

# Verified Construction of Fair Voting Rules

Michael Kirsten

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

April 5, 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

<b>1</b>	<b>Social-Choice Types</b>	<b>9</b>
1.1	Preference Relation . . . . .	9
1.1.1	Definition . . . . .	9
1.1.2	Ranking . . . . .	10
1.1.3	Limited Preference . . . . .	10
1.1.4	Auxiliary Lemmas . . . . .	16
1.1.5	Lifting Property . . . . .	26
1.2	Norm . . . . .	36
1.2.1	Definition . . . . .	36
1.2.2	Auxiliary Lemmas . . . . .	36
1.2.3	Common Norms . . . . .	38
1.2.4	Properties . . . . .	38
1.2.5	Theorems . . . . .	38
1.3	Electoral Result . . . . .	39
1.3.1	Auxiliary Functions . . . . .	39
1.3.2	Definition . . . . .	40
1.4	Preference Profile . . . . .	40
1.4.1	Definition . . . . .	41
1.4.2	Vote Count . . . . .	42
1.4.3	Voter Permutations . . . . .	46
1.4.4	List Representation for Ordered Voters . . . . .	49
1.4.5	Preference Counts and Comparisons . . . . .	56
1.4.6	Condorcet Winner . . . . .	60
1.4.7	Limited Profile . . . . .	61
1.4.8	Lifting Property . . . . .	62
1.5	Social Choice Result . . . . .	65
1.5.1	Social Choice Result . . . . .	65
1.5.2	Auxiliary Lemmas . . . . .	66
1.6	Social Welfare Result . . . . .	69
1.6.1	Social Welfare Result . . . . .	69
1.7	Specific Electoral Result Types . . . . .	69
1.8	Function Symmetry Properties . . . . .	71
1.8.1	Functions . . . . .	71

1.8.2	Invariance and Equivariance . . . . .	72
1.8.3	Auxiliary Lemmas . . . . .	72
1.8.4	Rewrite Rules . . . . .	75
1.8.5	Group Actions . . . . .	79
1.8.6	Invariance and Equivariance . . . . .	82
1.9	Symmetry Properties of Voting Rules . . . . .	90
1.9.1	Definitions . . . . .	90
1.9.2	Auxiliary Lemmas . . . . .	92
1.9.3	Anonymity Lemmas . . . . .	102
1.9.4	Neutrality Lemmas . . . . .	110
1.9.5	Homogeneity Lemmas . . . . .	123
1.9.6	Reversal Symmetry Lemmas . . . . .	123
1.10	Result-Dependent Voting Rule Properties . . . . .	128
1.10.1	Properties Dependent on the Result Type . . . . .	128
1.10.2	Interpretations . . . . .	129
<b>2</b>	<b>Refined Types</b>	<b>130</b>
2.1	Preference List . . . . .	130
2.1.1	Well-Formedness . . . . .	130
2.1.2	Auxiliary Lemmas About Lists . . . . .	130
2.1.3	Ranking . . . . .	135
2.1.4	Definition . . . . .	135
2.1.5	Limited Preference . . . . .	144
2.1.6	Auxiliary Definitions . . . . .	151
2.1.7	Auxiliary Lemmas . . . . .	151
2.1.8	First Occurrence Indices . . . . .	154
2.2	Preference (List) Profile . . . . .	156
2.2.1	Definition . . . . .	156
2.3	Ordered Relation Type . . . . .	157
2.4	Alternative Election Type . . . . .	160
<b>3</b>	<b>Quotient Rules</b>	<b>161</b>
3.1	Quotients of Equivalence Relations . . . . .	161
3.1.1	Definitions . . . . .	161
3.1.2	Well-Definedness . . . . .	161
3.1.3	Equivalence Relations . . . . .	164
3.2	Quotients of Equivalence Relations on Election Sets . . . . .	165
3.2.1	Auxiliary Lemmas . . . . .	165
3.2.2	Anonymity Quotient: Grid . . . . .	168
3.2.3	Homogeneity Quotient: Simplex . . . . .	174

<b>4</b>	<b>Component Types</b>	<b>198</b>
4.1	Distance	198
4.1.1	Definition	198
4.1.2	Conditions	199
4.1.3	Standard Distance Property	199
4.1.4	Auxiliary Lemmas	199
4.1.5	Swap Distance	200
4.1.6	Spearman Distance	201
4.1.7	Properties	202
4.2	Votewise Distance	208
4.2.1	Definition	208
4.2.2	Inference Rules	208
4.3	Consensus	216
4.3.1	Definition	216
4.3.2	Consensus Conditions	216
4.3.3	Properties	217
4.3.4	Auxiliary Lemmas	217
4.3.5	Theorems	220
4.4	Electoral Module	224
4.4.1	Definition	224
4.4.2	Auxiliary Definitions	225
4.4.3	Properties	225
4.4.4	Reversal Symmetry of Social Welfare Rules	227
4.4.5	Social Choice Modules	227
4.4.6	Equivalence Definitions	229
4.4.7	Auxiliary Lemmas	230
4.4.8	Non-Blocking	241
4.4.9	Electing	241
4.4.10	Properties	242
4.4.11	Inference Rules	247
4.4.12	Social Choice Properties	250
4.5	Electoral Module on Election Quotients	252
4.6	Evaluation Function	253
4.6.1	Definition	253
4.6.2	Property	253
4.6.3	Theorems	253
4.7	Elimination Module	255
4.7.1	General Definitions	255
4.7.2	Social Choice Definitions	256
4.7.3	Common Social Choice Eliminators	256
4.7.4	Soundness	257
4.7.5	Only participating voters impact the result	258
4.7.6	Non-Blocking	261
4.7.7	Non-Electing	262

4.7.8	Inference Rules . . . . .	263
4.8	Aggregator . . . . .	266
4.8.1	Definition . . . . .	266
4.8.2	Properties . . . . .	266
4.9	Maximum Aggregator . . . . .	267
4.9.1	Definition . . . . .	267
4.9.2	Auxiliary Lemma . . . . .	267
4.9.3	Soundness . . . . .	268
4.9.4	Properties . . . . .	268
4.10	Termination Condition . . . . .	270
4.10.1	Definition . . . . .	270
4.11	Defer Equal Condition . . . . .	270
4.11.1	Definition . . . . .	271
<b>5</b>	<b>Basic Modules</b>	<b>272</b>
5.1	Defer Module . . . . .	272
5.1.1	Definition . . . . .	272
5.1.2	Soundness . . . . .	272
5.1.3	Properties . . . . .	272
5.2	Elect First Module . . . . .	273
5.2.1	Definition . . . . .	273
5.2.2	Soundness . . . . .	273
5.3	Consensus Class . . . . .	274
5.3.1	Definition . . . . .	274
5.3.2	Consensus Choice . . . . .	274
5.3.3	Auxiliary Lemmas . . . . .	275
5.3.4	Consensus Rules . . . . .	278
5.3.5	Properties . . . . .	278
5.3.6	Inference Rules . . . . .	279
5.3.7	Theorems . . . . .	282
5.4	Distance Rationalization . . . . .	289
5.4.1	Definitions . . . . .	289
5.4.2	Standard Definitions . . . . .	289
5.4.3	Auxiliary Lemmas . . . . .	290
5.4.4	Soundness . . . . .	298
5.4.5	Inference Rules . . . . .	299
5.5	Votewise Distance Rationalization . . . . .	308
5.5.1	Common Rationalizations . . . . .	308
5.5.2	Theorems . . . . .	308
5.5.3	Equivalence Lemmas . . . . .	310
5.6	Symmetry in Distance-Rationalizable Rules . . . . .	311
5.6.1	Minimizer function . . . . .	311
5.6.2	Distance Rationalization as Minimizer . . . . .	317
5.6.3	Symmetry Property Inference Rules . . . . .	325

5.6.4	Further Properties . . . . .	328
5.7	Distance Rationalization on Election Quotients . . . . .	329
5.7.1	Quotient Distances . . . . .	329
5.7.2	Quotient Consensus and Results . . . . .	339
5.7.3	Quotient Distance Rationalization . . . . .	344
5.8	Result and Property Locale Code Generation . . . . .	350
5.9	Drop Module . . . . .	352
5.9.1	Definition . . . . .	353
5.9.2	Soundness . . . . .	353
5.9.3	Non-Electing . . . . .	354
5.9.4	Properties . . . . .	354
5.10	Pass Module . . . . .	354
5.10.1	Definition . . . . .	355
5.10.2	Soundness . . . . .	355
5.10.3	Non-Blocking . . . . .	356
5.10.4	Non-Electing . . . . .	356
5.10.5	Properties . . . . .	357
5.11	Elect Module . . . . .	363
5.11.1	Definition . . . . .	363
5.11.2	Soundness . . . . .	363
5.11.3	Electing . . . . .	363
5.12	Plurality Module . . . . .	363
5.12.1	Definition . . . . .	364
5.12.2	Soundness . . . . .	368
5.12.3	Non-Blocking . . . . .	369
5.12.4	Non-Electing . . . . .	370
5.12.5	Property . . . . .	370
5.13	Borda Module . . . . .	375
5.13.1	Definition . . . . .	375
5.13.2	Soundness . . . . .	375
5.13.3	Non-Blocking . . . . .	375
5.13.4	Non-Electing . . . . .	375
5.14	Condorcet Module . . . . .	376
5.14.1	Definition . . . . .	376
5.14.2	Soundness . . . . .	376
5.14.3	Property . . . . .	376
5.15	Copeland Module . . . . .	378
5.15.1	Definition . . . . .	378
5.15.2	Soundness . . . . .	378
5.15.3	Only participating voters impact the result . . . . .	378
5.15.4	Lemmas . . . . .	379
5.15.5	Property . . . . .	382
5.16	Minimax Module . . . . .	383
5.16.1	Definition . . . . .	384

5.16.2	Soundness . . . . .	384
5.16.3	Lemma . . . . .	384
5.16.4	Property . . . . .	385
<b>6</b>	<b>Compositional Structures</b>	<b>388</b>
6.1	Drop And Pass Compatibility . . . . .	388
6.1.1	Properties . . . . .	388
6.2	Revision Composition . . . . .	393
6.2.1	Definition . . . . .	393
6.2.2	Soundness . . . . .	393
6.2.3	Composition Rules . . . . .	394
6.3	Sequential Composition . . . . .	397
6.3.1	Definition . . . . .	398
6.3.2	Soundness . . . . .	403
6.3.3	Lemmas . . . . .	404
6.3.4	Composition Rules . . . . .	408
6.4	Parallel Composition . . . . .	433
6.4.1	Definition . . . . .	433
6.4.2	Soundness . . . . .	433
6.4.3	Composition Rule . . . . .	434
6.5	Loop Composition . . . . .	436
6.5.1	Definition . . . . .	437
6.5.2	Soundness . . . . .	442
6.5.3	Lemmas . . . . .	443
6.5.4	Composition Rules . . . . .	455
6.6	Maximum Parallel Composition . . . . .	457
6.6.1	Definition . . . . .	457
6.6.2	Soundness . . . . .	457
6.6.3	Lemmas . . . . .	458
6.6.4	Composition Rules . . . . .	469
6.7	Elect Composition . . . . .	477
6.7.1	Definition . . . . .	477
6.7.2	Auxiliary Lemmas . . . . .	478
6.7.3	Soundness . . . . .	478
6.7.4	Electing . . . . .	478
6.7.5	Composition Rule . . . . .	480
6.8	Defer One Loop Composition . . . . .	482
6.8.1	Definition . . . . .	482
<b>7</b>	<b>Voting Rules</b>	<b>484</b>
7.1	Plurality Rule . . . . .	484
7.1.1	Definition . . . . .	484
7.1.2	Soundness . . . . .	486
7.1.3	Electing . . . . .	486

7.1.4	Property . . . . .	488
7.2	Borda Rule . . . . .	489
7.2.1	Definition . . . . .	489
7.2.2	Soundness . . . . .	489
7.2.3	Anonymity Property . . . . .	490
7.3	Pairwise Majority Rule . . . . .	490
7.3.1	Definition . . . . .	490
7.3.2	Soundness . . . . .	491
7.3.3	Condorcet Consistency Property . . . . .	491
7.4	Copeland Rule . . . . .	491
7.4.1	Definition . . . . .	491
7.4.2	Soundness . . . . .	491
7.4.3	Condorcet Consistency Property . . . . .	492
7.5	Minimax Rule . . . . .	492
7.5.1	Definition . . . . .	492
7.5.2	Soundness . . . . .	492
7.5.3	Condorcet Consistency Property . . . . .	492
7.6	Black's Rule . . . . .	493
7.6.1	Definition . . . . .	493
7.6.2	Soundness . . . . .	493
7.6.3	Condorcet Consistency Property . . . . .	493
7.7	Nanson-Baldwin Rule . . . . .	494
7.7.1	Definition . . . . .	494
7.7.2	Soundness . . . . .	494
7.8	Classic Nanson Rule . . . . .	494
7.8.1	Definition . . . . .	495
7.8.2	Soundness . . . . .	495
7.9	Schwartz Rule . . . . .	495
7.9.1	Definition . . . . .	495
7.9.2	Soundness . . . . .	495
7.10	Sequential Majority Comparison . . . . .	496
7.10.1	Definition . . . . .	496
7.10.2	Soundness . . . . .	496
7.10.3	Electing . . . . .	497
7.10.4	(Weak) Monotonicity Property . . . . .	498
7.11	Kemeny Rule . . . . .	501
7.11.1	Definition . . . . .	501
7.11.2	Soundness . . . . .	501
7.11.3	Anonymity Property . . . . .	501
7.11.4	Neutrality Property . . . . .	501



# Chapter 1

## Social-Choice Types

### 1.1 Preference Relation

```
theory Preference-Relation
  imports Main
begin
```

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

#### 1.1.1 Definition

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

```
type-synonym 'a Preference-Relation = 'a rel

type-synonym 'a Vote = 'a set × 'a Preference-Relation

fun is-less-preferred-than :: 'a ⇒ 'a Preference-Relation ⇒ 'a ⇒ bool
  (- ≤r - [50, 1000, 51] 50) where
    a ≤r b = ((a, b) ∈ r)

fun alts-ℳ :: 'a Vote ⇒ 'a set where
  alts-ℳ V = fst V

fun pref-ℳ :: 'a Vote ⇒ 'a Preference-Relation where
  pref-ℳ V = snd V

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

```

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

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

```

### 1.1.2 Ranking

```

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

```

```

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

```

### 1.1.3 Limited Preference

```

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

```

```

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

```

```

assumes
   $a \preceq_r b$  and
   $\text{limited } A \ r$ 
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
proof
from assms
show  $r \subseteq A \times A$ 
  unfolding connex-def limited-def
  by simp
next
fix a :: 'a
assume  $a \in A$ 
with assms
have  $a \preceq_r a$ 
  unfolding connex-def
  by metis
thus  $(a, a) \in r$ 
  by simp
qed

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

```

```

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

```

```

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

```

```

preorder-on-def refl-on-def total-on-def, safe)

fix
  a :: 'a and
  b :: 'a
assume (a, b) ∈ r
thus a ∈ A
  using connex-r refl-on-domain connex-imp-refl
  by metis
next
fix
  a :: 'a and
  b :: 'a
assume (a, b) ∈ r
thus b ∈ A
  using connex-r refl-on-domain connex-imp-refl
  by metis
next
fix a :: 'a
assume a ∈ A
thus (a, a) ∈ r
  using connex-r connex-imp-refl refl-onD
  by metis
next
from trans-r
show trans r
  by simp
next
from antisym-r
show antisym r
  by simp
next
fix
  a :: 'a and
  b :: 'a
assume
  a ∈ A and
  b ∈ A and
  (b, a) ∉ r
moreover from this
have a ≤r b ∨ b ≤r a
  using connex-r
  unfolding connex-def
  by metis
hence (a, b) ∈ r ∨ (b, a) ∈ r
  by simp
ultimately show (a, b) ∈ r
  by metis
qed

```

```

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

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

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

```

```

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

lemma limit-presv-lin-ord:
  fixes
     $A :: 'a \text{ set}$  and
     $B :: 'a \text{ set}$  and
     $r :: 'a \text{ Preference-Relation}$ 
  assumes
     $\text{linear-order-on } B \ r$  and
     $A \subseteq B$ 
  shows  $\text{linear-order-on } A \ (\text{limit } A \ r)$ 
  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
     $a :: 'a$  and
     $b :: 'a$ 
  assumes
     $a \preceq_r b$  and
     $a \in A$  and
     $b \in A$ 
  shows let  $s = \text{limit } A \ r$  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
     $a :: 'a$  and
     $b :: 'a$ 
  assumes  $(a, b) \in \text{limit } A \ r$ 
  shows  $a \preceq_r b$ 
  using mem-Collect-eq assms
  by simp

```

