have  $n\text{-min} - 1 > 0 \longrightarrow \text{False}$ 
  using neq n-min-pos diff-less zero-less-one
  by metis
moreover have  $n\text{-app } (m - 1) f x \in B$ 
  using in-int m-gt-n-min n-min-pos
  by simp
ultimately show False
  using x-el eq
  by simp
qed
ultimately have  $\exists n > 0. \exists m > 0. m < n \wedge n\text{-app } m f x \notin A$ 
  by blast
hence  $\exists n > 0. n\text{-app } n f x \notin A \wedge (\forall m < n. \neg (m > 0 \wedge n\text{-app } m f x \notin A))$ 
  using exists-least-iff[of  $\lambda n. n > 0 \wedge n\text{-app } n f x \notin A$ ]
  by blast
then obtain  $n :: \text{nat}$  where
   $n\text{-pos}: n > 0$  and
   $\text{not-in-}A: n\text{-app } n f x \notin A$  and
   $\text{less-in-}A: \forall m. (0 < m \wedge m < n) \longrightarrow n\text{-app } m f x \in A$ 
  by blast
moreover have  $n\text{-app } 0 f x \in A$ 
  using x-el
  by simp
ultimately have  $n\text{-app } (n - 1) f x \in A$ 
  using bot-nat-0.not-eq-extremum diff-less zero-less-one
  by metis
moreover have  $n\text{-app } n f x = f (n\text{-app } (n - 1) f x)$ 
  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
 $\forall n. (\forall m > 0. m < n \longrightarrow n\text{-app } m \ f \ x \in A)$
 $\longrightarrow (\forall m > 0. m < n \longrightarrow n\text{-app } (m - 1) \ f \ x \in A)$
using *bot-nat-0.not-eq-extremum less-imp-diff-less*
by *metis*
moreover have $\forall m > 0. n\text{-app } m \ f \ x = f \ (n\text{-app } (m - 1) \ f \ x)$
using *bot-nat-0.not-eq-extremum comp-apply diff-Suc-1 n-app.elims*
by (*metis (mono-tags, lifting)*)
ultimately have
 $\forall n. (\forall m > 0. m < n \longrightarrow n\text{-app } m \ f \ x \in A)$
 $\longrightarrow (\forall m > 0. m \leq n \longrightarrow n\text{-app } m \ f \ x \in B)$
using *bij-f n-app-id n-app-f-x-in-A diff-Suc-1 gr0-conv-Suc imageI*
linorder-not-le nless-le not-less-eq-eq
unfolding *bij-betw-def*
by *metis*
hence $\exists n > 0. n\text{-app } n \ f \ x \in B - A$
 $\wedge (\forall m > 0. m < n \longrightarrow n\text{-app } m \ f \ x \in A \cap B)$
using *IntI nless-le ex-A*
by *metis*
thus *?thesis*
using *that*
by *blast*
qed

lemma *n-app-rev*:

fixes
 $A \ B :: 'x \text{ set}$ **and**
 $f :: 'x \Rightarrow 'x$ **and**
 $m \ n :: \text{nat}$ **and**
 $x \ y :: 'x$
assumes
 $x\text{-in-}A: x \in A$ **and**
 $y\text{-in-}A: y \in A$ **and**
 $n\text{-geq-}m: n \geq m$ **and**
 $n\text{-app-eq-}m\text{-n}: n\text{-app } n \ f \ x = n\text{-app } m \ f \ y$ **and**
 $n\text{-app-}x\text{-in-}A: \forall n' < n. n\text{-app } n' \ f \ x \in A$ **and**
 $n\text{-app-}y\text{-in-}A: \forall m' < m. n\text{-app } m' \ f \ y \in A$ **and**
 $\text{fin-}A: \text{finite } A$ **and**
 $\text{fin-}B: \text{finite } B$ **and**
 $\text{bij-}f\text{-}A\text{-}B: \text{bij-betw } f \ A \ B$
shows $n\text{-app } (n - m) \ f \ x = y$
using *assms*
proof (*induction n f arbitrary: m x y rule: n-app.induct*)
case (1 *f*)
fix
 $f :: 'x \Rightarrow 'x$ **and**
 $m :: \text{nat}$ **and**
 $x \ y :: 'x$
assume

```

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

```

```

    using Suc-diff-eq-diff-pred
    by presburger
  moreover have  $m = 0 \longrightarrow n\text{-app } (Suc\ n - m) f\ x = y$ 
    using eq
    by simp
  ultimately show  $n\text{-app } (Suc\ n - m) f\ x = y$ 
    by blast
qed

lemma n-app-inv:
  fixes
     $A\ B :: 'x\ set$  and
     $f :: 'x \Rightarrow 'x$  and
     $n :: nat$  and
     $x :: 'x$ 
  assumes
     $x \in B$  and
     $\forall\ m \geq 0. m < n \longrightarrow n\text{-app } m\ (the\text{-inv-into } A\ f)\ x \in B$  and
     $bij\text{-betw } f\ A\ B$ 
  shows  $n\text{-app } n\ f\ (n\text{-app } n\ (the\text{-inv-into } A\ f)\ x) = x$ 
    using assms
  proof (induction  $n\ f$  arbitrary:  $x$  rule: n-app.induct)
    case (1  $f$ )
    fix  $f :: 'x \Rightarrow 'x$ 
    show ?case
      by simp
  next
    case (2  $n\ f$ )
    fix
       $n :: nat$  and
       $f :: 'x \Rightarrow 'x$  and
       $x :: 'x$ 
    assume
       $x\text{-in-}B: x \in B$  and
       $bij\text{-}f: bij\text{-betw } f\ A\ B$  and
       $stays\text{-in-}B: \forall\ m \geq 0. m < Suc\ n \longrightarrow n\text{-app } m\ (the\text{-inv-into } A\ f)\ x \in B$  and
       $hyp: \bigwedge\ x. x \in B \Longrightarrow$ 
         $\forall\ m \geq 0. m < n \longrightarrow n\text{-app } m\ (the\text{-inv-into } A\ f)\ x \in B \Longrightarrow$ 
         $bij\text{-betw } f\ A\ B \Longrightarrow n\text{-app } n\ f\ (n\text{-app } n\ (the\text{-inv-into } A\ f)\ x) = x$ 
    have  $n\text{-app } (Suc\ n) f\ (n\text{-app } (Suc\ n) (the\text{-inv-into } A\ f)\ x) =$ 
       $n\text{-app } n\ f\ (f\ (n\text{-app } (Suc\ n) (the\text{-inv-into } A\ f)\ x))$ 
      using n-app-rewrite
      by simp
    also have  $\dots = n\text{-app } n\ f\ (n\text{-app } n\ (the\text{-inv-into } A\ f)\ x)$ 
      using stays-in-B bij-f
      by (simp add: f-the-inv-into-f-bij-betw)
    finally show  $n\text{-app } (Suc\ n) f\ (n\text{-app } (Suc\ n) (the\text{-inv-into } A\ f)\ x) = x$ 
      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**

$f :: 'x \Rightarrow 'x$

assumes

fin-A: *finite A* **and**

fin-B: *finite B* **and**

bij-f: *bij-betw f A B*

obtains $g :: 'x \Rightarrow 'x$ **where**

bij g **and**

$\forall a \in A. g\ a = f\ a$ **and**

$\forall b \in B - A. g\ b \in A - B \wedge (\exists\ n > 0. n\text{-app}\ n\ f\ (g\ b) = b)$ **and**

$\forall x \in UNIV - A - B. g\ x = x$

proof –

assume *existence-witness*:

$\bigwedge g. \textit{bij}\ g \Longrightarrow$

$\forall a \in A. g\ a = f\ a \Longrightarrow$

$\forall b \in B - A. g\ b \in A - B \wedge (\exists\ n > 0. n\text{-app}\ n\ f\ (g\ b) = b) \Longrightarrow$

$\forall x \in UNIV - A - B. g\ x = x \Longrightarrow ?thesis$

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: $\forall x \in B - A. n\text{-app}\ (g'\ x)\ (the\text{-inv-into}\ A\ f)\ x \in A - B$ **and**

minimal: $\forall x \in B - A. \forall n > 0.$

$n < g'\ x \longrightarrow n\text{-app}\ n\ (the\text{-inv-into}\ A\ f)\ x \in B \cap A$

using *n-app-leaves-set fin-A fin-B*

by *metis*

obtain $g :: 'x \Rightarrow 'x$ **where**

def-g:

$g = (\lambda x. \text{if } x \in A \text{ then } f\ x \text{ else}$

$(\text{if } x \in B - A \text{ then } n\text{-app}\ (g'\ x)\ (the\text{-inv-into}\ A\ f)\ x \text{ else } x))$

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 $\forall x \in B - A. n\text{-app}\ 0\ (the\text{-inv-into}\ A\ f)\ x \in B$

by *simp*

moreover have

$\forall x \in B - A. \forall n > 0.$

$n < g'\ x \longrightarrow n\text{-app}\ n\ (the\text{-inv-into}\ A\ f)\ x \in B$

using *minimal*

by *blast*

ultimately have

$\forall x \in B - A. n\text{-app } (g' x) f (n\text{-app } (g' x) (the\text{-inv-into } A f) x) = x$
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 $g^{-1} (UNIV - A - B) = UNIV - A - B$
by *simp*
moreover **have** $g^{-1} A = B$
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*: $g^{-1} (UNIV - (B - A)) = UNIV - (A - B)$
using *image-Un*
by *metis*
have *inj-on* $g A \wedge inj\text{-on } g (UNIV - A - B)$
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** $(\lambda x. n\text{-app } (g' x) (the\text{-inv-into } A f) x)^{-1} (B - A) \subseteq A - B$
using *in-set-diff*
by *blast*
moreover **have** *inj-on* $(\lambda x. n\text{-app } (g' x) (the\text{-inv-into } A f) x) (B - A)$
proof (*unfold inj-on-def, safe*)
fix $x y :: 'x$
assume
 $x\text{-in-}B: x \in B$ **and**
 $x\text{-not-in-}A: x \notin A$ **and**

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

fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $X :: 'x \text{ set}$ **and**
 $Y :: 'y \text{ set}$
assumes $\text{bij-betw } f \ X \ Y$
shows $\text{bij-betw } (\text{extensional-continuation } f \ X) \ X \ Y$
proof –
have $\forall x \in X. \text{extensional-continuation } f \ X \ x = f \ x$
by *simp*
thus *?thesis*
using *assms bij-betw-cong*
by *metis*
qed

1.10.3 Anonymity Lemmas

lemma *anon-rel-vote-count*:
fixes
 $\mathcal{E} :: ('a, 'v) \text{ Election set}$ **and**
 $E \ E' :: ('a, 'v) \text{ Election}$
assumes
 $\text{finite } (\text{voters-}\mathcal{E} \ E)$ **and**
 $(E, E') \in \text{anonymity}_{\mathcal{R}} \ \mathcal{E}$
shows $\text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E' \wedge E \in \mathcal{E}$
 $\wedge (\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E')$
proof –
have $E \in \mathcal{E}$
using *assms*
unfolding *anonymity_R.simps action-induced-rel.simps*
by *safe*
with *assms*
obtain $\pi :: 'v \Rightarrow 'v$ **where**
 $\text{bijection-}\pi: \text{bij } \pi$ **and**
 $\text{renamed: } E' = \text{rename } \pi \ E$
unfolding *anonymity_R.simps anonymity_G-def*
using *universal-set-carrier-imp-bij-group*
by *auto*
have $\text{eq-alts: alternatives-}\mathcal{E} \ E' = \text{alternatives-}\mathcal{E} \ E$
using *eq-fst-iff rename.simps alternatives-}\mathcal{E}.elems renamed*
by *(metis (no-types))*
have $\forall v \in \text{voters-}\mathcal{E} \ E'. (\text{profile-}\mathcal{E} \ E') \ v = (\text{profile-}\mathcal{E} \ E) \ (\text{the-inv } \pi \ v)$
unfolding *profile-}\mathcal{E}.simps*
using *renamed rename.simps comp-apply prod.collapse snd-conv*
by *(metis (no-types, lifting))*
hence *rewrite*:
 $\forall p. \{v \in (\text{voters-}\mathcal{E} \ E'). (\text{profile-}\mathcal{E} \ E') \ v = p\} =$
 $\{v \in (\text{voters-}\mathcal{E} \ E'). (\text{profile-}\mathcal{E} \ E) \ (\text{the-inv } \pi \ v) = p\}$
by *blast*
have $\forall v \in \text{voters-}\mathcal{E} \ E'. \text{the-inv } \pi \ v \in \text{voters-}\mathcal{E} \ E$

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

lemma *vote-count-anon-rel*:

fixes

$\mathcal{E} :: ('a, 'v) \text{ Election set}$ **and**

$E \ E' :: ('a, 'v) \text{ Election}$

assumes

fin-voters-E: *finite (voters- $\mathcal{E} \ E)$* **and**

fin-voters-E': *finite (voters- $\mathcal{E} \ E')$* **and**

default-non-v: $\forall v. v \notin \text{voters-}\mathcal{E} \ E \longrightarrow \text{profile-}\mathcal{E} \ E \ v = \{\}$ **and**

default-non-v': $\forall v. v \notin \text{voters-}\mathcal{E} \ E' \longrightarrow \text{profile-}\mathcal{E} \ E' \ v = \{\}$ **and**

eq: $\text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E}$

$\wedge (\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E')$
shows $(E, E') \in \text{anonymity}_{\mathcal{R}} \ \mathcal{E}$
proof –
have $\forall p. \text{card } \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} =$
 $\text{card } \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\}$
using *eq*
unfolding *vote-count.simps*
by *blast*
moreover have
 $\forall p. \text{finite } \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}$
 $\wedge \text{finite } \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\}$
using *assms*
by *simp*
ultimately have
 $\forall p. \exists \pi_p. \text{bij-betw } \pi_p$
 $\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}$
 $\{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\}$
using *bij-betw-iff-card*
by *blast*
then obtain $\pi :: 'a \text{ Preference-Relation} \Rightarrow ('v \Rightarrow 'v) \text{ where}$
 $\text{bij-}\pi: \forall p. \text{bij-betw } (\pi \ p) \ \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}$
 $\{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\}$
by (*metis (no-types)*)
obtain $\pi' :: 'v \Rightarrow 'v \text{ where}$
 $\pi'\text{-perm}: \forall v \in \text{voters-}\mathcal{E} \ E. \pi' \ v = \pi (\text{profile-}\mathcal{E} \ E \ v) \ v$
by *fastforce*
hence $\forall v \in \text{voters-}\mathcal{E} \ E. \forall v' \in \text{voters-}\mathcal{E} \ E.$
 $\pi' \ v = \pi' \ v' \longrightarrow \pi (\text{profile-}\mathcal{E} \ E \ v) \ v = \pi (\text{profile-}\mathcal{E} \ E \ v') \ v'$
by *simp*
moreover have
 $\forall w \in \text{voters-}\mathcal{E} \ E. \forall w' \in \text{voters-}\mathcal{E} \ E.$
 $\pi (\text{profile-}\mathcal{E} \ E \ w) \ w = \pi (\text{profile-}\mathcal{E} \ E \ w') \ w'$
 $\longrightarrow \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w\}$
 $\cap \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w'\} \neq \{\}$
using *bij-π*
unfolding *bij-betw-def*
by *blast*
moreover have
 $\forall w \ w'.$
 $\{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w\}$
 $\cap \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w'\} \neq \{\}$
 $\longrightarrow \text{profile-}\mathcal{E} \ E \ w = \text{profile-}\mathcal{E} \ E \ w'$
by *blast*
ultimately have *eq-prof*:
 $\forall v \in \text{voters-}\mathcal{E} \ E. \forall v' \in \text{voters-}\mathcal{E} \ E.$
 $\pi' \ v = \pi' \ v' \longrightarrow \text{profile-}\mathcal{E} \ E \ v = \text{profile-}\mathcal{E} \ E \ v'$
by *blast*
hence $\forall v \in \text{voters-}\mathcal{E} \ E. \forall v' \in \text{voters-}\mathcal{E} \ E.$
 $\pi' \ v = \pi' \ v' \longrightarrow \pi (\text{profile-}\mathcal{E} \ E \ v) \ v = \pi (\text{profile-}\mathcal{E} \ E \ v') \ v'$

using π' -perm
by *metis*
hence $\forall v \in \text{voters-}\mathcal{E} \ E. \forall v' \in \text{voters-}\mathcal{E} \ E. \pi' v = \pi' v' \longrightarrow v = v'$
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 $\pi' \text{ ' voters-}\mathcal{E} \ E = \{\pi (\text{profile-}\mathcal{E} \ E \ v) \ v \mid v. v \in \text{voters-}\mathcal{E} \ E\}$
using π' -perm
unfolding *Setcompr-eq-image*
by *simp*
also have
 $\dots = \bigcup \{\pi \ p \text{ ' } \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\}$
unfolding *Union-eq*
by *blast*
also have
 $\dots = \bigcup \{\{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\} \mid p. p \in \text{UNIV}\}$
using *bij- π*
unfolding *bij-betw-def*
by (*metis (mono-tags, lifting)*)
finally have $\pi' \text{ ' voters-}\mathcal{E} \ E = \text{voters-}\mathcal{E} \ E'$
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 $\pi\text{-global} :: \text{'v} \Rightarrow \text{'v}$ **where**
bijection- π_g : bij $\pi\text{-global}$ and
 $\pi\text{-global-eq-}\pi': \forall v \in \text{voters-}\mathcal{E} \ E. \pi\text{-global} \ v = \pi' \ v$ **and**
 $\pi\text{-global-eq-n-app-}\pi':$
 $\forall v \in \text{voters-}\mathcal{E} \ E' - \text{voters-}\mathcal{E} \ E.$
 $\pi\text{-global} \ v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E' \wedge$
 $(\exists n > 0. n\text{-app } n \ \pi' (\pi\text{-global} \ v) = v)$ **and**
 $\pi\text{-global-non-voters: } \forall v \in \text{UNIV} - \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \pi\text{-global} \ v = v$
using *fin-voters-E fin-voters-E' bij-betw-finite-ind-global-bij*
by *blast*
hence *inv: $\forall v \ v'. (\pi\text{-global} \ v' = v) = (v' = \text{the-inv } \pi\text{-global} \ v)$*
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
 $\forall v \in \text{UNIV} - (\text{voters-}\mathcal{E} \ E' - \text{voters-}\mathcal{E} \ E).$
 $\pi\text{-global} \ v \in \text{UNIV} - (\text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E')$
using $\pi\text{-global-eq-}\pi' \ \pi\text{-global-non-voters} \ \text{bij'} \ \text{bijection-}\pi_g$
 $\text{DiffD1 DiffD2 DiffI bij-betwE}$
by (*metis (no-types, lifting)*)
ultimately have
 $\forall v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'.$

$the_inv \pi_global v \in voters_E E' - voters_E E$
using $bijection-\pi_g \pi_global_eq-n-app-\pi' DiffD2 DiffI UNIV-I$
by *metis*
hence $\forall v \in voters_E E - voters_E E'. \forall n > 0.$
 $profile_E E (the_inv \pi_global v) = \{\}$
using *default-non-v*
by *simp*
moreover have $\forall v \in voters_E E - voters_E E'. profile_E E' v = \{\}$
using *default-non-v'*
by *simp*
ultimately have *comp-on-voters-diff*:
 $\forall v \in voters_E E - voters_E E'.$
 $profile_E E' v = (profile_E E \circ the_inv \pi_global) v$
by *auto*
have $\forall v \in voters_E E'. \exists v' \in voters_E E. \pi_global v' = v \wedge \pi' v' = v$
using *bij' imageE $\pi_global_eq-\pi'$*
unfolding *bij-betw-def*
by (*metis (mono-tags, opaque-lifting)*)
hence $\forall v \in voters_E E'. \exists v' \in voters_E E. v' = the_inv \pi_global v \wedge \pi' v' = v$
using *inv*
by *metis*
hence $\forall v \in voters_E E'.$
 $the_inv \pi_global v \in voters_E E \wedge \pi' (the_inv \pi_global v) = v$
by *blast*
moreover have $\forall v' \in voters_E E. profile_E E' (\pi' v') = profile_E E v'$
using *π' -perm $bij-\pi$ $bij-betwE$ *mem-Collect-eq**
by *fastforce*
ultimately have *comp-on-E'-voters*:
 $\forall v \in voters_E E'. profile_E E' v = (profile_E E \circ the_inv \pi_global) v$
unfolding *comp-def*
by *metis*
have $\forall v \in UNIV - voters_E E - voters_E E'.$
 $profile_E E' v = (profile_E E \circ the_inv \pi_global) v$
using *$\pi_global-non-voters$ *default-non-v* *default-non-v'* *inv**
by *simp*
hence $profile_E E' = profile_E E \circ the_inv \pi_global$
using *comp-on-voters-diff comp-on-E'-voters*
by *blast*
moreover have $\pi_global ' (voters_E E) = voters_E E'$
using *$\pi_global_eq-\pi'$ bij' $bij-betw-imp-surj-on$*
by *fastforce*
ultimately have $E' = rename \pi_global E$
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*
 $action-induced-rel.simps \varphi-anon.simps anonymity \mathcal{G} -def$
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 $\text{rename } \pi \circ \text{rename } \pi' = \text{rename } (\pi \circ \pi')$

proof

fix $E :: ('a, 'v) \text{ Election}$

have $\text{rename } \pi' E =$

$(\text{alternatives-}\mathcal{E} E, \pi' \text{ ` } (\text{voters-}\mathcal{E} E), (\text{profile-}\mathcal{E} E) \circ (\text{the-inv } \pi'))$

unfolding *alternatives-}\mathcal{E}.simps voters-}\mathcal{E}.simps profile-}\mathcal{E}.simps*

using *prod.collapse rename.simps*

by *metis*

hence

$(\text{rename } \pi \circ \text{rename } \pi') E =$

$\text{rename } \pi (\text{alternatives-}\mathcal{E} E, \pi' \text{ ` } (\text{voters-}\mathcal{E} E), (\text{profile-}\mathcal{E} E) \circ (\text{the-inv } \pi'))$

unfolding *comp-def*

by *presburger*

also have

$\dots = (\text{alternatives-}\mathcal{E} E, \pi \text{ ` } \pi' \text{ ` } (\text{voters-}\mathcal{E} E),$
 $(\text{profile-}\mathcal{E} E) \circ (\text{the-inv } \pi') \circ (\text{the-inv } \pi))$

by *simp*

also have

$\dots = (\text{alternatives-}\mathcal{E} E, (\pi \circ \pi') \text{ ` } (\text{voters-}\mathcal{E} E),$
 $(\text{profile-}\mathcal{E} E) \circ \text{the-inv } (\pi \circ \pi'))$

using *assms the-inv-comp[of \pi - - \pi']*

unfolding *comp-def image-image*

by *simp*

finally show $\text{rename } \pi \circ \text{rename } \pi' E = \text{rename } (\pi \circ \pi') E$

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 anonymity_G well-formed-elections \varphi-anon well-formed-elections

proof (*unfold group-action-def group-hom-def anonymity_G-def*

group-hom-axioms-def hom-def, intro conjI group-BijGroup, safe)

fix $\pi :: 'v \Rightarrow 'v$

assume *bij-carrier*: $\pi \in \text{carrier } (\text{BijGroup UNIV})$

hence *bij-\pi*: *bij* π

using *rewrite-carrier*

by *blast*

hence $\text{rename } \pi \text{ ` } \text{well-formed-elections} = \text{well-formed-elections}$

using *rename-surj bij-\pi*

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 $\pi' :: 'v \Rightarrow 'v$
assume *bij-carrier*: $\pi' \in$ *carrier* (*BijGroup* *UNIV*)
hence *bij- π'* : *bij* π'
using *rewrite-carrier*
by *blast*
hence *rename* $\pi' \text{ ' well-formed-elections = well-formed-elections$
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* $\pi' \text{ ' well-formed-elections} \subseteq$ *well-formed-elections*
using *bij-betw-imp-surj-on*
by *blast*
moreover have φ -anon *well-formed-elections* $\pi' \in$ *extensional* *well-formed-elections*
unfolding *extensional-def*
by *force*
ultimately have *bij-car-elect'*:
 φ -anon *well-formed-elections* $\pi' \in$ *carrier* (*BijGroup* *well-formed-elections*)
unfolding *BijGroup-def* *Bij-def*
by *simp*
have
 φ -anon *well-formed-elections* π
 \otimes *BijGroup* *well-formed-elections* (φ -anon *well-formed-elections*) $\pi' =$

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

qed

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

1.10.4 Neutrality Lemmas

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


```

lemma rel-rename-sound:
  fixes
     $\pi :: 'a \Rightarrow 'a$  and
     $r :: 'a \text{ rel}$  and
     $A :: 'a \text{ set}$ 
  assumes inj  $\pi$ 
  shows
     $\text{refl-on } A \ r \longrightarrow \text{refl-on } (\pi \text{ ` } A) \ (\text{rel-rename } \pi \ r)$  and
     $\text{antisym } r \longrightarrow \text{antisym } (\text{rel-rename } \pi \ r)$  and
     $\text{total-on } A \ r \longrightarrow \text{total-on } (\pi \text{ ` } A) \ (\text{rel-rename } \pi \ r)$  and
     $\text{Relation.trans } r \longrightarrow \text{Relation.trans } (\text{rel-rename } \pi \ r)$ 
proof (unfold antisym-def total-on-def Relation.trans-def, safe)
  assume  $\text{refl-on } A \ r$ 
  thus  $\text{refl-on } (\pi \text{ ` } A) \ (\text{rel-rename } \pi \ r)$ 
    unfolding refl-on-def rel-rename.simps
    by blast
next
  fix  $a \ b :: 'a$ 
  assume
     $(a, b) \in \text{rel-rename } \pi \ r$  and
     $(b, a) \in \text{rel-rename } \pi \ r$ 
  then obtain
     $c \ c' \ d \ d' :: 'a$  where
       $c\text{-rel-}d: (c, d) \in r$  and
       $d'\text{-rel-}c': (d', c') \in r$  and
       $\pi_c\text{-eq-}a: \pi \ c = a$  and
       $\pi_{c'}\text{-eq-}a: \pi \ c' = a$  and
       $\pi_d\text{-eq-}b: \pi \ d = b$  and
       $\pi_{d'}\text{-eq-}b: \pi \ d' = b$ 
    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  $d'\text{-rel-}c' \ c\text{-rel-}d$ 
    by simp
  thus  $a = b$ 
    using  $\pi_c\text{-eq-}a \ \pi_d\text{-eq-}b$ 
    by simp
next
  fix  $a \ b :: 'a$ 
  assume
     $\text{total}: \forall x \in A. \forall y \in A. x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r$  and
     $a\text{-in-}A: a \in A$  and
     $b\text{-in-}A: b \in A$  and
     $\pi_a\text{-neq-}\pi_b: \pi \ a \neq \pi \ b$  and

```

$\pi_b\text{-not-rel-}\pi_a: (\pi\ b, \pi\ a) \notin \text{rel-rename } \pi\ r$
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 $(\pi\ a, \pi\ b) \in \text{rel-rename } \pi\ r$
unfolding *rel-rename.simps*
by *blast*
next
fix $a\ b\ c :: 'a$
assume
 $(a, b) \in \text{rel-rename } \pi\ r$ **and**
 $(b, c) \in \text{rel-rename } \pi\ r$
then obtain
 $d\ e\ s\ t :: 'a$ **where**
 $d\text{-rel-}e: (d, e) \in r$ **and**
 $s\text{-rel-}t: (s, t) \in r$ **and**
 $\pi_d\text{-eq-}a: \pi\ d = a$ **and**
 $\pi_s\text{-eq-}b: \pi\ s = b$ **and**
 $\pi_t\text{-eq-}c: \pi\ t = c$ **and**
 $\pi_e\text{-eq-}b: \pi\ e = b$
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 $(a, c) \in \text{rel-rename } \pi\ r$
unfolding *rel-rename.simps*
using $\pi_d\text{-eq-}a\ \pi_t\text{-eq-}c$
by *blast*
qed

lemma *rename-subset:*
fixes
 $r\ s :: 'a\ \text{rel}$ **and**
 $a\ b :: 'a$ **and**
 $\pi :: 'a \Rightarrow 'a$
assumes
 $\text{bij-}\pi: \text{bij } \pi$ **and**
 $\text{rel-rename } \pi\ r = \text{rel-rename } \pi\ s$ **and**

```

      (a, b) ∈ r
    shows (a, b) ∈ s
  proof -
    have (π a, π b) ∈ {(π a, π b) | a b. (a, b) ∈ s}
      using assms
      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 (a, b) ∈ s
      by blast
  qed

```

lemma *rel-rename-bij*:

```

  fixes π :: 'a ⇒ '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 $\text{rel-rename } \pi \ (\text{rel-rename } (\text{the-inv } \pi) \ r) = r$
 by *simp*
 thus $\exists \ s. \ r = \text{rel-rename } \pi \ s$
 by *blast*
 qed

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

lemma *well-formed-elects-closed*:
 fixes
 $A \ A' :: 'a \text{ set}$ and
 $V \ V' :: 'v \text{ set}$ and
 $p \ p' :: ('a, 'v) \text{ Profile}$ and
 $\pi :: 'a \Rightarrow 'a$
 assumes
 bij- π : *bij* π and
 wf-elects: $(A, V, p) \in \text{well-formed-elections}$ and
 renamed: $(A', V', p') = \text{alternatives-rename } \pi \ (A, V, p)$
 shows $(A', V', p') \in \text{well-formed-elections}$
proof –
 have
 $A' = \pi \ ' \ A$ and
 $V = V'$
 using *renamed*
 by (*simp, simp*)
 moreover from *this* have $\forall \ v \in V'. \text{linear-order-on } A \ (p \ v)$
 using *wf-elects*

```

    unfolding well-formed-elections-def profile-def
    by simp
  moreover have  $\forall v \in V'. p' v = \text{rel-rename } \pi (p v)$ 
    using renamed
    by simp
  ultimately have  $\forall v \in V'. \text{linear-order-on } A' (p' v)$ 
    unfolding linear-order-on-def partial-order-on-def preorder-on-def
    using bij- $\pi$  rel-rename-sound bij-is-inj
    by metis
  thus  $(A', V', p') \in \text{well-formed-elections}$ 
    unfolding well-formed-elections-def profile-def
    by simp
qed

lemma alternatives-rename-bij:
  fixes  $\pi :: ('a \Rightarrow 'a)$ 
  assumes  $\text{bij-}\pi$ :  $\text{bij } \pi$ 
  shows  $\text{bij-betw } (\text{alternatives-rename } \pi) \text{ well-formed-elections well-formed-elections}$ 
  proof (unfold bij-betw-def, safe, intro inj-onI, clarify)
    fix
       $A A' :: 'a \text{ set}$  and
       $V V' :: 'v \text{ set}$  and
       $p p' :: ('a, 'v) \text{ Profile}$ 
    assume
      renamed:  $\text{alternatives-rename } \pi (A, V, p) = \text{alternatives-rename } \pi (A', V', p')$ 
    hence
       $\pi\text{-eq-img-}A\text{-}A'$ :  $\pi \text{ ` } A = \pi \text{ ` } A'$  and
      rel-rename-eq:  $\text{rel-rename } \pi \circ p = \text{rel-rename } \pi \circ p'$ 
      by (simp, simp)
    hence  $(\text{the-inv } (\text{rel-rename } \pi)) \circ \text{rel-rename } \pi \circ p =$ 
       $(\text{the-inv } (\text{rel-rename } \pi)) \circ \text{rel-rename } \pi \circ p'$ 
      using fun.map-comp
      by metis
    also have  $(\text{the-inv } (\text{rel-rename } \pi)) \circ \text{rel-rename } \pi = \text{id}$ 
      using bij- $\pi$  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- $\pi$   $\pi\text{-eq-img-}A\text{-}A'$  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

```

$A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$
assume *renamed*: $(A', V', p') = \text{alternatives-rename } \pi \ (A, V, p)$
hence *rewr*: $V = V' \wedge A' = \pi \text{ ` } A$
by *simp*
moreover **assume** $(A, V, p) \in \text{well-formed-elections}$
ultimately **have** $\forall v \in V'. \text{linear-order-on } A \ (p \ v)$
unfolding *well-formed-elections-def profile-def*
by *simp*
moreover **have** $\forall v \in V'. p' \ v = \text{rel-rename } \pi \ (p \ v)$
using *renamed*
by *simp*
ultimately **have** $\forall v \in V'. \text{linear-order-on } A' \ (p' \ v)$
unfolding *linear-order-on-def partial-order-on-def preorder-on-def*
using *rewr rel-rename-sound bij-is-inj assms*
by *metis*
thus $(A', V', p') \in \text{well-formed-elections}$
unfolding *well-formed-elections-def profile-def*
by *simp*
next
fix
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assume *wf-elects*: $(A, V, p) \in \text{well-formed-elections}$
have *rename-inv*:
 $\text{alternatives-rename } (\text{the-inv } \pi) \ (A, V, p) =$
 $((\text{the-inv } \pi) \text{ ` } A, V, \text{rel-rename } (\text{the-inv } \pi) \circ p)$
by *simp*
also **have**
 $\text{alternatives-rename } \pi \ ((\text{the-inv } \pi) \text{ ` } A, V, \text{rel-rename } (\text{the-inv } \pi) \circ p) =$
 $(\pi \text{ ` } (\text{the-inv } \pi) \text{ ` } A, V, \text{rel-rename } \pi \circ \text{rel-rename } (\text{the-inv } \pi) \circ p)$
by *auto*
also **have** $\dots = (A, V, \text{rel-rename } (\pi \circ \text{the-inv } \pi) \circ p)$
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** $(A, V, \text{rel-rename } (\pi \circ \text{the-inv } \pi) \circ p) = (A, V, \text{rel-rename } \text{id} \circ p)$
using *UNIV-I assms comp-apply f-the-inv-into-f-bij-betw id-apply*
by *metis*
finally **have**
 $\text{alternatives-rename } \pi \ (\text{alternatives-rename } (\text{the-inv } \pi) \ (A, V, p)) =$
 (A, V, p)
unfolding *rel-rename.simps*
by *auto*
moreover **have** $\text{alternatives-rename } (\text{the-inv } \pi) \ (A, V, p) \in \text{well-formed-elections}$
using *rename-inv wf-elects well-formed-elects-closed bij- π bij-betw-the-inv-into*
by (*metis (no-types)*)
ultimately **show** $(A, V, p) \in \text{alternatives-rename } \pi \text{ ` } \text{well-formed-elections}$

using *image-eqI*
 by *metis*
 qed

interpretation φ -neutral-action: group-action neutrality_G well-formed-elections
 φ -neutral well-formed-elections

proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def
 neutrality_G-def, intro conjI group-BijGroup, safe)

fix $\pi :: 'a \Rightarrow 'a$

assume *bij-carrier*: $\pi \in \text{carrier } (\text{BijGroup UNIV})$

hence

bij-betw (φ -neutral well-formed-elections π) well-formed-elections well-formed-elections

using *universal-set-carrier-imp-bij-group alternatives-rewrite-bij bij-betw-ext*

unfolding φ -neutral.simps

by *metis*

thus *bij-carrier-elect*:

φ -neutral well-formed-elections $\pi \in \text{carrier } (\text{BijGroup well-formed-elections})$

unfolding φ -neutral.simps BijGroup-def Bij-def extensional-def

by *simp*

fix $\pi' :: 'a \Rightarrow 'a$

assume *bij-carrier'*: $\pi' \in \text{carrier } (\text{BijGroup UNIV})$

hence

bij-betw (φ -neutral well-formed-elections π') well-formed-elections well-formed-elections

using *universal-set-carrier-imp-bij-group alternatives-rewrite-bij bij-betw-ext*

unfolding φ -neutral.simps

by *metis*

hence *bij-carrier-elect'*:

φ -neutral well-formed-elections $\pi' \in \text{carrier } (\text{BijGroup well-formed-elections})$

unfolding φ -neutral.simps BijGroup-def Bij-def extensional-def

by *simp*

hence *carrier-elects*:

φ -neutral well-formed-elections $\pi \in \text{carrier } (\text{BijGroup well-formed-elections})$

$\wedge \varphi$ -neutral well-formed-elections $\pi' \in \text{carrier } (\text{BijGroup well-formed-elections})$

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 $\pi' \text{ 'well-formed-elections } \subseteq \text{ well-formed-elections}$

using *bij-betw-imp-surj-on*

by *blast*

have φ -neutral well-formed-elections π

$\otimes \text{BijGroup well-formed-elections } \varphi$ -neutral well-formed-elections $\pi' =$
extensional-continuation

(φ -neutral well-formed-elections $\pi \circ \varphi$ -neutral well-formed-elections π')
well-formed-elections

using *carrier-elects rewrite-mult*

by *auto*

moreover have
 $\forall \mathcal{E} \in \text{well-formed-elections. extensional-continuation}$
 $(\varphi\text{-neutral well-formed-elections } \pi \circ \varphi\text{-neutral well-formed-elections } \pi') \mathcal{E} =$
 $\text{well-formed-elections } \mathcal{E} =$
 $(\varphi\text{-neutral well-formed-elections } \pi \circ \varphi\text{-neutral well-formed-elections } \pi') \mathcal{E}$
by simp
moreover have
 $\forall \mathcal{E} \in \text{well-formed-elections.}$
 $(\varphi\text{-neutral well-formed-elections } \pi \circ \varphi\text{-neutral well-formed-elections } \pi') \mathcal{E} =$
 $\text{alternatives-rename } \pi (\text{alternatives-rename } \pi' \mathcal{E})$
unfolding $\varphi\text{-neutral.simps}$
using wf-closed'
by auto
moreover have
 $\forall \mathcal{E} \in \text{well-formed-elections.}$
 $\text{alternatives-rename } \pi (\text{alternatives-rename } \pi' \mathcal{E}) =$
 $\text{alternatives-rename } (\pi \circ \pi') \mathcal{E}$
using $\text{alternatives-rename-comp comp-apply}$
by metis
moreover have
 $\forall \mathcal{E} \in \text{well-formed-elections. alternatives-rename } (\pi \circ \pi') \mathcal{E} =$
 $\varphi\text{-neutral well-formed-elections } (\pi \otimes \text{BijGroup UNIV } \pi') \mathcal{E}$
using $\text{rewrite-mult-univ bij-carrier bij-carrier'}$
unfolding $\varphi\text{-anon.simps } \varphi\text{-neutral.simps extensional-continuation.simps}$
by metis
moreover have
 $\forall \mathcal{E}. \mathcal{E} \notin \text{well-formed-elections} \longrightarrow$
 $\text{extensional-continuation}$
 $(\varphi\text{-neutral well-formed-elections } \pi \circ \varphi\text{-neutral well-formed-elections } \pi')$
 $\text{well-formed-elections } \mathcal{E} = \text{undefined}$
by simp
moreover have
 $\forall \mathcal{E}. \mathcal{E} \notin \text{well-formed-elections}$
 $\longrightarrow \varphi\text{-neutral well-formed-elections } (\pi \otimes \text{BijGroup UNIV } \pi') \mathcal{E} = \text{undefined}$
by simp
ultimately have
 $\forall \mathcal{E}. \varphi\text{-neutral well-formed-elections } (\pi \otimes \text{BijGroup UNIV } \pi') \mathcal{E} =$
 $(\varphi\text{-neutral well-formed-elections } \pi$
 $\otimes \text{BijGroup well-formed-elections } \varphi\text{-neutral well-formed-elections } \pi') \mathcal{E}$
by metis
thus
 $\varphi\text{-neutral well-formed-elections } (\pi \otimes \text{BijGroup UNIV } \pi') =$
 $\varphi\text{-neutral well-formed-elections } \pi$
 $\otimes \text{BijGroup well-formed-elections } \varphi\text{-neutral well-formed-elections } \pi'$
by blast
qed

interpretation $\psi\text{-neutral}_c\text{-action: group-action neutrality}_{\mathcal{G}} \text{ UNIV } \psi\text{-neutral}_c$
proof $(\text{unfold group-action-def group-hom-def hom-def neutrality}_{\mathcal{G}}\text{-def}$


```

      group-hom-axioms-def, intro conjI group-BijGroup, safe)
fix  $\pi :: 'a \Rightarrow 'a$ 
assume  $\pi \in \text{carrier } (\text{BijGroup UNIV})$ 
hence bij  $\pi$ 
  unfolding BijGroup-def Bij-def
  by simp
thus  $\psi\text{-neutral}_c \pi \in \text{carrier } (\text{BijGroup UNIV})$ 
  unfolding  $\psi\text{-neutral}_c.\text{simps}$ 
  using rewrite-carrier
  by blast
fix  $\pi' :: 'a \Rightarrow 'a$ 
show  $\psi\text{-neutral}_c (\pi \otimes \text{BijGroup UNIV } \pi') =$ 
   $\psi\text{-neutral}_c \pi \otimes \text{BijGroup UNIV } \psi\text{-neutral}_c \pi'$ 
  unfolding  $\psi\text{-neutral}_c.\text{simps}$ 
  by safe
qed

interpretation  $\psi\text{-neutral}_w\text{-action}$ : group-action neutralityG UNIV  $\psi\text{-neutral}_w$ 
proof (unfold group-action-def group-hom-def hom-def neutralityG-def
  group-hom-axioms-def, intro conjI group-BijGroup, safe)
fix  $\pi :: 'a \Rightarrow 'a$ 
assume bij-carrier:  $\pi \in \text{carrier } (\text{BijGroup UNIV})$ 
hence bij  $\pi$ 
  unfolding neutralityG-def BijGroup-def Bij-def
  by simp
hence bij  $(\psi\text{-neutral}_w \pi)$ 
  unfolding neutralityG-def BijGroup-def Bij-def  $\psi\text{-neutral}_w.\text{simps}$ 
  using rel-rename-bij
  by blast
thus group-elem:  $\psi\text{-neutral}_w \pi \in \text{carrier } (\text{BijGroup UNIV})$ 
  using rewrite-carrier
  by blast
moreover fix  $\pi' :: 'a \Rightarrow 'a$ 
assume bij-carrier':  $\pi' \in \text{carrier } (\text{BijGroup UNIV})$ 
hence bij  $\pi'$ 
  unfolding neutralityG-def BijGroup-def Bij-def
  by simp
hence bij  $(\psi\text{-neutral}_w \pi')$ 
  unfolding neutralityG-def BijGroup-def Bij-def  $\psi\text{-neutral}_w.\text{simps}$ 
  using rel-rename-bij
  by blast
hence group-elem':  $\psi\text{-neutral}_w \pi' \in \text{carrier } (\text{BijGroup UNIV})$ 
  using rewrite-carrier
  by blast
moreover have  $\psi\text{-neutral}_w (\pi \otimes \text{BijGroup UNIV } \pi') = \psi\text{-neutral}_w (\pi \circ \pi')$ 
  using bij-carrier bij-carrier' rewrite-mult-univ
  by metis
ultimately show
   $\psi\text{-neutral}_w (\pi \otimes \text{BijGroup UNIV } \pi') =$ 

```

```

       $\psi\text{-neutral}_w \pi \otimes \text{BijGroup UNIV } \psi\text{-neutral}_w \pi'$ 
    using rewrite-mult-univ
    by fastforce
  qed

lemma neutrality-SCF: is-symmetry ( $\lambda \mathcal{E}. \text{limit-SCF (alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV}$ )
  (action-induced-equivariance (carrier neutralityG) well-formed-elections
   ( $\varphi$ -neutral well-formed-elections) (set-action  $\psi\text{-neutral}_c$ ))
proof (unfold rewrite-equivariance, safe)
  fix
     $\pi :: 'a \Rightarrow 'a$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: 'v \Rightarrow ('a \times 'a) \text{ set}$  and
     $r :: 'a$ 
  assume
    carrier- $\pi$ :  $\pi \in \text{carrier neutrality}_G$  and
    prof:  $(A, V, p) \in \text{well-formed-elections}$ 
  {
    moreover assume
       $r \in \text{limit-SCF}$ 
      ( $\text{alternatives-}\mathcal{E} (\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) \text{ UNIV}$ )
    ultimately show
       $r \in \text{set-action } \psi\text{-neutral}_c \pi (\text{limit-SCF (alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV})$ 
    by auto
  }
  {
    moreover assume
       $r \in \text{set-action } \psi\text{-neutral}_c \pi (\text{limit-SCF (alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV})$ 
    ultimately show
       $r \in \text{limit-SCF}$ 
      ( $\text{alternatives-}\mathcal{E} (\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) \text{ UNIV}$ )
    using prof
    by simp
  }
}
qed

lemma neutrality-SWF: is-symmetry ( $\lambda \mathcal{E}. \text{limit-SWF (alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV}$ )
  (action-induced-equivariance (carrier neutralityG) well-formed-elections
   ( $\varphi$ -neutral well-formed-elections) (set-action  $\psi\text{-neutral}_w$ ))
proof (unfold rewrite-equivariance voters- $\mathcal{E}. \text{sims profile-}\mathcal{E}. \text{sims set-action.sims}$ ,
  safe)
  show  $\bigwedge \pi A V p r.$ 
     $\pi \in \text{carrier neutrality}_G \implies (A, V, p) \in \text{well-formed-elections}$ 
     $\implies r \in \text{limit-SWF}$ 
    ( $\text{alternatives-}\mathcal{E} (\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) \text{ UNIV}$ )
     $\implies r \in \psi\text{-neutral}_w \pi \text{ ' limit-SWF (alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV}$ 
  proof -
    fix

```

$\pi :: 'c \Rightarrow 'c$ **and**
 $A :: 'c$ *set* **and**
 $V :: 'v$ *set* **and**
 $p :: ('c, 'v)$ *Profile* **and**
 $r :: 'c$ *rel*
let $?r\text{-inv} = \psi\text{-neutral}_w$ ($the\text{-inv}$ π) r
assume
 $carrier\text{-}\pi$: $\pi \in carrier\ neutrality_G$ **and**
 $prof$: $(A, V, p) \in well\text{-formed}\text{-elections}$
have $inv\text{-}carrier$: $the\text{-inv}$ $\pi \in carrier\ neutrality_G$
using $carrier\text{-}\pi$ $bij\text{-}betw\text{-}the\text{-inv}\text{-}into$
unfolding $neutrality_G\text{-}def$ $rewrite\text{-}carrier$
by $simp$
moreover **have** $the\text{-inv}$ $\pi \circ \pi = id$
using $carrier\text{-}\pi$ $universal\text{-}set\text{-}carrier\text{-}imp\text{-}bij\text{-}group$ $bij\text{-}is\text{-}inj$ $the\text{-inv}\text{-}f\text{-}f$
unfolding $neutrality_G\text{-}def$
by $fastforce$
moreover **have** 1 $neutrality_G = id$
unfolding $neutrality_G\text{-}def$ $BijGroup\text{-}def$
by $auto$
ultimately **have** $the\text{-inv}$ $\pi \otimes neutrality_G$ $\pi = 1$ $neutrality_G$
using $carrier\text{-}\pi$ $rewrite\text{-}mult\text{-}univ$
unfolding $neutrality_G\text{-}def$
by $metis$
hence inv $neutrality_G$ $\pi = the\text{-inv}$ π
using $carrier\text{-}\pi$ $inv\text{-}carrier$ $\psi\text{-neutral}_c\text{-}action$. $group\text{-}hom$ $group$. $inv\text{-}closed$
 $group$. $inv\text{-}solve\text{-}right$ $group$. $l\text{-}inv$ $group\text{-}BijGroup$ $group\text{-}hom$. $hom\text{-}one$
 $group\text{-}hom$. $one\text{-}closed$
unfolding $neutrality_G\text{-}def$
by $metis$
hence $neutral\text{-}r$: $r = \psi\text{-neutral}_w$ π $?r\text{-inv}$
using $carrier\text{-}\pi$ $inv\text{-}carrier$ $iso\text{-}tuple\text{-}UNIV\text{-}I$ $\psi\text{-neutral}_w\text{-}action$. $orbit\text{-}sym\text{-}aux$
by $metis$
have $bij\text{-}inv$: bij ($the\text{-inv}$ π)
using $carrier\text{-}\pi$ $bij\text{-}betw\text{-}the\text{-inv}\text{-}into$ $universal\text{-}set\text{-}carrier\text{-}imp\text{-}bij\text{-}group$
unfolding $neutrality_G\text{-}def$
by $blast$
hence $the\text{-inv}\text{-}\pi$: ($the\text{-inv}$ π) ' π ' $A = A$
using $carrier\text{-}\pi$ $UNIV\text{-}I$ $bij\text{-}betw\text{-}imp\text{-}surj$ $universal\text{-}set\text{-}carrier\text{-}imp\text{-}bij\text{-}group$
 $f\text{-}the\text{-inv}\text{-}into\text{-}f\text{-}bij\text{-}betw$ $image\text{-}f\text{-}inv\text{-}f$ $surj\text{-}imp\text{-}inv\text{-}eq$
unfolding $neutrality_G\text{-}def$
by $metis$
assume
 $r \in limit\text{-}SWF$
 $(alternatives\text{-}\mathcal{E} (\varphi\text{-neutral well}\text{-formed}\text{-elections } \pi (A, V, p)))$ $UNIV$
hence $r \in limit\text{-}SWF$ (π ' A) $UNIV$
unfolding $\varphi\text{-neutral.simps}$
using $prof$
by $simp$

hence $\text{linear-order-on } (\pi \text{ ' } A) \text{ } r$
by *auto*
hence $\text{lin-inv: linear-order-on } A \text{ } ?r\text{-inv}$
using *rel-rename-sound bij-inv bij-is-inj the-inv- π*
unfolding $\psi\text{-neutral}_w.\text{sims linear-order-on-def preorder-on-def partial-order-on-def}$
by *metis*
hence $\forall (a, b) \in ?r\text{-inv}. a \in A \wedge b \in A$
unfolding $\text{linear-order-on-def partial-order-on-def preorder-on-def}$
using *reft-on-def'*
by *metis*
hence $\text{limit } A \text{ } ?r\text{-inv} = \{(a, b). (a, b) \in ?r\text{-inv}\}$
by *auto*
also have $\dots = ?r\text{-inv}$
by *blast*
finally have $\dots = \text{limit } A \text{ } ?r\text{-inv}$
by *blast*
hence $?r\text{-inv} \in \text{limit-SWF } (\text{alternatives-}\mathcal{E} (A, V, p)) \text{ } UNIV$
unfolding $\text{limit-SWF.sims alternatives-}\mathcal{E}.\text{sims}$
using *lin-inv UNIV-I fst-conv mem-Collect-eq iso-tuple-UNIV-I CollectI*
by *(metis (mono-tags, lifting))*
thus $\text{lim-el-}\pi$:
 $r \in \psi\text{-neutral}_w \pi \text{ ' } \text{limit-SWF } (\text{alternatives-}\mathcal{E} (A, V, p)) \text{ } UNIV$
using *neutral-r*
by *blast*
qed
moreover
fix
 $\pi :: 'a \Rightarrow 'a \text{ and}$
 $A :: 'a \text{ set and}$
 $V :: 'v \text{ set and}$
 $p :: ('a, 'v) \text{ Profile and}$
 $r :: 'a \text{ rel}$
assume
 $\text{carrier-}\pi: \pi \in \text{carrier neutrality}_G \text{ and}$
 $\text{prof: } (A, V, p) \in \text{well-formed-elections}$
hence $\text{prof-}\pi$:
 $\varphi\text{-neutral well-formed-elections } \pi (A, V, p) \in \text{well-formed-elections}$
using $\varphi\text{-neutral-action.element-image}$
by *blast*
moreover have $\text{inv-group-elem: inv neutrality}_G \pi \in \text{carrier neutrality}_G$
using $\text{carrier-}\pi \psi\text{-neutral}_c\text{-action.group-hom group.inv-closed}$
unfolding group-hom-def
by *metis*
moreover have $\varphi\text{-neutral well-formed-elections } (\text{inv neutrality}_G \pi)$
 $(\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) \in \text{well-formed-elections}$
using $\text{prof } \varphi\text{-neutral-action.element-image inv-group-elem prof-}\pi$
by *metis*
moreover assume $r \in \text{limit-SWF } (\text{alternatives-}\mathcal{E} (A, V, p)) \text{ } UNIV$
hence $r \in \text{limit-SWF}$

```

    (alternatives- $\mathcal{E}$  ( $\varphi$ -neutral well-formed-elections ( $\text{inv } \text{neutrality}_{\mathcal{G}} \pi$ )
      ( $\varphi$ -neutral well-formed-elections  $\pi (A, V, p)$ ))) UNIV
  using  $\varphi$ -neutral-action.orbit-sym-aux carrier- $\pi$  prof
  by metis
ultimately have
   $r \in \psi\text{-neutral}_{\mathcal{W}} (\text{inv } \text{neutrality}_{\mathcal{G}} \pi)$  ‘
    limit-SWF
    (alternatives- $\mathcal{E}$  ( $\varphi$ -neutral well-formed-elections  $\pi (A, V, p)$ )) UNIV
  using prod.collapse
  by metis
thus  $\psi\text{-neutral}_{\mathcal{W}} \pi r \in \text{limit-SWF}$ 
    (alternatives- $\mathcal{E}$  ( $\varphi$ -neutral well-formed-elections  $\pi (A, V, p)$ )) UNIV
  using carrier- $\pi$   $\psi\text{-neutral}_{\mathcal{W}}$ -action.group-action-axioms
     $\psi\text{-neutral}_{\mathcal{W}}$ -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 $\text{reflp-on}' :: 'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $\text{reflp-on}' A r \equiv \text{reflp-on } A (\lambda x y. (x, y) \in r)$

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

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

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

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

1.10.6 Reversal Symmetry Lemmas

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

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

lemma *reverse-rel-lin-ord*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ rel}$
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_G-group*: *group reversal_G*

proof

show $\mathbf{1}_{\text{reversal}_G} \in \text{carrier reversal}_G$
unfolding *reversal_G-def*
by *simp*

next

show $\text{carrier reversal}_G \subseteq \text{Units reversal}_G$
unfolding *reversal_G-def Units-def*
using *reverse-reverse-id*
by *auto*

next

fix $\alpha :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$
show $\alpha \otimes_{\text{reversal}_G} \mathbf{1}_{\text{reversal}_G} = \alpha$
unfolding *reversal_G-def*
by *auto*

assume $\alpha\text{-elem}$: $\alpha \in \text{carrier reversal}_G$

thus $\mathbf{1}_{\text{reversal}_G} \otimes_{\text{reversal}_G} \alpha = \alpha$
unfolding *reversal_G-def*
by *auto*

fix $\alpha' :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$

assume $\alpha'\text{-elem}$: $\alpha' \in \text{carrier reversal}_G$

thus $\alpha \otimes_{\text{reversal}_G} \alpha' \in \text{carrier reversal}_G$
using $\alpha\text{-elem}$ *reverse-reverse-id*
unfolding *reversal_G-def*
by *auto*

fix $z :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$

```

assume  $z \in \text{carrier reversal}_G$ 
thus  $\alpha \otimes_{\text{reversal}_G} \alpha' \otimes_{\text{reversal}_G} z = \alpha \otimes_{\text{reversal}_G} (\alpha' \otimes_{\text{reversal}_G} z)$ 
  using  $\alpha\text{-elem } \alpha'\text{-elem}$ 
  unfolding  $\text{reversal}_G\text{-def}$ 
  by auto
qed

interpretation  $\varphi\text{-reverse-action: group-action reversal}_G \text{ well-formed-elections}$ 
   $\varphi\text{-reverse well-formed-elections}$ 
proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def,
  intro conjI group-BijGroup CollectI ballI funcsetI)
  show  $\text{Group.group reversal}_G$ 
    by safe
next
  show carrier-elect-gen:
     $\bigwedge \pi. \pi \in \text{carrier reversal}_G$ 
     $\implies \varphi\text{-reverse well-formed-elections } \pi \in \text{carrier (BijGroup well-formed-elections)}$ 
  proof –
    fix  $\pi :: 'c \text{ rel} \Rightarrow 'c \text{ rel}$ 
    assume  $\pi \in \text{carrier reversal}_G$ 
    hence  $\pi\text{-cases: } \pi \in \{\text{id}, \text{reverse-rel}\}$ 
      unfolding  $\text{reversal}_G\text{-def}$ 
      by auto
    hence [simp]:  $\text{rel-app } \pi \circ \text{rel-app } \pi = \text{id}$ 
      using reverse-reverse-id
      by fastforce
    have  $\forall \mathcal{E}. \text{rel-app } \pi (\text{rel-app } \pi \mathcal{E}) = \mathcal{E}$ 
      by (simp add: pointfree-idE)
    moreover have  $\forall \mathcal{E} \in \text{well-formed-elections}. \text{rel-app } \pi \mathcal{E} \in \text{well-formed-elections}$ 
      unfolding well-formed-elections-def profile-def
      using  $\pi\text{-cases reverse-rel-lin-ord rel-app.simps fun.map-id}$ 
      by fastforce
    hence  $\text{rel-app } \pi \text{ ` well-formed-elections } \subseteq \text{well-formed-elections}$ 
      by blast
    ultimately have bij-betw (rel-app  $\pi$ ) well-formed-elections well-formed-elections
      using bij-betw-byWitness[of well-formed-elections]
      by blast
    hence bij-betw ( $\varphi\text{-reverse well-formed-elections } \pi$ )
      well-formed-elections well-formed-elections
      unfolding  $\varphi\text{-reverse.simps}$ 
      using bij-betw-ext
      by blast
    moreover have  $\varphi\text{-reverse well-formed-elections } \pi \in \text{extensional well-formed-elections}$ 
      unfolding extensional-def
      by simp
    ultimately show
       $\varphi\text{-reverse well-formed-elections } \pi \in \text{carrier (BijGroup well-formed-elections)}$ 
      unfolding BijGroup-def Bij-def
      by simp

```

qed
moreover $\text{fix } \pi \ \pi' :: 'a \ \text{rel} \Rightarrow 'a \ \text{rel}$
assume
 $\text{rev}: \pi \in \text{carrier reversal}_G$ **and**
 $\text{rev}': \pi' \in \text{carrier reversal}_G$
ultimately have carrier-elect :
 $\varphi\text{-reverse well-formed-elections } \pi \in \text{carrier } (\text{BijGroup well-formed-elections})$
by *blast*
have $\varphi\text{-reverse well-formed-elections } (\pi \otimes \text{reversal}_G \ \pi') =$
 $\text{extensional-continuation } (\text{rel-app } (\pi \circ \pi')) \text{ well-formed-elections}$
unfolding $\text{reversal}_G\text{-def}$
by *simp*
moreover have $\text{rel-app } (\pi \circ \pi') = \text{rel-app } \pi \circ \text{rel-app } \pi'$
using *rel-app.simps*
by *fastforce*
ultimately have
 $\varphi\text{-reverse well-formed-elections } (\pi \otimes \text{reversal}_G \ \pi') =$
 $\text{extensional-continuation } (\text{rel-app } \pi \circ \text{rel-app } \pi') \text{ well-formed-elections}$
by *metis*
moreover have
 $\forall A \ V \ p. \forall v \in V. \text{linear-order-on } A \ (p \ v) \longrightarrow \text{linear-order-on } A \ (\pi' \ (p \ v))$
using *empty-iff id-apply insert-iff rev' reverse-rel-lin-ord*
unfolding *partial-object.simps reversal_G-def*
by *metis*
hence *extensional-continuation*
 $(\varphi\text{-reverse well-formed-elections } \pi \circ \varphi\text{-reverse well-formed-elections } \pi')$
 $\text{well-formed-elections} =$
 $\text{extensional-continuation } (\text{rel-app } \pi \circ \text{rel-app } \pi') \text{ well-formed-elections}$
unfolding *well-formed-elections-def profile-def*
by *fastforce*
moreover have *extensional-continuation*
 $(\varphi\text{-reverse well-formed-elections } \pi \circ \varphi\text{-reverse well-formed-elections } \pi')$
 $\text{well-formed-elections} =$
 $\varphi\text{-reverse well-formed-elections } \pi$
 $\otimes \text{BijGroup well-formed-elections } \varphi\text{-reverse well-formed-elections } \pi'$
using *carrier-elect-gen carrier-elect rev' rewrite-mult*
by *metis*
ultimately show
 $\varphi\text{-reverse well-formed-elections } (\pi \otimes \text{reversal}_G \ \pi') =$
 $\varphi\text{-reverse well-formed-elections } \pi$
 $\otimes \text{BijGroup well-formed-elections } \varphi\text{-reverse well-formed-elections } \pi'$
by *metis*
qed

interpretation $\psi\text{-reverse-action: group-action reversal}_G \ \text{UNIV } \psi\text{-reverse}$
proof (*unfold group-action-def group-hom-def group-hom-axioms-def hom-def $\psi\text{-reverse.simps}$,*
intro conjI group-BijGroup CollectI ballI funcsetI)
show *Group.group reversal_G*
by *safe*


```

next
  fix  $\pi :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$ 
  assume  $\pi \in \text{carrier reversal}_{\mathcal{G}}$ 
  hence  $\pi \in \{id, \text{reverse-rel}\}$ 
    unfolding reversalG-def
    by force
  hence bij  $\pi$ 
    using reverse-reverse-id bij-id insertE o-bij singleton-iff
    by metis
  thus  $\pi \in \text{carrier} (\text{BijGroup UNIV})$ 
    using rewrite-carrier
    by blast
next
  fix  $\pi \pi' :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$ 
  assume
     $\pi \in \text{carrier reversal}_{\mathcal{G}}$  and
     $\pi' \in \text{carrier reversal}_{\mathcal{G}}$ 
  hence bij  $\pi' \wedge \text{bij } \pi$ 
    using singleton-iff comp-apply id-apply involuntary-imp-bij reverse-reverse-id
    unfolding bij-id insert-iff reversalG-def partial-object.select-convs
    by (metis (mono-tags, opaque-lifting))
  hence  $\pi \otimes \text{BijGroup UNIV } \pi' = \pi \circ \pi'$ 
    using rewrite-carrier rewrite-mult-univ
    by blast
  also have  $\dots = \pi \otimes \text{reversal}_{\mathcal{G}} \pi'$ 
    unfolding reversalG-def
    by force
  finally show  $\pi \otimes \text{reversal}_{\mathcal{G}} \pi' = \pi \otimes \text{BijGroup UNIV } \pi'$ 
    by presburger
qed

lemma reversal-symmetry: is-symmetry ( $\lambda \mathcal{E}. \text{limit-SWF} (\text{alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV}$ )
  (action-induced-equivariance (carrier reversalG) well-formed-elections
   ( $\varphi$ -reverse well-formed-elections) (set-action  $\psi$ -reverse))
proof (unfold rewrite-equivariance, clarify)
  fix
     $\pi :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  assume  $\pi \in \text{carrier reversal}_{\mathcal{G}}$ 
  hence cases:  $\pi \in \{id, \text{reverse-rel}\}$ 
    unfolding reversalG-def
    by force
  assume  $(A, V, p) \in \text{well-formed-elections}$ 
  hence eq-A:
    alternatives-}\mathcal{E} (\varphi\text{-reverse well-formed-elections } \pi (A, V, p)) = A
    by simp
  have

```

$\forall r \in \{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A \ (\text{limit } A \ r)\}.$
 $\exists r' \in \text{UNIV}. \text{reverse-rel } r = \text{limit } A \ (\text{reverse-rel } r')$
 $\wedge \text{reverse-rel } r' \in \text{UNIV} \wedge \text{linear-order-on } A \ (\text{limit } A \ (\text{reverse-rel } r'))$
using *reverse-rel-limit[of A] reverse-rel-lin-ord*
by force
hence
 $\forall r \in \{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A \ (\text{limit } A \ r)\}.$
 $\text{reverse-rel } r \in \{\text{limit } A \ (\text{reverse-rel } r')$
 $\mid r'. \text{reverse-rel } r' \in \text{UNIV}$
 $\wedge \text{linear-order-on } A \ (\text{limit } A \ (\text{reverse-rel } r'))\}$
by blast
moreover have
 $\{\text{limit } A \ (\text{reverse-rel } r') \mid$
 $r'. \text{reverse-rel } r' \in \text{UNIV} \wedge \text{linear-order-on } A \ (\text{limit } A \ (\text{reverse-rel } r'))\}$
 $\subseteq \{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A \ (\text{limit } A \ r)\}$
by blast
ultimately have
 $\forall r \in \text{limit-SWF } A \ \text{UNIV}. \text{reverse-rel } r \in \text{limit-SWF } A \ \text{UNIV}$
unfolding *limit-SWF.simps*
by blast
hence subset:
 $\forall r \in \text{limit-SWF } A \ \text{UNIV}. \pi \ r \in \text{limit-SWF } A \ \text{UNIV}$
using cases
by fastforce
hence $\forall r \in \text{limit-SWF } A \ \text{UNIV}. r \in \pi \text{ ' limit-SWF } A \ \text{UNIV}$
using *reverse-reverse-id comp-apply empty-iff id-apply image-eqI insert-iff cases*
by metis
hence $\pi \text{ ' limit-SWF } A \ \text{UNIV} = \text{limit-SWF } A \ \text{UNIV}$
using subset
by blast
hence *set-action ψ -reverse π (limit-SWF A UNIV) = limit-SWF A UNIV*
unfolding *set-action.simps*
by simp
also have
 $\dots = \text{limit-SWF}$
 $(\text{alternatives-}\mathcal{E} \ (\varphi\text{-reverse well-formed-elections } \pi \ (A, V, p))) \ \text{UNIV}$
using eq-A
by simp
finally show
 $\text{limit-SWF} \ (\text{alternatives-}\mathcal{E} \ (\varphi\text{-reverse well-formed-elections } \pi \ (A, V, p))) \ \text{UNIV}$
 $=$
 $\text{set-action } \psi\text{-reverse } \pi \ (\text{limit-SWF} \ (\text{alternatives-}\mathcal{E} \ (A, V, p))) \ \text{UNIV}$
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  $\psi$ -neutral :: ('a  $\Rightarrow$  'a, 'b) binary-fun and
        voter-type :: 'v itself
  assumes
    action-neutral: group-action neutralityG UNIV  $\psi$ -neutral and
    neutrality:
      is-symmetry ( $\lambda \mathcal{E} :: ('a, 'v)$  Election. limit (alternatives- $\mathcal{E}$   $\mathcal{E}$ ) UNIV)
      (action-induced-equivariance (carrier neutralityG)
       well-formed-elections
       ( $\varphi$ -neutral well-formed-elections) (set-action  $\psi$ -neutral))

```

```

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

```

1.11.2 Interpretations

```

global-interpretation SCF-properties: result-properties well-formed-SCF
  limit-SCF  $\psi$ -neutralc
  unfolding result-properties-def result-properties-axioms-def
  using neutrality-SCF  $\psi$ -neutralc-action.group-action-axioms
        SCF-result.result-axioms
  by blast

```

```

global-interpretation SWF-properties: result-properties well-formed-SWF
  limit-SWF  $\psi$ -neutralw
  unfolding result-properties-def result-properties-axioms-def
  using neutrality-SWF  $\psi$ -neutralw-action.group-action-axioms
        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  $(x \in S \wedge (\nexists y. y \in S \wedge f\ y < f\ x)) = (x \in S \wedge (\nexists y. y \in S \wedge g\ y < g\ x))$ 
    by safe
next
  case x-in-S: False
  thus  $(x \in S \wedge (\nexists y. y \in S \wedge f\ y < f\ x)) = (x \in S \wedge (\nexists y. y \in S \wedge g\ y < g\ x))$ 
```

```

proof (cases  $\exists y. (\lambda s. s \in S) y \wedge f y < f x$ )
  case y: True
  then obtain y :: 'a where
     $(\lambda s. s \in S) y \wedge f y < f x$ 
  by metis
  hence  $(\lambda s. s \in S) y \wedge g y < g x$ 
  using x-in-S assms
  by metis
  thus ?thesis
  using y
  by metis
next
  case not-y: False
  have  $\neg (\exists y. (\lambda s. s \in S) y \wedge g y < g x)$ 
  proof (safe)
    fix y :: 'a
    assume
      y  $\in S$  and
      g y < g x
    moreover have  $\forall a \in S. f a = g a$ 
    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  $\in A \wedge l \in S$ }
proof –
  let ?P =  $\lambda A. \text{finite } \{a\#l \mid a\ l. a \in A \wedge l \in S\}$ 
  have  $\forall a\ A'. \text{finite } A' \longrightarrow a \notin A' \longrightarrow ?P\ A' \longrightarrow ?P\ (\text{insert } a\ A')$ 
  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 have finite {a'#l | a' l. a' ∈ insert a A' ∧ l ∈ S}
  by simp
  thus ?P (insert a A')
  by simp
qed
moreover have ?P {}
  by simp
ultimately show ?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 (!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 (!i)
  by auto
  moreover assume
    ∀ i::nat < length l. finite (!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 $l :: 'a \text{ set list}$

shows $\forall l' :: 'a \text{ list}. l' \in \text{listset } l \longrightarrow \text{length } l' = \text{length } l$

proof (*induct l, safe*)

case *Nil*

fix $l :: 'a \text{ list}$

assume $l \in \text{listset } []$

thus $\text{length } l = \text{length } []$

by *simp*

next

case (*Cons a l*)

moreover fix

$a :: 'a \text{ set}$ **and**

$l :: 'a \text{ set list}$ **and**

$m :: 'a \text{ list}$

assume

$\forall l'. l' \in \text{listset } l \longrightarrow \text{length } l' = \text{length } l$ **and**

$m \in \text{listset } (a \# l)$

moreover have

$\forall a' l' :: 'a \text{ set list}. \text{listset } (a' \# l') =$

$\{b \# m \mid b m. b \in a' \wedge m \in \text{listset } l'\}$

by (*simp add: set-Cons-def*)

ultimately show $\text{length } m = \text{length } (a \# l)$

by *force*

qed

lemma *all-ls-elems-in-ls-set*:

fixes $l :: 'a \text{ set list}$

shows $\forall l' \in \text{listset } l. \forall i :: \text{nat} < \text{length } l'. l'[i] \in l[i]$

proof (*induct l, safe*)

case *Nil*

fix

$l' :: 'a \text{ list}$ **and**

$i :: \text{nat}$

assume

$l' \in \text{listset } []$ **and**

$i < \text{length } l'$

thus $l'[i] \in [][i]$

by *simp*

next

case (*Cons a l*)

moreover fix

$a :: 'a \text{ set}$ **and**

$l :: 'a \text{ set list}$ **and**

$l' :: 'a \text{ list}$ **and**

$i :: \text{nat}$

assume

$\forall l' \in \text{listset } l. \forall i :: \text{nat} < \text{length } l'. l'!i \in l!i$ **and**
 $l' \in \text{listset } (a \# l)$ **and**
 $i < \text{length } l'$
moreover from this have $l' \in \text{set-Cons } a (\text{listset } l)$
by *simp*
hence $\exists b m. l' = b \# m \wedge b \in a \wedge m \in (\text{listset } l)$
unfolding *set-Cons-def*
by *simp*
ultimately show $l'!i \in (a \# l)!i$
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 $l :: 'a \text{ set list}$
shows $\forall l'. \text{length } l' = \text{length } l$
 $\wedge (\forall i < \text{length } l'. l'!i \in l!i) \longrightarrow l' \in \text{listset } l$
proof (*induction l, safe*)
case *Nil*
fix $l' :: 'a \text{ list}$
assume $\text{length } l' = \text{length } []$
thus $l' \in \text{listset } []$
by *simp*
next
case (*Cons a l*)
fix
 $l :: 'a \text{ set list}$ **and**
 $l' :: 'a \text{ list}$ **and**
 $s :: 'a \text{ set}$
assume $\text{length } l' = \text{length } (s \# l)$
moreover then obtain
 $t :: 'a \text{ list}$ **and**
 $x :: 'a$ **where**
 $l' \text{-cons}: l' = x \# t$
using *length-Suc-conv*
by *metis*
moreover assume
 $\forall m. \text{length } m = \text{length } l \wedge (\forall i < \text{length } m. m!i \in l!i)$
 $\longrightarrow m \in \text{listset } l$ **and**
 $\forall i < \text{length } l'. l'!i \in (s \# l)!i$
ultimately have
 $x \in s$ **and**
 $t \in \text{listset } l$
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 $l' \in \text{listset } (s \# l)$
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  $\Rightarrow$  'a  $\Rightarrow$  nat where
  rank-l l a = (if a  $\in$  set l then index l a + 1 else 0)

```

```

fun rank-l-idx :: 'a Preference-List  $\Rightarrow$  'a  $\Rightarrow$  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  $\notin$  set p
using assms
by force