```

lemma limit-trans:
  fixes
     $A :: 'a \text{ set}$  and
     $B :: 'a \text{ set}$  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

```

#### 1.1.4 Auxiliary Lemmas

```

lemma above-trans:
  fixes
     $r :: 'a \text{ Preference-Relation}$  and
     $a :: 'a$  and
     $b :: 'a$ 
  assumes
    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**

*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**

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

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

$a :: 'a$

**assumes**

*linear-order-on*  $A$   $r$  **and**

*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**

*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**

$a :: 'a$  **and**

$b :: 'a$

**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
     $a :: 'a$  and
     $b :: 'a$ 
  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
     $A :: 'a \text{ set}$  and
     $B :: 'a \text{ set}$  and
     $r :: 'a \text{ Preference-Relation}$  and
     $a :: 'a$  and
     $b :: 'a$ 
  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  $\text{linear-order-on } A \ r \wedge \text{finite } A \wedge A \neq \{\} \wedge n + 1 = \text{card } A \longrightarrow$ 
     $(\exists \ a. a \in A \wedge \text{above } r \ a = \{a\})$ 
  proof (induction n arbitrary: A r)
    case 0

```

```

show ?case
proof (clarify)
  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  $a$  where  $A' = \{a\}$ 
    using card-1-singletonE add.left-neutral
    by metis
  hence  $a \in A' \wedge \text{above } r' a = \{a\}$ 
    using above-def lin-ord-r connex-imp-refl above-refl lin-ord-imp-connex
    refl-on-domain
    by fastforce
  thus  $\exists a'. a' \in A' \wedge \text{above } r' a' = \{a'\}$ 
    by metis
qed
next
case (Suc  $n$ )
show ?case
proof (clarify)
  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  $B$  where
    subset-B-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  $a$  where
     $a: A' - B = \{a\}$ 
  using Suc-eq-plus1 add-diff-cancel-left' fin-A len-A-n-plus-one card-1-singletonE
    card-Diff-subset finite-subset
    by metis
  have  $\exists a' \in B. \text{above } (\text{limit } B r') a' = \{a'\}$ 
  using subset-B-card Suc.IH add-diff-cancel-left' lin-ord-r card-eq-0-iff diff-le-self
    leD lessI limit-presv-lin-ord
    unfolding One-nat-def
    by metis
  then obtain  $b$  where
    alt-b: above (limit  $B r'$ )  $b = \{b\}$ 
    by blast
  hence  $b\text{-above: } \{a'. (b, a') \in \text{limit } B r'\} = \{b\}$ 

```

**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**  $\exists a'. a' \in A' \wedge \text{above } r' a' = \{a'\}$   
**proof** (*cases*)  
**assume**  $a\text{-pref-}r\text{-}b$ :  $a \preceq_{r'} b$   
**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** *connex-refl*:  $\forall A'' r''. \text{connex } (A''::'a \text{ set}) r'' \longrightarrow \text{refl-on } A'' r''$   
**using** *connex-imp-refl*  
**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'*:  $\text{refl-on } A' r'$   
**using** *connex-refl lin-ord-r*  
**by** *metis*  
**hence**  $a \in A' \wedge b \in A'$   
**using** *refl-A a-pref-r-b*  
**by** *simp*  
**hence**  $b\text{-in-}r$ :  $\forall a'. a' \in A' \longrightarrow b = a' \vee (b, a') \in r' \vee (a', b) \in r'$   
**using** *lin-ord-r*  
**unfolding** *linear-order-on-def total-on-def*  
**by** *metis*  
**have**  $b\text{-in-lim-}B\text{-}r$ :  $(b, b) \in \text{limit } B r'$   
**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**  $b\text{-refl}$ :  $(b, b) \in \{(a', a''). (a', a'') \in r' \wedge a' \in B \wedge a'' \in B\}$   
**using** *b-in-lim-B-r*  
**by** *simp*  
**moreover have**  $b\text{-wins-}B$ :  $\forall b' \in B. b \in \text{above } r' b'$   
**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**  $b\text{-wins}$ :  $\forall a' \in A'. b \in \text{above } r' a'$   
**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:  $\text{partial-order-on } B (\text{limit } B r') \wedge \text{total-on } B (\text{limit } B r')$ 
using lin-ord-subset-A subset-B-card lin-ord-r
unfolding linear-order-on-def
by metis
have
 $\forall A'' r''. \text{total-on } A'' r'' =$ 
 $(\forall a'. (a'::'a) \notin A''$ 
 $\vee (\forall a''. a'' \notin A'' \vee a' = a'' \vee (a', a'') \in r'' \vee (a'', a') \in r''))$ 
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  $\text{above } r' \ a = \{a\}$   
 using *a*  
 unfolding *above-def*  
 by *blast*  
 thus ?thesis  
 using *above-a-in-A*  
 by *blast*  
 qed  
 qed  
 qed  
 hence  $\exists a. a \in A \wedge \text{above } r \ a = \{a\}$   
 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 :: 'a and
    b :: 'a
  assumes
    lin-ord: linear-order-on A r and
    fin-A: finite A and
    not-empty-A: A  $\neq$  {} and
    above-a: above r a = {a} and
    above-b: above r b = {b}
  shows a = b
proof -
  have a  $\preceq_r$  a
    using above-a singletonI pref-imp-in-above
    by metis
  also have b  $\preceq_r$  b
    using above-b singletonI pref-imp-in-above
    by metis
  moreover have
     $\exists a' \in A. \text{above } r \ a' = \{a'\} \wedge (\forall a'' \in A. \text{above } r \ a'' = \{a''\} \longrightarrow a'' = a')$ 
    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 Preference-Relation and
    a :: 'a
  assumes above r a = {a}
  shows rank r a = 1
  using assms
  by simp

```

```

lemma rank-one-imp-above-one:
  fixes
    A :: 'a set and

```

```

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

```

```

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

```

```

lemma rank-unique:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation and
    a :: 'a and
    b :: 'a
  assumes
    lin-ord: linear-order-on A r and
    fin-A: finite A and
    a-in-A: a ∈ A and
    b-in-A: b ∈ A and
    a-neq-b: a ≠ b
  shows rank r a ≠ rank r b

```



```

proof (unfold rank.simps above-def, clarify)
  assume card-eq: card {a'. (a, a') ∈ r} = card {a'. (b, a') ∈ 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) ∈ r
    using b-in-A
    unfolding refl-on-def
    by (metis (no-types))
  have rel-refl-a: (a, a) ∈ r
    using a-in-A refl-r refl-onD
    by (metis (full-types))
  obtain p :: 'a ⇒ bool where
    rel-b: ∀ y. p y = ((b, y) ∈ r)
    using is-less-preferred-than.simps
    by metis
  hence finite (Collect p)
    using refl-r refl-on-domain fin-A rev-finite-subset mem-Collect-eq subsetI
    by metis
  hence finite {a'. (b, a') ∈ r}
    using rel-b
    by (simp add: Collect-mono rev-finite-subset)
  moreover from this
  have finite {a'. (a, a') ∈ r}
    using card-eq card-gt-0-iff rel-refl-b
    by force
  moreover have trans r
    using lin-ord lin-imp-trans
    by metis
  moreover have (a, b) ∈ r ∨ (b, a) ∈ r
    using lin-ord a-in-A b-in-A a-neq-b
    unfolding linear-order-on-def total-on-def
    by metis
  ultimately have sets-eq: {a'. (a, a') ∈ r} = {a'. (b, a') ∈ r}
    using 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**  
 $A :: 'a \text{ set}$  **and**  
 $r :: 'a \text{ Preference-Relation}$  **and**  
 $a :: 'a$   
**shows**  $\text{above } (\text{limit } A \ r) \ a \subseteq A$   
**unfolding** *above-def*  
**by** *auto*

### 1.1.5 Lifting Property

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

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

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

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

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

```

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

```

```

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

```

**lemma** *lifted-mono*:

```

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

```

```

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

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

```

```

have refl-r: (a, a) ∈ 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' ∈ 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') ∈ r
  using Diff-iff singletonD
  by (metis (full-types))
qed

```

**lemma** *lifted-above-mono*:

```

fixes
  A :: 'a set and
  r :: 'a Preference-Relation and
  r' :: 'a Preference-Relation and
  a :: 'a and
  a' :: 'a
assumes
  lifted-a: lifted A r r' a and
  a'-in-A-sub-a: a' ∈ A - {a}
shows above r a' ⊆ above r' a' ∪ {a}
proof (safe)
  fix b :: 'a
  assume
    b-in-above-r: b ∈ above r a' and
    b-not-in-above-s: b ∉ above r' a'
  have ∀ b' ∈ A - {a}. (a' ⪯r b') = (a' ⪯r' b')
    using a'-in-A-sub-a lifted-a
    unfolding lifted-def equiv-rel-except-a-def
    by metis
  hence ∀ b' ∈ A - {a}. (b' ∈ above r a') = (b' ∈ above r' a')
    unfolding above-def
    by simp
  hence (b ∈ above r a') = (b ∈ above r' a')
    using lifted-a b-not-in-above-s lifted-mono limited-dest lifted-def lin-ord-imp-connex
      member-remove pref-imp-in-above
    unfolding equiv-rel-except-a-def remove-def connex-def
    by metis
  thus b = a
    using b-in-above-r b-not-in-above-s
    by simp
qed

```

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

```

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

```

$\forall A'' r''. \text{connex } A'' r'' =$   
 $(\text{limited } A'' r''$   
 $\quad \wedge (\forall b b'. (b::'a) \in A'' \longrightarrow b' \in A'' \longrightarrow (b \preceq_{r''} b' \vee b' \preceq_{r''} b)))$   
**unfolding** *connex-def*  
**by** (*simp add: Ball-def-raw*)  
**hence** *limit-rel-r*:  
 $\text{limited } A (\text{limit } A r)$   
 $\quad \wedge (\forall b b'. b \in A \wedge b' \in A \longrightarrow (b, b') \in \text{limit } A r \vee (b', b) \in \text{limit } A r)$   
**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*:  
 $\text{limited } A (\text{limit } A r')$   
 $\quad \wedge (\forall b b'. b \in A \wedge b' \in A \longrightarrow (b, b') \in \text{limit } A r' \vee (b', b) \in \text{limit } A r')$   
**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*  
 $\quad \text{strict-pref-to-a not-worse}$   
**by** *metis*  
**hence**  
 $\forall a' \in A - \{a\}.$   
 $(\text{let } q = \text{limit } A r \text{ in } a \preceq_q a') = (\text{let } q = \text{limit } A r' \text{ in } a \preceq_q a')$   
**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*  
 $\quad \text{limit-rel-s limit-rel-r}$   
**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**

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

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

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

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

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

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

$a :: 'a$  **and**

$a' :: 'a$

**assumes**

*lifted-a*:  $\text{lifted } A \ r \ r' \ a$  **and**

*a'-above-a'*:  $\text{above } r \ a' = \{a'\}$  **and**

*fin-A*: *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 above r' a' = {a'}
    thus ?thesis
      by simp
  next
    assume a'-not-above-a': above r' a' ≠ {a'}
    have ∀ a'' ∈ A. a'' ≼r a'
    proof (safe)
      fix b :: 'a
      assume y-in-A: b ∈ A
      hence A ≠ {}
      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 ≼r a'
      using y-in-A a'-above-a' lin-ord-imp-connex pref-imp-in-above
        singletonD limited-dest singletonI
      unfolding connex-def
      by (metis (no-types))
    qed
  moreover have equiv-rel-except-a A r r' a
  using lifted-a
  unfolding lifted-def
  by metis
  moreover have a' ∈ 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 ∀ a'' ∈ A - {a}. a'' ≼r' a'
  using DiffD1 lifted-a
  unfolding equiv-rel-except-a-def
  by metis
  hence ∀ a'' ∈ A - {a}. above r' a'' ≠ {a''}
  using a'-not-above-a' empty-iff insert-iff pref-imp-in-above
  by metis
  hence above r' a = {a}
  using Diff-iff all-not-in-conv lifted-a above-one singleton-iff fin-A
  unfolding lifted-def equiv-rel-except-a-def
  by metis
  thus above r' a' = {a'} ∨ above r' a = {a}
  by simp
qed
qed

theorem lifted-above-winner-single:
  fixes
    A :: 'a set and

```

```

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

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

end

```

## 1.2 Norm

```

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

```

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

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

### 1.2.1 Definition

```

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

```

```

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

```

### 1.2.2 Auxiliary Lemmas

```

lemma sum-over-image-of-bijection:

```

```

  fixes
    A :: 'a set and
    A' :: 'b set and
    f :: 'a  $\Rightarrow$  'b and
    g :: 'a  $\Rightarrow$  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
  hence card A' = 0
  using bij-betw-same-card assms
  by metis
  hence  $(\sum a \in A. g\ a) = 0 \wedge (\sum a' \in A'. g\ (\text{the-inv-into } A\ f\ a')) = 0$ 
  using 0 card-0-eq sum.empty sum.infinite

```

```

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

```

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

### 1.2.3 Common Norms

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

### 1.2.4 Properties

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

### 1.2.5 Theorems

**theorem** *l-one-is-sym*: *symmetry l-one*

**proof** (*unfold symmetry-def, safe*)

**fix**

*l* :: *ereal list* **and**

*l'* :: *ereal list*

**assume** *perm*:  $l <\sim\sim> l'$

**from** *perm* **obtain**  $\pi$

**where**

*perm* $_{\pi}$ :  $\pi$  *permutes*  $\{..< \text{length } l\}$  **and**

*l* $_{\pi}$ : *permute-list*  $\pi l = l'$

**using** *mset-eq-permutation*

**by** *metis*

```

from permπ lπ
have (∑ i < length l. |l!i|) = (∑ i < length l. |l!(π i)|)
  using permute-list-nth
  by fastforce
also have ... = (∑ i < length l. |l!(π (inv π i))|)
  using permπ permutes-inv-eq f-the-inv-into-f-bij-betw permutes-imp-bij
    sum.cong sum-over-image-of-bijection
  by (smt (verit, ccfv-SIG))
also have ... = (∑ i < length l. |l!i|)
  using permπ permutes-inv-eq
  by metis
finally have (∑ i < length l. |l!i|) = (∑ i < length l. |l!i|)
  by simp
moreover have length l = length l'
  using perm perm-length
  by metis
ultimately show l-one l = l-one l'
  using l-one.elims
  by metis
qed

end

```

## 1.3 Electoral Result

```

theory Result
  imports Main
begin

```

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

### 1.3.1 Auxiliary Functions

```

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

```

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

```

fun disjoint3 :: 'r Result ⇒ bool where

```

$$\begin{aligned} \text{disjoint3 } (e, r, d) = \\ ((e \cap r = \{\}) \wedge \\ (e \cap d = \{\}) \wedge \\ (r \cap d = \{\})) \end{aligned}$$

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

### 1.3.2 Definition

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

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

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

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

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

## 1.4 Preference Profile

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



**begin**

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

### 1.4.1 Definition

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

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

**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**  $\text{finite-elections-}\mathcal{V} :: (\text{'a}, \text{'v}) \text{ Election set} \textbf{ where}$

$finite\text{-elections-}\mathcal{V} = \{E :: ('a, 'v) \text{ Election. finite (voters-}\mathcal{E} \ E)\}$

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

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

— This function subsumes elections with fixed alternatives, finite voters, and a default value for the profile value on non-voters.

**fun**  $elections\text{-}\mathcal{A} :: 'a \text{ set} \Rightarrow ('a, 'v) \text{ Election set where}$   
 $elections\text{-}\mathcal{A} \ A = valid\text{-elections} \cap$   
 $\{E. \text{alternatives-}\mathcal{E} \ E = A \wedge \text{finite (voters-}\mathcal{E} \ E) \wedge (\forall v. v \notin \text{voters-}\mathcal{E} \ E \longrightarrow$   
 $\text{profile-}\mathcal{E} \ E \ v = \{\})\}$

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

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

### 1.4.2 Vote Count

**lemma**  $sum\text{-comp}$ :

**fixes**

$f :: 'x \Rightarrow 'z :: comm\text{-monoid-add}$  **and**

$g :: 'y \Rightarrow 'x$  **and**

$X :: 'x \text{ set}$  **and**

$Y :: 'y \text{ set}$

**assumes**  $bij\text{-betw } g \ Y \ X$

**shows**  $sum \ f \ X = sum \ (f \circ g) \ Y$

**using**  $assms$

**proof** ( $induction \ card \ X \ arbitrary: X \ Y \ f \ g$ )

**case**  $0$

**assume**  $bij\text{-betw } g \ Y \ X$

**hence**  $card \ Y = 0$

**using**  $bij\text{-betw-same-card } 0.hyps$

**unfolding**  $0.hyps$

**by**  $simp$

**hence**  $sum \ f \ X = 0 \wedge sum \ (f \circ g) \ Y = 0$

**using**  $assms \ 0 \ card\text{-}0\text{-eq} \ sum.empty \ sum.infinite$

**by**  $metis$

**thus**  $?case$

**by**  $simp$

**next**

**case**  $(Suc \ n)$

**assume**

$card\text{-}X: Suc \ n = card \ X$  **and**

$bij: bij\text{-betw } g \ Y \ X$  **and**

$hyp: \bigwedge X \ Y \ f \ g. n = card \ X \Longrightarrow bij\text{-betw } g \ Y \ X \Longrightarrow sum \ f \ X = sum \ (f \circ g) \ Y$

```

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

lemma vote-count-sum:
  fixes  $E :: ('a, 'v) \text{ Election}$ 
  assumes
     $\text{finite } (\text{voters-}\mathcal{E} \ E)$  and
     $\text{finite } (\text{UNIV} :: ('a \times 'a) \text{ set})$ 
  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  $\text{disjoint } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\}$ 
  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**  
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\} =$   
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \cup$   
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}$   
**by** *blast*  
**moreover have**  
 $\{\} = \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} \cap$   
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}$   
**by** *blast*  
**ultimately have**  $\text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p. p \in \text{UNIV}\}$   
 $=$   
 $\text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\} +$   
 $\text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}$   
**using** *sum.union-disjoint*[of  
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\}$   
 $\{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}]$   
**by** *simp*  
**moreover have**  
 $\forall X \in \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\}\}. \text{card } X = 0$   
**using** *card-eq-0-iff*  
**by** *fastforce*  
**ultimately have** *card-eq-sum*:  
 $\text{card } (\text{voters-}\mathcal{E} \ E) = \text{sum card } \{\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \mid p.$   
 $p \in \text{UNIV} \wedge \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\}$   
**using** *card-eq-sum'*  
**by** *simp*  
**have** *inj-on*  $(\lambda p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\})$   
 $\{p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\}$   
**unfolding** *inj-on-def*  
**by** *blast*  
**moreover have**  
 $(\lambda p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}) \text{ ' } \{p. \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} \neq \{\}\}$

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

```

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

```

### 1.4.3 Voter Permutations

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

```

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

```

**lemma** rename-sound:

```

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

```

**lemma** rename-finite:

```

fixes

```

```

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

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

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

```

```

shows inj (rename  $\pi$ )
proof (unfold inj-def split-paired-All rename.simps prod.inject,
        intro allI impI, elim conjE)
  fix
     $A :: 'a \text{ set}$  and
     $A' :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $V' :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  and
     $p' :: ('a, 'v) \text{ Profile}$ 
  assume
     $A = A'$  and
     $p \circ \text{the-inv } \pi = p' \circ \text{the-inv } \pi$  and
     $\pi \text{ ' } V = \pi \text{ ' } V'$ 
  thus  $A = A' \wedge V = V' \wedge p = p'$ 
    using assms bij-betw-the-inv-into bij-is-surj surj-fun-eq
      bij-betw-imp-inj-on inj-image-eq-iff
    by metis
qed

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

```



```

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

```

#### 1.4.4 List Representation for Ordered Voters

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** *map2-helper*:

```

fixes
  f :: 'x ⇒ 'y ⇒ 'z and
  g :: 'x ⇒ 'x and
  h :: 'y ⇒ 'y and
  l1 :: 'x list and
  l2 :: 'y list
shows map2 f (map g l1) (map h l2) = map2 (λ x y. f (g x) (h y)) l1 l2
proof –
  have map2 f (map g l1) (map h l2) = map (λ (x, y). f x y) (zip (map g l1) (map
  h l2))
  by simp

```

```

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

```

**lemma** *to-list-simp*:

```

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

```

**lemma** *to-list-comp*:

```

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

```

```

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

```

```

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

```

```

lemma sorted-list-of-set-nth-equals-card:
  fixes
     $V :: 'v::\text{linorder set}$  and

```

$x :: 'v$   
**assumes**  
 $fin-V: \text{finite } V$  **and**  
 $x-V: x \in V$   
**shows**  $\text{sorted-list-of-set } V!(\text{card } \{v \in V. v < x\}) = x$   
**proof** –  
**let**  $?c = \text{card } \{v \in V. v < x\}$  **and**  
 $?set = \{v \in V. v < x\}$   
**have**  $ex\text{-index}: \forall v \in V. \exists n. n < \text{card } V \wedge (\text{sorted-list-of-set } V!n) = v$   
**using**  $\text{sorted-list-of-set.distinct-sorted-key-list-of-set}$   
 $\text{sorted-list-of-set.length-sorted-key-list-of-set}$   
 $\text{sorted-list-of-set.set-sorted-key-list-of-set}$   
 $\text{distinct-Ex1 } fin-V$   
**by** *metis*  
**then obtain**  $\varphi$  **where**  
 $index\text{-}\varphi: \forall v \in V. \varphi v < \text{card } V \wedge (\text{sorted-list-of-set } V!(\varphi v)) = v$   
**by** *metis*  
–  $\varphi x = ?c$ , i.e.,  $\varphi x \geq ?c$  and  $\varphi x \leq ?c$   
**let**  $?i = \varphi x$   
**have**  $inj\text{-}\varphi: \text{inj-on } \varphi V$   
**using**  $inj\text{-onI } index\text{-}\varphi$   
**by** *metis*  
**have**  $mono\text{-}\varphi: \forall v v'. v \in V \wedge v' \in V \wedge v < v' \longrightarrow \varphi v < \varphi v'$   
**using**  $\text{sorted-list-of-set.idem-if-sorted-distinct dual-order.strict-trans2 } fin-V \text{ in-}$   
 $\text{dex-}\varphi$   
 $\text{finite-sorted-distinct-unique linorder-neqE-nat sorted-wrt-iff-nth-less}$   
 $\text{sorted-list-of-set.length-sorted-key-list-of-set order-less-irrefl}$   
**by** (*metis (full-types)*)  
**have**  $\forall v \in ?set. v < x$   
**by** *simp*  
**hence**  $\forall v \in ?set. \varphi v < ?i$   
**using**  $mono\text{-}\varphi x-V$   
**by** *simp*  
**hence**  $\forall j \in \{\varphi v \mid v. v \in ?set\}. ?i > j$   
**by** *blast*  
**moreover have**  $fin\text{-}img: \text{finite } ?set$   
**using**  $fin-V$   
**by** *simp*  
**ultimately have**  $?i \geq \text{card } \{\varphi v \mid v. v \in ?set\}$   
**using**  $\text{set-card-upper-bound}$   
**by** *simp*  
**also have**  $\text{card } \{\varphi v \mid v. v \in ?set\} = ?c$   
**using**  $inj\text{-}\varphi$   
**by** (*simp add: card-image inj-on-subset setcompr-eq-image*)  
**finally have**  $geq: ?i \geq ?c$   
**by** *simp*  
**have**  $\text{sorted-}\varphi:$   
 $\forall i j. i < \text{card } V \wedge j < \text{card } V \wedge i < j$   
 $\longrightarrow (\text{sorted-list-of-set } V!i) < (\text{sorted-list-of-set } V!j)$

by (*simp add: sorted-wrt-nth-less*)  
 have *leq*:  $?i \leq ?c$   
 proof (rule *ccontr*, cases  $?c < \text{card } V$ )  
   case *True*  
   let  $?A = \lambda j. \{\text{sorted-list-of-set } V!j\}$   
   assume  $\neg ?i \leq ?c$   
   hence  $?i > ?c$   
   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- $\varphi$  dual-order.strict-trans2 geq index- $\varphi$  x-V fin-V*  
       *nth-mem sorted-list-of-set.length-sorted-key-list-of-set*  
       *sorted-list-of-set.set-sorted-key-list-of-set*  
   by (*metis (mono-tags, lifting)*)  
   hence  $\{\text{sorted-list-of-set } V!j \mid j. j \leq ?c\} \subseteq \{v \in V. v < x\}$   
   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  $\forall i \leq ?c. \forall j \leq ?c. i \neq j \longrightarrow \text{sorted-list-of-set } V!i \neq \text{sorted-list-of-set } V!j$   
 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 *finite*  $\{0 \dots (?c+1)\}$   
   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*  
   also have  $(\sum j \in \{0 \dots (?c+1)\}. 1) = ?c + 1$   
   by *auto*  
   finally have  $\text{card } (\bigcup j \in \{0 \dots (?c+1)\}. ?A j) = ?c + 1$   
   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
  assume  $\neg ?i \leq ?c$ 
  thus False
  using False  $x-V$  index- $\varphi$  geq order-le-less-trans
  by blast
qed
thus ?thesis
  using geq leq  $x-V$  index- $\varphi$ 
  by simp
qed

lemma to-list-permutes-under-bij:
  fixes
     $\pi :: 'v::\text{linorder} \Rightarrow 'v$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  assumes
     $\text{bij}: \text{bij } \pi$ 
  shows
    let  $\varphi = (\lambda i. \text{card } \{v \in \pi \text{ ` } V. v < \pi ((\text{sorted-list-of-set } V)!i)\})$ 
    in  $(\text{to-list } V p) = \text{permute-list } \varphi (\text{to-list } (\pi \text{ ` } V) (\lambda x. p (\text{the-inv } \pi x)))$ 
  proof (cases finite V)
  case False
  — If  $V$  is infinite, both lists are empty.
  hence  $\text{to-list } V p = []$ 
  by simp
  moreover have  $\text{to-list } (\pi \text{ ` } V) (\lambda x. p (\text{the-inv } \pi x)) = []$ 
  proof —
  have infinite  $(\pi \text{ ` } V)$ 
  using False assms bij-betw-finite bij-betw-subset top-greatest
  by metis
  thus ?thesis
  by simp
qed
ultimately show ?thesis
  by simp
next
  case True
  let
     $?q = \lambda x. p (\text{the-inv } \pi x)$  and
     $?img = \pi \text{ ` } V$  and
     $?n = \text{length } (\text{to-list } V p)$  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:  $\text{card } ?img = \text{card } V$ 
  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 (λ i. to-list ?img ?q ! card {v ∈ ?img. v < π (sorted-list-of-set
V!i)}))
          [0 ..< length (to-list ?img ?q)])
    using eq-length
    by simp
  next
    — The ith entries of the lists coincide.
    fix i :: nat
    assume in-bnds: i < ?n
    let ?c = card {v ∈ ?img. v < π (sorted-list-of-set V!i)}
    have map (λ i. (to-list ?img ?q)!?c) [0 ..< ?n]!i = p ((sorted-list-of-set V)!i)
    proof —
      have ∀ v. v ∈ ?img ⟶ {v' ∈ ?img. v' < v} ⊆ ?img − {v}
        by blast
      moreover have elem-of-img: π (sorted-list-of-set V!i) ∈ ?img
        using True in-bnds image-eqI nth-mem card-length-V
          sorted-list-of-set.length-sorted-key-list-of-set
          sorted-list-of-set.set-sorted-key-list-of-set
        by metis
      ultimately have {v ∈ ?img. v < π (sorted-list-of-set V!i)}
        ⊆ ?img − {π (sorted-list-of-set V!i)}
        by simp
      hence {v ∈ ?img. v < π (sorted-list-of-set V!i)} ⊂ ?img
        using elem-of-img
        by blast
      moreover have img-card-eq-V-length: card ?img = ?n
        using card-eq card-length-V
        by presburger
      ultimately have card-in-bnds: ?c < ?n
        using True finite-imageI psubset-card-mono
        by (metis (mono-tags, lifting))
      moreover have img-list-map:
        map (λ i. to-list ?img ?q!?c) [0 ..< ?n]!i = to-list ?img ?q!?c
        using in-bnds
        by simp
      also have img-list-card-eq-inv-img-list:

```

```

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

```

#### 1.4.5 Preference Counts and Comparisons

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

```

fun win-count :: 'v set  $\Rightarrow$  ('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 infinity)

```

```

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 infinity)

```

**lemma** pref-count-voter-set-card:

```

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

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

$f :: 'a \Rightarrow 'a \text{ set}$

**shows**  $\{f\ x \mid x. x \in A\} = f\ ` A$

**by** *auto*

**lemma** *pref-count-set-compr*:

**fixes**

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

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

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

$a :: 'a$

**shows**  $\{prefer\_count\ V\ p\ a\ a' \mid a'. a' \in A - \{a\}\} = (prefer\_count\ V\ p\ a)\ ` (A - \{a\})$

**by** *auto*

**lemma** *pref-count*:

**fixes**

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

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

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

$a :: 'a$  **and**

$b :: 'a$

**assumes**

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

*fin*: *finite*  $V$  **and**

*a-in-A*:  $a \in A$  **and**

*b-in-A*:  $b \in A$  **and**

*neg*:  $a \neq b$

**shows**  $prefer\_count\ V\ p\ a\ b = card\ V - (prefer\_count\ V\ p\ b\ a)$

**proof**  $-$

**have**  $\forall\ v \in V. \text{connex}\ A\ (p\ v)$

**using** *prof*

**unfolding** *profile-def*

**by** (*simp add: lin-ord-imp-connex*)

**hence** *asym*:  $\forall\ v \in V. \neg (\text{let } r = (p\ v) \text{ in } (b \preceq_r a)) \longrightarrow (\text{let } r = (p\ v) \text{ in } (a \preceq_r b))$

**using** *a-in-A b-in-A*

**unfolding** *connex-def*

**by** *metis*

**have**  $\forall\ v \in V. ((b, a) \in (p\ v) \longrightarrow (a, b) \notin (p\ v))$

**using** *antisymD neg lin-imp-antisym prof*

**unfolding** *profile-def*

**by** *metis*

**hence**  $\{v \in V. (\text{let } r = (p\ v) \text{ in } (b \preceq_r a))\} =$

$V - \{v \in V. (\text{let } r = (p\ v) \text{ in } (a \preceq_r b))\}$

**using** *asym*

**by** *auto*

```

thus ?thesis
  by (simp add: card-Diff-subset Collect-mono fin)
qed

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

```

```

    by simp
  thus ?thesis
    by simp
next
  assume inf-V: infinite V
  have prefer-count V p c a = infinity
    using inf-V
    by simp
  moreover have prefer-count V p b c = infinity
    using inf-V
    by simp
  thus ?thesis
    by simp
qed

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

fun wins :: 'v set  $\Rightarrow$  'a  $\Rightarrow$  ('a, 'v) Profile  $\Rightarrow$  'a  $\Rightarrow$  bool where
  wins V a p b =
    (prefer-count V p a b > prefer-count V p b a)

lemma wins-inf-voters:
  fixes
    p :: ('a, 'v) Profile and
    a :: 'a and
    b :: 'a and
    V :: 'v set
  assumes infinite V
  shows wins V b p a = False
  using assms
  by simp

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

lemma wins-antisym:
  fixes
    p :: ('a, 'v) Profile and
    a :: 'a and
    b :: 'a and
    V :: 'v set

```

**assumes**  $\text{wins } V a p b$  — 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*

#### 1.4.6 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**  
 $a :: 'a$  **and**  
 $b :: 'a$   
**assumes**  
 $\text{condorcet-winner } V A p a$  **and**  
 $\text{condorcet-winner } V A p b$   
**shows**  $b = a$   
**proof** (*rule ccontr*)  
**assume**  $b \neq a$   
**have**  $\text{wins } V b p a$   
**using** *b-neq-a insert-Diff insert-iff assms*  
**by** *simp*  
**hence**  $\neg \text{wins } V a p b$   
**by** (*simp add: wins-antisym*)  
**moreover have**  $a \text{-wins-against-} b: \text{wins } V a p b$   
**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-2:
  fixes
    V :: 'v set and
    A :: 'a set and
    p :: ('a, 'v) Profile and
    a :: 'a and
    b :: 'a
  assumes
    condorcet-winner V A p a and
    b ≠ a
  shows ¬ condorcet-winner V A p b
  using cond-winner-unique-eq assms
  by metis

```

### 1.4.7 Limited Profile

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

```

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

```

```

lemma limit-prof-trans:
  fixes
    A :: 'a set and
    B :: 'a set and
    C :: 'a set and
    p :: ('a, 'v) Profile

```

```

assumes
   $B \subseteq A$  and
   $C \subseteq B$ 
shows  $\text{limit-profile } C \ p = \text{limit-profile } C \ (\text{limit-profile } B \ p)$ 
using assms
by auto

lemma limit-profile-sound:
fixes
   $A :: 'a \text{ set}$  and
   $B :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
assumes
  profile:  $\text{profile } V \ B \ p$  and
  subset:  $A \subseteq B$ 
shows  $\text{profile } V \ A \ (\text{limit-profile } A \ p)$ 
proof –
have  $\forall v \in V. \text{linear-order-on } A \ (\text{limit } A \ (p \ v))$ 
using profile subset limit-presv-lin-ord
unfolding profile-def
by metis
hence  $\forall v \in V. \text{linear-order-on } A \ ((\text{limit-profile } A \ p) \ v)$ 
by simp
thus ?thesis
unfolding profile-def
by simp
qed

```

#### 1.4.8 Lifting Property

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

$$\text{equiv-prof-except-a } V \ A \ p \ p' \ a \equiv$$

$$\text{profile } V \ A \ p \wedge \text{profile } V \ A \ p' \wedge a \in A \wedge$$

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

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

**definition** *lifted* ::  $'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow 'a \Rightarrow \text{bool}$  **where**

$$\text{lifted } V \ A \ p \ p' \ a \equiv$$

$$\text{finite-profile } V \ A \ p \wedge \text{finite-profile } V \ A \ p' \wedge a \in A$$

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

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

**lemma** *lifted-imp-equiv-prof-except-a*:

**fixes**

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

```

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

```

**lemma** *negl-diff-imp-eq-limit-prof*:

```

fixes
  A :: 'a set and
  A' :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  p' :: ('a, 'v) Profile and
  a :: 'a
assumes
  change: equiv-prof-except-a V A' p q a and
  subset: A ⊆ A' and
  not-in-A: a ∉ A

```

**shows**  $\forall v \in V. (\text{limit-profile } A \ p) \ v = (\text{limit-profile } A \ q) \ v$

— With the current definitions of *equiv-prof-except-a* and *limit-prof*, we can only conclude that the limited profiles coincide on the given voter set, since *limit-prof* may change the profiles everywhere, while *equiv-prof-except-a* only makes statements about the voter set.