```

```

definition above-l :: 'a Preference-List  $\Rightarrow$  'a  $\Rightarrow$  'a Preference-List where
  above-l r a  $\equiv$  take (rank-l r a) r

```

2.1.4 Definition

```

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

```

```

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

```

```

definition pl- $\alpha$  :: 'a Preference-List  $\Rightarrow$  'a Preference-Relation where
  pl- $\alpha$  l  $\equiv$  {(a, b). a  $\lesssim_l$  b}

```

```

lemma rel-trans:
  fixes  $l :: 'a \text{ Preference-List}$ 
  shows trans ( $pl\text{-}\alpha \ l$ )
  unfolding Relation.trans-def  $pl\text{-}\alpha\text{-def}$ 
  by simp

lemma pl- $\alpha$ -lin-order:
  fixes
     $A :: 'a \text{ set}$  and
     $r :: 'a \text{ rel}$ 
  assumes  $r \in pl\text{-}\alpha \text{ 'permutations-of-set } A$ 
  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
     $(x, y) \in r \implies x = y$  and
     $x \in A \implies (x, y) \in r \vee (y, x) \in r$ 
    using assms True
    unfolding  $pl\text{-}\alpha\text{-def}$ 
    by (simp, simp, simp, simp)
  next
  case False
  fix  $x \ y :: 'a$ 
  show  $((refl\text{-on } A \ r \wedge trans \ r) \wedge (\forall x \ y. (x, y) \in r \longrightarrow (y, x) \in r \longrightarrow x = y)) \wedge (\forall x \in A. \forall y \in A. x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r)$ 
  proof (intro conjI ballI allI impI)
    have  $\forall l \in \text{permutations-of-set } A. l \neq []$ 
      using assms False permutations-of-setD
      by force
    hence  $\forall a \in A. \forall l \in \text{permutations-of-set } A. (a, a) \in pl\text{-}\alpha \ l$ 
      unfolding is-less-preferred-than-l.simps
      permutations-of-set-def  $pl\text{-}\alpha\text{-def}$ 
      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  $pl\text{-}\alpha\text{-def}$  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
     $\forall x y. \forall l \in \text{permutations-of-set } A. x \lesssim_l y \wedge y \lesssim_l x \longrightarrow x = y$ 
    using is-less-preferred-than-l.simps index-eq-index-conv nle-le
    unfolding permutations-of-set-def
    by metis
  hence  $\forall x y. \forall l \in \text{pl-}\alpha \text{ 'permutations-of-set } A.$ 
     $(x, y) \in l \wedge (y, x) \in l \longrightarrow x = y$ 
    unfolding pl-}\alpha\text{-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
     $\forall x \in A. \forall y \in A. \forall l \in \text{permutations-of-set } A.$ 
     $x \neq y \wedge (\neg y \lesssim_l x) \longrightarrow x \lesssim_l y$ 
    using is-less-preferred-than-l.simps
    unfolding permutations-of-set-def
    by auto
  hence  $\forall x \in A. \forall y \in A. \forall l \in \text{pl-}\alpha \text{ 'permutations-of-set } A.$ 
     $x \neq y \wedge (y, x) \notin l \longrightarrow (x, y) \in l$ 
    using is-less-preferred-than-l.simps
    unfolding permutations-of-set-def
    unfolding pl-}\alpha\text{-def permutations-of-set-def}
    by blast
  ultimately show  $(x, y) \in r \vee (y, x) \in r$ 
    using assms
    by metis
qed
qed

lemma lin-order-pl-}\alpha\text{:}
  fixes
    r :: 'a rel and

```

```

  A :: 'a set
assumes
  lin-order: linear-order-on A r and
  fin: finite A
shows r ∈ pl-α ' permutations-of-set A
proof -
let ?φ = λ a. card ((underS r a) ∩ A)
let ?inv = the-inv-into A ?φ
let ?l = map (λ x. ?inv x) (rev [0 ..< card A])
have antisym:
  ∀ a ∈ A. ∀ b ∈ A.
    a ∈ (underS r b) ∧ b ∈ (underS r a) ⟶ False
using lin-order
unfolding underS-def linear-order-on-def partial-order-on-def antisym-def
by blast
hence ∀ a ∈ A. ∀ b ∈ A. ∀ c ∈ A.
  a ∈ (underS r b) ⟶ b ∈ (underS r c) ⟶ a ∈ (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:
  ∀ a ∈ A. ∀ b ∈ A. a ∈ (underS r b) ⟶ (underS r a) ⊂ (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: ∀ a ∈ A. ∀ b ∈ A. a ∈ (underS r b) ⟶ ?φ a < ?φ 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:
  ∀ a ∈ A. ∀ b ∈ A. a ≠ b ⟶ a ∈ (underS r b) ∨ b ∈ (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 ∀ a ∈ A. ∀ b ∈ A. a ≠ b ⟶ ?φ a ≠ ?φ b
using order-less-imp-not-eq2
by metis
hence inj: inj-on ?φ A
using inj-on-def
by blast
have in-bounds: ∀ a ∈ A. ?φ a < card A
using CollectD IntD1 card-seteq fin inf-le2 linorder-le-less-linear
unfolding underS-def
by (metis (mono-tags, lifting))
hence ?φ ' A ⊆ {0 ..< card A}
using atLeast0LessThan
by blast

```

moreover have $\text{card } (? \varphi \text{ ' } A) = \text{card } A$
using *inj fin card-image*
by *blast*
ultimately have $? \varphi \text{ ' } A = \{0 \dots \text{card } A\}$
by (*simp add: card-subset-eq*)
hence $\text{bij-}A$: $\text{bij-betw } ? \varphi \text{ ' } A \{0 \dots \text{card } A\}$
using *inj*
unfolding *bij-betw-def*
by *safe*
hence bij-inv : $\text{bij-betw } ? \text{inv } \{0 \dots \text{card } A\} A$
using *bij-betw-the-inv-into*
by *metis*
hence $? \text{inv} \text{ ' } \{0 \dots \text{card } A\} = A$
unfolding *bij-betw-def*
by *metis*
hence $\text{set-eq-}A$: $\text{set } ?l = A$
by *simp*
moreover have $\text{dist-}l$: $\text{distinct } ?l$
using *bij-inv*
unfolding *distinct-map*
using *bij-betw-imp-inj-on*
by *simp*
ultimately have $?l \in \text{permutations-of-set } A$
by *auto*
moreover have index-eq : $\forall a \in A. \text{index } ?l \ a = \text{card } A - 1 - ? \varphi \ a$
proof
fix $a :: 'a$
assume $a\text{-in-}A$: $a \in A$
have $\forall l. \forall i < \text{length } l. (\text{rev } l)!i = l!(\text{length } l - 1 - i)$
using *rev-nth*
by *auto*
hence $\forall i < \text{length } [0 \dots \text{card } A]. (\text{rev } [0 \dots \text{card } A])!i =$
 $[0 \dots \text{card } A]!(\text{length } [0 \dots \text{card } A] - 1 - i)$
by *blast*
moreover have $\forall i < \text{card } A. [0 \dots \text{card } A]!i = i$
by *simp*
moreover have $\text{card-}A\text{-len}$: $\text{length } [0 \dots \text{card } A] = \text{card } A$
by *simp*
ultimately have $\forall i < \text{card } A. (\text{rev } [0 \dots \text{card } A])!i = \text{card } A - 1 - i$
using *diff-Suc-eq-diff-pred diff-less diff-self-eq-0*
 $\text{less-imp-diff-less zero-less-Suc}$
by *metis*
moreover have $\forall i < \text{card } A. ?l!i = ? \text{inv } ((\text{rev } [0 \dots \text{card } A])!i)$
by *simp*
ultimately have $\forall i < \text{card } A. ?l!i = ? \text{inv } (\text{card } A - 1 - i)$
by *presburger*
moreover have
 $\text{card } A - 1 - (\text{card } A - 1 - \text{card } (\text{underS } r \ a \cap A)) =$
 $\text{card } (\text{underS } r \ a \cap A)$

```

    using in-bounds a-in-A
    by auto
  moreover have ?inv (card (underS r a  $\cap$  A)) = a
    using a-in-A inj the-inv-into-f-f
    by fastforce
  ultimately have ?!(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- $\alpha$  ?l = r
proof (intro equalityI, unfold pl- $\alpha$ -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
 $\forall a \in A. \forall b \in A. ?\varphi a < ?\varphi b \longrightarrow a \in \text{underS } r \ b$
 using *mon total-underS antisym order-less-not-sym*
 by *metis*
 hence $?\varphi a < ?\varphi b \longrightarrow a \in \text{underS } r \ b$
 using *element-a el-b in-bounds-a in-bounds-b set-eq-A*
 by *blast*
 hence $?\varphi a < ?\varphi b \longrightarrow (a, b) \in r$
 unfolding *underS-def*
 by *simp*
 moreover have $\forall a \in A. \forall b \in A. ?\varphi a = ?\varphi b \longrightarrow a = b$
 using *mon total-underS antisym order-less-not-sym*
 by *metis*
 hence $?\varphi a = ?\varphi b \longrightarrow a = b$
 using *element-a el-b in-bounds-a in-bounds-b set-eq-A*
 by *blast*
 hence $?\varphi a = ?\varphi b \longrightarrow (a, b) \in r$
 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 $(a, b) \in r$
 by *auto*
 next
 fix $a \ b :: 'a$
 assume $a\text{-}b\text{-rel}: (a, b) \in r$
 hence
 $a\text{-in-}A: a \in A$ and
 $b\text{-in-}A: b \in A$ and
 $a\text{-under-}b\text{-or-eq}: a \in \text{underS } r \ b \vee a = b$
 using *lin-order*
 unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def underS-def*
 by *auto*
 thus
 $a \in \text{set } ?l$ and
 $b \in \text{set } ?l$
 using *bij-inv set-eq-A*
 by *(metis, metis)*
 hence $?\varphi a \leq ?\varphi b$
 using *mon le-eq-less-or-eq a-under-b-or-eq a-in-A b-in-A*
 by *auto*
 thus $\text{index } ?l \ b \leq \text{index } ?l \ a$
 using *index-eq a-in-A b-in-A diff-le-mono2*
 by *metis*
 qed

ultimately show $r \in pl\text{-}\alpha$ ‘ *permutations-of-set A*
 by *auto*
 qed

lemma *index-helper*:
 fixes
 $l :: 'x \text{ list}$ and
 $x :: 'x$
 assumes
 $finite \ (set \ l)$ and
 $distinct \ l$ and
 $x \in set \ l$
 shows $index \ l \ x = card \ \{y \in set \ l. index \ l \ y < index \ l \ x\}$
proof –
 have *bij-l*: $bij\text{-}betw \ (index \ l) \ (set \ l) \ \{0 ..< length \ l\}$
 using *assms* *bij-betw-index*
 by *blast*
 hence $card \ \{y \in set \ l. index \ l \ y < index \ l \ x\} =$
 $card \ (index \ l \ \{y \in set \ l. index \ l \ y < index \ l \ x\})$
 using *CollectD* *bij-betw-same-card* *bij-betw-subset* *subsetI*
 by (*metis* (*no-types*, *lifting*))
 also have $index \ l \ \{y \in set \ l. index \ l \ y < index \ l \ x\} =$
 $\{m \mid m. m \in index \ l \ \{y \in set \ l. index \ l \ y < index \ l \ x\}\}$
 by *blast*
 also have
 $\{m \mid m. m \in index \ l \ \{y \in set \ l. index \ l \ y < index \ l \ x\}\} =$
 $\{m \mid m. m < index \ l \ x\}$
 using *bij-l* *assms* *atLeastLessThan-iff* *bot-nat-0.extremum*
 index-image *index-less-size-conv* *order-less-trans*
 by *metis*
 also have $card \ \{m \mid m. m < index \ l \ x\} = index \ l \ x$
 by *simp*
 finally show ?thesis
 by *simp*
 qed

lemma *pl- α -eq-imp-list-eq*:
 fixes $l \ l' :: 'x \text{ list}$
 assumes
 $fin\text{-}set\text{-}l$: $finite \ (set \ l)$ and
 $set\text{-}eq$: $set \ l = set \ l'$ and
 $dist\text{-}l$: $distinct \ l$ and
 $dist\text{-}l'$: $distinct \ l'$ and
 $pl\text{-}\alpha\text{-eq}$: $pl\text{-}\alpha \ l = pl\text{-}\alpha \ l'$
 shows $l = l'$
proof (*rule ccontr*)
 assume $l \neq l'$
 moreover with *set-eq*
 have $l \neq [] \wedge l' \neq []$


```

    by auto
  ultimately obtain
    i :: nat and
    x :: 'x where
      i < length l and
      l!i ≠ l'!i and
      x = l!i and
    x-in-l: x ∈ 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 ≠ index l' x
    using dist-l index-nth-id nth-index
  by metis
  ultimately have
    card {y ∈ set l. index l y < index l x} ≠
    card {y ∈ 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 ∈ set l and
    y-neq-x: y ≠ x and
    neq-indices:
      (index l y < index l x ∧ index l' y > index l' x)
      ∨ (index l' y < index l' x ∧ 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 ∧ is-less-preferred-than-l y l' x)
    ∨ (is-less-preferred-than-l x l' y ∧ 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) ∈ pl-α l ∧ (x, y) ∉ pl-α l')
    ∨ ((x, y) ∈ pl-α l' ∧ (x, y) ∉ pl-α 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  $pl\text{-}\alpha$  (permutations-of-set  $X$ ) { $r$ . linear-order-on  $X$   $r$ }
proof (unfold bij-betw-def, safe)
  show inj-on  $pl\text{-}\alpha$  (permutations-of-set  $X$ )
    unfolding inj-on-def permutations-of-set-def
    using  $pl\text{-}\alpha\text{-eq-imp-list-eq}$  assms
    by fastforce
next
  fix  $l :: 'x$  list
  assume  $l \in \text{permutations-of-set } X$ 
  thus linear-order-on  $X$  ( $pl\text{-}\alpha$   $l$ )
    using assms  $pl\text{-}\alpha\text{-lin-order}$ 
    by blast
next
  fix  $r :: 'x$  rel
  assume linear-order-on  $X$   $r$ 
  thus  $r \in pl\text{-}\alpha$  'permutations-of-set  $X$ '
    using assms lin-order-pl- $\alpha$ 
    by blast
qed

```

2.1.5 Limited Preference

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

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

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 $pl\text{-}\alpha$ (*limit-l* A l) = *limit* A ($pl\text{-}\alpha$ l)
using *assms*
proof (*induction* l)

```

case Nil
show  $pl\text{-}\alpha$  (limit-l A []) = limit A (pl- $\alpha$  [])
  unfolding pl- $\alpha$ -def
  by simp
next
case (Cons a l)
fix
  a :: 'a and
  l :: 'a list
assume
  wf-imp-limit: well-formed-l l  $\implies$  pl- $\alpha$  (limit-l A l) = limit A (pl- $\alpha$  l) and
  wf-a-l: well-formed-l (a#l)
show  $pl\text{-}\alpha$  (limit-l A (a#l)) = limit A (pl- $\alpha$  (a#l))
proof (unfold limit-l.simps limit.simps, intro equalityI, safe)
  fix b c :: 'a
  assume b-less-c: (b, c)  $\in$  pl- $\alpha$  (filter ( $\lambda$  a. a  $\in$  A) (a#l))
  moreover have limit-preference-list-assoc:
    pl- $\alpha$  (limit-l A l) = limit A (pl- $\alpha$  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- $\alpha$ -def
  by (metis, metis)
thus (b, c)  $\in$  pl- $\alpha$  (a#l)
proof (unfold pl- $\alpha$ -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*:
 $b \in \text{set } (\text{limit-}l \ A \ l) \wedge c \in \text{set } (\text{limit-}l \ A \ l)$
 $\wedge \text{index } (\text{limit-}l \ A \ l) \ c \leq \text{index } (\text{limit-}l \ A \ l) \ b$
 $\longrightarrow b \in \text{set } l \wedge c \in \text{set } l \wedge \text{index } l \ c \leq \text{index } l \ b$
by *auto*
have $b \lesssim_{(\text{filter } (\lambda a. a \in A) (a \# l))} c$
using *b-less-c case-prodD mem-Collect-eq*
unfolding *pl- α -def*
by (*metis (no-types)*)
moreover obtain
 $f \ h :: 'a \Rightarrow 'a \ \text{list} \Rightarrow 'a \Rightarrow 'a$ **and**
 $g :: 'a \Rightarrow 'a \ \text{list} \Rightarrow 'a \Rightarrow 'a \ \text{list}$ **where**
 $\forall \ d \ s \ e. \ d \lesssim_s e \longrightarrow$
 $d = f \ e \ s \ d \wedge s = g \ e \ s \ d \wedge e = h \ e \ s \ d$
 $\wedge f \ e \ s \ d \in \text{set } (g \ e \ s \ d) \wedge h \ e \ s \ d \in \text{set } (g \ e \ s \ d)$
 $\wedge \text{index } (g \ e \ s \ d) \ (h \ e \ s \ d) \leq \text{index } (g \ e \ s \ d) \ (f \ e \ s \ d)$
by *fastforce*
ultimately have
 $b = f \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b$
 $\wedge \text{filter } (\lambda a. a \in A) (a \# l) =$
 $g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b$
 $\wedge c = h \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b$
 $\wedge f \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b$
 $\in \text{set } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\wedge h \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b$
 $\in \text{set } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\wedge \text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(h \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\leq \text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(f \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
by *blast*
moreover have $\text{filter } (\lambda a. a \in A) \ l = \text{limit-}l \ A \ l$
by *simp*
moreover have
 $\text{index } (\text{limit-}l \ A \ l) \ c \neq$
 $\text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(h \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\vee \text{index } (\text{limit-}l \ A \ l) \ b \neq$
 $\text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(f \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\vee \text{index } (\text{limit-}l \ A \ l) \ c \leq \text{index } (\text{limit-}l \ A \ l) \ b$
 $\vee \neg \text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(h \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $\leq \text{index } (g \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
 $(f \ c \ (\text{filter } (\lambda a. a \in A) (a \# l)) \ b)$
by *presburger*
ultimately have $a \neq c \longrightarrow \text{index } (a \# l) \ c \leq \text{index } (a \# l) \ b$

```

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

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

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

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

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

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

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

definition *connex-l* :: 'a set \Rightarrow 'a Preference-List \Rightarrow bool **where**
connex-l A l $\equiv \text{limited } A \ l \wedge (\forall a \in A. \forall b \in A. a \lesssim_l b \vee b \lesssim_l a)$

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

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- $\alpha$  l) b

```

```

unfolding pl- $\alpha$ -def
by simp

theorem above-equiv:
  fixes
    l :: 'a Preference-List and
    a :: 'a
  shows set (above-l l a) = above (pl- $\alpha$  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- $\alpha$  l) a
    using less-preferred-l-rel-equiv pref-imp-in-above
    by metis
next
  fix b :: 'a
  assume b  $\in$  above (pl- $\alpha$  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- $\alpha$  l) a
proof (unfold rank-l.simps rank.simps, cases a  $\in$  set l)
  case True
  moreover have above (pl- $\alpha$  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 ∈ 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 ∈ 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
    reft-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 reft-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
     $\forall (j::nat) \leq n. !j \neq a$  and

```

```

     $l!(n - 1) = a$ 
  shows  $\text{rank-}l\ l\ a = n$ 
  using assms
proof (induction  $l$  arbitrary:  $n$ )
  case Nil
  thus ?case
    by simp
next
fix
   $l :: 'a\ \text{Preference-List}$  and
   $a :: 'a$ 
  case (Cons  $a\ l$ )
  thus ?case
    by simp
qed

lemma ranked-alt-not-at-pos-before:
  fixes
     $l :: 'a\ \text{Preference-List}$  and
     $a :: 'a$  and
     $n :: \text{nat}$ 
  assumes
     $a \in \text{set } l$  and
     $n < (\text{rank-}l\ l\ a) - 1$ 
  shows  $l!n \neq a$ 
  using index-first member-def rank-l.simps
    assms add-diff-cancel-right'
  by metis

lemma pos-in-list-yields-pos:
  fixes
     $l :: 'a\ \text{Preference-List}$  and
     $a :: 'a$ 
  assumes  $a \in \text{set } l$ 
  shows  $l!(\text{rank-}l\ l\ a - 1) = a$ 
  using assms
proof (induction  $l$ )
  case Nil
  thus ?case
    by simp
next
fix
   $l :: 'a\ \text{Preference-List}$  and
   $b :: 'a$ 
  case (Cons  $b\ l$ )
  assume  $a \in \text{set } (b\#l)$ 
  moreover from this
  have  $\text{rank-}l\ (b\#l)\ a = 1 + \text{index } (b\#l)\ a$ 
    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- $\alpha$  l
proof (unfold relation-of-def, safe)
  fix a b :: 'a
  assume a  $\lesssim_l$  b
  moreover have (a  $\lesssim_l$  b) = (a  $\preceq_{(pl-\alpha\ l)}$  b)
    using less-preferred-l-rel-equiv
  by (metis (no-types))
  ultimately show (a, b)  $\in$  pl- $\alpha$  l
    by simp
next
  fix a b :: 'a
  assume (a, b)  $\in$  pl- $\alpha$  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 (Preference-List.pl-α) pl)!n
 else {})

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

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

2.3 Ordered Relation Type

```

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

lemma fin-ordered:
  fixes  $X :: 'x \text{ set}$ 
  assumes finite X
  obtains  $ord :: 'x \text{ rel}$  where
    linear-order-on X ord
proof –
  obtain  $l :: 'x \text{ list}$  where
    set-l: set l = X
  using finite-list assms
  by blast
  let  $?r = pl\text{-}\alpha\ l$ 
  have antisym ?r
    using set-l Collect-mono-iff antisym index-eq-index-conv pl- $\alpha$ -def
  unfolding antisym-def
  by fastforce
  moreover have refl-on X ?r
    using set-l
  unfolding refl-on-def pl- $\alpha$ -def is-less-preferred-than-l.simps
  by blast
  moreover have Relation.trans ?r
    unfolding Relation.trans-def pl- $\alpha$ -def is-less-preferred-than-l.simps
  by auto
  moreover have total-on X ?r
    using set-l
  unfolding total-on-def pl- $\alpha$ -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
     $\bigwedge ord. \text{linear-order-on } X \text{ } ord \implies ?thesis$ 
  ultimately show ?thesis
    by blast
qed

typedef  $'a \text{ Ordered-Preference} =$ 
   $\{p :: 'a :: \text{finite } \text{Preference-Relation}. \text{linear-order-on } (UNIV :: 'a \text{ set}) \ p\}$ 
  morphisms ord2pref pref2ord
proof (unfold mem-Collect-eq)
  have finite (UNIV :: 'a set)
    by simp

```

```

then obtain  $p :: 'a \text{ Preference-Relation}$  where
   $\text{linear-order-on } (UNIV :: 'a \text{ set}) \ p$ 
using fin-ordered
by metis
thus  $\exists \ p :: 'a \text{ Preference-Relation. linear-order } p$ 
by blast
qed

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

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

lemma card-ord-pref:  $\text{card } (UNIV :: 'a :: \text{finite Ordered-Preference set}) =$ 
   $\text{fact } (\text{card } (UNIV :: 'a \text{ set}))$ 
proof –
  let  $?n = \text{card } (UNIV :: 'a \text{ set})$  and
     $?perm = \text{permutations-of-set } (UNIV :: 'a \text{ set})$ 
have  $(UNIV :: 'a \text{ Ordered-Preference set}) =$ 
   $\text{pref2ord } \{p :: 'a \text{ Preference-Relation.}$ 
     $\text{linear-order-on } (UNIV :: 'a \text{ set}) \ p\}$ 
using type-definition-Ordered-Preference type-definition.Abs-image
by blast
moreover have
   $\text{inj-on pref2ord } \{p :: 'a \text{ Preference-Relation.}$ 
     $\text{linear-order-on } (UNIV :: 'a \text{ set}) \ p\}$ 
using inj-onCI pref2ord-inject
by metis
ultimately have
   $\text{bij-betw pref2ord}$ 
   $\{p :: 'a \text{ Preference-Relation.}$ 
     $\text{linear-order-on } (UNIV :: 'a \text{ set}) \ p\}$ 

```

```

      (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- $\alpha$ -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'  $\longrightarrow$  election-equality E' E and
  election-equality E E'  $\longrightarrow$  election-equality E' F
   $\longrightarrow$  election-equality E F
proof (safe)
  have  $\forall E. E = (fst E, fst (snd E), snd (snd E))$ 
  by simp
  thus
    election-equality E E and
    election-equality E E'  $\implies$  election-equality E' E and
    election-equality E E'  $\implies$  election-equality E' F
     $\implies$  election-equality E F
  using election-equality.simps[of
    fst E fst (snd E) snd (snd E)]
    election-equality.simps[of
    fst E' fst (snd E') snd (snd E')]
    fst E fst (snd E) snd (snd E)]
    election-equality.simps[of
    fst E' fst (snd E') snd (snd E')]
    fst F fst (snd F) snd (snd F)]
  by (metis, metis, metis)
qed

```

```

quotient-type ('a, 'v) Election_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_Q :: ('a, 'v) Election_Q ⇒ 'a set where
  fst_Q E = Product-Type.fst (rep-Election_Q E)

fun snd_Q :: ('a, 'v) Election_Q ⇒ 'v set × ('a, 'v) Profile where
  snd_Q E = Product-Type.snd (rep-Election_Q E)

abbreviation alternatives-ℰ_Q :: ('a, 'v) Election_Q ⇒ 'a set where
  alternatives-ℰ_Q E ≡ fst_Q E

abbreviation voters-ℰ_Q :: ('a, 'v) Election_Q ⇒ 'v set where
  voters-ℰ_Q E ≡ Product-Type.fst (snd_Q E)

abbreviation profile-ℰ_Q :: ('a, 'v) Election_Q ⇒ ('a, 'v) Profile where
  profile-ℰ_Q E ≡ Product-Type.snd (snd_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-set s = (if (card s = 1) then (the-inv ( $\lambda$  x. {x}) s) else undefined)
— This is undefined if  $\text{card } s \neq 1$ . Note that " $\text{undefined} = \text{undefined}$ " is the only
provable equality for  $\text{undefined}$ .
```

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  $\pi_Q$  :: ('x  $\Rightarrow$  'y)  $\Rightarrow$  ('x set  $\Rightarrow$  'y) where
   $\pi_Q$  f s = singleton-set (f ` s)
```

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

```
fun inv- $\pi_Q$  :: ('x  $\Rightarrow$  'x set)  $\Rightarrow$  ('x set  $\Rightarrow$  'y)  $\Rightarrow$  ('x  $\Rightarrow$  'y) where
  inv- $\pi_Q$  cls f x = f (cls x)
```

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

3.1.2 Well-Definedness

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

```

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

```

```

lemma singleton-set-def-if-card-one:
  fixes  $s :: 'x \text{ set}$ 
  assumes  $\text{card } s = 1$ 
  shows  $\exists! x. x = \text{singleton-set } s \wedge \{x\} = s$ 
  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
     $r :: 'x \text{ rel}$  and
     $s :: 'x \text{ set}$ 
  assumes
     $f$  respects  $r$  and
    equiv  $s$   $r$ 
  shows  $\forall t \in s // r. \forall x \in t. \pi_Q f t = f x$ 
proof (safe)
  fix
     $t :: 'x \text{ set}$  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
     $x \text{-in-} t: x \in t$ 
  moreover from this have  $r `` \{x\} \in s // r$ 
    using assms quotient-eq-iff equiv-class-eq-iff quotientI
    by metis
  ultimately have  $r \text{-img-elem-} x \text{-eq-} t: r `` \{x\} = t$ 
    using assms quotient-eq-iff Image-singleton-iff
    by metis
  hence  $\{f x \mid y. y \in r `` \{x\}\} = \{f x\}$ 
    using  $x \text{-in-} t$ 
    by blast
  hence  $f ` t = \{f x\}$ 

```

```

    using Setcompr-eq-image r-img-elem-x-eq-t func-eq-x
    by metis
  thus  $\pi_Q f t = f x$ 
    using singleton-set-def-if-card-one is-singletonI
      is-singleton-altdef the-elem-eq
    unfolding  $\pi_Q.simps$ 
    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
     $f :: 'x \text{ set} \Rightarrow 'x$  and
     $r :: 'x \text{ rel}$  and
     $s :: 'x \text{ set}$ 
  assumes equiv s r
  defines induced-fun  $\equiv (inv\text{-}\pi_Q (relation\text{-}class\ r) f)$ 
  shows
    induced-fun respects r and
     $\forall A \in s // r. \pi_Q \text{ induced-fun } A = f A$ 
proof (safe)
  have  $\forall (a, b) \in r. relation\text{-}class\ r\ a = relation\text{-}class\ r\ b$ 
    using assms equiv-class-eq
    unfolding relation-class.simps
    by fastforce
  hence  $\forall (a, b) \in r. induced\text{-}fun\ a = induced\text{-}fun\ b$ 
    unfolding induced-fun-def inv- $\pi_Q.simps$ 
    by auto
  thus induced-fun respects r
    unfolding congruent-def
    by metis
  moreover fix  $A :: 'x \text{ set}$ 
  assume  $A \in s // r$ 
  moreover with assms
  obtain  $a :: 'x$  where
     $a \in A$  and
     $A\text{-eq-rel-class-r-a}: A = relation\text{-}class\ r\ a$ 
    using equiv-Eps-in proj-Eps
    unfolding proj-def relation-class.simps
    by metis
  ultimately have  $\pi_Q \text{ induced-fun } A = induced\text{-}fun\ a$ 
    using pass-to-quotient assms
    by blast
  thus  $\pi_Q \text{ induced-fun } A = f A$ 
    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
     $m :: 'x \text{ monoid}$  and
     $s :: 'y \text{ set}$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$ 
  assumes group-action m s  $\varphi$ 
  shows equiv s (action-induced-rel (carrier m) s  $\varphi$ )
proof (unfold equiv-def refl-on-def sym-def Relation.trans-def
         action-induced-rel.simps, safe)
  fix  $y :: 'y$ 
  assume  $y \in s$ 
  hence  $\varphi \mathbf{1} \ m \ y = y$ 
    using assms group-action.id-eq-one restrict-apply'
    by metis
  thus  $\exists g \in \text{carrier } m. \varphi \ g \ y = y$ 
    using assms group.is-monoid group-hom.axioms
    unfolding group-action-def
    by blast
next
fix
   $y :: 'y$  and
   $g :: 'x$ 
  assume
    y-in-s:  $y \in s$  and
    carrier-g:  $g \in \text{carrier } m$ 
  hence  $y = \varphi \ (\text{inv } m \ g) \ (\varphi \ g \ y)$ 
    using assms
    by (simp add: group-action.orbit-sym-aux)
  thus  $\exists h \in \text{carrier } m. \varphi \ h \ (\varphi \ g \ y) = y$ 
    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:  $y \in s$  and
    carrier-g:  $g \in \text{carrier } m$  and
    carrier-h:  $h \in \text{carrier } m$ 
  hence  $\varphi \ (h \otimes m \ g) \ y = \varphi \ h \ (\varphi \ g \ y)$ 
    using assms
    by (simp add: group-action.composition-rule)

```

```

thus  $\exists f \in \text{carrier } m. \varphi f y = \varphi h (\varphi g y)$ 
  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
   $g \in \text{carrier } m$ 
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
   $g \in \text{carrier } m$ 
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
     $A :: 'a \text{ set}$  and
     $N :: 'b \Rightarrow \text{nat}$  and
     $B :: 'b \text{ set}$ 
  assumes

```

```

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

```

$\text{finite } X \implies$
 $\text{finite } Y \implies$
 $\text{sum } N \ Y = \text{card } X \implies$
 $\exists \mathcal{X}.$
 $X = \bigcup \{ \mathcal{X} \ i \mid i. i \in Y \} \wedge$
 $(\forall i \in Y. \text{card } (\mathcal{X} \ i) = N \ i) \wedge$
 $(\forall i \ j. i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X} \ i \cap \mathcal{X} \ j = \{\})$

then obtain

$B' :: 'b \text{ set}$ **and**
 $y :: 'b$ **where**
 $\text{ins-}B: B = \text{insert } y \ B'$ **and**
 $\text{card-}B': \text{card } B' = x$ **and**
 $\text{fin-}B': \text{finite } B'$ **and**
 $y\text{-not-in-}B': y \notin B'$
using $\text{card-Suc-eq-finite}$
by $(\text{metis } (\text{no-types}, \text{lifting}))$

hence $N \ y \leq \text{card } A$

using $\text{card-}A \ \text{card-}B \ \text{fin-}B \ \text{le-add1 } n\text{-not-Suc-}n \ \text{sum.insert}$
by metis

then obtain $A' :: 'a \text{ set}$ **where**

$X'\text{-in-}X: A' \subseteq A$ **and**
 $\text{card-}X': \text{card } A' = N \ y$
using $\text{fin-}A \ \text{ex-card}$
by metis

hence $\text{finite } (A - A') \wedge \text{card } (A - A') = \text{sum } N \ B'$

using $\text{card-}B \ \text{card-}A \ \text{fin-}A \ \text{fin-}B \ \text{ins-}B \ \text{card-}B' \ \text{fin-}B'$
 $\text{Suc-}n\text{-not-}n \ \text{add-diff-cancel-left}' \ \text{card-Diff-subset} \ \text{card-insert-if}$
 $\text{finite-Diff} \ \text{finite-subset} \ \text{sum.insert}$
by metis

then obtain $\mathcal{X} :: 'b \Rightarrow 'a \text{ set}$ **where**

$\text{part: } A - A' = \bigcup \{ \mathcal{X} \ i \mid i. i \in B' \}$ **and**
 $\text{disj: } \forall i \ j. i \neq j \longrightarrow i \in B' \wedge j \in B' \longrightarrow \mathcal{X} \ i \cap \mathcal{X} \ j = \{\}$ **and**
 $\text{card: } \forall i \in B'. \text{card } (\mathcal{X} \ i) = N \ i$
using $\text{hyp}[of \ B' \ A - A'] \ \text{fin-}B' \ \text{card-}B'$
by auto

then obtain $\mathcal{X}' :: 'b \Rightarrow 'a \text{ set}$ **where**

$\text{map': } \mathcal{X}' = (\lambda z. \text{if } (z = y) \text{ then } A' \text{ else } \mathcal{X} \ z)$
by simp

hence $\text{eq-}\mathcal{X}: \forall i \in B'. \mathcal{X}' \ i = \mathcal{X} \ i$

using $y\text{-not-in-}B'$
by simp

have $B = \{y\} \cup B'$

using $\text{ins-}B$
by simp

hence $\forall f. \{f \ i \mid i. i \in B\} = \{f \ y\} \cup \{f \ i \mid i. i \in B'\}$

by blast

hence $\{\mathcal{X}' \ i \mid i. i \in B\} = \{\mathcal{X}' \ y\} \cup \{\mathcal{X}' \ i \mid i. i \in B'\}$

by metis

hence $\bigcup \{ \mathcal{X}' \ i \mid i. i \in B \} = \mathcal{X}' \ y \cup \bigcup \{ \mathcal{X}' \ i \mid i. i \in B' \}$


```

    by simp
  also have  $\mathcal{X}' y = A'$ 
    using map'
    by presburger
  also have  $\bigcup \{\mathcal{X}' i \mid i. i \in B'\} = \bigcup \{\mathcal{X} i \mid i. i \in B'\}$ 
    using eq- $\mathcal{X}$ 
    by blast
  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  $\forall i \in B. \text{card } (\mathcal{X}' i) = N i$ 
    using map' card card- $X'$  ins-B
    by simp
  ultimately show
     $\exists \mathcal{X}. A = \bigcup \{\mathcal{X} i \mid i. i \in B\} \wedge$ 
       $(\forall i \in B. \text{card } (\mathcal{X} i) = N i) \wedge$ 
       $(\forall i j. i \neq j \longrightarrow i \in B \wedge j \in B \longrightarrow \mathcal{X} i \cap \mathcal{X} j = \{\})$ 
    using part'
    by blast
qed

```

3.2.2 Anonymity Quotient: Grid

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

— 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 $\text{vote-count}_{\mathcal{Q}} :: 'a \text{ Preference-Relation} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow \text{nat}$ **where**
 $\text{vote-count}_{\mathcal{Q}} p = \pi_{\mathcal{Q}} (\text{vote-count } p)$

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

lemma $\text{anon-rel-equiv}: \text{equiv } (\text{elections-}\mathcal{A} \text{ UNIV}) (\text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$

proof —

have $\text{subset}: \text{elections-}\mathcal{A} \text{ UNIV} \subseteq \text{well-formed-elections}$

```

by simp
have equiv: equiv well-formed-elections (anonymityR well-formed-elections)
using rel-ind-by-group-act-equiv[of
  anonymityG well-formed-elections  $\varphi$ -anon well-formed-elections]
  rel-ind-by-coinciding-action-on-subset-eq-restr
by (simp add: anonymous-group-action.group-action-axioms)
have closed:
  closed-restricted-rel
  (anonymityR well-formed-elections) well-formed-elections (elections- $\mathcal{A}$  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)  $\in$  elections- $\mathcal{A}$  UNIV
hence wf-elections: (A, V, p)  $\in$  well-formed-elections
  unfolding elections- $\mathcal{A}$ .simps
  by blast
assume ((A, V, p), A', V', p')  $\in$  anonymityR well-formed-elections
then obtain  $\pi :: 'd \Rightarrow 'd$  where
  bij- $\pi$ : bij  $\pi$  and
  img: (A', V', p') = rename  $\pi$  (A, V, p)
  unfolding anonymityR.simps action-induced-rel.simps
    anonymityG-def  $\varphi$ -anon.simps rewrite-carrier
    extensional-continuation.simps
  by auto
hence (A', V', p')  $\in$  well-formed-elections
  using wf-elections rename-sound
  unfolding well-formed-elections-def
  by fastforce
moreover have A' = A  $\wedge$  finite V
  using bij- $\pi$  elt img rename.simps wf-elections well-formed-elections-def
  by auto
moreover have  $\forall v. v \notin V' \longrightarrow (\text{the-inv } \pi v) \notin V$ 
  using elt Pair-inject UNIV-I  $\langle$  bij  $\pi \rangle$  rename.simps
    f-the-inv-into-f-bij-betw image-eqI img
  unfolding elections- $\mathcal{A}$ .simps
  by (metis (mono-tags, opaque-lifting))
moreover have  $\forall v. v \notin V' \longrightarrow p' v = p (\text{the-inv } \pi v)$ 
  using img
  by simp
ultimately show (A', V', p')  $\in$  elections- $\mathcal{A}$  UNIV
  using elt img
  unfolding elections- $\mathcal{A}$ .simps rename.simps
  by auto
qed
have
  anonymityR (elections- $\mathcal{A}$  UNIV) =
  restricted-rel (anonymityR well-formed-elections) (elections- $\mathcal{A}$  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') ∈ anonymityR (elections- $\mathcal{A}$  UNIV)
  hence (A, V, p) ∈ well-formed-elections
    unfolding anonymityR.simps action-induced-rel.simps elections- $\mathcal{A}$ .simps
    by blast
  moreover obtain  $\pi$  :: 'd  $\Rightarrow$  'd where
    bij  $\pi$  and
    (A', V', p') = rename  $\pi$  (A, V, p)
  using rel
    unfolding anonymityR.simps action-induced-rel.simps
      anonymityG-def  $\varphi$ -anon.simps rewrite-carrier
      extensional-continuation.simps
    by auto
  ultimately show ((A, V, p), A', V', p') ∈ anonymityR well-formed-elections
    unfolding anonymityR.simps action-induced-rel.simps
      anonymityG-def  $\varphi$ -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') ∈ anonymityR (elections- $\mathcal{A}$  UNIV)
  thus (A, V, p) ∈ elections- $\mathcal{A}$  UNIV
    unfolding anonymityR.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') ∈ anonymityR (elections- $\mathcal{A}$  UNIV)
  hence ((A, V, p), A', V', p') ∈ anonymityR well-formed-elections
    unfolding anonymityR.simps action-induced-rel.simps
    by fastforce
  moreover have elt: (A, V, p) ∈ elections- $\mathcal{A}$  UNIV
    using rel
    unfolding anonymityR.simps action-induced-rel.simps
    by blast
  ultimately have
    ((A, V, p), A', V', p') ∈ restricted-rel
    (anonymityR well-formed-elections) (elections- $\mathcal{A}$  UNIV) well-formed-elections
    using equiv

```

```

    unfolding restricted-rel.simps equiv-def refl-on-def
    by blast
  hence  $(A', V', p') \in \text{elections-}\mathcal{A} \text{ UNIV}$ 
    using closed elt
    unfolding closed-restricted-rel.simps
    by blast
  thus  $(A', V', p') \in \text{well-formed-elections}$ 
    using subset
    by blast
next
fix
  A A' :: 'c set and
  V V' :: 'd set and
  p p' :: ('c, 'd) Profile
assume
   $(A, V, p) \in \text{elections-}\mathcal{A} \text{ UNIV}$  and
   $((A, V, p), A', V', p') \in \text{anonymity}_{\mathcal{R}} \text{ well-formed-elections}$ 
moreover from this
obtain  $\pi :: 'd \Rightarrow 'd$  where
  bij  $\pi$  and
   $(A', V', p') = \text{rename } \pi (A, V, p)$ 
  unfolding anonymity $_{\mathcal{R}}$ .simps action-induced-rel.simps
    anonymity $_{\mathcal{G}}$ -def  $\varphi$ -anon.simps rewrite-carrier
    extensional-continuation.simps
  by auto
ultimately show  $((A, V, p), A', V', p') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$ 
  unfolding anonymity $_{\mathcal{R}}$ .simps action-induced-rel.simps
    anonymity $_{\mathcal{G}}$ -def  $\varphi$ -anon.simps rewrite-carrier
    extensional-continuation.simps
  by auto
qed
also have ... = Restr (anonymity $_{\mathcal{R}}$  well-formed-elections) (elections- $\mathcal{A}$  UNIV)
  using restr-equals-restricted-rel closed subset
  by blast
finally have
  anonymity $_{\mathcal{R}}$  (elections- $\mathcal{A}$  UNIV) =
    Restr (anonymity $_{\mathcal{R}}$  well-formed-elections) (elections- $\mathcal{A}$  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 corre-

sponds 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
 \Rightarrow $\text{nat}^{\wedge}('a \text{ Ordered-Preference})$) (*anonymity_Q* (*UNIV* :: 'a set))
 (*UNIV* :: ($\text{nat}^{\wedge}('a \text{ Ordered-Preference})$) set)

proof (*unfold bij-betw-def inj-on-def, intro conjI ballI impI*)

fix *X Y* :: ('a, 'v) Election set

assume

class-X: *X* ∈ *anonymity_Q* *UNIV* **and**

class-Y: *Y* ∈ *anonymity_Q* *UNIV* **and**

eq-vec: *anonymity-class* *X* = *anonymity-class* *Y*

have $\forall E \in \text{elections-}\mathcal{A} \text{ UNIV. finite (voters-}\mathcal{E} \ E)$

by *simp*

hence $\forall (E, E') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV). finite (voters-}\mathcal{E} \ E)$

by *simp*

moreover **have** *subset*: *elections-}\mathcal{A} \text{ UNIV} \subseteq \text{well-formed-elections}*

by *simp*

ultimately **have**

$\forall (E, E') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV).}$

$\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E'$

using *anon-rel-vote-count*

by *blast*

hence *vote-count-invar*:

$\forall p. (\text{vote-count } p) \text{ respects } (\text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$

unfolding *congruent-def*

by *blast*

have *quotient-count*:

$\forall X \in \text{anonymity}_{\mathcal{Q}} \text{ UNIV. } \forall p. \forall E \in X. \text{vote-count}_{\mathcal{Q}} \ p \ X = \text{vote-count } p \ E$

using *pass-to-quotient[of anonymity_R (elections-}\mathcal{A} \text{ UNIV)]*

vote-count-invar anon-rel-equiv

unfolding *anonymity_Q.simps anonymity_R.simps vote-count_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_Q.simps*

by *metis*

ultimately **have**

$\forall p. \text{vote-count}_{\mathcal{Q}} \ p \ X = \text{vote-count } p \ E \wedge \text{vote-count}_{\mathcal{Q}} \ p \ Y = \text{vote-count } p \ E'$

using *class-X class-Y*

by *blast*

moreover **with** *eq-vec* **have**

$\forall p. \text{vote-count}_{\mathcal{Q}} (\text{ord2pref } p) \ X = \text{vote-count}_{\mathcal{Q}} (\text{ord2pref } p) \ Y$

unfolding *anonymity-class.simps*

using *UNIV-I vec-lambda-inverse*
 by *metis*
 ultimately have $\forall p. \text{vote-count } (\text{ord2pref } p) \ E = \text{vote-count } (\text{ord2pref } p) \ E'$
 by *simp*
 hence eq: $\forall p \in \{p. \text{linear-order-on } (\text{UNIV} :: 'a \text{ set}) \ p\}.$
 $\text{vote-count } p \ E = \text{vote-count } p \ E'$
 using *pref2ord-inverse*
 by *metis*
 from *anon-rel-equiv class-X class-Y* have *subset-fixed-alts*:
 $X \subseteq \text{elections-}\mathcal{A} \ \text{UNIV} \wedge Y \subseteq \text{elections-}\mathcal{A} \ \text{UNIV}$
 unfolding *anonymity_Q.simps*
 using *in-quotient-imp-subset*
 by *blast*
 hence eq-alts: $\text{alternatives-}\mathcal{E} \ E = \text{UNIV} \wedge \text{alternatives-}\mathcal{E} \ E' = \text{UNIV}$
 using *E-in-X E'-in-Y*
 unfolding *elections-}\mathcal{A}.sims*
 by *blast*
 with *subset-fixed-alts* have *eq-complement*:
 $\forall p \in \text{UNIV} - \{p. \text{linear-order-on } (\text{UNIV} :: 'a \text{ set}) \ p\}.$
 $\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}$
 $\wedge \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\} = \{\}$
 using *E-in-X E'-in-Y*
 unfolding *elections-}\mathcal{A}.sims* *well-formed-elections-def* *profile-def*
 by *auto*
 hence $\forall p \in \text{UNIV} - \{p. \text{linear-order-on } (\text{UNIV} :: 'a \text{ set}) \ p\}.$
 $\text{vote-count } p \ E = 0 \wedge \text{vote-count } p \ E' = 0$
 unfolding *card-eq-0-iff vote-count.sims*
 by *simp*
 with eq have *eq-vote-count*: $\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E'$
 using *DiffI UNIV-I*
 by *metis*
 moreover from *subset-fixed-alts E-in-X E'-in-Y*
 have $\text{finite } (\text{voters-}\mathcal{E} \ E) \wedge \text{finite } (\text{voters-}\mathcal{E} \ E')$
 unfolding *elections-}\mathcal{A}.sims*
 by *blast*
 moreover from *subset-fixed-alts E-in-X E'-in-Y*
 have $(E, E') \in (\text{elections-}\mathcal{A} \ \text{UNIV}) \times (\text{elections-}\mathcal{A} \ \text{UNIV})$
 by *blast*
 moreover from *this*
 have $(\forall v. v \notin \text{voters-}\mathcal{E} \ E \longrightarrow \text{profile-}\mathcal{E} \ E \ v = \{\})$
 $\wedge (\forall v. v \notin \text{voters-}\mathcal{E} \ E' \longrightarrow \text{profile-}\mathcal{E} \ E' \ v = \{\})$
 by *simp*
 ultimately have $(E, E') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \ \text{UNIV})$
 using *eq-alts vote-count-anon-rel*
 by *metis*
 hence $\text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \ \text{UNIV}) \text{ “ } \{E\} =$
 $\text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \ \text{UNIV}) \text{ “ } \{E'\}$
 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 $\mathcal{Q}$ .simps
by (metis (no-types, lifting))
finally show  $X = Y$ 
by simp
next
have (UNIV :: (nat, 'a Ordered-Preference) vec set)  $\subseteq$ 
  (anonymity-class :: ('a, 'v) Election set  $\Rightarrow$  (nat, 'a Ordered-Preference) vec) ‘
  anonymity $\mathcal{Q}$  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  $\in$  UNIV}
using finite-Atleast-Atmost-nat
by blast
hence sum ( $\lambda$  i. x $\$$ i) UNIV  $< \infty$ 
using enat-ord-code
by simp
moreover have  $0 \leq \text{sum } (\lambda i. x\$i) \text{ UNIV}$ 
by blast
ultimately obtain V :: 'v set where
  fin-V: finite V and
  card V = sum ( $\lambda$  i. x $\$$ i) UNIV
using assms infinite-arbitrarily-large
by metis
then obtain X' :: 'a Ordered-Preference  $\Rightarrow$  'v set where
  card':  $\forall i. \text{card } (X' i) = x\$i$  and
  partition':  $V = \bigcup \{X' i \mid i. i \in \text{UNIV}\}$  and
  disjoint':  $\forall i j. i \neq j \longrightarrow X' i \cap X' j = \{\}$ 
using obtain-partition[of V UNIV ($) x]
by auto
obtain X :: 'a Preference-Relation  $\Rightarrow$  'v set where
  def-X:  $X = (\lambda i. \text{if } (i \in \{i. \text{linear-order } i\})$ 
     $\text{then } X' (\text{pref2ord } i) \text{ else } \{\})$ 
by simp
hence {X i | i. i  $\notin$  {i. linear-order i}}  $\subseteq \{\{\}\}$ 
by auto
moreover have
  {X i | i. i  $\in$  {i. linear-order i}} =
  {X' (pref2ord i) | i. i  $\in$  {i. linear-order i}}
using def-X
by metis
moreover have

```

$\{X\ i \mid i. i \in UNIV\} =$
 $\{X\ i \mid i. i \in \{i. \text{linear-order } i\}\}$
 $\cup \{X\ i \mid i. i \in UNIV - \{i. \text{linear-order } i\}\}$
 by *blast*
ultimately have
 $\{X\ i \mid i. i \in UNIV\} = \{X' (\text{pref2ord } i) \mid i. i \in \{i. \text{linear-order } i\}\}$
 $\vee \{X\ i \mid i. i \in UNIV\} =$
 $\{X' (\text{pref2ord } i) \mid i. i \in \{i. \text{linear-order } i\}\} \cup \{\{\}\}$
 by *auto*
also have
 $\{X' (\text{pref2ord } i) \mid i. i \in \{i. \text{linear-order } i\}\} =$
 $\{X' i \mid i. i \in UNIV\}$
 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 $p' :: 'v \Rightarrow 'a \text{ Preference-Relation}$ **where**
 $p\text{-}X: \forall v \in V. v \in X\ (p'\ v)$ **and**
 $p\text{-}disj: \forall v \in V. \forall i. i \neq p'\ v \longrightarrow v \notin X\ i$
 by *metis*
then obtain $p :: 'v \Rightarrow 'a \text{ Preference-Relation}$ **where**
 $p\text{-}in\text{-}V\text{-then-}p': p = (\lambda v. \text{if } v \in V \text{ then } p'\ v \text{ else } \{\})$
 by *simp*
hence $lin\text{-}ord: \forall v \in V. \text{linear-order } (p\ v)$
 using *def-X p-X p-disj*
 by *fastforce*
hence $wf\text{-elections}: (UNIV, V, p) \in \text{elections-}\mathcal{A}\ UNIV$
 using *fin-V*
unfolding $p\text{-}in\text{-}V\text{-then-}p' \text{ elections-}\mathcal{A}.simps$
 $\text{well-formed-elections-def profile-def}$
 by *auto*
hence $\forall i. \forall E \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A}\ UNIV) \text{ “ } \{(UNIV, V, p)\}.$
 $\text{vote-count } i\ E = \text{vote-count } i\ (UNIV, V, p)$
 using *fin-V anon-rel-vote-count[of (UNIV, V, p) - elections-}\mathcal{A}\ UNIV]*
 by *simp*
moreover have

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

```

ultimately show
  x ∈ (λ X :: ('a, 'v) Election set. χ p. vote-countQ (ord2pref p) X)
    'anonymityQ UNIV
using anonymity-class.elims
by blast
qed
thus (anonymity-class :: ('a, 'v) Election set
      ⇒ (nat, 'a Ordered-Preference) vec) '
      anonymityQ UNIV =
      (UNIV :: (nat, 'a Ordered-Preference) vec set)
by blast
qed

```

3.2.3 Homogeneity Quotient: Simplex

```

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

fun anonymity-homogeneityR :: ('a, 'v) Election set ⇒ ('a, 'v) Election rel where
  anonymity-homogeneityR  $\mathcal{E}$  =
    {(E, E') | E E'. E ∈  $\mathcal{E}$  ∧ E' ∈  $\mathcal{E}$ 
      ∧ (finite (voters- $\mathcal{E}$  E) = finite (voters- $\mathcal{E}$  E'))
      ∧ (∀ r. vote-fraction r E = vote-fraction r E')}

fun anonymity-homogeneityQ :: 'a set ⇒ ('a, 'v) Election set set where
  anonymity-homogeneityQ A =
    quotient (elections- $\mathcal{A}$  A) (anonymity-homogeneityR (elections- $\mathcal{A}$  A))

fun vote-fractionQ :: 'a Preference-Relation ⇒ ('a, 'v) Election set ⇒ rat where
  vote-fractionQ p = πQ (vote-fraction p)

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

fun rat-vector-set :: (real^b) set ⇒ (rat^b) set where
  rat-vector-set V = rat-vector ' {v ∈ V. ∀ i. v$i ∈ Q}

definition standard-basis :: (real^b) set where
  standard-basis ≡ {v. ∃ b. v$b = 1 ∧ (∀ c ≠ b. v$c = 0)}

```

The rational points in the simplex.

definition *vote-simplex* :: (rat^b) set **where**
vote-simplex ≡
 insert 0 (rat-vector-set (convex hull (standard-basis :: (real^b) set)))

Auxiliary Lemmas

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

$x\text{-sum}: x = (\sum x \in X. f\ x *_R\ x)$
 by *blast*
 have $\text{card } X > 0$
 using *card*
 by *linarith*
 hence $\text{fin}: \text{finite } X$
 using *card-gt-0-iff*
 by *blast*
 have $n = 0 \longrightarrow \text{card } X = 1$
 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 fin.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 $n > 0 \longrightarrow x \in \text{convex hull } X$
 proof (*safe*)
 assume $0 < n$
 hence *card-X-gt-one*: $\text{card } X > 1$
 using *card*
 by *simp*
 have $(\forall y \in X. f\ y \geq 1) \longrightarrow \text{sum } f\ X \geq \text{sum } (\lambda x. 1)\ X$
 using *fin sum-mono*
 by *metis*
 moreover have $\text{sum } (\lambda x. 1)\ X = \text{card } X$
 by *force*
 ultimately have $(\forall y \in X. f\ y \geq 1) \longrightarrow \text{card } X \leq \text{sum } f\ X$
 by *force*
 hence $(\forall y \in X. f\ y \geq 1) \longrightarrow 1 < \text{sum } f\ X$
 using *card-X-gt-one*
 by *linarith*
 then obtain $y :: \text{real}^b$ where
 y-in-X: $y \in X$ and
 f-y-lt-one: $f\ y < 1$
 using *sum*
 by *auto*
 hence $1 - f\ y \neq 0 \wedge x = f\ y *_R\ y + (\sum x \in X - \{y\}. f\ x *_R\ x)$
 using *fin sum.remove x-sum*
 by *simp*
 moreover have
 $\forall \alpha \neq 0. (\sum x \in X - \{y\}. f\ x *_R\ x) =$
 $\alpha *_R (\sum x \in X - \{y\}. (f\ x / \alpha) *_R\ x)$
 unfolding *scaleR-sum-right*
 by *simp*

ultimately have *convex-comb*:
 $x = f y *_R y + (1 - f y) *_R (\sum x \in X - \{y\}. (f x / (1 - f y)) *_R x)$
by *simp*
obtain $f' :: \text{real}^b \Rightarrow \text{real}$ **where**
 $\text{def}' : f' = (\lambda x. f x / (1 - f y))$
by *simp*
hence $\forall x \in X - \{y\}. f' x \geq 0$
using *nonneg f-y-lt-one*
by *fastforce*
moreover have
 $\text{sum } f' (X - \{y\}) = (\text{sum } (\lambda x. f x) (X - \{y\})) / (1 - f y)$
unfolding *def' sum-divide-distrib*
by *simp*
moreover have
 $(\text{sum } (\lambda x. f x) (X - \{y\})) / (1 - f y) = (1 - f y) / (1 - f y)$
using *sum y-in-X*
by (*simp add: fin sum.remove*)
moreover have $(1 - f y) / (1 - f y) = 1$
using *f-y-lt-one*
by *simp*
ultimately have
 $\text{sum } f' (X - \{y\}) = 1 \wedge (\forall x \in X - \{y\}. 0 \leq f' x)$
 $\wedge (\sum x \in X - \{y\}. (f x / (1 - f y)) *_R x) =$
 $(\sum x \in X - \{y\}. f' x *_R x)$
using *def'*
by *metis*
hence $\exists f'. \text{sum } f' (X - \{y\}) = 1 \wedge (\forall x \in X - \{y\}. 0 \leq f' x)$
 $\wedge (\sum x \in X - \{y\}. (f x / (1 - f y)) *_R x) =$
 $(\sum x \in X - \{y\}. f' x *_R x)$
by *metis*
moreover have $\text{card } (X - \{y\}) = n$
using *card y-in-X*
by *simp*
ultimately have
 $(\sum x \in X - \{y\}. (f x / (1 - f y)) *_R x) \in \text{convex hull } (X - \{y\})$
using *hyp*
by *blast*
hence $(\sum x \in X - \{y\}. (f x / (1 - f y)) *_R x) \in \text{convex hull } X$
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 $y \in \text{convex hull } X$
using *y-in-X*
by (*simp add: hull-inc*)
moreover have

$\forall x y. x \in \text{convex hull } X \wedge y \in \text{convex hull } X \longrightarrow$
 $(\forall a \geq 0. \forall b \geq 0. a + b = 1 \longrightarrow a *_R x + b *_R y \in \text{convex hull } X)$
using *convex-def convex-convex-hull*
by (*metis (no-types, opaque-lifting)*)
ultimately show $x \in \text{convex hull } X$
using *convex-comb*
by *simp*
qed
ultimately show $x \in \text{convex hull } X$
using *hull-inc*
by *fastforce*
qed

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

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

then obtain $ind :: (real \hat{=} b) \Rightarrow 'b$ **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 $ind-inv :: 'b \Rightarrow (real \hat{=} b)$ **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-neg-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 UNIV ind] in-basis inj-ind*
unfolding *image-subset-iff*
by *simp*
obtain $f :: (real \hat{=} b) \Rightarrow real$ **where**
def: $f = (\lambda v. if v \in standard-basis then x\$ (ind v) else 0)$
by *blast*
hence $sum f standard-basis = sum (\lambda v. x\$ (ind v)) standard-basis$
by *simp*
also have $sum (\lambda v. x\$ (ind v)) standard-basis =$
 $sum ((\$) x \circ ind) standard-basis$
unfolding *comp-def*
by *simp*
also have $\dots = sum ((\$) x) (ind ' standard-basis)$
using *bij-ind-std sum-comp[of ind standard-basis*
 $ind ' standard-basis (\$) x]$
by *simp*

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

```

    sum (λ b'. (χ i. if i = b' then x$b' else 0)$b) UNIV
  using sum-component
  by blast
moreover have
  ∀ b. (λ b'. (χ i. if i = b' then x$b' else 0)$b) =
    (λ b'. if b' = b then x$b else 0)
  by force
moreover have
  ∀ b. sum (λ b'. if b' = b then x$b else 0) UNIV =
    x$b + sum (λ b'. 0) (UNIV - {b})
  by simp
ultimately have
  ∀ b. (sum (λ x. (f x) *R x) standard-basis)$b = x$b
  by simp
hence sum (λ x. (f x) *R x) standard-basis = x
  unfolding vec-eq-iff
  by simp
hence ∃ f :: (real^b) ⇒ real.
  sum f standard-basis = 1 ∧ (∀ x ∈ standard-basis. f x ≥ 0)
  ∧ x = sum (λ x. (f x) *R x) standard-basis
  using sum-eq-one nonneg
  by blast
hence x ∈ convex hull (standard-basis :: (real^b) set)
  using convex-combination-in-convex-hull
  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}
  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- $\mathcal{E}$  E)
  shows equiv X (anonymity-homogeneity $\mathcal{R}$  X)
proof (unfold equiv-def, safe)
  show refl-on X (anonymity-homogeneity $\mathcal{R}$  X)
    unfolding refl-on-def anonymity-homogeneity $\mathcal{R}$ .simps
    by blast
next

```

```

show sym (anonymity-homogeneity $\mathcal{R}$  X)
  unfolding sym-def anonymity-homogeneity $\mathcal{R}$ .simps
  using sup-commute
  by simp
next
show Relation.trans (anonymity-homogeneity $\mathcal{R}$  X)
proof
  fix E E' F :: ('a, 'v) Election
  assume
    rel: (E, E') ∈ anonymity-homogeneity $\mathcal{R}$  X and
    rel': (E', F) ∈ anonymity-homogeneity $\mathcal{R}$  X
  hence fin: finite (voters- $\mathcal{E}$  E')
  unfolding anonymity-homogeneity $\mathcal{R}$ .simps
  using assms
  by fastforce
from rel rel' have eq-fraction:
  ( $\forall$  r. vote-fraction r E = vote-fraction r E')  $\wedge$ 
  ( $\forall$  r. vote-fraction r E' = vote-fraction r F)
  unfolding anonymity-homogeneity $\mathcal{R}$ .simps
  by blast
hence  $\forall$  r. vote-fraction r E = vote-fraction r F
  by metis
thus (E, F) ∈ 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 ( $\lambda$  a. Fract (f a) b) A = Fract (sum f A) b
using assms
proof (induction card A arbitrary: A f b)
case 0
fix
  A :: 'x set and
  f :: 'x  $\Rightarrow$  int and
  b :: int
assume
  0 = card A and
  finite A and
  b  $\neq$  0

```

```

hence  $\text{sum } (\lambda a. \text{Fract } (f a) b) A = 0 \wedge \text{sum } f A = 0$ 
  by simp
thus ?case
  using 0 rat-number-collapse
  by simp
next
case (Suc n)
fix
   $A :: 'x \text{ set}$  and
   $f :: 'x \Rightarrow \text{int}$  and
   $b :: \text{int}$  and
   $n :: \text{nat}$ 
assume
  card-A: Suc n = card A and
  fin-A: finite A and
  b-non-zero: b ≠ 0 and
  hyp:  $\bigwedge A f b.$ 
     $n = \text{card } (A :: 'x \text{ set}) \implies$ 
     $\text{finite } A \implies b \neq 0 \implies (\sum a \in A. \text{Fract } (f a) b) = \text{Fract } (\text{sum } f A) b$ 
hence  $A \neq \{\}$ 
  by auto
then obtain  $c :: 'x$  where
  c-in-A: c ∈ A
  by blast
hence  $(\sum a \in A. \text{Fract } (f a) b) =$ 
   $(\sum a \in A - \{c\}. \text{Fract } (f a) b) + \text{Fract } (f c) b$ 
  using fin-A
  by (simp add: sum-diff1)
also have  $\dots = \text{Fract } (\text{sum } f (A - \{c\})) b + \text{Fract } (f c) b$ 
  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  $\dots = \text{Fract } (\text{sum } f (A - \{c\}) + f c) b$ 
  using c-in-A b-non-zero fract-distr-helper
  by metis
also have  $\dots = \text{Fract } (\text{sum } f A) b$ 
  using c-in-A fin-A
  by (simp add: sum-diff1)
finally show  $(\sum a \in A. \text{Fract } (f a) b) = \text{Fract } (\text{sum } f A) b$ 
  by blast
qed

```

Simplex Bijection

We assume all our elections to consist of a fixed finite alternative set of size n and finite subsets of an infinite voter universe. Profiles are linear orders on the alternatives. Then we can work on the standard simplex of dimension $n!$ instead of the equivalence classes of the equivalence relation for anonymous + homogeneous voting rules (anon hom): Each dimension corresponds to

one possible linear order on the alternative set, i.e., the possible preferences. Each equivalence class of elections corresponds to a vector whose entries denote the fraction of voters per election in that class who vote the respective corresponding preference.

theorem *anonymity-homogeneity_Q-isomorphism:*

assumes *infinite* (*UNIV* :: 'v set)

shows

bij-betw (*anonymity-homogeneity-class* :: ('a :: finite, 'v) Election set \Rightarrow
 $\text{rat}^{\sim}('a \text{ Ordered-Preference}))$ (*anonymity-homogeneity_Q* (*UNIV* :: 'a set))
(vote-simplex :: ($\text{rat}^{\sim}('a \text{ Ordered-Preference}))$ 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_Q* *UNIV* **and**

class-Y: *Y* \in *anonymity-homogeneity_Q* *UNIV* **and**

eq-vec: *anonymity-homogeneity-class X* = *anonymity-homogeneity-class Y*

have *equiv*:

equiv (*elections-A UNIV*) (*anonymity-homogeneity_R* (*elections-A UNIV*))

using *anonymity-homogeneity-is-equivalence CollectD IntD1 inf-commute*

unfolding *elections-A.simps*

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

hence *subset*:

X $\neq \{\}$ \wedge *X* \subseteq *elections-A UNIV* \wedge *Y* $\neq \{\}$ \wedge *Y* \subseteq *elections-A UNIV*

using *class-X class-Y in-quotient-imp-non-empty in-quotient-imp-subset*

unfolding *anonymity-homogeneity_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_R* (*elections-A UNIV*) “ $\{E\} = X$

using *class-X equiv Image-singleton-iff equiv-class-eq quotientE*

unfolding *anonymity-homogeneity_Q.simps*

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

hence $\forall F \in X. (E, F) \in$ *anonymity-homogeneity_R* (*elections-A UNIV*)

unfolding *Image-def*

by *blast*

hence $\forall F \in X. \forall p. \text{vote-fraction } p F = \text{vote-fraction } p E$

unfolding *anonymity-homogeneity_R.simps*

by *fastforce*

hence $\forall p. \text{vote-fraction } p \text{ ` } X = \{\text{vote-fraction } p E\}$

using *E-in-X*

by *blast*

hence $\forall p. \text{vote-fraction}_Q p X = \text{vote-fraction } p E$

using *is-singletonI singleton-set-def-if-card-one the-elem-eq*

unfolding *is-singleton-altdef vote-fraction_Q.simps π_Q .simps singleton-set.simps*

by *metis*

hence *eq-X-E*:

$\forall p. (\text{anonymity-homogeneity-class } X)\$p = \text{vote-fraction } (\text{ord2pref } p) E$

```

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  $\forall F \in Y. (E', F) \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$ 
unfolding Image-def
by blast
hence  $\forall F \in Y. \forall p. \text{vote-fraction } p \ E' = \text{vote-fraction } p \ F$ 
unfolding anonymity-homogeneity $\mathcal{R}$ .simps
by blast
hence  $\forall p. \text{vote-fraction } p \ `Y = \{\text{vote-fraction } p \ E'\}$ 
using E'-in-Y
by fastforce
hence  $\forall p. \text{vote-fraction}_{\mathcal{Q}} p \ Y = \text{vote-fraction } p \ E'$ 
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':
 $\forall p. (\text{anonymity-homogeneity-class } Y) \$ p = \text{vote-fraction } (\text{ord2pref } p) \ E'$ 
unfolding anonymity-homogeneity-class.simps
using vec-lambda-beta
by metis
with eq-X-E eq-vec
have  $\forall p. \text{vote-fraction } (\text{ord2pref } p) \ E = \text{vote-fraction } (\text{ord2pref } p) \ E'$ 
by metis
hence eq-ord:  $\forall p. \text{linear-order } p \longrightarrow \text{vote-fraction } p \ E = \text{vote-fraction } p \ E'$ 
using mem-Collect-eq pref2ord-inverse
by metis
have  $(\forall v. v \in \text{voters-}\mathcal{E} \ E \longrightarrow \text{linear-order } (\text{profile-}\mathcal{E} \ E \ v)) \wedge$ 
 $(\forall v. v \in \text{voters-}\mathcal{E} \ E' \longrightarrow \text{linear-order } (\text{profile-}\mathcal{E} \ E' \ v))$ 
using subset E-in-X E'-in-Y
unfolding elections- $\mathcal{A}$ .simps well-formed-elections-def profile-def
by fastforce
hence  $\forall p. \neg \text{linear-order } p \longrightarrow \text{vote-count } p \ E = 0 \wedge \text{vote-count } p \ E' = 0$ 
unfolding vote-count.simps
using card.infinite card-0-eq Collect-empty-eq
by (metis (mono-tags, lifting))
hence  $\forall p. \neg \text{linear-order } p \longrightarrow \text{vote-fraction } p \ E = 0 \wedge \text{vote-fraction } p \ E' = 0$ 
using int-ops rat-number-collapse
by simp
with eq-ord have  $\forall p. \text{vote-fraction } p \ E = \text{vote-fraction } p \ E'$ 
by metis
hence  $(E, E') \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$ 
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
     $\Rightarrow \text{rat}^{\sim}('a \text{ Ordered-Preference})$ )
    ' anonymity-homogeneityQ UNIV = vote-simplex
  proof (unfold vote-simplex-def, safe)
    fix  $X :: ('a, 'v) \text{ Election set}$ 
    assume
      quot:  $X \in \text{anonymity-homogeneity}_Q \text{ UNIV}$  and
      not-simplex:
        anonymity-homogeneity-class  $X \notin \text{rat-vector-set (convex hull standard-basis)}$ 
    have equiv-rel:
      equiv (elections-A UNIV) (anonymity-homogeneityR (elections-A UNIV))
      using anonymity-homogeneity-is-equivalence[of elections-A UNIV]
        elections-A.simps
      by blast
    then obtain  $E :: ('a, 'v) \text{ Election}$  where
      E-in-X:  $E \in X$  and
       $X = \text{anonymity-homogeneity}_R (\text{elections-A UNIV}) \text{ `` } \{E\}$ 
      using quot anonymity-homogeneity_Q.simps equiv-Eps-in proj-Eps
      unfolding proj-def
      by metis
    hence rel:  $\forall E' \in X. (E, E') \in \text{anonymity-homogeneity}_R (\text{elections-A UNIV})$ 
      by simp
    hence  $\forall p. \forall E' \in X.$ 
       $\text{vote-fraction } (\text{ord2pref } p) E' = \text{vote-fraction } (\text{ord2pref } p) E$ 
      unfolding anonymity-homogeneity_R.simps
      by fastforce
    hence  $\forall p. \text{vote-fraction } (\text{ord2pref } p) \text{ ` } X = \{\text{vote-fraction } (\text{ord2pref } p) E\}$ 
      using E-in-X
      by blast
    hence repr:  $\forall p. \text{vote-fraction}_Q (\text{ord2pref } p) X = \text{vote-fraction } (\text{ord2pref } p) E$ 
      using is-singletonI singleton-set-def-if-card-one the-elem-eq
      unfolding vote-fraction_Q.simps pi_Q.simps is-singleton-altdef
      by metis
    have  $\forall p. \text{vote-count } (\text{ord2pref } p) E \geq 0$ 
      by simp
    hence  $\forall p. \text{card } (\text{voters-}\mathcal{E} E) > 0 \longrightarrow$ 
       $\text{Fract } (\text{int } (\text{vote-count } (\text{ord2pref } p) E)) (\text{int } (\text{card } (\text{voters-}\mathcal{E} E))) \geq 0$ 
      using zero-le-Fract-iff
      by simp
    hence  $\forall p. \text{vote-fraction } (\text{ord2pref } p) E \geq 0$ 
      unfolding vote-fraction.simps card-gt-0-iff
      by simp
    hence  $\forall p. \text{vote-fraction}_Q (\text{ord2pref } p) X \geq 0$ 
      using repr
      by simp

```

hence *geq-zero*: $\forall p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X) \geq 0$
using *zero-le-of-rat-iff*
by *blast*
have *voters- \mathcal{E}* $E = \{\}$ \vee *infinite* (*voters- \mathcal{E}* E) \longrightarrow
 $(\forall p. \text{real-of-rat } (\text{vote-fraction } p E) = 0)$
by *simp*
hence *zero-case*:
voters- \mathcal{E} $E = \{\}$ \vee *infinite* (*voters- \mathcal{E}* E) \longrightarrow
 $(\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X)) = 0$
using *repr*
unfolding *zero-vec-def*
by *simp*
let *?sum* = $\text{sum } (\lambda p. \text{vote-count } p E) \text{ UNIV}$
have *finite* (*UNIV* :: $('a \times 'a) \text{ set}$)
by *simp*
hence *eq-card*: *finite* (*voters- \mathcal{E}* E) \longrightarrow $\text{card } (\text{voters-}\mathcal{E} E) = ?\text{sum}$
using *vote-count-sum*
by *metis*
hence *finite* (*voters- \mathcal{E}* E) \wedge *voters- \mathcal{E}* $E \neq \{\}$ \longrightarrow
 $\text{sum } (\lambda p. \text{vote-fraction } p E) \text{ UNIV} =$
 $\text{sum } (\lambda p. \text{Fract } (\text{vote-count } p E) ?\text{sum}) \text{ UNIV}$
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 $?\text{sum} > 0$
using *eq-card*
by *fastforce*
hence *finite* (*voters- \mathcal{E}* E) \wedge *voters- \mathcal{E}* $E \neq \{\}$ \longrightarrow
 $\text{sum } (\lambda p. \text{Fract } (\text{vote-count } p E) ?\text{sum}) \text{ UNIV} = \text{Fract } ?\text{sum } ?\text{sum}$
using *fract-distr[of UNIV ?sum $\lambda p. \text{int } (\text{vote-count } p E)$]*
card-0-eq eq-card finite-class.finite-UNIV
of-nat-eq-0-iff of-nat-sum sum.cong
by (*metis (no-types, lifting)*)
moreover have
finite (*voters- \mathcal{E}* E) \wedge *voters- \mathcal{E}* $E \neq \{\}$ \longrightarrow $\text{Fract } ?\text{sum } ?\text{sum} = 1$
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}* $E \neq \{\}$
 $\longrightarrow \text{sum } (\lambda p. \text{vote-fraction } p E) \text{ UNIV} = 1$
by *presburger*
have *inv-of-rat*: $\forall x \in \mathbb{Q}. \text{the-inv of-rat } (\text{of-rat } x) = x$
unfolding *Rats-def*
using *the-inv-f-f injI of-rat-eq-iff*
by *metis*
have $E \in \text{elections-}\mathcal{A} \text{ UNIV}$
using *quot E-in-X equiv-class-eq-iff equiv-rel rel*
unfolding *anonymity-homogeneity $_{\mathcal{Q}}$.simps quotient-def*

by *fastforce*
hence $\forall v \in \text{voters-}\mathcal{E} \ E. \text{linear-order } (\text{profile-}\mathcal{E} \ E \ v)$
 unfolding *elections-A.simps well-formed-elections-def profile-def*
 by *fastforce*
hence $\forall p. \neg \text{linear-order } p \longrightarrow \text{vote-count } p \ E = 0$
 unfolding *vote-count.simps*
 using *card.infinite card-0-eq*
 by *blast*
hence $\forall p. \neg \text{linear-order } p \longrightarrow \text{vote-fraction } p \ E = 0$
 using *rat-number-collapse*
 by *simp*
moreover have $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ \text{UNIV} =$
 $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ \{p. \text{linear-order } p\} +$
 $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ (\text{UNIV} - \{p. \text{linear-order } p\})$
 using *finite CollectD Collect-mono UNIV-I add commute*
sum.subset-diff top-set-def
 by *metis*
ultimately have $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ \text{UNIV} =$
 $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ \{p. \text{linear-order } p\}$
 by *simp*
moreover have *bij-betw ord2pref UNIV {p. linear-order p}*
 using *inj-def ord2pref-inject range-ord2pref*
 unfolding *bij-betw-def*
 by *blast*
ultimately have
 $\text{sum } (\lambda p. \text{vote-fraction } p \ E) \ \text{UNIV} =$
 $\text{sum } (\lambda p. \text{vote-fraction } (\text{ord2pref } p) \ E) \ \text{UNIV}$
 using *comp-def[of $\lambda p. \text{vote-fraction } p \ E \ \text{ord2pref}$]*
sum-comp[of ord2pref UNIV {p. linear-order p} $\lambda p. \text{vote-fraction } p \ E$]
 by *auto*
hence *finite (voters- \mathcal{E} E) \wedge voters- \mathcal{E} E $\neq \{\}$*
 $\longrightarrow \text{sum } (\lambda p. \text{vote-fraction } (\text{ord2pref } p) \ E) \ \text{UNIV} = 1$
 using *fin-imp-sum-eq-one*
 by *presburger*
hence *finite (voters- \mathcal{E} E) \wedge voters- \mathcal{E} E $\neq \{\}$*
 $\longrightarrow \text{sum } (\lambda p. \text{real-of-rat } (\text{vote-fraction } (\text{ord2pref } p) \ E)) \ \text{UNIV} = 1$
 using *of-rat-1 of-rat-sum*
 by *metis*
with *zero-case*
have $(\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) \ X)) = 0$
 $\vee \text{sum } (\lambda p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) \ X)) \ \text{UNIV} = 1$
 using *repr*
 by *force*
hence $(\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) \ X)) = 0 \vee$
 $((\forall p. (\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) \ X)) \$ p \geq 0))$
 $\wedge \text{sum } ((\$) (\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) \ X))) \ \text{UNIV} = 1)$
 using *geq-zero*
 by *force*
moreover have *rat-entries:*

$\forall p. (\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X)) \$ p \in \mathcal{Q}$
 by *simp*
ultimately have *simplex-el*:
 $(\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X))$
 $\in \{x \in \text{insert } 0 (\text{convex hull standard-basis}). \forall i. x \$ i \in \mathcal{Q}\}$
 using *standard-simplex-rewrite*
 by *blast*
moreover have
 $\forall p. (\text{rat-vector } (\chi p. \text{of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X))) \$ p =$
 $\text{the-inv real-of-rat } ((\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X)) \$ p)$
 unfolding *rat-vector.simps*
 using *vec-lambda-beta*
 by *blast*
moreover have
 $\forall p. \text{the-inv real-of-rat}$
 $((\chi p. \text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X)) \$ p) =$
 $\text{the-inv real-of-rat } (\text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X))$
 by *simp*
moreover have
 $\forall p. \text{the-inv real-of-rat } (\text{real-of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X)) =$
 $\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X$
 using *rat-entries inv-of-rat Rats-eq-range-nat-to-rat-surj surj-nat-to-rat-surj*
 by *blast*
moreover have
 $\forall p. \text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X = (\text{anonymity-homogeneity-class } X) \$ p$
 by *simp*
ultimately have
 $\forall p. (\text{rat-vector } (\chi p. \text{of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X))) \$ p =$
 $(\text{anonymity-homogeneity-class } X) \$ p$
 by *metis*
hence $\text{rat-vector } (\chi p. \text{of-rat } (\text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p) X))$
 $= \text{anonymity-homogeneity-class } X$
 by *simp*
with *simplex-el*
have $\exists x \in \{x \in \text{insert } 0 (\text{convex hull standard-basis}). \forall i. x \$ i \in \mathcal{Q}\}.$
 $\text{rat-vector } x = \text{anonymity-homogeneity-class } X$
 by *blast*
with *not-simplex*
have $\text{rat-vector } 0 = \text{anonymity-homogeneity-class } X$
 using *image-iff insertE mem-Collect-eq*
 unfolding *rat-vector-set.simps*
 by (*metis (mono-tags, lifting)*)
thus $\text{anonymity-homogeneity-class } X = 0$
 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 *non-empty*:
 $(\text{UNIV}, \{\}, \lambda v. \{\})$

$\in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”})$
unfolding $\text{anonymity-homogeneity}_{\mathcal{R}}.\text{simps}$ $\text{Image-def elections-}\mathcal{A}.\text{simps}$
 $\text{well-formed-elections-def profile-def}$
by simp
have $\text{in-els: } (UNIV, \{\}, \lambda v. \{\}) \in \text{elections-}\mathcal{A} \text{ UNIV}$
unfolding $\text{elections-}\mathcal{A}.\text{simps}$ $\text{well-formed-elections-def profile-def}$
by simp
have $\forall r :: 'a \text{ Preference-Relation.}$
 $\text{vote-fraction } r (UNIV, \{\}, (\lambda v. \{\})) = 0$
by simp
hence
 $\forall E \in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, (\lambda v. \{\}))\}.\forall r. \text{vote-fraction } r E = 0$
unfolding $\text{anonymity-homogeneity}_{\mathcal{R}}.\text{simps}$
by auto
moreover have
 $\forall E \in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, (\lambda v. \{\}))\}.\text{finite (voters-}\mathcal{E} \text{ } E)$
unfolding $\text{Image-def anonymity-homogeneity}_{\mathcal{R}}.\text{simps}$
by fastforce
ultimately have all-zero:
 $\forall r. \forall E \in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, (\lambda v. \{\}))\}.\text{vote-fraction } r E = 0$
by blast
hence $\forall r. 0 \in \text{vote-fraction } r$
 $\text{“}(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, (\lambda v. \{\}))\}$
using $\text{non-empty image-eqI}$
by $(\text{metis (mono-tags, lifting)})$
hence $\forall r. \{0\} \subseteq \text{vote-fraction } r$
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”})$
by blast
moreover have $\forall r. \{0\} \supseteq \text{vote-fraction } r$
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”})$
using all-zero
by blast
ultimately have
 $\forall r. \text{vote-fraction } r$
 $\text{“}(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}) = \{0\}$
by blast
hence
 $\forall r.$
 $\text{card } (\text{vote-fraction } r$
 $\text{“}(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\})) = 1$
 $\wedge \text{the-inv } (\lambda x. \{x\})$
 $(\text{vote-fraction } r$
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$

$\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”} = 0$
using *is-singletonI singleton-insert-inj-eq' singleton-set-def-if-card-one*
unfolding *is-singleton-altdef singleton-set.simps*
by *metis*
hence
 $\forall r. \text{vote-fraction}_{\mathcal{Q}} r$
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”} = 0$
unfolding *vote-fraction_Q.simps $\pi_{\mathcal{Q}}$.simps singleton-set.simps*
by *metis*
hence $\forall r :: 'a \text{ Ordered-Preference}. \text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } r)$
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”} = 0$
by *metis*
hence $\forall r :: 'a \text{ Ordered-Preference}.$
 $(\text{anonymity-homogeneity-class } ((\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”}))\$r = 0$
unfolding *anonymity-homogeneity-class.simps*
using *vec-lambda-beta*
by *(metis (no-types))*
moreover have $\forall r :: 'a \text{ Ordered-Preference}. 0\$r = 0$
by *simp*
ultimately have $\forall r :: 'a \text{ Ordered-Preference}.$
 $(\text{anonymity-homogeneity-class}$
 $((\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”}))\$r = 0\$r$
by *(metis (no-types))*
hence *anonymity-homogeneity-class*
 $((\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}))$
 $\text{“}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”}) = (0 :: (\text{rat } ^{('a \text{ Ordered-Preference}})))$
using *vec-eq-iff*
by *blast*
moreover have
 $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “}\{(UNIV, \{\}, \lambda v. \{\})\}\text{”})$
 $\in \text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV}$
unfolding *anonymity-homogeneity_Q.simps quotient-def*
using *in-els*
by *blast*
ultimately show $0 \in \text{anonymity-homogeneity-class } ^{(\text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV})}$
using *image-eqI*
by *(metis (no-types))*
next
fix $x :: \text{rat } ^{('a \text{ Ordered-Preference})}$
assume $x \in \text{rat-vector-set } (\text{convex hull standard-basis})$
— The following converts a rational vector x to real vector x' .
then obtain $x' :: \text{real } ^{('a \text{ Ordered-Preference})}$ **where**
 $\text{conv: } x' \in \text{convex hull standard-basis}$ **and**
 $\text{inv: } \forall p. x\$p = \text{the-inv real-of-rat } (x'\$p)$ **and**

$rat: \forall p. x' \$ p \in \mathbb{Q}$
unfolding *rat-vector-set.simps rat-vector.simps*
by *force*
hence *convex*: $(\forall p. 0 \leq x' \$ p) \wedge \text{sum } ((\$) x') \text{ UNIV} = 1$
using *standard-simplex-rewrite*
by *blast*
have *map*: $\forall p. \text{real-of-rat } (x \$ p) = x' \$ p$
using *inv rat the-inv-f-f[of real-of-rat] f-the-inv-into-f*
inj-onCI of-rat-eq-iff
unfolding *Rats-def*
by *metis*
have $\forall p. \exists \text{ fract. Fract } (\text{fst fract}) (\text{snd fract}) = x \$ p \wedge 0 < \text{snd fract}$
using *quotient-of-unique*
by *metis*
then obtain *fraction'* :: 'a *Ordered-Preference* $\Rightarrow (int \times int)$ **where**
 $\forall p. x \$ p = \text{Fract } (\text{fst } (\text{fraction}' p)) (\text{snd } (\text{fraction}' p))$ **and**
 $\text{pos}': \forall p. 0 < \text{snd } (\text{fraction}' p)$
by *metis*
with *map*
have *fract'*: $\forall p. x' \$ p = (\text{fst } (\text{fraction}' p)) / (\text{snd } (\text{fraction}' p))$
using *div-by-0 divide-less-cancel of-int-0 of-int-pos of-rat-rat*
by *metis*
with *convex*
have $\forall p. (\text{fst } (\text{fraction}' p)) / (\text{snd } (\text{fraction}' p)) \geq 0$
by *fastforce*
with *pos'*
have $\forall p. \text{fst } (\text{fraction}' p) \geq 0$
using *not-less of-int-0-le-iff of-int-pos zero-le-divide-iff*
by *metis*
with *pos'*
have $\forall p. \text{fst } (\text{fraction}' p) \in \mathbb{N} \wedge \text{snd } (\text{fraction}' p) \in \mathbb{N}$
using *nonneg-int-cases of-nat-in-Nats order-less-le*
by *metis*
hence $\forall p. \exists (n::nat) (m::nat). \text{fst } (\text{fraction}' p) = n \wedge \text{snd } (\text{fraction}' p) = m$
using *Nats-cases*
by *metis*
hence $\forall p. \exists m::nat \times nat. \text{fst } (\text{fraction}' p) = \text{int } (\text{fst } m)$
 $\wedge \text{snd } (\text{fraction}' p) = \text{int } (\text{snd } m)$
by *simp*
then obtain *fraction* :: 'a *Ordered-Preference* $\Rightarrow (nat \times nat)$ **where**
 $\text{eq}: \forall p. \text{fst } (\text{fraction}' p) = \text{int } (\text{fst } (\text{fraction } p)) \wedge$
 $\text{snd } (\text{fraction}' p) = \text{int } (\text{snd } (\text{fraction } p))$
by *metis*
with *fract'*
have *fract*: $\forall p. x' \$ p = (\text{fst } (\text{fraction } p)) / (\text{snd } (\text{fraction } p))$
by *simp*
from *eq pos'*
have *pos*: $\forall p. 0 < \text{snd } (\text{fraction } p)$
by *simp*

```

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

```

by *metis*
then obtain $\text{prof} :: 'v \Rightarrow 'a \text{ Preference-Relation}$ **where**
 $\text{prof}: \text{prof} = (\lambda v. \text{if } v \in V \text{ then } \text{ord2pref } (\text{prof}' v) \text{ else } \{\})$
 by *blast*
hence $\text{election}: (UNIV, V, \text{prof}) \in \text{elections-}\mathcal{A} \text{ UNIV}$
unfolding $\text{elections-}\mathcal{A}.\text{sims}$ $\text{well-formed-elections-def}$ profile-def
using $\text{fin-}V \text{ ord2pref}$
 by *auto*
have $\forall p. \{v \in V. \text{prof}' v = p\} = \{v \in V. v \in \text{part } p\}$
using $\text{maps-to-prof}' \text{ exactly-one-prof}$
 by *blast*
hence $\forall p. \{v \in V. \text{prof}' v = p\} = \text{part } p$
using partition
 by *fastforce*
hence $\forall p. \text{card } \{v \in V. \text{prof}' v = p\} = \text{voter-amount } p$
using card
 by *presburger*
moreover have
 $\forall p. \forall v. (v \in \{v \in V. \text{prof}' v = p\}) = (v \in \{v \in V. \text{prof } v = (\text{ord2pref } p)\})$
using prof
 by $(\text{simp add: ord2pref-inject})$
ultimately have $\forall p. \text{card } \{v \in V. \text{prof } v = (\text{ord2pref } p)\} = \text{voter-amount } p$
 by *simp*
hence $\forall p :: 'a \text{ Ordered-Preference.}$
 $\text{vote-fraction } (\text{ord2pref } p) (UNIV, V, \text{prof}) =$
 $\text{Fract } (\text{voter-amount } p) (\text{card } V)$
using $\text{rat-number-collapse fin-}V$
 by *simp*
moreover have
 $\forall p. \text{Fract } (\text{voter-amount } p) (\text{card } V) = (\text{voter-amount } p) / (\text{card } V)$
unfolding $\text{Fract-of-int-quotient of-rat-divide}$
 by *simp*
moreover have
 $\forall p. (\text{voter-amount } p) / (\text{card } V) =$
 $((\text{fst } (\text{fraction } p)) * (?prod \text{ div } (\text{snd } (\text{fraction } p)))) / ?prod$
using $\text{card def card-}V\text{-eq-sum rewrite-sum}$
 by *presburger*
moreover have
 $\forall p. ((\text{fst } (\text{fraction } p)) * (?prod \text{ div } (\text{snd } (\text{fraction } p)))) / ?prod =$
 $(\text{fst } (\text{fraction } p)) / (\text{snd } (\text{fraction } p))$
using $\text{rewrite-div pos-prod}$
 by *auto*
 — The following are the percentages of voters voting for each linearly ordered profile in $(UNIV, V, \text{prof})$ that equals the entries of the given vector.
ultimately have eq-vec:
 $\forall p :: 'a \text{ Ordered-Preference.}$
 $\text{vote-fraction } (\text{ord2pref } p) (UNIV, V, \text{prof}) = x'\p
using fract
 by *presburger*

moreover have

$\forall E \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\}.$
 $\forall p. \text{vote-fraction } (\text{ord2pref } p) E =$
 $\text{vote-fraction } (\text{ord2pref } p) (UNIV, V, \text{prof})$

unfolding *anonymity-homogeneity_R.simps*

by *fastforce*

ultimately have

$\forall E \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\}.$
 $\forall p. \text{vote-fraction } (\text{ord2pref } p) E = x' \$ p$

by *simp*

hence

$\forall E \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\}.$
 $\forall p. \text{vote-fraction } (\text{ord2pref } p) E = x' \$ p$

using *eq-vec*

by *metis*

hence *vec-entries-match-E-vote-frac*:

$\forall p. \forall E \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$
 $\text{“ } \{(UNIV, V, \text{prof})\}. \text{vote-fraction } (\text{ord2pref } p) E = x' \$ p$

by *blast*

have $\forall x \in \mathbb{Q}. \forall y. \text{complex-of-rat } y = \text{complex-of-real } x \longrightarrow \text{real-of-rat } y = x$

using *Re-complex-of-real Re-divide-of-real of-rat.rep-eq of-real-of-int-eq*

by *metis*

hence $\forall x \in \mathbb{Q}. \forall y. \text{complex-of-rat } y = \text{complex-of-real } x$

$\longrightarrow y = \text{the-inv real-of-rat } x$

using *injI of-rat-eq-iff the-inv-f-f*

by *metis*

with *vec-entries-match-E-vote-frac*

have *all-eq-vec*:

$\forall p. \forall E \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$
 $\text{“ } \{(UNIV, V, \text{prof})\}. \text{vote-fraction } (\text{ord2pref } p) E = x' \$ p$

using *rat inv*

by *metis*

moreover have

$(UNIV, V, \text{prof}) \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$
 $\text{“ } \{(UNIV, V, \text{prof})\}$

using *anonymity-homogeneity_R.simps election*

by *blast*

ultimately have $\forall p. \text{vote-fraction } (\text{ord2pref } p) \text{ “ }$

anonymity-homogeneity_R (elections- \mathcal{A} UNIV) “ $\{(UNIV, V, \text{prof})\} \supseteq \{x' \$ p\}$

using *image-insert insert-iff mk-disjoint-insert singletonD subsetI*

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

with *all-eq-vec*

have $\forall p. \text{vote-fraction } (\text{ord2pref } p) \text{ “ }$

anonymity-homogeneity_R (elections- \mathcal{A} UNIV) “ $\{(UNIV, V, \text{prof})\} = \{x' \$ p\}$

by *blast*

hence $\forall p. \text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p)$

(anonymity-homogeneity_R (elections- \mathcal{A} UNIV) “ $\{(UNIV, V, \text{prof})\}) = x' \$ p$

using *is-singletonI singleton-inject singleton-set-def-if-card-one*

unfolding *is-singleton-altdef vote-fraction_Q.simps π_Q .simps*


```

    by metis
  hence  $x = \text{anonymity-homogeneity-class}$ 
    ( $\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “}\{(UNIV, V, \text{prof})\}\text{”}$ )
  unfolding  $\text{anonymity-homogeneity-class.simps}$ 
  using  $\text{vec-lambda-unique}$ 
  by ( $\text{metis (no-types, lifting)}$ )
  moreover have
    ( $\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})$ 
      “ $\{(UNIV, V, \text{prof})\} \in \text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV}$ ”
    )
  unfolding  $\text{anonymity-homogeneity}_{\mathcal{Q}.simps}$   $\text{quotient-def}$ 
  using  $\text{election}$ 
  by blast
  ultimately show
     $x \in (\text{anonymity-homogeneity-class}$ 
      ::  $(\text{'a}, \text{'v}) \text{ Election set} \Rightarrow \text{rat}^{\sim}(\text{'a} \text{ Ordered-Preference}))$ 
      ‘ $\text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV}$ ’
    )
  by blast
qed
qed
end

```

Chapter 4

Component Types

4.1 Distance

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

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

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

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

4.1.1 Definition

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

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

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

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

4.1.2 Conditions

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

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

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

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

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

4.1.3 Standard-Distance Property

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

4.1.4 Auxiliary Lemmas

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

lemma *arg-min-subset*:
fixes
 $B :: 'b\ \text{set}$ **and**
 $f :: 'b \Rightarrow 'a :: \text{ord}$
shows *arg-min-set* $f\ B \subseteq B$
unfolding *arg-min-set.simps is-arg-min-def*
by *safe*

lemma *sum-monotone*:
fixes
 $A :: 'a\ \text{set}$ **and**
 $f\ g :: 'a \Rightarrow \text{int}$
assumes $\forall\ a \in A. f\ a \leq g\ a$
shows $(\sum\ a \in A. f\ a) \leq (\sum\ a \in A. g\ a)$
using *assms*
proof (*induction A rule: infinite-finite-induct*)
case (*infinite A*)
fix $A :: 'a\ \text{set}$
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[of f]
  by simp

```

```

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  $\Rightarrow$  'a Preference-Relation  $\Rightarrow$ 
    'a Preference-Relation  $\Rightarrow$  ('a  $\times$  'a) set where
    pairwise-disagreements' A r s =
        Set.filter ( $\lambda$  (a, b). a  $\neq$  b  $\wedge$  neq-ord r s a b) (A  $\times$  A)

```

lemma set-eq-filter:

```

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

```

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

fun swap :: 'a Vote Distance **where**

```

    swap (A, r) (A', r') =
        (if A = A'
         then card (pairwise-disagreements A r r')
         else  $\infty$ )

```

lemma swap-case-infinity:

```

fixes x y :: 'a Vote
assumes alts- $\mathcal{V}$  x  $\neq$  alts- $\mathcal{V}$  y
shows swap x y =  $\infty$ 
using assms
by (induction rule: swap.induct, simp)

```

lemma swap-case-fin:

```

fixes x y :: 'a Vote
assumes alts- $\mathcal{V}$  x = alts- $\mathcal{V}$  y
shows swap x y = card (pairwise-disagreements (alts- $\mathcal{V}$  x) (pref- $\mathcal{V}$  x) (pref- $\mathcal{V}$  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  $\sum a \in A. \text{abs} (\text{int} (\text{rank } x \ a) - \text{int} (\text{rank } y \ a))$ 
         else  $\infty$ )

```

lemma spearman-case-inf:

```

fixes x y :: 'a Vote
assumes alts- $\mathcal{V}$  x  $\neq$  alts- $\mathcal{V}$  y
shows spearman x y =  $\infty$ 

```

```

using assms
by (induction rule: spearman.induct, simp)

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

```

4.1.7 Properties

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

Definitions

```

fun total-invarianceD ::  $'x\ Distance \Rightarrow 'x\ rel \Rightarrow bool$  where
  total-invarianceD  $d\ rel = is-symmetry\ (tup\ d)\ (Invariance\ (product\ rel))$ 

fun invarianceD ::  $'y\ Distance \Rightarrow 'x\ set \Rightarrow 'y\ set \Rightarrow$ 
   $( 'x, 'y)\ binary-fun \Rightarrow bool$  where
  invarianceD  $d\ X\ Y\ \varphi = is-symmetry\ (tup\ d)\ (Invariance\ (equivariance\ X\ Y\ \varphi))$ 

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

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

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

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

```

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

Auxiliary Lemmas

lemma *rewrite-total-invariance* _{\mathcal{D}} :

fixes

d :: 'x Distance **and**

r :: 'x rel

shows *total-invariance* _{\mathcal{D}} d r = (\forall (x, y) \in r. \forall (a, b) \in r. d a x = d b y)

proof (*unfold total-invariance* _{\mathcal{D}} .*simps is-symmetry.simps product.simps, safe*)

fix a b x y :: 'x

assume

\forall x y. (x, y) \in {(p, p')}.

(fst p, fst p') \in r \wedge (snd p, snd p') \in r

\longrightarrow tup d x = tup d y **and**

(a, b) \in r **and**

(x, y) \in r

thus d a x = d b y

unfolding *total-invariance* _{\mathcal{D}} .*simps is-symmetry.simps*

by *simp*

next

fix a b x y :: 'x

assume

\forall (x, y) \in r. \forall (a, b) \in r. d a x = d b y **and**

(fst (x, a), fst (y, b)) \in r **and**

(snd (x, a), snd (y, b)) \in r

hence d x a = d y b

by *auto*

thus tup d (x, a) = tup d (y, b)

by *simp*

qed

lemma *rewrite-invariance* _{\mathcal{D}} :

fixes

d :: 'y Distance **and**

X :: 'x set **and**

Y :: 'y set **and**

φ :: ('x, 'y) binary-fun

shows *invariance* _{\mathcal{D}} d X Y φ =

(\forall x \in X. \forall y \in Y. \forall z \in Y. d y z = d (φ x y) (φ x z))

proof (*unfold invariance* _{\mathcal{D}} .*simps is-symmetry.simps equivariance.simps, safe*)

fix

x :: 'x **and**

y z :: 'y

assume

x \in X **and**

y \in Y **and**

```

    z ∈ Y and
    ∀ x y. (x, y) ∈ {(u, v), x, y}. (u, v) ∈ Y × Y
        ∧ (∃ z ∈ X. x = φ z u ∧ y = φ z v)}
        → tup d x = tup d y
  thus d y z = d (φ x y) (φ x z)
    by fastforce
next
fix
  x :: 'x and
  a b :: 'y
assume
  ∀ x ∈ X. ∀ y ∈ Y. ∀ z ∈ Y. d y z = d (φ x y) (φ x z) and
  x ∈ X and
  a ∈ Y and
  b ∈ Y
hence d a b = d (φ x a) (φ x b)
  by blast
thus tup d (a, b) = tup d (φ x a, φ x b)
  by simp
qed

```

lemma *invar-dist-image*:

```

fixes
  d :: 'y Distance and
  G :: 'x monoid and
  Y Y' :: 'y set and
  φ :: ('x, 'y) binary-fun and
  y :: 'y and
  g :: 'x
assumes
  invar-d: invarianceD d (carrier G) Y φ and
  Y'-in-Y: Y' ⊆ Y and
  action-φ: group-action G Y φ and
  g-carrier: g ∈ carrier G and
  y-in-Y: y ∈ Y
shows d (φ g y) ' (φ g) ' Y' = d y ' Y'
proof (safe)
  fix y' :: 'y
  assume y'-in-Y': y' ∈ Y'
  hence ((y, y'), ((φ g y), (φ g y'))) ∈ equivariance (carrier G) Y φ
    using Y'-in-Y y-in-Y g-carrier
  unfolding equivariance.simps
  by blast
hence eq-dist: tup d ((φ g y), (φ g y')) = tup d (y, y')
  using invar-d
  unfolding invarianceD.simps
  by fastforce
thus d (φ g y) (φ g y') ∈ d y ' Y'
  using y'-in-Y'

```



```

    by simp
  have  $\varphi \ g \ y' \in \varphi \ g \ ' \ Y'$ 
    using  $y'\text{-in-}Y'$ 
    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: invarianceD swap (carrier neutralityG)
  UNIV  $(\lambda \ \pi \ (A, q). (\pi \ ' \ A, \text{rel-rename } \pi \ q))$ 
proof (unfold rewrite-invarianceD, safe)
  fix
     $\pi :: 'a \Rightarrow 'a$  and
     $A \ A' :: 'a \text{ set}$  and
     $q \ q' :: 'a \text{ rel}$ 
  assume  $\pi \in \text{carrier neutrality}_G$ 
  hence  $\text{bij-}\pi$ :  $\text{bij } \pi$ 
    unfolding neutralityG-def
    using rewrite-carrier
    by blast
  show swap  $(A, q) \ (A', q') =$ 
    swap  $(\pi \ ' \ A, \text{rel-rename } \pi \ q) \ (\pi \ ' \ A', \text{rel-rename } \pi \ q')$ 
  proof (cases  $A = A'$ )
    let ?f =  $(\lambda \ (a, b). (\pi \ a, \pi \ b))$ 
    let ?swap-set =  $\{(a, b) \in A \times A. a \neq b \wedge \text{neq-ord } q \ q' \ a \ b\}$ 
    let ?swap-set' =
       $\{(a, b) \in \pi \ ' \ A \times \pi \ ' \ A. a \neq b$ 
         $\wedge \text{neq-ord } (\text{rel-rename } \pi \ q) \ (\text{rel-rename } \pi \ q') \ a \ b\}$ 
    let ?rel =  $\{(a, b) \in A \times A. a \neq b \wedge \text{neq-ord } q \ q' \ a \ b\}$ 
    case True
    hence  $\pi \ ' \ A = \pi \ ' \ A'$ 
      by simp
    hence swap  $(\pi \ ' \ A, \text{rel-rename } \pi \ q) \ (\pi \ ' \ A', \text{rel-rename } \pi \ q') = \text{card } ?\text{swap-set}'$ 
      by simp
    moreover have  $\text{bij-betw } ?f \ ?\text{swap-set} \ ?\text{swap-set}'$ 
  proof (unfold bij-betw-def inj-on-def, intro conjI impI ballI)
    fix  $x \ y :: 'a \times 'a$ 
    assume
       $x \in ?\text{swap-set}$  and
       $y \in ?\text{swap-set}$  and
       $?f \ x = ?f \ y$ 
    hence
       $\pi \ (\text{fst } x) = \pi \ (\text{fst } y)$  and
       $\pi \ (\text{snd } x) = \pi \ (\text{snd } y)$ 
    by auto
    hence
       $\text{fst } x = \text{fst } y$  and
       $\text{snd } x = \text{snd } y$ 

```

```

    using bij- $\pi$  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- $\pi$  bij-pointE
    by metis
  moreover have
     $\forall a b. \text{neg-ord } q \ q' \ a \ b$ 
     $\longrightarrow \text{neg-ord } (\text{rel-rename } \pi \ q) \ (\text{rel-rename } \pi \ q') \ (\pi a) \ (\pi b)$ 
    unfolding neg-ord.simps rel-rename.simps
    by auto
  ultimately show ?f ‘ ?swap-set  $\subseteq$  ?swap-set’
    by auto
next
have  $\forall a b. (a, b) \in (\text{rel-rename } \pi \ q) \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi \ b) \in q$ 
  unfolding rel-rename.simps
  using bij- $\pi$  bij-is-inj the-inv-f-f
  by fastforce
moreover have
   $\forall a b. (a, b) \in (\text{rel-rename } \pi \ q') \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi \ b) \in q'$ 
  unfolding rel-rename.simps
  using bij- $\pi$  bij-is-inj the-inv-f-f
  by fastforce
ultimately have
   $\forall a b. \text{neg-ord } (\text{rel-rename } \pi \ q) \ (\text{rel-rename } \pi \ q') \ a \ b$ 
   $\longrightarrow \text{neg-ord } q \ q' \ (\text{the-inv } \pi \ a) \ (\text{the-inv } \pi \ b)$ 
  by simp
moreover have
   $\forall a b. (a, b) \in \pi ‘ A \times \pi ‘ A \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi \ b) \in A \times A$ 
  using bij- $\pi$  bij-is-inj f-the-inv-into-f inj-image-mem-iff
  by fastforce
moreover have  $\forall a b. a \neq b \longrightarrow \text{the-inv } \pi \ a \neq \text{the-inv } \pi \ b$ 
  using bij- $\pi$  UNIV-I bij-betw-imp-surj bij-is-inj f-the-inv-into-f
  by metis
ultimately have
   $\forall a b. (a, b) \in ?\text{swap-set}' \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi \ b) \in ?\text{swap-set}$ 
  by blast
moreover have  $\forall a b. (a, b) = ?f \ (\text{the-inv } \pi \ a, \text{the-inv } \pi \ b)$ 
  using f-the-inv-into-f-bij-betw bij- $\pi$ 
  by fastforce
ultimately show ?swap-set'  $\subseteq$  ?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  $\pi \text{ ` } A \neq \pi \text{ ` } A'$ 
  using bij- $\pi$  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  $\Rightarrow$  Norm  $\Rightarrow$ 
  ('a, 'v :: linorder) Election Distance where
  votewise-distance d n (A, V, p) (A', V', p') =
    (if (finite V)  $\wedge$  V = V'  $\wedge$  (V  $\neq$  {}  $\vee$  A = A')
     then n (map2 ( $\lambda$  q q'. d (A, q) (A', q')) (to-list V p) (to-list V' p'))
     else  $\infty$ )

```

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

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

```

lemma *linorder-rank-injective*:

fixes

$V :: 'v :: \text{linorder set}$ **and**

$v \ v' :: 'v$

assumes

$v\text{-in-}V: v \in V$ **and**

$v'\text{-in-}V: v' \in V$ **and**

$v'\text{-neq-}v: v' \neq v$ **and**

$\text{fin-}V: \text{finite } V$

shows $\text{card } \{x \in V. x < v\} \neq \text{card } \{x \in V. x < v'\}$

proof –

have $v < v' \vee v' < v$

using $v'\text{-neq-}v$ *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 $v\text{-in-}V$ $v'\text{-in-}V$ *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

$l :: 'v \text{ list}$ **and**

$\pi_1 \ \pi_2 :: \text{nat} \Rightarrow \text{nat}$

assumes $\forall \ i < \text{length } l. \ \pi_1 \ i = \pi_2 \ i$

shows $\text{permute-list } \pi_1 \ l = \text{permute-list } \pi_2 \ l$

using *assms*

unfolding *permute-list-def*

by *simp*

lemma *symmetric-norm-imp-distance-anonymous*:

fixes

$d :: 'a \ \text{Vote Distance}$ **and**

$n :: \text{Norm}$

assumes *symmetry n*

shows *distance-anonymity (votewise-distance d n)*

proof (*unfold distance-anonymity-def, safe*)

fix

$A \ A' :: 'a \ \text{set}$ **and**

$V \ V' :: 'v :: \text{linorder set}$ **and**

$p \ p' :: ('a, 'v) \ \text{Profile}$ **and**

$\pi :: 'v \Rightarrow 'v$

let $?rn1 = \text{rename } \pi \ (A, V, p)$ **and**

$?rn2 = \text{rename } \pi \ (A', V', p')$ **and**

$?rn\text{-}V = \pi \text{ ` } V$ **and**

$?rn\text{-}V' = \pi \text{ ` } V'$ **and**

$?rn\text{-}p = p \circ (\text{the-inv } \pi)$ **and**

$?rn-p' = p' \circ (the-inv \ \pi)$ **and**
 $?len = length \ (to-list \ V \ p)$ **and**
 $?sl-V = sorted-list-of-set \ V$
let $?perm = \lambda \ i. (card \ (\{v \in ?rn-V. \ v < \pi \ (?sl-V!i)\}))$ **and**
— Use a total permutation function in order to apply facts such as *mset-permute-list*.
 $?perm-total = (\lambda \ i. (if \ (i < ?len)$
 $\quad then \ card \ (\{v \in ?rn-V. \ v < \pi \ (?sl-V!i)\})$
 $\quad else \ i))$
assume $bij-\pi: bij \ \pi$
show $votewise-distance \ d \ n \ (A, \ V, \ p) \ (A', \ V', \ p') =$
 $\quad votewise-distance \ d \ n \ ?rn1 \ ?rn2$
proof —
have $rn-A-eq-A: fst \ ?rn1 = A$
by *simp*
have $rn-A'-eq-A': fst \ ?rn2 = A'$
by *simp*
have $rn-V-eq-pi-V: fst \ (snd \ ?rn1) = ?rn-V$
by *simp*
have $rn-V'-eq-pi-V': fst \ (snd \ ?rn2) = ?rn-V'$
by *simp*
have $rn-p-eq-pi-p: snd \ (snd \ ?rn1) = ?rn-p$
by *simp*
have $rn-p'-eq-pi-p': snd \ (snd \ ?rn2) = ?rn-p'$
by *simp*
show $?thesis$
proof (*cases finite* $V \wedge V = V' \wedge (V \neq \{\} \vee A = A')$)
case *False*
— Case: Both distances are infinite.
hence $inf-dist: votewise-distance \ d \ n \ (A, \ V, \ p) \ (A', \ V', \ p') = \infty$
by *auto*
moreover **have** $infinite \ V \longrightarrow infinite \ ?rn-V$
using *False bij- π bij-betw-finite bij-betw-subset False subset-UNIV*
by *metis*
moreover **have** $V \neq V' \longrightarrow ?rn-V \neq ?rn-V'$
using *bij- π bij-def inj-image-mem-iff subsetI subset-antisym*
by *metis*
moreover **have** $V = \{\} \longrightarrow ?rn-V = \{\}$
using *bij- π*
by *simp*
ultimately **have** $inf-dist-rewrite: votewise-distance \ d \ n \ ?rn1 \ ?rn2 = \infty$
using *False*
by *auto*
thus $votewise-distance \ d \ n \ (A, \ V, \ p) \ (A', \ V', \ p') =$
 $\quad votewise-distance \ d \ n \ ?rn1 \ ?rn2$
using *inf-dist*
by *simp*
next
case *True*
— Case: Both distances are finite.

```

have perm-funs-coincide:  $\forall i < ?len. ?perm\ i = ?perm-total\ i$ 
  by presburger
have lengths-eq:  $?len = length\ (to-list\ V'\ p')$ 
  using True
  by simp
have rn-V-permutes:  $(to-list\ V\ p) = permute-list\ ?perm\ (to-list\ ?rn-V\ ?rn-p)$ 
  using assms to-list-permutes-under-bij bij- $\pi$  to-list-permutes-under-bij
  unfolding comp-def
  by (metis (no-types))
hence len-V-rn-V-eq:  $?len = length\ (to-list\ ?rn-V\ ?rn-p)$ 
  by simp
hence permute-list ?perm  $(to-list\ ?rn-V\ ?rn-p) =$ 
  permute-list ?perm-total  $(to-list\ ?rn-V\ ?rn-p)$ 
  using permute-invariant-under-coinciding-funs[of  $(to-list\ ?rn-V\ ?rn-p)$ ]
  perm-funs-coincide
  by presburger
hence rn-list-perm-list-V:
   $(to-list\ V\ p) = permute-list\ ?perm-total\ (to-list\ ?rn-V\ ?rn-p)$ 
  using rn-V-permutes
  by metis
have rn-V'-permutes:
   $(to-list\ V'\ p') = permute-list\ ?perm\ (to-list\ ?rn-V'\ ?rn-p')$ 
  unfolding comp-def
  using True bij- $\pi$  to-list-permutes-under-bij
  by (metis (no-types))
hence permute-list ?perm  $(to-list\ ?rn-V'\ ?rn-p')$ 
  = permute-list ?perm-total  $(to-list\ ?rn-V'\ ?rn-p')$ 
  using permute-invariant-under-coinciding-funs[of  $(to-list\ ?rn-V'\ ?rn-p')$ ]
  perm-funs-coincide lengths-eq
  by fastforce
hence rn-list-perm-list-V':
   $(to-list\ V'\ p') = permute-list\ ?perm-total\ (to-list\ ?rn-V'\ ?rn-p')$ 
  using rn-V'-permutes
  by metis
have rn-lengths-eq:  $length\ (to-list\ ?rn-V\ ?rn-p) = length\ (to-list\ ?rn-V'\ ?rn-p')$ 
  using len-V-rn-V-eq lengths-eq rn-V'-permutes
  by simp
have perm:  $?perm-total\ permutes\ \{0 ..< ?len\}$ 
proof -
  have  $\forall i\ j. (i < ?len \wedge j < ?len \wedge i \neq j$ 
     $\longrightarrow \pi\ ((sorted-list-of-set\ V)!i) \neq \pi\ ((sorted-list-of-set\ V)!j))$ 
    using bij- $\pi$  bij-pointE True nth-eq-iff-index-eq length-map
      sorted-list-of-set.distinct-sorted-key-list-of-set to-list.elims
    by (metis (mono-tags, opaque-lifting))
  moreover have in-bnds-imp-img-el:
     $\forall i. i < ?len \longrightarrow \pi\ ((sorted-list-of-set\ V)!i) \in \pi\ 'V$ 
    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
 $\forall i < ?len. \forall j < ?len. (?perm-total\ i = ?perm-total\ j \longrightarrow i = j)$
using *linorder-rank-injective Collect-cong True finite-imageI*
by (*metis (no-types, lifting)*)
moreover have $\forall i. i < ?len \longrightarrow i \in \{0 ..< ?len\}$
by *simp*
ultimately have $\forall i \in \{0 ..< ?len\}. \forall j \in \{0 ..< ?len\}. (?perm-total\ i = ?perm-total\ j \longrightarrow i = j)$
by *simp*
hence inj: *inj-on ?perm-total $\{0 ..< ?len\}$*
unfolding *inj-on-def*
by *simp*
have $\forall v' \in (\pi \text{ ' } V). (card\ (\{v \in (\pi \text{ ' } V). v < v'\})) < card\ (\pi \text{ ' } V)$
using *card-seteq True finite-imageI less-irrefl*
linorder-not-le mem-Collect-eq subsetI
by (*metis (no-types, lifting)*)
moreover have $\forall i < ?len. \pi\ ((sorted-list-of-set\ V)!i) \in \pi \text{ ' } V$
using *in-bnds-imp-img-el*
by *simp*
moreover have $card\ (\pi \text{ ' } V) = card\ V$
using *bij- π bij-betw-same-card bij-betw-subset top-greatest*
by *metis*
moreover have $card\ V = ?len$
by *simp*
ultimately have *bounded-img:*
 $\forall i. (i < ?len \longrightarrow ?perm-total\ i \in \{0 ..< ?len\})$
using *atLeast0LessThan lessThan-iff*
by (*metis (full-types)*)
hence $\forall i. i < ?len \longrightarrow ?perm-total\ i \in \{0 ..< ?len\}$
by *simp*
moreover have $\forall i. i \in \{0 ..< ?len\} \longrightarrow i < ?len$
using *atLeastLessThan-iff*
by *blast*
ultimately have $\forall i. i \in \{0 ..< ?len\} \longrightarrow ?perm-total\ i \in \{0 .. ?len\}$
by *fastforce*
hence $?perm-total \text{ ' } \{0 ..< ?len\} \subseteq \{0 ..< ?len\}$
using *bounded-img*
by *force*
hence $?perm-total \text{ ' } \{0 ..< ?len\} = \{0 ..< ?len\}$
using *inj card-image card-subset-eq finite-atLeastLessThan*
by *blast*
hence *bij-perm:* *bij-betw ?perm-total $\{0 ..< ?len\}$ $\{0 ..< ?len\}$*
using *inj bij-betw-def atLeast0LessThan*
by *blast*
thus *?thesis*
using *atLeast0LessThan bij-imp-permutes*
by *fastforce*
qed
have *votewise-distance d n ?rn1 ?rn2 =*


```

      n (map2 (λ q q'. d (A, q) (A', q'))
        (to-list ?rn-V ?rn-p) (to-list ?rn-V' ?rn-p'))
using True rn-A-eq-A rn-A'-eq-A' rn-V-eq-pi-V
      rn-V'-eq-pi-V' rn-p-eq-pi-p rn-p'-eq-pi-p'
by force
also have ... = n (map2 (λ q q'. d (A, q) (A', q'))
      (permute-list ?perm-total (to-list ?rn-V ?rn-p))
      (permute-list ?perm-total (to-list ?rn-V' ?rn-p')))
using symmetric-norm-inv-under-map-permute[of
      ?perm-total to-list ?rn-V ?rn-p]
      assms perm rn-lengths-eq len-V-rn-V-eq
by simp
also have ... = n (map2 (λ q q'. d (A, q) (A', q'))
      (to-list V p) (to-list V' p'))
using rn-list-perm-list-V rn-list-perm-list-V'
by presburger
also have votewise-distance d n (A, V, p) (A', V', p') =
      n (map2 (λ q q'. d (A, q) (A', q')) (to-list V p) (to-list V' p'))
using True
by force
finally show
      votewise-distance d n (A, V, p) (A', V', p') =
      votewise-distance d n ?rn1 ?rn2
by linarith
qed
qed
qed

```

lemma *neutral-dist-imp-neutral-votewise-dist:*

```

fixes
  d :: 'a Vote Distance and
  n :: Norm
defines vote-action ≡ (λ π (A, q). (π ' A, rel-rename π q))
assumes invar: invarianceD d (carrier neutralityG) UNIV vote-action
shows distance-neutrality well-formed-elections (votewise-distance d n)
proof (unfold distance-neutrality.simps rewrite-invarianceD, 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 neutralityG and
  valid: (A, V, p) ∈ well-formed-elections and
  valid': (A', V', p') ∈ well-formed-elections
hence bij-π: bij π
unfolding neutralityG-def
using rewrite-carrier
by blast

```

thus $\text{votewise-distance } d \ n \ (A, V, p) \ (A', V', p') =$
 $\text{votewise-distance } d \ n$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A, V, p))$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A', V', p'))$
proof $(\text{cases finite } V \wedge V = V' \wedge (V \neq \{\} \vee A = A'))$
case *True*
hence $\text{finite } V \wedge V = V' \wedge (V \neq \{\} \vee \pi \ ' A = \pi \ ' A')$
by *metis*
hence $\text{votewise-distance } d \ n$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A, V, p))$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A', V', p')) =$
 $n \ (\text{map2 } (\lambda \ q \ q'. \ d \ (\pi \ ' A, q) \ (\pi \ ' A', q'))$
 $(\text{to-list } V \ (\text{rel-rename } \pi \circ p)) \ (\text{to-list } V' \ (\text{rel-rename } \pi \circ p')))$
using *valid valid'*
by *auto*
also have
 $(\text{map2 } (\lambda \ q \ q'. \ d \ (\pi \ ' A, q) \ (\pi \ ' A', q'))$
 $(\text{to-list } V \ (\text{rel-rename } \pi \circ p)) \ (\text{to-list } V' \ (\text{rel-rename } \pi \circ p')) =$
 $(\text{map2 } (\lambda \ q \ q'. \ d \ (\pi \ ' A, q) \ (\pi \ ' A', q'))$
 $(\text{map } (\text{rel-rename } \pi) \ (\text{to-list } V \ p)) \ (\text{map } (\text{rel-rename } \pi) \ (\text{to-list } V' \ p')))$
using *to-list-comp*
by *metis*
also have
 $(\text{map2 } (\lambda \ q \ q'. \ d \ (\pi \ ' A, q) \ (\pi \ ' A', q'))$
 $(\text{map } (\text{rel-rename } \pi) \ (\text{to-list } V \ p))$
 $(\text{map } (\text{rel-rename } \pi) \ (\text{to-list } V' \ p')) =$
 $(\text{map2 } (\lambda \ q \ q'. \ d \ (\pi \ ' A, \text{rel-rename } \pi \ q) \ (\pi \ ' A', \text{rel-rename } \pi \ q'))$
 $(\text{to-list } V \ p) \ (\text{to-list } V' \ p'))$
using *map-helper*
by *blast*
also have
 $(\lambda \ q \ q'. \ d \ (\pi \ ' A, \text{rel-rename } \pi \ q) \ (\pi \ ' A', \text{rel-rename } \pi \ q')) =$
 $(\lambda \ q \ q'. \ d \ (A, q) \ (A', q'))$
using $\text{rewrite-invariance}_{\mathcal{D}}[\text{of } d \text{ carrier neutrality}_{\mathcal{G}} \ \text{UNIV vote-action}]$
 $\text{invar carrier UNIV-I case-prod-conv}$
unfolding *vote-action-def*
by *(metis (no-types, lifting))*
finally have $\text{votewise-distance } d \ n$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A, V, p))$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A', V', p')) =$
 $n \ (\text{map2 } (\lambda \ q \ q'. \ d \ (A, q) \ (A', q')) \ (\text{to-list } V \ p) \ (\text{to-list } V' \ p'))$
by *simp*
also have $\text{votewise-distance } d \ n \ (A, V, p) \ (A', V', p') =$
 $n \ (\text{map2 } (\lambda \ q \ q'. \ d \ (A, q) \ (A', q')) \ (\text{to-list } V \ p) \ (\text{to-list } V' \ p'))$
using *True*
by *auto*
finally show *?thesis*
by *simp*
next

```

case False
hence  $\neg (\text{finite } V \wedge V = V' \wedge (V \neq \{\} \vee \pi \text{ ' } A = \pi \text{ ' } A'))$ 
  using bij- $\pi$  bij-is-inj inj-image-eq-iff
  by metis
hence votewise-distance d n
  ( $\varphi$ -neutral well-formed-elections  $\pi (A, V, p)$ )
    ( $\varphi$ -neutral well-formed-elections  $\pi (A', V', p')$ ) =  $\infty$ 
  using valid valid'
  by auto
also have votewise-distance d n ( $A, V, p$ ) ( $A', V', p'$ ) =  $\infty$ 
  using False
  by auto
finally show ?thesis
  by simp
qed
qed
end

```

4.3 Consensus

```

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

```

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

4.3.1 Definition

```

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

```

4.3.2 Consensus Conditions

Nonempty alternative set.

```

fun nonempty-setC :: ('a, 'v) Consensus where
  nonempty-setC (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-profileC :: ('a, 'v) Consensus where
  nonempty-profileC (A, V, p) = (V  $\neq$   $\{\}$ )

```

Equal top ranked alternatives.

fun *equal-top_C'* :: ('a, 'v) *Consensus* **where**
equal-top_C' a (A, V, p) = (a ∈ A ∧ (∀ v ∈ V. above (p v) a = {a}))

fun *equal-top_C* :: ('a, 'v) *Consensus* **where**
equal-top_C c = (∃ a. *equal-top_C'* a c)

Equal votes.

fun *equal-vote_C'* :: ('a, 'v) *Consensus* **where**
equal-vote_C' r (A, V, p) = (∀ v ∈ V. (p v) = r)

fun *equal-vote_C* :: ('a, 'v) *Consensus* **where**
equal-vote_C c = (∃ r. *equal-vote_C'* r c)

Unanimity condition.

fun *unanimity_C* :: ('a, 'v) *Consensus* **where**
unanimity_C c = (*nonempty-set_C* c ∧ *nonempty-profile_C* c ∧ *equal-top_C* c)

Strong unanimity condition.

fun *strong-unanimity_C* :: ('a, 'v) *Consensus* **where**
strong-unanimity_C c = (*nonempty-set_C* c ∧ *nonempty-profile_C* c ∧ *equal-vote_C* c)

4.3.3 Properties

definition *consensus-anonymity* :: ('a, 'v) *Consensus* ⇒ bool **where**
consensus-anonymity c ≡
 (∀ A V p π :: ('v ⇒ 'v).
 bij π ⟶
 (let (A', V', q) = (*rename* π (A, V, p)) in
 profile V A p ⟶ *profile* V' A' q
 ⟶ c (A, V, p) ⟶ c (A', V', q)))

fun *consensus-neutrality* :: ('a, 'v) *Election set* ⇒ ('a, 'v) *Consensus* ⇒ bool **where**
consensus-neutrality X c = *is-symmetry* c (*Invariance* (*neutrality_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* (λ e. c e ∧ 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**

π :: 'v ⇒ 'v

assume

bij- π : *bij* π **and**
renamed: *rename* π $(A, V, p) = (A', V', q)$ **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**
 π $:: 'v \Rightarrow 'v$
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- $\pi$  renamed prof-p
  by fastforce
thus ?thesis
  using anon cond-c renamed rename-finite bij- $\pi$  prof-p
  unfolding consensus-anonymity-def Let-def
  by auto
qed

lemma ex-anon-cons-imp-cons-anonymous:
  fixes
     $b :: ('a, 'v) \text{ Consensus}$  and
     $b' :: 'b \Rightarrow ('a, 'v) \text{ Consensus}$ 
  assumes
    general-cond-b:  $b = (\lambda E. \exists x. b' x E)$  and
    all-cond-anon:  $\forall x. \text{consensus-anonymity } (b' x)$ 
  shows consensus-anonymity  $b$ 
proof (unfold consensus-anonymity-def Let-def, safe)
  fix
     $A A' :: 'a \text{ set}$  and
     $V V' :: 'v \text{ set}$  and
     $p q :: ('a, 'v) \text{ Profile}$  and
     $\pi :: 'v \Rightarrow 'v$ 
  assume
    bij- $\pi$ : bij  $\pi$  and
    cond-b:  $b (A, V, p)$  and
    prof-p: profile  $V A p$  and
    renamed:  $\text{rename } \pi (A, V, p) = (A', V', q)$ 
  have  $\exists x. b' x (A, V, p)$ 
    using cond-b general-cond-b
    by simp
  then obtain  $x :: 'b$  where
     $b' x (A, V, p)$ 
    by blast
  moreover have consensus-anonymity  $(b' x)$ 
    using all-cond-anon
    by simp
  moreover have profile  $V' A' q$ 
    using prof-p renamed bij- $\pi$  rename-sound
    by fastforce
  ultimately have  $b' x (A', V', q)$ 
    using all-cond-anon bij- $\pi$  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 nonempty-set_C*
unfolding *consensus-anonymity-def*
by *simp*

lemma *nonempty-profile-cons-anonymous: consensus-anonymity nonempty-profile_C*

proof (*unfold consensus-anonymity-def Let-def, clarify*)

fix

$A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ q :: ('a, 'v) \text{ Profile}$ **and**
 $\pi :: 'v \Rightarrow 'v$

assume

bij- π : *bij π* **and**
renamed: *rename π (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 *nonempty-profile_C (A, V, p)*

ultimately show *nonempty-profile_C (A', V', q)*

using *length-0-conv renamed*
unfolding *nonempty-profile_C.simps*
by *auto*

qed

lemma *equal-top-cons'-anonymous:*

fixes $a :: 'a$
shows *consensus-anonymity (equal-top_C' a)*

proof (*unfold consensus-anonymity-def Let-def, clarify*)

fix

$A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ q :: ('a, 'v) \text{ Profile}$ **and**
 $\pi :: 'v \Rightarrow 'v$

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_C' a (A, V, p)*

have $\forall v' \in V'. q \ v' = p \ ((the_inv \ \pi) \ v')$

using *renamed*

by *auto*

moreover have $\forall v' \in V'. (the_inv \ \pi) \ v' \in V$

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:  $\forall v \in V. \text{above } (p \ v) \ a = \{a\}$ 
    using top-cons-a
    by simp
  ultimately have  $\forall v' \in V'. \text{above } (q \ v') \ a = \{a\}$ 
    by simp
  moreover have  $a \in A$ 
    using top-cons-a
    by simp
  ultimately show  $\text{equal-top}_C' \ a \ (A', V', q)$ 
    using renamed
    unfolding  $\text{equal-top}_C'.\text{sims}$ 
    by simp
qed

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

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

```


by *metis*
qed

lemma *eq-vote-cons-anonymous: consensus-anonymity equal-vote_C*
unfolding *equal-vote_C.sims*
using *eq-vote-cons'-anonymous ex-anon-cons-imp-cons-anonymous*
by *blast*

Neutrality

lemma *nonempty-set_C-neutral: consensus-neutrality well-formed-elections nonempty-set_C*
unfolding *well-formed-elections-def*
by *auto*

lemma *nonempty-profile_C-neutral: consensus-neutrality well-formed-elections nonempty-profile_C*
unfolding *well-formed-elections-def*
by *auto*

lemma *equal-vote_C-neutral: consensus-neutrality well-formed-elections equal-vote_C*
proof (*unfold well-formed-elections-def consensus-neutrality.sims is-symmetry.sims,*
intro allI impI,
unfold split-paired-all neutrality_R.sims action-induced-rel.sims
voters- \mathcal{E} .sims alternatives- \mathcal{E} .sims profile- \mathcal{E} .sims φ -neutral.sims
extensional-continuation.sims equal-vote_C.sims equal-vote_C'.sims
alternatives-rename.sims case-prod-unfold mem-Collect-eq fst-conv
snd-conv mem-Sigma-iff conj-assoc If-def simp-thms, safe)

fix
 $A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$ **and**
 $\pi :: 'a \Rightarrow 'a$ **and**
 $r :: 'a \text{ rel}$

assume
 $\text{profile } V \ A \ p$ **and**
 $(THE \ z. \ (\text{profile } V \ A \ p \longrightarrow z = (\pi \text{ ` } A, V, \text{rel-rename } \pi \circ p)))$
 $\wedge (\neg \text{profile } V \ A \ p \longrightarrow z = \text{undefined}) = (A', V', p')$

hence
equal-voters: $V' = V$ **and**
perm-profile: $p' = (\lambda x. \{(\pi \ a, \pi \ b) \mid a \ b. (a, b) \in p \ x\})$
unfolding *comp-def*
by (*simp, simp*)

have
 $(\forall v \in V. p \ v = r)$
 $\longrightarrow (\exists r'. \forall v \in V. \{(\pi \ a, \pi \ b) \mid a \ b. (a, b) \in p \ v\} = r')$
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  $\pi \in \text{carrier neutrality}_G$ 
  hence  $\text{bij } \pi$ 
    using rewrite-carrier
    unfolding neutralityG-def
    by blast
  hence  $\forall a. \text{the-inv } \pi (\pi a) = a$ 
    using bij-is-inj the-inv-f-f
    by metis
  moreover have
     $(\forall v \in V. \{(\pi a, \pi b) \mid a b. (a, b) \in p v\} = r) \longrightarrow$ 
     $(\forall v \in V. \{(\text{the-inv } \pi (\pi a), \text{the-inv } \pi (\pi b)) \mid a b. (a, b) \in p v\} =$ 
     $\{(\text{the-inv } \pi a, \text{the-inv } \pi b) \mid a b. (a, b) \in r\})$ 
    by fastforce
  ultimately have
     $(\forall v \in V. \{(\pi a, \pi b) \mid a b. (a, b) \in p v\} = r) \longrightarrow$ 
     $(\forall v \in V. \{(a, b) \mid a b. (a, b) \in p v\} =$ 
     $\{(\text{the-inv } \pi a, \text{the-inv } \pi b) \mid a b. (a, b) \in r\})$ 
    by auto
  hence
     $(\forall v' \in V. \{(\pi a, \pi b) \mid a b. (a, b) \in p v'\} = r)$ 
     $\longrightarrow (\exists r'. \forall v' \in V. p v' = r')$ 
    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-unanimityC-neutral: consensus-neutrality
  well-formed-elections strong-unanimityC
  using nonempty-setC-neutral equal-voteC-neutral nonempty-profileC-neutral
  cons-conjunction-invariant[of
    {nonempty-setC, nonempty-profileC, equal-voteC}
    neutralityR well-formed-elections]
  unfolding strong-unanimityC.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 funE :: ('v set  $\Rightarrow$  'a set  $\Rightarrow$  ('a, 'v) Profile  $\Rightarrow$  'r)  $\Rightarrow$ 
  (('a, 'v) Election  $\Rightarrow$  'r) where
  funE m = ( $\lambda$  E. m (voters- $\mathcal{E}$  E) (alternatives- $\mathcal{E}$  E) (profile- $\mathcal{E}$  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 where
  elect m V A p  $\equiv$  elect-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 where
  reject m V A p  $\equiv$  reject-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 where
  defer m V A p  $\equiv$  defer-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 $m = (\forall A V p. \text{profile } V A p \longrightarrow \text{well-formed } A (m V A p))$

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 $m :: ('a, 'v, ('r Result)) \text{ Electoral-Module}$
assumes $\forall A V p. \text{profile } V A p \longrightarrow \text{well-formed } A (m V A p)$
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 $m \wedge$
 $(\forall A V p \pi :: ('v \Rightarrow 'v). \text{bij } \pi \longrightarrow (\text{let } (A', V', q) = (\text{rename } \pi (A, V, p)) \text{ in } \text{profile } V A p \wedge \text{profile } V' A' q \longrightarrow m V A p = m V' A' q))$

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

fun *anonymity'* :: ('a, 'v) Election set \Rightarrow ('a, 'v, 'r) Electoral-Module \Rightarrow bool **where**
anonymity' $X m = \text{is-symmetry } (\text{fun}_{\mathcal{E}} m) (\text{Invariance } (\text{anonymity}_{\mathcal{R}} X))$

Homogeneity

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

fun (in result) *homogeneity* :: ('a, 'v) Election set \Rightarrow
 ('a, 'v, ('r Result)) Electoral-Module \Rightarrow bool **where**
homogeneity X m = is-symmetry (fun_E m) (Invariance (homogeneity_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 :: linorder) Election set \Rightarrow
 ('a, 'v, ('r Result)) Electoral-Module \Rightarrow bool **where**
homogeneity' X m = is-symmetry (fun_E m) (Invariance (homogeneity_R' X))

lemma (in result) *hom-imp-anon*:

fixes

X :: ('a, 'v) Election set **and**

m :: ('a, 'v, ('r Result)) Electoral-Module

assumes

homogeneity X m **and**

$\forall E \in X. \text{finite } (\text{voters-}\mathcal{E} \ E)$

shows *anonymity'* X m

proof (unfold *anonymity'.simps* is-symmetry.simps, intro allI impI)

fix E E' :: ('a, 'v) Election

assume rel: (E, E') \in *anonymity_R* X

then obtain $\pi :: 'v \Rightarrow 'v$ **where**

$\pi \in \text{carrier } \text{anonymity}_G$ **and**

$E' = \varphi\text{-anon } X \ \pi \ E$

unfolding *anonymity_R.simps* action-induced-rel.simps

by blast

moreover from this have bij π

unfolding *anonymity_G-def* rewrite-carrier

by simp

moreover from this have in-election-set: $E \in X$

using rel

unfolding *anonymity_R.simps* action-induced-rel.simps

by blast

ultimately have finite (voters- \mathcal{E} E')

using *assms* rename.simps rename-finite split-pairs

unfolding $\varphi\text{-anon.simps}$ 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_R.simps* action-induced-rel.simps

by blast

moreover have $\forall r. \text{vote-count } r \ E = 1 * (\text{vote-count } r \ E')$

using fin-E anon-rel-vote-count rel mult-1

```

    by metis
  moreover have alternatives- $\mathcal{E}$   $E = \text{alternatives-}\mathcal{E} \ E'$ 
    using fin- $E$  anon-rel-vote-count rel
  by metis
  ultimately show  $\text{fun}_{\mathcal{E}} \ m \ E = \text{fun}_{\mathcal{E}} \ m \ E'$ 
    using assms in-election-set
  unfolding homogeneity.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 :: ('a, 'v) Election set  $\Rightarrow$ 
  ('a, 'v, 'b Result) Electoral-Module  $\Rightarrow$  bool where
  neutrality  $X \ m =$ 
    is-symmetry ( $\text{fun}_{\mathcal{E}} \ m$ ) (action-induced-equivariance (carrier neutrality $_{\mathcal{G}}$ )  $X$ 
      ( $\varphi$ -neutral  $X$ ) (result-action  $\psi$ -neutral))

```

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

```

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  $n \ m \equiv$ 
    SCF-result.electoral-module  $m \wedge$ 
    ( $\forall \ A \ V \ p. (\text{card } A \geq n \wedge \text{finite } A \wedge \text{profile } V \ A \ p) \longrightarrow \text{card } (\text{defer } m \ V \ A \ p) = n$ )

```

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

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

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

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

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

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

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

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

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

4.4.6 Equivalence Definitions

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

definition $\text{prof-leq-result} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow 'a \Rightarrow \text{bool}$ **where**

$\text{prof-leq-result } m \ V \ A \ p \ q \ a \equiv$
 $\text{SCF-result.electoral-module } m \wedge$
 $\text{profile } V \ A \ p \wedge \text{profile } V \ A \ q \wedge a \in A \wedge$
 $(a \in \text{reject } m \ V \ A \ p \longrightarrow a \in \text{reject } m \ V \ A \ q) \wedge$
 $(a \in \text{defer } m \ V \ A \ p \longrightarrow a \notin \text{elect } m \ V \ A \ q)$

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

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

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

4.4.7 Auxiliary Lemmas

lemma *elect-rej-def-combination:*

fixes

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$V :: 'v \text{ set}$ **and**

$A :: 'a \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$ **and**

$e \ r \ d :: 'a \text{ set}$

assumes

$\text{elect } m \ V \ A \ p = e$ **and**

$\text{reject } m \ V \ A \ p = r$ **and**

$\text{defer } m \ V \ A \ p = d$


```

shows  $m \ V \ A \ p = (e, r, d)$ 
using assms
by auto

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

lemma result-presv-alts:
fixes
   $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
assumes
  SCF-result.electoral-module  $m$  and
  profile  $V \ A \ p$ 
shows  $(\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p) = A$ 
proof (safe)
fix  $a :: 'a$ 
have
  partition-imp-exist:
 $\forall p'. \text{set-equals-partition } A \ p' \longrightarrow (\exists E \ R \ D. p' = (E, R, D) \wedge E \cup R \cup D = A)$  and
  partition-A:
 $\text{set-equals-partition } A \ (m \ V \ A \ p)$ 
using assms
by (simp, simp)
{
  assume  $a \in \text{elect } m \ V \ A \ p$ 
  with partition-imp-exist partition-A
  show  $a \in A$ 
  using UnI1 fstI
  by (metis (no-types))
}
{
  assume  $a \in \text{reject } m \ V \ A \ p$ 
  with partition-imp-exist partition-A
  show  $a \in A$ 
  using UnI1 fstI sndI subsetD sup-ge2
  by metis
}

```

```

}
{
  assume  $a \in \text{defer } m \ V \ A \ p$ 
  with partition-imp-exist partition-A
  show  $a \in A$ 
    using sndI subsetD sup-ge2
    by metis
}
{
  assume
     $a \in A$  and
     $a \notin \text{defer } m \ V \ A \ p$  and
     $a \notin \text{reject } m \ V \ A \ p$ 
  with partition-imp-exist partition-A
  show  $a \in \text{elect } m \ V \ A \ p$ 
    using fst-conv snd-conv Un-iff
    by metis
}
qed

lemma result-disj:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $A :: 'a \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  and
     $V :: 'v \text{ set}$ 
  assumes
    SCF-result.electoral-module  $m$  and
    profile  $V \ A \ p$ 
  shows
     $(\text{elect } m \ V \ A \ p) \cap (\text{reject } m \ V \ A \ p) = \{\}$   $\wedge$ 
     $(\text{elect } m \ V \ A \ p) \cap (\text{defer } m \ V \ A \ p) = \{\}$   $\wedge$ 
     $(\text{reject } m \ V \ A \ p) \cap (\text{defer } m \ V \ A \ p) = \{\}$ 
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
       $a \in \text{elect } m \ V \ A \ p$  and
       $a \in \text{reject } m \ V \ A \ p$ 
    with wf disj
    show  $a \in \{\}$ 
      using prod.exhaust-sel DiffE UnCI result-imp-rej

```

```

    by (metis (no-types))
  }
  {
    assume
      elect-a:  $a \in \text{elect } m \ V \ A \ p$  and
      defer-a:  $a \in \text{defer } m \ V \ A \ p$ 
    then obtain
      e :: ' $a$  Result  $\Rightarrow$  ' $a$  set and
      r :: ' $a$  Result  $\Rightarrow$  ' $a$  set and
      d :: ' $a$  Result  $\Rightarrow$  ' $a$  set
    where
      m V A p =
        (e (m V A p), r (m V A p), d (m V A p))  $\wedge$ 
        e (m V A p)  $\cap$  r (m V A p) = {}  $\wedge$ 
        e (m V A p)  $\cap$  d (m V A p) = {}  $\wedge$ 
        r (m V A p)  $\cap$  d (m V A p) = {}
    using IntI emptyE prod.collapse disj disjoint3.simps
    by metis
  hence ((elect m V A p)  $\cap$  (reject m V A p) = {})  $\wedge$ 
        ((elect m V A p)  $\cap$  (defer m V A p) = {})  $\wedge$ 
        ((reject m V A p)  $\cap$  (defer m V A p) = {})
    using eq-snd-iff fstI
    by metis
  thus  $a \in \{\}$ 
    using elect-a defer-a disjoint-iff-not-equal
    by (metis (no-types))
  }
  {
    assume
      a  $\in$  reject m V A p and
      a  $\in$  defer m V A p
    with wf disj
    show  $a \in \{\}$ 
      using prod.exhaust-sel DiffE UnCI result-imp-rej
      by (metis (no-types))
  }
}
qed

```

lemma *elect-in-alts*:

fixes

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ and

$A :: 'a \text{ set}$ and

$p :: ('a, 'v) \text{ Profile}$

assumes

$SCF\text{-result.electoral-module } m$ and

$\text{profile } V \ A \ p$

shows $\text{elect } m \ V \ A \ p \subseteq A$

using *le-supI1 assms result-presv-alts sup-ge1*

by *metis*

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

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

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

```

    profile V A p and
    finite A
  shows
    upper-card-bound-for-elect:  $\text{card } (\text{elect } m \ V \ A \ p) \leq \text{card } A$  and
    upper-card-bound-for-reject:  $\text{card } (\text{reject } m \ V \ A \ p) \leq \text{card } A$  and
    upper-card-bound-for-defer:  $\text{card } (\text{defer } m \ V \ A \ p) \leq \text{card } A$ 
  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
    SCF-result.electoral-module m and
    profile V A p
  shows  $\text{reject } m \ V \ A \ p = A - (\text{elect } m \ V \ A \ p) - (\text{defer } m \ V \ A \ p)$ 
proof -
  from assms have  $(\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p) = A$ 
  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
    SCF-result.electoral-module m and
    profile V A p
  shows  $\text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p = A - (\text{reject } m \ V \ A \ p)$ 
proof -
  from assms have  $(\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p) = A$ 
  using result-presv-alts
  by blast
  with assms show ?thesis
  using result-disj
  by blast
qed

```

lemma *defer-not-elec-or-rej*:

fixes

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$A :: 'a \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$

assumes

$SCF\text{-result.electoral-module } m$ **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

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$ **and**

$a :: 'a$

assumes

$SCF\text{-result.electoral-module } m$ **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

$m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$ **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

$SCF\text{-result.electoral-module } n$ **and**

$SCF\text{-result.electoral-module } m$ **and**

$profile \ V \ A \ p$ **and**

```

    a ∈ A
    using assms
    unfolding mod-contains-result-def
    by safe
next
show
  a ∈ elect n V A p ⇒ a ∈ elect m V A p and
  a ∈ reject n V A p ⇒ a ∈ reject m V A p and
  a ∈ defer n V A p ⇒ a ∈ 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
    SCF-result.electoral-module m and
    profile V A p and
    a ∈ A and
    a ∉ reject m V A p
  shows a ∈ elect m V A p ∨ a ∈ 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 ⊂ 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