**proof** (*clarify*)

```

fix
   $v :: 'v$ 
assume  $v \in V$ 
hence  $\text{equiv-rel-except-a } A' (p\ v) (q\ v)\ a$ 
  using  $\text{change equiv-prof-except-a-def}$ 
  by  $\text{metis}$ 
hence  $\text{limit } A (p\ v) = \text{limit } A (q\ v)$ 
  using  $\text{not-in-A negl-diff-imp-eq-limit subset}$ 
  by  $\text{metis}$ 
thus  $\text{limit-profile } A\ p\ v = \text{limit-profile } A\ q\ v$ 
  by  $\text{simp}$ 
qed

lemma  $\text{limit-prof-eq-or-lifted}$ :
fixes
   $A :: 'a\ \text{set}$  and
   $A' :: 'a\ \text{set}$  and
   $V :: 'v\ \text{set}$  and
   $p :: ('a, 'v)\ \text{Profile}$  and
   $p' :: ('a, 'v)\ \text{Profile}$  and
   $a :: 'a$ 
assumes
   $\text{lifted-a: lifted } V\ A'\ p\ p'\ a$  and
   $\text{subset: } A \subseteq A'$ 
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  $(\text{cases})$ 
assume  $a\text{-in-}A: a \in A$ 
have  $\forall v \in V. (\text{Preference-Relation.lifted } A'\ (p\ v)\ (p'\ v)\ a \vee (p\ v) = (p'\ v))$ 
  using  $\text{lifted-a}$ 
  unfolding  $\text{lifted-def}$ 
  by  $\text{metis}$ 
hence  $\text{one:}$ 
   $\forall v \in V.$ 
     $(\text{Preference-Relation.lifted } A\ (\text{limit } A\ (p\ v))\ (\text{limit } A\ (p'\ v))\ a \vee$ 
     $(\text{limit } A\ (p\ v)) = (\text{limit } A\ (p'\ v)))$ 
  using  $\text{limit-lifted-imp-eq-or-lifted subset}$ 
  by  $\text{metis}$ 
thus  $?thesis$ 
proof  $(\text{cases})$ 
assume  $\forall v \in V. (\text{limit } A\ (p\ v)) = (\text{limit } A\ (p'\ v))$ 
thus  $?thesis$ 
  by  $\text{simp}$ 
next
assume  $\text{forall-limit-p-q:}$ 
   $\neg (\forall v \in V. (\text{limit } A\ (p\ v)) = (\text{limit } A\ (p'\ v)))$ 
let  $?p = \text{limit-profile } A\ p$ 
let  $?q = \text{limit-profile } A\ p'$ 
have  $\text{profile } V\ A\ ?p \wedge \text{profile } V\ A\ ?q$ 

```



```

    using lifted-a limit-profile-sound subset
    unfolding lifted-def
    by metis
  moreover have
     $\exists v \in V. \text{Preference-Relation.lifted } A \text{ } (?p \ v) \text{ } (?q \ v) \ a$ 
    using forall-limit-p-q lifted-a limit-profile.simps one
    unfolding lifted-def
    by (metis (no-types, lifting))
  moreover have
     $\forall v \in V. (\neg \text{Preference-Relation.lifted } A \text{ } (?p \ v) \text{ } (?q \ v) \ a) \longrightarrow (?p \ v) = (?q \ v)$ 
    using lifted-a limit-profile.simps one
    unfolding lifted-def
    by metis
  ultimately have lifted V A ?p ?q a
    using a-in-A lifted-a rev-finite-subset subset
    unfolding lifted-def
    by (metis (no-types, lifting))
  thus ?thesis
    by simp
qed
next
  assume  $a \notin A$ 
  thus ?thesis
    using lifted-a negl-diff-imp-eq-limit-prof subset lifted-imp-equiv-prof-except-a
    by metis
qed
end

```

## 1.5 Social Choice Result

```

theory Social-Choice-Result
  imports Result
begin

```

### 1.5.1 Social Choice Result

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

```

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

```

```

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

```

### 1.5.2 Auxiliary Lemmas

**lemma** *result-imp-rej*:

```

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

```

```

    by simp
  ultimately show False
    by blast
qed

```

**lemma** *result-count*:

```

  fixes
    A :: 'a set and
    e :: 'a set and
    r :: 'a set and
    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 ∈ defer-r r
  moreover obtain
    f :: 'a Result ⇒ 'a set ⇒ 'a set and
    g :: 'a Result ⇒ 'a set ⇒ 'a Result where
    A = f r A ∧ r = g r A ∧ disjoint3 (g r A) ∧ set-equals-partition (f r A) (g r A)
    using assms
    by simp
  moreover have ∀ p. ∃ e r d. set-equals-partition A p ⟶ (e, r, d) = p ∧ e ∪
r ∪ d = A
    by simp
  ultimately show a ∈ A
    using UnCI snd-conv
    by metis
qed

```

```

lemma elect-subset:
  fixes
     $A :: 'a \text{ set}$  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

```

## 1.6 Social Welfare Result

```

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

```

### 1.6.1 Social Welfare Result

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

```

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

```

```

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

end

```

## 1.7 Specific Electoral Result Types

```

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

```

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

```

setup Locale-Code.open-block

```

Results from social choice functions (*SCFs*), 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-set-SCF
proof (unfold-locales, safe)
  fix

```

```

  A :: 'a set and
  e :: 'a set and
  r :: 'a set and
  d :: 'a set
assume
  set-equals-partition (limit-set-SCF A UNIV) (e, r, d) 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.

**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 and
  e :: 'b set set and
  r :: 'b set set and
  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 (*SWFs*), 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-set-SWF

```

**proof** (*unfold-locales, safe*)

```

fix
  A :: 'a set and
  e :: ('a Preference-Relation) set and
  r :: ('a Preference-Relation) set and
  d :: ('a Preference-Relation) set
assume
  partition: set-equals-partition (limit-set-SWF A UNIV) (e, r, d) and
  disj: disjoint3 (e, r, d)
have limit-set-SWF A UNIV =
   $\{limit A r' \mid r'. r' \in UNIV \wedge linear-order-on A (limit A r')\}$ 
  by simp
also have ... =  $\{limit A r' \mid r'. r' \in UNIV\} \cap$ 
   $\{limit A r' \mid r'. linear-order-on A (limit A r')\}$ 
  by blast
also have ... =  $\{limit A r' \mid r'. linear-order-on A (limit A r')\}$ 

```

```

    by blast
  also have ... = {r'. linear-order-on A r'}
  proof (safe)
    fix r' :: 'a Preference-Relation
    assume lin-ord: linear-order-on A r'
    hence  $\forall a b. (a, b) \in r' \longrightarrow (a, b) \in \text{limit } A \ r'$ 
      unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
      by force
    hence  $r' \subseteq \text{limit } A \ r'$ 
      by slow
    moreover have  $\text{limit } A \ r' \subseteq r'$ 
      by auto
    ultimately have  $r' = \text{limit } A \ r'$ 
      by safe
    thus  $\exists x. r' = \text{limit } A \ x \wedge \text{linear-order-on } A \ (\text{limit } A \ x)$ 
      using lin-ord
      by metis
  qed
  thus well-formed-SWF A (e, r, d)
    using partition disj
    by simp
  qed

  setup Locale-Code.close-block

end

```

## 1.8 Function Symmetry Properties

```

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

```

### 1.8.1 Functions

```

type-synonym ('x, 'y) 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}

```

Relations

```

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-induced-rel s t  $\varphi$  = {(y, y')  $\in$  t  $\times$  t.  $\exists$  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-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 set-closed-rel :: 'x set  $\Rightarrow$  'x rel  $\Rightarrow$  bool where
  set-closed-rel s r = ( $\forall$  x y. (x, y)  $\in$  r  $\longrightarrow$  x  $\in$  s  $\longrightarrow$  y  $\in$  s)

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

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

```

## 1.8.2 Invariance and Equivariance

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

```

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

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

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

```

## 1.8.3 Auxiliary Lemmas

```

lemma inj-imp-inj-on-set-system:
  fixes f :: 'x  $\Rightarrow$  'y
  assumes inj f
  shows inj ( $\lambda$  s. {f 'x | x. x  $\in$  s})
proof (unfold inj-def, safe)
  fix

```



```

s :: 'x set set and
t :: 'x set set and
x :: 'x set
assume f-elem-s-eq-f-elem-t: {f ' x' | x'. x' ∈ s} = {f ' x' | x'. x' ∈ t}
then obtain y :: 'x set where
  f ' y = f ' x
  by metis
hence y-eq-x: y = x
  using image-inv-f-f assms
  by metis
moreover have
  x ∈ t ⟶ f ' x ∈ {f ' x' | x'. x' ∈ s} and
  x ∈ s ⟶ f ' x ∈ {f ' x' | x'. x' ∈ t}
  using f-elem-s-eq-f-elem-t
  by auto
ultimately have x ∈ t ⟶ y ∈ s and x ∈ s ⟶ y ∈ t
  using assms
  by (simp add: inj-image-eq-iff, simp add: inj-image-eq-iff)
thus x ∈ t ⟹ x ∈ s and x ∈ s ⟹ x ∈ t
  using y-eq-x
  by (simp, simp)
qed

```

```

lemma inj-and-surj-imp-surj-on-set-system:
  fixes f :: 'x ⇒ 'y
  assumes
    inj f and
    surj f
  shows surj (λ s. {f ' x | x. x ∈ s})
proof (unfold surj-def, safe)
  fix s :: 'y set set
  have ∀ x. f ' (the-inv f) ' x = x
    using image-f-inv-f assms surj-imp-inv-eq the-inv-f-f
    by (metis (no-types, opaque-lifting))
  hence s = {f ' (the-inv f) ' x | x. x ∈ s}
    by simp
  also have {f ' (the-inv f) ' x | x. x ∈ s} =
    {f ' x | x. x ∈ {(the-inv f) ' x | x. x ∈ s}}
    by blast
  finally show ∃ t. s = {f ' x | x. x ∈ t}
    by blast
qed

```

```

lemma bij-imp-bij-on-set-system:
  fixes f :: 'x ⇒ 'y
  assumes bij f
  shows bij (λ s. {f ' x | x. x ∈ s})
proof (unfold bij-def)
  have range f = UNIV

```

```

    using assms
    unfolding bij-betw-def
    by safe
  moreover have inj f
    using assms
    unfolding bij-betw-def
    by safe
  ultimately show inj ( $\lambda s. \{f \, 'x \mid x. x \in s\}$ )  $\wedge$  surj ( $\lambda s. \{f \, 'x \mid x. x \in s\}$ )
    using inj-imp-inj-on-set-system
    by (simp add: inj-and-surj-imp-surj-on-set-system)
qed

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

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

```

**ultimately have**  $(f \circ g) (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)) = x$   
**by** *simp*  
**hence**  $the\_inv\_into\ s\ (f \circ g)\ x =$   
 $the\_inv\_into\ s\ (f \circ g)\ ((f \circ g)\ (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)))$   
**by** *presburger*  
**also have**  
 $the\_inv\_into\ s\ (f \circ g)\ ((f \circ g)\ (the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x))) =$   
 $the\_inv\_into\ s\ g\ (the\_inv\_into\ t\ f\ x)$   
**using** *assms bij-betw-apply bij-betw-imp-inj-on bij-betw-the-inv-into bij-betw-trans*  
 $the\_inv\_into-f-eq$   
**by** *(metis (no-types, lifting))*  
**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**

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

#### 1.8.4 Rewrite Rules

**theorem** *rewrite-invar-as-equivar*:  
**fixes**  
 $f :: 'x \Rightarrow 'y$  **and**  
 $s :: 'x\ set$  **and**  
 $t :: 'z\ set$  **and**

```

   $\varphi :: ('z, 'x) \text{ binary-fun}$ 
shows is-symmetry  $f$  (Invariance (action-induced-rel  $t\ s\ \varphi$ )) =
  is-symmetry  $f$  (action-induced-equivariance  $t\ s\ \varphi\ (\lambda\ g.\ id)$ )
proof (unfold action-induced-equivariance-def is-symmetry.simps action-induced-rel.simps,
safe)
fix
   $x :: 'x$  and
   $y :: 'z$ 
assume
   $x \in s$  and
   $y \in t$  and
   $\varphi\ y\ x \in s$ 
thus
   $\forall x' y'. (x', y') \in \{(y, y'). (y, y') \in s \times s \wedge (\exists z \in t. \varphi\ z\ y = y')\}$ 
     $\longrightarrow f\ x' = f\ y' \implies f\ (\varphi\ y\ x) = id\ (f\ x)$  and
   $\forall (\varphi', \psi') \in \{(\varphi\ x, id) \mid x. x \in t\}.$ 
     $\forall x' \in s. \varphi'\ x' \in s \longrightarrow f\ (\varphi'\ x') = \psi'\ (f\ x') \implies f\ x = f\ (\varphi\ y\ x)$ 
unfolding id-def
using SigmaI case-prodI mem-Collect-eq
by (metis (mono-tags, lifting), fastforce)
qed

```

**lemma** *rewrite-invar-ind-by-act*:

```

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

```

ultimately show *is-symmetry*  $f$  (*Invariance* (*action-induced-rel*  $s$   $t$   $\varphi$ ))  
 by *auto*  
 qed

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

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

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

lemma *universal-set-carrier-imp-bij-group*:  
 fixes  $f :: 'a \Rightarrow 'a$

```

assumes  $f \in \text{carrier } (\text{BijGroup } \text{UNIV})$ 
shows  $\text{bij } f$ 
using  $\text{rewrite-carrier } \text{assms}$ 
by  $\text{blast}$ 

lemma  $\text{rewrite-sym-group}$ :
  fixes
     $f :: 'a \Rightarrow 'a$  and
     $g :: 'a \Rightarrow 'a$  and
     $s :: 'a \text{ set}$ 
  assumes
     $f\text{-carrier}: f \in \text{carrier } (\text{BijGroup } s)$  and
     $g\text{-carrier}: g \in \text{carrier } (\text{BijGroup } s)$ 
  shows
     $\text{rewrite-mult}: f \otimes \text{BijGroup } s \ g = \text{extensional-continuation } (f \circ g) \ s$  and
     $\text{rewrite-mult-univ}: s = \text{UNIV} \longrightarrow f \otimes \text{BijGroup } s \ g = f \circ g$ 
proof -
  show  $f \otimes \text{BijGroup } s \ g = \text{extensional-continuation } (f \circ g) \ s$ 
    using  $f\text{-carrier } g\text{-carrier}$ 
    unfolding  $\text{BijGroup-def compose-def comp-def restrict-def}$ 
    by  $\text{simp}$ 
next
  show  $s = \text{UNIV} \longrightarrow f \otimes \text{BijGroup } s \ g = f \circ g$ 
    using  $f\text{-carrier } g\text{-carrier}$ 
    unfolding  $\text{BijGroup-def compose-def comp-def restrict-def}$ 
    by  $\text{fastforce}$ 
qed