```

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$
 $A :: 'a \text{ set and}$
 $V :: 'v \text{ set and}$
 $p \ q :: ('a, 'v) \text{ Profile}$
assumes
 $eq: \forall a \in A. \text{ prof-contains-result } m \ V \ A \ p \ q \ a \text{ and}$
 $mod\text{-}m: \text{ SCF-result.electoral-module } m \text{ and}$
 $prof\text{-}p: \text{ profile } V \ A \ p \text{ and}$
 $prof\text{-}q: \text{ profile } V \ A \ q$
shows $m \ V \ A \ p = m \ V \ A \ q$
proof –
have
 $elect\text{-in-}A: \text{ elect } m \ V \ A \ q \subseteq A \text{ and}$
 $reject\text{-in-}A: \text{ reject } m \ V \ A \ q \subseteq A \text{ and}$
 $defer\text{-in-}A: \text{ defer } m \ V \ A \ q \subseteq A$
using $mod\text{-}m \ prof\text{-}q$
by $(metis \text{ elect-in-alts, reject-in-alts, defer-in-alts})$
have
 $\forall a \in \text{ elect } m \ V \ A \ p. a \in \text{ elect } m \ V \ A \ q \text{ and}$
 $\forall a \in \text{ reject } m \ V \ A \ p. a \in \text{ reject } m \ V \ A \ q \text{ and}$
 $\forall a \in \text{ defer } m \ V \ A \ p. a \in \text{ defer } m \ V \ A \ q$
using $eq \ mod\text{-}m \ prof\text{-}p \text{ in-mono}$
unfolding $prof\text{-contains-result-def}$
by $(metis \text{ (no-types, lifting) elect-in-alts,}$
 reject-in-alts,
 $\text{ defer-in-alts})$
moreover have
 $\forall a \in \text{ elect } m \ V \ A \ q. a \in \text{ elect } m \ V \ A \ p \text{ and}$
 $\forall a \in \text{ reject } m \ V \ A \ q. a \in \text{ reject } m \ V \ A \ p \text{ and}$
 $\forall a \in \text{ defer } m \ V \ A \ q. a \in \text{ defer } m \ V \ A \ p$
proof $(safe)$
fix $a :: 'a$
assume $q\text{-elect-}a: a \in \text{ elect } m \ V \ A \ q$
hence $a \in A$
using $elect\text{-in-}A$
by $blast$
moreover have
 $a \notin \text{ defer } m \ V \ A \ q \text{ and}$
 $a \notin \text{ reject } m \ V \ A \ q$
using $q\text{-elect-}a \ prof\text{-}q \ mod\text{-}m \ result\text{-disj} \ disjoint\text{-iff-not-equal}$
by $(metis, metis)$
ultimately show $a \in \text{ elect } m \ V \ A \ p$
using $eq \ electoral\text{-mod-defer-elem}$
unfolding $prof\text{-contains-result-def}$
by $metis$
next
fix $a :: 'a$
assume $q\text{-rejects-}a: a \in \text{ reject } m \ V \ A \ q$
hence $a \in A$


```

    using rejected-in-A
    by blast
  moreover have
     $a \notin \text{defer } m \ V \ A \ q$  and
     $a \notin \text{elect } m \ V \ A \ q$ 
    using  $q\text{-rejects-}a \ \text{prof-}q \ \text{mod-}m \ \text{result-disj} \ \text{disjoint-iff-not-equal}$ 
    by (metis, metis)
  ultimately show  $a \in \text{reject } m \ V \ A \ p$ 
    using  $eq \ \text{electoral-mod-defer-elem}$ 
    unfolding  $\text{prof-contains-result-def}$ 
    by metis
next
fix  $a :: 'a$ 
assume  $q\text{-defers-}a: a \in \text{defer } m \ V \ A \ q$ 
moreover have  $a \in A$ 
  using  $q\text{-defers-}a \ \text{deferred-in-}A$ 
  by blast
moreover have
   $a \notin \text{elect } m \ V \ A \ q$  and
   $a \notin \text{reject } m \ V \ A \ q$ 
  using  $q\text{-defers-}a \ \text{prof-}q \ \text{mod-}m \ \text{result-disj} \ \text{disjoint-iff-not-equal}$ 
  by (metis, metis)
ultimately show  $a \in \text{defer } m \ V \ A \ p$ 
  using  $eq \ \text{electoral-mod-defer-elem}$ 
  unfolding  $\text{prof-contains-result-def}$ 
  by metis
qed
ultimately show ?thesis
  using  $\text{prod.collapse} \ \text{subsetI} \ \text{subset-antisym}$ 
  by (metis (no-types))
qed

lemma  $eq\text{-def-and-elect-imp-eq}$ :
  fixes
     $m \ n :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$  and
     $A :: 'a \ \text{set}$  and
     $V :: 'v \ \text{set}$  and
     $p \ q :: ('a, 'v) \ \text{Profile}$ 
  assumes
     $\text{mod-}m: \text{SCF-result.electoral-module } m$  and
     $\text{mod-}n: \text{SCF-result.electoral-module } n$  and
     $\text{fin-}p: \text{profile } V \ A \ p$  and
     $\text{fin-}q: \text{profile } V \ A \ q$  and
     $\text{elec-eq}: \text{elect } m \ V \ A \ p = \text{elect } n \ V \ A \ q$  and
     $\text{def-eq}: \text{defer } m \ V \ A \ p = \text{defer } n \ V \ A \ q$ 
  shows  $m \ V \ A \ p = n \ V \ A \ q$ 
proof -
  have
     $\text{reject } m \ V \ A \ p = A - ((\text{elect } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p))$  and

```

```

    reject n V A q = A - ((elect n V A q)  $\cup$  (defer n V A q))
    using elect-rej-def-combination result-imp-rej mod-m mod-n fin-p fin-q
    unfolding 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 \Rightarrow bool **where**
non-blocking m \equiv
 SCF-result.electoral-module m \wedge
 $(\forall A V p. ((A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p) \longrightarrow \text{reject } m V A p \neq A))$

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 \Rightarrow bool **where**
electing m \equiv
 SCF-result.electoral-module m \wedge
 $(\forall A V p. (A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p) \longrightarrow \text{elect } m V A p \neq \{\})$

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  $\subseteq$  A
    using electing prof elect-in-alts
    unfolding electing-def
    by metis
  show A  $\subseteq$  elect m V A p
proof (intro subsetI)
  fix a :: 'a
  assume a  $\in$  A
  thus a  $\in$  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 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
  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 \Rightarrow bool **where**
non-electing m \equiv
 SCF-result.electoral-module m
 $\wedge (\forall A V p. \text{profile } V A p \longrightarrow \text{elect } m V A p = \{\})$

lemma *single-rej-decr-def-card*:
 fixes
 m :: ('a, 'v, 'a Result) Electoral-Module and
 A :: 'a set and

$V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assumes
rejecting: *rejects 1 m* **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile V A p*
shows $\text{card } (\text{defer } m \ V \ A \ p) = \text{card } A - 1$
proof –
have *no-elect*:
 $\text{SCF-result.electoral-module } m$
 $\wedge (\forall \ V \ A \ q. \text{profile } V \ A \ q \longrightarrow \text{elect } m \ V \ A \ q = \{\})$
using *non-electing*
unfolding *non-electing-def*
by (*metis (no-types)*)
hence $\text{reject } m \ V \ A \ p \subseteq A$
using *f-prof reject-in-alts*
by *metis*
moreover have $A = A - \text{elect } m \ V \ A \ p$
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
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assumes
eliminating: *eliminates 1 m* **and**
non-electing: *non-electing m* **and**
not-empty: $\text{card } A > 1$ **and**
prof-p: *profile V A p*
shows $\text{card } (\text{defer } m \ V \ A \ p) = \text{card } A - 1$
proof –
have *no-elect*:
 $\text{SCF-result.electoral-module } m$
 $\wedge (\forall \ A \ V \ q. \text{profile } V \ A \ q \longrightarrow \text{elect } m \ V \ A \ q = \{\})$
using *non-electing*
unfolding *non-electing-def*
by (*metis (no-types)*)
hence $\text{reject } m \ V \ A \ p \subseteq A$
using *prof-p reject-in-alts*

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

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

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

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

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

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

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

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

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

$$(a \in \text{defer } m \ V \ A \ p \wedge \text{lifted } V \ A \ p \ q \ a) \longrightarrow a \in \text{defer } m \ V \ A \ q)$$

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

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

fun *dli-rel* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow ('a, 'v) Election rel **where**
dli-rel m = $\{((A, V, p), (A, V, q)) \mid A \ V \ p \ q. (\exists \ a \in \text{defer } m \ V \ A \ p. \text{lifted } V \ A \ p \ q \ a)\}$

lemma *rewrite-dli-as-invariance*:

fixes m :: ('a, 'v, 'a Result) Electoral-Module

shows

defer-lift-invariance m =
 $(\text{SCF-result.electoral-module } m \wedge (\text{is-symmetry } (\text{fun}_{\mathcal{E}} \ m) \ (\text{Invariance } (\text{dli-rel } m))))$

proof (*unfold is-symmetry.simps, safe*)

assume *defer-lift-invariance* m

thus *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: $((A, V, p), (A', V', q)) \in \text{dli-rel } m$

then obtain *a* :: 'a **where**

$a \in \text{defer } m \ V \ A \ p \wedge \text{lifted } V \ A \ p \ q \ a$

unfolding *dli-rel.simps*

by *blast*

moreover with *rel* **have** $A = A' \wedge V = V'$

by *simp*

ultimately show $\text{fun}_{\mathcal{E}} \ m \ (A, V, p) = \text{fun}_{\mathcal{E}} \ m \ (A', V', q)$

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

SCF-result.electoral-module m **and**

$\forall \ E \ E'. (E, E') \in \text{dli-rel } m \longrightarrow \text{fun}_{\mathcal{E}} \ m \ E = \text{fun}_{\mathcal{E}} \ m \ E'$

hence *SCF-result.electoral-module* m $\wedge (\forall \ A \ V \ p \ q.$

$((A, V, p), (A, V, q)) \in \text{dli-rel } m \longrightarrow m \ V \ A \ p = m \ V \ A \ q)$

```

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

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

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

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

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

4.4.11 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:

fixes

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **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* $m V A p \neq \{a\}$

moreover have *def-one*: *defers* 1 m

using *dd*

unfolding *defer-deciding-def*

by *metis*

hence *c-win*: *finite-profile* $V A p \wedge a \in A \wedge (\forall b \in A - \{a\}. \text{wins } V a p b)$

using *winner*

by *auto*

ultimately have $\exists b \in A. b \neq a \wedge \text{defer } m V A p = \{b\}$

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 $a \notin \text{defer } m V A p$

by *force*

hence $a \in \text{reject } m V A p$

using *ccomp c-win electoral-mod-defer-elem dd equals0D*

unfolding *defer-deciding-def non-electing-def condorcet-compatibility-def*

by *metis*

moreover have $a \notin \text{reject } m V A p$

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 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

assumes

ccomp: *condorcet-compatibility* m **and**
dd: *defer-deciding* m

shows *defer-condorcet-consistency* m

proof (*unfold defer-condorcet-consistency-def, safe*)

show *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 ∈ 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 ∈ 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
    SCF-result.electoral-module m and
    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 ⊆ A ∧
  (∀ a ∈ B.

```

$$\text{indep-of-alt } m \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } m \ V \ A \ p)) \wedge$$

$$(\forall \ a \in A - B.$$

$$\text{indep-of-alt } n \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } n \ V \ A \ p))$$
using *assms*
unfolding *disjoint-compatibility-def*
by *metis*
hence

$$\exists \ B \subseteq A.$$

$$(\forall \ a \in A - B.$$

$$\text{indep-of-alt } n \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } n \ V \ A \ p)) \wedge$$

$$(\forall \ a \in B.$$

$$\text{indep-of-alt } m \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } m \ V \ A \ p))$$
by *blast*
thus $\exists \ B \subseteq A.$

$$(\forall \ a \in B.$$

$$\text{indep-of-alt } n \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } n \ V \ A \ p)) \wedge$$

$$(\forall \ a \in A - B.$$

$$\text{indep-of-alt } m \ V \ A \ a \wedge (\forall \ p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } m \ V \ A \ p))$$
by *fastforce*
qed

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

theorem *dl-inv-imp-def-mono[simp]*:
fixes $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
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* $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 bool **where**
 $\text{condorcet-consistency } m \equiv$
 $\text{SCF-result.electoral-module } m \wedge$
 $(\forall \ A \ V \ p \ a. \text{condorcet-winner } V \ A \ p \ a \longrightarrow$
 $(m \ V \ A \ p = (\{e \in A. \text{condorcet-winner } V \ A \ p \ e\}, A - (\text{elect } m \ V \ A \ p), \{\})))$

lemma *condorcet-consistency'*:
fixes $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
shows *condorcet-consistency* $m =$
 $(\text{SCF-result.electoral-module } m \wedge$
 $(\forall \ A \ V \ p \ a. \text{condorcet-winner } V \ A \ p \ a \longrightarrow$
 $(m \ V \ A \ p = (\{a\}, A - (\text{elect } m \ V \ A \ p), \{\}))))$

proof (*safe*)
assume *condorcet-consistency* m
thus *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
  SCF-result.electoral-module m and
   $\forall A V p a. \text{condorcet-winner } V A p a \longrightarrow m V A p = (\{a\}, A - \text{elect } m V A p, \{\})$ 
thus condorcet-consistency m
  using cond-winner-unique
  unfolding condorcet-consistency-def
  by (metis (mono-tags, lifting))
qed

lemma condorcet-consistency'':
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  shows condorcet-consistency m =
    (SCF-result.electoral-module m  $\wedge$ 
     ( $\forall A V p a. \text{condorcet-winner } V A p a \longrightarrow m V A p = (\{a\}, A - \{a\}, \{\})$ ))
proof (unfold condorcet-consistency', 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
       $\forall A V p a'. \text{condorcet-winner } V A p a' \longrightarrow m V A p = (\{a\}, A - \text{elect } m V A p, \{\})$ 
    ultimately show m V A p = ({a}, A - {a}, {})
      using fst-conv
      by metis
  }
  {
    moreover assume

```

```

     $\forall A V p a'. \text{condorcet-winner } V A p a' \longrightarrow m V A p = (\{a'\}, A - \{a'\}, \{\})$ 
  ultimately show  $m V A p = (\{a\}, A - \text{elect } m V A p, \{\})$ 
    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-Module  $\Rightarrow$  bool where
  monotonicity m  $\equiv$ 
    SCF-result.electoral-module m  $\wedge$ 
    ( $\forall A V p q a. a \in \text{elect } m V A p \wedge \text{lifted } V A p q a \longrightarrow a \in \text{elect } m V A q$ )
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 (funE m) (Invariance r)) = (funE 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

```

```

     $X :: ('a, 'v) \text{ Election set}$ 
assumes
     $\text{equiv } X \text{ } r$  and
     $\text{is-symmetry } (\text{fun}_{\mathcal{E}} \text{ } m) \text{ (Invariance } r)$ 
shows  $\forall A \in X // r. \forall E \in A. \pi_Q (\text{fun}_{\mathcal{E}} \text{ } m) A = \text{fun}_{\mathcal{E}} \text{ } m E$ 
using  $\text{invariance-is-congruence pass-to-quotient assms}$ 
by  $\text{blast}$ 

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  $('a, 'v) \text{ Evaluation-Function} =$ 
 $'v \text{ set} \Rightarrow 'a \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow \text{enat}$ 

```

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

```

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

```

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

```

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 $p = \text{Max } \{e$ V b A $p \mid b. b \in A\}$
proof –
let $?set = \{e$ V b A $p \mid b. b \in A\}$ **and**
 $?eMax = \text{Max } \{e$ 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 $?set \neq \{\}$
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 $e \ t \ r \ V \ A \ p = (\text{if } \text{finite } A \text{ then } \{a \in A . r (e \ V \ a \ A \ p) \ t\} \text{ else } \{\})$

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

4.7.2 Social-Choice Definitions

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

4.7.3 Social-Choice Eliminators

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

fun *max-eliminator* :: ('a, 'v) *Evaluation-Function* \Rightarrow ('a, 'v, 'a *Result*) *Electoral-Module* **where**
max-eliminator $e \ V \ A \ p =$
 $\text{less-eliminator } e \ (\text{Max } \{e \ V \ x \ A \ p \mid x. x \in A\}) \ V \ A \ p$

fun *leq-eliminator* :: ('a, 'v) *Evaluation-Function* \Rightarrow *Threshold-Value* \Rightarrow ('a, 'v, 'a *Result*) *Electoral-Module* **where**
leq-eliminator $e \ t \ V \ A \ p = \text{elimination-module } e \ t \ (\leq) \ V \ A \ p$

fun *min-eliminator* :: ('a, 'v) *Evaluation-Function* \Rightarrow ('a, 'v, 'a *Result*) *Electoral-Module* **where**
min-eliminator $e \ V \ A \ p =$
 $\text{leq-eliminator } e \ (\text{Min } \{e \ V \ x \ A \ p \mid x. x \in A\}) \ V \ A \ p$

fun *less-average-eliminator* :: ('a, 'v) *Evaluation-Function* \Rightarrow ('a, 'v, 'a *Result*) *Electoral-Module* **where**
less-average-eliminator $e \ V \ A \ p = \text{less-eliminator } e \ (\text{average } e \ V \ A \ p) \ V \ A \ p$

fun *leq-average-eliminator* :: ('a, 'v) *Evaluation-Function* \Rightarrow ('a, 'v, 'a *Result*) *Electoral-Module* **where**
leq-average-eliminator $e \ V \ A \ p = \text{leq-eliminator } e \ (\text{average } e \ V \ A \ p) \ V \ A \ p$

4.7.4 Soundness

lemma *elim-mod-sound[simp]*:
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $t :: \text{Threshold-Value}$ **and**


```

    r :: Threshold-Relation
  shows SCF-result.electoral-module (elimination-module e t r)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma less-elim-sound[simp]:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    t :: Threshold-Value
  shows SCF-result.electoral-module (less-eliminator e t)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma leq-elim-sound[simp]:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    t :: Threshold-Value
  shows SCF-result.electoral-module (leq-eliminator e t)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma max-elim-sound[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  shows SCF-result.electoral-module (max-eliminator e)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma min-elim-sound[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  shows SCF-result.electoral-module (min-eliminator e)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma less-avg-elim-sound[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  shows SCF-result.electoral-module (less-average-eliminator e)
  unfolding SCF-result.electoral-module.simps
  by auto

lemma leq-avg-elim-sound[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  shows SCF-result.electoral-module (leq-average-eliminator e)
  unfolding 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  $\text{Max } \{e\ V\ x\ A\ p \mid x. x \in A\} = \text{Max } \{e\ V\ x\ A\ p' \mid x. x \in A\}$ 
    by metis
  thus less-eliminator e (Max {e V x A p | x. x ∈ A}) V A p =
    less-eliminator e (Max {e V x A p' | x. x ∈ 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  $\text{Min } \{e\ V\ x\ A\ p \mid x. x \in A\} = \text{Min } \{e\ V\ x\ A\ p' \mid x. x \in A\}$ 
    by metis
  thus leq-eliminator e (Min {e V x A p | x. x ∈ A}) V A p =
    leq-eliminator e (Min {e V x A p' | x. x ∈ 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
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$
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 $\text{average } e \ V \ A \ p = \text{average } e \ V \ A \ p'$
unfolding *average.simps*
by *auto*
thus $\text{less-eliminator } e \ (\text{average } e \ V \ A \ p) \ V \ A \ p =$
 $\text{less-eliminator } e \ (\text{average } e \ V \ A \ p') \ V \ A \ p'$
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 $e :: ('a, 'v) \text{ Evaluation-Function}$
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
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$
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 $\text{average } e \ V \ A \ p = \text{average } e \ V \ A \ p'$
unfolding *average.simps*
by *auto*
thus $\text{leq-eliminator } e \ (\text{average } e \ V \ A \ p) \ V \ A \ p =$
 $\text{leq-eliminator } e \ (\text{average } e \ V \ A \ p') \ V \ A \ p'$
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**
 $t :: \text{Threshold-Value}$ **and**
 $r :: \text{Threshold-Relation}$
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**
 $t :: \text{Threshold-Value}$
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**
 $t :: \text{Threshold-Value}$
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 *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
 $c\text{-win} :: \text{condorcet-winner } V A p a$ **and**
 $rej\text{-}a :: a \in \text{reject } (\text{max-eliminator } e) V A p$
have $e V a A p = \text{Max } \{e V b A p \mid b. b \in A\}$
using *c-win cond-winner-imp-max-eval-val assms*
by *fastforce*
hence $a \notin \text{reject } (\text{max-eliminator } e) V A p$
by *simp*
thus *False*
using *rej-a*
by *linarith*
next
fix
 $A :: 'a$ *set* **and**
 $V :: 'v$ *set* **and**
 $p :: ('a, 'v)$ *Profile* **and**
 $a :: 'a$
assume $a \in \text{elect } (\text{max-eliminator } e) V A p$
moreover have $a \notin \text{elect } (\text{max-eliminator } e) V A p$
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
 $\forall A e e' d d' r r'.$
 $(\text{well-formed-SCF } A (e, r, d) \wedge \text{well-formed-SCF } A (e', r', d')) \longrightarrow$
 $\text{well-formed-SCF } A (\text{agg } A (e, r, d) (e', r', d'))$

4.8.2 Properties

definition *agg-commutative* :: *'a Aggregator* \Rightarrow *bool* **where**

agg-commutative agg \equiv
 $\text{aggregator agg} \wedge (\forall A e e' d d' r r'.$
 $\text{agg } A (e, r, d) (e', r', d') = \text{agg } A (e', r', d') (e, r, d))$

definition *agg-conservative* :: *'a Aggregator* \Rightarrow *bool* **where**

agg-conservative agg \equiv
 $\text{aggregator agg} \wedge$
 $(\forall A e e' d d' r r'.$
 $((\text{well-formed-SCF } A (e, r, d) \wedge \text{well-formed-SCF } A (e', r', d')) \longrightarrow$
 $\text{elect-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (e \cup e') \wedge$
 $\text{reject-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (r \cup r') \wedge$
 $\text{defer-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (d \cup d'))$)

end

4.9 Maximum Aggregator

theory *Maximum-Aggregator*

imports *Aggregator*

begin

The max(imum) aggregator takes two partitions of an alternative set A as input. It returns a partition where every alternative receives the maximum result of the two input partitions.

4.9.1 Definition

fun *max-aggregator* :: *'a Aggregator* **where**

max-aggregator A (e, r, d) (e', r', d') =

$$\begin{aligned}
& (e \cup e', \\
& A - (e \cup e' \cup d \cup d'), \\
& (d \cup d') - (e \cup e'))
\end{aligned}$$

4.9.2 Auxiliary Lemma

lemma *max-agg-rej-set*:

fixes

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ **and**

$a :: 'a$

assumes

wf-first-mod: *well-formed-SCF* $A \ (e, r, d)$ **and**

wf-second-mod: *well-formed-SCF* $A \ (e', r', d')$

shows *reject-r* (*max-aggregator* $A \ (e, r, d) \ (e', r', d')$) = $r \cap r'$

proof –

have $A - (e \cup d) = r$

using *wf-first-mod result-imp-rej*

by *metis*

moreover have $A - (e' \cup d') = r'$

using *wf-second-mod result-imp-rej*

by *metis*

ultimately have $A - (e \cup e' \cup d \cup d') = r \cap r'$

by *blast*

moreover have $\{l \in A. l \notin e \cup e' \cup d \cup d'\} = A - (e \cup e' \cup d \cup d')$

unfolding *set-diff-eq*

by *simp*

ultimately show *reject-r* (*max-aggregator* $A \ (e, r, d) \ (e', r', d')$) = $r \cap r'$

by *simp*

qed

4.9.3 Soundness

theorem *max-agg-sound*[*simp*]: *aggregator max-aggregator*

proof (*unfold aggregator-def max-aggregator.simps well-formed-SCF.simps disjoint3.simps set-equals-partition.simps, safe*)

fix

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ **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

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ **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

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ and

$a :: 'a$

assume

elect-a: $a \in \text{elect-}r \ (\text{max-aggregator } A \ (e, r, d) \ (e', r', d'))$ and

a-not-in-e': $a \notin e'$

have $a \in e \cup e'$

using *elect-a*

by *simp*

thus $a \in e$

using *a-not-in-e'*

by *simp*

next

fix

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ and

$a :: 'a$

assume

wf-result: *well-formed-SCF* $A \ (e', r', d')$ and

reject-a: $a \in \text{reject-}r \ (\text{max-aggregator } A \ (e, r, d) \ (e', r', d'))$ and

a-not-in-r': $a \notin r'$

have $a \in r \cup r'$

using *wf-result reject-a*

by *force*

thus $a \in r$

using *a-not-in-r'*

by *simp*

next

fix

$A \ e \ e' \ d \ d' \ r \ r' :: 'a \ \text{set}$ and

$a :: 'a$

assume

defer-a: $a \in \text{defer-}r \ (\text{max-aggregator } A \ (e, r, d) \ (e', r', d'))$ and

```

    a-not-in-d':  $a \notin d'$ 
  have  $a \in d \cup d'$ 
    using defer-a
    by force
  thus  $a \in d$ 
    using a-not-in-d'
    by simp
qed

```

The max-aggregator is commutative.

```

theorem max-agg-comm[simp]: agg-commutative max-aggregator
  unfolding agg-commutative-def
  by auto

end

```

4.10 Termination Condition

```

theory Termination-Condition
  imports Social-Choice-Types/Result
begin

```

The termination condition is used in loops. It decides whether or not to terminate the loop after each iteration, depending on the current state of the loop.

```

type-synonym 'r Termination-Condition = 'r Result  $\Rightarrow$  bool

end

```

4.11 Defer Equal Condition

```

theory Defer-Equal-Condition
  imports Termination-Condition
begin

```

This is a family of termination conditions. For a natural number n , the according defer-equal condition is true if and only if the given result's defer-set contains exactly n elements.

```

fun defer-equal-condition :: nat  $\Rightarrow$  'a Termination-Condition where

```

defer-equal-condition $n(e, r, d) = (\text{card } d = n)$

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]: SCF-result.electoral-module defer-module
  unfolding 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 ( $\lambda$  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*: *SCF-result.electoral-module elect-first-module*

proof (*intro 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-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 × ('a, 'v, 'r) Electoral-Module

fun consensus-K :: ('a, 'v, 'r) Consensus-Class ⇒ ('a, 'v) Consensus where
  consensus-K K = fst K

fun rule-K :: ('a, 'v, 'r) Consensus-Class ⇒ ('a, 'v, 'r) Electoral-Module where
  rule-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  $\mathcal{K}_E$  :: ('a, 'v, 'r Result) Consensus-Class ⇒ 'r ⇒ ('a, 'v) Election set where
   $\mathcal{K}_E$  K w =
    {(A, V, p) | A V p. (consensus-K K) (A, V, p) ∧ finite-profile V A p
      ∧ elect (rule-K K) V A p = {w}}

fun elections-K :: ('a, 'v, 'r Result) Consensus-Class ⇒ ('a, 'v) Election set where
  elections-K K =  $\bigcup ((\mathcal{K}_E K) \text{ 'UNIV})$ 

```

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 ⇒ ('a, 'v, 'r) Electoral-Module ⇒
  bool where
  well-formed c m ≡
    ∀ A V V' p p'.
      profile V A p ∧ profile V' A p' ∧ c (A, V, p) ∧ c (A, V', p')
        → 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
       $w = (\lambda V A p. \text{if } c(A, V, p) \text{ then } m V A p \text{ else defer-module } V A p)$ 
    in ( $c, w$ ))
```

5.3.3 Auxiliary Lemmas

lemma unanimity'-consensus-imp-elect-fst-mod-well-formed:

fixes $a :: 'a$

shows well-formed

$(\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c$
 $\wedge \text{equal-top}_C' a c) \text{ elect-first-module}$

proof (unfold well-formed-def, safe)

fix

$a :: 'a$ **and**

$A :: 'a \text{ set}$ **and**

$V V' :: 'v :: \text{wellorder set}$ **and**

$p p' :: ('a, 'v) \text{ Profile}$

let $?cond = \lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-top}_C' a c$

assume

$\text{prof-p: profile } V A p$ **and**

$\text{prof-p': profile } V' A p'$ **and**

$\text{eq-top-p: equal-top}_C' a (A, V, p)$ **and**

$\text{eq-top-p': equal-top}_C' a (A, V', p')$ **and**

$\text{not-empty-A: nonempty-set}_C (A, V, p)$ **and**

$\text{not-empty-A': nonempty-set}_C (A, V', p')$ **and**

$\text{not-empty-p: nonempty-profile}_C (A, V, p)$ **and**

$\text{not-empty-p': nonempty-profile}_C (A, V', p')$

hence

$\text{cond-Ap: ?cond } (A, V, p)$ **and**

$\text{cond-Ap': ?cond } (A, V', p')$

by simp-all

have $\forall a' \in A.$

$((\text{above } (p \text{ (least } V)) a' = \{a'\}) = (\text{above } (p' \text{ (least } V')) a' = \{a'\}))$

proof

fix $a' :: 'a$

assume $a' \text{-in-A: } a' \in A$

show $(\text{above } (p \text{ (least } V)) a' = \{a'\}) = (\text{above } (p' \text{ (least } V')) a' = \{a'\})$

proof (cases)

assume $a' = a$

thus ?thesis

using cond-Ap cond-Ap' Collect-mem-eq LeastI empty-Collect-eq equal-top_C'.simps
 nonempty-profile_C.simps least.simps

```

    by (metis (no-types, lifting))
next
  assume a'-neq-a:  $a' \neq a$ 
  have non-empty:  $V \neq \{\}$   $\wedge$   $V' \neq \{\}$ 
    using not-empty-p not-empty-p'
    by simp
  hence  $A \neq \{\}$   $\wedge$  linear-order-on  $A$  ( $p$  (least  $V$ ))
     $\wedge$  linear-order-on  $A$  ( $p'$  (least  $V'$ ))
    using not-empty-A not-empty-A' prof-p prof-p' 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 ( $a \in \text{above } (p \text{ (least } V))$ )  $a' \vee a' \in \text{above } (p \text{ (least } V))$   $a$ 
     $\wedge$  ( $a \in \text{above } (p' \text{ (least } V'))$ )  $a' \vee a' \in \text{above } (p' \text{ (least } V'))$   $a$ 
    using a'-in-A a'-neq-a eq-top-p
  unfolding above-def linear-order-on-def total-on-def
  by auto
  hence
    ( $\text{above } (p \text{ (least } V))$   $a = \{a\} \wedge \text{above } (p \text{ (least } V))$   $a' = \{a'\}$ 
       $\longrightarrow a = a'$ )
     $\wedge$  ( $\text{above } (p' \text{ (least } V'))$   $a = \{a\} \wedge \text{above } (p' \text{ (least } V'))$   $a' = \{a'\}$ 
       $\longrightarrow a = a'$ )
    by auto
  thus ?thesis
    using bot-nat-0.not-eq-extremum card-0-eq cond-Ap cond-Ap'
      enumerate-0 enumerate-in-set equal-topC'.simps
      finite-enumerate-in-set non-empty least.simps
    by metis
qed
qed
thus elect-first-module  $V$   $A$   $p = \text{elect-first-module } V' A p'$ 
  by auto
qed

lemma strong-unanimity'consensus-imp-elect-fst-mod-completely-determined:
  fixes  $r :: 'a$  Preference-Relation
  shows well-formed
    ( $\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C' r c$ ) elect-first-module
proof (unfold well-formed-def, clarify)
fix
   $a :: 'a$  and
   $A :: 'a$  set and
   $V V' :: 'v :: \text{wellorder set}$  and
   $p p' :: ('a, 'v)$  Profile
let ?cond =  $\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C' r c$ 
assume
  prof-p: profile  $V$   $A$   $p$  and
  prof-p': profile  $V'$   $A$   $p'$  and

```

$eq\text{-}vote\text{-}p$: $equal\text{-}vote_C' r (A, V, p)$ and
 $eq\text{-}vote\text{-}p'$: $equal\text{-}vote_C' r (A, V', p')$ and
 $not\text{-}empty\text{-}A$: $nonempty\text{-}set_C (A, V, p)$ and
 $not\text{-}empty\text{-}A'$: $nonempty\text{-}set_C (A, V', p')$ and
 $not\text{-}empty\text{-}p$: $nonempty\text{-}profile_C (A, V, p)$ and
 $not\text{-}empty\text{-}p'$: $nonempty\text{-}profile_C (A, V', p')$
hence
 $cond\text{-}Ap$: $?cond (A, V, p)$ and
 $cond\text{-}Ap'$: $?cond (A, V', p')$
by *simp-all*
have $p (\text{least } V) = r \wedge p' (\text{least } V') = r$
using $eq\text{-}vote\text{-}p$ $eq\text{-}vote\text{-}p'$ $not\text{-}empty\text{-}p$ $not\text{-}empty\text{-}p'$
 $bot\text{-}nat\text{-}0.not\text{-}eq\text{-}extremum$ $card\text{-}0\text{-}eq$ $enumerate\text{-}0$
 $enumerate\text{-}in\text{-}set$ $equal\text{-}vote_C'.simps$ $finite\text{-}enumerate\text{-}in\text{-}set$
 $nonempty\text{-}profile_C.simps$ $least.elims$
by (*metis* (*no-types*, *lifting*))
thus $elect\text{-}first\text{-}module\ V\ A\ p = elect\text{-}first\text{-}module\ V'\ A\ p'$
by *auto*
qed

lemma *strong-unanimity'consensus-imp-elect-fst-mod-well-formed*:
fixes $r :: 'a\ \text{Preference-Relation}$
shows *well-formed*
 $(\lambda\ c. nonempty\text{-}set_C\ c \wedge nonempty\text{-}profile_C\ c$
 $\wedge equal\text{-}vote_C' r\ c) elect\text{-}first\text{-}module$
using *strong-unanimity'consensus-imp-elect-fst-mod-completely-determined*
by *blast*

lemma *cons-domain-valid*:
fixes $C :: ('a, 'v, 'r\ \text{Result})\ \text{Consensus-Class}$
shows $elections\text{-}\mathcal{K}\ C \subseteq well\text{-}formed\text{-}elections$
proof
fix $E :: ('a, 'v)\ \text{Election}$
assume $E \in elections\text{-}\mathcal{K}\ C$
hence $fun_{\mathcal{E}}\ profile\ E$
unfolding $\mathcal{K}_{\mathcal{E}}.simps$
by *force*
thus $E \in well\text{-}formed\text{-}elections$
unfolding *well-formed-elections-def*
by *simp*
qed

lemma *cons-domain-finite*:
fixes $C :: ('a, 'v, 'r\ \text{Result})\ \text{Consensus-Class}$
shows
 $finite$: $elections\text{-}\mathcal{K}\ C \subseteq finite\text{-}elections$ **and**
 $finite\text{-}voters$: $elections\text{-}\mathcal{K}\ C \subseteq finite\text{-}elections\text{-}\mathcal{V}$
proof –
have $\forall\ E \in elections\text{-}\mathcal{K}\ C.$

```

    funE 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 funE.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 $c \equiv \exists K. \text{consensus-}\mathcal{K} \ c \ K$

Unanimity condition.

definition *unanimity* :: ($'a$, $'v :: \text{wellorder}$, $'a \text{ Result}$) *Consensus-Class* **where**
unanimity $\equiv \text{consensus-choice unanimity}_{\mathcal{C}} \text{ elect-first-module}$

Strong unanimity condition.

definition *strong-unanimity* :: ($'a$, $'v :: \text{wellorder}$, $'a \text{ Result}$) *Consensus-Class* **where**
strong-unanimity $\equiv \text{consensus-choice strong-unanimity}_{\mathcal{C}} \text{ elect-first-module}$

5.3.5 Properties

definition *consensus-rule-anonymity* :: ($'a$, $'v$, $'r$) *Consensus-Class* \Rightarrow bool **where**
consensus-rule-anonymity $c \equiv$
 $(\forall A \ V \ p \ \pi :: ('v \Rightarrow 'v).$
 $\text{bij } \pi \longrightarrow$
 $(\text{let } (A', V', q) = (\text{rename } \pi \ (A, V, p)) \text{ in}$
 $\text{profile } V \ A \ p \longrightarrow \text{profile } V' \ A' \ q$
 $\longrightarrow \text{consensus-}\mathcal{K} \ c \ (A, V, p)$
 $\longrightarrow (\text{consensus-}\mathcal{K} \ c \ (A', V', q) \wedge (\text{rule-}\mathcal{K} \ c \ V \ A \ p = \text{rule-}\mathcal{K} \ c \ V' \ A' \ q))))$

fun *consensus-rule-anonymity'* :: ($'a$, $'v$) *Election set* \Rightarrow
 $('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow$ bool **where**
consensus-rule-anonymity' $X \ C =$
 $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)) \ (\text{Invariance } (\text{anonymity}_{\mathcal{R}} \ X))$

fun (in *result-properties*) *consensus-rule-neutrality* :: ($'a$, $'v$) *Election set* \Rightarrow
 $('a, 'v, 'b \text{ Result}) \text{ Consensus-Class} \Rightarrow$ bool **where**
consensus-rule-neutrality $X \ C =$
 $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C))$
 $(\text{action-induced-equivariance}$
 $(\text{carrier neutrality}_{\mathcal{G}}) \ X \ (\varphi\text{-neutral } X) \ (\text{set-action } \psi\text{-neutral}))$

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

$(\text{'a}, \text{'v}, \text{'a rel Result}) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**
 $\text{consensus-rule-reversal-symmetry } X \ C = \text{is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$
 $(\text{action-induced-equivariance } (\text{carrier reversal}_G) \ X \ (\varphi\text{-reverse } X) \ (\text{set-action } \psi\text{-reverse}))$

5.3.6 Inference Rules

lemma *if-else-cons-equivar*:

fixes

$m \ n :: (\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module}$ **and**
 $c :: (\text{'a}, \text{'v}) \text{ Consensus}$ **and**
 $G :: \text{'b set}$ **and**
 $X :: (\text{'a}, \text{'v}) \text{ Election set}$ **and**
 $\varphi :: (\text{'b}, (\text{'a}, \text{'v}) \text{ Election}) \text{ binary-fun}$ **and**
 $\psi :: (\text{'b}, \text{'a}) \text{ binary-fun}$ **and**
 $f :: \text{'a Result} \Rightarrow \text{'a set}$

defines

$\text{equivar} \equiv \text{action-induced-equivariance } G \ X \ \varphi \ (\text{set-action } \psi)$ **and**
 $\text{if-else-cons} \equiv (c, (\lambda \ V \ A \ p. \text{if } c \ (A, \ V, \ p) \text{ then } m \ V \ A \ p \text{ else } n \ V \ A \ p))$

assumes

$\text{equivar-m}: \text{is-symmetry } (f \circ \text{fun}_{\mathcal{E}} \ m)$ **equivar** **and**
 $\text{equivar-n}: \text{is-symmetry } (f \circ \text{fun}_{\mathcal{E}} \ n)$ **equivar** **and**
 $\text{invar-cons}: \text{is-symmetry } c \ (\text{Invariance } (\text{action-induced-rel } G \ X \ \varphi))$

shows $\text{is-symmetry } (f \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ \text{if-else-cons}))$

$(\text{action-induced-equivariance } G \ X \ \varphi \ (\text{set-action } \psi))$

proof $(\text{unfold rewrite-equivariance, intro ballI impI})$

fix

$E :: (\text{'a}, \text{'v}) \text{ Election}$ **and**
 $g :: \text{'b}$

assume

$g\text{-in-}G: g \in G$ **and**
 $E\text{-in-}X: E \in X$

show $(f \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ \text{if-else-cons})) \ (\varphi \ g \ E) =$

$\text{set-action } \psi \ g \ ((f \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ \text{if-else-cons})) \ E)$

proof $(\text{cases } c \ E)$

case *True*

hence $c \ (\varphi \ g \ E)$

using *invar-cons rewrite-invar-ind-by-act g-in-G E-in-X*

by *metis*

hence $(f \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ \text{if-else-cons})) \ (\varphi \ g \ E) =$

$(f \circ \text{fun}_{\mathcal{E}} \ m) \ (\varphi \ g \ E)$

unfolding *if-else-cons-def*

by *simp*

also have $(f \circ \text{fun}_{\mathcal{E}} \ m) \ (\varphi \ g \ E) =$

$\text{set-action } \psi \ g \ ((f \circ \text{fun}_{\mathcal{E}} \ m) \ E)$

using *equivar-m E-in-X g-in-G rewrite-equivariance*

unfolding *equivar-def*

by $(\text{metis } (\text{mono-tags, lifting}))$

also have $(f \circ \text{fun}_{\mathcal{E}} \ m) \ E =$

```

      (f ∘ funε (rule- $\mathcal{K}$  if-else-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 ¬ c (φ g E)
    using invar-cons rewrite-invar-ind-by-act g-in-G E-in-X
    by metis
  hence (f ∘ funε (rule- $\mathcal{K}$  if-else-cons)) (φ g E) =
    (f ∘ funε n) (φ g E)
    unfolding if-else-cons-def
    by simp
  also have (f ∘ funε n) (φ g E) =
    set-action ψ g ((f ∘ funε n) E)
    using equivar-n E-in-X g-in-G rewrite-equivariance
    unfolding equivar-def
    by (metis (mono-tags, lifting))
  also have (f ∘ funε n) E =
    (f ∘ funε (rule- $\mathcal{K}$  if-else-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
    α β :: ('a, 'v) Consensus and
    m :: ('a, 'v, 'a Result) Electoral-Module and
    β' :: 'b ⇒ ('a, 'v) Consensus
  assumes
    beta-sat: β = (λ E. ∃ a. β' a E) and
    beta'-anon: ∀ x. consensus-anonymity (β' x) and
    anon-cons-cond: consensus-anonymity α and
    conditions-univ: ∀ x. well-formed (λ E. α E ∧ β' x E) m
  shows consensus-rule-anonymity (consensus-choice (λ E. α E ∧ β 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 ⇒ 'v
  assume
    bij-π: bij π and
    prof-p: profile V A p and

```

prof-q: profile $V' A' q$ **and**
renamed: rename $\pi (A, V, p) = (A', V', q)$ **and**
consensus-cond:
 $\text{consensus-}\mathcal{K} (\text{consensus-choice } (\lambda E. \alpha E \wedge \beta E) m) (A, V, p)$
hence $(\lambda E. \alpha E \wedge \beta E) (A, V, p)$
by *simp*
hence
 $\text{alpha-}Ap: \alpha (A, V, p)$ **and**
 $\text{beta-}Ap: \beta (A, V, p)$
by *simp-all*
have $\text{alpha-}A\text{-perm-}p: \alpha (A', V', q)$
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*
 $\text{ex-anon-cons-imp-cons-anonymous}[of \beta \beta'] \text{bij-}\pi$
 $\text{prof-p renamed beta'-anon cons-anon-invariant}[of \beta]$
unfolding *consensus-anonymity-def*
by *blast*
ultimately show *em-cond-perm*:
 $\text{consensus-}\mathcal{K} (\text{consensus-choice } (\lambda E. \alpha E \wedge \beta E) m) (A', V', q)$
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**
 $\text{beta'-}x\text{-}Ap: \beta' x (A, V, p)$
by *metis*
hence $\text{beta'-}x\text{-}A\text{-perm-}p: \beta' x (A', V', q)$
using *beta'-anon bij- π prof-p renamed*
 $\text{cons-anon-invariant prof-q}$
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*
 $\text{conditions-univ prof-p prof-q rename.simps prod.inject renamed}$
unfolding *well-formed-def*
by *metis*
thus $\text{rule-}\mathcal{K} (\text{consensus-choice } (\lambda E. \alpha E \wedge \beta E) m) V A p =$
 $\text{rule-}\mathcal{K} (\text{consensus-choice } (\lambda E. \alpha E \wedge \beta E) m) V' A' q$
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* = ($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c$)

have *consensus-anonymity ?ne-cond*

using *nonempty-set-cons-anonymous nonempty-profile-cons-anonymous cons-anon-conj*
by *auto*

moreover have *equal-top_C* = ($\lambda c. \exists a. \text{equal-top}_C' a c$)

by *fastforce*

ultimately have *consensus-rule-anonymity*

(*consensus-choice*

($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-top}_C c$) *elect-first-module*)

using *consensus-choice-anonymous[of equal-top_C]*

equal-top-cons'-anonymous unanimity'-consensus-imp-elect-fst-mod-well-formed

by *fastforce*

moreover have *consensus-choice*

($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-top}_C c$)

elect-first-module =

consensus-choice unanimity_C elect-first-module

using *unanimity_C.simps*

by *metis*

ultimately show *consensus-rule-anonymity (consensus-choice unanimity_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* ($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c$)

using *nonempty-set-cons-anonymous nonempty-profile-cons-anonymous cons-anon-conj*

unfolding *consensus-anonymity-def*

by *simp*

moreover have *equal-vote_C* = ($\lambda c. \exists v. \text{equal-vote}_C' v c$)

by *fastforce*

ultimately have *consensus-rule-anonymity*

(*consensus-choice*

($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C c$) *elect-first-module*)

using *consensus-choice-anonymous[of equal-vote_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 ($\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C c$)

elect-first-module =

consensus-choice strong-unanimity_C elect-first-module

unfolding *strong-unanimity_C.simps*

by *metis*

ultimately show

$\text{consensus-rule-anonymity } (\text{consensus-choice strong-unanimity}_C \text{ elect-first-module})$
 $\text{by } (\text{metis } (\text{no-types}))$
qed

Neutrality

lemma *defer-winners-equivariant:*

fixes
 $G :: 'b \text{ set and}$
 $E :: ('a, 'v) \text{ Election set and}$
 $\varphi :: ('b, ('a, 'v) \text{ Election}) \text{ binary-fun and}$
 $\psi :: ('b, 'a) \text{ binary-fun}$
shows $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_E \text{ defer-module})$
 $(\text{action-induced-equivariance } G \ E \ \varphi \ (\text{set-action } \psi))$
using *rewrite-equivariance*
by *fastforce*

lemma *elect-first-winners-neutral: is-symmetry* $(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module})$
 $(\text{action-induced-equivariance } (\text{carrier neutrality}_G))$
 $\text{well-formed-elections } (\varphi\text{-neutral well-formed-elections})$
 $(\text{set-action } \psi\text{-neutral}_C))$

proof *(unfold rewrite-equivariance, clarify)*

fix
 $A :: 'a \text{ set and}$
 $V :: 'v :: \text{wellorder set and}$
 $p :: ('a, 'v) \text{ Profile and}$
 $\pi :: 'a \Rightarrow 'a$
assume
 $\text{bij-carrier-}\pi: \pi \in \text{carrier neutrality}_G \text{ and}$
 $\text{valid: } (A, V, p) \in \text{well-formed-elections}$
hence *bijective- π :* $\text{bij } \pi$
unfolding *neutrality $_G$ -def*
using *rewrite-carrier*
by *blast*

hence *inv:* $\forall a. a = \pi (\text{the-inv } \pi \ a)$
by *(simp add: f-the-inv-into-f-bij-betw)*

from *bij-carrier- π valid* **have**
 $(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module})$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A, V, p)) =$
 $\{a \in \pi \text{ ' } A. \text{ above } (\text{rel-rename } \pi \ (p \ (\text{least } V))) \ a = \{a\}\}$
by *simp*

moreover have
 $\{a \in \pi \text{ ' } A. \text{ above } (\text{rel-rename } \pi \ (p \ (\text{least } V))) \ a = \{a\}\} =$
 $\{a \in \pi \text{ ' } A. \{b. (a, b) \in \{(\pi \ a, \pi \ b) \mid a \ b. (a, b) \in p \ (\text{least } V)\}\} = \{a\}\}$
unfolding *above-def*
by *simp*

ultimately have *elect-simp:*

$(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module})$
 $(\varphi\text{-neutral well-formed-elections } \pi \ (A, V, p)) =$

$\{a \in \pi \text{ ' } A. \{b. (a, b) \in \{(\pi a, \pi b) \mid a b. (a, b) \in p \text{ (least } V)\}\} = \{a\}\}$
by simp
have $\forall a \in \pi \text{ ' } A. \{b. (a, b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p \text{ (least } V)\}\} =$
 $\{\pi b \mid b. (a, \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p \text{ (least } V)\}\}$
by blast
moreover have $\forall a \in \pi \text{ ' } A.$
 $\{\pi b \mid b. (a, \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p \text{ (least } V)\}\} =$
 $\{\pi b \mid b. (\pi (\text{the-inv } \pi a), \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p \text{ (least } V)\}\}$
using bijective- π
by (simp add: f-the-inv-into-f-bij-betw)
moreover have $\forall a \in \pi \text{ ' } A. \forall b.$
 $((\pi (\text{the-inv } \pi a), \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p \text{ (least } V)\}) =$
 $((\text{the-inv } \pi a, b) \in \{(x, y) \mid x y. (x, y) \in p \text{ (least } V)\})$
using bijective- π rel-rename-helper[*of* π]
by auto
moreover have $\{(x, y) \mid x y. (x, y) \in p \text{ (least } V)\} = p \text{ (least } V)$
by simp
ultimately have
 $\forall a \in \pi \text{ ' } A. (\{b. (a, b) \in \{(\pi a, \pi b) \mid a b. (a, b) \in p \text{ (least } V)\}\} = \{a\}) =$
 $(\{\pi b \mid b. (\text{the-inv } \pi a, b) \in p \text{ (least } V)\} = \{a\})$
by force
hence $\{a \in \pi \text{ ' } A.$
 $\{b. (a, b) \in \{(\pi a, \pi b) \mid a b. (a, b) \in p \text{ (least } V)\}\} = \{a\}\} =$
 $\{a \in \pi \text{ ' } A. \{\pi b \mid b. (\text{the-inv } \pi a, b) \in p \text{ (least } V)\} = \{a\}\}$
by auto
hence $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{ elect-first-module})$
 $(\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) =$
 $\{a \in \pi \text{ ' } A. \{\pi b \mid b. (\text{the-inv } \pi a, b) \in p \text{ (least } V)\} = \{a\}\}$
using elect-simp
by simp
also have $\{a \in \pi \text{ ' } A. \{\pi b \mid b. (\text{the-inv } \pi a, b) \in p \text{ (least } V)\} = \{a\}\} =$
 $\{\pi a \mid a. a \in A \wedge \{\pi b \mid b. (a, b) \in p \text{ (least } V)\} = \{\pi a\}\}$
using bijective- π inv bij-is-inj the-inv-f-f
by fastforce
also have $\{\pi a \mid a. a \in A \wedge \{\pi b \mid b. (a, b) \in p \text{ (least } V)\} = \{\pi a\}\} =$
 $\pi \text{ ' } \{a \in A. \{\pi b \mid b. (a, b) \in p \text{ (least } V)\} = \{\pi a\}\}$
by blast
also have $\pi \text{ ' } \{a \in A. \{\pi b \mid b. (a, b) \in p \text{ (least } V)\} = \{\pi a\}\} =$
 $\pi \text{ ' } \{a \in A. \pi \text{ ' } \{b \mid b. (a, b) \in p \text{ (least } V)\} = \pi \text{ ' } \{a\}\}$
by blast
finally have
 $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{ elect-first-module})$
 $(\varphi\text{-neutral well-formed-elections } \pi (A, V, p)) =$
 $\pi \text{ ' } \{a \in A. \pi \text{ ' } (\text{above } (p \text{ (least } V)) a) = \pi \text{ ' } \{a\}\}$
unfolding above-def
by simp
moreover have
 $\forall a. (\pi \text{ ' } (\text{above } (p \text{ (least } V)) a) = \pi \text{ ' } \{a\}) =$
 $(\text{the-inv } \pi \text{ ' } \pi \text{ ' } \text{above } (p \text{ (least } V)) a = \text{the-inv } \pi \text{ ' } \pi \text{ ' } \{a\})$

using *bijjective- π bij-betw-the-inv-into bij-def inj-image-eq-iff*
by *metis*
moreover have
 $\forall a. (the_inv\ \pi\ ' \pi\ ' \text{above}\ (p\ (least\ V))\ a = the_inv\ \pi\ ' \pi\ ' \{a\}) =$
 $(\text{above}\ (p\ (least\ V))\ a = \{a\})$
using *bijjective- π bij-betw-imp-inj-on bij-betw-the-inv-into inj-image-eq-iff*
by *metis*
ultimately have
 $(elect_r \circ fun_{\mathcal{E}}\ elect_first_module)$
 $(\varphi\text{-neutral}\ well\text{-formed}\text{-elections}\ \pi\ (A, V, p)) =$
 $\pi\ ' \{a \in A. \text{above}\ (p\ (least\ V))\ a = \{a\}\}$
by *presburger*
moreover have
 $elect\ elect_first_module\ V\ A\ p = \{a \in A. \text{above}\ (p\ (least\ V))\ a = \{a\}\}$
by *simp*
moreover have *set-action ψ -neutral_c π*
 $((elect_r \circ fun_{\mathcal{E}}\ elect_first_module)\ (A, V, p)) =$
 $\pi\ ' (elect\ elect_first_module\ V\ A\ p)$
by *auto*
ultimately show
 $(elect_r \circ fun_{\mathcal{E}}\ elect_first_module)$
 $(\varphi\text{-neutral}\ well\text{-formed}\text{-elections}\ \pi\ (A, V, p)) =$
 $set\text{-action}\ \psi\text{-neutral}_c\ \pi$
 $((elect_r \circ fun_{\mathcal{E}}\ elect_first_module)\ (A, V, p))$
by *blast*
qed

lemma *strong-unanimity-neutral:*
defines $domain \equiv well\text{-formed}\text{-elections} \cap Collect\ strong\text{-unanimity}_C$
 — We want to show neutrality on a set as general as possible, as this implies
 subset neutrality.
shows *SCF-properties.consensus-rule-neutrality domain strong-unanimity*
proof —
have *coincides:*
 $\forall \pi. \forall E \in domain. \varphi\text{-neutral}\ domain\ \pi\ E =$
 $\varphi\text{-neutral}\ well\text{-formed}\text{-elections}\ \pi\ E$
unfolding *domain-def φ -neutral.simps*
by *auto*
hence $neutrality_{\mathcal{R}}\ domain \subseteq neutrality_{\mathcal{R}}\ well\text{-formed}\text{-elections}$
unfolding *neutrality_R.simps action-induced-rel.simps*
using *domain-def*
by *auto*
hence *consensus-neutrality domain strong-unanimity_C*
using *strong-unanimity_C-neutral invar-under-subset-rel*
unfolding *consensus-neutrality.simps*
by *blast*
hence *is-symmetry strong-unanimity_C*
 $(Invariance\ (action\text{-induced}\text{-rel}\ (carrier\ neutrality_{\mathcal{G}})$
 $domain\ (\varphi\text{-neutral}\ well\text{-formed}\text{-elections))))$

unfolding *consensus-neutrality.simps neutrality_R.simps*
using *coincides coinciding-actions-ind-equal-rel*
by *metis*
moreover have *is-symmetry (elect-r \circ fun_E elect-first-module)*
(action-induced-equivariance (carrier neutrality_G)
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_E (rule- \mathcal{K} strong-unanimity))*
(action-induced-equivariance (carrier neutrality_G) domain
(φ -neutral well-formed-elections) (set-action ψ -neutral_C))
using *defer-winners-equivariant[of*
carrier neutrality_G domain φ -neutral well-formed-elections ψ -neutral_C]
if-else-cons-equivar[of
elect-r elect-first-module carrier neutrality_G
domain φ -neutral well-formed-elections ψ -neutral_C defer-module
strong-unanimity_C]
unfolding *strong-unanimity-def*
by *fastforce*
thus *?thesis*
unfolding *SCF-properties.consensus-rule-neutrality.simps*
using *coincides equivar-ind-by-act-coincide*
by *(metis (no-types, lifting))*
qed

lemma *strong-unanimity-neutral': 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_C*
unfolding *well-formed-elections-def K_E.simps strong-unanimity-def*
by *force*
moreover from this have *coincide:*
 $\forall \pi. \forall E \in \text{elections-}\mathcal{K} \text{ strong-unanimity.}$
 $\varphi\text{-neutral (well-formed-elections} \cap \text{Collect strong-unanimity}_C) \pi E =$
 $\varphi\text{-neutral (elections-}\mathcal{K} \text{ strong-unanimity)} \pi E$
unfolding *φ -neutral.simps*
using *extensional-continuation-subset*
by *(metis (no-types, lifting))*
ultimately have
is-symmetry (elect-r \circ fun_E (rule- \mathcal{K} strong-unanimity))
(action-induced-equivariance (carrier neutrality_G) (elections- \mathcal{K} strong-unanimity)
(φ -neutral (well-formed-elections \cap Collect strong-unanimity_C))
(set-action ψ -neutral_C))
using *strong-unanimity-neutral*
equivar-under-subset[of
elect-r \circ fun_E (rule- \mathcal{K} strong-unanimity)
well-formed-elections \cap Collect strong-unanimity_C

```

      {( $\varphi$ -neutral ( $well\text{-}formed\text{-}elections \cap Collect\ strong\text{-}unanimity_C$ )  $g$ ,
        set-action  $\psi$ -neutralC  $g$ ) |  $g. g \in carrier\ neutrality_G$ }
      elections- $\mathcal{K}$  strong-unanimity]
unfolding action-induced-equivariance-def SCF-properties.consensus-rule-neutrality.simps
by blast
thus ?thesis
unfolding SCF-properties.consensus-rule-neutrality.simps
using coincide
      equivar-ind-by-act-coincide[of
        carrier neutralityG elections- $\mathcal{K}$  strong-unanimity
         $\varphi$ -neutral (elections- $\mathcal{K}$  strong-unanimity)
         $\varphi$ -neutral ( $well\text{-}formed\text{-}elections \cap Collect\ strong\text{-}unanimity_C$ )
        elect-r  $\circ$  fun $\mathcal{E}$  (rule- $\mathcal{K}$  strong-unanimity) set-action  $\psi$ -neutralC]
by (metis (no-types))
qed

lemma strong-unanimity-closed-under-neutrality: closed-restricted-rel
  (neutrality $\mathcal{R}$   $well\text{-}formed\text{-}elections$ )  $well\text{-}formed\text{-}elections$ 
  (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
   $prof: (A, V, p) \in well\text{-}formed\text{-}elections$  and
   $cons: (A, V, p) \in \mathcal{K}_{\mathcal{E}}\ strong\text{-}unanimity\ a$  and
   $bij\text{-}carrier\text{-}\pi: \pi \in carrier\ neutrality_G$  and
   $img: \varphi\text{-neutral}\ well\text{-}formed\text{-}elections\ \pi\ (A, V, p) = (A', V', p')$ 
hence  $fin: (A, V, p) \in finite\text{-}elections$ 
unfolding  $\mathcal{K}_{\mathcal{E}}$ .simps finite-elections-def
by simp
hence  $valid': (A', V', p') \in well\text{-}formed\text{-}elections$ 
using  $bij\text{-}carrier\text{-}\pi\ img\ \varphi\text{-neutral}\text{-}action.\text{group-action-axioms}$ 
   $group\text{-}action.\text{element-image}\ prof$ 
unfolding finite-elections-def
by (metis (mono-tags, lifting))
moreover have  $V' = V \wedge A' = \pi\ `A$ 
using  $img\ fin\ alternatives\text{-}rename.\text{elims}\ fstI\ prof\ sndI$ 
unfolding extensional-continuation.simps  $\varphi\text{-neutral.simps}$ 
   $alternatives\text{-}\mathcal{E}.simps\ voters\text{-}\mathcal{E}.simps$ 
by (metis (no-types, lifting))
ultimately have  $prof': finite\text{-}profile\ V'\ A'\ p'$ 
using  $fin\ bij\text{-}carrier\text{-}\pi\ CollectD\ finite\text{-}imageI\ fst\text{-}eqD\ snd\text{-}eqD$ 
unfolding finite-elections-def  $well\text{-}formed\text{-}elections\text{-}def\ alternatives\text{-}\mathcal{E}.simps$ 
   $voters\text{-}\mathcal{E}.simps\ profile\text{-}\mathcal{E}.simps$ 

```

by (*metis* (*no-types*, *lifting*))
 let $?domain = well\text{-}formed\text{-}elections \cap Collect\ strong\text{-}unanimity_C$
 have $((A, V, p), (A', V', p')) \in neutrality_{\mathcal{R}}\ well\text{-}formed\text{-}elections$
 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 *K $_{\mathcal{E}}$.simps strong-unanimity-def well-formed-elections-def*
 by *simp*
 ultimately have *unanimous'*: $(A', V', p') \in ?domain$
 using *strong-unanimity $_C$ -neutral valid'*
 unfolding *consensus-neutrality.simps*
 by *force*
 have *rewrite*: $\forall \pi \in carrier\ neutrality_{\mathcal{G}}.$
 $\varphi\text{-}neutral\ ?domain\ \pi\ (A, V, p) \in ?domain$
 $\longrightarrow (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))$
 $(\varphi\text{-}neutral\ ?domain\ \pi\ (A, V, p)) =$
 $set\text{-}action\ \psi\text{-}neutral_c\ \pi$
 $((elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))\ (A, V, p))$
 using *strong-unanimity-neutral unanimous*
 rewrite-equivariance[*of*
 $elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity)$
 carrier neutrality $_{\mathcal{G}}$?domain
 $\varphi\text{-}neutral\ ?domain\ set\text{-}action\ \psi\text{-}neutral_c]$
 unfolding *SCF-properties.consensus-rule-neutrality.simps*
 by *metis*
 have *img'*: $\varphi\text{-}neutral\ ?domain\ \pi\ (A, V, p) = (A', V', p')$
 using *img unanimous*
 by *simp*
 hence $elect\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity)\ V'\ A'\ p' =$
 $(elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))$
 $(\varphi\text{-}neutral\ ?domain\ \pi\ (A, V, p))$
 by *simp*
 also have
 $(elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))\ (\varphi\text{-}neutral\ ?domain\ \pi\ (A, V, p)) =$
 $set\text{-}action\ \psi\text{-}neutral_c\ \pi$
 $((elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))\ (A, V, p))$
 using *bij-carrier- π img' unanimous' rewrite*
 by *metis*
 also have $(elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity))\ (A, V, p) = \{a\}$
 using *cons*
 unfolding *K $_{\mathcal{E}}$.simps*
 by *simp*
 finally have $elect\ (rule\text{-}\mathcal{K}\ strong\text{-}unanimity)\ V'\ A'\ p' = \{\psi\text{-}neutral_c\ \pi\ a\}$
 by *simp*
 hence $(A', V', p') \in K_{\mathcal{E}}\ strong\text{-}unanimity\ (\psi\text{-}neutral_c\ \pi\ a)$
 unfolding *K $_{\mathcal{E}}$.simps strong-unanimity-def consensus-choice.simps*

```

    using unanimous' prof'
    by simp
  hence  $(A', V', p') \in \text{elections-}\mathcal{K} \text{ strong-unanimity}$ 
    by simp
  hence  $((A, V, p), (A', V', p'))$ 
     $\in \bigcup (\text{range } (\mathcal{K}_{\mathcal{E}} \text{ strong-unanimity})) \times \bigcup (\text{range } (\mathcal{K}_{\mathcal{E}} \text{ strong-unanimity}))$ 
    unfolding elections- $\mathcal{K}$ .simps
    using cons
    by blast
  moreover have
     $\exists \pi \in \text{carrier neutrality}_{\mathcal{G}}.$ 
     $\varphi\text{-neutral well-formed-elections } \pi (A, V, p) = (A', V', p')$ 
    using img bij-carrier- $\pi$ 
    unfolding neutrality $\mathcal{G}$ -def
    by blast
  ultimately show  $(A', V', p') \in \bigcup (\text{range } (\mathcal{K}_{\mathcal{E}} \text{ strong-unanimity}))$ 
    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 score :: ('a, 'v) Election Distance  $\Rightarrow$  ('a, 'v, 'r Result) Consensus-Class  $\Rightarrow$ 
  ('a, 'v) Election  $\Rightarrow$  'r  $\Rightarrow$  ereal where
  score d K E w = Inf (d E ' ( $\mathcal{K}_{\mathcal{E}}$  K w))

```

```

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

5.4.2 Standard Definitions

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

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

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

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

fun $\mathcal{K}_{\mathcal{E}}\text{-std} :: ('a, 'v, 'r \text{ Result})$ Consensus-Class $\Rightarrow 'r \Rightarrow 'a$ set $\Rightarrow 'v$ set \Rightarrow
 $('a, 'v)$ Election set **where**
 $\mathcal{K}_{\mathcal{E}}\text{-std } K w A V =$
 $(\lambda p. (A, V, p)) \text{ ' } (\text{Set.filter}$
 $(\lambda p. (\text{consensus-}\mathcal{K} K) (A, V, p) \wedge \text{elect } (\text{rule-}\mathcal{K} K) V A p = \{w\})$
 $(\text{profiles } V A))$

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 $\Rightarrow ('a, 'v, 'r \text{ Result})$ Consensus-Class \Rightarrow
 $('a, 'v)$ Election $\Rightarrow 'r \Rightarrow$ ereal **where**
score-std $d K E w =$
(if $\mathcal{K}_{\mathcal{E}}\text{-std } K w (\text{alternatives-}\mathcal{E} E) (\text{voters-}\mathcal{E} E) = \{\}$
then ∞ else $\text{Min } (d E \text{ ' } (\mathcal{K}_{\mathcal{E}}\text{-std } K w (\text{alternatives-}\mathcal{E} E) (\text{voters-}\mathcal{E} E))))$

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

```

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 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  $\in$  A  $\wedge$  l  $\in$  S}
proof -
  let ?P =  $\lambda$  A. finite {a#l | a l. a  $\in$  A  $\wedge$  l  $\in$  S}
  have  $\forall$  a A'. finite A'  $\longrightarrow$  a  $\notin$  A'  $\longrightarrow$  ?P A'  $\longrightarrow$  ?P (insert a A')
  proof (clarify)
    fix
      a :: 'a and
      A' :: 'a set
    assume
      fin: finite A' and
      not-in: a  $\notin$  A' and
      fin-set: finite {a#l | a l. a  $\in$  A'  $\wedge$  l  $\in$  S}
    have {a'#l | a' l. a'  $\in$  insert a A'  $\wedge$  l  $\in$  S}
      = {a#l | a l. a  $\in$  A'  $\wedge$  l  $\in$  S}  $\cup$  {a#l | l. l  $\in$  S}

```

```

    by auto
  moreover have finite {a#l | l. l ∈ S}
    using fin-B
    by simp
  ultimately have finite {a'#l | a' l. a' ∈ insert a A' ∧ l ∈ S}
    using fin-set
    by simp
  thus ?P (insert a A')
    by simp
qed
moreover have ?P {}
  by simp
ultimately show ?P A
  using finite-induct[of A ?P] fin-A
  by simp
qed

```

lemma *listset-finiteness*:

```

  fixes l :: 'a set list
  assumes ∀ i::nat. i < length l ⟶ finite (!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 (!i)
    by auto
  moreover assume
    ∀ i::nat < length l. finite (!i) ⟹ finite (listset l)
  ultimately have finite {a'#l' | a' l'. a' ∈ a ∧ l' ∈ (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

```

```

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

```

```

lemma all-ls-elems-same-len:
  fixes l :: 'a set list
  shows  $\forall l' :: 'a \text{ list}. l' \in \text{listset } l \longrightarrow \text{length } l' = \text{length } l$ 
proof (induct l, safe)
  case Nil
  fix l :: 'a list
  assume  $l \in \text{listset } []$ 
  thus  $\text{length } l = \text{length } []$ 
  by simp
next
  case (Cons a l)
  fix
    a :: 'a set and
    l :: 'a set list and
    l' :: 'a list
  assume
     $\forall l'. l' \in \text{listset } l \longrightarrow \text{length } l' = \text{length } l$  and
     $l' \in \text{listset } (a\#l)$ 
  moreover have
     $\forall a' l' :: 'a \text{ set list}. \text{listset } (a'\#l') = \{b\#m \mid b m. b \in a' \wedge m \in \text{listset } l'\}$ 
  by (simp add: set-Cons-def)
  ultimately show  $\text{length } l' = \text{length } (a\#l)$ 
  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- $\alpha$  ' permutations-of-set A = {}
    unfolding pl- $\alpha$ -def
    by simp
  hence  $\forall p \in \text{profiles } V A. \forall v. v \in V \longrightarrow p v = \{\}$ 
    by (simp add: image-subset-iff)
  hence  $\forall p \in ?\text{profs}. (\forall v. v \in V \longrightarrow p v = \{\}) \wedge (\forall v. v \notin V \longrightarrow p v = x)$ 
    by simp
  hence  $\forall p \in ?\text{profs}. p = (\lambda v. \text{if } v \in V \text{ then } \{\} \text{ else } x)$ 
    by (metis (no-types, lifting))
  hence  $?\text{profs} \subseteq \{\lambda v. \text{if } v \in V \text{ then } \{\} \text{ else } x\}$ 
    by blast
  thus finite ?profs
    using finite.emptyI finite-insert finite-subset
    by (metis (no-types, lifting))
next
  let ?profs = (profiles V A  $\cap$  {p.  $\forall v. v \notin V \longrightarrow p v = x$ })
  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- $\alpha$  list-V and
    perm: list-V  $\in$  permutations-of-set V
    using lin-order-pl- $\alpha$  fin-V image-iff length-finite-permutations-of-set
    by metis
  let ?map =  $\lambda p :: ('a, 'v) \text{Profile}. \text{map } p \text{ list-V}$ 
  have  $\forall p \in \text{profiles } V A. \forall v \in V. p v \in (\text{pl-}\alpha \text{ ' permutations-of-set } A)$ 
    by (simp add: image-subset-iff)
  hence  $\forall p \in \text{profiles } V A. (\forall v \in V. \text{linear-order-on } A (p v))$ 

```

using *pl- α -lin-order fin-A False*
by *metis*
moreover have $\forall p \in ?profs. \forall i < \text{length } (?map\ p). (?map\ p)!i = p\ (list-V!i)$
by *simp*
moreover have $\forall i < \text{length } list-V. list-V!i \in V$
using *perm nth-mem*
unfolding *permutations-of-set-def*
by *safe*
moreover have *lens-eq*: $\forall p \in ?profs. \text{length } (?map\ p) = \text{length } list-V$
by *simp*
ultimately have
 $\forall p \in ?profs. \forall i < \text{length } (?map\ p). \text{linear-order-on } A\ ((?map\ p)!i)$
by *simp*
hence *subset-map-profs*: $?map\ ` ?profs \subseteq \{xs. \text{length } xs = \text{card } V \wedge$
 $(\forall i < \text{length } xs. \text{linear-order-on } A\ (xs!i))\}$
using *len lens-eq*
by *fastforce*
have $\forall p1\ p2.$
 $p1 \in ?profs \wedge p2 \in ?profs \wedge p1 \neq p2 \longrightarrow (\exists v \in V. p1\ v \neq p2\ v)$
by *fastforce*
hence $\forall p1\ p2.$
 $p1 \in ?profs \wedge p2 \in ?profs \wedge p1 \neq p2$
 $\longrightarrow (\exists v \in \text{set } list-V. p1\ v \neq p2\ v)$
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
 $\text{finite } \{xs. \text{length } xs = \text{card } V \wedge (\forall i < \text{length } xs. \text{linear-order-on } A\ (xs!i))\}$
proof –
have *finite* $\{r. \text{linear-order-on } A\ r\}$
using *fin-A*
unfolding *linear-order-on-def partial-order-on-def preorder-on-def refl-on-def*
by *simp*
hence *fin-supset*:
 $\forall n. \text{finite } \{xs. \text{length } xs = n \wedge \text{set } xs \subseteq \{r. \text{linear-order-on } A\ r\}\}$
using *Collect-mono finite-lists-length-eq rev-finite-subset*
by *(metis (no-types, lifting))*
have $\forall l \in \{xs. \text{length } xs = \text{card } V \wedge$
 $(\forall i < \text{length } xs. \text{linear-order-on } A\ (xs!i))\}.$
 $\text{set } l \subseteq \{r. \text{linear-order-on } A\ r\}$
using *in-set-conv-nth mem-Collect-eq subsetI*
by *(metis (no-types, lifting))*
hence $\{xs. \text{length } xs = \text{card } V \wedge$
 $(\forall i < \text{length } xs. \text{linear-order-on } A\ (xs!i))\}$

```

       $\subseteq \{xs. \text{length } xs = \text{card } V \wedge \text{set } xs \subseteq \{r. \text{linear-order-on } A \ r\}\}$ 
    by blast
  thus ?thesis
    using fin-supset rev-finite-subset
    by blast
qed
moreover have  $\forall f X Y. \text{inj-on } f X \wedge \text{finite } Y \wedge f ` X \subseteq Y \longrightarrow \text{finite } X$ 
  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  $\forall v \in V. p \ v \in \text{pl-}\alpha \ ` \text{permutations-of-set } A$ 
    using fin-A fin-V
    by auto
  hence  $\forall v \in V. \text{linear-order-on } A \ (p \ v)$ 
    using fin-A pl- $\alpha$ -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- $\alpha \ ` \text{permutations-of-set } A$ )}
    using fin-A lin-order-pl- $\alpha$  prof
    unfolding profile-def

```

by *blast*
 thus $p \in \text{profiles } V \ A$
 using *fin-A fin-V*
 unfolding *profiles.simps*
 by *metis*
 qed
 qed
 next
 case *False*
 assume *not-fin-empty*: $\neg (\text{finite } A \wedge \text{finite } V \wedge A \neq \{\})$
 have $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \longrightarrow \text{permutations-of-set } A = \{\{\}\}$
 unfolding *permutations-of-set-def*
 by *fastforce*
 hence *pl-empty*:
 $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \longrightarrow \text{pl-}\alpha \text{ ' permutations-of-set } A = \{\{\}\}$
 unfolding *pl-}\alpha\text{-def}*
 by *simp*
 hence $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \longrightarrow$
 $(\forall \pi \in \{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\}. \forall v \in V. \pi \ v = \{\})$
 by *fastforce*
 hence $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \longrightarrow$
 $\{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\} = \{\pi. \forall v \in V. \pi \ v = \{\}\}$
 using *image-subset-iff singletonD singletonI pl-empty*
 by *fastforce*
 moreover have $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow \text{profiles } V \ A = \{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\}$
 by *simp*
 ultimately have *all-prof-eq*: $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow \text{profiles } V \ A = \{\pi. \forall v \in V. \pi \ v = \{\}\}$
 by *simp*
 have $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow (\forall p \in \{p. \text{finite-profile } V \ A \ p \wedge (\forall v. v \notin V \longrightarrow p \ v = \{\})\}.$
 $(\forall v \in V. \text{linear-order-on } \{\} \ (p \ v)))$
 unfolding *profile-def*
 by *simp*
 moreover have $\forall r. \text{linear-order-on } \{\} \ r \longrightarrow r = \{\}$
 using *lin-ord-not-empty*
 by *metis*
 ultimately have *non-voters*:
 $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow (\forall p \in \{p. \text{finite-profile } V \ A \ p \wedge (\forall v. v \notin V \longrightarrow p \ v = \{\})\}.$
 $\forall v. p \ v = \{\})$
 by *blast*
 hence $(\forall p. \text{profile } V \ \{\} \ p \wedge (\forall v. v \notin V \longrightarrow p \ v = \{\}))$
 $\longrightarrow (\forall v. p \ v = \{\}) \longrightarrow \text{finite } V \longrightarrow A = \{\}$
 $\longrightarrow \{p. \text{profile } V \ \{\} \ p\} = \{p. \forall v \in V. p \ v = \{\}\}$
 unfolding *profile-def*
 using *lin-ord-not-empty*
 by *auto*

hence $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow (\{p. \text{finite-profile } V A p\} = \{p. \forall v \in V. p v = \{\}\})$
using *non-voters*
by *blast*
hence $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$
 $\longrightarrow \text{profiles } V A = \{p. \text{finite-profile } V A p\}$
using *all-prof-eq*
by *simp*
moreover have $\text{infinite } A \vee \text{infinite } V \longrightarrow \text{profiles } V A = \{\}$
by *simp*
moreover have $\text{infinite } A \vee \text{infinite } V \longrightarrow$
 $\{p. \text{finite-profile } V A p \wedge (\forall v. v \notin V \longrightarrow p v = \{\})\} = \{\}$
by *auto*
moreover have $\text{infinite } A \vee \text{infinite } V \vee A = \{\}$
using *not-fin-empty*
by *simp*
ultimately show $\text{profiles } V A = \{p. \text{finite-profile } V A p\}$
by *blast*
qed

5.4.4 Soundness

lemma (*in result*) *\mathcal{R} -sound*:
fixes
 $K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $d :: ('a, 'v) \text{ Election Distance}$
shows *electoral-module* (*distance- \mathcal{R} d K*)
proof (*unfold electoral-module.simps, safe*)
fix
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
have $\mathcal{R}_W d K V A p \subseteq (\text{limit } A \text{ UNIV})$
using *$\mathcal{R}_W.simps$ 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 Inference Rules

lemma (*in result*) *standard-distance-imp-equal-score*:
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**

$K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class and}$
 $A :: 'a \text{ set and}$
 $V :: 'v \text{ set and}$
 $p :: ('a, 'v) \text{ Profile and}$
 $w :: 'r$
assumes
 $\text{irr-non-}V$: $\text{voters-determine-distance } d \text{ and}$
 std : $\text{standard } d$
shows $\text{score } d \ K \ (A, V, p) \ w = \text{score-std } d \ K \ (A, V, p) \ w$
proof –
have profile-perm-set :
 $\text{profiles } V \ A =$
 $\{p' :: ('a, 'v) \text{ Profile. finite-profile } V \ A \ p'\}$
using $\text{profile-permutation-set}$
by metis
hence $\text{eq-intersect: } \mathcal{K}_E\text{-std } K \ w \ A \ V =$
 $\mathcal{K}_E \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p' :: ('a, 'v) \text{ Profile. finite-profile } V \ A \ p'\}$
by force
have $(\mathcal{K}_E \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p' :: ('a, 'v) \text{ Profile. finite-profile } V \ A \ p'\})$
 $\subseteq (\mathcal{K}_E \ K \ w)$
by simp
hence $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_E \ K \ w)) \leq$
 $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_E \ K \ w \cap$
 $\text{Pair } A \ ' \text{Pair } V \ ' \{p' :: ('a, 'v) \text{ Profile. finite-profile } V \ A \ p'\}))$
using $\text{INF-superset-mono dual-order.refl}$
by metis
moreover **have** $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_E \ K \ w)) \geq$
 $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_E \ K \ w \cap$
 $\text{Pair } A \ ' \text{Pair } V \ ' \{p' :: ('a, 'v) \text{ Profile. finite-profile } V \ A \ p'\}))$
proof (rule INF-greatest)
let $?inf = \text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_E \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\}))$
let $?compl = (\mathcal{K}_E \ K \ w) -$
 $(\mathcal{K}_E \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})$
fix $i :: ('a, 'v) \text{ Election}$
assume el : $i \in \mathcal{K}_E \ K \ w$
have in-intersect :
 $i \in (\mathcal{K}_E \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})$
 $\longrightarrow ?inf \leq d \ (A, V, p) \ i$
using $\text{Complete-Lattices.complete-lattice-class.INF-lower}$
by metis
have $\text{compl-imp-neither-voter-nor-alt-nor-infinite-prof}$:
 $i \in ?compl \longrightarrow (V \neq \text{fst } (\text{snd } i)$
 $\vee A \neq \text{fst } i$
 $\vee \neg \text{finite-profile } V \ A \ (\text{snd } (\text{snd } i)))$
by fastforce
moreover **have** $\text{not-voters-imp-infity: } V \neq \text{fst } (\text{snd } i) \longrightarrow d \ (A, V, p) \ i = \infty$
using std prod.collapse
unfolding standard-def

```

    by metis
  moreover have not-alts-imp-infty:  $A \neq \text{fst } i \longrightarrow d(A, V, p) i = \infty$ 
    using std.prod.collapse
    unfolding standard-def
    by metis
  moreover have  $V = \text{fst } (\text{snd } i) \wedge A = \text{fst } i$ 
     $\wedge \neg \text{finite-profile } V A (\text{snd } (\text{snd } i)) \longrightarrow \text{False}$ 
    using el
    by fastforce
  hence  $i \in ?\text{compl} \longrightarrow d(A, V, p) i = \infty$ 
    using not-alts-imp-infty not-voters-imp-infty
    compl-imp-neither-voter-nor-alt-nor-infinite-prof
    by fastforce
  ultimately have
     $i \in ?\text{compl}$ 
     $\longrightarrow \text{Inf } (d(A, V, p)$ 
       $\quad (\mathcal{K}_{\mathcal{E}} K w \cap \text{Pair } A \text{ ' Pair } V \text{ ' } \{p'. \text{finite-profile } V A p'\}))$ 
       $\leq d(A, V, p) i$ 
    using ereal-less-eq
    by (metis (no-types, lifting))
  thus  $\text{Inf } (d(A, V, p) \text{ ' } (\mathcal{K}_{\mathcal{E}} K w \cap$ 
     $\text{Pair } A \text{ ' Pair } V \text{ ' } \{p'. \text{finite-profile } V A p'\}))$ 
     $\leq d(A, V, p) i$ 
    using in-intersect el
    by blast
qed
ultimately have  $\text{Inf } (d(A, V, p) \text{ ' } \mathcal{K}_{\mathcal{E}} K w) =$ 
   $\text{Inf } (d(A, V, p) \text{ ' } (\mathcal{K}_{\mathcal{E}} K w \cap \text{Pair } A \text{ ' Pair } V \text{ ' } \{p'. \text{finite-profile } V A p'\}))$ 
  using order-antisym
  by simp
also have inf-eq-min-for-std-cons:
   $\dots = \text{score-std } d K (A, V, p) w$ 
proof (cases  $\mathcal{K}_{\mathcal{E}}\text{-std } K w A V = \{\}$ )
case True
  hence  $\text{Inf } (d(A, V, p) \text{ ' } (\mathcal{K}_{\mathcal{E}} K w \cap \text{Pair } A \text{ ' Pair } V \text{ ' } \{p'. \text{finite-profile } V A p'\})) = \infty$ 
    using eq-intersect
    using 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 $\mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ A \ V =$
 $(\mathcal{K}_{\mathcal{E}} \ K \ w) \cap \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p'\}$
using *eq-intersect*
by *blast*
hence *subset-dist- $\mathcal{K}_{\mathcal{E}}$ -std*:
 $d \ (A, \ V, \ p) \ '(\mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ A \ V) \subseteq$
 $d \ (A, \ V, \ p) \ ' \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p'\}$
by *blast*
let $?finite\text{-prof} = \lambda \ p' \ v. \ (if \ (v \in V) \ then \ p' \ v \ else \ \{\})$
have $\forall \ p'. \text{finite-profile } V \ A \ p' \longrightarrow$
 $\text{finite-profile } V \ A \ (?finite\text{-prof } p')$
unfolding *If-def profile-def*
by *simp*
moreover **have** $\forall \ p'. (\forall \ v. v \notin V \longrightarrow ?finite\text{-prof } p' \ v = \{\})$
by *simp*
ultimately **have**
 $\forall \ (A', \ V', \ p') \in \{(A', \ V', \ p'). A' = A \wedge V' = V \wedge \text{finite-profile } V \ A \ p'\}.$
 $(A', \ V', \ ?finite\text{-prof } p') \in \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p'\}$
by *force*
moreover **have**
 $\forall \ p'. d \ (A, \ V, \ p) \ (A, \ V, \ p') = d \ (A, \ V, \ p) \ (A, \ V, \ ?finite\text{-prof } p')$
using *irr-non-V*
unfolding *voters-determine-distance-def*
by *simp*
ultimately **have**
 $\forall \ (A', \ V', \ p') \in \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p'\}.$
 $(\exists \ (X, \ Y, \ z) \in \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p' \wedge (\forall \ v. v \notin V \longrightarrow p' \ v = \{\})\}).$
 $d \ (A, \ V, \ p) \ (A', \ V', \ p') = d \ (A, \ V, \ p) \ (X, \ Y, \ z))$
by *fastforce*
hence
 $\forall \ (A', \ V', \ p')$
 $\in \{(A', \ V', \ p'). A' = A \wedge V' = V \wedge \text{finite-profile } V \ A \ p'\}.$
 $d \ (A, \ V, \ p) \ (A', \ V', \ p') \in$
 $d \ (A, \ V, \ p) \ ' \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p' \wedge (\forall \ v. v \notin V \longrightarrow p' \ v = \{\})\}$
by *fastforce*
hence *subset-dist-restrict-non-voters*:
 $d \ (A, \ V, \ p) \ ' \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p'\}$
 $\subseteq d \ (A, \ V, \ p) \ ' \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p' \wedge (\forall \ v. v \notin V \longrightarrow p' \ v = \{\})\}$
by *fastforce*
have $\forall \ (A', \ V', \ p') \in \{(A, \ V, \ p') \mid p'. \text{finite-profile } V \ A \ p' \wedge (\forall \ v. v \notin V \longrightarrow p' \ v = \{\})\}.$
 $(\forall \ v \in V. \text{linear-order-on } A \ (p' \ v))$

$\wedge (\forall v. v \notin V \longrightarrow p' v = \{\})$

using *fin*
unfolding *profile-def*
by *simp*
hence *subset-lin-ord*:
 $\{(A, V, p') \mid p'. \text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$
 $\subseteq \{(A, V, p') \mid p'. p' \in \{p'.$
 $(\forall v \in V. \text{linear-order-on } A (p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}\}$
by *blast*
have $\{p'. (\forall v \in V. \text{linear-order-on } A (p' v))$
 $\wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$
 $\subseteq \text{profiles } V A \cap \{p. \forall v. v \notin V \longrightarrow p v = \{\}\}$
using *lin-order-pl-α fin*
by *fastforce*
moreover **have** *finite* $(\text{profiles } V A \cap \{p. \forall v. v \notin V \longrightarrow p v = \{\}\})$
using *fin fin-all-profs*
by *blast*
ultimately **have**
finite $\{p'. (\forall v \in V.$
 $\text{linear-order-on } A (p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$
using *rev-finite-subset*
by *blast*
hence *finite* $\{(A, V, p') \mid p'. p' \in \{p'.$
 $(\forall v \in V. \text{linear-order-on } A (p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}\}$
by *simp*
hence *finite* $\{(A, V, p') \mid p'.$
 $\text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$
using *subset-lin-ord rev-finite-subset*
by *simp*
hence *finite* $(d (A, V, p) \text{ ‘ } \{(A, V, p') \mid p'.$
 $\text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\})$
by *simp*
hence *finite* $(d (A, V, p) \text{ ‘ } \{(A, V, p') \mid p'. \text{finite-profile } V A p'\})$
using *subset-dist-restrict-non-voters rev-finite-subset*
by *simp*
hence *finite* $(d (A, V, p) \text{ ‘ } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V))$
using *subset-dist- $\mathcal{K}_{\mathcal{E}}$ -std rev-finite-subset*
by *blast*
moreover **have** $d (A, V, p) \text{ ‘ } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V) \neq \{\}$
using *False*
by *simp*
ultimately **have**
 $\text{Inf } (d (A, V, p) \text{ ‘ } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V)) =$
 $\text{Min } (d (A, V, p) \text{ ‘ } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V))$
using *Min-Inf False*
by *metis*
also **have** $\dots = \text{score-std } d K (A, V, p) w$
using *False*
by *simp*

```

also have  $\text{Inf } (d \ (A, V, p) \ '(\mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ A \ V)) =$ 
 $\text{Inf } (d \ (A, V, p) \ '(\mathcal{K}_{\mathcal{E}} \ K \ w \cap$ 
 $\text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\}))$ 
using eq-intersect
by simp
ultimately show ?thesis
by simp
qed
finally show  $\text{score } d \ K \ (A, V, p) \ w = \text{score-std } d \ K \ (A, V, p) \ w$ 
by simp
qed

lemma (in result) anonymous-distance-and-consensus-imp-rule-anonymity:
fixes
 $d :: ('a, 'v) \text{ Election Distance}$  and
 $K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
assumes
 $d\text{-anon: distance-anonymity } d$  and
 $K\text{-anon: consensus-rule-anonymity } K$ 
shows anonymity ( $\text{distance-}\mathcal{R} \ d \ K$ )
proof (unfold anonymity-def Let-def, safe)
show electoral-module ( $\text{distance-}\mathcal{R} \ d \ K$ )
using  $\mathcal{R}$ -sound
by metis
next
fix
 $A \ A' :: 'a \text{ set}$  and
 $V \ V' :: 'v \text{ set}$  and
 $p \ q :: ('a, 'v) \text{ Profile}$  and
 $\pi :: 'v \Rightarrow 'v$ 
assume
 $\text{bijective: bij } \pi$  and
 $\text{renamed: rename } \pi \ (A, V, p) = (A', V', q)$ 
hence eq-univ: limit  $A \ \text{UNIV} = \text{limit } A' \ \text{UNIV}$ 
by simp
have dist-rename-inv:
 $\forall E :: ('a, 'v) \text{ Election. } d \ (A, V, p) \ E = d \ (A', V', q) \ (\text{rename } \pi \ E)$ 
using d-anon bijective renamed surj-pair
unfolding distance-anonymity-def
by metis
hence  $\forall S :: ('a, 'v) \text{ Election set.}$ 
 $(d \ (A, V, p) \ 'S) \subseteq (d \ (A', V', q) \ '(\text{rename } \pi \ 'S))$ 
by blast
moreover have
 $\forall S :: ('a, 'v) \text{ Election set.}$ 
 $((d \ (A', V', q) \ '(\text{rename } \pi \ 'S)) \subseteq (d \ (A, V, p) \ 'S))$ 
proof (clarify)
fix
 $S :: ('a, 'v) \text{ Election set}$  and

```

$X \ X' :: 'a \text{ set}$ **and**
 $Y \ Y' :: 'v \text{ set}$ **and**
 $z \ z' :: ('a, 'v) \text{ Profile}$
assume $(X', Y', z') = \text{rename } \pi \ (X, Y, z)$
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 $d \ (A', V', q) \ (X', Y', z') \in d \ (A, V, p) \text{ ``} S$
by *simp*
qed
ultimately have *eq-range*:
 $\forall \ S :: ('a, 'v) \text{ Election set.}$
 $(d \ (A, V, p) \text{ ``} S) = (d \ (A', V', q) \text{ ``} (\text{rename } \pi \text{ ``} S))$
by *blast*
have $\forall \ w. \text{rename } \pi \text{ ``} (\mathcal{K}_{\mathcal{E}} \ K \ w) \subseteq (\mathcal{K}_{\mathcal{E}} \ K \ w)$
proof (*clarify*)
fix
 $w :: 'r$ **and**
 $A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$
assume $(A, V, p) \in \mathcal{K}_{\mathcal{E}} \ K \ w$
hence *cons*:
 $(\text{consensus-}\mathcal{K} \ K) \ (A, V, p) \wedge \text{finite-profile } V \ A \ p$
 $\wedge \text{elect } (\text{rule-}\mathcal{K} \ K) \ V \ A \ p = \{w\}$
by *simp*
moreover assume *renamed*: $(A', V', p') = \text{rename } \pi \ (A, V, p)$
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*:
 $\text{consensus-}\mathcal{K} \ K \ (A', V', p') \wedge (\text{rule-}\mathcal{K} \ K \ V \ A \ p = \text{rule-}\mathcal{K} \ K \ V' \ A' \ p')$
using *K-anon renamed bijective cons*
unfolding *consensus-rule-anonymity-def Let-def*
by *simp*
ultimately show $(A', V', p') \in \mathcal{K}_{\mathcal{E}} \ K \ w$
using *cons*
by *simp*
qed
moreover have $\forall \ w. (\mathcal{K}_{\mathcal{E}} \ K \ w) \subseteq \text{rename } \pi \text{ ``} (\mathcal{K}_{\mathcal{E}} \ K \ w)$
proof (*clarify*)
fix
 $w :: 'r$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assume $(A, V, p) \in \mathcal{K}_{\mathcal{E}} \ K \ w$

hence *cons*:
 $(\text{consensus-}\mathcal{K} \ K) \ (A, V, p) \wedge \text{finite-profile } V \ A \ p$
 $\wedge \text{elect } (\text{rule-}\mathcal{K} \ K) \ V \ A \ p = \{w\}$
by *simp*
let $?inv = \text{rename } (\text{the-inv } \pi) \ (A, V, p)$
have $\text{inv-inv-id: the-inv } (\text{the-inv } \pi) = \pi$
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 $?inv = (A, ((\text{the-inv } \pi) \text{ ` } V), p \circ (\text{the-inv } (\text{the-inv } \pi)))$
by *simp*
moreover **have** $(p \circ (\text{the-inv } (\text{the-inv } \pi))) \circ (\text{the-inv } \pi) = p$
using *bijective inv-inv-id*
unfolding *bij-betw-def comp-def*
by *(simp add: f-the-inv-into-f)*
moreover **have** $\pi \text{ ` } (\text{the-inv } \pi) \text{ ` } V = V$
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**
 $\text{consensus-}\mathcal{K} \ K \ ?inv \wedge$
 $(\text{rule-}\mathcal{K} \ K \ V \ A \ p =$
 $\text{rule-}\mathcal{K} \ K \ (\text{fst } (\text{snd } ?inv)) \ (\text{fst } ?inv) \ (\text{snd } (\text{snd } ?inv)))$
using *K-anon renamed bijective cons*
unfolding *consensus-rule-anonymity-def Let-def*
by *simp*
moreover from this **have**
 $\text{elect } (\text{rule-}\mathcal{K} \ K) \ (\text{fst } (\text{snd } ?inv)) \ (\text{fst } ?inv) \ (\text{snd } (\text{snd } ?inv)) = \{w\}$
using *cons*
by *simp*
ultimately **have** $?inv \in \mathcal{K}_{\mathcal{E}} \ K \ w$
using *fin-preimg*
by *simp*
thus $(A, V, p) \in \text{rename } \pi \text{ ` } \mathcal{K}_{\mathcal{E}} \ K \ w$
using *preimg image-eqI*
by *metis*
qed
ultimately **have** $\forall w. (\mathcal{K}_{\mathcal{E}} \ K \ w) = \text{rename } \pi \text{ ` } (\mathcal{K}_{\mathcal{E}} \ K \ w)$


```

    by blast
  hence  $\forall w. \text{score } d \ K \ (A, V, p) \ w = \text{score } d \ K \ (A', V', q) \ w$ 
    using eq-range
    by simp
  hence  $\text{arg-min-set } (\text{score } d \ K \ (A, V, p)) \ (\text{limit } A \ UNIV) =$ 
     $\text{arg-min-set } (\text{score } d \ K \ (A', V', q)) \ (\text{limit } A' \ UNIV)$ 
    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  $\text{distance-}\mathcal{R} \ d \ K \ V \ A \ p = \text{distance-}\mathcal{R} \ d \ K \ V' \ A' \ q$ 
    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 \mathbb{R} to \mathbb{N} .

5.5.1 Common Rationalizations

```

fun swap- $\mathcal{R} :: ('a, 'v :: \text{linorder}, 'a \ \text{Result}) \ \text{Consensus-Class} \Rightarrow$ 
     $('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$  where
  swap- $\mathcal{R} \ K = \text{SCF-result.distance-}\mathcal{R} \ (\text{votewise-distance swap l-one}) \ K$ 

```

5.5.2 Theorems

```

lemma votewise-non-voters-irrelevant:
  fixes
     $d :: 'a \ \text{Vote Distance}$  and
     $N :: \text{Norm}$ 
  shows voters-determine-distance  $(\text{votewise-distance } d \ N)$ 
proof (unfold voters-determine-distance-def, clarify)
  fix
     $A \ A' :: 'a \ \text{set}$  and
     $V \ V' :: 'v :: \text{linorder set}$  and

```

```

  p p' q :: ('a, 'v) Profile
assume coincide:  $\forall v \in V. p v = q v$ 
have  $\forall i < \text{length } (\text{sorted-list-of-set } V). (\text{sorted-list-of-set } V)!i \in V$ 
  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  $(\text{to-list } V p) = (\text{to-list } V q)$ 
  using coincide length-map nth-equalityI to-list.simps
  by auto
thus  $\text{votewise-distance } d N (A, V, p) (A', V', p') =$ 
   $\text{votewise-distance } d N (A, V, q) (A', V', p') \wedge$ 
   $\text{votewise-distance } d N (A', V', p') (A, V, p) =$ 
   $\text{votewise-distance } d N (A', V', p') (A, V, q)$ 
  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 =  $(\lambda l1 l2. (\text{map2 } (\lambda q q'. \text{swap } (A, q) (A', q')) l1 l2))$ 
  have  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \longrightarrow$ 
     $(\forall l1 l2. l1 \neq [] \wedge l2 \neq [] \longrightarrow (\forall i < \text{length } (?l l1 l2). (?l l1 l2)!i = \infty))$ 
  by simp
  moreover have
     $V = V' \wedge V \neq \{\} \wedge \text{finite } V \longrightarrow (\text{to-list } V p) \neq [] \wedge (\text{to-list } V' p') \neq []$ 
  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  $\forall l. (\exists i < \text{length } l. l!i = \infty) \longrightarrow \text{l-one } l = \infty$ 
proof (safe)
  fix
    l :: ereal list and
    i :: nat
  assume
    i < length l and
    l ! i =  $\infty$ 
  hence  $(\sum j < \text{length } l. |l!j|) = \infty$ 
  using sum-Pinfy finite-lessThan lessThan-iff abs-ereal.simps
  by metis
  thus  $\text{l-one } l = \infty$ 
  by auto
qed
ultimately have  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V$ 

```

$\longrightarrow l\text{-one } (?l \ (to\text{-list } V \ p) \ (to\text{-list } V' \ p)) = \infty$
using *length-greater-0-conv map-is-Nil-conv zip-eq-Nil-iff*
by *metis*
hence $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \longrightarrow$
 $\text{votewise-distance swap } l\text{-one } (A, V, p) (A', V', p') = \infty$
by *force*
moreover have
 $V \neq V'$
 $\longrightarrow \text{votewise-distance swap } l\text{-one } (A, V, p) (A', V', p') = \infty$
by *simp*
moreover have
 $A \neq A' \wedge V = \{\}$
 $\longrightarrow \text{votewise-distance swap } l\text{-one } (A, V, p) (A', V', p') = \infty$
by *simp*
moreover have
 $(A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V)$
 $\vee \text{infinite } V \vee (A \neq A' \wedge V = \{\}) \vee V \neq V'$
using *assms*
by *blast*
ultimately show $\text{votewise-distance swap } l\text{-one } (A, V, p) (A', V', p') = \infty$
by *fastforce*
qed

5.5.3 Equivalence Lemmas

type-synonym $('a, 'v) \text{ score-type} = ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election} \Rightarrow 'a \Rightarrow \text{ereal}$

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

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

type-synonym $('a, 'v) \text{ dist-type} = ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

lemma *equal-score-swap*: $(\text{score} :: ('a, 'v :: \text{linorder}) \text{ score-type})$
 $(\text{votewise-distance swap } l\text{-one}) = \text{score-std } (\text{votewise-distance swap } l\text{-one})$
using *votewise-non-voters-irrelevant swap-standard*
 $\text{SCF-result.standard-distance-imp-equal-score}$
by *fast*

lemma *swap- \mathcal{R} -code*[code]: $\text{swap-}\mathcal{R} =$
 $(\text{SCF-result.distance-}\mathcal{R}\text{-std} :: ('a, 'v :: \text{linorder}) \text{ dist-rat-std-type})$
 $(\text{votewise-distance swap } l\text{-one})$

unfolding $\text{swap-}\mathcal{R}.\text{sims } \text{SCF-result.distance-}\mathcal{R}.\text{sims } \text{SCF-result.distance-}\mathcal{R}\text{-std}.\text{sims}$
 $\text{SCF-result.}\mathcal{R}_{\mathcal{W}}.\text{sims } \text{SCF-result.}\mathcal{R}_{\mathcal{W}}\text{-std}.\text{sims } \text{equal-score-swap}$
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* ' $\{y \in (f \text{' } A). \forall z \in f \text{' } A. y \leq z\}$)
proof (*safe*)
fix *x* :: *'a*
assume *arg-min*: *x* \in *arg-min-set f A*
hence *is-arg-min f* ($\lambda a. a \in A$) *x*
by *simp*
hence $\forall x' \in A. f x' \geq f x$
by (*simp add: is-arg-min-linorder*)
hence $\forall z \in f \text{' } A. f x \leq z$
by *blast*
moreover **have** *f x* \in *f* ' *A*
using *arg-min*
by (*simp add: is-arg-min-linorder*)
ultimately **have** *f x* \in $\{y \in f \text{' } A. \forall z \in f \text{' } A. y \leq z\}$
by *blast*
moreover **have** *x* \in *preimg f A* (*f x*)
using *arg-min*
by (*simp add: is-arg-min-linorder*)

ultimately show $x \in \bigcup (preimg\ f\ A\ \{y \in (f\ 'A). \forall\ z \in f\ 'A. y \leq z\})$
by *blast*
next
fix $x\ x'\ b :: 'a$
assume
same-img: $x \in preimg\ f\ A\ (f\ x')$ **and**
min: $\forall\ z \in f\ 'A. f\ x' \leq z$
hence $f\ x = f\ x'$
by *simp*
hence $\forall\ z \in f\ 'A. f\ x \leq z$
using *min*
by *simp*
moreover have $x \in A$
using *same-img*
by *simp*
ultimately show $x \in arg-min-set\ f\ A$
by (*simp add: is-arg-min-linorder*)
qed

Equivariance

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

lemma *restr-induced-rel*:

fixes
 $A :: 'a\ set$ **and**
 $B\ B' :: 'b\ set$ **and**
 $\varphi :: ('a, 'b)\ binary-fun$
assumes $B' \subseteq B$
shows *Restr* (*action-induced-rel* $A\ B\ \varphi$) $B' = action-induced-rel\ A\ B'\ \varphi$
using *assms*
by *auto*

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**
 $\varphi :: ('c, 'a)\ binary-fun$ **and**
 $\psi :: ('c, 'b)\ binary-fun$
defines *equivar-prop-set-valued* \equiv
action-induced-equivariance (*carrier* G) $X\ \varphi\ (set-action\ \psi)$
assumes
action- φ : *group-action* $G\ X\ \varphi$ **and**
group-action-res: *group-action* $G\ UNIV\ \psi$ **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_D* d (*carrier* G) X φ **and**
equivar-f:
is-symmetry f (*action-induced-equivariance* (*carrier* G) *domain_f* φ ψ)
shows *is-symmetry* ($\lambda x. \text{minimizer } f \text{ domain}_f d (\text{well-formed-img } x) x$) *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: $g \in \text{carrier } G$ **and**
x-in-X: $x \in X$
hence *img-X:* $\varphi \ g \ x \in X$
using *action- φ group-action.element-image*
by *metis*
let $?x' = \varphi \ g \ x$
let $?c = \text{closest-preimg-distance } f \text{ domain}_f d \ x$ **and**
 $?c' = \text{closest-preimg-distance } f \text{ domain}_f d \ ?x'$
have $\forall y. \text{preimg } f \text{ domain}_f y \subseteq X$
using *dom-in-X*
by *fastforce*
hence *invar-dist-img:*
 $\forall y. d \ x \ ' (\text{preimg } f \text{ domain}_f y) = d \ ?x' \ ' (\varphi \ g \ ' (\text{preimg } f \text{ domain}_f y))$
using *x-in-X group-elem invar-dist-image invar-d action- φ*
by *metis*
have $\forall y. \text{preimg } f \text{ domain}_f (\psi \ g \ y) = (\varphi \ g) \ ' (\text{preimg } f \text{ domain}_f y)$
using *group-action-equivar-f-imp-equivar-preimg[of G X φ ψ domain_f f g]*
assms group-elem
by *blast*
hence $\forall y. d \ ?x' \ ' \text{preimg } f \text{ domain}_f (\psi \ g \ y) =$
 $d \ ?x' \ ' (\varphi \ g) \ ' (\text{preimg } f \text{ domain}_f y)$
by *presburger*
hence $\forall y. \text{Inf } (d \ ?x' \ ' \text{preimg } f \text{ domain}_f (\psi \ g \ y)) =$
 $\text{Inf } (d \ x \ ' \text{preimg } f \text{ domain}_f y)$
using *invar-dist-img*
by *metis*
hence $\forall y. \text{distance-infimum } d (\text{preimg } f \text{ domain}_f (\psi \ g \ y)) \ ?x' =$
 $\text{distance-infimum } d (\text{preimg } f \text{ domain}_f y) \ x$
by *simp*
hence $\forall y. \text{closest-preimg-distance } f \text{ domain}_f d \ ?x' (\psi \ g \ y) =$
 $\text{closest-preimg-distance } f \text{ domain}_f d \ x \ y$
by *simp*
hence *comp:*
 $\text{closest-preimg-distance } f \text{ domain}_f d \ x =$
 $(\text{closest-preimg-distance } f \text{ domain}_f d \ ?x') \circ (\psi \ g)$
by *auto*

hence $\forall Y \alpha. \text{preimg } ?c' (\psi g \text{ ' } Y) \alpha = \psi g \text{ ' } \text{preimg } ?c Y \alpha$
 using *preimg-comp*
 by *auto*
 hence $\forall Y A. \{\text{preimg } ?c' (\psi g \text{ ' } Y) \alpha \mid \alpha. \alpha \in A\} =$
 $\{\psi g \text{ ' } \text{preimg } ?c Y \alpha \mid \alpha. \alpha \in A\}$
 by *simp*
 moreover have
 $\forall Y A. \{\psi g \text{ ' } \text{preimg } ?c Y \alpha \mid \alpha. \alpha \in A\} = \{\psi g \text{ ' } \beta \mid \beta. \beta \in \text{preimg } ?c Y \text{ ' } A\}$
 by *blast*
 moreover have
 $\forall Y A. \text{preimg } ?c' (\psi g \text{ ' } Y) \text{ ' } A = \{\text{preimg } ?c' (\psi g \text{ ' } Y) \alpha \mid \alpha. \alpha \in A\}$
 by *blast*
 ultimately have
 $\forall Y A. \text{preimg } ?c' (\psi g \text{ ' } Y) \text{ ' } A = \{\psi g \text{ ' } \alpha \mid \alpha. \alpha \in \text{preimg } ?c Y \text{ ' } A\}$
 by *simp*
 hence $\forall Y A. \bigcup (\text{preimg } ?c' (\psi g \text{ ' } Y) \text{ ' } A) =$
 $\bigcup \{\psi g \text{ ' } \alpha \mid \alpha. \alpha \in \text{preimg } ?c Y \text{ ' } A\}$
 by *simp*
 moreover have
 $\forall Y A. \bigcup \{\psi g \text{ ' } \alpha \mid \alpha. \alpha \in \text{preimg } ?c Y \text{ ' } A\} = \psi g \text{ ' } \bigcup (\text{preimg } ?c Y \text{ ' } A)$
 by *blast*
 ultimately have *eq-preimg-unions*:
 $\forall Y A. \bigcup (\text{preimg } ?c' (\psi g \text{ ' } Y) \text{ ' } A) = \psi g \text{ ' } \bigcup (\text{preimg } ?c Y \text{ ' } A)$
 by *simp*
 have $\forall Y. ?c' \text{ ' } \psi g \text{ ' } Y = ?c \text{ ' } Y$
 using *comp*
 unfolding *image-comp*
 by *simp*
 hence $\forall Y. \{\alpha \in ?c \text{ ' } Y. \forall \beta \in ?c \text{ ' } Y. \alpha \leq \beta\} =$
 $\{\alpha \in ?c' \text{ ' } \psi g \text{ ' } Y. \forall \beta \in ?c' \text{ ' } \psi g \text{ ' } Y. \alpha \leq \beta\}$
 by *simp*
 hence $\forall Y. \text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f d \text{ ' } x') (\psi g \text{ ' } Y) =$
 $(\psi g) \text{ ' } (\text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f d x) Y)$
 using *rewrite-arg-min-set[of ?c'] rewrite-arg-min-set[of ?c] eq-preimg-unions*
 by *presburger*
 moreover have *well-formed-img* $(\varphi g x) = \psi g \text{ ' } \text{well-formed-img } x$
 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
 $\text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f d (\varphi g x))$
 $(\text{well-formed-img } (\varphi g x)) =$
 $\psi g \text{ ' } \text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f d x)$
 $(\text{well-formed-img } x)$
 by *presburger*
 qed

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist:*

fixes
 $f :: 'a \Rightarrow 'b$ **and**
 $domain_f :: 'a \text{ set}$ **and**
 $d :: 'a \text{ Distance}$ **and**
 $rel :: 'a \text{ rel}$
assumes
 $r\text{-refl}: \text{reflp-on}' \ domain_f \ (Restr p \ rel \ domain_f)$ **and**
 $tot\text{-invar-d}: total\text{-invariance}_{\mathcal{D}} \ d \ rel$
shows $is\text{-symmetry} \ (closest\text{-preimg-distance} \ f \ domain_f \ d) \ (Invariance \ rel)$
proof $(unfold \ is\text{-symmetry.simps}, \ intro \ allI \ impI \ ext)$
fix
 $a \ b :: 'a$ **and**
 $y :: 'b$
assume $rel: (a, b) \in rel$
have $\forall \ c \in domain_f. (c, c) \in rel$
using $r\text{-refl}$
unfolding $reflp\text{-on}'\text{-def} \ reflp\text{-on-def}$
by $simp$
hence $\forall \ c \in domain_f. d \ a \ c = d \ b \ c$
using $rel \ tot\text{-invar-d}$
unfolding $rewrite\text{-total-invariance}_{\mathcal{D}}$
by $blast$
thus $closest\text{-preimg-distance} \ f \ domain_f \ d \ a \ y =$
 $closest\text{-preimg-distance} \ f \ domain_f \ d \ b \ y$
by $simp$
qed

lemma *refl-rel-and-tot-invar-dist-imp-invar-minimizer:*

fixes
 $f :: 'a \Rightarrow 'b$ **and**
 $domain_f :: 'a \text{ set}$ **and**
 $d :: 'a \text{ Distance}$ **and**
 $rel :: 'a \text{ rel}$ **and**
 $img :: 'b \text{ set}$
assumes
 $r\text{-refl}: \text{reflp-on}' \ domain_f \ (Restr p \ rel \ domain_f)$ **and**
 $tot\text{-invar-d}: total\text{-invariance}_{\mathcal{D}} \ d \ rel$
shows $is\text{-symmetry} \ (minimizer \ f \ domain_f \ d \ img) \ (Invariance \ rel)$
proof $-$
have $is\text{-symmetry} \ (closest\text{-preimg-distance} \ f \ domain_f \ d) \ (Invariance \ rel)$
using $r\text{-refl} \ tot\text{-invar-d} \ closest\text{-dist-invar-under-refl-rel-and-tot-invar-dist}$
by $metis$
thus $?thesis$
by $simp$
qed

theorem *group-act-invar-dist-and-invar-f-imp-invar-minimizer:*


```

fixes
   $f :: 'a \Rightarrow 'b$  and
   $\text{domain}_f A :: 'a \text{ set}$  and
   $d :: 'a \text{ Distance}$  and
   $\text{img} :: 'b \text{ set}$  and
   $G :: 'c \text{ monoid}$  and
   $\varphi :: ('c, 'a) \text{ binary-fun}$ 
defines
   $\text{rel} \equiv \text{action-induced-rel } (\text{carrier } G) A \varphi$  and
   $\text{rel}' \equiv \text{action-induced-rel } (\text{carrier } G) \text{domain}_f \varphi$ 
assumes
   $\text{action-}\varphi$ :  $\text{group-action } G A \varphi$  and
   $\text{domain}_f \subseteq A$  and
   $\text{closed-domain}$ :  $\text{closed-restricted-rel } \text{rel } A \text{domain}_f$  and
   $\text{invar-d}$ :  $\text{invariance}_{\mathcal{D}} d (\text{carrier } G) A \varphi$  and
   $\text{invar-f}$ :  $\text{is-symmetry } f (\text{Invariance } \text{rel}')$ 
shows  $\text{is-symmetry } (\text{minimizer } f \text{domain}_f d \text{img}) (\text{Invariance } \text{rel})$ 
proof –
let
   $? \psi = \lambda g. \text{id}$  and
   $? \text{img} = \lambda x. \text{img}$ 
have  $\text{is-symmetry } f (\text{action-induced-equivariance } (\text{carrier } G) \text{domain}_f \varphi ? \psi)$ 
  using  $\text{invar-f rewrite-invar-as-equivar}$ 
  unfolding  $\text{rel'-def}$ 
  by  $\text{blast}$ 
moreover have  $\text{group-action } G \text{UNIV } ? \psi$ 
  using  $\text{const-id-is-group-action action-}\varphi$ 
  unfolding  $\text{group-action-def group-hom-def}$ 
  by  $\text{blast}$ 
moreover have
   $\text{is-symmetry } ? \text{img } (\text{action-induced-equivariance } (\text{carrier } G) A \varphi (\text{set-action } ? \psi))$ 
  unfolding  $\text{action-induced-equivariance-def}$ 
  by  $\text{fastforce}$ 
ultimately have
   $\text{is-symmetry } (\lambda x. \text{minimizer } f \text{domain}_f d (? \text{img } x) x)$ 
   $(\text{action-induced-equivariance } (\text{carrier } G) A \varphi (\text{set-action } ? \psi))$ 
  using  $\text{assms}$ 
   $\text{group-action-invar-dist-and-equivar-f-imp-equivar-minimizer[of } G A \varphi ? \psi \text{domain}_f ? \text{img } d f]$ 
  by  $\text{blast}$ 
hence  $\text{is-symmetry } (\text{minimizer } f \text{domain}_f d \text{img})$ 
   $(\text{action-induced-equivariance } (\text{carrier } G) A \varphi (\text{set-action } ? \psi))$ 
  by  $\text{blast}$ 
thus  $? \text{thesis}$ 
  unfolding  $\text{rel-def set-action.simps}$ 
  using  $\text{rewrite-invar-as-equivar image-id}$ 
  by  $\text{metis}$ 
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 $\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{w\} = \mathcal{K}_{\mathcal{E}} \ C \ w$

proof –

have $\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{w\} =$
 $\{E \in \text{elections-}\mathcal{K} \ C. (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) \ E = \{w\}\}$

by *simp*

also have $\{E \in \text{elections-}\mathcal{K} \ C. (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) \ E = \{w\}\} =$
 $\{E \in \text{elections-}\mathcal{K} \ C.$
 $\text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$

by *simp*

also have $\{E \in \text{elections-}\mathcal{K} \ C.$
 $\text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\} =$
 $\text{elections-}\mathcal{K} \ C$
 $\cap \{E. \text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$

by *blast*

also have $\text{elections-}\mathcal{K} \ C$
 $\cap \{E. \text{elect } (\text{rule-}\mathcal{K} \ C)$
 $(\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\} =$
 $\mathcal{K}_{\mathcal{E}} \ C \ w$

proof

show $\text{elections-}\mathcal{K} \ C$
 $\cap \{E. \text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$
 $\subseteq \mathcal{K}_{\mathcal{E}} \ C \ w$

unfolding $\mathcal{K}_{\mathcal{E}}.\text{simps}$

by *force*

next

have $\forall \ E \in \mathcal{K}_{\mathcal{E}} \ C \ w. E \in \{E. \text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E)$
 $(\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$

unfolding $\mathcal{K}_{\mathcal{E}}.\text{simps}$

by *force*

hence $\forall \ E \in \mathcal{K}_{\mathcal{E}} \ C \ w.$

$E \in \text{elections-}\mathcal{K} \ C$
 $\cap \{E. \text{elect } (\text{rule-}\mathcal{K} \ C)$
 $(\text{voters-}\mathcal{E} \ E) \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$

by *simp*

thus $\mathcal{K}_{\mathcal{E}} \ C \ w \subseteq \text{elections-}\mathcal{K} \ C \cap \{E. \text{elect } (\text{rule-}\mathcal{K} \ C) \ (\text{voters-}\mathcal{E} \ E)$
 $(\text{alternatives-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E) = \{w\}\}$

by *blast*

qed

finally show $\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{w\} = \mathcal{K}_{\mathcal{E}} \ C \ w$

by *simp*

qed

lemma *score-is-closest-preimg-dist:*

fixes

$d :: ('a, 'v)$ *Election Distance* **and**

$C :: ('a, 'v, 'r)$ *Result* *Consensus-Class* **and**

$E :: ('a, 'v)$ *Election* **and**

$w :: 'r$

shows $\text{score } d \ C \ E \ w =$

$\text{closest-preimg-distance } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{w\}$

proof $-$

have $\text{score } d \ C \ E \ w = \text{Inf } (d \ E \ ' (\mathcal{K}_{\mathcal{E}} \ C \ w))$

by *simp*

also have $\mathcal{K}_{\mathcal{E}} \ C \ w = \text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{w\}$

using *$\mathcal{K}_{\mathcal{E}}$ -is-preimg*

by *metis*

also have

$\text{Inf } (d \ E \ ' (\text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{w\})) =$

$\text{closest-preimg-distance } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{w\}$

by *simp*

finally show *?thesis*

by *simp*

qed

lemma (*in result*) *$\mathcal{R}_{\mathcal{W}}$ -is-minimizer:*

fixes

$d :: ('a, 'v)$ *Election Distance* **and**

$C :: ('a, 'v, 'r)$ *Result* *Consensus-Class*

shows $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) =$

$(\lambda E. \bigcup (\text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d$
 $\quad (\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})) \ E))$

proof

fix $E :: ('a, 'v)$ *Election*

let $?min = (\text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d$

$\quad (\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})) \ E)$

have $?min =$

arg-min-set

$(\text{closest-preimg-distance } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E)$

$(\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}))$

by *simp*

also have

$\dots = \text{singleton-set-system}$

$(\text{arg-min-set } (\text{score } d \ C \ E) (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}))$

proof (*safe*)

fix $R :: 'r$ *set*

assume

$\text{min: } R \in \text{arg-min-set}$

$(\text{closest-preimg-distance}$

$(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E)$

$(\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}))$

hence $R \in \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$

```

    using arg-min-subset subsetD
    by (metis (no-types, lifting))
  then obtain  $r :: 'r$  where
    res-singleton:  $R = \{r\}$  and
    r-in-lim-set:  $r \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ 
    by auto
  have  $\nexists R'. R' \in \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$ 
     $\wedge$  closest-preimg-distance
      (elect- $r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ R'$ 
       $<$  closest-preimg-distance
      (elect- $r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ R$ 
    using min arg-min-set.simps is-arg-min-def CollectD
    by (metis (mono-tags, lifting))
  hence  $\nexists r'. r' \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ 
     $\wedge$  closest-preimg-distance
      (elect- $r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{r'\}$ 
       $<$  closest-preimg-distance
      (elect- $r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{r\}$ 
    using res-singleton
    by auto
  hence
     $\nexists r'. r' \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ 
       $\wedge$  score  $d \ C \ E \ r' < \text{score } d \ C \ E \ r$ 
    using score-is-closest-preimg-dist
    by metis
  hence  $r \in \text{arg-min-set } (\text{score } d \ C \ E) (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$ 
    using r-in-lim-set arg-min-set.simps is-arg-min-def CollectI
    by metis
  thus  $R \in \text{singleton-set-system}$ 
    (arg-min-set (score  $d \ C \ E$ ) (limit (alternatives- $\mathcal{E}$   $E$ ) UNIV))
    using res-singleton
    by simp
next
fix  $R :: 'r \text{ set}$ 
assume
   $R \in \text{singleton-set-system}$ 
    (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:
     $r \in \text{arg-min-set } (\text{score } d \ C \ E) (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$ 
  by auto
hence  $\nexists r'. r' \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ 
   $\wedge$  score  $d \ C \ E \ r' < \text{score } d \ C \ E \ r$ 
  using CollectD arg-min-set.simps is-arg-min-def
  by metis
hence
   $\nexists r'. r' \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ 
     $\wedge$  closest-preimg-distance

```

$(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{r'\}$
 $< \text{closest-preimg-distance}$
 $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ \{r\}$
using *score-is-closest-preimg-dist*
by *metis*
moreover have
 $\forall R' \in \text{singleton-set-system} (\text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}).$
 $\exists r' \in \text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}. R' = \{r'\}$
by *auto*
ultimately have
 $\nexists R'. R' \in \text{singleton-set-system} (\text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
 $\wedge \text{closest-preimg-distance}$
 $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ R'$
 $< \text{closest-preimg-distance}$
 $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ R$
using *res-singleton*
by *auto*
moreover have
 $R \in \text{singleton-set-system} (\text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
using *r-min-lim-set res-singleton arg-min-subset*
by *fastforce*
ultimately show
 $R \in \text{arg-min-set}$
 $(\text{closest-preimg-distance}$
 $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E)$
 $(\text{singleton-set-system} (\text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})))$
using *arg-min-set.simps is-arg-min-def CollectI*
by *(metis (mono-tags, lifting))*
qed
also have
 $(\text{arg-min-set} (\text{score} \ d \ C \ E) (\text{limit} (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})) =$
 $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E$
by *simp*
finally have $\bigcup \ ?min = \bigcup (\text{singleton-set-system} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E))$
by *presburger*
thus $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E = \bigcup \ ?min$
using *un-left-inv-singleton-set-system*
by *auto*
qed

Invariance

theorem (*in result*) *tot-invar-dist-imp-invar-dr-rule:*

fixes

$d :: ('a, 'v) \text{ Election Distance}$ **and**

$C :: ('a, 'v, 'r) \text{ Result Consensus-Class}$ **and**

$\text{rel} :: ('a, 'v) \text{ Election rel}$

assumes

$r\text{-refl}: \text{reflp-on}' (\text{elections-}\mathcal{K} \ C) (\text{Restrp rel} (\text{elections-}\mathcal{K} \ C))$ **and**

tot-invar-d: total-invariance_D d rel **and**
invar-res:
is-symmetry ($\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$) (*Invariance rel*)
shows *is-symmetry* ($\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C)$) (*Invariance rel*)
proof –
let *?min* =
 $\lambda E. \bigcup \circ (\text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)) \ (\text{elections-}\mathcal{K} \ C) \ d$
 $\quad (\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)))$
have $\forall E. \text{is-symmetry } (?min \ E) \ (\text{Invariance rel})$
using *r-refl tot-invar-d invar-comp*
 $\text{refl-rel-and-tot-invar-dist-imp-invar-minimizer[of}$
 $\text{elections-}\mathcal{K} \ C \ \text{rel} \ d \ \text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)]$
by *blast*
moreover have *is-symmetry ?min (Invariance rel)*
using *invar-res*
by *auto*
ultimately have *is-symmetry* ($\lambda E. ?min \ E \ E$) (*Invariance rel*)
using *invar-parameterized-fun[of ?min rel]*
by *blast*
also have ($\lambda E. ?min \ E \ E$) = $\text{fun}_{\mathcal{E}} \ (\mathcal{R}_{\mathcal{W}} \ d \ C)$
using *$\mathcal{R}_{\mathcal{W}}$ -is-minimizer*
unfolding *comp-def fun_E.sims*
by *metis*
finally have *invar- $\mathcal{R}_{\mathcal{W}}$: is-symmetry* ($\text{fun}_{\mathcal{E}} \ (\mathcal{R}_{\mathcal{W}} \ d \ C)$) (*Invariance rel*)
by *simp*
hence
is-symmetry ($\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV - \text{fun}_{\mathcal{E}} \ (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E$)
 $\quad (\text{Invariance rel})$
using *invar-res*
by *fastforce*
thus *is-symmetry* ($\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C)$) (*Invariance rel*)
using *invar- $\mathcal{R}_{\mathcal{W}}$*
by *auto*
qed

theorem (*in result*) *invar-dist-cons-imp-invar-dr-rule:*

fixes

$d :: ('a, 'v) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $G :: 'b \text{ monoid}$ **and**
 $\varphi :: ('b, ('a, 'v) \text{ Election}) \text{ binary-fun}$ **and**
 $B :: ('a, 'v) \text{ Election set}$

defines

$\text{rel} \equiv \text{action-induced-rel } (\text{carrier } G) \ B \ \varphi$ **and**
 $\text{rel}' \equiv \text{action-induced-rel } (\text{carrier } G) \ (\text{elections-}\mathcal{K} \ C) \ \varphi$

assumes

action- φ : group-action $G \ B \ \varphi$ **and**
consensus-C-in-B: elections- $\mathcal{K} \ C \subseteq B$ **and**
closed-domain:

closed-restricted-rel $\text{rel } B \text{ (elections-}\mathcal{K} \text{ } C)$ **and**
invar-res:
is-symmetry $(\lambda E. \text{limit (alternatives-}\mathcal{E} \text{ } E) \text{ UNIV}) \text{ (Invariance rel)}$ **and**
invar-d: *invariance* _{\mathcal{D}} $d \text{ (carrier } G) B \varphi$ **and**
invar-C-winners: *is-symmetry* $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{ (rule-}\mathcal{K} \text{ } C)) \text{ (Invariance rel')}$
shows *is-symmetry* $(\text{fun}_{\mathcal{E}} \text{ (distance-}\mathcal{R} \text{ } d \text{ } C)) \text{ (Invariance rel)}$
proof –
let $?min =$
 $\lambda E. \bigcup \circ (\text{minimizer (elect-r} \circ \text{fun}_{\mathcal{E}} \text{ (rule-}\mathcal{K} \text{ } C)) \text{ (elections-}\mathcal{K} \text{ } C) \text{ } d$
 $\text{ (singleton-set-system (limit (alternatives-}\mathcal{E} \text{ } E) \text{ UNIV})))$
have $\forall E. \text{is-symmetry } (?min \text{ } E) \text{ (Invariance rel)}$
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* $?min \text{ (Invariance rel)}$
using *invar-res*
by *auto*
ultimately have
is-symmetry $(\lambda E. ?min \text{ } E \text{ } E) \text{ (Invariance rel)}$
using *invar-parameterized-fun[of ?min -]*
by *blast*
also have $(\lambda E. ?min \text{ } E \text{ } E) = \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \text{ } d \text{ } C)$
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 $(\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \text{ } d \text{ } C)) \text{ (Invariance rel)}$
by *simp*
hence *is-symmetry* $(\lambda E. \text{limit (alternatives-}\mathcal{E} \text{ } E) \text{ UNIV} -$
 $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \text{ } d \text{ } C) \text{ } E) \text{ (Invariance rel)}$
using *invar-res*
by *fastforce*
thus *is-symmetry* $(\text{fun}_{\mathcal{E}} \text{ (distance-}\mathcal{R} \text{ } d \text{ } C)) \text{ (Invariance rel)}$
using *invar- $\mathcal{R}_{\mathcal{W}}$*
by *simp*
qed

Equivariance

theorem (in *result*) *invar-dist-equivar-cons-imp-equivar-dr-rule*:

fixes

$d :: ('a, 'v) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $G :: 'b \text{ monoid}$ **and**
 $\varphi :: ('b, ('a, 'v) \text{ Election}) \text{ binary-fun}$ **and**
 $\psi :: ('b, 'r) \text{ binary-fun}$ **and**
 $B :: ('a, 'v) \text{ Election set}$

defines

$rel \equiv \text{action-induced-rel } (\text{carrier } G) \ B \ \varphi \text{ and}$
 $rel' \equiv \text{action-induced-rel } (\text{carrier } G) \ (\text{elections-}\mathcal{K} \ C) \ \varphi \text{ and}$
 $\text{equivar-prop} \equiv$
 $\quad \text{action-induced-equivariance } (\text{carrier } G) \ (\text{elections-}\mathcal{K} \ C)$
 $\quad \varphi \ (\text{set-action } \psi) \text{ and}$
 $\text{equivar-prop-global-set-valued} \equiv$
 $\quad \text{action-induced-equivariance } (\text{carrier } G) \ B \ \varphi \ (\text{set-action } \psi) \text{ and}$
 $\text{equivar-prop-global-result-valued} \equiv$
 $\quad \text{action-induced-equivariance } (\text{carrier } G) \ B \ \varphi \ (\text{result-action } \psi)$
assumes
 $\text{action-}\varphi$: $\text{group-action } G \ B \ \varphi \text{ and}$
 group-act-res : $\text{group-action } G \ \text{UNIV } \psi \text{ and}$
 cons-elect-set : $\text{elections-}\mathcal{K} \ C \subseteq B \text{ and}$
 closed-domain : $\text{closed-restricted-rel } rel \ B \ (\text{elections-}\mathcal{K} \ C) \text{ and}$
 equivar-res :
 $\quad \text{is-symmetry } (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
 $\quad \text{equivar-prop-global-set-valued and}$
 invar-d : $\text{invariance}_{\mathcal{D}} \ d \ (\text{carrier } G) \ B \ \varphi \text{ and}$
 equivar-C-winners : $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)) \ \text{equivar-prop}$
shows $\text{is-symmetry } (\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C)) \ \text{equivar-prop-global-result-valued}$
proof –
let $?min-E =$
 $\quad \lambda E. \text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)) \ (\text{elections-}\mathcal{K} \ C) \ d$
 $\quad (\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})) \ E$
let $?min =$
 $\quad \lambda E. \bigcup \circ (\text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)) \ (\text{elections-}\mathcal{K} \ C) \ d$
 $\quad (\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})))$
let $? \psi' = \text{set-action } (\text{set-action } \psi)$
let $? \text{equivar-prop-global-set-valued}' =$
 $\quad \text{action-induced-equivariance } (\text{carrier } G) \ B \ \varphi \ ? \psi'$
have $\forall E \ g. g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$
 $\quad \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}) =$
 $\quad \{\{r\} \mid r. r \in \text{limit } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}\}$
by *simp*
moreover have
 $\forall E \ g. g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$
 $\quad \text{limit } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV} =$
 $\quad \psi \ g \ ‘ (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
using $\text{equivar-res action-}\varphi \ \text{group-action.element-image}$
unfolding $\text{equivar-prop-global-set-valued-def action-induced-equivariance-def}$
by *fastforce*
ultimately have $\forall E \ g. g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$
 $\quad \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}) =$
 $\quad \{\{r\} \mid r. r \in \psi \ g \ ‘ (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})\}$
by *simp*
moreover have
 $\forall E \ g. \{\{r\} \mid r. r \in \psi \ g \ ‘ (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})\} =$
 $\quad \{\psi \ g \ ‘ \{r\} \mid r. r \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}\}$
by *blast*

moreover have
 $\forall E g. \{\psi g \text{ ' } \{r\} \mid r. r \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV\} =$
 $\text{?}\psi' g \ \{\{r\} \mid r. r \in \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV\}$
unfolding *set-action.simps*
by *blast*
ultimately have
 $\text{is-symmetry } (\lambda E. \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV))$
 $\text{?equivar-prop-global-set-valued'}$
using *rewrite-equivariance*[*of*
 $\lambda E. \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$
 $\text{carrier } G \ B \ \varphi \ \text{?}\psi^\uparrow]$
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* $\text{?min-}E \ \text{?equivar-prop-global-set-valued'}$
using *action- φ invar-d cons-elect-set closed-domain equivar- C -winners*
group-action-invar-dist-and-equivar-f-imp-equivar-minimizer[*of*
 $G \ B \ \varphi \ \text{set-action } \psi \ \text{elections-}\mathcal{K} \ C$
 $\lambda E. \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$
 $d \ \text{elect-}r \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)]$
unfolding *rel'-def rel-def equivar-prop-def*
by *metis*
moreover have
is-symmetry
 $\bigcup \ (\text{action-induced-equivariance}$
 $(\text{carrier } G) \ UNIV \ \text{?}\psi' \ (\text{set-action } \psi))$
using *equivar-union-under-image-action*[*of* - ψ]
by *simp*
ultimately have *is-symmetry* $(\bigcup \circ \text{?min-}E) \ \text{equivar-prop-global-set-valued}$
unfolding *equivar-prop-global-set-valued-def*
using *equivar-ind-by-action-comp*[*of* - - $UNIV$]
by *simp*
moreover have $(\lambda E. \text{?min } E \ E) = \bigcup \circ \text{?min-}E$
unfolding *comp-def*
by *simp*
ultimately have
is-symmetry $(\lambda E. \text{?min } E \ E) \ \text{equivar-prop-global-set-valued}$
by *simp*
moreover have $(\lambda E. \text{?min } E \ E) = \text{fun}_{\mathcal{E}} \ (\mathcal{R}_{\mathcal{W}} \ d \ C)$
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 $(\text{fun}_{\mathcal{E}} \ (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ \text{equivar-prop-global-set-valued}$
by *simp*
moreover have $\forall g \in \text{carrier } G. \text{bij } (\psi \ g)$
using *group-act-res*

unfolding *bij-betw-def*
by (*simp add: group-action.inj-prop group-action.surj-prop*)
ultimately have
is-symmetry ($\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV} - \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E$)
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* ($\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C)$) *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'* *well-formed-elections* (*distance-}\mathcal{R} \ d \ C*)
proof –
have $\forall \pi. \forall E \in \text{elections-}\mathcal{K} \ C.$
 $\varphi\text{-anon } (\text{elections-}\mathcal{K} \ C) \ \pi \ E = \varphi\text{-anon } \text{well-formed-elections } \pi \ E$
using *cons-domain-valid extensional-continuation-subset*
unfolding *\varphi-anon.simps*
by *metis*
hence *action-induced-rel* (*carrier anonymity}_{\mathcal{G}}*) (*elections-}\mathcal{K} \ C*)
 $(\varphi\text{-anon } \text{well-formed-elections}) =$
 $\text{action-induced-rel } (\text{carrier anonymity}_{\mathcal{G}}) (\text{elections-}\mathcal{K} \ C)$
 $(\varphi\text{-anon } (\text{elections-}\mathcal{K} \ C))$
using *coinciding-actions-ind-equal-rel*
by *metis*
hence *is-symmetry* (*elect-r* $\circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)$)
 $(\text{Invariance } (\text{action-induced-rel}$
 $(\text{carrier anonymity}_{\mathcal{G}}) (\text{elections-}\mathcal{K} \ C) (\varphi\text{-anon } \text{well-formed-elections})))$
using *anon-C*
unfolding *consensus-rule-anonymity'.simps anonymity}_{\mathcal{R}}.simps*
by *presburger*
thus *?thesis*
using *cons-domain-valid assms anonymous-group-action.group-action-axioms*

$\text{anonymity invar-dist-cons-imp-invar-dr-rule}$
unfolding $\text{distance-anonymity'}.sims \text{anonymity}_{\mathcal{R}}.sims \text{anonymity'}.sims$
 $\text{consensus-rule-anonymity'}.sims$
 by *blast*
qed

theorem (in *result-properties*) *neutr-dist-and-cons-imp-neutr-dr*:

fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'b \text{ Result}) \text{ Consensus-Class}$
assumes
 $\text{neutral-d: distance-neutrality well-formed-elections } d$ **and**
 $\text{neutral-C: consensus-rule-neutrality (elections-}\mathcal{K}\text{ } C) C$ **and**
 $\text{closed-C: closed-restricted-rel (neutrality}_{\mathcal{R}} \text{ well-formed-elections)}$
 $\text{well-formed-elections (elections-}\mathcal{K}\text{ } C)$
shows $\text{neutrality well-formed-elections (distance-}\mathcal{R}\text{ } d C)$
proof –
have $\forall \pi. \forall E \in \text{elections-}\mathcal{K}\text{ } C.$
 $\varphi\text{-neutral well-formed-elections } \pi E = \varphi\text{-neutral (elections-}\mathcal{K}\text{ } C) \pi E$
using *cons-domain-valid extensional-continuation-subset*
unfolding $\varphi\text{-neutral}.sims$
 by *metis*
hence $\text{is-symmetry (elect-r } \circ \text{ fun}_{\mathcal{E}} (\text{rule-}\mathcal{K}\text{ } C))$
 $(\text{action-induced-equivariance (carrier neutrality}_{\mathcal{G}}) (\text{elections-}\mathcal{K}\text{ } C))$
 $(\varphi\text{-neutral well-formed-elections (set-action } \psi\text{-neutral}))$
using *neutral-C equivar-ind-by-act-coincide*
unfolding *consensus-rule-neutrality.sims*
 by (*metis (no-types, lifting)*)
thus *?thesis*
using *neutral-d closed-C φ -neutral-action.group-action-axioms*
 $\text{neutrality action-neutral cons-domain-valid[of } C]$
 $\text{invar-dist-equivar-cons-imp-equivar-dr-rule[of}$
 $\text{- } \varphi\text{-neutral well-formed-elections]}$
 by *simp*
qed

theorem *reversal-sym-dist-and-cons-imp-reversal-sym-dr*:

fixes
 $d :: ('a, 'c) \text{ Election Distance}$ **and**
 $C :: ('a, 'c, 'a \text{ rel Result}) \text{ Consensus-Class}$
assumes
 $\text{reverse-sym-d: distance-reversal-symmetry well-formed-elections } d$ **and**
 $\text{reverse-sym-C: consensus-rule-reversal-symmetry (elections-}\mathcal{K}\text{ } C) C$ **and**
 $\text{closed-C: closed-restricted-rel (reversal}_{\mathcal{R}} \text{ well-formed-elections)}$
 $\text{well-formed-elections (elections-}\mathcal{K}\text{ } C)$
shows $\text{reversal-symmetry well-formed-elections (SWF-result.distance-}\mathcal{R}\text{ } d C)$
proof –
have $\forall \pi. \forall E \in \text{elections-}\mathcal{K}\text{ } C.$
 $\varphi\text{-reverse well-formed-elections } \pi E = \varphi\text{-reverse (elections-}\mathcal{K}\text{ } C) \pi E$

using *cons-domain-valid extensional-continuation-subset*
unfolding φ -reverse.simps
by *metis*
hence *is-symmetry* (*elect-r* \circ *fun_E* (*rule-K C*))
 (*action-induced-equivariance* (*carrier reversal_G*) (*elections-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 *SWF-result.invar-dist-equivar-cons-imp-equivar-dr-rule*
 reversal-symmetry cons-domain-valid reverse-sym-d closed-C
 φ -reverse-action.group-action-axioms
 ψ -reverse-action.group-action-axioms
unfolding *reversal-symmetry-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 finite-elections-V (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 *reftp-on'* (*elections-K C*)
 (*Restrp* (*homogeneity_R finite-elections-V*) (*elections-K C*))
using *reft-homogeneity_R[of elections-K C] cons-domain-finite[of C]*
by *presburger*
moreover have
 is-symmetry ($\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}$)
 (*Invariance* (*homogeneity_R finite-elections-V*))
using *homogeneity*
by *simp*
ultimately show *?thesis*
using *assms tot-invar-dist-imp-invar-dr-rule*
unfolding *distance-homogeneity-def homogeneity.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- $\mathcal{V}$  d
  shows homogeneity' finite-elections- $\mathcal{V}$  (distance- $\mathcal{R}$  d C)
proof (unfold homogeneity'.simps)
  have Restrp (homogeneity $\mathcal{R}$ ' finite-elections- $\mathcal{V}$ ) (elections- $\mathcal{K}$  C) =
    homogeneity $\mathcal{R}$ ' (elections- $\mathcal{K}$  C)
  using cons-domain-finite
  unfolding homogeneity $\mathcal{R}$ '.simps finite-elections- $\mathcal{V}$ -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 ( $\lambda$  E. limit (alternatives- $\mathcal{E}$  E) UNIV)
    (Invariance (homogeneity $\mathcal{R}$ ' finite-elections- $\mathcal{V}$ ))
  using homogeneity'
  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

```

5.6.4 Properties

```

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

```

5.7 Distance Rationalization on Election Quotients

```

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

```

5.7.1 Distances

```

fun distanceQ :: 'x Distance  $\Rightarrow$  'x set Distance where
  distanceQ d A B = (if (A = {}  $\wedge$  B = {}) then 0 else
    (if (A = {}  $\vee$  B = {}) then  $\infty$  else

```

$$\pi_Q (\text{tup } d) (A \times B)))$$

fun *relation-paths* :: 'x rel \Rightarrow 'x list set **where**
relation-paths *r* =
 $\{p. \exists k. (\text{length } p = 2 * k \wedge (\forall i < k. (p!(2 * i), p!(2 * i + 1)) \in r))\}$

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

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

fun *distance-infimum*_Q :: 'x Distance \Rightarrow 'x set Distance **where**
*distance-infimum*_Q *d* *A* *B* = $\text{Inf } \{d \ a \ b \mid a \ b. a \in A \wedge b \in B\}$

fun *simple* :: 'x rel \Rightarrow 'x set \Rightarrow 'x Distance \Rightarrow bool **where**
simple *r* *X* *d* =
 $(\forall A \in X // r. (\exists a \in A. \forall B \in X // r. \text{distance-infimum}_Q \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. b \in B\}))$

— 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*_D *d* *r*) = (*tup* *d* respects (*product* *r*))

unfolding *total-invariance*_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-invarianceD 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  $\longrightarrow$  distanceQ 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  $A \times B \in (X \times X) // (\text{product } r)$ 
  unfolding quotient-def
  by fastforce
moreover have equiv  $(X \times X) (\text{product } r)$ 
  using equiv-X-r product-helper UNIV-Times-UNIV equivE equivI
  by metis
moreover have tup d respects  $(\text{product } r)$ 
  using tot-inv-dist-d-r tot-dist-invariance-is-congruence
  by metis
ultimately show  $\text{distance}_{\mathcal{Q}} d A B = d a b$ 
  unfolding  $\text{distance}_{\mathcal{Q}}.\text{simps}$ 
  using pass-to-quotient a-in-A b-in-B
  by fastforce
qed

```

```

lemma relation-paths-subset:
  fixes
     $n :: \text{nat}$  and
     $p :: 'x \text{ list}$  and
     $r :: 'x \text{ rel}$  and
     $X :: 'x \text{ set}$ 
  assumes  $r \subseteq X \times X$ 
  shows  $\forall p. p \in \text{relation-paths } r \longrightarrow (\forall i < \text{length } p. p!i \in X)$ 
proof (safe)
  fix
     $p :: 'x \text{ list}$  and
     $i :: \text{nat}$ 
  assume  $p \in \text{relation-paths } r$ 
  then obtain  $k :: \text{nat}$  where
     $\text{len-}p: \text{length } p = 2 * k$  and
     $\text{rel}: \forall i < k. (p!(2 * i), p!(2 * i + 1)) \in r$ 
    by auto
  moreover obtain  $k' :: \text{nat}$  where
     $i\text{-cases}: i = 2 * k' \vee i = 2 * k' + 1$ 
    using diff-Suc-1 even-Suc oddE odd-two-times-div-two-nat
    by metis
  moreover assume  $i < \text{length } p$ 
  ultimately have  $k' < k$ 
    by linarith
  thus  $p!i \in X$ 
    using assms rel i-cases
    by blast
qed

```

```

lemma admissible-path-len:
  fixes
     $d :: 'x \text{ Distance}$  and
     $r :: 'x \text{ rel}$  and
     $X :: 'x \text{ set}$  and

```



```

    a b :: 'x and
    p :: 'x list
  assumes refl-on X r
  shows triangle-ineq X d  $\wedge$  p  $\in$  relation-paths r  $\wedge$  total-invarianceD d r
     $\wedge$  a  $\in$  X  $\wedge$  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-invarianceD d r and
      hyp:
         $\wedge$  a b. triangle-ineq X d  $\implies$  xs  $\in$  relation-paths r
           $\implies$  total-invarianceD d r  $\implies$  a  $\in$  X  $\implies$  b  $\in$  X
           $\implies$  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  $\forall x y. x \in X \wedge y \in X \longrightarrow d x y \leq$  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 $\text{path-length } (a\#(x\#y\#xs)@[b]) \ d \geq d \ a \ x + d \ y \ b$
 using *assms add-left-mono assms refl-onD2 b-in-X*
 unfolding *refl-on-def*
 by *metis*
 moreover have $d \ a \ x + d \ y \ b = d \ a \ x + d \ x \ b$
 using *invar x-rel-y rewrite-total-invariance_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 $d \ a \ b \leq \text{path-length } (a\#(x\#y\#xs)@[b]) \ d$
 by *simp*
 qed

lemma *quotient-dist-coincides-with-dist_Q*:

fixes
 $d :: 'x \text{ Distance}$ and
 $r :: 'x \text{ rel}$ and
 $X :: 'x \text{ set}$
 assumes
 equiv: $\text{equiv } X \ r$ and
 tri: $\text{triangle-ineq } X \ d$ and
 invar: $\text{total-invariance}_D \ d \ r$
 shows $\forall \ A \in X \ // \ r. \ \forall \ B \in X \ // \ r. \ \text{quotient-dist } r \ d \ A \ B = \text{distance}_Q \ d \ A \ B$
 proof (clarify)
 fix $A \ B :: 'x \text{ set}$
 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: $\text{distance}_Q \ d \ A \ B = d \ a \ b$
 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 $\text{equiv-class}: A = r \ `` \ \{a\} \wedge B = r \ `` \ \{b\}$
 using *A-in-quot-X el equiv quotientI quotient-eq-iff*

```

  by slow
have  $\forall p \in \text{admissible-paths } r \ A \ B.$ 
   $(\exists p' \ x \ y. \ x \in A \wedge y \in B \wedge p' \in \text{relation-paths } r \wedge p = x\#p'@[y])$ 
  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  $\forall p. \ p \in \text{admissible-paths } r \ A \ B \longrightarrow \text{path-length } p \ d \geq d \ a \ b$ 
  using admissible-path-len[of X r d] tri el invar in-mono
  by metis
hence  $\forall l. \ l \in \bigcup \{ \{ \text{path-length } p \ d \mid p. \ p \in \text{admissible-paths } r \ A \ B \} \}$ 
   $\longrightarrow l \geq d \ a \ b$ 
  by blast
hence geq: quotient-dist  $r \ d \ A \ B \geq \text{distance}_{\mathcal{Q}} \ d \ A \ B$ 
  unfolding quotient-dist.simps le-Inf-iff
  using def-dist
  by simp
have  $[a, b] \in \text{admissible-paths } r \ A \ B$ 
  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  $r \ d \ A \ B = \text{distance}_{\mathcal{Q}} \ d \ A \ B$ 
  using geq def-dist nle-le
  by metis
qed

lemma inf-dist-coincides-with-distQ:
  fixes
     $d :: 'x \text{ Distance}$  and
     $r :: 'x \text{ rel}$  and
     $X :: 'x \text{ set}$ 
  assumes
    equiv-X-r: equiv X r and
    tot-inv-d-r: total-invarianceD d r
  shows  $\forall A \in X // r. \ \forall B \in X // r.$ 