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

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

lemma  $\text{rel-ind-by-coinciding-action-on-subset-eq-restr}$ :

```

```

fixes
   $\varphi :: ('a, 'b) \text{ binary-fun}$  and
   $\psi :: ('a, 'b) \text{ binary-fun}$  and
   $s :: 'a \text{ set}$  and
   $t :: 'b \text{ set}$  and
   $u :: 'b \text{ set}$ 
assumes
   $u \subseteq t$  and
   $\forall x \in s. \forall y \in u. \psi \ x \ y = \varphi \ x \ y$ 
shows  $\text{action-induced-rel } s \ u \ \psi = \text{Restr } (\text{action-induced-rel } s \ t \ \varphi) \ u$ 
proof ( $\text{unfold action-induced-rel.simps}$ )
have  $\{(x, y). (x, y) \in u \times u \wedge (\exists z \in s. \psi \ z \ x = y)\}$ 
   $= \{(x, y). (x, y) \in u \times u \wedge (\exists z \in s. \varphi \ z \ x = y)\}$ 
using  $\text{assms}$ 
by  $\text{auto}$ 
also have  $\dots = \text{Restr } \{(x, y). (x, y) \in t \times t \wedge (\exists z \in s. \varphi \ z \ x = y)\} \ u$ 
using  $\text{assms}$ 
by  $\text{blast}$ 
finally show  $\{(x, y). (x, y) \in u \times u \wedge (\exists z \in s. \psi \ z \ x = y)\} =$ 
   $\text{Restr } \{(x, y). (x, y) \in t \times t \wedge (\exists z \in s. \varphi \ z \ x = y)\} \ u$ 
by  $\text{simp}$ 
qed

```

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

```

fixes
   $s :: 'x \text{ set}$  and
   $t :: 'y \text{ set}$  and
   $\varphi :: ('x, 'y) \text{ binary-fun}$  and
   $\psi :: ('x, 'y) \text{ binary-fun}$ 
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  $\text{extensional-continuation.simps}$ 
using  $\text{assms}$ 
by  $\text{auto}$ 

```

### 1.8.5 Group Actions

**lemma** *const-id-is-group-act:*

```

fixes  $m :: 'x \text{ monoid}$ 
assumes  $\text{group } m$ 
shows  $\text{group-action } m \ \text{UNIV } (\lambda x. \text{id})$ 
proof ( $\text{unfold group-action-def group-hom-def group-hom-axioms-def hom-def, safe}$ )
show  $\text{group } m$ 
using  $\text{assms}$ 
by  $\text{blast}$ 
next
show  $\text{group } (\text{BijGroup } \text{UNIV})$ 
using  $\text{group-BijGroup}$ 
by  $\text{metis}$ 

```

```

next
  show  $id \in \text{carrier } (\text{BijGroup } UNIV)$ 
    unfolding BijGroup-def Bij-def
    by simp
  thus  $id = id \otimes \text{BijGroup } UNIV \text{ } id$ 
    using rewrite-mult-univ comp-id
    by metis
qed

theorem group-act-induces-set-group-act:
  fixes
     $m :: 'x \text{ monoid}$  and
     $s :: 'y \text{ set}$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$ 
  defines  $\varphi\text{-img} \equiv (\lambda x. \text{extensional-continuation } (\text{image } (\varphi x)) (Pow s))$ 
  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 car-x:  $x \in \text{carrier } m$ 
    hence bij-betw  $(\varphi x) s s$ 
      using assms group-action.surj-prop
      unfolding bij-betw-def
      by (simp add: group-action.inj-prop)
    hence bij-betw  $(\text{image } (\varphi x)) (Pow s) (Pow s)$ 
      using bij-betw-Pow
      by metis
    moreover have  $\forall t \in Pow s. \varphi\text{-img } x t = \text{image } (\varphi x) t$ 
      unfolding  $\varphi\text{-img-def}$ 
      by simp
    ultimately have bij-betw  $(\varphi\text{-img } x) (Pow s) (Pow s)$ 
      using bij-betw-cong
      by fastforce
    moreover have  $\varphi\text{-img } x \in \text{extensional } (Pow s)$ 
      unfolding  $\varphi\text{-img-def extensional-def}$ 
      by simp
    ultimately show  $\varphi\text{-img } x \in \text{carrier } (\text{BijGroup } (Pow s))$ 
      unfolding BijGroup-def Bij-def
      by simp
  }

```



```

}
fix
  x :: 'x and
  y :: 'x
note
  car-x-el = ⟨x ∈ carrier m ⟹ φ-img x ∈ carrier (BijGroup (Pow s))⟩ and
  car-y-el = ⟨y ∈ carrier m ⟹ φ-img y ∈ carrier (BijGroup (Pow s))⟩
assume
  car-x: x ∈ carrier m and
  car-y: y ∈ carrier m
  hence car-els: φ-img x ∈ carrier (BijGroup (Pow s)) ∧ φ-img y ∈ carrier
  (BijGroup (Pow s))
  using car-x-el car-y-el car-y
  by blast
  hence h-closed: ∀ t. t ∈ Pow s ⟶ φ-img y t ∈ Pow s
  using bij-betw-apply Int-Collect partial-object.select-convs(1)
  unfolding BijGroup-def Bij-def
  by metis
from car-els
have φ-img x ⊗ BijGroup (Pow s) φ-img y =
  extensional-continuation (φ-img x ∘ φ-img y) (Pow s)
  using rewrite-mult
  by blast
moreover have
  ∀ t. t ∉ Pow s ⟶ extensional-continuation (φ-img x ∘ φ-img y) (Pow s) t =
  undefined
  by simp
moreover have ∀ t. t ∉ Pow s ⟶ φ-img (x ⊗m y) t = undefined
  unfolding φ-img-def
  by simp
moreover have
  ∀ t. t ∈ Pow s ⟶ extensional-continuation (φ-img x ∘ φ-img y) (Pow s) t =
  φ x ‘ φ y ‘ t
  using h-closed
  unfolding φ-img-def
  by simp
moreover have ∀ t. t ∈ Pow s ⟶ φ-img (x ⊗m y) t = φ x ‘ φ y ‘ t
  unfolding φ-img-def extensional-continuation.simps
  using rewrite-group-action-img car-x car-y assms PowD
  by metis
ultimately have ∀ t. φ-img (x ⊗m y) t = (φ-img x ⊗ BijGroup (Pow s) φ-img
y) t
  by metis
thus φ-img (x ⊗m y) = φ-img x ⊗ BijGroup (Pow s) φ-img y
  by blast
qed

```

### 1.8.6 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*  $s$   $t$   $\varphi$   $\psi$ ) **and**  
*action- $\varphi$* : *group-action*  $m$   $t$   $\varphi$  **and**  
*action- $\psi$* : *group-action*  $m$  ( $f$  ‘  $t$ )  $\psi$  **and**  
*gen*: *carrier*  $m$  = *generate*  $m$   $s$

**shows** *is-symmetry*  $f$  (*action-induced-equivariance* (*carrier*  $m$ )  $t$   $\varphi$   $\psi$ )

**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- $\varphi$  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- $\psi$  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- $\varphi$  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 generate.incl group-action.element-image 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-hom.axioms(1) group-action.group-hom local.inv*  
**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 g<sub>1</sub> g<sub>2</sub>*)  
**assume**  
*equivar<sub>1</sub>*:  $\forall x \in t. f (\varphi \ g_1 \ x) = \psi \ g_1 (f \ x)$  **and**  
*equivar<sub>2</sub>*:  $\forall x \in t. f (\varphi \ g_2 \ x) = \psi \ g_2 (f \ x)$  **and**  
*gen<sub>1</sub>*:  $g_1 \in \text{generate } m \ s$  **and**  
*gen<sub>2</sub>*:  $g_2 \in \text{generate } m \ s$   
**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** *equivar<sub>1</sub>*  
**by** *simp*  
**moreover have**  $\forall x \in t. f (\varphi \ g_2 \ x) = \psi \ g_2 (f \ x)$   
**using** *equivar<sub>2</sub>*  
**by** *simp*  
**ultimately show** *?case*  
**using** *action-φ action-ψ gen gen<sub>1</sub> gen<sub>2</sub> group-action.composition-rule imageI*  
**by** (*metis (no-types, lifting)*)

```

qed
thus  $f (\varphi g x) = \psi g (f x)$ 
  using  $x\text{-in-}t$ 
  by  $\text{simp}$ 
qed

```

```

lemma invar-parameterized-fun:
  fixes
     $f :: 'x \Rightarrow ('x \Rightarrow 'y)$  and
     $r :: 'x \text{ rel}$ 
  assumes
     $\text{param-invar: } \forall x. \text{is-symmetry } (f x) \text{ (Invariance } r)$  and
     $\text{invar: is-symmetry } f \text{ (Invariance } r)$ 
  shows  $\text{is-symmetry } (\lambda x. f x x) \text{ (Invariance } r)$ 
  using  $\text{invar param-invar}$ 
  by  $\text{auto}$ 

```

```

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

```

```

lemma equivar-ind-by-act-coincide:
  fixes
     $s :: 'x \text{ set}$  and
     $t :: 'y \text{ set}$  and
     $f :: 'y \Rightarrow 'z$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$  and
     $\varphi' :: ('x, 'y) \text{ binary-fun}$  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  $\text{assms}$ 
  unfolding  $\text{rewrite-equivariance}$ 
  by  $\text{simp}$ 

```

```

lemma equivar-under-subset:
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $s :: 'x \text{ set}$  and
     $t :: 'x \text{ set}$  and
     $\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y)) \text{ set}$ 

```

```

assumes
  is-symmetry f (Equivariance s  $\tau$ ) and
   $t \subseteq s$ 
shows is-symmetry f (Equivariance t  $\tau$ )
using assms
unfolding is-symmetry.simps
by blast

lemma equivar-under-subset':
fixes
   $f :: 'x \Rightarrow 'y$  and
   $s :: 'x \text{ set}$  and
   $\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y)) \text{ set}$  and
   $v :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y)) \text{ set}$ 
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-act-equivar-f-imp-equivar-preimg:
fixes
   $f :: 'x \Rightarrow 'y$  and
   $\mathcal{D}_f :: 'x \text{ set}$  and
   $s :: 'x \text{ set}$  and
   $m :: 'z \text{ monoid}$  and
   $\varphi :: ('z, 'x) \text{ binary-fun}$  and
   $\psi :: ('z, 'y) \text{ binary-fun}$  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 group-hom.axioms(1) action-φ.group-hom group-elem-x
    by metis
  fix
    y :: 'y and
    z :: 'x
  assume preimg-el: z ∈ preimg f Df (ψ x y)
  obtain a :: 'x where
    img: a = φ (inv m x) z
    by simp
  have domain: z ∈ Df ∧ z ∈ s
    using preimg-el dom-in-s
    by auto
  hence a ∈ s
    using dom-in-s action-φ group-elem-inv preimg-el img action-φ.element-image
    by auto
  hence (z, a) ∈ (action-induced-rel (carrier m) s φ) ∩ (Df × s)
    using img preimg-el domain group-elem-inv
    by auto
  hence a ∈ ((action-induced-rel (carrier m) s φ) ∩ (Df × s)) “ Df
    using img preimg-el domain group-elem-inv
    by auto
  hence a-in-domain: a ∈ Df
    using closed-domain
    by auto
  moreover have (φ (inv m x), ψ (inv m x)) ∈ {(φ g, ψ g) | g. g ∈ carrier m}
    using group-elem-inv
    by auto
  ultimately have f a = ψ (inv m x) (f z)
    using domain equivar-f img
    unfolding equivar-prop-def action-induced-equivariance-def
    by simp
  also have f z = ψ x y
    using preimg-el
    by simp
  also have ψ (inv m x) (ψ 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 Df y
    using a-in-domain
    by simp
  moreover have z = φ x a
    using group-hom.axioms(1) action-φ.group-hom action-φ.orbit-sym-aux
    img domain a-in-domain group-elem-x group-elem-inv group.inv-inv
    by metis
  ultimately show z ∈ (φ x) “ (preimg f Df y)
    by simp
next

```

```

fix
   $y :: 'y$  and
   $z :: 'x$ 
assume  $\text{preimg-el}: z \in \text{preimg } f \ \mathcal{D}_f \ y$ 
hence  $\text{domain}: f \ z = y \wedge z \in \mathcal{D}_f \wedge z \in s$ 
  using  $\text{dom-in-}s$ 
  by  $\text{auto}$ 
hence  $\varphi \ x \ z \in s$ 
  using  $\text{group-elem-}x \ \text{group-action.element-image } \text{action-}\varphi$ 
  by  $\text{metis}$ 
hence  $(z, \varphi \ x \ z) \in (\text{action-induced-rel } (\text{carrier } m) \ s \ \varphi) \cap (\mathcal{D}_f \times s) \cap \mathcal{D}_f \times s$ 
  using  $\text{group-elem-}x \ \text{domain}$ 
  by  $\text{auto}$ 
hence  $\varphi \ x \ z \in \mathcal{D}_f$ 
  using  $\text{closed-domain}$ 
  by  $\text{auto}$ 
moreover have  $(\varphi \ x, \psi \ x) \in \{(\varphi \ a, \psi \ a) \mid a. a \in \text{carrier } m\}$ 
  using  $\text{group-elem-}x$ 
  by  $\text{blast}$ 
ultimately show  $\varphi \ x \ z \in \text{preimg } f \ \mathcal{D}_f \ (\psi \ x \ y)$ 
  using  $\text{equivar-}f \ \text{domain}$ 
  unfolding  $\text{equivar-prop-def } \text{action-induced-equivariance-def}$ 
  by  $\text{simp}$ 
qed

```

## Invariance and Equivariance Function Composition

**lemma**  $\text{invar-comp}$ :

```

fixes
   $f :: 'x \Rightarrow 'y$  and
   $g :: 'y \Rightarrow 'z$  and
   $r :: 'x \text{ rel}$ 
assumes  $\text{is-symmetry } f \ (\text{Invariance } r)$ 
shows  $\text{is-symmetry } (g \circ f) \ (\text{Invariance } r)$ 
using  $\text{assms}$ 
by  $\text{simp}$ 

```

**lemma**  $\text{equivar-comp}$ :

```

fixes
   $f :: 'x \Rightarrow 'y$  and
   $g :: 'y \Rightarrow 'z$  and
   $s :: 'x \text{ set}$  and
   $t :: 'y \text{ set}$  and
   $\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y)) \text{ set}$  and
   $v :: (('y \Rightarrow 'y) \times ('z \Rightarrow 'z)) \text{ set}$ 
defines
   $\text{transitive-acts} \equiv$ 
   $\{(\varphi, \psi). \exists \chi :: 'y \Rightarrow 'y. (\varphi, \chi) \in \tau \wedge (\chi, \psi) \in v \wedge \chi \text{ ' } f \text{ ' } s \subseteq t\}$ 
assumes

```

$f \text{ ' } s \subseteq t$  **and**  
 $\text{is-symmetry } f \text{ (Equivariance } s \tau)$  **and**  
 $\text{is-symmetry } g \text{ (Equivariance } t v)$   
**shows**  $\text{is-symmetry } (g \circ f) \text{ (Equivariance } s \text{ transitive-acts})$   
**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**  
 $x\text{-in-}X: x \in s$  **and**  
 $\varphi\text{-}x\text{-in-}X: \varphi x \in s$  **and**  
 $\chi\text{-img}_f\text{-img}_s\text{-in-}t: \chi \text{ ' } f \text{ ' } s \subseteq t$  **and**  
 $\text{act-}f: (\varphi, \chi) \in \tau$  **and**  
 $\text{act-}g: (\chi, \psi) \in v$   
**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  $\varphi\text{-}x\text{-in-}X$*   
**by** *fastforce*  
**finally show**  $g (f (\varphi x)) = \psi (g (f x))$   
**by** *simp*  
**qed**

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

**fixes**  
 $f :: 'x \Rightarrow 'y$  **and**  
 $g :: 'y \Rightarrow 'z$  **and**  
 $s :: 'w \text{ set}$  **and**  
 $t :: 'x \text{ set}$  **and**  
 $u :: 'y \text{ set}$  **and**  
 $\varphi :: ('w, 'x) \text{ binary-fun}$  **and**  
 $\chi :: ('w, 'y) \text{ binary-fun}$  **and**  
 $\psi :: ('w, 'z) \text{ binary-fun}$   
**assumes**  
 $f \text{ ' } t \subseteq u$  **and**  
 $\forall x \in s. \chi x \text{ ' } f \text{ ' } t \subseteq u$  **and**  
 $\text{is-symmetry } f \text{ (action-induced-equivariance } s t \varphi \chi)$  **and**  
 $\text{is-symmetry } g \text{ (action-induced-equivariance } s u \chi \psi)$   
**shows**  $\text{is-symmetry } (g \circ f) \text{ (action-induced-equivariance } s t \varphi \psi)$   
**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**  $\{(\varphi \ a, \psi \ a) \mid a. a \in s\} \subseteq$   
 $\{(\varphi, \psi). \exists v. (\varphi, v) \in ?a_\varphi \wedge (v, \psi) \in ?a_\psi \wedge v \ ' f \ ' t \subseteq u\}$   
**by** *blast*  
**hence** *is-symmetry*  $(g \circ f)$  (*Equivariance*  $t \{(\varphi \ a, \psi \ a) \mid a. a \in s\}$ )  
**using** *assms* *equivar-comp*[*of f t u ?a<sub>φ</sub> g ?a<sub>ψ</sub>*] *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**  
 $f :: 'x \Rightarrow 'y \text{ set}$  **and**  
 $g :: 'x \Rightarrow 'y \text{ set}$  **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*  $\psi$ )) **and**  
*g-equivar*: *is-symmetry*  $g$  (*action-induced-equivariance*  $s \ t \ \varphi$  (*set-action*  $\psi$ )) **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*  $\psi$ ))  
**proof** –  
**have**  $\forall a \in s. \forall x \in t. \varphi \ a \ x \in t \longrightarrow f \ (\varphi \ a \ x) = \psi \ a \ ' (f \ x)$   
**using** *f-equivar*  
**unfolding** *rewrite-equivariance*  
**by** *simp*  
**moreover have**  $\forall a \in s. \forall x \in t. \varphi \ a \ x \in t \longrightarrow g \ (\varphi \ a \ x) = \psi \ a \ ' (g \ x)$   
**using** *g-equivar*  
**unfolding** *rewrite-equivariance*  
**by** *simp*  
**ultimately have**  
 $\forall a \in s. \forall b \in t. \varphi \ a \ b \in t \longrightarrow f \ (\varphi \ a \ b) - g \ (\varphi \ a \ b) = \psi \ a \ ' (f \ b) - \psi \ a \ ' (g \ b)$   
**by** *blast*  
**moreover have**  $\forall a \in s. \forall u \ v. \psi \ a \ ' u - \psi \ a \ ' v = \psi \ a \ ' (u - v)$   
**using** *bij-a image-set-diff*  
**unfolding** *bij-def*  
**by** *blast*  
**ultimately show** *?thesis*  
**unfolding** *set-action.simps*  
**using** *rewrite-equivariance*