```

$distance_infimum_Q \ d \ A \ B = distance_Q \ d \ A \ B$

proof (*clarify*)
fix $A \ B :: 'x \ set$
assume
 $A\text{-in-quot-}X: A \in X \ // \ r$ **and**
 $B\text{-in-quot-}X: B \in X \ // \ r$
then obtain
 $a \ b :: 'x$ **where**
 $el: a \in A \wedge b \in B$ **and**
 $def\text{-}dist: distance_Q \ d \ A \ B = d \ a \ b$
using $dist\text{-}pass\text{-}to\text{-}quotient \ equiv\text{-}X\text{-}r \ tot\text{-}inv\text{-}d\text{-}r$
 $in\text{-}quotient\text{-}imp\text{-}non\text{-}empty \ ex\text{-}in\text{-}conv$
by (*metis* (*full-types*))
from $def\text{-}dist \ equiv\text{-}X\text{-}r \ tot\text{-}inv\text{-}d\text{-}r$
have $\forall \ x \ y. x \in A \wedge y \in B \longrightarrow d \ x \ y = d \ a \ b$
using $dist\text{-}pass\text{-}to\text{-}quotient \ A\text{-in-quot-}X \ B\text{-in-quot-}X$
by force
hence $\{d \ x \ y \mid x \ y. x \in A \wedge y \in B\} = \{d \ a \ b\}$
using el
by blast
thus $distance_infimum_Q \ d \ A \ B = distance_Q \ d \ A \ B$
unfolding $distance_infimum_Q.simps$
using $def\text{-}dist$
by simp
qed

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

```

then obtain  $\alpha :: \text{ereal}$  where
   $\text{inf}: \alpha \in \{\text{Inf } \{d \ a \ b \mid b. b \in B\} \mid a. a \in A\}$  and
   $\text{less}: \alpha < \text{Inf } \{d \ a \ b \mid a \ b. a \in A \wedge b \in B\}$ 
  using Inf-less-iff
  by (metis (no-types, lifting))
then obtain  $a :: 'x$  where
   $a\text{-in-}A: a \in A$  and
   $\alpha = \text{Inf } \{d \ a \ b \mid b. b \in B\}$ 
  by blast
with less
have  $\text{inf-less}: \text{Inf } \{d \ a \ b \mid b. b \in B\} < \text{Inf } \{d \ a \ b \mid a \ b. a \in A \wedge b \in B\}$ 
  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  $\text{Inf } \{d \ a \ b \mid a \ b. a \in A \wedge b \in B\} \leq \text{Inf } \{d \ a \ b \mid b. b \in B\}$ 
  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
     $d :: 'y \text{ Distance}$  and
     $G :: 'x \text{ monoid}$  and
     $Y :: 'y \text{ set}$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$ 
  assumes
     $\text{action-}\varphi: \text{group-action } G \ Y \ \varphi$  and
     $\text{invar}: \text{invariance}_{\mathcal{D}} \ d \ (\text{carrier } G) \ Y \ \varphi$ 
  shows simple ( $\text{action-induced-rel } (\text{carrier } G) \ Y \ \varphi$ )  $Y \ d$ 
proof (unfold simple.simps, safe)
  fix  $A :: 'y \text{ set}$ 
  assume  $\text{class}_Y: A \in Y$  //  $\text{action-induced-rel } (\text{carrier } G) \ Y \ \varphi$ 
  moreover have equiv-rel:
     $\text{equiv } Y \ (\text{action-induced-rel } (\text{carrier } G) \ Y \ \varphi)$ 
  using assms rel-ind-by-group-act-equiv
  by blast
  ultimately obtain  $a :: 'y$  where
     $a\text{-in-}A: a \in A$ 
  using equiv-Eps-in
  by blast
  have subset:  $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) \ Y \ \varphi. B \subseteq Y$ 

```

using *equiv-rel in-quotient-imp-subset*
by *blast*
hence $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\forall B' \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\forall b \in B. \forall c \in B'. b \in Y \wedge c \in Y$
using *class_Y*
by *blast*
hence *eq-dist*:
 $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\forall B' \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\forall b \in B. \forall c \in B'. \forall g \in \text{carrier } G.$
 $d(\varphi g c)(\varphi g b) = d c b$
using *invar rewrite-invariance_D class_Y*
by *metis*
have $\forall b \in Y. \forall g \in \text{carrier } G.$
 $(b, \varphi g b) \in \text{action-induced-rel } (\text{carrier } G) Y \varphi$
unfolding *action-induced-rel.simps*
using *group-action.element-image action-φ*
by *fastforce*
hence $\forall b \in Y. \forall g \in \text{carrier } G.$
 $\varphi g b \in \text{action-induced-rel } (\text{carrier } G) Y \varphi \text{ “}\{b\}$
unfolding *Image-def*
by *blast*
moreover have *equiv-class*:
 $\forall B. B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi \longrightarrow$
 $(\forall b \in B. B = \text{action-induced-rel } (\text{carrier } G) Y \varphi \text{ “}\{b\})$
using *Image-singleton-iff equiv-class-eq-iff equiv-rel*
 $\text{quotientI quotient-eq-iff}$
by *meson*
ultimately have *closed-class*:
 $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\forall b \in B. \forall g \in \text{carrier } G. \varphi g b \in B$
using *equiv-rel subset*
by *blast*
with *eq-dist class_Y*
have *a-subset-A*:
 $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$
 $\{d a b \mid b. b \in B\} \subseteq \{d a b \mid a b. a \in A \wedge b \in B\}$
using *a-in-A*
by *blast*
have $\forall a' \in A. A = \text{action-induced-rel } (\text{carrier } G) Y \varphi \text{ “}\{a'\}$
using *class_Y equiv-rel equiv-class*
by *presburger*
hence $\forall a' \in A. (a', a) \in \text{action-induced-rel } (\text{carrier } G) Y \varphi$
using *a-in-A*
by *blast*
hence $\forall a' \in A. \exists g \in \text{carrier } G. \varphi g a' = a$
by *simp*
hence $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G) Y \varphi.$

```

     $\forall a' b. a' \in A \wedge b \in B \longrightarrow (\exists g \in \text{carrier } G. d\ a'\ b = d\ a\ (\varphi\ g\ b))$ 
  using eq-dist classY
  by metis
  hence  $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G)\ Y\ \varphi.$ 
     $\forall a' b. a' \in A \wedge b \in B \longrightarrow d\ a'\ b \in \{d\ a\ b \mid b. b \in B\}$ 
  using closed-class mem-Collect-eq
  by fastforce
  hence  $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G)\ Y\ \varphi.$ 
     $\{d\ a\ b \mid b. b \in B\} \supseteq \{d\ a\ b \mid a\ b. a \in A \wedge b \in B\}$ 
  using closed-class
  by blast
  with a-subset-A
  have  $\forall B \in Y // \text{action-induced-rel } (\text{carrier } G)\ Y\ \varphi.$ 
     $\text{distance-infimum}_{\mathcal{Q}}\ d\ A\ B = \text{Inf } \{d\ a\ b \mid b. b \in B\}$ 
  unfolding distance-infimumQ.simps
  by fastforce
  thus  $\exists a \in A. \forall B \in Y // \text{action-induced-rel } (\text{carrier } G)\ Y\ \varphi.$ 
     $\text{distance-infimum}_{\mathcal{Q}}\ d\ A\ B = \text{Inf } \{d\ a\ b \mid b. b \in B\}$ 
  using a-in-A
  by blast
qed

```

lemma *tot-invar-dist-simple*:

```

  fixes
     $d :: 'x\ \text{Distance}$  and
     $r :: 'x\ \text{rel}$  and
     $X :: 'x\ \text{set}$ 
  assumes
    equiv-on-X:  $\text{equiv } X\ r$  and
    invar:  $\text{total-invariance}_{\mathcal{D}}\ d\ r$ 
  shows simple  $r\ X\ d$ 
  proof (unfold simple.simps, safe)
    fix  $A :: 'x\ \text{set}$ 
    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  $\forall a\ a'. a \in A \wedge a' \in A \longrightarrow (a, a') \in r$ 
    by blast
    moreover have  $\forall B \in X // r. \forall b \in B. (b, b) \in r$ 
    using equiv-on-X quotient-eq-iff
    by metis
    ultimately have
       $\forall B \in X // r. \forall a\ a'\ b. a \in A \wedge a' \in A \wedge b \in B \longrightarrow d\ a\ b = d\ a'\ b$ 
    using invar rewrite-total-invarianceD

```

by *simp*
 hence $\forall B \in X // r. \{d \ a \ b \mid a \ b. \ a \in A \wedge b \in B\} = \{d \ a \ b \mid a' \ b. \ a' \in A \wedge b \in B\}$
 using *a-in-A*
 by *blast*
 moreover have
 $\forall B \in X // r. \{d \ a \ b \mid a' \ b. \ a' \in A \wedge b \in B\} = \{d \ a \ b \mid b. \ b \in B\}$
 using *a-in-A*
 by *blast*
 ultimately have
 $\forall B \in X // r. \text{Inf } \{d \ a \ b \mid a \ b. \ a \in A \wedge b \in B\} = \text{Inf } \{d \ a \ b \mid b. \ b \in B\}$
 by *simp*
 hence $\forall B \in X // r. \text{distance-infimum}_{\mathcal{Q}} \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. \ b \in B\}$
 by *simp*
 thus $\exists a \in A. \forall B \in X // r. \text{distance-infimum}_{\mathcal{Q}} \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. \ b \in B\}$
 using *a-in-A*
 by *blast*
 qed

5.7.2 Consensus and Results

fun *elections- $\mathcal{K}_{\mathcal{Q}}$* :: $('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set set}$ **where**
elections- $\mathcal{K}_{\mathcal{Q}}$ *r C* = (*elections- \mathcal{K}* *C*) // *r*

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

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:

fixes

X Y Z :: $'x \text{ set}$ **and**

r :: $'x \text{ rel}$

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) ∈ r
    using assms
    unfolding quotient-def Image-def
    by blast
  hence (y, z) ∈ r ∩ Y × X
    using assms
    unfolding equiv-def refl-on-def
    by blast
  hence z ∈ {z. ∃ y ∈ Y. (y, z) ∈ r ∩ Y × X}
    by blast
  thus z ∈ 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: $\text{equiv } X \ r$ **and**
cons-subset: $\text{elections-}\mathcal{K} \ C \subseteq X$ **and**
invar-res: $\text{is-symmetry } (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ (\text{Invariance } r)$

shows $\forall a \in A. \text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV} = \text{limit}_{\mathcal{Q}} A \ \text{UNIV}$

proof

fix $a :: ('a, 'v)$ Election

assume $a\text{-in-}A$: $a \in A$

hence $\forall b \in A. (a, b) \in r$

using *quot-class equiv-rel quotient-eq-iff*

by *metis*

hence $\forall b \in A.$

$\text{limit } (\text{alternatives-}\mathcal{E} \ b) \ \text{UNIV} = \text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV}$

using *invar-res*

unfolding *is-symmetry.simps*

by (*metis (mono-tags, lifting)*)

hence $\text{limit}_{\mathcal{Q}} A \ \text{UNIV} = \bigcap \{\text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV}\}$

unfolding *limit_Q.simps*

using $a\text{-in-}A$

by *blast*

thus $\text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV} = \text{limit}_{\mathcal{Q}} A \ \text{UNIV}$

by *simp*

qed

lemma (in result) *preimg-invar*:

fixes

```

f :: 'x ⇒ '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 ⊆ X and
  closed-domain: closed-restricted-rel r X domain_f and
  invar-f: is-symmetry f (Invariance (Restr r domain_f))
shows ∀ y. (preimg f domain_f y) // r = preimg (π_Q f) (domain_f // r) y
proof (safe)
  fix
    A :: 'x set and
    y :: 'y
  assume preimg-quot: A ∈ preimg f domain_f y // r
  hence A-in-dom: A ∈ domain_f // r
    unfolding preimg.simps quotient-def
    by blast
  obtain x :: 'x where
    x ∈ preimg f domain_f y and
    A-eq-img-singleton-r: A = r “ {x}
    using equiv-rel preimg-quot quotientE
    unfolding quotient-def
    by blast
  hence x-in-dom-and-f-x-y: x ∈ domain_f ∧ f x = y
    unfolding preimg.simps
    by blast
  moreover have r “ {x} ⊆ X
    using equiv-rel equiv-type
    by fastforce
  ultimately have r “ {x} ⊆ domain_f
    using closed-domain A-eq-img-singleton-r A-in-dom
    by fastforce
  hence ∀ x' ∈ r “ {x}. (x, x') ∈ Restr r domain_f
    using x-in-dom-and-f-x-y in-mono
    by blast
  hence ∀ x' ∈ r “ {x}. f x' = y
    using invar-f x-in-dom-and-f-x-y
    unfolding is-symmetry.simps
    by metis
  moreover have x ∈ 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 π_Q f A = y
    unfolding π_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  $A \in \text{preimg } (\pi_Q f) (\text{domain}_f // r) y$ 
    using A-in-dom
    unfolding preimg.simps
    by blast
next
fix
  A :: 'x set and
  y :: 'y
  assume quot-preimg:  $A \in \text{preimg } (\pi_Q f) (\text{domain}_f // r) y$ 
  hence A-in-dom-rel-r:  $A \in \text{domain}_f // r$ 
    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:  $A \subseteq \text{domain}_f$ 
    using closed-under-equiv-rel-subset[of X r domain_f A]
      closed-domain cons-subset A-in-dom-rel-r equiv-rel
    by blast
  moreover obtain x :: 'x where
    x-in-A:  $x \in A$  and
    A-eq-r-img-single-x:  $A = r `` \{x\}$ 
    using A-in-dom-rel-r equiv-rel cons-subset equiv-class-self in-mono quotientE
    by metis
  ultimately have  $\forall x' \in A. (x, x') \in \text{Restr } r \text{ domain}_f$ 
    by blast
  hence  $\forall x' \in A. f x' = f x$ 
    using invar-f
    by fastforce
  hence  $f ` A = \{f x\}$ 
    using x-in-A
    by blast
  hence  $\pi_Q f A = f x$ 
    unfolding  $\pi_Q.simps$  singleton-set.simps
    using is-singleton-altdef singleton-set-def-if-card-one
    by fastforce
  also have  $\pi_Q f A = y$ 
    using quot-preimg
    unfolding preimg.simps
    by blast
  finally have  $f x = y$ 
    by simp
  moreover have  $x \in \text{domain}_f$ 
    using x-in-A A-in-dom
    by blast
  ultimately have  $x \in \text{preimg } f \text{ domain}_f y$ 

```

```

    by simp
  thus  $A \in \text{preimg } f \text{ domain}_f y // r$ 
    using  $A\text{-eq-r-img-single-}x$ 
    unfolding  $\text{quotient-def}$ 
    by blast
qed

lemma minimizer-helper:
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $\text{domain}_f :: 'x \text{ set}$  and
     $d :: 'x \text{ Distance}$  and
     $Y :: 'y \text{ set}$  and
     $x :: 'x$  and
     $y :: 'y$ 
  shows  $y \in \text{minimizer } f \text{ domain}_f d Y \ x =$ 
     $(y \in Y \wedge (\forall y' \in Y.$ 
       $\text{Inf } (d \ x \ ' (\text{preimg } f \text{ domain}_f y)) \leq \text{Inf } (d \ x \ ' (\text{preimg } f \text{ domain}_f y'))))$ 
    unfolding  $\text{is-arg-min-def minimizer.simps arg-min-set.simps}$ 
    by auto

lemma rewr-singleton-set-system-union:
  fixes
     $Y :: 'x \text{ set set}$  and
     $X :: 'x \text{ set}$ 
  assumes  $Y \subseteq \text{singleton-set-system } X$ 
  shows
     $\text{singleton-set-union: } x \in \bigcup Y \longleftrightarrow \{x\} \in Y$  and
     $\text{obtain-singleton: } A \in \text{singleton-set-system } X \longleftrightarrow (\exists x \in X. A = \{x\})$ 
    unfolding  $\text{singleton-set-system.simps}$ 
    using  $\text{assms}$ 
    by auto

lemma union-inf:
  fixes  $X :: \text{ereal set set}$ 
  shows  $\text{Inf } \{\text{Inf } A \mid A. A \in X\} = \text{Inf } (\bigcup X)$ 
proof -
  let  $?inf = \text{Inf } \{\text{Inf } A \mid A. A \in X\}$ 
  have  $\forall A \in X. \forall x \in A. ?inf \leq x$ 
    using  $\text{INF-lower2 Inf-lower Setcompr-eq-image}$ 
    by metis
  hence  $\forall x \in \bigcup X. ?inf \leq x$ 
    by simp
  hence  $\text{le: } ?inf \leq \text{Inf } (\bigcup X)$ 
    using  $\text{Inf-greatest}$ 
    by blast
  have  $\forall A \in X. \text{Inf } (\bigcup X) \leq \text{Inf } A$ 
    using  $\text{Inf-superset-mono Union-upper}$ 
    by metis

```

hence $\text{Inf} (\bigcup X) \leq \text{Inf} \{ \text{Inf} A \mid A. A \in X \}$
 using *le-Inf-iff*
 by *auto*
 thus *?thesis*
 using *le*
 by *simp*
 qed

5.7.3 Distance Rationalization

fun (in result) $\mathcal{R}_{\mathcal{Q}} :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ set}$ **where**
 $\mathcal{R}_{\mathcal{Q}} r d C A =$
 $\bigcup (\text{minimizer } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} r C)$
 $(\text{distance-infimum}_{\mathcal{Q}} d) (\text{singleton-set-system } (\text{limit}_{\mathcal{Q}} A \text{ UNIV})) A)$

fun (in result) $\text{distance-}\mathcal{R}_{\mathcal{Q}} :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ Result}$ **where**
 $\text{distance-}\mathcal{R}_{\mathcal{Q}} r d C A =$
 $(\mathcal{R}_{\mathcal{Q}} r d C A,$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} E) \text{ UNIV}) A - \mathcal{R}_{\mathcal{Q}} r d C A,$
 $\{\})$

Proposition 4.17 by Hadjibeyli and Wilson [3].

theorem (in result) *invar-dr-simple-dist-imp-quotient-dr-winners:*

fixes

$d :: ('a, 'v) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $r :: ('a, 'v) \text{ Election rel}$ **and**
 $X A :: ('a, 'v) \text{ Election set}$

assumes

simple: $\text{simple } r X d$ **and**
closed-domain: $\text{closed-restricted-rel } r X (\text{elections-}\mathcal{K} C)$ **and**
invar-res:
 $\text{is-symmetry } (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} E) \text{ UNIV}) (\text{Invariance } r)$ **and**
invar-C: $\text{is-symmetry } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C))$
 $(\text{Invariance } (\text{Restr } r (\text{elections-}\mathcal{K} C)))$ **and**
invar-dr: $\text{is-symmetry } (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C)) (\text{Invariance } r)$ **and**
quot-class: $A \in X // r$ **and**
equiv-rel: $\text{equiv } X r$ **and**
cons-subset: $\text{elections-}\mathcal{K} C \subseteq X$

shows $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C)) A = \mathcal{R}_{\mathcal{Q}} r d C A$

proof –

have *preimg-img-imp-cls:*

$\forall y B. B \in \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) y$
 $\longrightarrow B \in (\text{elections-}\mathcal{K} C) // r$

by *simp*

have $\forall y'. \forall E$

$\in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) y'. E \in r `` \{E\}$

using *equiv-rel cons-subset equiv-class-self equiv-rel in-mono*
unfolding *equiv-def preimg.simps*
by *fastforce*
hence $\forall y'.$

$$\bigcup (preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) y' // r) \supseteq$$

$$preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) y'$$
unfolding *quotient-def*
by *blast*
moreover have $\forall y'.$

$$\bigcup (preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) y' // r) \subseteq$$

$$preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) y'$$
proof (*intro allI subsetI*)
fix
 $Y' :: 'r \text{ set}$ **and**
 $E :: ('a, 'v) \text{ Election}$
assume $E \in \bigcup (preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) Y' // r)$
then obtain $B :: ('a, 'v) \text{ Election set}$ **where**
 $E\text{-in-}B: E \in B$ **and**
 $B \in preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) Y' // r$
by *blast*
then obtain $E' :: ('a, 'v) \text{ Election}$ **where**
 $B = r \text{ `` } \{E'\}$ **and**
 $map\text{-to-}Y': E' \in preimg (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (elections-\mathcal{K} C) Y'$
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*: $\forall y'.$

$$\bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r) =$$

$$\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'$$

by *blast*

have *quot-classes-subset*: $(\text{elections-}\mathcal{K} \ C) // r \subseteq X // r$

using *cons-subset*

unfolding *quotient-def*

by *blast*

obtain $a :: ('a, 'v) \text{ Election}$ where

$a\text{-in-}A$: $a \in A$ and

$a\text{-def-inf-dist}$:

$\forall B \in X // r.$

$\text{distance-infimum}_{\mathcal{Q}} \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. b \in B\}$

using *simple quot-class*

unfolding *simple.simps*

by *blast*

hence *inf-dist-preimg-sets*:

$\forall y' B. B \in \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y'$

$\longrightarrow \text{distance-infimum}_{\mathcal{Q}} \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. b \in B\}$

using *preimg-img-imp-cls quot-classes-subset*

by *blast*

have *wf-res-eq*: $\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV}) =$

$\text{singleton-set-system } (\text{limit}_{\mathcal{Q}} \ A \ \text{UNIV})$

using *invar-res a-in-A quot-class cons-subset equiv-rel limit-invar*

by *metis*

have *inf-le-iff*: $\forall x.$

$(\forall y' \in \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV}).$

$\text{Inf } \{d \ a \ ' \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{x\}\}$

$\leq \text{Inf } \{d \ a \ ' \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'\}$

$= (\forall y' \in \text{singleton-set-system } (\text{limit}_{\mathcal{Q}} \ A \ \text{UNIV}).$

$\text{Inf } (\text{distance-infimum}_{\mathcal{Q}} \ d \ A \ ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)))$

$\quad (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ \{x\})$

$\leq \text{Inf } (\text{distance-infimum}_{\mathcal{Q}} \ d \ A \ ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)))$

$\quad (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y'))$

proof –

have *preimg-partition-dist*: $\forall y'.$

$\text{Inf } \{d \ a \ b \mid b. b \in$

$\bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r)\} =$

$\text{Inf } \{d \ a \ ' \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'\}$

using *Setcompr-eq-image preimg-partition*

by *metis*

have $\forall y'.$

$\{\text{Inf } \{d \ a \ b \mid b. b \in B\}$

$\mid B. B \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r\}$

$= \{\text{Inf } E \mid E. E \in \{\{d \ a \ b \mid b. b \in B\}$

$\mid B. B \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r\}\}$

by *blast*

hence $\forall y'.$

$\text{Inf } \{\text{Inf } \{d \ a \ b \mid b. b \in B\} \mid B.$

$B \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r\} =$

$$\text{Inf } (\bigcup \{ \{ d \ a \ b \mid b. \ b \in B \} \mid B. \\ B \in (\text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r) \})$$
using *union-inf*
by *presburger*
moreover have

$$\forall \ y'. \\ \{ d \ a \ b \mid b. \ b \in \bigcup \\ (\text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) \\ (\text{elections-}\mathcal{K} \ C) \ y' // r) \} = \\ \bigcup \{ \{ d \ a \ b \mid b. \ b \in B \} \mid B. \\ B \in (\text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) \\ (\text{elections-}\mathcal{K} \ C) \ y' // r) \}$$
by *blast*
ultimately have *rewrite-inf-dist*:

$$\forall \ y'. \text{Inf } \{ \text{Inf } \{ d \ a \ b \mid b. \ b \in B \} \\ \mid B. \ B \in \text{preimg} \\ (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r \} = \\ \text{Inf } \{ d \ a \ b \\ \mid b. \ b \in \bigcup (\text{preimg} \\ (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r) \}$$
by *presburger*
have $\forall \ y'. \text{distance-infimum}_{\mathcal{Q}} \ d \ A \ ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) \\ (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y' = \\ \{ \text{Inf } \{ d \ a \ b \mid b. \ b \in B \} \\ \mid B. \ B \in \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y' \}$
using *inf-dist-preimg-sets*
unfolding *Image-def*
by *auto*
moreover have $\forall \ y'. \\ \{ \text{Inf } \{ d \ a \ b \mid b. \ b \in B \} \mid B. \\ B \in \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y' \} = \\ \{ \text{Inf } \{ d \ a \ b \mid b. \ b \in B \} \mid B. \\ B \in (\text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y') // r \}$
unfolding *elections- $\mathcal{K}_{\mathcal{Q}}$.sims*
using *preimg-invar closed-domain cons-subset equiv-rel invar-C*
by *blast*
ultimately have

$$\forall \ y'. \text{Inf } (\text{distance-infimum}_{\mathcal{Q}} \ d \ A \ ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) \\ (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C) \ y') = \\ \text{Inf } \{ \text{Inf } \{ d \ a \ b \mid b. \ b \in B \} \\ \mid B. \ B \in \text{preimg } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r \}$$
by *simp*
thus *?thesis*
using *wf-res-eq rewrite-inf-dist preimg-partition-dist*
by *presburger*
qed
from *a-in-A*
have $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A = \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ a$
using *invar-dr equiv-rel quot-class pass-to-quotient invariance-is-congruence*


```

by blast
moreover have  $\forall x. x \in \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) a \longleftrightarrow x \in \mathcal{R}_{\mathcal{Q}} r d C A$ 
proof
  fix  $x :: 'r$ 
  have  $(x \in \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) a) =$ 
     $(x \in \bigcup (\text{minimizer } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) d$ 
       $(\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} a) \text{ UNIV})) a))$ 
  using  $\mathcal{R}_{\mathcal{W}}\text{-is-minimizer}$ 
  by metis
  also have  $\dots =$ 
     $(\{x\} \in \text{minimizer } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) d$ 
       $(\text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} a) \text{ UNIV})) a)$ 
  using singleton-set-union
  unfolding minimizer.simps arg-min-set.simps is-arg-min-def
  by auto
  also have  $\dots = (\{x\} \in \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} a) \text{ UNIV})$ 
     $\wedge (\forall y' \in \text{singleton-set-system } (\text{limit } (\text{alternatives-}\mathcal{E} a) \text{ UNIV}).$ 
       $\text{Inf } (d a \text{ 'preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) \{x\})$ 
       $\leq \text{Inf } (d a \text{ 'preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) y'))$ 
  using minimizer-helper
  by (metis (no-types, lifting))
  also have  $\dots = (\{x\} \in \text{singleton-set-system } (\text{limit}_{\mathcal{Q}} A \text{ UNIV})$ 
     $\wedge (\forall y' \in \text{singleton-set-system } (\text{limit}_{\mathcal{Q}} A \text{ UNIV}).$ 
       $\text{Inf } (\text{distance-infimum}_{\mathcal{Q}} d A \text{ 'preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)))$ 
       $(\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) \{x\})$ 
       $\leq \text{Inf } (\text{distance-infimum}_{\mathcal{Q}} d A \text{ 'preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)))$ 
       $(\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) y'))$ 
  using wf-res-eq inf-le-iff
  by blast
  also have  $\dots =$ 
     $(\{x\} \in \text{minimizer}$ 
       $(\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} r C)$ 
       $(\text{distance-infimum}_{\mathcal{Q}} d)$ 
       $(\text{singleton-set-system } (\text{limit}_{\mathcal{Q}} A \text{ UNIV})) A)$ 
  using minimizer-helper
  by (metis (no-types, lifting))
  also have  $\dots =$ 
     $(x \in \bigcup (\text{minimizer}$ 
       $(\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} r C)$ 
       $(\text{distance-infimum}_{\mathcal{Q}} d)$ 
       $(\text{singleton-set-system } (\text{limit}_{\mathcal{Q}} A \text{ UNIV})) A))$ 
  using singleton-set-union
  unfolding minimizer.simps arg-min-set.simps is-arg-min-def
  by auto
  finally show  $(x \in \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) a) = (x \in \mathcal{R}_{\mathcal{Q}} r d C A)$ 
  unfolding  $\mathcal{R}_{\mathcal{Q}}.\text{simps}$ 
  by safe
qed
ultimately show  $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C)) A = \mathcal{R}_{\mathcal{Q}} r d C A$ 

```

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 $(\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
 (Invariance r) **and**
invar-C: *is-symmetry* $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$
 (Invariance $(\text{Restr } r \ (\text{elections-}\mathcal{K} \ C)))$ **and**
invar-dr: *is-symmetry* $(\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C))$ (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 $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C)) \ A = \text{distance-}\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$

proof –

have $\forall E. \text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C) \ E =$
 $(\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E,$
 $\text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV} - \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E,$
 $\{\})$

by simp

moreover have $\forall E \in A. \text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C) \ E = \pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A$
 using *invar-dr invariance-is-congruence pass-to-quotient quot-class equiv-rel*
 by blast

moreover have $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A = \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$
 using *invar-dr-simple-dist-imp-quotient-dr-winners assms*
 by blast

moreover have

$\forall E \in A. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV} =$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ A$
 using *invar-res invariance-is-congruence' pass-to-quotient quot-class equiv-rel*
 by blast

ultimately have *all-eq*:

$\forall E \in A. \text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C) \ E =$
 $(\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ A - \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\{\})$

by fastforce

hence

$\{(\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ A - \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\{\})\} \supseteq \text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C) \ ` \ A$

```

    by blast
  moreover have  $A \neq \{\}$ 
    using quot-class equiv-rel in-quotient-imp-non-empty
    by metis
  ultimately have single-img:
     $\{(\mathcal{R}_Q \ r \ d \ C \ A,$ 
       $\pi_Q \ (\lambda \ E. \text{limit} \ (\text{alternatives-}\mathcal{E} \ E) \ UNIV) \ A - \mathcal{R}_Q \ r \ d \ C \ A,$ 
       $\{\})\} =$ 
       $\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C) \ 'A$ 
    using empty-is-image subset-singletonD
    by (metis (no-types, lifting))
  moreover from this
  have  $\text{card} \ (\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C) \ 'A) = 1$ 
    using is-singleton-altdef is-singletonI
    by (metis (no-types, lifting))
  moreover from this single-img
  have  $\text{the-inv} \ (\lambda \ x. \ \{x\}) \ (\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C) \ 'A) =$ 
     $(\mathcal{R}_Q \ r \ d \ C \ A,$ 
       $\pi_Q \ (\lambda \ E. \text{limit} \ (\text{alternatives-}\mathcal{E} \ E) \ UNIV) \ A - \mathcal{R}_Q \ r \ d \ C \ A,$ 
       $\{\})$ 
    using singleton-insert-inj-eq singleton-set.elims singleton-set-def-if-card-one
    by (metis (no-types))
  ultimately show ?thesis
    unfolding distance- $\mathcal{R}_Q$ .simps
    using  $\pi_Q$ .simps[of  $\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C)$ ]
      singleton-set.simps[of  $\text{fun}_{\mathcal{E}} \ (\text{distance-}\mathcal{R} \ d \ C) \ 'A]$ 
    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-SCF-code-lemma:
  fixes  $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 

```

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

lemma *\mathcal{R}_W -SCF-code-lemma:*
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows *SCF-result. \mathcal{R}_W* $d K V A p =$
 $\text{arg-min-set } (\text{score } d K (A, V, p)) (\text{limit-SCF } A \text{ UNIV})$
unfolding *SCF-result. \mathcal{R}_W .simps*
by *safe*

lemma *distance- \mathcal{R} -SCF-code-lemma:*
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows *SCF-result.distance- \mathcal{R}* $d K V A p =$
 $(\text{SCF-result.}\mathcal{R}_W \text{ } d K V A p,$
 $(\text{limit-SCF } A \text{ UNIV}) - \text{SCF-result.}\mathcal{R}_W \text{ } d K V A p,$
 $\{\})$
unfolding *SCF-result.distance- \mathcal{R} .simps*
by *safe*

lemma *\mathcal{R}_W -std-SCF-code-lemma:*
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows *SCF-result. \mathcal{R}_W -std* $d K V A p =$
 $\text{arg-min-set } (\text{score-std } d K (A, V, p)) (\text{limit-SCF } A \text{ UNIV})$
unfolding *SCF-result. \mathcal{R}_W -std.simps*
by *safe*

lemma *distance- \mathcal{R} -std-SCF-code-lemma:*
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**

```

  p :: ('a, 'v) Profile
shows SCF-result.distance- $\mathcal{R}$ -std d K V A p =
  (SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std d K V A p,
   (limit-SCF A UNIV) - SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std d K V A p,
   {})
unfolding SCF-result.distance- $\mathcal{R}$ -std.simps
by safe

lemma anonymity-SCF-code-lemma: SCF-result.anonymity =
  ( $\lambda$  m :: ('a, 'v, 'a Result) Electoral-Module.
   SCF-result.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)))
unfolding 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 SCF-result.electoral-module electoral-module-SCF-code-lemma]]
declare [[lc-add SCF-result. $\mathcal{R}_{\mathcal{W}}$   $\mathcal{R}_{\mathcal{W}}$ -SCF-code-lemma]]
declare [[lc-add SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std  $\mathcal{R}_{\mathcal{W}}$ -std-SCF-code-lemma]]
declare [[lc-add SCF-result.distance- $\mathcal{R}$  distance- $\mathcal{R}$ -SCF-code-lemma]]
declare [[lc-add SCF-result.distance- $\mathcal{R}$ -std distance- $\mathcal{R}$ -std-SCF-code-lemma]]
declare [[lc-add SCF-result.anonymity anonymity-SCF-code-lemma]]

```

Constant aliases to use instead of the interpreted functions.

```

definition  $\mathcal{R}_{\mathcal{W}}$ -SCF-code  $\equiv$  SCF-result. $\mathcal{R}_{\mathcal{W}}$ 
definition  $\mathcal{R}_{\mathcal{W}}$ -std-SCF-code  $\equiv$  SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std
definition distance- $\mathcal{R}$ -SCF-code  $\equiv$  SCF-result.distance- $\mathcal{R}$ 
definition distance- $\mathcal{R}$ -std-SCF-code  $\equiv$  SCF-result.distance- $\mathcal{R}$ -std
definition electoral-module-SCF-code  $\equiv$  SCF-result.electoral-module
definition anonymity-SCF-code  $\equiv$  SCF-result.anonymity

```

```

setup Locale-Code.close-block

```

```

end

```

5.9 Drop Module

```

theory Drop-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 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  $\leq$  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 SCF-result.electoral-module (drop-module n r)
proof (unfold 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-SCF A (?mod V A p)
using set-partition
by simp
qed
```

```

lemma voters-determine-drop-mod:
  fixes
     $r :: 'a \text{ Preference-Relation}$  and
     $n :: \text{nat}$ 
  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
     $r :: 'a \text{ Preference-Relation}$  and
     $n :: \text{nat}$ 
  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
     $r :: 'a \text{ Preference-Relation}$  and
     $n :: \text{nat}$ 
  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  $\Rightarrow$  'a Preference-Relation  $\Rightarrow$ 
  ('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 SCF-result.electoral-module (pass-module n r)
proof (unfold 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}  $\vee$ 
    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}  $\wedge$ 
        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-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
     $n :: \text{nat}$ 
  assumes
    order: linear-order  $r$  and
    greater-zero:  $n > 0$ 
  shows non-blocking (pass-module  $n$   $r$ )
proof (unfold non-blocking-def, safe)
  show 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
     $\exists b \in A. \text{above } (\text{limit } A \ r) \ b = \{b\}$ 
     $\wedge (\forall c \in A. \text{above } (\text{limit } A \ r) \ c = \{c\} \longrightarrow c = b)$ 
    using fin-A a-in-A lin above-one
    by blast
  moreover have  $\{b \in A. \text{rank } (\text{limit } A \ r) \ b > n\} \neq A$ 
    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
   $r :: 'a \text{ Preference-Relation}$  and
   $n :: \text{nat}$ 
assumes  $\text{linear-order } r$ 
shows  $\text{non-electing } (\text{pass-module } n \ r)$ 
unfolding  $\text{non-electing-def}$ 
using  $\text{assms}$ 
by  $\text{force}$ 

```

5.10.5 Properties

The pass module is strictly defer-monotone.

theorem $\text{pass-mod-dl-inv}[\text{simp}]$:

```

fixes
   $r :: 'a \text{ Preference-Relation}$  and
   $n :: \text{nat}$ 
assumes  $\text{linear-order } r$ 
shows  $\text{defer-lift-invariance } (\text{pass-module } n \ r)$ 
unfolding  $\text{defer-lift-invariance-def}$ 
using  $\text{assms pass-mod-sound}$ 
by  $\text{simp}$ 

```

theorem $\text{pass-zero-mod-def-zero}[\text{simp}]$:

```

fixes  $r :: 'a \text{ Preference-Relation}$ 
assumes  $\text{linear-order } r$ 
shows  $\text{defers } 0 \ (\text{pass-module } 0 \ r)$ 

```

proof $(\text{unfold } \text{defers-def}, \text{ safe})$

```

show  $\text{SCF-result.electoral-module } (\text{pass-module } 0 \ r)$ 
  using  $\text{pass-mod-sound assms}$ 
  by  $\text{metis}$ 

```

next

```

fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 

```

assume

```

   $\text{card-pos: } 0 \leq \text{card } A$  and
   $\text{finite-A: finite } A$  and
   $\text{prof-A: profile } V \ A \ p$ 

```

have $\text{linear-order-on } A \ (\text{limit } A \ r)$

```

  using  $\text{assms limit-presv-lin-ord}$ 
  by  $\text{blast}$ 

```

hence $\text{limit-is-connex: connex } A \ (\text{limit } A \ r)$

```

  using  $\text{lin-ord-imp-connex}$ 
  by  $\text{simp}$ 

```

have $\forall n. (n::\text{nat}) \leq 0 \longrightarrow n = 0$

by blast

hence $\forall a \ A'. a \in A' \wedge a \in A \longrightarrow \text{connex } A' \ (\text{limit } A \ r) \longrightarrow$

$\neg \text{rank } (\text{limit } A \ r) \ a \leq 0$

```

using above-connex above-presv-limit card-eq-0-iff equals0D finite-A
      assms rev-finite-subset
unfolding rank.simps
by (metis (no-types))
hence  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 0\} = \{\}$ 
using limit-is-connex
by simp
hence  $\text{card } \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 0\} = 0$ 
using card.empty
by metis
thus  $\text{card } (\text{defer } (\text{pass-module } 0 \ r) \ V \ A \ p) = 0$ 
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  $r :: 'a \text{ Preference-Relation}$ 
  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
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
assume
  card-pos:  $1 \leq \text{card } A$  and
  finite-A: finite  $A$  and
  prof-A: profile  $V \ A \ p$ 
show  $\text{card } (\text{defer } (\text{pass-module } 1 \ r) \ V \ A \ p) = 1$ 
proof -
  have  $A \neq \{\}$ 
    using card-pos
    by auto
moreover have lin-ord-on-A: linear-order-on  $A \ (\text{limit } A \ r)$ 
    using assms limit-presv-lin-ord
    by blast
ultimately have winner-exists:
   $\exists a \in A. \text{above } (\text{limit } A \ r) \ a = \{a\} \wedge$ 
   $(\forall b \in A. \text{above } (\text{limit } A \ r) \ b = \{b\} \longrightarrow b = a)$ 
    using finite-A above-one
    by simp
then obtain  $w :: 'a$  where
   $w\text{-unique-top}$ 

```

$\text{above } (\text{limit } A \ r) \ w = \{w\} \wedge$
 $(\forall a \in A. \text{above } (\text{limit } A \ r) \ a = \{a\} \longrightarrow a = w)$
using *above-one*
by *auto*
hence $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 1\} = \{w\}$
proof
assume
 $w\text{-top}: \text{above } (\text{limit } A \ r) \ w = \{w\}$ **and**
 $w\text{-unique}: \forall a \in A. \text{above } (\text{limit } A \ r) \ a = \{a\} \longrightarrow a = w$
have $\text{rank } (\text{limit } A \ r) \ w \leq 1$
using *w-top*
by *auto*
hence $\{w\} \subseteq \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 1\}$
using *winner-exists w-unique-top*
by *blast*
moreover have $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 1\} \subseteq \{w\}$
proof
fix $a :: 'a$
assume $a\text{-in-winner-set}: a \in \{b \in A. \text{rank } (\text{limit } A \ r) \ b \leq 1\}$
hence $a\text{-in-}A: a \in A$
by *auto*
hence $\text{connex-limit}: \text{connex } A \ (\text{limit } A \ r)$
using *lin-ord-imp-connex lin-ord-on-A*
by *simp*
hence $\text{let } q = \text{limit } A \ r \text{ in } a \preceq_q a$
using *connex-limit above-connex pref-imp-in-above a-in-A*
by *metis*
hence $(a, a) \in \text{limit } A \ r$
by *simp*
hence $a\text{-above-}a: a \in \text{above } (\text{limit } A \ r) \ a$
unfolding *above-def*
by *simp*
have $\text{above } (\text{limit } A \ r) \ a \subseteq A$
using *above-presv-limit assms*
by *fastforce*
hence $\text{above-finite}: \text{finite } (\text{above } (\text{limit } A \ r) \ a)$
using *finite-A finite-subset*
by *simp*
have $\text{rank } (\text{limit } A \ r) \ a \leq 1$
using *a-in-winner-set*
by *simp*
moreover have $\text{rank } (\text{limit } A \ r) \ a \geq 1$
using *Suc-leI above-finite card-eq-0-iff equals0D neq0-conv a-above-a*
unfolding *rank.simps One-nat-def*
by *metis*
ultimately have $\text{rank } (\text{limit } A \ r) \ a = 1$
by *simp*
hence $\{a\} = \text{above } (\text{limit } A \ r) \ a$
using *a-above-a lin-ord-on-A rank-one-imp-above-one*

```

      by metis
    hence  $a = w$ 
      using  $w\text{-unique } a\text{-in-}A$ 
      by simp
    thus  $a \in \{w\}$ 
      by simp
  qed
  ultimately have  $\{w\} = \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 1\}$ 
    by auto
  thus ?thesis
    by simp
  qed
  thus  $\text{card } (\text{defer } (\text{pass-module } 1 \ r) \ V \ A \ p) = 1$ 
    by simp
  qed
qed

theorem pass-two-mod-def-two:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes  $\text{linear-order } r$ 
  shows  $\text{defers } 2 \ (\text{pass-module } 2 \ r)$ 
proof (unfold defers-def, safe)
  show  $\text{SCF-result.electoral-module } (\text{pass-module } 2 \ r)$ 
    using assms pass-mod-sound
    by metis
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
assume
   $\text{min-card-two: } 2 \leq \text{card } A$  and
   $\text{fin-A: finite } A$  and
   $\text{prof-A: profile } V \ A \ p$ 
from min-card-two
have  $\text{not-empty-A: } A \neq \{\}$ 
  by auto
moreover have  $\text{limit-A-order: linear-order-on } A \ (\text{limit } A \ r)$ 
  using limit-presv-lin-ord assms
  by auto
ultimately obtain  $a :: 'a$  where
   $\text{above } (\text{limit } A \ r) \ a = \{a\}$ 
  using above-one min-card-two fin-A prof-A
  by blast
hence  $\forall b \in A. \text{let } q = \text{limit } A \ r \text{ in } (b \preceq_q a)$ 
  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 $a\text{-best}$: $\forall b \in A. (b, a) \in \text{limit } A \ r$
by *simp*
hence $a\text{-above}$: $\forall b \in A. a \in \text{above } (\text{limit } A \ r) \ b$
unfolding *above-def*
by *simp*
hence $a \in \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq 2\}$
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 $a\text{-in-defer}$: $a \in \text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p$
by *simp*
have $\text{finite } (A - \{a\})$
using *fin-A*
by *simp*
moreover have $A\text{-not-only-a}$: $A - \{a\} \neq \{\}$
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 $\text{limit-A-without-a-order}$:
 $\text{linear-order-on } (A - \{a\}) \ (\text{limit } (A - \{a\}) \ r)$
using *limit-presv-lin-ord assms top-greatest*
by *blast*
ultimately obtain $b :: 'a$ **where**
 top-b : $\text{above } (\text{limit } (A - \{a\}) \ r) \ b = \{b\}$
using *above-one*
by *metis*
hence $\forall c \in A - \{a\}. \text{let } q = \text{limit } (A - \{a\}) \ r \text{ in } (c \preceq_q b)$
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 $b\text{-in-limit}$: $\forall c \in A - \{a\}. (c, b) \in \text{limit } (A - \{a\}) \ r$
by *simp*
hence $b\text{-best}$: $\forall c \in A - \{a\}. (c, b) \in \text{limit } A \ r$
by *auto*
hence $\forall c \in A - \{a, b\}. c \notin \text{above } (\text{limit } A \ r) \ b$
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 : $\text{above } (\text{limit } A \ r) \ b \subseteq A$
using *above-presv-limit assms*
by *metis*
moreover have $b\text{-above-b}$: $b \in \text{above } (\text{limit } A \ r) \ b$
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 : $\text{above } (\text{limit } A \ r) \ b = \{a, b\}$
using *a-above*
by *auto*

```

hence card-above-b-eq-two:  $\text{rank } (\text{limit } A \ r) \ b = 2$ 
  using A-not-only-a b-in-limit
  by auto
hence b-in-defer:  $b \in \text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p$ 
  using b-above-b above-subset
  by auto
have b-above:  $\forall \ c \in A - \{a\}. \ b \in \text{above } (\text{limit } A \ r) \ c$ 
  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  $\forall \ c \in A. \ c \in \text{above } (\text{limit } A \ r) \ c$ 
  using above-connex
  by metis
hence  $\forall \ c \in A - \{a, b\}. \ \{a, b, c\} \subseteq \text{above } (\text{limit } A \ r) \ c$ 
  using a-above b-above
  by auto
moreover have  $\forall \ c \in A - \{a, b\}. \ \text{card } \{a, b, c\} = 3$ 
  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  $\forall \ c \in A - \{a, b\}. \ \text{rank } (\text{limit } A \ r) \ c \geq 3$ 
  using card-mono fin-A finite-subset above-presv-limit assms
  unfolding rank.simps
  by metis
hence  $\forall \ c \in A - \{a, b\}. \ \text{rank } (\text{limit } A \ r) \ c > 2$ 
  using Suc-le-eq Suc-1 numeral-3-eq-3
  unfolding One-nat-def
  by metis
hence  $\forall \ c \in A - \{a, b\}. \ c \notin \text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p$ 
  by (simp add: not-le)
moreover have  $\text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p \subseteq A$ 
  by auto
ultimately have  $\text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p \subseteq \{a, b\}$ 
  by blast
hence  $\text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p = \{a, b\}$ 
  using a-in-defer b-in-defer
  by fastforce
thus  $\text{card } (\text{defer } (\text{pass-module } 2 \ r) \ V \ A \ p) = 2$ 
  using above-b-eq-ab card-above-b-eq-two
  unfolding rank.simps
  by presburger
qed

end

```

5.11 Elect Module

```
theory Elect-Module
  imports Component-Types/Electoral-Module
begin
```

The elect module is not concerned about the voter's ballots, and just elects all alternatives. It is primarily used in sequence after an electoral module that only defers alternatives to finalize the decision, thereby inducing a proper voting rule in the social choice sense.

5.11.1 Definition

```
fun elect-module :: ('a, 'v, 'a Result) Electoral-Module where
  elect-module V A p = (A, {}, {})
```

5.11.2 Soundness

```
theorem elect-mod-sound[simp]: 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 ∈ A. ∃ x ∈ A. *win-count* V p x > *win-count* V p a},
 {a ∈ A. ∀ x ∈ A. *win-count* V p x ≤ *win-count* V p a})
 else ({}, {}, A))

lemma *enat-leq-enat-set-max*:

fixes

x :: *enat* **and**

X :: *enat set*

assumes

x ∈ *X* **and**

finite *X*

shows *x* ≤ *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'.sims*)

have *no-winners-or-in-domain*:

finite {*win-count* V p a | a. a ∈ A} ⟶

{*win-count* V p a | a. a ∈ A} = {} ∨

Max {*win-count* V p a | a. a ∈ A} ∈ {*win-count* V p a | a. a ∈ A}

using *Max-in*

by *blast*

moreover **have** *only-one-max*:

finite A ⟶

{a ∈ A. ∃ x ∈ A. *win-count* V p a < *win-count* V p x} =

{a ∈ A. *win-count* V p a < *Max* {*win-count* V p x | x. x ∈ A}}

proof

assume *fin-A*: *finite* A

hence $\text{finite } \{ \text{win-count } V p x \mid x. x \in A \}$
by *simp*
hence
 $\forall a \in A. \forall b \in A. \text{win-count } V p a < \text{win-count } V p b \longrightarrow$
 $\text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \}$
using *CollectI Max-gr-iff empty-Collect-eq*
by (*metis (mono-tags, lifting)*)
hence
 $\{ a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x \}$
 $\subseteq \{ a \in A. \text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \} \}$
by *blast*
moreover have
 $\{ a \in A. \text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \} \}$
 $\subseteq \{ a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x \}$
using *fin-A no-winners-or-in-domain*
by *fastforce*
ultimately show
 $\{ a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x \} =$
 $\{ a \in A. \text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \} \}$
by *fastforce*
qed
ultimately have
 $\text{finite } A \longrightarrow$
 $\text{reject plurality } V A p = \{ a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p$
 $x \}$
by *force*
moreover have
 $\text{finite } A \longrightarrow$
 $\text{defer plurality } V A p = \{ a \in A. \forall b \in A. \text{win-count } V p b \leq \text{win-count } V p$
 $a \}$
proof
assume *fin-A: finite A*
have $\{ a \in A. \exists x \in A. \text{win-count } V p x > \text{win-count } V p a \}$
 $\cap \{ a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a \} = \{ \}$
by *force*
moreover have
 $\{ a \in A. \exists x \in A. \text{win-count } V p x > \text{win-count } V p a \}$
 $\cup \{ a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a \} = A$
by *force*
ultimately have
 $\{ a \in A. \forall b \in A. \text{win-count } V p b \leq \text{win-count } V p a \} =$
 $A - \{ a \in A. \text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \} \}$
using *fin-A only-one-max Diff-cancel Int-Diff-Un Un-Diff inf-commute*
by (*metis (no-types, lifting)*)
moreover have
 $\{ a \in A. \text{win-count } V p a < \text{Max } \{ \text{win-count } V p x \mid x. x \in A \} \} = A \longrightarrow$
 $\{ a \in A. \forall b \in A. \text{win-count } V p b \leq \text{win-count } V p a \} = A$
using *fin-A no-winners-or-in-domain*
by *fastforce*

```

ultimately show
  defer plurality V A p = {a ∈ A. ∀ b ∈ A. win-count V p b ≤ 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 →
  plurality V A p =
    ({}, {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})
  using split-pairs
  by (metis (no-types, lifting))
thus plurality V A p =
  (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
qed

```

5.12.2 Soundness

```

theorem plurality-sound[simp]: SCF-result.electoral-module plurality
  unfolding plurality.simps
  using max-elim-sound
  by metis

```

```

theorem plurality'-sound[simp]: SCF-result.electoral-module plurality'

```

```

proof (unfold SCF-result.electoral-module.simps, safe)

```

```

  fix

```

```

    A :: 'a set and

```

```

    V :: 'v set and

```

```

    p :: ('a, 'v) Profile

```

```

  have disjoint3 (

```

```

    {},

```

```

    {a ∈ A. ∃ a' ∈ A. win-count V p a < win-count V p a'},

```

```

    {a ∈ A. ∀ a' ∈ A. win-count V p a' ≤ win-count V p a})

```

```

  by auto

```

```

  moreover have

```

```

    {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} = A

```

```

  using not-le-imp-less

```

```

  by blast

```

```

ultimately show well-formed-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

$A :: 'b \text{ set}$ **and**

$V :: 'a \text{ set}$ **and**

$p \ p' :: ('b, 'a) \text{ Profile}$ **and**

$a :: 'b$

assume

$\forall v \in V. p \ v = p' \ v$ **and**

$a \in A$

hence *finite* $V \longrightarrow$

$\text{card } \{v \in V. \text{above } (p \ v) \ a = \{a\}\} = \text{card } \{v \in V. \text{above } (p' \ v) \ a = \{a\}\}$

using *Collect-cong*

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

thus *win-count* $V \ p \ a = \text{win-count } V \ p' \ a$

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

$A :: 'k \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p \ p' :: ('k, 'v) \text{ Profile}$

assume $\forall v \in V. p \ v = p' \ v$

thus *plurality'* $V \ A \ p = \text{plurality}' \ V \ A \ p'$

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

$A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p \ q :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$

assumes

defer-a: $a \in \text{defer } \text{plurality } V \ A \ p$ **and**
lift-a: *lifted* $V \ A \ p \ q \ a$

shows $\text{defer } \text{plurality } V \ A \ q = \text{defer } \text{plurality } V \ A \ p$
 $\vee \text{defer } \text{plurality } V \ A \ q = \{a\}$

proof –

have *set-disj*: $\forall \ b \ c. \ b \notin \{c\} \vee b = c$
by *blast*

have *lifted-winner*: $\forall \ b \in A. \forall \ i \in V.$
 $\text{above } (p \ i) \ b = \{b\} \longrightarrow (\text{above } (q \ i) \ b = \{b\} \vee \text{above } (q \ i) \ a = \{a\})$
using *lift-a lifted-above-winner-alts*
unfolding *Profile.lifted-def*
by *metis*

hence $\forall \ i \in V. (\text{above } (p \ i) \ a = \{a\} \longrightarrow \text{above } (q \ i) \ a = \{a\})$
using *defer-a lift-a*
unfolding *Profile.lifted-def*
by *metis*

hence *a-win-subset*:

$\{i \in V. \text{above } (p \ i) \ a = \{a\}\} \subseteq \{i \in V. \text{above } (q \ i) \ a = \{a\}\}$
by *blast*

moreover have *lifted-prof*: *profile* $V \ A \ q$

using *lift-a*
unfolding *Profile.lifted-def*
by *metis*

ultimately have *win-count-a*: $\text{win-count } V \ p \ a \leq \text{win-count } V \ q \ a$

```

  by (simp add: card-mono)
have fin-A: finite A
  using lift-a
  unfolding Profile.lifted-def
  by blast
hence  $\forall b \in A - \{a\}$ .
   $\forall i \in V. (\text{above } (q \ i) \ a = \{a\} \longrightarrow \text{above } (q \ i) \ b \neq \{b\})$ 
  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:
   $\forall b \in A - \{a\}$ .
   $\forall i \in V. (\text{above } (q \ i) \ b = \{b\} \longrightarrow \text{above } (p \ i) \ b = \{b\})$ 
  using lifted-above-winner-other lift-a
  unfolding Profile.lifted-def
  by metis
hence  $\forall b \in A - \{a\}$ .
   $\{i \in V. \text{above } (q \ i) \ b = \{b\}\} \subseteq \{i \in V. \text{above } (p \ i) \ b = \{b\}\}$ 
  by (simp add: Collect-mono)
hence win-count-other:  $\forall b \in A - \{a\}. \text{win-count } V \ p \ b \geq \text{win-count } V \ q \ b$ 
  by (simp add: card-mono)
show defer plurality V A q = defer plurality V A p
   $\vee \text{defer plurality } V \ A \ q = \{a\}$ 
proof (cases)
  assume win-count V p a = win-count V q a
  hence card  $\{i \in V. \text{above } (p \ i) \ a = \{a\}\} = \text{card } \{i \in V. \text{above } (q \ i) \ a = \{a\}\}$ 
    using win-count.simps Profile.lifted-def enat.inject lift-a
    by (metis (mono-tags, lifting))
  moreover have finite  $\{i \in V. \text{above } (q \ i) \ a = \{a\}\}$ 
    using Collect-mem-eq Profile.lifted-def finite-Collect-conjI lift-a
    by (metis (mono-tags))
  ultimately have  $\{i \in V. \text{above } (p \ i) \ a = \{a\}\} = \{i \in V. \text{above } (q \ i) \ a = \{a\}\}$ 
    using a-win-subset
    by (simp add: card-subset-eq)
  hence above-pq:  $\forall i \in V. (\text{above } (p \ i) \ a = \{a\}) = (\text{above } (q \ i) \ a = \{a\})$ 
    by blast
  moreover have
     $\forall b \in A - \{a\}. \forall i \in V.$ 
     $(\text{above } (p \ i) \ b = \{b\} \longrightarrow (\text{above } (q \ i) \ b = \{b\} \vee \text{above } (q \ i) \ a = \{a\}))$ 
    using lifted-winner
    by auto
  moreover have
     $\forall b \in A - \{a\}. \forall i \in V. (\text{above } (p \ i) \ b = \{b\} \longrightarrow \text{above } (p \ i) \ a \neq \{a\})$ 
  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 : $\text{above } (p \ i) \ b = \{b\}$ and
 abv - a : $\text{above } (p \ i) \ a = \{a\}$
ultimately show *False*
using *above-one-eq A-not-empty fin-A*
by (*metis (no-types)*)
qed
ultimately have *above-PtoQ*:
 $\forall b \in A - \{a\}. \forall i \in V. (\text{above } (p \ i) \ b = \{b\} \longrightarrow \text{above } (q \ i) \ b = \{b\})$
by *simp*
hence $\forall b \in A.$
 $\text{card } \{i \in V. \text{above } (p \ i) \ b = \{b\}\} =$
 $\text{card } \{i \in V. \text{above } (q \ i) \ b = \{b\}\}$
proof (*safe*)
fix $b :: 'a$
assume $b \in A$
thus $\text{card } \{i \in V. \text{above } (p \ i) \ b = \{b\}\} =$
 $\text{card } \{i \in V. \text{above } (q \ i) \ b = \{b\}\}$
using *DiffI set-disj above-PtoQ above-QtoP above-pq*
by (*metis (no-types, lifting)*)
qed
hence $\{b \in A. \forall c \in A. \text{win-count } V \ p \ c \leq \text{win-count } V \ p \ b\} =$
 $\{b \in A. \forall c \in A. \text{win-count } V \ q \ c \leq \text{win-count } V \ q \ b\}$
by *auto*
hence *defer plurality' V A q = defer plurality' V A p*
 $\vee \text{defer plurality' V A q} = \{a\}$
by *simp*
hence *defer plurality V A q = defer plurality V A p*
 $\vee \text{defer plurality V A q} = \{a\}$
using *plurality-mod-equiv lift-a*
unfolding *Profile.lifted-def*
by (*metis (no-types, opaque-lifting)*)
thus *?thesis*
by *simp*
next
assume $\text{win-count } V \ p \ a \neq \text{win-count } V \ q \ a$
hence *strict-less: win-count V p a < win-count V q a*
using *win-count-a*
by *simp*
have $a \in \text{defer plurality V A p}$

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-profile* $V A p$
 using *lift-a*
 unfolding *Profile.lifted-def*
 by *simp*
 ultimately have $a \in \text{defer plurality}' V A p$
 using *plurality-mod-equiv*
 by *metis*
 hence *a-in-win-p*:
 $a \in \{b \in A. \forall c \in A. \text{win-count } V p c \leq \text{win-count } V p b\}$
 using *fin-A*
 by *simp*
 hence $\forall b \in A. \text{win-count } V p b \leq \text{win-count } V p a$
 by *simp*
 hence *less*: $\forall b \in A - \{a\}. \text{win-count } V q b < \text{win-count } V q a$
 using *DiffD1 antisym dual-order.trans not-le-imp-less*
 win-count-a strict-less win-count-other
 by *metis*
 hence $\forall b \in A - \{a\}. \neg (\forall c \in A. \text{win-count } V q c \leq \text{win-count } V q b)$
 using *lift-a not-le*
 unfolding *Profile.lifted-def*
 by *metis*
 hence $\forall b \in A - \{a\}. b \notin \{c \in A. \forall b \in A. \text{win-count } V q b \leq \text{win-count } V q c\}$
 by *blast*
 hence $\forall b \in A - \{a\}. b \notin \text{defer plurality}' V A q$
 using *fin-A*
 by *simp*
 hence $\forall b \in A - \{a\}. b \notin \text{defer plurality } V A q$
 using *lift-a non-empty-A plurality-mod-equiv*
 unfolding *Profile.lifted-def*
 by (*metis (no-types, lifting)*)
 hence $\forall b \in A - \{a\}. b \notin \text{defer plurality } V A q$
 by *simp*
 moreover have $a \in \text{defer plurality } V A q$
 proof –
 have $\forall b \in A - \{a\}. \text{win-count } V q b \leq \text{win-count } V q a$
 using *less less-imp-le*
 by *metis*
 moreover have $\text{win-count } V q a \leq \text{win-count } V q a$
 by *simp*
 ultimately have $\forall b \in A. \text{win-count } V q b \leq \text{win-count } V q a$
 by *auto*
 moreover have $a \in A$


```

    using a-in-win-p
    by simp
  ultimately have
     $a \in \{b \in A. \forall c \in A. \text{win-count } V \ q \ c \leq \text{win-count } V \ q \ b\}$ 
    by simp
  hence  $a \in \text{defer plurality}' \ V \ A \ q$ 
    by simp
  hence  $a \in \text{defer plurality} \ V \ A \ q$ 
    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  $\text{defer plurality} \ V \ A \ q \subseteq A$ 
  by simp
ultimately show ?thesis
  by blast
qed
qed

```

lemma *plurality'-def-inv-mono-alts*:

```

  fixes
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p \ q :: ('a, 'v) \text{ Profile}$  and
     $a :: 'a$ 
  assumes
     $a \in \text{defer plurality}' \ V \ A \ p$  and
     $\text{lifted } V \ A \ p \ q \ a$ 
  shows  $\text{defer plurality}' \ V \ A \ q = \text{defer plurality}' \ V \ A \ p$ 
     $\vee \text{defer plurality}' \ V \ A \ q = \{a\}$ 
  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 SCF-result.electoral-module plurality
    using plurality-sound
    by metis

```

next

```

  show non-electing plurality
    by simp

```

next

```

  fix
     $A :: 'b \text{ set}$  and
     $V :: 'a \text{ set}$  and
     $p \ q :: ('b, 'a) \text{ Profile}$  and

```

```

  a :: 'b
assume a ∈ defer plurality V A p ∧ Profile.lifted V A p q a
hence defer plurality V A q = defer plurality V A p
      ∨ 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
      ∨ 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 = (∑ y ∈ A. (prefer-count V p x y))

```

```

fun borda :: ('a, 'v, 'a Result) Electoral-Module where
  borda V A p = max-eliminator borda-score V A p

```

5.13.2 Soundness

```

theorem borda-sound: SCF-result.electoral-module borda

```

```

unfolding borda.simps
using max-elim-sound
by metis

```

5.13.3 Non-Blocking

The Borda module is non-blocking.

```

theorem borda-mod-non-blocking[simp]: non-blocking borda
  unfolding borda.simps
  using max-elim-non-blocking
  by metis

```

5.13.4 Non-Electing

The Borda module is non-electing.

```

theorem borda-mod-non-electing[simp]: non-electing borda
  using max-elim-non-electing
  unfolding borda.simps non-electing-def
  by metis

```

end

5.14 Condorcet Module

```

theory Condorcet-Module
  imports Component-Types/Elimination-Module
begin

```

This is the Condorcet module used by the Condorcet (voting) rule. The Condorcet rule is a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.14.1 Definition

```

fun condorcet-score :: ('a, 'v) Evaluation-Function where
  condorcet-score V x A p =
    (if (condorcet-winner V A p x) then 1 else 0)

fun condorcet :: ('a, 'v, 'a Result) Electoral-Module where
  condorcet V A p = (max-eliminator condorcet-score) V A p

```

5.14.2 Soundness

theorem *condorcet-sound: SCF-result.electoral-module condorcet*
unfolding *condorcet.simps*
using *max-elim-sound*
by *metis*

5.14.3 Property

theorem *condorcet-score-is-condorcet-rating: condorcet-rating condorcet-score*

proof (*unfold condorcet-rating-def, safe*)

fix

A :: 'b set **and**

V :: 'a set **and**

p :: ('b, 'a) Profile **and**

w l :: 'b

assume

c-win: *condorcet-winner V A p w* **and**

l-neq-w: *l* ≠ *w*

have ¬ *condorcet-winner V A p l*

using *cond-winner-unique-eq c-win l-neq-w*

by *metis*

thus *condorcet-score V l A p* < *condorcet-score V w A p*

using *c-win zero-less-one*

unfolding *condorcet-score.simps*

by (*metis (full-types)*)

qed

theorem *condorcet-is-dcc: defer-condorcet-consistency condorcet*

proof (*unfold defer-condorcet-consistency-def SCF-result.electoral-module.simps, safe*)

fix

A :: 'b set **and**

V :: 'a set **and**

p :: ('b, 'a) Profile

assume *profile V A p*

hence *well-formed-SCF A (max-eliminator condorcet-score V A p)*

using *max-elim-sound*

unfolding *SCF-result.electoral-module.simps*

by *metis*

thus *well-formed-SCF A (condorcet V A p)*

by *simp*

next

fix

A :: 'b set **and**

V :: 'a set **and**

p :: ('b, 'a) Profile **and**

a :: 'b

assume *c-win-w: condorcet-winner V A p a*

let ?*m* = (*max-eliminator condorcet-score*) :: ('b, 'a, 'b Result) Electoral-Module

```

have defer-condorcet-consistency ?m
  using cr-eval-imp-dcc-max-elim condorcet-score-is-condorcet-rating
  by metis
hence ?m V A p =
  ({}, A - defer ?m V A p, {b ∈ A. condorcet-winner V A p b})
  using c-win-w
  unfolding defer-condorcet-consistency-def
  by (metis (no-types))
thus condorcet V A p =
  ({},
   A - defer condorcet V A p,
   {d ∈ A. condorcet-winner V A p d})
  by simp
qed

end

```

5.15 Copeland Module

```

theory Copeland-Module
  imports Component-Types/Elimination-Module
begin

```

This is the Copeland module used by the Copeland voting rule. The Copeland rule elects the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.15.1 Definition

```

fun copeland-score :: ('a, 'v) Evaluation-Function where
  copeland-score V x A p =
    card {y ∈ A . wins V x p y} - card {y ∈ A . wins V y p x}

fun copeland :: ('a, 'v, 'a Result) Electoral-Module where
  copeland V A p = max-eliminator copeland-score V A p

```

5.15.2 Soundness

```

theorem copeland-sound: SCF-result.electoral-module copeland
  unfolding copeland.simps
  using max-elim-sound
  by metis

```

5.15.3 Lemmas

lemma *voters-determine-copeland-score: voters-determine-evaluation copeland-score*

proof (*unfold copeland-score.simps voters-determine-evaluation.simps, safe*)

fix

$A :: 'b \text{ set}$ **and**

$V :: 'a \text{ set}$ **and**

$p \ p' :: ('b, 'a) \text{ Profile}$ **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 $\forall x \ y.$

$\text{card } \{y \in A. \text{wins } V \ x \ p \ y\} = \text{card } \{y \in A. \text{wins } V \ x \ p' \ y\}$

$\wedge \text{card } \{x \in A. \text{wins } V \ x \ p \ y\} = \text{card } \{x \in A. \text{wins } V \ x \ p' \ y\}$

by *simp*

thus $\text{card } \{y \in A. \text{wins } V \ a \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ a\} =$

$\text{card } \{y \in A. \text{wins } V \ a \ p' \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p' \ a\}$

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: $|\{y \in A. \text{wins } V \ w \ p \ y\}| = |A| - 1$.

lemma *cond-winner-imp-win-count:*

fixes

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$ **and**

$w :: 'a$

assumes *condorcet-winner* $V \ A \ p \ w$

shows $\text{card } \{a \in A. \text{wins } V \ w \ p \ a\} = \text{card } A - 1$

proof –

have $\forall a \in A - \{w\}. \text{wins } V \ w \ p \ a$

using *assms*

by *auto*

hence $\{a \in A - \{w\}. \text{wins } V \ w \ p \ a\} = A - \{w\}$

by *blast*

hence *winner-wins-against-all-others:*

$\text{card } \{a \in A - \{w\}. \text{wins } V \ w \ p \ a\} = \text{card } (A - \{w\})$

by *simp*

have $w \in A$

using *assms*

by *simp*

```

hence  $\text{card } (A - \{w\}) = \text{card } A - 1$ 
  using card-Diff-singleton assms
  by metis
hence winner-amount-one:  $\text{card } \{a \in A - \{w\}. \text{ wins } V w p a\} = \text{card } (A) - 1$ 
  using winner-wins-against-all-others
  by linarith
have win-for-winner-not-reflexive:  $\forall a \in \{w\}. \neg \text{ wins } V a p a$ 
  by (simp add: wins-irreflex)
hence  $\{a \in \{w\}. \text{ wins } V w p a\} = \{\}$ 
  by blast
hence winner-amount-zero:  $\text{card } \{a \in \{w\}. \text{ wins } V w p a\} = 0$ 
  by simp
have union:
   $\{a \in A - \{w\}. \text{ wins } V w p a\} \cup \{x \in \{w\}. \text{ wins } V w p x\} =$ 
   $\{a \in A. \text{ wins } V w p a\}$ 
  using win-for-winner-not-reflexive
  by blast
have finite-defeated:  $\text{finite } \{a \in A - \{w\}. \text{ wins } V w p a\}$ 
  using assms
  by simp
have  $\text{finite } \{a \in \{w\}. \text{ wins } V w p a\}$ 
  by simp
hence  $\text{card } (\{a \in A - \{w\}. \text{ wins } V w p a\} \cup \{a \in \{w\}. \text{ wins } V w p a\}) =$ 
 $\text{card } \{a \in A - \{w\}. \text{ wins } V w p a\} + \text{card } \{a \in \{w\}. \text{ wins } V w p a\}$ 
  using finite-defeated card-Un-disjoint
  by blast
hence  $\text{card } \{a \in A. \text{ wins } V w p a\} =$ 
 $\text{card } \{a \in A - \{w\}. \text{ wins } V w p a\} + \text{card } \{a \in \{w\}. \text{ wins } V w p a\}$ 
  using union
  by simp
thus ?thesis
  using winner-amount-one winner-amount-zero
  by linarith
qed

```

For a Condorcet winner w , we have: $|\{y \in A . \text{ wins } V y p w\}| = 0$.

lemma *cond-winner-imp-loss-count*:

```

fixes
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $w :: 'a$ 
assumes condorcet-winner  $V A p w$ 
shows  $\text{card } \{a \in A. \text{ wins } V a p w\} = 0$ 
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
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  and
     $w :: 'a$ 
  assumes condorcet-winner  $V A p w$ 
  shows copeland-score  $V w A p = \text{card } A - 1$ 
proof (unfold copeland-score.simps)
  have  $\text{card } \{a \in A. \text{wins } V w p a\} = \text{card } A - 1$ 
    using cond-winner-imp-win-count assms
    by metis
  moreover have  $\text{card } \{a \in A. \text{wins } V a p w\} = 0$ 
    using cond-winner-imp-loss-count assms
    by (metis (no-types))
  ultimately show
     $\text{enat } (\text{card } \{a \in A. \text{wins } V w p a\} - \text{card } \{a \in A. \text{wins } V a p w\}) = \text{enat } (\text{card } A - 1)$ 
    by simp
qed

```

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

```

lemma non-cond-winner-imp-win-count:
  fixes
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  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  $\text{card } \{a \in A . \text{wins } V l p a\} \leq \text{card } A - 2$ 
proof -
  have  $\text{wins } V w p l$ 
    using assms
    by auto
  hence  $\neg \text{wins } V l p w$ 
    using wins-antisym
    by simp
  moreover have  $\neg \text{wins } V l p l$ 
    using wins-irreflex
    by simp
  ultimately have wins-of-loser-eq-without-winner:
     $\{y \in A . \text{wins } V l p y\} = \{y \in A - \{l, w\} . \text{wins } V l p y\}$ 
    by blast
  have  $\forall M f. \text{finite } M \longrightarrow \text{card } \{x \in M . f x\} \leq \text{card } M$ 
    by (simp add: card-mono)

```



```

moreover have finite ( $A - \{l, w\}$ )
  using finite-Diff winner
  by simp
ultimately have  $\text{card } \{y \in A - \{l, w\} . \text{wins } V \ l \ p \ y\} \leq \text{card } (A - \{l, w\})$ 
  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

```

   $A :: 'b \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('b, 'v) \text{ Profile}$  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 $\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} \leq \text{card } A - 2$

```

  using non-cond-winner-imp-win-count
  by (metis (mono-tags, lifting))

```

hence $\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ l\} \leq \text{card } A - 2$

```

  using diff-le-self order.trans
  by simp

```

moreover have $\text{card } A - 2 < \text{card } A - 1$

```

  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

```

   $\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ l\} < \text{card } A - 1$ 
  using order-le-less-trans
  by fastforce

```

moreover have $\text{card } \{a \in A. \text{wins } V \ a \ p \ w\} = 0$

```

  using cond-winner-imp-loss-count winner
  by metis

```

moreover have $\text{card } A - 1 = \text{card } \{a \in A. \text{wins } V \ w \ p \ a\}$

```

  using cond-winner-imp-win-count winner
  by (metis (full-types))

```

ultimately show

```

   $\text{enat } (\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ l\}) <$ 

```

```

      enat (card {y ∈ A. wins V w p y} - card {y ∈ 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 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-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 $\forall E n. \text{infinite } E \longrightarrow (\exists e. \neg e \leq \text{enat } n \wedge e \in E)$
 using *finite-enat-bounded*
 by *blast*
ultimately show *finite* $\{\text{prefer-count } V p l y \mid y. y \in A - \{l\}\}$
 using *pref-count-voter-set-card*
 by *fastforce*
next
 have $w \in A$
 using *winner*
 by *simp*
thus $\text{prefer-count } V p l w \in \{\text{prefer-count } V p l y \mid y. y \in A - \{l\}\}$
 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

$A :: 'b \text{ set}$ **and**
 $V :: 'a \text{ set}$ **and**
 $p :: ('b, 'a) \text{ Profile}$ **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:
 $\neg \text{Min } \{\text{if finite } V$
 $\quad \text{then enat } (\text{card } \{v \in V. \text{let } r = p v \text{ in } y \preceq_r l\})$
 $\quad \text{else } \infty \mid y. y \in A - \{l\}\}$
 $< \text{Min } \{\text{if finite } V$
 $\quad \text{then enat } (\text{card } \{v \in V. \text{let } r = p v \text{ in } y \preceq_r w\})$
 $\quad \text{else } \infty \mid y. y \in A - \{w\}\}$

hence *min-count-ineq:*

$\text{Min } \{\text{prefer-count } V p l y \mid y. y \in A - \{l\}\} \geq$
 $\text{Min } \{\text{prefer-count } V p w y \mid y. y \in A - \{w\}\}$

by *simp*

have *pref-count-gte-min:*

$\text{prefer-count } V p l w \geq \text{Min } \{\text{prefer-count } V p l y \mid y. y \in A - \{l\}\}$

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:  $\{\text{prefer-count } V p w y \mid y . y \in A - \{w\}\} \neq \{\}$ 
  by blast
have finite  $(A - \{w\})$ 
  using condorcet-winner.simps winner finite-Diff
  by metis
hence finite  $\{\text{prefer-count } V p w y \mid y . y \in A - \{w\}\}$ 
  by simp
hence  $\exists n \in A - \{w\} . \text{prefer-count } V p w n =$ 
   $\text{Min } \{\text{prefer-count } V p w y \mid y . y \in A - \{w\}\}$ 
  using pref-counts-non-empty Min-in
  by fastforce
then obtain  $n :: 'b$  where
  pref-count-eq-min:
  prefer-count  $V p w n =$ 
   $\text{Min } \{\text{prefer-count } V p w y \mid y . y \in A - \{w\}\}$  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  $V p w n > \text{prefer-count } V p n w$ 
  using n-not-w winner
  by auto
have pref-count-l-w-n-ineq: prefer-count  $V p l w \geq \text{prefer-count } V p w n$ 
  using pref-count-gte-min min-count-ineq pref-count-eq-min
  by auto
hence prefer-count  $V p n w \geq \text{prefer-count } V p w l$ 
  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  $V p l w > \text{prefer-count } V p w l$ 
  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 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-SCF A (max-eliminator minimax-score V A p)
    using max-elim-sound par-comp-result-sound
    by metis
  thus well-formed-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  $r :: 'a \text{ Preference-Relation}$ 
  assumes  $\text{linear-order } r$ 
  shows  $\text{rejects } 0 \ (\text{drop-module } 0 \ r)$ 
proof (unfold rejects-def, safe)
  show  $\text{SCF-result.electoral-module } (\text{drop-module } 0 \ r)$ 
    using  $\text{assms drop-mod-sound}$ 
    by metis
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
assume
   $\text{fin-}A: \text{finite } A$  and
   $\text{prof-}A: \text{profile } V \ A \ p$ 
have  $\text{connex UNIV } r$ 
  using  $\text{assms lin-ord-imp-connex}$ 
  by auto
hence  $\text{connex: connex } A \ (\text{limit } A \ r)$ 
  using  $\text{limit-presv-connex subset-UNIV}$ 
  by metis
have  $\forall B \ a. B \neq \{\} \vee (a :: 'a) \notin B$ 
  by simp
hence  $\forall a \ B. a \in A \wedge a \in B \longrightarrow \text{connex } B \ (\text{limit } A \ r) \longrightarrow$ 
```

```

      ¬ 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 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 $\forall a \in A. \text{card } \{a\} \leq \text{card } (\text{above } (\text{limit } A \ r) \ a)$
 using *card-mono fin-A rev-finite-subset above-presv-limit*
 by *metis*
 hence *rank-geq-one*: $\forall a \in A. 1 \leq \text{rank } (\text{limit } A \ r) \ a$
 by *simp*
 with *leq* have $\forall a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ \text{card } A\}$
 by *simp*
 hence $\text{rank } (\text{limit } A \ r) \ 'A \subseteq \{1 \ .. \ \text{card } A\}$
 by *auto*
 moreover have *inj*: $\text{inj-on } (\text{rank } (\text{limit } A \ r)) \ A$
 using *fin-A inj-onI rank-unique lin-ord-limit*
 by *metis*
 ultimately have *bij-A*: $\text{bij-betw } (\text{rank } (\text{limit } A \ r)) \ A \ \{1 \ .. \ \text{card } A\}$
 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*: $\text{bij-betw } ?\text{inv-rank } \{1 \ .. \ \text{card } A\} \ A$
 using *bij-betw-the-inv-into*
 by *blast*
 hence $\forall S \subseteq \{1 \ .. \ \text{card } A\}. \text{card } (? \text{inv-rank } 'S) = \text{card } S$
 using *fin-A bij-betw-same-card bij-betw-subset*
 by *metis*
 moreover have *subset*: $\{1 \ .. \ n\} \subseteq \{1 \ .. \ \text{card } A\}$
 using *card-n*
 by *simp*
 ultimately have $\text{card } (? \text{inv-rank } ' \{1 \ .. \ n\}) = n$
 using *numeral-One numeral-eq-iff card-atLeastAtMost diff-Suc-1*
 by *presburger*
 also have $? \text{inv-rank } ' \{1 \ .. \ n\} = \{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\}$
 proof
 show $? \text{inv-rank } ' \{1 \ .. \ n\} \subseteq \{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\}$
 proof
 fix $a :: 'a$
 assume $a \in ? \text{inv-rank } ' \{1 \ .. \ n\}$
 then obtain $b :: \text{nat}$ where
 $b \text{-img}: b \in \{1 \ .. \ n\} \wedge ? \text{inv-rank } b = a$
 by *auto*
 hence $\text{rank } (\text{limit } A \ r) \ a = b$
 using *subset f-the-inv-into-f-bij-betw subsetD bij-A*
 by *metis*
 hence $\text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}$
 using *b-img*
 by *simp*
 moreover have $a \in A$
 using *b-img bij-inv bij-betwE subset*
 by *blast*
 ultimately show $a \in \{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\}$
 by *blast*

```

qed
next
show  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\}$ 
 $\subseteq \text{the-inv-into } A \ (\text{rank } (\text{limit } A \ r)) \ ' \{1 \ .. \ n\}$ 
proof
  fix a :: 'a
  assume el:  $a \in \{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\}$ 
  then obtain b :: nat where
    b-img:  $b \in \{1..n\} \wedge \text{rank } (\text{limit } A \ r) \ a = b$ 
  by auto
  moreover have  $a \in A$ 
  using el
  by simp
  ultimately have ?inv-rank b = a
  using inj the-inv-into-f-f
  by metis
  thus  $a \in ?inv\text{-rank } ' \{1 \ .. \ n\}$ 
  using b-img
  by auto
qed
qed
finally have  $\text{card } \{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1..n\}\} = n$ 
by blast
also have  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \in \{1 \ .. \ n\}\} =$ 
 $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\}$ 
using rank-geq-one
by auto
also have  $\dots = \text{reject } (\text{drop-module } n \ r) \ V \ A \ p$ 
by simp
finally show  $\text{card } (\text{reject } (\text{drop-module } n \ r) \ V \ A \ p) = n$ 
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
   $A :: 'a \text{ set}$  and
   $V :: 'b \text{ set}$ 
have linear-order-on  $A$  (limit  $A$   $r$ )
  using assms limit-presv-lin-ord
  by blast
hence profile  $V$   $A$  ( $\lambda v. (\text{limit } A \ r)$ )
  using profile-def
  by blast
then obtain  $p :: ('a, 'b) \text{ Profile}$  where
  profile  $V$   $A$   $p$ 
  by blast
show  $\exists B \subseteq A. (\forall a \in B. \text{indep-of-alt } (\text{drop-module } n \ r) \ V \ A \ a \wedge$ 
   $(\forall p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } (\text{drop-module } n \ r) \ V \ A \ p)) \wedge$ 
   $(\forall a \in A - B. \text{indep-of-alt } (\text{pass-module } n \ r) \ V \ A \ a \wedge$ 
   $(\forall p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } (\text{pass-module } n \ r) \ V \ A \ p))$ 
proof
  have same-A:
     $\forall p \ q. (\text{profile } V \ A \ p \wedge \text{profile } V \ A \ q) \longrightarrow$ 
     $\text{reject } (\text{drop-module } n \ r) \ V \ A \ p = \text{reject } (\text{drop-module } n \ r) \ V \ A \ q$ 
    by auto
  let  $?A = \text{reject } (\text{drop-module } n \ r) \ V \ A \ p$ 
  have  $?A \subseteq A$ 
    by auto
  moreover have  $\forall a \in ?A. \text{indep-of-alt } (\text{drop-module } n \ r) \ V \ A \ a$ 
    using assms drop-mod-sound
    unfolding drop-module.simps indep-of-alt-def
    by (metis (mono-tags, lifting))
  moreover have
     $\forall a \in ?A. \forall p. \text{profile } V \ A \ p$ 
     $\longrightarrow a \in \text{reject } (\text{drop-module } n \ r) \ V \ A \ p$ 
    by auto
  moreover have  $\forall a \in A - ?A. \text{indep-of-alt } (\text{pass-module } n \ r) \ V \ A \ a$ 
    using assms pass-mod-sound
    unfolding pass-module.simps indep-of-alt-def
    by metis
  moreover have
     $\forall a \in A - ?A. \forall p.$ 
     $\text{profile } V \ A \ p \longrightarrow a \in \text{reject } (\text{pass-module } n \ r) \ V \ A \ p$ 
    by auto
  ultimately show  $?A \subseteq A \wedge$ 
     $(\forall a \in ?A. \text{indep-of-alt } (\text{drop-module } n \ r) \ V \ A \ a \wedge$ 
     $(\forall p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } (\text{drop-module } n \ r) \ V \ A \ p)) \wedge$ 
     $(\forall a \in A - ?A. \text{indep-of-alt } (\text{pass-module } n \ r) \ V \ A \ a \wedge$ 
     $(\forall p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } (\text{pass-module } n \ r) \ V \ A \ p))$ 
    by simp
qed
qed

```

end

6.2 Revision Composition

theory *Revision-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

A revised electoral module rejects all originally rejected or deferred alternatives, and defers the originally elected alternatives. It does not elect any alternatives.

6.2.1 Definition

fun *revision-composition* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow
 ('a, 'v, 'a Result) Electoral-Module **where**
revision-composition m V A p = ({}, A - elect m V A p, elect m V A p)

abbreviation *rev* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow
 ('a, 'v, 'a Result) Electoral-Module (\downarrow 50) **where**
m \downarrow \equiv *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 \forall A V p. *profile* V A p \longrightarrow *elect* m V A p \subseteq A
using *elect-in-alts*
by *metis*
hence \forall A V p. *profile* V A p \longrightarrow (A - *elect* m V A p) \cup *elect* m V A p = A
by *blast*
hence \forall A V p. *profile* V A p \longrightarrow
set-equals-partition A (*revision-composition* m V A p)
by *simp*
moreover **have** \forall A V p. *profile* V A p \longrightarrow (A - *elect* m V A p) \cap *elect* m V
A p = {}
by *blast*
hence \forall A V p. *profile* V A p \longrightarrow *disjoint3* (*revision-composition* m V A p)
by *simp*
ultimately show *?thesis*
by *simp*

qed

lemma *voters-determine-rev-comp*:
fixes $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
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 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes *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 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes *electing* m
shows *non-blocking* ($m \downarrow$)
proof (*unfold non-blocking-def, safe*)
show *SCF-result.electoral-module* ($m \downarrow$)
using *assms rev-comp-sound*
unfolding *electing-def*
by (*metis (no-types, lifting)*)
next
fix
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $x :: 'a$
assume
 $\text{fin-}A$: *finite* A **and**
 $\text{prof-}A$: *profile* $V A p$ **and**
 $\text{reject-}A$: *reject* ($m \downarrow$) $V A p = A$ **and**
 $\text{x-in-}A$: $x \in A$
hence *non-electing* m
using *assms empty-iff Diff-disjoint Int-absorb2*
 elect-in-alt s *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  $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
  assumes invariant-monotonicity m
  shows defer-invariant-monotonicity (m↓)
proof (unfold defer-invariant-monotonicity-def, safe)
  show 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
       $A :: 'a \text{ set}$  and
       $V :: 'v \text{ set}$  and
       $p \ q :: ('a, 'v) \text{ Profile}$  and
       $a \ x \ x' :: 'a$ 
    assume
      rev-p-defer-a: a ∈ defer (m↓) V A p and
      a-lifted: lifted V A p q a and
      rev-q-defer-x: x ∈ defer (m↓) V A q and
      x-non-eq-a: x ≠ a and
      rev-q-defer-x': x' ∈ defer (m↓) V A q
    from rev-p-defer-a
    have elect-a-in-p: a ∈ 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 ≠ {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  $x' \in \text{defer } (m\downarrow) \ V \ A \ p$ 
      using rev-q-defer-x'
      by simp
  next

```

```

fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p \ q :: ('a, 'v) \text{ Profile}$  and
   $a \ x \ x' :: 'a$ 
assume
   $\text{rev-p-defer-a}: a \in \text{defer } (m\downarrow) \ V \ A \ p$  and
   $\text{a-lifted}: \text{lifted } V \ A \ p \ q \ a$  and
   $\text{rev-q-defer-x}: x \in \text{defer } (m\downarrow) \ V \ A \ q$  and
   $\text{x-non-eq-a}: x \neq a$  and
   $\text{rev-p-defer-x'}: x' \in \text{defer } (m\downarrow) \ V \ A \ p$ 
have reject-and-defer:
   $(A - \text{elect } m \ V \ A \ q, \text{elect } m \ V \ A \ q) = \text{snd } ((m\downarrow) \ V \ A \ q)$ 
by force
have elect-p-eq-defer-rev-p:  $\text{elect } m \ V \ A \ p = \text{defer } (m\downarrow) \ V \ A \ p$ 
by simp
hence elect-a-in-p:  $a \in \text{elect } m \ V \ A \ p$ 
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  $x' \in \text{defer } (m\downarrow) \ V \ A \ q$ 
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
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p \ q :: ('a, 'v) \text{ Profile}$  and
   $a \ x \ x' :: 'a$ 
assume
   $a \in \text{defer } (m\downarrow) \ V \ A \ p$  and
   $\text{lifted } V \ A \ p \ q \ a$  and
   $x' \in \text{defer } (m\downarrow) \ V \ A \ q$ 
with assms
show  $x' \in \text{defer } (m\downarrow) \ V \ A \ p$ 
using empty-iff insertE snd-conv revision-composition.elims
unfolding invariant-monotonicity-def
by metis
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p \ q :: ('a, 'v) \text{ Profile}$  and
   $a \ x \ x' :: 'a$ 

```

```

assume
  rev-p-defer-a:  $a \in \text{defer } (m \downarrow) \ V \ A \ p$  and
  a-lifted:  $\text{lifted } V \ A \ p \ q \ a$  and
  rev-q-not-defer-a:  $a \notin \text{defer } (m \downarrow) \ V \ A \ q$ 
moreover from assms
have lifted-inv:
   $\forall \ A \ V \ p \ q \ a. \ a \in \text{elect } m \ V \ A \ p \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$ 
   $\text{elect } m \ V \ A \ q = \text{elect } m \ V \ A \ p \vee \text{elect } m \ V \ A \ q = \{a\}$ 
  unfolding invariant-monotonicity-def
  by (metis (no-types))
moreover have p-defer-rev-eq-elect:  $\text{defer } (m \downarrow) \ V \ A \ p = \text{elect } m \ V \ A \ p$ 
  by simp
moreover have defer (m↓) V A q = elect m V A q
  by simp
ultimately show  $x' \in \text{defer } (m \downarrow) \ V \ A \ q$ 
  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  $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module
    (infix  $\triangleright$  50) where
      m  $\triangleright$  n  $\equiv$  sequential-composition m n

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 (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  $\cup$  n-e, m-r  $\cup$  n-r, n-d))

lemma voters-determine-seq-comp:
  fixes m n :: ('a, 'v, 'a Result) Electoral-Module
  assumes voters-determine-election m  $\wedge$  voters-determine-election n
  shows voters-determine-election (m  $\triangleright$  n)
proof (unfold voters-determine-election.simps, clarify)
  fix
    A :: 'a set and
    V :: 'v set and
    p p' :: ('a, 'v) Profile
  assume coincide:  $\forall v \in V. p v = p' v$ 
  hence eq: m V A p = m V A p'  $\wedge$  n V A p = n V A p'
    using assms
    unfolding voters-determine-election.simps
    by blast
  hence coincide-limit:
     $\forall v \in V. \text{limit-profile } (\text{defer } m \text{ V A } p) p v =$ 
       $\text{limit-profile } (\text{defer } m \text{ V A } p') p' v$ 
    using coincide
    by simp
  moreover 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'
       $\cup$  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
       $\cup$  reject 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')
    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 ∈ elect m V A p and
    x ∈ reject n V (defer m V A p)
    (limit-profile (defer m V A p) p)
  thus x ∈ {}
  using elect-reject-diff
  by blast
next
  fix x :: 'a
  assume
    x ∈ elect n V (defer m V A p)
    (limit-profile (defer m V A p) p) and
    x ∈ reject m V A p
  thus x ∈ {}
  using rej-intersect-new-elect-empty
  by blast
next
  fix x :: 'a
  assume
    x ∈ elect 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 disjoint-iff-not-equal module-n prof-def-lim result-disj prof
  by metis
qed
moreover have
  (elect m V A p ∪ elect n V ?new-A ?new-p)
  ∩ (defer n V ?new-A ?new-p) = {}
  using Int-Un-distrib2 Un-empty elect-defer-diff module-n
  prof-def-lim result-disj prof
  by (metis (no-types))
moreover have
  (reject m V A p ∪ reject n V ?new-A ?new-p)
  ∩ (defer n V ?new-A ?new-p) = {}
proof (safe)
  fix x :: 'a
  assume x ∈ defer n V (defer m V A p) (limit-profile (defer m V A p) p)
  hence x ∈ defer m V A p
  using defer-in-A module-n prof-def-lim prof
  by metis
  moreover assume x ∈ reject m V A p
  ultimately have x ∈ reject m V A p ∩ defer m V A p
  by fastforce
  thus x ∈ {}

```

```

    using disj-n
    by blast
next
fix  $x :: 'a$ 
assume
 $x \in \text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$  and
 $x \in \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$ 
thus  $x \in \{\}$ 
using module-n prof-def-lim reject-not-elected-or-deferred
by blast
qed
ultimately have
 $\text{disjoint3 } (\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ ?\text{new-A } ?\text{new-p},$ 
 $\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ ?\text{new-A } ?\text{new-p},$ 
 $\text{defer } n \ V \ ?\text{new-A } ?\text{new-p})$ 
by simp
thus ?thesis
unfolding sequential-composition.simps
by metis
qed

lemma seq-comp-presv-alts:
fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
 $A :: 'a \text{ set}$  and
 $V :: 'v \text{ set}$  and
 $p :: ('a, 'v) \text{ Profile}$ 
assumes
 $\text{module-m: } \text{SCF-result.electoral-module } m$  and
 $\text{module-n: } \text{SCF-result.electoral-module } n$  and
 $\text{prof: profile } V \ A \ p$ 
shows set-equals-partition  $A \ ((m \triangleright n) \ V \ A \ p)$ 
proof -
let  $?new-A = \text{defer } m \ V \ A \ p$ 
let  $?new-p = \text{limit-profile } ?new-A \ p$ 
have  $\text{elect-reject-diff: } \text{elect } m \ V \ A \ p \cup \text{reject } m \ V \ A \ p \cup ?new-A = A$ 
using module-m prof
by (simp add: result-presv-alts)
have  $\text{elect } n \ V \ ?new-A \ ?new-p \cup$ 
 $\text{reject } n \ V \ ?new-A \ ?new-p \cup$ 
 $\text{defer } n \ V \ ?new-A \ ?new-p = ?new-A$ 
using module-m module-n prof def-presv-prof result-presv-alts
by metis
hence  $(\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ ?new-A \ ?new-p) \cup$ 
 $(\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ ?new-A \ ?new-p) \cup$ 
 $\text{defer } n \ V \ ?new-A \ ?new-p = A$ 
using elect-reject-diff
by blast
hence set-equals-partition  $A$ 

```

```

      (elect m V A p  $\cup$  elect n V ?new-A ?new-p,
       reject m V A p  $\cup$  reject n V ?new-A ?new-p,
       defer n V ?new-A ?new-p)
    by simp
  thus ?thesis
    unfolding sequential-composition.simps
    by metis
qed

lemma seq-comp-alt-eq[fundef-cong, code]: sequential-composition = sequential-composition'
proof (unfold sequential-composition'.simps sequential-composition.simps)
  have  $\forall m n V A E.$ 
    (case m V A E of (e, r, d)  $\Rightarrow$ 
     case n V d (limit-profile d E) of (e', r', d')  $\Rightarrow$ 
     (e  $\cup$  e', r  $\cup$  r', d')) =
    (elect m V A E
      $\cup$  elect n V (defer m V A E) (limit-profile (defer m V A E) E),
     reject m V A E
      $\cup$  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
    ( $\lambda m n V A p.$ 
     let A' = defer m V A p; p' = limit-profile A' p in
     (elect m V A p  $\cup$  elect n V A' p',
      reject m V A p  $\cup$  reject n V A' p',
      defer n V A' p')) =
    ( $\lambda m n V A pr.$ 
     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
    SCF-result.electoral-module m and
    SCF-result.electoral-module n
  shows SCF-result.electoral-module (m  $\triangleright$  n)
proof (unfold 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 $\forall r. \text{well-formed-SCF } (A :: 'a \text{ set}) \ r =$
 $(\text{disjoint3 } r \wedge \text{set-equals-partition } A \ r)$
by *simp*
ultimately show $\text{well-formed-SCF } A \ ((m \triangleright n) \ V \ A \ p)$
using *assms seq-comp-presv-disj seq-comp-presv-alts*
by *metis*
qed

6.3.3 Lemmas

lemma *seq-comp-decrease-only-defer:*

fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assumes
 $\text{module-m: SCF-result.electoral-module } m$ **and**
 $\text{module-n: SCF-result.electoral-module } n$ **and**
 $\text{prof: profile } V \ A \ p$ **and**
 $\text{empty-defer: defer } m \ V \ A \ p = \{\}$
shows $(m \triangleright n) \ V \ A \ p = m \ V \ A \ p$
proof –
have $\forall m' \ A' \ V' \ p'.$
 $(\text{SCF-result.electoral-module } m' \wedge \text{profile } V' \ A' \ p') \longrightarrow$
 $\text{profile } V' \ (\text{defer } m' \ V' \ A' \ p') \ (\text{limit-profile } (\text{defer } m' \ V' \ A' \ p') \ p')$
using *def-presv-prof prof*
by *metis*
hence $\text{prof-no-alt: profile } V \ \{\} \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$
using *empty-defer prof module-m*
by *metis*
show *?thesis*
proof
have $(\text{elect } m \ V \ A \ p)$
 $\cup (\text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)) =$
 $\text{elect } m \ V \ A \ p$
using *elect-in-alts[of n V defer m V A p (limit-profile (defer m V A p) p)]*
 $\text{empty-defer module-n prof prof-no-alt}$
by *auto*
thus $\text{elect } (m \triangleright n) \ V \ A \ p = \text{elect } m \ V \ A \ p$
using *fst-conv*
unfolding *sequential-composition.simps*
by *metis*
next
have *rej-empty:*
 $\forall m' \ V' \ p'.$
 $(\text{SCF-result.electoral-module } m'$
 $\wedge \text{profile } V' \ \{\} \ p') \longrightarrow \text{reject } m' \ V' \ \{\} \ p' = \{\}$
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:  $\exists a \in A. m \vee A p = (\{\}, A - \{a\}, \{a\})$ 
  using Diff-empty elect-rej-def-combination reject-not-elected-or-deferred
  unfolding non-electing-def
  by metis
hence  $\exists a \in A. \text{elect } (m \triangleright n) \vee A p = \text{elect } n \vee \{a\} \text{ (limit-profile } \{a\} p)$ 
  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  $\exists a \in A. \text{elect } (m \triangleright n) \vee A p = \{a\}$ 
  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-Module and
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile
assumes
  SCF-result.electoral-module m and
  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-Module and
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile
assumes
  SCF-result.electoral-module m and
  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 $defer\ (m \triangleright n)\ V\ A\ p =$
 $defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\ profile\ (defer\ m\ V\ A\ p)\ p)$
using *snd-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-then-elect-elec-set:*

fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
shows $elect\ (m \triangleright n)\ V\ A\ p =$
 $elect\ n\ V\ (defer\ m\ V\ A\ p)$
 $(limit\ profile\ (defer\ m\ V\ A\ p)\ p) \cup (elect\ m\ V\ A\ p)$
using *Un-commute fst-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-elim-one-red-def-set:*

fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **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 $defer\ (m \triangleright n)\ V\ A\ p \subset defer\ m\ V\ A\ p$
using *assms snd-conv def-presv-prof single-elim-imp-red-def-set*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-set-trans:*

fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $A :: 'a\ set$ **and**

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

```

with non-blocking-m non-blocking-n
show ?thesis
proof (unfold non-blocking-def)
  assume
    emod-reject-m:
    SCF-result.electoral-module m
     $\wedge (\forall A V p. A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p$ 
       $\longrightarrow \text{reject } m V A p \neq A)$  and
    emod-reject-n:
    SCF-result.electoral-module n
     $\wedge (\forall A V p. A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p$ 
       $\longrightarrow \text{reject } n V A p \neq A)$ 
  show
    SCF-result.electoral-module (m  $\triangleright$  n)
     $\wedge (\forall A V p. A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p$ 
       $\longrightarrow \text{reject } (m \triangleright n) V A p \neq A)$ 
  proof (safe)
    show 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)
     $\wedge \text{profile } V (\text{defer } m V A p) (\text{limit-profile } (\text{defer } m V A p) p)$ 
    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)
     $(\text{limit-profile } (\text{defer } m V A p) p) \cup \text{elect } m V A p$ 
    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 ▷ n) V A p ∪ defer (m ▷ n) V A p =
    A - reject (m ▷ 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) ∪
    elect m V A p ∪
    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) = {} →
    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) = {} →
    elect m V A p = elect m V A p ∩ defer m V A p
    using elect-def-disj
    by blast
  thus x ∈ {}
    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 ▷ n)
proof (unfold non-electing-def, safe)
  have SCF-result.electoral-module m ∧ SCF-result.electoral-module n
    using assms
    unfolding non-electing-def
    by blast
  thus SCF-result.electoral-module (m ▷ n)
    using seq-comp-sound
    by metis
next

```

```

fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $x :: 'a$ 
assume
   $\text{profile } V \ A \ p$  and
   $x \in \text{elect } (m \triangleright n) \ V \ A \ p$ 
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  $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
  assumes
     $\text{def-one-}m$ :  $\text{defers } 1 \ m$  and
     $\text{electing-}n$ :  $\text{electing } n$ 
  shows  $\text{electing } (m \triangleright n)$ 
proof –
  have defer-card-eq-one:
     $\forall \ A \ V \ p. (\text{card } A \geq 1 \wedge \text{finite } A \wedge \text{profile } V \ A \ p) \longrightarrow \text{card } (\text{defer } m \ V \ A \ p) = 1$ 
    using def-one-m
    unfolding defers-def
    by metis
  hence def-m-not-empty:
     $\forall \ A \ V \ p. (A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V \ A \ p) \longrightarrow \text{defer } m \ V \ A \ p \neq \{\}$ 
    using One-nat-def Suc-leI card-eq-0-iff card-gt-0-iff zero-neq-one
    by metis
  thus ?thesis
proof –
  have  $\forall \ m'.$ 
     $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m'$ 
     $\wedge (\forall \ A' \ V' \ p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' \ A' \ p') \longrightarrow \text{elect } m' \ V' \ A' \ p' \neq \{\}))$ 
     $\wedge (\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m' \vee$ 
     $(\exists \ A \ V \ p. (A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V \ A \ p \wedge \text{elect } m' \ V \ A \ p = \{\})))$ 
    unfolding electing-def
    by blast
  hence  $\forall \ m'.$ 
     $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m'$ 
     $\wedge (\forall \ A' \ V' \ p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' \ A' \ p') \longrightarrow \text{elect } m' \ V' \ A' \ p' \neq \{\}))$ 

```

$\longrightarrow \text{elect } m' \ V' \ A' \ p' \neq \{\})$
 $\wedge (\exists \ A \ V \ p. (\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m' \vee A \neq \{}$
 $\wedge \text{finite } A \wedge \text{profile } V \ A \ p \wedge \text{elect } m' \ V \ A \ p = \{\}))$
by simp
then obtain
 $A :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'a \text{ set}$ **and**
 $V :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'v \text{ set}$ **and**
 $p :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow ('a, 'v) \text{ Profile}$ **where**
f-mod:
 $\forall \ m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module.}$
 $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m' \wedge$
 $(\forall \ A' \ V' \ p'. (A' \neq \{ \} \wedge \text{finite } A' \wedge \text{profile } V' \ A' \ p') \longrightarrow \text{elect } m' \ V' \ A' \ p' \neq \{\}))$
 $\wedge (\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m' \vee A \ m' \neq \{ \}$
 $\wedge \text{finite } (A \ m') \wedge \text{profile } (V \ m') \ (A \ m') \ (p \ m')$
 $\wedge \text{elect } m' \ (V \ m') \ (A \ m') \ (p \ m') = \{\})$
by metis
hence f-elect:
 $\text{SCF-result.electoral-module } n \wedge$
 $(\forall \ A \ V \ p. (A \neq \{ \} \wedge \text{finite } A \wedge \text{profile } V \ A \ p) \longrightarrow \text{elect } n \ V \ A \ p \neq \{\})$
using electing-n
unfolding electing-def
by metis
have def-card-one:
 $\text{SCF-result.electoral-module } m$
 $\wedge (\forall \ A \ V \ p. (1 \leq \text{card } A \wedge \text{finite } A \wedge \text{profile } V \ A \ p)$
 $\longrightarrow \text{card } (\text{defer } m \ V \ A \ p) = 1)$
using def-one-m defer-card-eq-one
unfolding defers-def
by blast
hence 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
 $\text{def-m-not-empty bot-eq-sup-iff finite-subset}$
unfolding electing-def
by metis
qed
qed
lemma def-lift-inv-seq-comp-help:
fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p \ q :: ('a, 'v) \text{ Profile}$ **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: $a \in (\text{defer } (m \triangleright n) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a$
shows $(m \triangleright n) \ V \ A \ p = (m \triangleright n) \ V \ A \ q$
proof –
let $?new\text{-}Ap = \text{defer } m \ V \ A \ p$
let $?new\text{-}Aq = \text{defer } m \ V \ A \ q$
let $?new\text{-}p = \text{limit-profile } ?new\text{-}Ap \ p$
let $?new\text{-}q = \text{limit-profile } ?new\text{-}Aq \ q$
from *monotone-m monotone-n*
have *modules: SCF-result.electoral-module m \wedge SCF-result.electoral-module n*
unfolding *defer-lift-invariance-def*
by *simp*
hence $\text{profile } V \ A \ p \longrightarrow \text{defer } (m \triangleright n) \ V \ A \ p \subseteq \text{defer } m \ V \ A \ p$
using *seq-comp-def-set-bounded*
by *metis*
moreover **have** *profile-p: lifted V A p q a \longrightarrow finite-profile V A p*
unfolding *lifted-def*
by *simp*
ultimately **have** *defer-subset: defer (m \triangleright n) V A p \subseteq defer m V A p*
using *def-and-lifted*
by *blast*
hence *mono-m: m V A p = m V A q*
using *monotone-m def-and-lifted modules profile-p*
seq-comp-def-set-trans
unfolding *defer-lift-invariance-def*
by *metis*
hence *new-A-eq: ?new-Ap = ?new-Aq*
by *presburger*
have *defer-eq: defer (m \triangleright n) V A p = defer n V ?new-Ap ?new-p*
using *snd-conv*
unfolding *sequential-composition.simps*
by *metis*
have *mono-n: n V ?new-Ap ?new-p = n V ?new-Aq ?new-q*
proof (*cases*)
assume *lifted V ?new-Ap ?new-p ?new-q a*
thus *?thesis*
using *defer-eq mono-m monotone-n def-and-lifted*
unfolding *defer-lift-invariance-def*
by (*metis (no-types, lifting)*)
next
assume *unlifted-a: $\neg \text{lifted } V \ ?new\text{-}Ap \ ?new\text{-}p \ ?new\text{-}q \ a$*
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 ∈ ?new-Ap
  by blast
ultimately have lifted-stmt:
  (∃ v ∈ V.
    Preference-Relation.lifted ?new-Ap (?new-p v) (?new-q v) a) →
  (∃ v ∈ V.
    ¬ Preference-Relation.lifted ?new-Ap (?new-p v) (?new-q v) a ∧
    (?new-p v) ≠ (?new-q v))
  using unfolded-a def-and-lifted defer-in-alts infinite-super modules profile-p
  unfolding lifted-def
  by metis
from def-and-lifted modules
have ∀ v ∈ V. (Preference-Relation.lifted A (p v) (q v) a ∨ (p v) = (q v))
  unfolding Profile.lifted-def
  by metis
with def-and-lifted modules mono-m
have ∀ v ∈ V.
  (Preference-Relation.lifted ?new-Ap (?new-p v) (?new-q v) a ∨
   (?new-p v) = (?new-q v))
  using limit-lifted-imp-eq-or-lifted defer-in-alts
  unfolding Profile.lifted-def limit-profile.simps
  by (metis (no-types, lifting))
with lifted-stmt
have ∀ v ∈ V. (?new-p v) = (?new-q v)
  by blast
with mono-m
show ?thesis
  using leI not-less-zero nth-equalityI 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 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 SCF-result.electoral-module m
    using non-electing-m
    unfolding non-electing-def
    by simp
  moreover have SCF-result.electoral-module n
    using def-one-n
    unfolding defers-def
    by simp
  ultimately show 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 ≤ card A and
  fin-A: finite A and
  prof-A: profile V A p
from pos-card
have A ≠ {}
  by auto
with fin-A prof-A
have reject m V A p ≠ A
  using non-blocking-m
  unfolding non-blocking-def
  by simp
hence ∃ a. a ∈ A ∧ a ∉ 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 ≠ {}
  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
  ∀ i m'. defers i m' =
    (SCF-result.electoral-module m' ∧
     (∀ A' V' p'. (i ≤ card A' ∧ finite A' ∧ profile V' A' p') →
      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 ▷ 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 ▷ 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* $(m \triangleright m')\ n$
proof (*unfold disjoint-compatibility-def, safe*)
show *SCF-result.electoral-module* $(m \triangleright m')$
 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 $(m \triangleright m') \wedge$ *SCF-result.electoral-module* n
 using *compatible module-m' seq-comp-sound*
 unfolding *disjoint-compatibility-def*
 by *metis*
obtain $A :: 'a\ set$ **where**
 rej-A:
 $A \subseteq S \wedge$
 $(\forall\ a \in A.$
 indep-of-alt $m\ V\ S\ a \wedge (\forall\ p.\ profile\ V\ S\ p \longrightarrow a \in reject\ m\ V\ S\ p)) \wedge$
 $(\forall\ a \in S - A.$
 indep-of-alt $n\ V\ S\ a \wedge (\forall\ p.\ profile\ V\ S\ p \longrightarrow a \in reject\ n\ V\ S\ p))$
 using *compatible*
 unfolding *disjoint-compatibility-def*
 by (*metis (no-types, lifting)*)
show
 $\exists\ A \subseteq S.$
 $(\forall\ a \in A.\ i\ n\ d\ e\ p\ e\ n\ d\ o\ f\ a\ l\ t\ (m \triangleright m')\ V\ S\ a \wedge$
 $(\forall\ p.\ profile\ V\ S\ p \longrightarrow a \in reject\ (m \triangleright m')\ V\ S\ p)) \wedge$
 $(\forall\ a \in S - A.$
 indep-of-alt $n\ V\ S\ a \wedge (\forall\ p.\ profile\ V\ S\ p \longrightarrow a \in reject\ n\ V\ S\ p))$
proof
 have $\forall\ a\ p\ q.\ a \in A \wedge equiv\text{-}prof\text{-}except\text{-}a\ V\ S\ p\ q\ a \longrightarrow$
 $(m \triangleright m')\ V\ S\ p = (m \triangleright m')\ V\ S\ q$
 proof (*safe*)

```

fix
  a :: 'a and
  p 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
from lifting-equiv-p-q
have profiles: profile V S p ∧ profile V S 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 defer-not-elec-or-rej[of m V A p]
    profiles rej-A IntI emptyE result-disj
  unfolding disjoint-compatibility-def
  by metis
ultimately have
  ∀ 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 V S]
  unfolding eq-defer limit-profile.simps
  by blast
with eq-defer
have m' V (defer m V S p) (limit-profile (defer m V S p) p) =
  m' V (defer m V S q) (limit-profile (defer m V S q) q)
  using 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 ▷ n)
proof (unfold condorcet-compatibility-def, safe)
  have SCF-result.electoral-module m
    using dcc-m
    unfolding defer-condorcet-consistency-def
    by presburger
  moreover have SCF-result.electoral-module n
    using nb-n
    unfolding non-blocking-def
    by presburger
  ultimately have SCF-result.electoral-module (m ▷ n)
    using seq-comp-sound
    by metis
  thus SCF-result.electoral-module (m ▷ 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 ∈ reject (m ▷ n) V A p
  hence ∃ a'. defer-condorcet-consistency m ∧ 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: SCF-result.electoral-module m
    using dcc-m
    unfolding defer-condorcet-consistency-def
    by presburger
  moreover have SCF-result.electoral-module n
    using nb-n

```

unfolding *non-blocking-def*
by *presburger*
ultimately have *sound-seq-m-n: 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 ∧ finite V ∧ profile V A p
 $\wedge a \in A \wedge (\forall a'. a' \in A - \{a'\} \longrightarrow \text{wins } V a p a')$
using *cw-a condorcet-winner.simps DiffD2 singletonI*
by (*metis* (*no-types*))
have $\forall a' A'. (a' :: 'a) \in A' \longrightarrow \text{insert } a' (A' - \{a'\}) = A'$
by *blast*
have *nb-n-full:*
SCF-result.electoral-module n ∧
 $(\forall A' V' p'. A' \neq \{\} \wedge \text{finite } A' \wedge \text{finite } V' \wedge \text{profile } V' A' p' \longrightarrow \text{reject } n V' A' p' \neq A')$
using *nb-n non-blocking-def*
by *metis*
have *def-seq-diff:*
 $\text{defer } (m \triangleright n) V A p = A - \text{elect } (m \triangleright n) V A p - \text{reject } (m \triangleright n) V A p$
using *defer-not-elec-or-rej cond-win sound-seq-m-n*
by *metis*
have *set-ins: $\forall a' A'. (a' :: 'a) \in A' \longrightarrow \text{insert } a' (A' - \{a'\}) = A'$*
by *fastforce*
have $\forall p' A' p''. p' = (A' :: 'a \text{ set}, p'' :: 'a \text{ set} \times 'a \text{ set}) \longrightarrow \text{snd } p' = p''$
by *simp*
hence
 $\text{snd } (\text{elect } m V A p \cup \text{elect } n V (\text{defer } m V A p) (\text{limit-profile } (\text{defer } m V A p) p),$
 $\text{reject } m V A p \cup \text{reject } n V (\text{defer } m V A p) (\text{limit-profile } (\text{defer } m V A p) p),$
 $\text{defer } n V (\text{defer } m V A p) (\text{limit-profile } (\text{defer } m V A p) p)) =$
 $(\text{reject } m V A p$

$\cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p),$
 $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p))$
by *blast*
hence *seq-snd-simplified*:
 $\text{snd } ((m \triangleright n) \ V \ A \ p) =$
 $(\text{reject } m \ V \ A \ p$
 $\cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p),$
 $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p))$
using *sequential-composition.simps*
by *metis*
hence *seq-rej-union-eq-rej*:
 $\text{reject } m \ V \ A \ p$
 $\cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) =$
 $\text{reject } (m \triangleright n) \ V \ A \ p$
by *simp*
hence *seq-rej-union-subset-A*:
 $\text{reject } m \ V \ A \ p$
 $\cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \subseteq A$
using *sound-seq-m-n cond-win reject-in-alts*
by (*metis* (*no-types*))
hence $A - \{a\} = \text{reject } (m \triangleright n) \ V \ A \ p - \{a\}$
using *seq-rej-union-eq-rej defer-not-elec-or-rej cond-win def-m diff-elect-m*
double-diff rej-m sound-m sup-ge1
by (*metis* (*no-types*))
hence $\text{reject } (m \triangleright n) \ V \ A \ p \subseteq A - \{a\}$
using *seq-rej-union-subset-A seq-snd-simplified set-ins def-seq-diff nb-n-full*
cond-win fst-conv Diff-empty Diff-eq-empty-iff a-in-rej-seq-m-n def-m
def-presv-prof sound-m ne-n diff-elect-m insert-not-empty defer-in-alts
reject-not-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
 $A :: 'a \ \text{set} \ \text{and}$
 $V :: 'v \ \text{set} \ \text{and}$
 $p :: ('a, 'v) \ \text{Profile} \ \text{and}$
 $a \ a' :: 'a$
assume
 $\text{cw-}a: \text{condorcet-winner } V \ A \ p \ a \ \text{and}$
 $\text{not-cw-}a': \neg \text{condorcet-winner } V \ A \ p \ a' \ \text{and}$
 $a'\text{-in-elect-seq-m-n: } a' \in \text{elect } (m \triangleright n) \ V \ A \ p$
hence $\exists \ a''. \text{defer-condorcet-consistency } m \wedge \text{condorcet-winner } V \ A \ p \ a''$
using *dcc-m*
by *blast*
hence *result-m*: $m \ V \ A \ p = (\{\}, A - (\text{defer } m \ V \ A \ p), \{a\})$

using *defer-condorcet-consistency-def cw-a cond-winner-unique*
by (*metis (no-types, lifting)*)
have *sound-m: SCF-result.electoral-module m*
using *dcc-m*
unfolding *defer-condorcet-consistency-def*
by *presburger*
moreover have *SCF-result.electoral-module n*
using *nb-n*
unfolding *non-blocking-def*
by *presburger*
ultimately have *sound-seq-m-n: SCF-result.electoral-module (m ▷ 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' ∈ 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 $\forall p' A' p''. p' = (A' :: 'a \text{ set}, p'' :: 'a \text{ set} \times 'a \text{ set}) \longrightarrow \text{snd } p' = p''$
by *simp*
hence *m-seq-n:*

$$\begin{aligned} & \text{snd } (\text{elect } m \text{ V } A \text{ } p \\ & \quad \cup \text{elect } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p), \\ & \quad \text{reject } m \text{ V } A \text{ } p \\ & \quad \cup \text{reject } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p), \\ & \quad \text{defer } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p)) = \\ & \quad (\text{reject } m \text{ V } A \text{ } p \\ & \quad \cup \text{reject } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p), \\ & \quad \text{defer } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p)) \end{aligned}$$
by *blast*
have *a' ∈ 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:*

$$\begin{aligned} & a \in \text{reject } m \text{ V } A \text{ } p \\ & \cup \text{reject } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p) \end{aligned}$$
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:*

$$\begin{aligned} & (m \triangleright n) \text{ V } A \text{ } p = \\ & \quad (\text{elect } m \text{ V } A \text{ } p \\ & \quad \cup \text{elect } n \text{ V } (\text{defer } m \text{ V } A \text{ } p) (\text{limit-profile } (\text{defer } m \text{ V } A \text{ } p) \text{ } p), \end{aligned}$$

```

    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 ∀ A' A''. (A' :: 'a set) = fst (A', A'' :: 'a set)
  by simp
hence a ∈ reject (m ▷ n) V A p
  using a-in-rej-union m-seq-n m-seq-n-full
  by presburger
moreover have
  finite A ∧ finite V ∧ profile V A p
  ∧ a ∈ A ∧ (∀ a''. a'' ∈ A - {a} ⟶ 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' ∈ A and
  not-cw-a': ¬ 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 ≠ a'
  using cw-a not-cw-a'
  by safe
ultimately have a' ∈ reject m V A p
  using DiffI a'-in-A singletonD
  by (metis (no-types))
hence a' ∈ reject m V A p
  ∪ reject n V (defer m V A p) (limit-profile (defer m V A p) p)
  by blast
moreover have
  (m ▷ 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
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  $a' \in \text{reject } (m \triangleright n) V A p$ 
  using fst-eqD
  by (metis (no-types))
qed

```

Composing a defer-condorcet-consistent electoral module in sequence with a non-blocking and non-electing electoral module results in a defer-condorcet-consistent module.

```

theorem seq-comp-dcc[simp]:
  fixes  $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ 
  assumes
    dcc-m: defer-condorcet-consistency m and
    nb-n: non-blocking n and
    ne-n: non-electing n
  shows defer-condorcet-consistency  $(m \triangleright n)$ 
proof (unfold defer-condorcet-consistency-def, safe)
  have SCF-result.electoral-module m
    using dcc-m
  unfolding defer-condorcet-consistency-def
  by metis
  thus SCF-result.electoral-module  $(m \triangleright n)$ 
    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  $\exists\ a'. \text{defer-condorcet-consistency } m \wedge \text{condorcet-winner } V A p a'$ 
    using dcc-m
  by blast
  hence result-m:  $m\ V\ A\ p = (\{\}, A - (\text{defer } m\ V\ A\ p), \{a\})$ 

```

```

    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: SCF-result.electoral-module m
    using dcc-m
    unfolding defer-condorcet-consistency-def
    by metis
  hence sound-seq-m-n: SCF-result.electoral-module (m ▷ n)
    using ne-n seq-comp-sound
    unfolding non-electing-def
    by metis
  have defer-eq-a: defer (m ▷ n) V A p = {a}
  proof (safe)
    fix a' :: 'a
    assume a'-in-def-seq-m-n: a' ∈ defer (m ▷ n) V A p
    have {a} = {a ∈ A. condorcet-winner V A p a}
      using cond-winner-unique cw-a
      by metis
    moreover have defer-condorcet-consistency m ⟶
      m V A p = ({}, A - defer m V A p, {a ∈ A. condorcet-winner V A p a})
      using cw-a defer-condorcet-consistency-def
      by (metis (no-types))
    ultimately have defer m V A p = {a}
      using dcc-m snd-conv
      by (metis (no-types, lifting))
    hence defer (m ▷ n) V A p = {a}
      using cw-a a'-in-def-seq-m-n 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 ∃ a'. defer-condorcet-consistency m ∧ condorcet-winner V A p a'
      using cw-a dcc-m
      by blast
    hence m V A p = ({}, A - (defer m V A p), {a})
      using defer-condorcet-consistency-def cw-a cond-winner-unique
      by (metis (no-types, lifting))
    hence elect-m-empty: elect m V A p = {}
      using eq-fst-iff
      by metis
    have profile V (defer m V A p) (limit-profile (defer m V A p) p)
      using condorcet-winner.simps cw-a def-presv-prof sound-m
      by (metis (no-types))
    hence elect n V (defer m V A p) (limit-profile (defer m V A p) p) = {}

```

```

    using ne-n non-electing-def
    by metis
  hence elect (m ▷ n) V A p = {}
    using elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral
    by (metis (no-types))
  moreover have condorcet-compatibility (m ▷ n)
    using dcc-m nb-n ne-n
    by simp
  hence a ∉ reject (m ▷ n) V A p
    unfolding condorcet-compatibility-def
    using cw-a
    by metis
  ultimately show a ∈ defer (m ▷ n) V A p
    using cw-a electoral-mod-defer-elem empty-iff
      sound-seq-m-n condorcet-winner.simps
    by metis
qed
have profile V (defer m V A p) (limit-profile (defer m V A p) p)
  using condorcet-winner.simps cw-a def-presv-prof sound-m
  by (metis (no-types))
hence elect n V (defer m V A p) (limit-profile (defer m V A p) p) = {}
  using ne-n
  unfolding non-electing-def
  by metis
hence elect (m ▷ 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 ▷ n) V A p = {a}
  using cw-a defer-eq-a
  by (metis (no-types))
ultimately have (m ▷ 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' ∈ A. condorcet-winner V A p a'} = {a}
  using cw-a cond-winner-unique
  by metis
ultimately show (m ▷ n) V A p
  = ({}, A - defer (m ▷ n) V A p, {a' ∈ A. condorcet-winner V A p a'})
  using def-seq-m-n-eq-a
  by metis
qed

```

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *seq-comp-mono*[simp]:
 fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
 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  $\triangleright$  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  $\in$  defer (m  $\triangleright$  n) V A p and
  lifted-a: Profile.lifted V A p q a
have non-electing-m: non-electing m
  using strong-def-mon-m
  unfolding defer-invariant-monotonicity-def
  by simp
have electoral-mod-m: 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  $\leq$  card A
  using Profile.lifted-def card-eq-0-iff emptyE less-one lifted-a linorder-le-less-linear
  by metis
hence n-defers-exactly-one-p: card (defer n V A p) = 1
  using finite-profile-p defers-one
  unfolding defers-def
  by (metis (no-types))
have fin-prof-def-m-q:
  profile V (defer m V A q) (limit-profile (defer m V A q) q)
  using def-presv-prof electoral-mod-m finite-profile-q
  by (metis (no-types))
have def-seq-m-n-q:

```

```

defer (m ▷ 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 ∉ {}
  by simp
have def-seq-m-n:
  defer (m ▷ 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 ≤ 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) 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 ▷ 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 ▷ n) V A p = {a}
  using defer-a-p is-singleton-altdef is-singleton-the-elem singletonD
  by (metis (no-types))
show (m ▷ n) V A p = (m ▷ n) V A q
proof (cases)
  assume defer m V A q ≠ 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 $(a \in \text{defer } m \ V \ A \ p) \longrightarrow \text{card } (\text{defer } (m \triangleright n) \ V \ A \ q) = 1$
using *card-eq-0-iff card-insert-disjoint defers-one electoral-mod-m empty-iff*
order-refl finite.emptyI seq-comp-defers-def-set def-presv-prof
finite-profile-q finite.insertI
unfolding *One-nat-def defers-def*
by *metis*
moreover have $a \in \text{defer } m \ V \ A \ p$
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 $\text{defer } (m \triangleright n) \ V \ A \ q = \{a\}$
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 $\text{defer } (m \triangleright n) \ V \ A \ p = \text{defer } (m \triangleright n) \ V \ A \ q$
using *def-seq-m-n-eq-a*
by *presburger*
moreover have $\text{elect } (m \triangleright n) \ V \ A \ p = \text{elect } (m \triangleright n) \ V \ A \ q$
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 $\neg (\text{defer } m \ V \ A \ q \neq \text{defer } m \ V \ A \ p)$
hence *def-eq*: $\text{defer } m \ V \ A \ q = \text{defer } m \ V \ A \ p$
by *presburger*
have $\text{elect } m \ V \ A \ p = \{\}$
using *finite-profile-p non-electing-m*
unfolding *non-electing-def*
by *simp*
moreover have $\text{elect } m \ V \ A \ q = \{\}$
using *finite-profile-q non-electing-m*
unfolding *non-electing-def*
by *simp*
ultimately have *elect-m-equal*:
 $\text{elect } m \ V \ A \ p = \text{elect } m \ V \ A \ q$
by *simp*
have $(\forall v \in V. (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \ v =$
 $(\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ v)$
 $\vee \text{lifted } V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$
 $(\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ a$
using *def-eq defer-in-alts electoral-mod-m lifted-a finite-profile-q*
limit-prof-eq-or-lifted

by *metis*
 moreover have

$$(\forall v \in V. (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \ v =$$

$$(\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ v)$$

$$\longrightarrow n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) =$$

$$n \ V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ q) \ q)$$
 using *voters-determine-n def-eq*
 unfolding *voters-determine-election.simps*
 by *presburger*
 moreover have

$$\text{lifted } V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$$

$$(\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ a$$

$$\longrightarrow \text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) =$$

$$\text{defer } n \ V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ q) \ q)$$
 proof (intro *impI*)
 assume *lifted*:

$$\text{Profile.lifted } V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$$

$$(\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ a$$
 hence $a \in \text{defer } n \ V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ q) \ q)$
 using *lifted-a def-seq-m-n defer-a-p defer-monotone-n*

$$\text{fin-prof-def-m-q def-eq}$$
 unfolding *defer-monotonicity-def*
 by *metis*
 hence $a \in \text{defer } (m \triangleright n) \ V \ A \ q$
 using *def-seq-m-n-q*
 by *simp*
 moreover have $\text{card } (\text{defer } (m \triangleright n) \ V \ A \ q) = 1$
 using *def-seq-m-n-q defers-one def-eq defer-seq-m-n-eq-one defers-def lifted*

$$\text{electoral-mod-m fin-prof-def-m-q finite-profile-p seq-comp-def-card-bounded}$$

$$\text{Profile.lifted-def}$$
 by (*metis (no-types, lifting)*)
 ultimately have $\text{defer } (m \triangleright n) \ V \ A \ q = \{a\}$
 using *a-non-empty card-1-singletonE insertE*
 by *metis*
 thus $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$

$$= \text{defer } n \ V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ q) \ q)$$
 using *def-seq-m-n-eq-a def-seq-m-n-q def-seq-m-n*
 by *presburger*
 qed
 ultimately have $\text{defer } (m \triangleright n) \ V \ A \ p = \text{defer } (m \triangleright n) \ V \ A \ q$
 using *def-seq-m-n def-seq-m-n-q*
 by *presburger*
 hence $\text{defer } (m \triangleright n) \ V \ A \ p = \text{defer } (m \triangleright n) \ V \ A \ q$
 using *a-non-empty def-eq def-seq-m-n def-seq-m-n-q*

$$\text{defer-a-p defer-monotone-n finite-profile-p}$$

$$\text{defer-seq-m-n-eq-one defers-one electoral-mod-m}$$

$$\text{fin-prof-def-m-q}$$
 unfolding *defers-def*
 by (*metis (no-types, lifting)*)

```

moreover from this
have reject ( $m \triangleright n$ )  $V A p = \text{reject } (m \triangleright n) V A q$ 
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 ( $m \triangleright n$ )  $V A p = \text{elect } (m \triangleright n) V A q$ 
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 :: ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module  $\Rightarrow$ 
    ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module  $\Rightarrow$   $'a \text{ Aggregator}$   $\Rightarrow$ 
    ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module where
    parallel-composition  $m n \text{ agg } V A p = \text{agg } A (m V A p) (n V A p)$ 

abbreviation parallel :: ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module  $\Rightarrow$   $'a \text{ Aggregator}$   $\Rightarrow$ 
    ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module  $\Rightarrow$  ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module
    ( $- \parallel -$  -  $[50, 1000, 51]$  50) where
     $m \parallel_a n \equiv \text{parallel-composition } m n a$ 

```

6.4.2 Soundness

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

fixes

$m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $a :: 'a\ Aggregator$

assumes

$SCF\text{-}result.electoral\text{-}module\ m$ **and**
 $SCF\text{-}result.electoral\text{-}module\ n$ **and**
 $aggregator\ a$

shows $SCF\text{-}result.electoral\text{-}module\ (m \parallel_a n)$

proof (*unfold 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

$\forall\ a'. aggregator\ a' =$
 $(\forall\ A'\ e\ r\ d\ e'\ r'\ d'.$
 $(well\text{-}formed\text{-}SCF\ (A' :: 'a\ set)\ (e, r', d)$
 $\wedge\ well\text{-}formed\text{-}SCF\ A'\ (r, d', e'))$
 $\longrightarrow well\text{-}formed\text{-}SCF\ A'\ (a'\ A'\ (e, r', d)\ (r, d', e')))$

unfolding *aggregator-def*

by *blast*

moreover have

$\forall\ m'\ V'\ A'\ p'.$
 $(SCF\text{-}result.electoral\text{-}module\ m' \wedge finite\ (A' :: 'a\ set)$
 $\wedge\ finite\ (V' :: 'v\ set) \wedge profile\ V'\ A'\ p')$
 $\longrightarrow well\text{-}formed\text{-}SCF\ A'\ (m'\ V'\ A'\ p')$

using *par-comp-result-sound*

by (*metis (no-types)*)

ultimately have $well\text{-}formed\text{-}SCF\ A\ (a\ A\ (m\ V\ A\ p)\ (n\ V\ A\ p))$

using *elect-rej-def-combination assms*

by (*metis par-comp-result-sound*)

thus $well\text{-}formed\text{-}SCF\ A\ ((m \parallel_a n)\ V\ A\ p)$

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 *SCF-result.electoral-module* m
using *non-electing-m*
unfolding *non-electing-def*
by *simp*
moreover have *SCF-result.electoral-module* n
using *non-electing-n*
unfolding *non-electing-def*
by *simp*
moreover have *aggregator* a
using *conservative*
unfolding *agg-conservative-def*
by *simp*
ultimately show *SCF-result.electoral-module* ($m \parallel_a n$)
using *par-comp-sound*
by *simp*
next
fix
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $w :: 'a$
assume
prof-A: *profile* $V A p$ **and**
w-wins: $w \in \text{elect } (m \parallel_a n) V A p$
have *emod-m*: *SCF-result.electoral-module* m
using *non-electing-m*
unfolding *non-electing-def*
by *simp*
have *emod-n*: *SCF-result.electoral-module* n
using *non-electing-n*
unfolding *non-electing-def*
by *simp*
have $\forall r r' d d' e e' A' f.$
 $((\text{well-formed-SCF } (A' :: 'a \text{ set}) (e', r', d') \wedge$
 $\text{well-formed-SCF } A' (e, r, d)) \longrightarrow$
 $\text{elect-}r (f A' (e', r', d') (e, r, d)) \subseteq e' \cup e \wedge$
 $\text{reject-}r (f A' (e', r', d') (e, r, d)) \subseteq r' \cup r \wedge$
 $\text{defer-}r (f A' (e', r', d') (e, r, d)) \subseteq d' \cup d) =$
 $((\text{well-formed-SCF } A' (e', r', d') \wedge$
 $\text{well-formed-SCF } A' (e, r, d)) \longrightarrow$
 $\text{elect-}r (f A' (e', r', d') (e, r, d)) \subseteq e' \cup e \wedge$
 $\text{reject-}r (f A' (e', r', d') (e, r, d)) \subseteq r' \cup r \wedge$
 $\text{defer-}r (f A' (e', r', d') (e, r, d)) \subseteq d' \cup d)$
by *linarith*
hence $\forall a'. \text{agg-conservative } a' =$

```

    (aggregator  $a' \wedge$ 
      ( $\forall A' e e' d d' r r'.$ 
        (well-formed-SCF ( $A' :: 'a \text{ set}$ ) ( $e, r, d$ )  $\wedge$ 
          well-formed-SCF  $A' (e', r', d')$ )  $\longrightarrow$ 
          elect- $r (a' A' (e, r, d) (e', r', d')) \subseteq e \cup e' \wedge$ 
          reject- $r (a' A' (e, r, d) (e', r', d')) \subseteq r \cup r' \wedge$ 
          defer- $r (a' A' (e, r, d) (e', r', d')) \subseteq d \cup d')$ )
      )
  unfolding aggregator-conservative-def
  by simp
hence aggregator  $a \wedge$ 
  ( $\forall A' e e' d d' r r'.$ 
    (well-formed-SCF  $A' (e, r, d) \wedge$ 
      well-formed-SCF  $A' (e', r', d')$ )  $\longrightarrow$ 
      elect- $r (a A' (e, r, d) (e', r', d')) \subseteq e \cup e' \wedge$ 
      reject- $r (a A' (e, r, d) (e', r', d')) \subseteq r \cup r' \wedge$ 
      defer- $r (a A' (e, r, d) (e', r', d')) \subseteq d \cup d')$ )
  using conservative
  by presburger
hence let  $c = (a A (m V A p) (n V A p))$  in
  (elect- $r c \subseteq ((\text{elect } m V A p) \cup (\text{elect } n V A p)))$ 
  using emod- $m$  emod- $n$  par-comp-result-sound
  prod.collapse prof- $A$ 
  by metis
hence  $w \in ((\text{elect } m V A p) \cup (\text{elect } n V A p))$ 
  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

$m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$t :: 'a \text{ Termination-Condition}$ **and**

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$

assumes

$\neg t \text{ (acc } V \text{ A } p)$ **and**

$\text{defer (acc } \triangleright m) \text{ V A } p \subset \text{defer acc V A } p$ **and**

$\text{finite (defer acc V A } p)$

shows $((\text{acc } \triangleright m, m, t, V, A, p), (\text{acc}, m, t, V, A, p)) \in$

$\text{measure } (\lambda (\text{acc}, m, t, V, A, p). \text{card (defer acc V A } p))$

using *assms psubset-card-mono*

by *simp*

This function handles the accumulator for the following loop composition function.

function *loop-comp-helper* :: $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'a \text{ Termination-Condition} \Rightarrow$

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **where**

loop-comp-helper-finite:

$\text{finite (defer acc V A } p) \wedge (\text{defer (acc } \triangleright m) \text{ V A } p) \subset (\text{defer acc V A } p)$

$\longrightarrow t \text{ (acc V A } p) \Longrightarrow$

$\text{loop-comp-helper acc m t V A } p = \text{acc V A } p \mid$

loop-comp-helper-infinite:

$\neg (\text{finite (defer acc V A } p) \wedge (\text{defer (acc } \triangleright m) \text{ V A } p) \subset (\text{defer acc V A } p))$

$\longrightarrow t \text{ (acc V A } p) \Longrightarrow$

$\text{loop-comp-helper acc m t V A } p = \text{loop-comp-helper (acc } \triangleright m) m t V A } p$

proof –

fix

$P :: \text{bool}$ **and**

$\text{accum} ::$

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \times ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

$\times 'a \text{ Termination-Condition} \times 'v \text{ set} \times 'a \text{ set} \times ('a, 'v) \text{ Profile}$

have *accum-exists*: $\exists m \ n \ t \ V \ A \ p. (m, n, t, V, A, p) = \text{accum}$

using *prod-cases5*

by *metis*

assume

$\bigwedge \text{acc V A } p \ m \ t.$

$\text{finite (defer acc V A } p) \wedge \text{defer (acc } \triangleright m) \text{ V A } p \subset \text{defer acc V A } p$

$\longrightarrow t \text{ (acc V A } p) \Longrightarrow \text{accum} = (\text{acc}, m, t, V, A, p) \Longrightarrow P$ **and**

$\bigwedge \text{acc } V \ A \ p \ m \ t.$
 $\neg (\text{finite } (\text{defer } \text{acc } V \ A \ p) \wedge \text{defer } (\text{acc } \triangleright m) \ V \ A \ p \subset \text{defer } \text{acc } V \ A \ p$
 $\longrightarrow t (\text{acc } V \ A \ p)) \implies \text{accum} = (\text{acc}, m, t, V, A, p) \implies P$
thus P
using *accum-exists*
by *metis*
next
fix
 $t \ t' :: 'a \ \text{Termination-Condition} \ \mathbf{and}$
 $\text{acc } \text{acc}' :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module} \ \mathbf{and}$
 $A \ A' :: 'a \ \text{set} \ \mathbf{and}$
 $V \ V' :: 'v \ \text{set} \ \mathbf{and}$
 $p \ p' :: ('a, 'v) \ \text{Profile} \ \mathbf{and}$
 $m \ m' :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$
assume
 $\text{finite } (\text{defer } \text{acc } V \ A \ p)$
 $\wedge \text{defer } (\text{acc } \triangleright m) \ V \ A \ p \subset \text{defer } \text{acc } V \ A \ p$
 $\longrightarrow t (\text{acc } V \ A \ p) \ \mathbf{and}$
 $\text{finite } (\text{defer } \text{acc}' \ V' \ A' \ p')$
 $\wedge \text{defer } (\text{acc}' \triangleright m') \ V' \ A' \ p' \subset \text{defer } \text{acc}' \ V' \ A' \ p'$
 $\longrightarrow t' (\text{acc}' \ V' \ A' \ p') \ \mathbf{and}$
 $(\text{acc}, m, t, V, A, p) = (\text{acc}', m', t', V', A', p')$
thus $\text{acc } V \ A \ p = \text{acc}' \ V' \ A' \ p'$
by *fastforce*
next
fix
 $t \ t' :: 'a \ \text{Termination-Condition} \ \mathbf{and}$
 $\text{acc } \text{acc}' :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module} \ \mathbf{and}$
 $A \ A' :: 'a \ \text{set} \ \mathbf{and}$
 $V \ V' :: 'v \ \text{set} \ \mathbf{and}$
 $p \ p' :: ('a, 'v) \ \text{Profile} \ \mathbf{and}$
 $m \ m' :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$
assume
 $\text{finite } (\text{defer } \text{acc } V \ A \ p)$
 $\wedge \text{defer } (\text{acc } \triangleright m) \ V \ A \ p \subset \text{defer } \text{acc } V \ A \ p$
 $\longrightarrow t (\text{acc } V \ A \ p) \ \mathbf{and}$
 $\neg (\text{finite } (\text{defer } \text{acc}' \ V' \ A' \ p')$
 $\wedge \text{defer } (\text{acc}' \triangleright m') \ V' \ A' \ p' \subset \text{defer } \text{acc}' \ V' \ A' \ p'$
 $\longrightarrow t' (\text{acc}' \ V' \ A' \ p')) \ \mathbf{and}$
 $(\text{acc}, m, t, V, A, p) = (\text{acc}', m', t', V', A', p')$
thus $\text{acc } V \ A \ p = \text{loop-comp-helper-sumC } (\text{acc}' \triangleright m', m', t', V', A', p')$
by *force*
next
fix
 $t \ t' :: 'a \ \text{Termination-Condition} \ \mathbf{and}$
 $\text{acc } \text{acc}' :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module} \ \mathbf{and}$
 $A \ A' :: 'a \ \text{set} \ \mathbf{and}$
 $V \ V' :: 'v \ \text{set} \ \mathbf{and}$
 $p \ p' :: ('a, 'v) \ \text{Profile} \ \mathbf{and}$


```

    m m' :: ('a, 'v, 'a Result) Electoral-Module
assume
  ¬ (finite (defer acc V A p)
    ∧ defer (acc ▷ m) V A p ⊆ defer acc V A p
      → t (acc V A p)) and
  ¬ (finite (defer acc' V' A' p')
    ∧ defer (acc' ▷ m') V' A' p' ⊆ defer acc' V' A' p'
      → t' (acc' V' A' p')) and
  (acc, m, t, V, A, p) = (acc', m', t', V', A', p')
thus loop-comp-helper-sumC (acc ▷ m, m, t, V, A, p) =
  loop-comp-helper-sumC (acc' ▷ 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:
  ∃ R. wf R ∧
    (finite (defer m V A p)
      ∧ defer (m ▷ n) V A p ⊆ defer m V A p
        → t (m V A p)
          ∨ ((m ▷ n, n, t, V, A, p), (m, n, t, V, A, p)) ∈ R)
using loop-termination-helper wf-measure termination
by (metis (no-types))
obtain
  R :: (((('b, 'a, 'b Result) Electoral-Module
    × ('b, 'a, 'b Result) Electoral-Module
    × ('b Termination-Condition) × 'a set × 'b set
    × ('b, 'a) Profile)
    × ('b, 'a, 'b Result) Electoral-Module
    × ('b, 'a, 'b Result) Electoral-Module
    × ('b Termination-Condition) × 'a set × 'b set
    × ('b, 'a) Profile) set where
    wf R ∧
    (finite (defer m V A p)
      ∧ defer (m ▷ n) V A p ⊆ defer m V A p
        → t (m V A p)
          ∨ ((m ▷ n, n, t, V, A, p), m, n, t, V, A, p) ∈ R)
using term-rel
by presburger
have ∀ R'.
  All (loop-comp-helper-dom ::
    ('b, 'a, 'b Result) Electoral-Module × ('b, 'a, 'b Result) Electoral-Module
    × 'b Termination-Condition × 'a set × 'b set × ('b, 'a) Profile ⇒ bool) ∨

```

```

    (∃ t' m' A' V' p' n'. wf R' →
      ((m' ▷ n', n', t', V' :: 'a set, A' :: 'b set, p'), m', n', t', V', A', p') ∉ R'
      ∧ finite (defer m' V' A' p') ∧ defer (m' ▷ n') V' A' p' ⊂ defer m' V' A' p'
      ∧ ¬ 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) ∨ ¬ ((defer (acc ▷ m) V A p) ⊂ (defer acc V A p))
      ∨ infinite (defer acc V A p))
      then (acc V A p) else (loop-comp-helper (acc ▷ m) m t V A p))

```

using *loop-comp-helper.simps*

by (*metis (no-types)*)

function *loop-composition* :: ('a, 'v, 'a Result) Electoral-Module ⇒

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

```

  t ({}, {}, A)
    ⇒ loop-composition m t V A p = defer-module V A p |
  ¬(t ({}, {}, A))
    ⇒ 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 ⇒