```

    by fastforce
qed

lemma equivar-union-under-img-act:
  fixes
    f :: 'x  $\Rightarrow$  'y and
    s :: 'z set and
     $\varphi$  :: ('z, 'x) 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) ' $\cdot$  ts)
    by (blast, blast)
qed

end

```

## 1.9 Symmetry Properties of Voting Rules

```

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

```

### 1.9.1 Definitions

```

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

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

```

## Anonymity

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

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

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

## Neutrality

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

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

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

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

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

**fun**  $\psi\text{-neutr}_c :: ('a \Rightarrow 'a, 'a) \text{ binary-fun } \mathbf{where}$   
 $\psi\text{-neutr}_c \pi r = \pi r$

**fun**  $\psi\text{-neutr}_w :: ('a \Rightarrow 'a, 'a \text{ rel}) \text{ binary-fun } \mathbf{where}$   
 $\psi\text{-neutr}_w \pi r = \text{rel-rename } \pi r$

## Homogeneity

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

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

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

## Reversal Symmetry

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

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

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

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

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

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

## 1.9.2 Auxiliary Lemmas

**fun**  $n\text{-app} :: \text{nat} \Rightarrow ('x \Rightarrow 'x) \Rightarrow ('x \Rightarrow 'x)$  **where**  
 $n\text{-app } 0 f = \text{id} \mid$   
 $n\text{-app} (\text{Suc } n) f = f \circ n\text{-app } n f$

**lemma**  $n\text{-app-rewrite}$ :

**fixes**

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

$n :: \text{nat}$  **and**

$x :: 'x$

**shows**  $(f \circ n\text{-app } n f) x = (n\text{-app } n f \circ f) x$

**proof** ( $\text{unfold comp-def}, \text{induction } n f \text{ arbitrary: } x \text{ rule: } n\text{-app.induct}$ )

**case**  $(1 f)$

**fix**

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

$x :: 'x$

**show**  $f (n\text{-app } 0 f x) = n\text{-app } 0 f (f x)$

**by**  $\text{simp}$

**next**

**case**  $(2 n f)$

```

fix
   $f :: 'x \Rightarrow 'x$  and
   $n :: nat$  and
   $x :: 'x$ 
assume  $\bigwedge y. f (n\text{-app } n f y) = n\text{-app } n f (f y)$ 
thus  $f (n\text{-app } (Suc\ n) f x) = n\text{-app } (Suc\ n) f (f x)$ 
  by simp
qed

lemma n-app-leaves-set:
fixes
   $A :: 'x\ set$  and
   $B :: 'x\ set$  and
   $f :: 'x \Rightarrow 'x$  and
   $x :: 'x$ 
assumes
  fin-A: finite A and
  fin-B: finite B and
  x-el:  $x \in A - B$  and
  bij: 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
  unfolding bij-betw-def
  by force
hence in-AB-imp-in-AB:
   $\forall n > 0. n\text{-app } n f x \in A \cap B \longrightarrow n\text{-app } (Suc\ n) f x \in A \cap B$ 
  using n-app.simps bij

```

```

    unfolding bij-betw-def
  by auto
have in-int:  $\forall n > 0. n\text{-app } n f x \in A \cap B$ 
proof (clarify)
  fix n :: nat
  assume n > 0
  thus n-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-app (Suc n) f x  $\in A \cap B$ 
    using x-el bij in-A in-AB-imp-in-AB
    unfolding bij-betw-def
    by blast
  qed
qed
hence {n-app n f x | n. n > 0}  $\subseteq A \cap B$ 
  by blast
hence finite {n-app n f x | n. n > 0}
  using fin-A fin-B rev-finite-subset
  by blast
moreover have
  inj-on ( $\lambda n. n\text{-app } n f x$ ) {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-app n f x | n. n > 0}
  by auto
ultimately have  $\neg$  inj-on ( $\lambda n. n\text{-app } n f x$ ) {n. n > 0}
  by metis
hence  $\exists n. n > 0 \wedge (\exists m > n. n\text{-app } n f x = n\text{-app } m f x)$ 
  using linorder-inj-onI' mem-Collect-eq
  by metis
hence  $\exists n\text{-min}. 0 < n\text{-min} \wedge (\exists m > n\text{-min}. n\text{-app } n\text{-min } f x = n\text{-app } m f$ 
x)  $\wedge$ 
  ( $\forall n < n\text{-min}. \neg (0 < n \wedge (\exists m > n. n\text{-app } n f x = n\text{-app } m f x))$ )
  using exists-least-iff[of  $\lambda n. n > 0 \wedge (\exists m > n. n\text{-app } n f x = n\text{-app } m f$ 
x)]
  by presburger
then obtain n-min :: nat where
  n-min-pos: n-min > 0 and
   $\exists m > n\text{-min}. n\text{-app } n\text{-min } f x = n\text{-app } m f x$  and

```

```

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

```

```

    by metis
  moreover have  $n\text{-app } n \ f \ x = f \ (n\text{-app } (n - 1) \ f \ x)$ 
    using  $n\text{-app.simps}(2)$   $Suc\text{-pred'}$   $n\text{-pos}$   $comp\text{-eq-id-dest}$   $fun.map\text{-id}$ 
    by (metis (mono-tags, opaque-lifting))
  ultimately show False
    using  $bij\ n\ ex\ not\text{-in-}A\ n\text{-pos}\ less\text{-in-}A$ 
    unfolding  $bij\text{-betw-def}$ 
    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  $bot\text{-nat-}0.\text{not-eq-extremum}$   $less\text{-imp-diff-less}$ 
    by metis
  moreover have  $\forall m > 0. n\text{-app } m \ f \ x = f \ (n\text{-app } (m - 1) \ f \ x)$ 
    using  $bot\text{-nat-}0.\text{not-eq-extremum}$   $comp\text{-apply}$   $diff\text{-Suc-}1\ n\text{-app.elims}$ 
    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  $bij\ n\text{-app.simps}(1)$   $n\text{-app-f-x-in-}A$   $diff\text{-Suc-}1$   $gr0\text{-conv-Suc}$   $imageI$ 
     $linorder\text{-not-le}$   $nless\text{-le}$   $not\text{-less-eq-eq}$ 
    unfolding  $bij\text{-betw-def}$ 
    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  $IntI$   $nless\text{-le}$   $ex\text{-}A$ 
    by metis
  thus ?thesis
    using that
    by blast
qed

```

**lemma**  $n\text{-app-rev}$ :

```

fixes
   $A :: 'x \text{ set}$  and
   $B :: 'x \text{ set}$  and
   $f :: 'x \Rightarrow 'x$  and
   $n :: nat$  and
   $m :: nat$  and
   $x :: 'x$  and
   $y :: 'x$ 
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
   $n\text{-app-x-in-}A: \forall n' < n. n\text{-app } n' \ f \ x \in A$  and
   $n\text{-app-y-in-}A: \forall m' < m. n\text{-app } m' \ f \ y \in A$  and

```



```

    fin-A: finite A and
    fin-B: finite B and
    bij-f-A-B: bij-betw f A B
  shows n-app (n - m) f x = y
  using assms
proof (induction n f arbitrary: m x y rule: n-app.induct)
  case (1 f)
  fix
    f :: 'x ⇒ 'x and
    m :: nat and
    x :: 'x and
    y :: 'x
  assume
    m ≤ 0 and
    n-app 0 f x = n-app m f y
  thus n-app (0 - m) f x = y
    by simp
next
  case (2 n f)
  fix
    f :: 'x ⇒ 'x and
    n :: nat and
    m :: nat and
    x :: 'x and
    y :: 'x
  assume
    bij: bij-betw f A B and
    x-in-A: x ∈ A and
    y-in-A: y ∈ A and
    m-leq-suc-n: m ≤ Suc n and
    x-dom: ∀ n' < Suc n. n-app n' f x ∈ A and
    y-dom: ∀ m' < m. n-app m' f y ∈ A and
    eq: n-app (Suc n) f x = n-app m f y and
    hyp:
      ∧ m x y.
        x ∈ A ⇒
        y ∈ A ⇒
        m ≤ n ⇒
        n-app n f x = n-app m f y ⇒
        ∀ n' < n. n-app n' f x ∈ A ⇒
        ∀ m' < m. n-app m' f y ∈ A ⇒
        finite A ⇒ finite B ⇒ bij-betw f A B ⇒ n-app (n - m) f x = y
  hence m > 0 ⟶ f (n-app n f x) = f (n-app (m - 1) f y)
    using Suc-pred' comp-apply n-app.simps(2)
    by (metis (mono-tags, opaque-lifting))
  moreover have n-app n f x ∈ A
    using x-in-A x-dom
    by blast
  moreover have m > 0 ⟶ n-app (m - 1) f y ∈ A

```

```

    using y-dom
    by simp
  ultimately have  $m > 0 \longrightarrow n\text{-app } n \ f \ x = n\text{-app } (m - 1) \ f \ y$ 
    using bij
    unfolding bij-betw-def inj-on-def
    by blast
  moreover have  $m - 1 \leq n$ 
    using m-leq-suc-n
    by simp
  hence  $m > 0 \longrightarrow n\text{-app } (n - (m - 1)) \ f \ x = y$ 
    using hyp x-in-A y-in-A x-dom y-dom Suc-pred fin-A fin-B
      bij calculation less-SucI
    unfolding One-nat-def
    by metis
  hence  $m > 0 \longrightarrow n\text{-app } (Suc \ n - m) \ f \ x = y$ 
    using Suc-diff-eq-diff-pred
    by presburger
  moreover have  $m = 0 \longrightarrow n\text{-app } (Suc \ n - m) \ f \ x = y$ 
    using eq
    by simp
  ultimately show  $n\text{-app } (Suc \ n - m) \ f \ x = y$ 
    by blast
qed

```

lemma *n-app-inv*:

```

  fixes
     $A :: 'x \text{ set}$  and
     $B :: 'x \text{ set}$  and
     $f :: 'x \Rightarrow 'x$  and
     $n :: \text{nat}$  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-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
       $n :: \text{nat}$  and
       $f :: 'x \Rightarrow 'x$  and
       $x :: 'x$ 
    assume

```

$x\text{-in-}B: x \in B$  **and**  
 $\text{bij}: \text{bij-betw } f A B$  **and**  
 $\text{stays-in-}B: \forall m \geq 0. m < \text{Suc } n \longrightarrow n\text{-app } m (\text{the-inv-into } A f) x \in B$  **and**  
 $\text{hyp}: \bigwedge x. x \in B \implies$   
 $\quad \forall m \geq 0. m < n \longrightarrow n\text{-app } m (\text{the-inv-into } A f) x \in B \implies$   
 $\quad \text{bij-betw } f A B \implies n\text{-app } n f (n\text{-app } n (\text{the-inv-into } A f) x) = x$   
**have**  $n\text{-app } (\text{Suc } n) f (n\text{-app } (\text{Suc } n) (\text{the-inv-into } A f) x) =$   
 $n\text{-app } n f (f (n\text{-app } (\text{Suc } n) (\text{the-inv-into } A f) x))$   
**using**  $n\text{-app-rewrite}$   
**by**  $\text{simp}$   
**also have**  $\dots = n\text{-app } n f (n\text{-app } n (\text{the-inv-into } A f) x)$   
**using**  $\text{stays-in-}B \text{ bij}$   
**by**  $(\text{simp add: } f\text{-the-inv-into-}f\text{-bij-betw})$   
**finally show**  $n\text{-app } (\text{Suc } n) f (n\text{-app } (\text{Suc } n) (\text{the-inv-into } A f) x) = x$   
**using**  $\text{hyp bij stays-in-}B x\text{-in-}B$   
**by**  $\text{simp}$   
**qed**

**lemma**  $\text{bij-betw-finite-ind-global-bij}$ :

**fixes**

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

$B :: 'x \text{ set}$  **and**

$f :: 'x \Rightarrow 'x$

**assumes**

$\text{fin-}A: \text{finite } A$  **and**

$\text{fin-}B: \text{finite } B$  **and**

$\text{bij}: \text{bij-betw } f A B$

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

$\text{bij } g$  **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**

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

**proof** –

**assume**  $\text{existence-witness}$ :

$\bigwedge g. \text{bij } g \implies$

$\quad \forall a \in A. g a = f a \implies$

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

$\quad \forall x \in \text{UNIV} - A - B. g x = x \implies ?thesis$

**have**  $\text{bij-inv}: \text{bij-betw } (\text{the-inv-into } A f) B A$

**using**  $\text{bij bij-betw-the-inv-into}$

**by**  $\text{blast}$

**then obtain**  $g' :: 'x \Rightarrow \text{nat}$  **where**

$\text{greater-0}: \forall x \in B - A. g' x > 0$  **and**

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

$\text{minimal}: \forall x \in B - A. \forall n > 0. n < g' x \longrightarrow n\text{-app } n (\text{the-inv-into } A f) x$

$\in B \cap A$

**using**  $n\text{-app-leaves-set}[\text{of } B A - \text{the-inv-into } A f \text{ False}] \text{ fin-}A \text{ fin-}B$

**by**  $\text{metis}$

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

$\text{def-}g$ :  
 $g = (\lambda x. \text{if } x \in A \text{ then } f x \text{ else } (\text{if } x \in B - A \text{ then } n\text{-app } (g' x) (\text{the-inv-into } A f) x \text{ else } x))$   
**by** *simp*  
**hence** *coincide*:  $\forall a \in A. g a = f a$   
**by** *simp*  
**have** *id*:  $\forall x \in \text{UNIV} - A - B. g x = x$   
**using** *def-g*  
**by** *simp*  
**have**  $\forall x \in B - A. n\text{-app } 0 (\text{the-inv-into } A f) x \in B$   
**by** *simp*  
**moreover** **have**  $\forall x \in B - A. \forall n > 0. n < g' x \longrightarrow n\text{-app } n (\text{the-inv-into } A f) x \in B$   
**using** *minimal*  
**by** *blast*  
**ultimately** **have**  $\forall x \in B - A. n\text{-app } (g' x) f (n\text{-app } (g' x) (\text{the-inv-into } A f) x) = x$   
**using** *n-app-inv bij 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** *greater-0 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**  $\forall x \in \text{UNIV} - A - B. g x = \text{id } x$   
**using** *def-g*  
**by** *simp*  
**hence**  $g' (\text{UNIV} - A - B) = \text{UNIV} - A - B$   
**by** *simp*  
**moreover** **have**  $g' A = B$   
**using** *def-g bij*  
**unfolding** *bij-betw-def*  
**by** *simp*  
**moreover** **have**  $A \cup (\text{UNIV} - A - B) = \text{UNIV} - (B - A) \wedge B \cup (\text{UNIV} - A - B) = \text{UNIV} - (A - B)$   
**by** *blast*  
**ultimately** **have** *surj-cases-13*:  $g' (\text{UNIV} - (B - A)) = \text{UNIV} - (A - B)$   
**using** *image-Un*  
**by** *metis*  
**have** *inj-on*  $g A \wedge \text{inj-on } g (\text{UNIV} - A - B)$   
**using** *def-g bij*  
**unfolding** *bij-betw-def inj-on-def*  
**by** *simp*  
**hence** *inj-cases-13*: *inj-on*  $g (\text{UNIV} - (B - A))$   
**unfolding** *inj-on-def*  
**using** *DiffD2 DiffI bij bij-betwE def-g*  
**by** (*metis (no-types, lifting)*)

```

have card A = card B
  using fin-A fin-B bij bij-betw-same-card
  by blast
with fin-A fin-B
have finite (B - A) ∧ finite (A - B) ∧ card (B - A) = card (A - B)
  using card-le-sym-Diff finite-Diff2 nle-le
  by metis
moreover have (λ x. n-app (g' x) (the-inv-into A f) x) ' (B - A) ⊆ A - B
  using in-set-diff
  by blast
moreover have inj-on (λ x. n-app (g' x) (the-inv-into A f) x) (B - A)
  proof (unfold inj-on-def, safe)
  fix
    x :: 'x and
    y :: 'x
  assume
    x-in-B: x ∈ B and
    x-not-in-A: x ∉ A and
    y-in-B: y ∈ B and
    y-not-in-A: y ∉ A and
    n-app (g' x) (the-inv-into A f) x = n-app (g' y) (the-inv-into A f) y
  moreover from this have
    ∀ n < g' x. n-app n (the-inv-into A f) x ∈ B and
    ∀ n < g' y. n-app n (the-inv-into A f) y ∈ B
  using minimal Diff-iff Int-iff bot-nat-0.not-eq-extremum eq-id-iff n-app.simps(1)
  by (metis, metis)
ultimately have x-to-y:
  n-app (g' x - g' y) (the-inv-into A f) x = y ∨
  n-app (g' y - g' x) (the-inv-into A f) y = x
  using x-in-B y-in-B bij-inv fin-A fin-B
  n-app-rev[of x] n-app-rev[of y B x g' x g' y]
  by fastforce
hence g' x ≠ g' y ⟶
  ((∃ n > 0. n < g' x ∧ n-app n (the-inv-into A f) x ∈ B - A) ∨
  (∃ n > 0. n < g' y ∧ n-app n (the-inv-into A f) y ∈ B - A))
  using greater-0 x-in-B x-not-in-A y-in-B y-not-in-A Diff-iff diff-less-mono2
  diff-zero id-apply less-Suc-eq-0-disj n-app.elims
  by (metis (full-types))
thus x = y
  using minimal x-in-B x-not-in-A y-in-B y-not-in-A x-to-y
  by force
qed
ultimately have bij-betw (λ x. n-app (g' x) (the-inv-into A f) x) (B - A) (A
- B)
  unfolding bij-betw-def
  by (simp add: card-image card-subset-eq)
hence bij-case2: bij-betw g (B - A) (A - B)
  using def-g
  unfolding bij-betw-def inj-on-def

```

```

    by simp
  hence  $g \circ UNIV = UNIV$ 
    using surj-cases-13 Un-Diff-cancel2 image-Un sup-top-left
    unfolding bij-betw-def
    by metis
  moreover have inj  $g$ 
    using inj-cases-13 bij-case2 DiffD2 DiffI imageI surj-cases-13
    unfolding bij-betw-def inj-def inj-on-def
    by metis
  ultimately have bij  $g$ 
    unfolding bij-def
    by safe
  thus ?thesis
    using coincide id reverse existence-witness
    by blast
qed

```

lemma *bij-betw-ext*:

```

  fixes
     $f :: 'x \Rightarrow 'y$  and
     $X :: 'x \text{ set}$  and
     $Y :: 'y \text{ set}$ 
  assumes bij-betw  $f X Y$ 
  shows bij-betw (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

```

### 1.9.3 Anonymity Lemmas

lemma *anon-rel-vote-count*:

```

  fixes
     $\mathcal{E} :: ('a, 'v) \text{ Election set}$  and
     $E :: ('a, 'v) \text{ Election}$  and
     $E' :: ('a, 'v) \text{ Election}$ 
  assumes
    finite (voters- $\mathcal{E}$   $E$ ) and
     $(E, E') \in \text{anonymity}_{\mathcal{R}} \mathcal{E}$ 
  shows alternatives- $\mathcal{E}$   $E = \text{alternatives-}\mathcal{E} E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E}$ 
     $\wedge (\forall p. \text{vote-count } p E = \text{vote-count } p E')$ 
proof -
  have  $E \in \mathcal{E}$ 
    using assms
    unfolding anonymity $_{\mathcal{R}}$ .simps action-induced-rel.simps
    by safe

```

**with** *assms*  
**obtain**  $\pi :: 'v \Rightarrow 'v$  **where**  
    *bijection- $\pi$* : *bij*  $\pi$  **and**  
    *renamed*:  $E' = \text{rename } \pi \ E$   
    **unfolding** *anonymity $\mathcal{R}$ .sims* *anonymity $\mathcal{G}$ -def*  
    **using** *universal-set-carrier-imp-bij-group*  
    **by** *auto*  
**have** *eq-alts*: *alternatives- $\mathcal{E}$*   $E' = \text{alternatives-}\mathcal{E} \ E$   
    **using** *eq-fst-iff* *rename.sims* *alternatives- $\mathcal{E}$ .elims* *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- $\mathcal{E}$ .sims*  
    **using** *renamed* *rename.sims* *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- $\mathcal{E}$ .sims*  
    **using** *renamed* *UNIV-I* *bijection- $\pi$*  *bij-betw-imp-surj* *bij-is-inj* *f-the-inv-into-f*  
    *prod.sel* *inj-image-mem-iff* *prod.collapse* *rename.sims*  
    **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- $\pi$*  *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- $\mathcal{E}$ .sims*  
    **using** *bijection- $\pi$*  *bij-is-inj* *prod.sel* *inj-image-mem-iff* *prod.collapse* *rename.sims*  
    **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- $\pi$*  *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- $\pi$*  *bij-betw-same-card* *bij-betw-subset* *top-greatest*

by (*metis* (*no-types*, *lifting*))  
 ultimately show  
 $\text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E} \wedge (\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E')$   
 using *eq-alts assms*  
 by *simp*  
 qed

**lemma** *vote-count-anon-rel*:

**fixes**

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

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

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

**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**  $\pi :: 'a \text{ Preference-Relation} \Rightarrow ('v \Rightarrow 'v)$  **where**

*bij*:  $\forall p. \text{bij-betw } (\pi \ p) \ \{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\}$

$\{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v = p\}$

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

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

$\pi'\text{-def}$ :  $\forall v \in \text{voters-}\mathcal{E} \ E. \pi' \ v = \pi \ (\text{profile-}\mathcal{E} \ E \ v) \ v$

by *fastforce*

**hence**  $\forall v \ v'. v \in \text{voters-}\mathcal{E} \ E \wedge v' \in \text{voters-}\mathcal{E} \ E \longrightarrow$

$\pi' \ v = \pi' \ v' \longrightarrow \pi \ (\text{profile-}\mathcal{E} \ E \ v) \ v = \pi \ (\text{profile-}\mathcal{E} \ E \ v') \ v'$

by *simp*

**moreover have**

$\forall w \ w'. w \in \text{voters-}\mathcal{E} \ E \wedge w' \in \text{voters-}\mathcal{E} \ E \longrightarrow \pi \ (\text{profile-}\mathcal{E} \ E \ w) \ w = \pi$



$(\text{profile-}\mathcal{E} \ E \ w') \ w' \longrightarrow$   
 $\{v \in \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w\}$   
 $\cap \{v \in \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' \ v = \text{profile-}\mathcal{E} \ E \ w'\} \neq \{\}$   
**using** *bij*  
**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*:  
 $\forall \ v \ v'. \ v \in \text{voters-}\mathcal{E} \ E \wedge v' \in \text{voters-}\mathcal{E} \ E \longrightarrow \pi' \ v = \pi' \ v' \longrightarrow \text{profile-}\mathcal{E} \ E \ v =$   
 $\text{profile-}\mathcal{E} \ E \ v'$   
**by** *presburger*  
**hence**  $\forall \ v \ v'. \ v \in \text{voters-}\mathcal{E} \ E \wedge v' \in \text{voters-}\mathcal{E} \ E \longrightarrow \pi' \ v = \pi' \ v' \longrightarrow$   
 $\pi (\text{profile-}\mathcal{E} \ E \ v) \ v = \pi (\text{profile-}\mathcal{E} \ E \ v) \ v'$   
**using**  $\pi'$ -def  
**by** *metis*  
**hence**  $\forall \ v \ v'. \ v \in \text{voters-}\mathcal{E} \ E \wedge v' \in \text{voters-}\mathcal{E} \ E \longrightarrow \pi' \ v = \pi' \ v' \longrightarrow v = v'$   
**using** *bij eq-prof*  
**unfolding** *bij-betw-def inj-on-def*  
**by** *simp*  
**hence** *inj*: *inj-on*  $\pi' (\text{voters-}\mathcal{E} \ E)$   
**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**  $\pi'$ -def  
**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*  
**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*  $\pi' (\text{voters-}\mathcal{E} \ E) (\text{voters-}\mathcal{E} \ E')$   
**using** *bij*  
**unfolding** *bij-betw-def*  
**by** *blast*  
**then obtain**  $\pi\text{-global} :: 'v \Rightarrow 'v$  **where**  
*bijection- $\pi_g$* : *bij*  $\pi\text{-global}$  **and**  
 $\pi\text{-global-def}$ :  $\forall \ v \in \text{voters-}\mathcal{E} \ E. \ \pi\text{-global} \ v = \pi' \ v$  **and**

$\pi$ -global-def':  
 $\forall v \in \text{voters-}\mathcal{E} \ E' - \text{voters-}\mathcal{E} \ E.$   
 $\pi$ -global  $v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E' \wedge$   
 $(\exists n > 0. n\text{-app } n \ \pi' (\pi\text{-global } v) = v) \text{ and}$   
 $\pi$ -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**  $\pi$ -global-def  $\pi$ -global-non-voters *bij' bijection- $\pi_g$  DiffD1 DiffD2 DiffI*  
*bij-betwE*  
**by** (*metis (no-types, lifting)*)  
**ultimately have**  $\forall v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \ \text{the-inv } \pi\text{-global } v \in \text{voters-}\mathcal{E}$   
 $E' - \text{voters-}\mathcal{E} \ E$   
**using** *bijection- $\pi_g$   $\pi$ -global-def' DiffD2 DiffI UNIV-I*  
**by** *metis*  
**hence**  $\forall v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \ \forall n > 0. \ \text{profile-}\mathcal{E} \ E (\text{the-inv } \pi\text{-global}$   
 $v) = \{\}$   
**using** *default-non-v*  
**by** *simp*  
**moreover have**  $\forall v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' v = \{\}$   
**using** *default-non-v'*  
**by** *simp*  
**ultimately have** *case-1*:  
 $\forall v \in \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' v = (\text{profile-}\mathcal{E} \ E \circ \text{the-inv } \pi\text{-global}$   
 $v)$   
**by** *auto*  
**have**  $\forall v \in \text{voters-}\mathcal{E} \ E'. \ \exists v' \in \text{voters-}\mathcal{E} \ E. \ \pi\text{-global } v' = v \wedge \pi' v' = v$   
**using** *bij' imageE  $\pi$ -global-def*  
**unfolding** *bij-betw-def*  
**by** (*metis (mono-tags, opaque-lifting)*)  
**hence**  $\forall v \in \text{voters-}\mathcal{E} \ E'. \ \exists v' \in \text{voters-}\mathcal{E} \ E. \ v' = \text{the-inv } \pi\text{-global } v \wedge \pi' v' = v$   
**using** *inv*  
**by** *metis*  
**hence**  $\forall v \in \text{voters-}\mathcal{E} \ E'. \ \text{the-inv } \pi\text{-global } v \in \text{voters-}\mathcal{E} \ E \wedge \pi' (\text{the-inv } \pi\text{-global}$   
 $v) = v$   
**by** *blast*  
**moreover have**  $\forall v' \in \text{voters-}\mathcal{E} \ E. \ \text{profile-}\mathcal{E} \ E' (\pi' v') = \text{profile-}\mathcal{E} \ E v'$   
**using**  $\pi'$ -def *bij bij-betwE mem-Collect-eq*  
**by** *fastforce*  
**ultimately have** *case-2*:  $\forall v \in \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' v = (\text{profile-}\mathcal{E} \ E \circ$   
 $\text{the-inv } \pi\text{-global}) v$   
**unfolding** *comp-def*  
**by** *metis*  
**have**  $\forall v \in \text{UNIV} - \text{voters-}\mathcal{E} \ E - \text{voters-}\mathcal{E} \ E'. \ \text{profile-}\mathcal{E} \ E' v = (\text{profile-}\mathcal{E} \ E \circ$

$the-inv \pi-global) v$   
**using**  $\pi-global-non-voters default-non-v default-non-v' inv$   
**by**  $simp$   
**hence**  $profile-\mathcal{E} E' = profile-\mathcal{E} E \circ the-inv \pi-global$   
**using**  $case-1 case-2$   
**by**  $blast$   
**moreover have**  $\pi-global \text{ ' } (voters-\mathcal{E} E) = voters-\mathcal{E} E'$   
**using**  $\pi-global-def bij' bij-betw-imp-surj-on$   
**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**  $eq bijection-\pi_g case-prodI rewrite-carrier$   
**by**  $auto$   
**qed**

**lemma**  $rename-comp$ :

**fixes**  
 $\pi :: 'v \Rightarrow 'v$  **and**  
 $\pi' :: 'v \Rightarrow 'v$   
**assumes**  
 $bij \pi$  **and**  
 $bij \pi'$   
**shows**  $rename \pi \circ rename \pi' = rename (\pi \circ \pi')$   
**proof**  
**fix**  $E :: ('a, 'v) Election$   
**have**  $rename \pi' E = (alternatives-\mathcal{E} E, \pi' \text{ ' } (voters-\mathcal{E} E), (profile-\mathcal{E} E) \circ (the-inv \pi'))$   
**unfolding**  $alternatives-\mathcal{E}.simps voters-\mathcal{E}.simps profile-\mathcal{E}.simps$   
**using**  $prod.collapse rename.simps$   
**by**  $metis$   
**hence**  
 $(rename \pi \circ rename \pi') E =$   
 $rename \pi (alternatives-\mathcal{E} E, \pi' \text{ ' } (voters-\mathcal{E} E), (profile-\mathcal{E} E) \circ (the-inv \pi'))$   
**unfolding**  $comp-def$   
**by**  $presburger$   
**also have**  
 $\dots = (alternatives-\mathcal{E} E, \pi \text{ ' } \pi' \text{ ' } (voters-\mathcal{E} E), (profile-\mathcal{E} E) \circ (the-inv \pi') \circ$   
 $(the-inv \pi))$   
**by**  $simp$   
**also have**  $\dots = (alternatives-\mathcal{E} E, (\pi \circ \pi') \text{ ' } (voters-\mathcal{E} E), (profile-\mathcal{E} E) \circ the-inv$   
 $(\pi \circ \pi'))$   
**using**  $assms the-inv-comp[of \pi UNIV UNIV \pi']$   
**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<sub>G</sub> valid-elections  $\varphi$ -anon valid-elections*  
**proof** (*unfold group-action-def group-hom-def anonymity<sub>G</sub>-def group-hom-axioms-def*  
*hom-def,*  
*safe, (rule group-BijGroup)+)*  
**show** *bij-car-el:*  
 $\bigwedge \pi. \pi \in \text{carrier } (\text{BijGroup UNIV}) \implies$   
 $\varphi\text{-anon valid-elections } \pi \in \text{carrier } (\text{BijGroup valid-elections})$   
**proof** –  
**fix**  $\pi :: 'v \Rightarrow 'v$   
**assume**  $\pi \in \text{carrier } (\text{BijGroup UNIV})$   
**hence** *bij: bij  $\pi$*   
**using** *rewrite-carrier*  
**by** *blast*  
**hence** *rename  $\pi$  ‘ valid-elections = valid-elections*  
**using** *rename-surj bij*  
**by** *blast*  
**moreover have** *inj-on (rename  $\pi$ ) valid-elections*  
**using** *rename-inj bij subset-inj-on*  
**by** *blast*  
**ultimately have** *bij-betw (rename  $\pi$ ) valid-elections valid-elections*  
**unfolding** *bij-betw-def*  
**by** *blast*  
**hence** *bij-betw ( $\varphi$ -anon valid-elections  $\pi$ ) valid-elections valid-elections*  
**unfolding**  *$\varphi$ -anon.simps extensional-continuation.simps*  
**using** *bij-betw-ext*  
**by** *simp*  
**moreover have**  *$\varphi$ -anon valid-elections  $\pi \in \text{extensional valid-elections}$*   
**unfolding** *extensional-def*  
**by** *force*  
**ultimately show**  *$\varphi$ -anon valid-elections  $\pi \in \text{carrier } (\text{BijGroup valid-elections})$*   
**unfolding** *BijGroup-def Bij-def*  
**by** *simp*  
**qed**

**fix**  
 $\pi :: 'v \Rightarrow 'v$  **and**  
 $\pi' :: 'v \Rightarrow 'v$   
**assume**  
*bij:  $\pi \in \text{carrier } (\text{BijGroup UNIV})$  and*  
*bij':  $\pi' \in \text{carrier } (\text{BijGroup UNIV})$*   
**hence** *car-els:  $\varphi$ -anon valid-elections  $\pi \in \text{carrier } (\text{BijGroup valid-elections}) \wedge$*   
 *$\varphi$ -anon valid-elections  $\pi' \in \text{carrier } (\text{BijGroup valid-elections})$*   
**using** *bij-car-el*

by *metis*  
 hence *bij-betw* ( $\varphi$ -anon *valid-elections*  $\pi'$ ) *valid-elections* *valid-elections*  
 unfolding *BijGroup-def* *Bij-def* *extensional-def*  
 by *auto*  
 hence *valid-closed'*:  $\varphi$ -anon *valid-elections*  $\pi' \text{ ' } \text{valid-elections} \subseteq \text{valid-elections}$   
 using *bij-betw-imp-surj-on*  
 by *blast*  
 from *car-els*  
 have  $\varphi$ -anon *valid-elections*  $\pi \otimes \text{BijGroup } \text{valid-elections} (\varphi\text{-anon } \text{valid-elections})$   
 $\pi' =$   
   *extensional-continuation*  
    $(\varphi\text{-anon } \text{valid-elections } \pi \circ \varphi\text{-anon } \text{valid-elections } \pi') \text{ valid-elections}$   
 using *rewrite-mult*  
 by *blast*  
 moreover have  
    $\forall E. E \in \text{valid-elections} \longrightarrow$   
   *extensional-continuation*  
    $(\varphi\text{-anon } \text{valid-elections } \pi \circ \varphi\text{-anon } \text{valid-elections } \pi') \text{ valid-elections } E =$   
    $(\varphi\text{-anon } \text{valid-elections } \pi \circ \varphi\text{-anon } \text{valid-elections } \pi') E$   
 by *simp*  
 moreover have  
    $\forall E. E \in \text{valid-elections} \longrightarrow$   
    $(\varphi\text{-anon } \text{valid-elections } \pi \circ \varphi\text{-anon } \text{valid-elections } \pi') E = \text{rename } \pi$   
 (*rename*  $\pi' E$ )  
 unfolding  $\varphi\text{-anon.simps}$   
 using *valid-closed'*  
 by *auto*  
 moreover have  $\forall E. E \in \text{valid-elections} \longrightarrow \text{rename } \pi (\text{rename } \pi' E) = \text{rename}$   
 ( $\pi \circ \pi'$ )  $E$   
 using *rename-comp* *bij* *bij'* *universal-set-carrier-imp-bij-group* *comp-apply*  
 by *metis*  
 moreover have  
    $\forall E. E \in \text{valid-elections} \longrightarrow$   
    $\text{rename } (\pi \circ \pi') E = \varphi\text{-anon } \text{valid-elections } (\pi \otimes \text{BijGroup } \text{UNIV } \pi') E$   
 using *rewrite-mult-univ* *bij* *bij'*  
 unfolding  $\varphi\text{-anon.simps}$   
 by *force*  
 moreover have  
    $\forall E. E \notin \text{valid-elections} \longrightarrow$   
   *extensional-continuation*  
    $(\varphi\text{-anon } \text{valid-elections } \pi \circ \varphi\text{-anon } \text{valid-elections } \pi') \text{ valid-elections } E =$   
 undefined  
 by *simp*  
 moreover have  
    $\forall E. E \notin \text{valid-elections} \longrightarrow \varphi\text{-anon } \text{valid-elections } (\pi \otimes \text{BijGroup } \text{UNIV } \pi') E$   
 = undefined  
 by *simp*  
 ultimately have  
    $\forall E. \varphi\text{-anon } \text{valid-elections } (\pi \otimes \text{BijGroup } \text{UNIV } \pi') E =$