'a Termination-Condition ⇒ ('a, 'v, 'a Result) Electoral-Module

(- ∘_t 50) **where**

$m \circ_t \equiv \text{loop-composition } m \ t$

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* $m\ t\ V\ A\ p =$
 $(\text{if } (t\ (\{\}, \{\}, A))$
 $\text{then } (\text{defer-module } V\ A\ p) \text{ else } (\text{loop-comp-helper } m\ m\ t)\ V\ A\ p)$
by *simp*

lemma *loop-comp-helper-imp-partit*:
fixes
 $m\ \text{acc} :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$ **and**
 $t :: 'a\ \text{Termination-Condition}$ **and**
 $A :: 'a\ \text{set}$ **and**
 $V :: 'v\ \text{set}$ **and**
 $p :: ('a, 'v)\ \text{Profile}$ **and**
 $n :: \text{nat}$
assumes
 $\text{module-m: } \text{SCF-result.electoral-module } m$ **and**
 $\text{profile: } \text{profile } V\ A\ p$ **and**
 $\text{module-acc: } \text{SCF-result.electoral-module } \text{acc}$ **and**
 $\text{defer-card-n: } n = \text{card } (\text{defer } \text{acc } V\ A\ p)$
shows *well-formed-SCF* $A\ (\text{loop-comp-helper } \text{acc } m\ t\ V\ A\ p)$
using *assms*
proof (*induct arbitrary: acc rule: less-induct*)
case (*less*)
have $\forall\ m'\ n'.$
 $(\text{SCF-result.electoral-module } m' \wedge \text{SCF-result.electoral-module } n')$
 $\longrightarrow \text{SCF-result.electoral-module } (m' \triangleright n')$
using *seq-comp-sound*
by *metis*
hence $\text{SCF-result.electoral-module } (\text{acc } \triangleright m)$
using *less.premis module-m*
by *blast*
hence $\neg t\ (\text{acc } V\ A\ p) \wedge \text{defer } (\text{acc } \triangleright m)\ V\ A\ p \subset \text{defer } \text{acc } V\ A\ p \wedge$
 $\text{finite } (\text{defer } \text{acc } V\ A\ p) \longrightarrow$
 $\text{well-formed-SCF } A\ (\text{loop-comp-helper } \text{acc } m\ t\ V\ A\ p)$
using *less.hyps less.premis loop-comp-helper-infinite*
 psubset-card-mono
by *metis*
moreover have *well-formed-SCF* $A\ (\text{acc } V\ A\ p)$
using *less.premis profile*
unfolding *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 SCF-result.electoral-module m
  shows SCF-result.electoral-module (m  $\odot_t$ )
  using def-mod-sound loop-composition.simps
        loop-comp-helper-imp-partit assms
  unfolding 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: SCF-result.electoral-module m and
    profile: profile V A p and
    mod-acc: 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  $\subseteq$  defer acc V A p
  using assms
proof (induct arbitrary: acc rule: less-induct)
  case (less)
  have emod-acc-m: SCF-result.electoral-module (acc  $\triangleright$  m)
    using less.prem module-m seq-comp-sound
  by blast
  have  $\forall A A'. (\text{finite } A \wedge A' \subset A) \longrightarrow \text{card } A' < \text{card } A$ 
    using psubset-card-mono
  by metis
  hence  $\neg t (acc \ V \ A \ p) \wedge \text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p \wedge$ 
    finite (defer acc V A p)  $\longrightarrow$ 
    defer (loop-comp-helper (acc  $\triangleright$  m) m t) V A p  $\subseteq$  defer acc V A p
    using emod-acc-m less.hyps less.prem
  by blast
  hence  $\neg t (acc \ V \ A \ p) \wedge \text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p \wedge$ 
    finite (defer acc V A p)  $\longrightarrow$ 
    defer (loop-comp-helper acc m t) V A p  $\subseteq$  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

$m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**

$t :: 'a \text{ Termination-Condition}$ **and**

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p :: ('a, 'v) \text{ Profile}$ **and**

$n :: \text{nat}$

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

$\forall q \ a. a \in (\text{defer } (\text{loop-comp-helper acc m } t) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$
 $(\text{loop-comp-helper acc m } t) \ V \ A \ p = (\text{loop-comp-helper acc m } t) \ V \ A \ q$

using *assms*

proof (*induct n arbitrary: acc rule: less-induct*)

case (*less n*)

have *defer-card-comp:*

defer-lift-invariance acc \longrightarrow

$(\forall q \ a. a \in (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$
 $\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) = \text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ q))$

using *monotone-m def-lift-inv-seq-comp-help voters-determine-m*

by *metis*

have *defer-lift-invariance acc* \longrightarrow

$(\forall q \ a. a \in (\text{defer acc } V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$
 $\text{card } (\text{defer acc } V \ A \ p) = \text{card } (\text{defer acc } V \ A \ q))$

unfolding *defer-lift-invariance-def*

by *simp*

hence *defer-card-acc:*

defer-lift-invariance acc \longrightarrow

$(\forall q \ a. (a \in (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a) \longrightarrow$
 $\text{card } (\text{defer acc } V \ A \ p) = \text{card } (\text{defer acc } V \ A \ q))$

using *assms seq-comp-def-set-trans*

unfolding *defer-lift-invariance-def*

by *metis*

thus *?case*

proof (*cases*)

assume *card-unchanged:*

$\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) = \text{card } (\text{defer acc } V \ A \ p)$

have *defer-lift-invariance acc* \longrightarrow

$(\forall q \ a. a \in (\text{defer acc } V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$
 $(\text{loop-comp-helper acc m } t) \ V \ A \ q = \text{acc } V \ A \ q)$

proof (*safe*)

fix

```

  q :: ('a, 'v) Profile and
  a :: 'a
assume
  dli-acc: defer-lift-invariance acc and
  a-in-def-acc: a ∈ 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)
  → 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) ∧ defer-lift-invariance acc
  → (∀ 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)
  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:
  ¬ (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  $\triangleright$  m) V A p)  $<$  card (defer acc V A p)
using monotone-m order.not-eq-order-implies-strict
      card-mono less.premis 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  $\longrightarrow$ 
     $(\forall q\ a.\ a \in (\text{defer } (acc \triangleright m) \ V\ A\ p) \wedge \text{lifted } V\ A\ p\ q\ a \longrightarrow$ 
      card (defer (acc  $\triangleright$  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:  $\neg t\ (acc\ V\ A\ p)$ 
hence t-not-satisfied-for-q:
  defer-lift-invariance acc  $\longrightarrow$ 
     $(\forall q\ a.\ a \in (\text{defer } (acc \triangleright m) \ V\ A\ p) \wedge \text{lifted } V\ A\ p\ q\ a$ 
       $\longrightarrow \neg t\ (acc\ V\ A\ q))$ 
using monotone-m prof seq-comp-def-set-trans
unfolding defer-lift-invariance-def
by metis
have dli-card-defer:
  defer-lift-invariance (acc  $\triangleright$  m)  $\wedge$  defer-lift-invariance acc
     $\longrightarrow (\forall q\ a.\ a \in (\text{defer } (acc \triangleright m) \ V\ A\ p) \wedge \text{Profile.lifted } V\ A\ p\ q\ a$ 
       $\longrightarrow \text{card } (\text{defer } (acc \triangleright m) \ V\ A\ q) \neq (\text{card } (\text{defer } acc\ V\ A\ q)))$ 
proof –
have
   $\forall m'.$ 
     $(\neg \text{defer-lift-invariance } m' \wedge \text{SCF-result.electoral-module } m'$ 
       $\longrightarrow (\exists V'\ A'\ p'\ q'\ a.$ 
         $m'\ V'\ A'\ p' \neq m'\ V'\ A'\ q' \wedge \text{lifted } V'\ A'\ p'\ q'\ a$ 
         $\wedge a \in \text{defer } m'\ V'\ A'\ p'))$ 
     $\wedge (\text{defer-lift-invariance } m'$ 
       $\longrightarrow \text{SCF-result.electoral-module } m'$ 
         $\wedge (\forall V'\ A'\ p'\ q'\ a.$ 
           $m'\ V'\ A'\ p' \neq m'\ V'\ A'\ q'$ 
           $\longrightarrow \text{lifted } V'\ A'\ p'\ q'\ a \longrightarrow a \notin \text{defer } m'\ V'\ A'\ p'))$ 
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:

```

$defer\text{-}lift\text{-}invariance\ (acc \triangleright m) \wedge defer\text{-}lift\text{-}invariance\ acc$
 $\longrightarrow (\forall\ p'\ a.\ a \in (defer\ (acc \triangleright m)\ V\ A\ p) \wedge lifted\ V\ A\ p\ p'\ a$
 $\longrightarrow defer\ (acc \triangleright m)\ V\ A\ p' \subseteq defer\ acc\ V\ A\ p')$
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*:
 $defer\text{-}lift\text{-}invariance\ (acc \triangleright m) \wedge defer\text{-}lift\text{-}invariance\ acc$
 $\longrightarrow (\forall\ q\ a.\ a \in (defer\ (acc \triangleright m)\ V\ A\ p) \wedge lifted\ V\ A\ p\ q\ a$
 $\longrightarrow loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ q =$
 $loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ q)$
proof (*safe*)
fix
 $q :: ('a, 'v)\ Profile\ \text{and}$
 $a :: 'a$
assume
 $a\text{-}in\text{-}def\text{-}imp\text{-}def\text{-}subset$:
 $\forall\ q'\ a'.\ a' \in defer\ (acc \triangleright m)\ V\ A\ p \wedge lifted\ V\ A\ p\ q'\ a' \longrightarrow$
 $defer\ (acc \triangleright m)\ V\ A\ q' \subseteq defer\ acc\ V\ A\ q'\ \text{and}$
 $dli\text{-}acc$: $defer\text{-}lift\text{-}invariance\ acc\ \text{and}$
 $a\text{-}in\text{-}def\text{-}seq\text{-}acc\text{-}m$: $a \in defer\ (acc \triangleright m)\ V\ A\ p\ \text{and}$
 $lifted\text{-}pq\text{-}a$: $lifted\ V\ A\ p\ q\ a$
hence $defer\ (acc \triangleright m)\ V\ A\ q \subseteq defer\ acc\ V\ A\ q$
by *metis*
moreover have *SCF-result.electoral-module acc*
using *dli-acc*
unfolding *defer-lift-invariance-def*
by *simp*
moreover have $\neg t\ (acc\ V\ A\ q)$
using *dli-acc a-in-def-seq-acc-m lifted-pq-a t-not-satisfied-for-q*
by *metis*
ultimately show $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ q$
 $= loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ q$
using *loop-comp-code-helper defer-in-alts finite-subset lifted-pq-a*
unfolding *lifted-def*
by (*metis (mono-tags, lifting)*)
qed
have *rec-step-p*:
 $SCF\text{-}result.electoral\text{-}module\ acc \longrightarrow$
 $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ p$
proof (*safe*)
assume *emod-acc: SCF-result.electoral-module acc*
have *sound-imp-defer-subset*:
 $SCF\text{-}result.electoral\text{-}module\ m$
 $\longrightarrow defer\ (acc \triangleright m)\ V\ A\ p \subseteq defer\ acc\ V\ A\ p$
using *emod-acc prof seq-comp-def-set-bounded*
by *blast*
hence $card\text{-}ineq$: $card\ (defer\ (acc \triangleright m)\ V\ A\ p) < card\ (defer\ acc\ V\ A\ p)$


```

    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.premis
    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.premis 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 p
  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-alt 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 SCF-result.electoral-module (loop-comp-helper acc m t)
    using loop-comp-helper-imp-partit assms
    unfolding 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 ∈ 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

```

$m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$
 $t :: 'a \text{ Termination-Condition and}$
 $A :: 'a \text{ set and}$
 $V :: 'v \text{ set and}$
 $p :: ('a, 'v) \text{ Profile and}$
 $n :: \text{nat}$
assumes
 $\text{non-electing-m: non-electing } m \text{ and}$
 $\text{non-electing-acc: non-electing } acc \text{ and}$
 $\text{prof: profile } V \ A \ p \text{ and}$
 $\text{acc-defer-card: } n = \text{card } (\text{defer } acc \ V \ A \ p)$
shows $\text{elect } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p = \{\}$
using $\text{acc-defer-card non-electing-acc}$
proof ($\text{induct } n \text{ arbitrary: acc rule: less-induct}$)
case ($\text{less } n$)
thus $?case$
proof (safe)
fix $x :: 'a$
assume
 acc-no-elect:
 $(\bigwedge i \text{ acc'. } i < \text{card } (\text{defer } acc \ V \ A \ p) \implies$
 $i = \text{card } (\text{defer } acc' \ V \ A \ p) \implies \text{non-electing } acc' \implies$
 $\text{elect } (\text{loop-comp-helper } acc' \ m \ t) \ V \ A \ p = \{\}) \text{ and}$
 $\text{acc-non-elect: non-electing } acc \text{ and}$
 $\text{x-in-acc-elect: } x \in \text{elect } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p$
have $\forall m' n'. \text{non-electing } m' \wedge \text{non-electing } n' \longrightarrow \text{non-electing } (m' \triangleright n')$
by simp
hence $\text{seq-acc-m-non-elect: non-electing } (acc \triangleright m)$
using $\text{acc-non-elect non-electing-m}$
by blast
have $\forall i m'.$
 $i < \text{card } (\text{defer } acc \ V \ A \ p) \wedge i = \text{card } (\text{defer } m' \ V \ A \ p) \wedge$
 $\text{non-electing } m' \longrightarrow$
 $\text{elect } (\text{loop-comp-helper } m' \ m \ t) \ V \ A \ p = \{\}$
using acc-no-elect
by blast
hence $\forall m'.$
 $\text{finite } (\text{defer } acc \ V \ A \ p) \wedge \text{defer } m' \ V \ A \ p \subset \text{defer } acc \ V \ A \ p \wedge$
 $\text{non-electing } m' \longrightarrow$
 $\text{elect } (\text{loop-comp-helper } m' \ m \ t) \ V \ A \ p = \{\}$
using psubset-card-mono
by metis
hence $\neg t \ (acc \ V \ A \ p) \wedge \text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p \wedge$
 $\text{finite } (\text{defer } acc \ V \ A \ p) \longrightarrow$
 $\text{elect } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p = \{\}$
using $\text{loop-comp-code-helper seq-acc-m-non-elect}$
by (metis (no-types))
moreover have $\text{elect } acc \ V \ A \ p = \{\}$
using $\text{acc-non-elect prof non-electing-def}$

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: $\forall r. t\ r = (\text{card } (\text{defer-r } r) = x)$ **and**
x-greater-zero: $x > 0$ **and**
prof: *profile V A p* **and**
n-acc-defer-card: $n = \text{card } (\text{defer } \text{acc } V\ A\ p)$ **and**
n-ge-x: $n \geq x$ **and**
def-card-gt-one: $\text{card } (\text{defer } \text{acc } V\ A\ p) > 1$ **and**
acc-nonelect: *non-electing acc*

shows $\text{card } (\text{defer } (\text{loop-comp-helper } \text{acc } m\ t)\ V\ A\ p) = x$

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*: *SCF-result.electoral-module acc*

using *less*

unfolding *non-electing-def*

by *metis*

hence *step-reduces-defer-set*: $\text{defer } (\text{acc } \triangleright m)\ V\ A\ p \subset \text{defer } \text{acc } V\ A\ p$

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*: $t\ (\text{acc } V\ A\ p)$

thus $\text{card } (\text{defer-r } (\text{loop-comp-helper } \text{acc } m\ t\ V\ A\ p)) = x$

using *loop-comp-code-helper term-satisfied terminate-if-n-left*

by *metis*

next

case *False*

hence *card-not-eq-x*: $\text{card } (\text{defer } \text{acc } V\ A\ p) \neq x$

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
     $\forall r. (t\ r) = (\text{card } (\text{defer}\text{-}r\ r) = x)$  and
     $x > 0$  and
    profile V A p and
     $\text{card } (\text{defer } \text{acc } V\ A\ p) \geq x$  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:  $\forall r. (t\ r) = (\text{card } (\text{defer-r } r) = x)$  and
    x-greater-zero:  $x > 0$  and
    prof: profile  $V\ A\ p$  and
    enough-alternatives:  $\text{card } A \geq x$ 
  shows  $\text{card } (\text{defer } (m \circ_t) V\ A\ p) = x$ 
proof (cases)
  assume  $\text{card } A = x$ 
  thus ?thesis
    using terminate-if-n-left
    by simp
next
  assume card-not-x:  $\neg \text{card } A = x$ 
  thus ?thesis
proof (cases)
  assume  $\text{card } A < x$ 
  thus ?thesis
    using enough-alternatives not-le
    by blast
next
  assume  $\neg \text{card } A < x$ 
  hence  $\text{card } A > x$ 
    using card-not-x
    by linarith
  moreover from this
  have  $\text{card } (\text{defer } m\ V\ A\ p) = \text{card } A - 1$ 
    using non-electing-m single-elimination single-elim-decr-def-card'
    prof x-greater-zero
    by fastforce
  ultimately have  $\text{card } (\text{defer } m\ V\ A\ p) \geq x$ 
    by linarith
  moreover have  $(m \circ_t) V\ A\ p = (\text{loop-comp-helper } m\ m\ t) V\ A\ p$ 
    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
     $m :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$  and
     $t :: 'a\ \text{Termination-Condition}$ 

```



```

assumes
  defer-lift-invariance  $m$  and
  voters-determine-election  $m$ 
shows defer-lift-invariance  $(m \circ_t)$ 
proof (unfold defer-lift-invariance-def, safe)
  have SCF-result.electoral-module  $m$ 
    using assms
    unfolding defer-lift-invariance-def
    by simp
  thus SCF-result.electoral-module  $(m \circ_t)$ 
    using loop-comp-sound
    by blast
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p \ q :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 
assume
   $a \in \text{defer } (m \circ_t) \ V \ A \ p$  and
  lifted  $V \ A \ p \ q \ a$ 
moreover have
   $\forall \ p' \ q' \ a'. \ a' \in (\text{defer } (m \circ_t) \ V \ A \ p') \wedge \text{lifted } V \ A \ p' \ q' \ a' \longrightarrow$ 
     $(m \circ_t) \ V \ A \ p' = (m \circ_t) \ V \ A \ q'$ 
    using assms lifted-imp-fin-prof loop-comp-helper-def-lift-inv
    loop-composition.simps defer-module.simps
    by (metis full-types)
  ultimately show  $(m \circ_t) \ V \ A \ p = (m \circ_t) \ V \ A \ q$ 
    by metis
qed

```

The loop composition preserves the property non-electing.

```

theorem loop-comp-presv-non-electing[simp]:
fixes
   $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $t :: 'a \text{ Termination-Condition}$ 
assumes non-electing  $m$ 
shows non-electing  $(m \circ_t)$ 
proof (unfold non-electing-def, safe)
show SCF-result.electoral-module  $(m \circ_t)$ 
  using loop-comp-sound assms
  unfolding non-electing-def
  by metis
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 

```

```

assume
  profile  $V\ A\ p$  and
   $a \in \text{elect}\ (m \circlearrowleft_t)\ V\ A\ p$ 
thus  $a \in \{\}$ 
  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
     $m :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$  and
     $t :: 'a\ \text{Termination-Condition}$  and
     $n :: \text{nat}$ 
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left:  $\forall\ r.\ t\ r = (\text{card}\ (\text{defer-r}\ r) = n)$  and
    x-greater-zero:  $n > 0$ 
  shows defers  $n\ (m \circlearrowleft_t)$ 
proof (unfold defers-def, safe)
  show SCF-result.electoral-module  $(m \circlearrowleft_t)$ 
    using loop-comp-sound non-electing-m
    unfolding non-electing-def
    by metis
next
  fix
     $A :: 'a\ \text{set}$  and
     $V :: 'v\ \text{set}$  and
     $p :: ('a, 'v)\ \text{Profile}$ 
  assume
     $n \leq \text{card}\ A$  and
    profile  $V\ A\ p$ 
  thus  $\text{card}\ (\text{defer}\ (m \circlearrowleft_t)\ V\ A\ p) = n$ 
    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-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
 SCF-result.electoral-module m **and**
 SCF-result.electoral-module n
 shows *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: *SCF-result.electoral-module* m **and**
module-n: *SCF-result.electoral-module* n **and**
prof-p: *profile* $V\ A\ p$ **and**
a-in-A: $a \in A$

shows *mod-contains-result* $(m \parallel_{\uparrow} n)\ m\ V\ A\ p\ a \vee$
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 $m'\ n'\ V'\ A'\ p'\ (a' :: 'a) =$
 $(SCF-result.electoral-module\ m'$
 $\wedge\ SCF-result.electoral-module\ n'$
 $\wedge\ profile\ V'\ A'\ p' \wedge a' \in A'$
 $\wedge\ (a' \notin elect\ m'\ V'\ A'\ p' \vee a' \in elect\ n'\ V'\ A'\ p')$
 $\wedge\ (a' \notin reject\ m'\ V'\ A'\ p' \vee a' \in reject\ n'\ V'\ A'\ p')$
 $\wedge\ (a' \notin defer\ m'\ V'\ A'\ p' \vee a' \in defer\ n'\ V'\ A'\ p'))$

unfolding *mod-contains-result-def*

by *simp*

moreover have *module-mn*: *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:  $a \in \text{defer } (m \parallel_{\uparrow} n) \vee A \ p$ 
  thus ?thesis
proof (safe)
  assume not-mod-cont-mn:  $\neg \text{mod-contains-result } (m \parallel_{\uparrow} n) \ n \vee A \ p \ a$ 
  have par-emod:  $\forall \ m' \ n'.$ 
     $\text{SCF-result.electoral-module } m' \wedge$ 
     $\text{SCF-result.electoral-module } n' \longrightarrow$ 
     $\text{SCF-result.electoral-module } (m' \parallel_{\uparrow} n')$ 
  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-SCF  $A \ (n \vee A \ p)$ 
  using prof-p module-n
  unfolding SCF-result.electoral-module.simps
  by blast
  have wf-m: well-formed-SCF  $A \ (m \vee A \ p)$ 
  using prof-p module-m
  unfolding SCF-result.electoral-module.simps
  by blast
  have e-mod-par:  $\text{SCF-result.electoral-module } (m \parallel_{\uparrow} n)$ 
  using par-emod module-m module-n
  by blast
  hence SCF-result.electoral-module  $(m \parallel_{\text{max-aggregator}} n)$ 
  by simp
  hence result-disj-max:
     $\text{elect } (m \parallel_{\text{max-aggregator}} n) \vee A \ p \cap$ 
     $\text{reject } (m \parallel_{\text{max-aggregator}} n) \vee A \ p = \{\}$   $\wedge$ 
     $\text{elect } (m \parallel_{\text{max-aggregator}} n) \vee A \ p \cap$ 
     $\text{defer } (m \parallel_{\text{max-aggregator}} n) \vee A \ p = \{\}$   $\wedge$ 
     $\text{reject } (m \parallel_{\text{max-aggregator}} n) \vee A \ p \cap$ 
     $\text{defer } (m \parallel_{\text{max-aggregator}} n) \vee A \ p = \{\}$ 
  using prof-p result-disj
  by metis
  have a-not-elect:  $a \notin \text{elect } (m \parallel_{\text{max-aggregator}} n) \vee A \ p$ 
  using result-disj-max a-in-defer
  by force
  have result-m:  $(\text{elect } m \vee A \ p, \text{reject } m \vee A \ p, \text{defer } m \vee A \ p) = m \vee A \ p$ 
  by auto
  have result-n:  $(\text{elect } n \vee A \ p, \text{reject } n \vee A \ p, \text{defer } n \vee A \ p) = n \vee A \ p$ 
  by auto
  have max-pq:
     $\forall \ (A' :: 'a \text{ set}) \ m' \ n'.$ 
     $\text{elect-r } (\text{max-aggregator } A' \ m' \ n') = \text{elect-r } m' \cup \text{elect-r } n'$ 
  by force
  have  $a \notin \text{elect } (m \parallel_{\text{max-aggregator}} n) \vee A \ p$ 
  using a-not-elect
  by blast

```

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

```

    unfolding mod-contains-result-def
    by simp
  hence  $a \notin \text{defer } n \ V \ A \ p$ 
    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  $a \in \text{reject } n \ V \ A \ p$ 
    using a-in-A a-not-elect-mn module-n not-rej-imp-elec-or-defer prof-p
    by metis
  hence  $a \notin \text{reject } m \ V \ A \ p$ 
    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  $a \notin \text{defer } (m \parallel_{\uparrow} n) \ V \ A \ p \vee a \in \text{defer } m \ V \ A \ p$ 
    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:  $a \notin \text{defer } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
  have el-rej-defer:  $(\text{elect } m \ V \ A \ p, \text{reject } m \ V \ A \ p, \text{defer } m \ V \ A \ p) = m \ V \ A \ p$ 
    by auto
  from not-a-elect not-a-defer
  have a-reject:  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
    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  $(e', r', d') \Rightarrow$ 
       $a \in \text{reject-r } (\text{max-aggregator } A \ (\text{elect } m \ V \ A \ p, r, d) \ (e', r', d'))$ 
    using el-rej-defer
    by force
  hence let  $(e, r, d) = m \ V \ A \ p;$ 
     $(e', r', d') = n \ V \ A \ p$  in
     $a \in \text{reject-r } (\text{max-aggregator } A \ (e, r, d) \ (e', r', d'))$ 
    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  $a \notin \text{elect } m \ V \ A \ p \cup (\text{defer } n \ V \ A \ p \cup \text{defer } m \ V \ A \ p)$ 
    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
    SCF-result.electoral-module m and
    SCF-result.electoral-module n
  shows (a ∈ reject (m ||↑ n) V A p) =
    (a ∈ reject m V A p ∧ a ∈ reject n V A p)
proof
  assume rej-a: a ∈ reject (m ||↑ n) V A p
  hence case n V A p of (e, r, d) ⇒
    a ∈ reject-r (max-aggregator A
      (elect m V A p, reject m V A p, defer m V A p) (e, r, d))
  by auto
  hence case snd (m V A p) of (r, d) ⇒
    case n V A p of (e', r', d') ⇒
      a ∈ reject-r (max-aggregator A (elect m V A p, r, d) (e', r', d'))
  by force
  with rej-a
  have let (e, r, d) = m V A p;
    (e', r', d') = n V A p in
    a ∈ reject-r (max-aggregator A (e, r, d) (e', r', d'))
  unfolding prod.case-eq-if
  by simp
  hence let (e, r, d) = m V A p;
    (e', r', d') = n V A p in
    a ∈ A - (e ∪ e' ∪ d ∪ d')
  by simp
  hence
    a ∈ A - (elect m V A p ∪ elect n V A p ∪ defer m V A p ∪ defer n V A p)
  by auto
  thus a ∈ reject m V A p ∧ a ∈ reject n V A p
    using Diff-iff Un-iff electoral-mod-defer-elem assms
    by metis
next
  assume a ∈ reject m V A p ∧ a ∈ reject n V A p
  moreover from this
  have a ∉ elect m V A p ∧ a ∉ defer m V A p

```


$\wedge a \notin \text{elect } n \ V \ A \ p \wedge a \notin \text{defer } n \ V \ A \ p$
using *IntI empty-iff assms result-disj*
by *metis*
ultimately show $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$
using *DiffD1 max-agg-eq-result mod-contains-result-comm mod-contains-result-def*
 $\text{reject-not-elected-or-deferred assms}$
by (*metis (no-types)*)
qed

lemma *max-agg-rej-fst-imp-seq-contained:*

fixes
 $m \ n :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$ **and**
 $A :: 'a \ \text{set}$ **and**
 $V :: 'v \ \text{set}$ **and**
 $p :: ('a, 'v) \ \text{Profile}$ **and**
 $a :: 'a$
assumes
 $f\text{-prof}: \text{finite-profile } V \ A \ p$ **and**
 $\text{module-m}: \text{SCF-result.electoral-module } m$ **and**
 $\text{module-n}: \text{SCF-result.electoral-module } n$ **and**
 $\text{rejected}: a \in \text{reject } n \ V \ A \ p$
shows $\text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$
using *assms*
proof (*unfold mod-contains-result-def, safe*)
show $\text{SCF-result.electoral-module } (m \parallel_{\uparrow} n)$
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 $a\text{-in-elect}: a \in \text{elect } m \ V \ A \ p$
hence $a\text{-not-reject}: a \notin \text{reject } m \ V \ A \ p$
using *disjoint-iff-not-equal f-prof module-m result-disj*
by *metis*
have $\text{reject } n \ V \ A \ p \subseteq A$
using *f-prof module-n*
by (*simp add: reject-in-alts*)
hence $a \in A$
using *in-mono rejected*
by *metis*
with $a\text{-in-elect } a\text{-not-reject}$
show $a \in \text{elect } (m \parallel_{\uparrow} n) \ V \ A \ p$
using *f-prof max-agg-eq-result module-m module-n rejected*
 $\text{max-agg-rej-iff-both-reject mod-contains-result-comm}$
 $\text{mod-contains-result-def}$
by *metis*
next

```

assume  $a \in \text{reject } m \ V \ A \ p$ 
hence  $a \in \text{reject } m \ V \ A \ p \wedge a \in \text{reject } n \ V \ A \ p$ 
  using rejected
  by simp
thus  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
  using f-prof max-agg-rej-iff-both-reject module-m module-n
  by (metis (no-types))
next
assume a-in-defer:  $a \in \text{defer } m \ V \ A \ p$ 
then obtain  $d :: 'a$  where
  defer-a:  $a = d \wedge d \in \text{defer } m \ V \ A \ p$ 
  by metis
have a-not-rej:  $a \notin \text{reject } m \ V \ A \ p$ 
  using disjoint-iff-not-equal f-prof defer-a module-m result-disj
  by (metis (no-types))
have
   $\forall m' A' V' p'. \quad$ 
    SCF-result.electoral-module  $m' \wedge \text{finite } A' \wedge \text{finite } V' \wedge \text{profile } V' A' p'$ 
     $\longrightarrow \text{elect } m' V' A' p' \cup \text{reject } m' V' A' p' \cup \text{defer } m' V' A' p' = A'$ 
  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  $a \in \text{defer } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
  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
   $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 
assumes
  finite-profile  $V \ A \ p$  and
  SCF-result.electoral-module  $m$  and
  SCF-result.electoral-module  $n$  and
   $a \in \text{reject } n \ V \ A \ p$ 
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  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
  thus  $a \in \text{reject } m \ V \ A \ p$ 

```

```

    using assms max-agg-rej-iff-both-reject
    by (metis (no-types))
next
  have mod-contains-result  $m \parallel_{\uparrow} n$   $V A p a$ 
    using assms max-agg-rej-fst-imp-seq-contained
    by (metis (full-types))
  thus
     $a \in \text{elect } m \parallel_{\uparrow} n \ V A p \implies a \in \text{elect } m \ V A p$  and
     $a \in \text{defer } m \parallel_{\uparrow} n \ V A p \implies a \in \text{defer } m \ V A p$ 
    using mod-contains-result-comm
    unfolding mod-contains-result-def
    by (metis (full-types), metis (full-types))
next
  show
     $\text{SCF-result.electoral-module } m \parallel_{\uparrow} n$  and
     $a \in A$ 
    using assms max-agg-rej-fst-imp-seq-contained
    unfolding mod-contains-result-def
    by (metis (full-types), metis (full-types))
next
  show
     $a \in \text{elect } m \ V A p \implies a \in \text{elect } m \parallel_{\uparrow} n \ V A p$  and
     $a \in \text{reject } m \ V A p \implies a \in \text{reject } m \parallel_{\uparrow} n \ V A p$  and
     $a \in \text{defer } m \ V A p \implies a \in \text{defer } m \parallel_{\uparrow} n \ V A p$ 
    using assms max-agg-rej-fst-imp-seq-contained
    unfolding mod-contains-result-def
    by (metis (no-types), metis (no-types), metis (no-types))
qed

lemma max-agg-rej-snd-imp-seq-contained:
  fixes
     $m n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  and
     $a :: 'a$ 
  assumes
     $f\text{-prof: finite-profile } V A p$  and
     $\text{module-m: SCF-result.electoral-module } m$  and
     $\text{module-n: SCF-result.electoral-module } n$  and
     $\text{rejected: } a \in \text{reject } m \ V A p$ 
  shows  $\text{mod-contains-result } n \parallel_{\uparrow} m \ V A p a$ 
  using assms
proof (unfold mod-contains-result-def, safe)
  show  $\text{SCF-result.electoral-module } m \parallel_{\uparrow} n$ 
    using module-m module-n max-par-comp-sound
    by metis
next
  show  $a \in A$ 

```

```

    using f-prof in-mono module-m reject-in-alts rejected
    by (metis (no-types))
next
  assume  $a \in \text{elect } n \ V \ A \ p$ 
  thus  $a \in \text{elect } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
    using max-aggregator.simps[of
       $A \ \text{elect } m \ V \ A \ p \ \text{reject } m \ V \ A \ p \ \text{defer } m \ V \ A \ p$ 
       $\text{elect } n \ V \ A \ p \ \text{reject } n \ V \ A \ p \ \text{defer } n \ V \ A \ p$ ]
    by simp
next
  assume  $a \in \text{reject } n \ V \ A \ p$ 
  thus  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
    using f-prof max-agg-rej-iff-both-reject module-m module-n rejected
    by metis
next
  assume  $a \in \text{defer } n \ V \ A \ p$ 
  moreover have  $a \in A$ 
    using f-prof max-agg-rej-fst-imp-seq-contained module-m rejected
    unfolding mod-contains-result-def
    by metis
  ultimately show  $a \in \text{defer } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
    using disjoint-iff-not-equal max-agg-eq-result max-agg-rej-iff-both-reject
      f-prof mod-contains-result-comm mod-contains-result-def
      module-m module-n rejected result-disj
    by (metis (no-types, opaque-lifting))
qed

lemma max-agg-rej-snd-equiv-seq-contained:
  fixes
     $m \ n :: ('a, 'v, 'a \ \text{Result}) \ \text{Electoral-Module}$  and
     $A :: 'a \ \text{set}$  and
     $V :: 'v \ \text{set}$  and
     $p :: ('a, 'v) \ \text{Profile}$  and
     $a :: 'a$ 
  assumes
    finite-profile  $V \ A \ p$  and
    SCF-result.electoral-module  $m$  and
    SCF-result.electoral-module  $n$  and
     $a \in \text{reject } m \ V \ A \ p$ 
  shows mod-contains-result-sym  $(m \parallel_{\uparrow} n) \ n \ V \ A \ p \ a$ 
  using assms
proof (unfold mod-contains-result-sym-def, safe)
  assume  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
  thus  $a \in \text{reject } n \ V \ A \ p$ 
    using assms max-agg-rej-iff-both-reject
    by (metis (no-types))
next
  have mod-contains-result  $n \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$ 
    using assms max-agg-rej-snd-imp-seq-contained

```

```

    by (metis (full-types))
  thus
    a ∈ elect (m ||↑ n) V A p ⇒ a ∈ elect n V A p and
    a ∈ defer (m ||↑ n) V A p ⇒ a ∈ defer n V A p
  using mod-contains-result-comm
  unfolding mod-contains-result-def
  by (metis (full-types), metis (full-types))
next
show
  SCF-result.electoral-module (m ||↑ n) and
  a ∈ 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 ∈ elect n V A p ⇒ a ∈ elect (m ||↑ n) V A p and
  a ∈ reject n V A p ⇒ a ∈ reject (m ||↑ n) V A p and
  a ∈ defer n V A p ⇒ a ∈ defer (m ||↑ 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-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    SCF-result.electoral-module m and
    SCF-result.electoral-module n and
    profile V A p and
    finite A
  shows reject (m ||↑ n) V A p = (reject m V A p) ∩ (reject n V A p)
proof -
  have A = (elect m V A p) ∪ (reject m V A p) ∪ (defer m V A p)
    ∧ A = (elect n V A p) ∪ (reject n V A p) ∪ (defer n V A p)
  using assms result-presv-alts
  by metis
  hence A - ((elect m V A p) ∪ (defer m V A p)) = (reject m V A p)
    ∧ A - ((elect n V A p) ∪ (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) ∪ (elect n V A p)
      ∪ (defer m V A p) ∪ (defer n V A p)) =
    (reject m V A p) ∩ (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

$m \ n :: ('a, 'v, 'a \ Result) \ Electoral\text{-}Module$ **and**

$A :: 'a \ set$ **and**

$V :: 'v \ set$ **and**

$a :: 'a$

assumes

disjoint-compatibility $m \ n$ **and**

$a \in A$

shows

$(\forall \ p. \text{finite-profile } V \ A \ p \longrightarrow \text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a)$

$\vee (\forall \ p. \text{finite-profile } V \ A \ p \longrightarrow \text{mod-contains-result } n \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a)$

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 $m \ n :: ('a, 'v, 'a \ Result) \ Electoral\text{-}Module$

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 $m \ n :: ('a, 'v, 'a \ Result) \ Electoral\text{-}Module$

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: SCF-result.electoral-module m
  using monotone-m
  unfolding defer-lift-invariance-def
  by simp
moreover have mod-n: SCF-result.electoral-module n
  using monotone-n
  unfolding defer-lift-invariance-def
  by simp
ultimately show 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-of-alt m V A b  $\wedge$ 
      ( $\forall$  p'. finite-profile V A p'  $\longrightarrow$  b  $\in$  reject m V A p'))
     $\wedge$  ( $\forall$  b  $\in$  A - B. indep-of-alt n V A b  $\wedge$ 
      ( $\forall$  p'. finite-profile V A p'  $\longrightarrow$  b  $\in$  reject n V A p'))
  using f-profs
  unfolding disjoint-compatibility-def
  by (metis (no-types, lifting))
have  $\forall$  b  $\in$  A. prof-contains-result (m  $\parallel_{\uparrow}$  n) V A p q b
proof (cases)
  assume a-in-B: a  $\in$  B
  hence a  $\in$  reject m V A p
    using alts f-profs
    by blast
  with defer-a
  have defer-n: a  $\in$  defer n V A p
    using compatible f-profs max-agg-rej-snd-equiv-seq-contained
    unfolding disjoint-compatibility-def mod-contains-result-sym-def
    by metis
  have  $\forall$  b  $\in$  B. mod-contains-result-sym (m  $\parallel_{\uparrow}$  n) n V A p b
    using alts compatible max-agg-rej-snd-equiv-seq-contained f-profs
    unfolding disjoint-compatibility-def
    by metis
  moreover have  $\forall$  b  $\in$  A. prof-contains-result n V A p q b
  proof (unfold prof-contains-result-def, clarify)

```

```

fix  $b :: 'a$ 
assume  $b\text{-in-}A$ :  $b \in A$ 
show  $SCF\text{-result.electoral-module } n \wedge \text{profile } V A p$ 
   $\wedge \text{profile } V A q \wedge b \in A \wedge$ 
   $(b \in \text{elect } n V A p \longrightarrow b \in \text{elect } n V A q) \wedge$ 
   $(b \in \text{reject } n V A p \longrightarrow b \in \text{reject } n V A q) \wedge$ 
   $(b \in \text{defer } n V A p \longrightarrow b \in \text{defer } n V A q)$ 
proof (safe)
  show  $SCF\text{-result.electoral-module } n$ 
    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
   $b \in \text{elect } n V A p \Longrightarrow b \in \text{elect } n V A q$  and
   $b \in \text{reject } n V A p \Longrightarrow b \in \text{reject } n V A q$  and
   $b \in \text{defer } n V A p \Longrightarrow b \in \text{defer } n V A q$ 
  using defer-n lifted-a monotone-n f-profs
  unfolding defer-lift-invariance-def
  by (metis, metis, metis)
qed
qed
moreover have  $\forall b \in B. \text{mod-contains-result } n (m \parallel_{\uparrow} n) V A q b$ 
  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:
   $\forall b \in B. \text{prof-contains-result } (m \parallel_{\uparrow} n) V A p q b$ 
  unfolding mod-contains-result-def mod-contains-result-sym-def
  prof-contains-result-def
  by simp
have  $\forall b \in A - B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) m V A p b$ 
  using alts max-agg-rej-fst-equiv-seq-contained monotone-m monotone-n f-profs
  unfolding defer-lift-invariance-def
  by metis
moreover have  $\forall b \in A. \text{prof-contains-result } m V A p q b$ 
proof (unfold prof-contains-result-def, clarify)
  fix  $b :: 'a$ 
  assume  $b\text{-in-}A$ :  $b \in A$ 
  show  $SCF\text{-result.electoral-module } m \wedge \text{profile } V A p \wedge$ 
    profile  $V A q \wedge b \in A \wedge$ 
     $(b \in \text{elect } m V A p \longrightarrow b \in \text{elect } m V A q) \wedge$ 

```



```

      (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 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 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 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 ∀ b ∈ A − B. mod-contains-result m (m ||↑ 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 ∀ b ∈ A − 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
thus ?thesis
  using prof-contains-result-of-comps-for-elems-in-B
  by blast
next
  assume a ∉ B
  hence a-in-set-diff: a ∈ A − B
  using DiffI lifted-a compatible f-profs
  unfolding Profile.lifted-def
  by (metis (no-types, lifting))
  hence reject-n: a ∈ reject n V A p
  using alts f-profs
  by blast
  hence defer-m: a ∈ 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 ∀ b ∈ B. mod-contains-result (m ||↑ n) n V A p b

```

```

using alts compatible f-profs max-agg-rej-snd-imp-seq-contained mod-contains-result-comm
unfolding disjoint-compatibility-def
by metis
have  $\forall b \in B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ b$ 
using alts max-agg-rej-snd-equiv-seq-contained monotone-m monotone-n f-profs
unfolding defer-lift-invariance-def
by metis
moreover have  $\forall b \in A. \text{prof-contains-result } n \ V \ A \ p \ q \ b$ 
proof (unfold prof-contains-result-def, clarify)
  fix  $b :: 'a$ 
  assume  $b \text{-in-} A: b \in A$ 
  show  $SCF\text{-result.electoral-module } n \wedge \text{profile } V \ A \ p \wedge$ 
     $\text{profile } V \ A \ q \wedge b \in A \wedge$ 
     $(b \in \text{elect } n \ V \ A \ p \longrightarrow b \in \text{elect } n \ V \ A \ q) \wedge$ 
     $(b \in \text{reject } n \ V \ A \ p \longrightarrow b \in \text{reject } n \ V \ A \ q) \wedge$ 
     $(b \in \text{defer } n \ V \ A \ p \longrightarrow b \in \text{defer } n \ V \ A \ q)$ 
  proof (safe)
    show  $SCF\text{-result.electoral-module } n$ 
    using monotone-n
    unfolding defer-lift-invariance-def
    by metis
  next
  show
     $\text{profile } V \ A \ p \text{ and}$ 
     $\text{profile } V \ A \ q \text{ and}$ 
     $b \in A$ 
    using f-profs b-in-A
    by (simp, simp, simp)
  next
  show
     $b \in \text{elect } n \ V \ A \ p \implies b \in \text{elect } n \ V \ A \ q \text{ and}$ 
     $b \in \text{reject } n \ V \ A \ p \implies b \in \text{reject } n \ V \ A \ q \text{ and}$ 
     $b \in \text{defer } n \ V \ A \ p \implies b \in \text{defer } n \ V \ A \ q$ 
    using alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a
    unfolding indep-of-alt-def
    by (metis, metis, metis)
  qed
qed
moreover have  $\forall b \in B. \text{mod-contains-result } n \ (m \parallel_{\uparrow} n) \ V \ A \ q \ b$ 
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:
   $\forall b \in B. \text{prof-contains-result } (m \parallel_{\uparrow} n) \ V \ A \ p \ q \ b$ 
  unfolding mod-contains-result-def mod-contains-result-sym-def
  prof-contains-result-def
  by simp
have  $\forall b \in A - B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ m \ V \ A \ p \ b$ 
using alts max-agg-rej-fst-equiv-seq-contained monotone-m monotone-n f-profs

```

```

unfolding defer-lift-invariance-def
by metis
moreover have  $\forall b \in A. \text{prof-contains-result } m \ V \ A \ p \ q \ b$ 
proof (unfold prof-contains-result-def, clarify)
  fix  $b :: 'a$ 
  assume  $b \text{-in-} A: b \in A$ 
  show  $SCF\text{-result.electoral-module } m \wedge \text{profile } V \ A \ p$ 
     $\wedge \text{profile } V \ A \ q \wedge b \in A$ 
     $\wedge (b \in \text{elect } m \ V \ A \ p \longrightarrow b \in \text{elect } m \ V \ A \ q)$ 
     $\wedge (b \in \text{reject } m \ V \ A \ p \longrightarrow b \in \text{reject } m \ V \ A \ q)$ 
     $\wedge (b \in \text{defer } m \ V \ A \ p \longrightarrow b \in \text{defer } m \ V \ A \ q)$ 
  proof (safe)
    show  $SCF\text{-result.electoral-module } m$ 
      using monotone-m
      unfolding defer-lift-invariance-def
      by simp
    next
      show
         $\text{profile } V \ A \ p$  and
         $\text{profile } V \ A \ q$  and
         $b \in A$ 
        using f-profs b-in-A
        by (simp, simp, simp)
    next
      show
         $b \in \text{elect } m \ V \ A \ p \implies b \in \text{elect } m \ V \ A \ q$  and
         $b \in \text{reject } m \ V \ A \ p \implies b \in \text{reject } m \ V \ A \ q$  and
         $b \in \text{defer } m \ V \ A \ p \implies b \in \text{defer } m \ V \ A \ q$ 
        using defer-m lifted-a monotone-m
        unfolding defer-lift-invariance-def
        by (metis, metis, metis)
      qed
    qed
  moreover have  $\forall x \in A - B. \text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ q \ x$ 
    using alts max-agg-rej-fst-imp-seq-contained monotone-m monotone-n f-profs
    unfolding defer-lift-invariance-def
    by metis
  ultimately have  $\forall x \in A - B. \text{prof-contains-result } (m \parallel_{\uparrow} n) \ V \ A \ p \ q \ x$ 
    unfolding mod-contains-result-def mod-contains-result-sym-def
      prof-contains-result-def
    by simp
  thus ?thesis
    using prof-contains-result-of-comps-for-elems-in-B
    by blast
  qed
thus  $(m \parallel_{\uparrow} n) \ V \ A \ p = (m \parallel_{\uparrow} n) \ V \ A \ q$ 
  using compatible f-profs eq-alts-in-profs-imp-eq-results max-par-comp-sound
  unfolding disjoint-compatibility-def
  by metis

```

qed

lemma *par-comp-rej-card*:

fixes

$m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **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: $B \subseteq A$

$\wedge (\forall\ a \in B. indep\ of\ alt\ m\ V\ A\ a \wedge$
 $(\forall\ q. profile\ V\ A\ q \longrightarrow a \in reject\ m\ V\ A\ q))$

$\wedge (\forall\ a \in A - B. indep\ of\ alt\ n\ V\ A\ a \wedge$
 $(\forall\ q. profile\ V\ A\ q \longrightarrow a \in reject\ n\ V\ A\ q))$

using *compatible* *prof*

unfolding *disjoint-compatibility-def*

by *metis*

have *reject-representation*:

$reject\ (m\ ||_{\uparrow}\ n)\ V\ A\ p = (reject\ m\ V\ A\ p) \cap (reject\ n\ V\ A\ p)$

using *prof* *fin-A* *compatible* *max-agg-rej-intersect*

unfolding *disjoint-compatibility-def*

by *metis*

have *SCF-result.electoral-module* $m \wedge SCF\ result.electoral\ module\ n$

using *compatible*

unfolding *disjoint-compatibility-def*

by *simp*

hence *subsets*: $(reject\ m\ V\ A\ p) \subseteq A \wedge (reject\ n\ V\ A\ p) \subseteq A$

using *prof*

by (*simp* *add*: *reject-in-alts*)

hence *finite* $(reject\ m\ V\ A\ p) \wedge finite\ (reject\ n\ V\ A\ p)$

using *rev-finite-subset* *prof* *fin-A*

by *metis*

hence *card-difference*:

$card\ (reject\ (m\ ||_{\uparrow}\ n)\ V\ A\ p)$
 $= card\ A + c - card\ ((reject\ m\ V\ A\ p) \cup (reject\ n\ V\ A\ p))$

using *card-Un-Int* *reject-representation* *reject-sum*

by *fastforce*

have $\forall\ a \in A. a \in (reject\ m\ V\ A\ p) \vee a \in (reject\ n\ V\ A\ p)$

using *alt-set* *prof* *fin-A*

by *blast*

hence $A = reject\ m\ V\ A\ p \cup reject\ n\ V\ A\ p$

```

    using subsets
    by force
  thus card (reject (m ||↑ 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 ||↑ n)
proof (unfold eliminates-def, safe)
  have SCF-result.electoral-module m
    using non-elec-m
    unfolding non-electing-def
    by simp
  moreover have SCF-result.electoral-module n
    using rejec-n-two
    unfolding rejects-def
    by simp
  ultimately show SCF-result.electoral-module (m ||↑ 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: 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  $\Rightarrow$ 
  ('a, 'v, 'a Result) Electoral-Module where
  elector m = (m  $\triangleright$  elect-module)
```

6.7.2 Auxiliary Lemmas

```
lemma elector-seqcomp-assoc:
  fixes a b :: ('a, 'v, 'a Result) Electoral-Module
  shows (a  $\triangleright$  (elector b)) = (elector (a  $\triangleright$  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 SCF-result.electoral-module m
  shows 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: SCF-result.electoral-module m and
    non-block-m: non-blocking m
  shows electing (elector m)
```

proof –
have $\forall m'$.
 $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m' \wedge$
 $(\forall A' V' p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' A' p') \longrightarrow \text{elect } m' V' A' p' \neq \{\})) \wedge$
 $(\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m'$
 $\vee (\exists A V p. (A \neq \{\} \wedge \text{finite } A \wedge \text{profile } V A p \wedge \text{elect } m' V A p = \{\})))$
unfolding *electing-def*
by *blast*
hence $\forall m'$.
 $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m' \wedge$
 $(\forall A' V' p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' A' p') \longrightarrow \text{elect } m' V' A' p' \neq \{\})) \wedge$
 $(\exists A V p. (\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m' \vee A \neq \{\}$
 $\wedge \text{finite } A \wedge \text{profile } V A p \wedge \text{elect } m' V A p = \{\})))$
by *simp*
then obtain
 $A :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'a \text{ set}$ **and**
 $V :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'v \text{ set}$ **and**
 $p :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow ('a, 'v) \text{ Profile}$ **where**
electing-mod:
 $\forall m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module.}$
 $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m' \wedge$
 $(\forall A' V' p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' A' p') \longrightarrow \text{elect } m' V' A' p' \neq \{\})) \wedge$
 $(\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m'$
 $\vee A m' \neq \{\} \wedge \text{finite } (A m') \wedge \text{profile } (V m') (A m') (p m')$
 $\wedge \text{elect } m' (V m') (A m') (p m') = \{\})$
by *metis*
moreover have *non-block*:
 $\text{non-blocking } (\text{elect-module} :: 'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow 'a \text{ Result})$
by (*simp add: electing-imp-non-blocking*)
moreover obtain
 $e :: 'a \text{ Result} \Rightarrow 'a \text{ set}$ **and**
 $r :: 'a \text{ Result} \Rightarrow 'a \text{ set}$ **and**
 $d :: 'a \text{ Result} \Rightarrow 'a \text{ set}$ **where**
 $\text{result: } \forall s. (e s, r s, d s) = s$
using *disjoint3.cases*
by (*metis (no-types)*)
moreover from this
have $\forall s. (\text{elect-r } s, r s, d s) = s$
by *simp*
moreover from this
have
 $\text{profile } (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m)) \wedge \text{finite } (A (\text{elector } m))$
 $\longrightarrow d (\text{elector } m) (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m)) = \{\}$
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 $\text{elector}(m)$ 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:  $\forall y \in A - \{w\}.$ 
    card  $\{i \in V. (w, y) \in (p\ i)\}$ 
      < card  $\{i \in V. (y, w) \in (p\ i)\}$ 

```

using *c-win fin-V*
by *simp*
have *rej-is-complement*:
 $\text{reject } m \ V \ A \ p = A - (\text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p)$
using *double-diff sup-bot.left-neutral Un-upper2 assms fin-A prof-A fin-V*
 $\text{defer-condorcet-consistency-def elec-and-def-not-rej reject-in-alts}$
by (*metis (no-types, opaque-lifting)*)
have *subset-in-win-set*: $\text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p \subseteq$
 $\{e \in A. e \in A \wedge (\forall x \in A - \{e\}.$
 $\text{card } \{i \in V. (e, x) \in p \ i\} < \text{card } \{i \in V. (x, e) \in p \ i\})\}$
proof (*safe-step*)
fix $x :: 'a$
assume *x-in-elect-or-defer*: $x \in \text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p$
hence *x-eq-w*: $x = w$
using *Diff-empty Diff-iff assms cond-winner-unique c-win fin-A fin-V insert-iff*
 $\text{snd-conv sup-bot.left-neutral fst-eqD}$
unfolding *defer-condorcet-consistency-def*
by (*metis (mono-tags, lifting)*)
have $\forall x. x \in \text{elect } m \ V \ A \ p \longrightarrow x \in A$
using *fin-A prof-A fin-V assms elect-in-alts in-mono*
unfolding *defer-condorcet-consistency-def*
by *metis*
moreover have $\forall x. x \in \text{defer } m \ V \ A \ p \longrightarrow x \in A$
using *fin-A prof-A fin-V assms defer-in-alts in-mono*
unfolding *defer-condorcet-consistency-def*
by *metis*
ultimately have $x \in A$
using *x-in-elect-or-defer*
by *auto*
thus $x \in \{e \in A. e \in A \wedge$
 $(\forall x \in A - \{e\}.$
 $\text{card } \{i \in V. (e, x) \in p \ i\}$
 $< \text{card } \{i \in V. (x, e) \in p \ i\})\}$
using *x-eq-w max-card-w*
by *auto*
qed
moreover have
 $\{e \in A. e \in A \wedge$
 $(\forall x \in A - \{e\}.$
 $\text{card } \{i \in V. (e, x) \in p \ i\} <$
 $\text{card } \{i \in V. (x, e) \in p \ i\})\}$
 $\subseteq \text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p$
proof (*safe*)
fix $x :: 'a$
assume
 $x\text{-not-in-defer: } x \notin \text{defer } m \ V \ A \ p \text{ and}$
 $x \in A \text{ and}$
 $\forall x' \in A - \{x\}.$
 $\text{card } \{i \in V. (x, x') \in p \ i\}$

```

    < card {i ∈ V. (x', x) ∈ p i}
  hence c-win-x: condorcet-winner V A p x
    using fin-A prof-A fin-V
    by simp
  have (SCF-result.electoral-module m ∧ ¬ defer-condorcet-consistency m →
    (∃ A V rs a. condorcet-winner V A rs a ∧
      m V A rs ≠ ({}, A - defer m V A rs,
        {a ∈ A. condorcet-winner V A rs a})))
    ∧ (defer-condorcet-consistency m →
      (∀ A V rs a. finite A → finite V → condorcet-winner V A rs a →
        m V A rs =
          ({} , A - defer m V A rs, {a ∈ A. condorcet-winner V A rs a})))
    unfolding defer-condorcet-consistency-def
    by blast
  hence
    m V A p = ({} , A - defer m V A p, {a ∈ A. condorcet-winner V A p a})
    using c-win-x assms fin-A fin-V
    by blast
  thus x ∈ elect m V A p
    using assms x-not-in-defer fin-A fin-V cond-winner-unique
      defer-condorcet-consistency-def insertCI snd-conv c-win-x
    by (metis (no-types, lifting))
qed
ultimately have
  elect m V A p ∪ defer m V A p =
    {e ∈ A. e ∈ A ∧
      (∀ x ∈ A - {e}.
        card {i ∈ V. (e, x) ∈ p i} <
          card {i ∈ V. (x, e) ∈ p i})}
    by blast
  thus elector m V A p =
    ({e ∈ A. condorcet-winner V A p e}, A - elect (elector m) V A p, {})
    using fin-A prof-A fin-V rej-is-complement
    by simp
qed
end

```

6.8 Defer-One Loop Composition

```

theory Defer-One-Loop-Composition
  imports Basic-Modules/Component-Types/Defer-Equal-Condition
    Loop-Composition
    Elect-Composition
begin

```

This is a family of loop compositions. It uses the same module in sequence until either no new decisions are made or only one alternative is remaining in the defer-set. The second family herein uses the above family and subsequently elects the remaining alternative.

```
fun iter :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$ 
  ('a, 'v, 'a Result) Electoral-Module where
  iter m =
    (let t = defer-equal-condition 1 in
     (m  $\odot_t$ ))
```

```
abbreviation defer-one-loop :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$ 
  ('a, 'v, 'a Result) Electoral-Module ( $-\odot_{\exists!d}$  50) where
  m  $\odot_{\exists!d} \equiv$  iter m
```

```
fun iter-elect :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$ 
  ('a, 'v, 'a Result) Electoral-Module where
  iter-elect m = elector (m  $\odot_{\exists!d}$ )
```

6.8.1 Soundness

```
theorem defer-one-loop-comp-sound:
fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  t :: 'a Termination-Condition
assumes SCF-result.electoral-module m
shows SCF-result.electoral-module (m  $\odot_{\exists!d}$ )
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 ∈ A. ∀ x ∈ A. win-count V p x ≤ win-count V p a},
           {a ∈ A. ∃ x ∈ 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↓) V A p
  by fastforce

lemma plurality'-revision-equiv:
```

```

fixes
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
shows  $\text{plurality}' V A p = (\text{plurality-rule}'\downarrow) V A p$ 
proof ( $\text{unfold } \text{plurality}'.\text{simps}$   $\text{revision-composition}.simps$ ,
   $\text{intro } \text{prod-eqI}$   $\text{equalityI}$   $\text{subsetI}$   $\text{prod.sel}$ )
fix  $b :: 'a$ 
assume
   $\text{assm}: b \in \text{fst}$ 
    ( $\text{if } \text{finite } A$ 
       $\text{then } (\{\}, \{a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x\},$ 
         $\{a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a\})$ 
       $\text{else } (\{\}, \{\}, A)$ )
have  $\text{finite } A \longrightarrow b \in \{\}$ 
  using  $\text{assm}$ 
  by  $\text{force}$ 
moreover have  $\text{infinite } A \longrightarrow b \in \{\}$ 
  using  $\text{assm}$ 
  by  $\text{fastforce}$ 
ultimately show
   $b \in \text{fst } (\{\}, A - \text{elect } \text{plurality-rule}' V A p, \text{elect } \text{plurality-rule}' V A p)$ 
  by  $\text{safe}$ 
next
fix  $b :: 'a$ 
assume  $b \in \text{fst } (\{\}, A - \text{elect } \text{plurality-rule}' V A p, \text{elect } \text{plurality-rule}' V A p)$ 
thus  $b \in \text{fst}$ 
  ( $\text{if } \text{finite } A$ 
     $\text{then } (\{\}, \{a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x\},$ 
       $\{a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a\})$ 
     $\text{else } (\{\}, \{\}, A)$ )
  by  $\text{force}$ 
next
fix  $b :: 'a$ 
assume
   $\text{assm}: b \in \text{fst } (\text{snd}$ 
    ( $\text{if } \text{finite } A$ 
       $\text{then } (\{\}, \{a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x\},$ 
         $\{a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a\})$ 
       $\text{else } (\{\}, \{\}, A)))$ 
  have  $\text{finite } A \longrightarrow$ 
     $b \in A - \{a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a\}$ 
  using  $\text{assm}$ 
  by  $\text{fastforce}$ 
moreover have  $\text{infinite } A \longrightarrow b \in \{\}$ 
  using  $\text{assm}$ 
  by  $\text{fastforce}$ 
ultimately show
   $b \in \text{fst } (\text{snd } (\{\}, A - \text{elect } \text{plurality-rule}' V A p, \text{elect } \text{plurality-rule}' V A p))$ 

```

```

    by force
next
  fix  $b :: 'a$ 
  assume
     $assm: b \in$ 
       $fst (snd (\{\}, A - elect\ plurality-rule'\ V\ A\ p, elect\ plurality-rule'\ V\ A\ p))$ 
  have  $finite\ A \longrightarrow$ 
     $b \in A - \{a \in A. \forall\ x \in A. win-count\ V\ p\ x \leq win-count\ V\ p\ a\}$ 
    using  $assm$ 
    by fastforce
  moreover have  $infinite\ A \longrightarrow b \in \{\}$ 
    using  $assm$ 
    by fastforce
  ultimately show
     $b \in fst (snd$ 
      (if  $finite\ A$ 
        then  $(\{\}, \{a \in A. \exists\ x \in A. win-count\ V\ p\ a < win-count\ V\ p\ x\},$ 
           $\{a \in A. \forall\ x \in A. win-count\ V\ p\ x \leq win-count\ V\ p\ a\})$ 
        else  $(\{\}, \{\}, A))$ 
      )
    using  $linorder-not-less$ 
    by force
next
  fix  $b :: 'a$ 
  assume
     $assm: b \in snd (snd$ 
      (if  $finite\ A$ 
        then  $(\{\}, \{a \in A. \exists\ x \in A. win-count\ V\ p\ a < win-count\ V\ p\ x\},$ 
           $\{a \in A. \forall\ x \in A. win-count\ V\ p\ x \leq win-count\ V\ p\ a\})$ 
        else  $(\{\}, \{\}, A))$ 
      )
  have  $finite\ A \longrightarrow$ 
     $b \in A - \{a \in A. \exists\ x \in A. win-count\ V\ p\ a < win-count\ V\ p\ x\}$ 
    using  $assm$ 
    by fastforce
  moreover have  $infinite\ A \longrightarrow b \in A$ 
    using  $assm$ 
    by fastforce
  ultimately have  $b \in elect\ plurality-rule'\ V\ A\ p$ 
    by force
  thus  $b \in snd (snd$ 
    ( $\{\}, A - elect\ plurality-rule'\ V\ A\ p, elect\ plurality-rule'\ V\ A\ p)$ 
    )
    by simp
next
  fix  $b :: 'a$ 
  assume  $assm:$ 
     $b \in snd (snd (\{\}, A - elect\ plurality-rule'\ V\ A\ p, elect\ plurality-rule'\ V\ A\ p))$ 
  have  $finite\ A \longrightarrow$ 
     $b \in A - \{a \in A. \exists\ x \in A. win-count\ V\ p\ a < win-count\ V\ p\ x\}$ 
    using  $assm$ 
    by fastforce

```

```

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

lemma plurality-rule-equiv:
  fixes
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  shows  $\text{plurality-rule } V A p = \text{plurality-rule}' V A p$ 
proof (unfold plurality-rule.simps)
  have plurality-rule'-rev-equiv-plurality:
     $\text{plurality } V A p = (\text{plurality-rule}' \downarrow) V A p$ 
  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  $\text{elector plurality } V A p = \text{plurality-rule}' V A p$ 
  using prod-eqI
  by (metis (mono-tags, lifting))
qed

```

7.1.2 Soundness

```

theorem plurality-rule-sound[simp]: SCF-result.electoral-module plurality-rule
  unfolding plurality-rule.simps
  using elector-sound plurality-sound
  by metis

theorem plurality-rule'-sound[simp]: SCF-result.electoral-module plurality-rule'
proof (unfold SCF-result.electoral-module.simps, safe)
  fix
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  have disjoint3 (

```



```

    {a ∈ A. ∀ a' ∈ A. win-count V p a' ≤ win-count V p a},
    {a ∈ A. ∃ a' ∈ A. win-count V p a < win-count V p a'},
    {}
  )
  by auto
  moreover have
    {a ∈ A. ∀ x ∈ A. win-count V p x ≤ win-count V p a} ∪
    {a ∈ A. ∃ x ∈ A. win-count V p a < win-count V p x} = A
  using not-le-imp-less
  by auto
  ultimately show well-formed-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-electors 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 ∀ v ∈ 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 ≠ {} and
    finite A
  shows elect plurality-rule V A p ≠ {}
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 ∧ a ∈ A

```

```

    using Max-in assms empty-is-image finite-imageI imageE
    by (metis (no-types, lifting))
  hence  $\forall a' \in A. \text{win-count } V \ p \ a' \leq \text{win-count } V \ p \ a$ 
    using assms max
    by simp
  moreover have  $a \in A$ 
    using max-a
    by simp
  ultimately have  $a \in \{a' \in A. \forall c \in A. \text{win-count } V \ p \ c \leq \text{win-count } V \ p \ a'\}$ 
    by blast
  hence  $a \in \text{elect } \text{plurality-rule}' \ V \ A \ p$ 
    by simp
  moreover have  $\text{elect } \text{plurality-rule}' \ V \ A \ p = \text{defer } \text{plurality} \ V \ A \ p$ 
    using plurality-revision-equiv plurality-rule-equiv snd-conv
    unfolding revision-composition.simps
    by (metis (no-types, opaque-lifting))
  ultimately have  $a \in \text{defer } \text{plurality} \ V \ A \ p$ 
    by blast
  hence  $a \in \text{elect } \text{plurality-rule} \ V \ A \ p$ 
    by simp
  thus False
    using plurality-elect-none all-not-in-conv
    by metis
qed

```

```

lemma plurality-rule'-elect-non-empty:
  fixes
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  assumes
     $A \neq \{\}$  and
    profile  $V \ A \ p$  and
    finite  $A$ 
  shows  $\text{elect } \text{plurality-rule}' \ V \ A \ p \neq \{\}$ 
  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 SCF-result.electoral-module plurality-rule
    using plurality-rule-sound
    by simp
next
  fix
     $A :: 'b \text{ set}$  and
     $V :: 'a \text{ set}$  and
     $p :: ('b, 'a) \text{ Profile}$  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 ∈ A
have ∀ B W q. B ≠ {} ∧ finite B ∧ profile W B q
  → elect plurality-rule W B q ≠ {}
  using plurality-rule-elect-non-empty
  by (metis (no-types))
hence empty-A: A = {}
  using fin-A prof-p elect-none
  by (metis (no-types))
thus a ∈ {}
  using a-in-A
  by simp
qed

theorem plurality-rule'-electing[simp]: electing plurality-rule'
proof (unfold electing-def, safe)
  show 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 ∈ A
have A-nonempty: A ≠ {}
  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 ≠ {}
  using fin-A prof-p A-nonempty plurality-rule'-elect-non-empty plurality-rule-equiv
  by metis
ultimately show a ∈ {}
  by force
qed

```

7.1.4 Properties

lemma *plurality-rule-inv-mono-eq*:

fixes

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p \ q :: ('a, 'v) \text{ Profile}$ **and**

$a :: 'a$

assumes

elect-a: $a \in \text{elect } \text{plurality-rule } V \ A \ p$ **and**

lift-a: *lifted* $V \ A \ p \ q \ a$

shows $\text{elect } \text{plurality-rule } V \ A \ q = \text{elect } \text{plurality-rule } V \ A \ p$
 $\vee \text{elect } \text{plurality-rule } V \ A \ q = \{a\}$

proof –

have $a \in \text{elect } (\text{elector plurality}) \ V \ A \ p$

using *elect-a*

by *simp*

moreover have eq-p : $\text{elect } (\text{elector plurality}) \ V \ A \ p = \text{defer plurality } V \ A \ p$

by *simp*

ultimately have $a \in \text{defer plurality } V \ A \ p$

by *blast*

hence $\text{defer plurality } V \ A \ q = \text{defer plurality } V \ A \ p$

$\vee \text{defer plurality } V \ A \ q = \{a\}$

using *lift-a plurality-def-inv-mono-alts*

by *metis*

moreover have $\text{elect } (\text{elector plurality}) \ V \ A \ q = \text{defer plurality } V \ A \ q$

by *simp*

ultimately show

$\text{elect } \text{plurality-rule } V \ A \ q = \text{elect } \text{plurality-rule } V \ A \ p$

$\vee \text{elect } \text{plurality-rule } V \ A \ q = \{a\}$

using *eq-p*

by *simp*

qed

lemma *plurality-rule'-inv-mono-eq*:

fixes

$A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**

$p \ q :: ('a, 'v) \text{ Profile}$ **and**

$a :: 'a$

assumes

$a \in \text{elect } \text{plurality-rule}' \ V \ A \ p$ **and**

lifted $V \ A \ p \ q \ a$

shows $\text{elect } \text{plurality-rule}' \ V \ A \ q = \text{elect } \text{plurality-rule}' \ V \ A \ p$
 $\vee \text{elect } \text{plurality-rule}' \ V \ A \ q = \{a\}$

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 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 ∈ elect plurality-rule V A p ∧ Profile.lifted V A p q a
  thus elect plurality-rule V A q = elect plurality-rule V A p
    ∨ 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

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_R* :: ('a, 'v :: wellorder, 'a Result) Electoral-Module **where**
borda-rule_R V A p = swap-R unanimity V A p

7.2.2 Soundness

theorem *borda-rule-sound: SCF-result.electoral-module borda-rule*
unfolding *borda-rule.simps*
using *elector-sound borda-sound*
by *metis*

theorem *borda-rule_R-sound: SCF-result.electoral-module borda-rule_R*
unfolding *borda-rule_R.simps swap-R.simps*
using *SCF-result.R-sound*
by *metis*

7.2.3 Anonymity

theorem *borda-rule_R-anonymous: SCF-result.anonymity borda-rule_R*
proof (*unfold borda-rule_R.simps swap-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 *SCF-result.anonymity (SCF-result.distance-R ?swap-dist unanimity)*
using *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*) $\odot_{\exists!d}$) 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*: *SCF-result.electoral-module pairwise-majority-rule*
unfolding *pairwise-majority-rule.simps*
using *condorcet-sound elector-sound*
by *metis*

theorem *condorcet'-sound*: *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*: *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-elect*
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: 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: 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  $\triangleright$  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-elect*
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*) $\circ_{\exists!d}$) A p

7.7.2 Soundness

theorem *nanson-baldwin-rule-sound: 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)  $\circ_{\exists!d}$ ) V A p

```

7.8.2 Soundness

```

theorem classic-nanson-rule-sound: 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)  $\circ_{\exists!d}$ ) V A p

```

7.9.2 Soundness

theorem *schwartz-rule-sound: 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* \Rightarrow (*'a, 'v, 'a Result*) *Electoral-Module* **where**
smc *x* *V A p* =
 ((*elector* (((*pass-module* 2 *x*) \triangleright ((*plurality-rule* \downarrow) \triangleright (*pass-module* 1 *x*)))) \parallel_{\uparrow}
 (*drop-module* 2 *x*)) $\circ_{\exists !d}$) *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 *SCF-result.electoral-module (smc x)*
proof (*unfold SCF-result.electoral-module.simps well-formed-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 $?pass2 = \text{pass-module } 2\ x$

let $?tie-breaker = (\text{pass-module } 1\ x)$

let $?plurality-defer = (\text{plurality-rule}\downarrow) \triangleright ?tie-breaker$

let $?compare-two = ?pass2 \triangleright ?plurality-defer$

let $?drop2 = \text{drop-module } 2\ x$

let $?eliminator = ?compare-two \parallel_{\uparrow} ?drop2$

let $?loop =$

let $t = \text{defer-equal-condition } 1 \text{ in } (?eliminator \circ_t)$

have *00011: non-electing* (*plurality-rule* \downarrow)

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* \downarrow)

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  $?pass2 = pass\text{-}module\ 2\ x$ 
  let  $?tie\text{-}breaker = pass\text{-}module\ 1\ x$ 
  let  $?plurality\text{-}defer = (plurality\text{-}rule\downarrow) \triangleright ?tie\text{-}breaker$ 
  let  $?compare\text{-}two = ?pass2 \triangleright ?plurality\text{-}defer$ 
  let  $?drop2 = drop\text{-}module\ 2\ x$ 
  let  $?eliminator = ?compare\text{-}two \parallel_{\uparrow} ?drop2$ 
  let  $?loop =$ 
    let  $t = defer\text{-}equal\text{-}condition\ 1\ in\ (?eliminator \circ_t)$ 

  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  $?tie\text{-}breaker$ 
    using assms
    by simp
  have 00013: defers  $1\ ?tie\text{-}breaker$ 
    using assms pass-one-mod-def-one
    by simp
  have 00014: defer-monotonicity  $?tie\text{-}breaker$ 
    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  $?plurality\text{-}defer$ 
    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  $?plurality\text{-}defer$ 
    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: SCF-result.electoral-module kemeny-rule
  unfolding kemeny-rule.simps swap- $\mathcal{R}$ .simps
  using SCF-result. $\mathcal{R}$ -sound
  by metis

```

7.11.3 Anonymity

```

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

```

7.11.4 Neutrality

```

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: SCF-properties.neutrality
  well-formed-elections kemeny-rule
  using strong-unanimity-neutral' swap-dist-neutral strong-unanimity-closed-under-neutrality
    SCF-properties.neutr-dist-and-cons-imp-neutr-dr
  unfolding kemeny-rule.simps swap- $\mathcal{R}$ .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](https://doi.org/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](https://doi.org/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](https://arxiv.org/abs/1610.01902).