```

      ( $\varphi$ -anon valid-elections  $\pi \otimes \text{BijGroup valid-elections } \varphi$ -anon valid-elections
 $\pi'$ )  $E$ 
    by metis
  thus
     $\varphi$ -anon valid-elections ( $\pi \otimes \text{BijGroup UNIV } \pi'$ ) =
       $\varphi$ -anon valid-elections  $\pi \otimes \text{BijGroup valid-elections } \varphi$ -anon valid-elections  $\pi'$ 
    by blast
qed

```

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

#### 1.9.4 Neutrality Lemmas

**lemma** *rel-rename-helper:*  
 fixes  
    $r :: 'a \text{ rel}$  and  
    $\pi :: 'a \Rightarrow 'a$  and  
    $a :: 'a$  and  
    $b :: 'a$   
 assumes *bij*  $\pi$   
 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  
    $x :: 'a$  and  
    $y :: 'a$   
 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  
    $x :: 'a$  and  
    $y :: 'a$   
 assume  $(a, b) \in r$   
 thus  $\exists x y. (\pi a, \pi b) = (\pi x, \pi y) \wedge (x, y) \in r$   
   by metis  
qed

**lemma** *rel-rename-comp:*  
 fixes  
    $\pi :: 'a \Rightarrow 'a$  and

```

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

```

lemma *rel-rename-sound*:

```

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

```

by *auto*  
 hence  $c = c' \wedge d = d'$   
 using *assms*  
 unfolding *inj-def*  
 by *presburger*  
 moreover assume  $\forall a b. (a, b) \in r \longrightarrow (b, a) \in r \longrightarrow a = b$   
 ultimately have  $c = d$   
 using *d'-rel-c' c-rel-d*  
 by *simp*  
 thus  $a = b$   
 using  *$\pi_c$ -eq-a  $\pi_d$ -eq-b*  
 by *simp*  
 next  
 fix  
 $a :: 'a$  and  
 $b :: 'a$   
 assume  
*total*:  $\forall x \in A. \forall y \in A. x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r$  and  
*a-in-A*:  $a \in A$  and  
*b-in-A*:  $b \in A$  and  
 *$\pi_a$ -neq- $\pi_b$* :  $\pi a \neq \pi b$  and  
 *$\pi_b$ -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-rewrite.simps*  
 by *blast*  
 hence  $(a, b) \in r$   
 using *a-in-A b-in-A total*  
 by *blast*  
 thus  $(\pi a, \pi b) \in \text{rel-rewrite } \pi r$   
 unfolding *rel-rewrite.simps*  
 by *blast*  
 next  
 fix  
 $a :: 'a$  and  
 $b :: 'a$  and  
 $c :: 'a$   
 assume  
 $(a, b) \in \text{rel-rewrite } \pi r$  and  
 $(b, c) \in \text{rel-rewrite } \pi r$   
 then obtain  
 $d :: 'a$  and  
 $e :: 'a$  and  
 $s :: 'a$  and  
 $t :: 'a$  where  
*d-rel-e*:  $(d, e) \in r$  and  
*s-rel-t*:  $(s, t) \in r$  and  
 *$\pi_d$ -eq-a*:  $\pi d = a$  and  
 *$\pi_s$ -eq-b*:  $\pi s = b$  and  
 *$\pi_t$ -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 rel-rename-bij:
  fixes  $\pi :: 'a \Rightarrow 'a$ 
  assumes bij- $\pi$ :  $\text{bij } \pi$ 
  shows  $\text{bij } (\text{rel-rename } \pi)$ 
proof (unfold bij-def inj-def surj-def, safe)
  show subset:
     $\bigwedge\ r\ s\ a\ b. \text{rel-rename } \pi\ r = \text{rel-rename } \pi\ s \implies (a, b) \in r \implies (a, b) \in s$ 
  proof -
    fix
       $r :: 'a\ \text{rel}$  and
       $s :: 'a\ \text{rel}$  and
       $a :: 'a$  and
       $b :: 'a$ 
    assume
       $\text{rel-rename } \pi\ r = \text{rel-rename } \pi\ s$  and
       $(a, b) \in r$ 
    hence  $(\pi\ a, \pi\ b) \in \{(\pi\ a, \pi\ b) \mid a\ b. (a, b) \in s\}$ 
    unfolding rel-rename.simps
    by blast
    hence  $\exists\ c\ d. (c, d) \in s \wedge \pi\ c = \pi\ a \wedge \pi\ d = \pi\ b$ 
    by fastforce
    moreover have  $\forall\ c\ d. \pi\ c = \pi\ d \longrightarrow c = d$ 
    using bij- $\pi$  bij-pointE
    by metis
    ultimately show  $(a, b) \in s$ 
    by blast
  qed
fix
   $r :: 'a\ \text{rel}$  and
   $s :: 'a\ \text{rel}$  and

```

```

    a :: 'a and
    b :: 'a
  assume
    rel-rename  $\pi$  r = rel-rename  $\pi$  s and
    (a, b)  $\in$  s
  thus (a, b)  $\in$  r
    using subset
    by presburger
next
  fix r :: 'a rel
  have rel-rename  $\pi$  {((the-inv  $\pi$ ) a, (the-inv  $\pi$ ) b) | a b. (a, b)  $\in$  r} =
    {( $\pi$  ((the-inv  $\pi$ ) a),  $\pi$  ((the-inv  $\pi$ ) b)) | a b. (a, b)  $\in$  r}
    by auto
  also have ... = {(a, b) | a b. (a, b)  $\in$  r}
    using the-inv-f-f bij- $\pi$ 
    by (simp add: f-the-inv-into-f-bij-betw)
  finally have rel-rename  $\pi$  (rel-rename (the-inv  $\pi$ ) r) = r
    by simp
  thus  $\exists$  s. r = rel-rename  $\pi$  s
    by blast
qed

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

lemma alternatives-rename-bij:
  fixes  $\pi$  :: ('a  $\Rightarrow$  'a)
  assumes bij- $\pi$ : bij  $\pi$ 

```

**shows** *bij-betw* (*alternatives-rename*  $\pi$ ) *valid-elections* *valid-elections*  
**proof** (*unfold bij-betw-def*, *safe*, *intro inj-onI*, *clarsimp*)  
**fix**  
 $A :: 'a \text{ set}$  **and**  
 $A' :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$   
**assume**  
 $\pi\text{-eq-img-}A\text{-}A': \pi \text{ ' } A = \pi \text{ ' } A'$  **and**  
 $\text{rel-rename-eq: rel-rename } \pi \circ p = \text{rel-rename } \pi \circ p'$   
**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*  
**thus**  $A = A' \wedge p = p'$   
**using** *bij- $\pi$   $\pi$ -eq-img- $A$ - $A'$  bij-betw-imp-inj-on inj-image-eq-iff*  
**by** *metis*  
**next**  
**fix**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**assume** *valid-elects:  $(A, V, p) \in \text{valid-elections}$*   
**have** *valid-elects-closed:*  
 $\bigwedge A' V' p'. \pi.$   
 $\text{bij } \pi \implies (A', V', p') = \text{alternatives-rename } \pi (A, V, p) \implies$   
 $(A', V', p') \in \text{valid-elections}$   
**proof** –  
**fix**  
 $A' :: 'a \text{ set}$  **and**  
 $V' :: 'v \text{ set}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$  **and**  
 $\pi :: 'a \Rightarrow 'a$   
**assume** *renamed:  $(A', V', p') = \text{alternatives-rename } \pi (A, V, p)$*   
**hence** *rewr:  $V = V' \wedge A' = \pi \text{ ' } A$*   
**by** *simp*  
**hence**  $\forall v \in V'. \text{linear-order-on } A (p \ v)$   
**using** *valid-elects*  
**unfolding** *valid-elections-def profile-def*  
**by** *simp*  
**moreover have**  $\forall v \in V'. p' \ v = \text{rel-rename } \pi (p \ v)$

```

    using renamed
    by simp
  moreover assume  $\text{bij-}\pi$ :  $\text{bij } \pi$ 
  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
    by metis
  thus  $(A', V', p') \in \text{valid-elections}$ 
    unfolding valid-elections-def profile-def
    by simp
qed
thus  $\bigwedge A' V' p'. (A', V', p') = \text{alternatives-rename } \pi (A, V, p) \implies$ 
   $(A, V, p) \in \text{valid-elections} \implies (A', V', p') \in \text{valid-elections}$ 
  using  $\text{bij-}\pi$  valid-elects
  by blast
have  $\text{alternatives-rename } (\text{the-inv } \pi) (A, V, p)$ 
   $= ((\text{the-inv } \pi) ' A, V, \text{rel-rename } (\text{the-inv } \pi) \circ p)$ 
  by simp
also have
   $\text{alternatives-rename } \pi ((\text{the-inv } \pi) ' A, V, \text{rel-rename } (\text{the-inv } \pi) \circ p) =$ 
   $(\pi ' (\text{the-inv } \pi) ' A, V, \text{rel-rename } \pi \circ \text{rel-rename } (\text{the-inv } \pi) \circ p)$ 
  by auto
also have  $\dots = (A, V, \text{rel-rename } (\pi \circ \text{the-inv } \pi) \circ p)$ 
  using  $\text{bij-}\pi$  rel-rename-comp[of  $\pi$ ] the-inv-f-f
  by (simp add: bij-betw-imp-surj-on bij-is-inj f-the-inv-into-f image-comp)
also have  $(A, V, \text{rel-rename } (\pi \circ \text{the-inv } \pi) \circ p) = (A, V, \text{rel-rename id } \circ p)$ 
  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  $\text{alternatives-rename } (\text{the-inv } \pi) (A, V, p) \in \text{valid-elections}$ 
  using valid-elects-closed  $\text{bij-}\pi$ 
  by (simp add: bij-betw-the-inv-into valid-elects)
ultimately show  $(A, V, p) \in \text{alternatives-rename } \pi ' \text{valid-elections}$ 
  using image-eqI
  by metis
qed

interpretation  $\varphi$ -neutr-act:
  group-action neutralityG valid-elections  $\varphi$ -neutr valid-elections
proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def neu-
  tralityG-def,
    safe, (rule group-BijGroup)+)
show  $\text{bij-car-el}$ :
   $\bigwedge \pi. \pi \in \text{carrier } (\text{BijGroup UNIV}) \implies$ 
   $\varphi\text{-neutr valid-elections } \pi \in \text{carrier } (\text{BijGroup valid-elections})$ 

```

```

proof –
  fix  $\pi :: 'c \Rightarrow 'c$ 
  assume  $\pi \in \text{carrier } (\text{BijGroup UNIV})$ 
  hence  $\text{bij-betw } (\varphi\text{-neutr valid-elections } \pi) \text{ valid-elections valid-elections}$ 
  using universal-set-carrier-imp-bij-group
  unfolding  $\varphi\text{-neutr.simps}$ 
  using alternatives-rename-bij bij-betw-ext
  by metis
  thus  $\varphi\text{-neutr valid-elections } \pi \in \text{carrier } (\text{BijGroup valid-elections})$ 
  unfolding  $\varphi\text{-neutr.simps BijGroup-def Bij-def extensional-def}$ 
  by simp
qed
fix
   $\pi :: 'a \Rightarrow 'a$  and
   $\pi' :: 'a \Rightarrow 'a$ 
  assume
     $\text{bij: } \pi \in \text{carrier } (\text{BijGroup UNIV})$  and
     $\text{bij': } \pi' \in \text{carrier } (\text{BijGroup UNIV})$ 
  hence  $\text{car-els: } \varphi\text{-neutr valid-elections } \pi \in \text{carrier } (\text{BijGroup valid-elections}) \wedge$ 
     $\varphi\text{-neutr valid-elections } \pi' \in \text{carrier } (\text{BijGroup valid-elections})$ 
  using bij-car-el
  by metis
  hence  $\text{bij-betw } (\varphi\text{-neutr valid-elections } \pi') \text{ valid-elections valid-elections}$ 
  unfolding BijGroup-def Bij-def extensional-def
  by auto
  hence  $\text{valid-closed': } \varphi\text{-neutr valid-elections } \pi' \text{ ' valid-elections } \subseteq \text{ valid-elections}$ 
  using bij-betw-imp-surj-on
  by blast
  have  $\varphi\text{-neutr valid-elections } \pi \otimes \text{BijGroup valid-elections } \varphi\text{-neutr valid-elections}$ 
 $\pi' =$ 
    extensional-continuation
     $(\varphi\text{-neutr valid-elections } \pi \circ \varphi\text{-neutr valid-elections } \pi') \text{ valid-elections}$ 
  using car-els rewrite-mult
  by auto
  moreover have
     $\forall \mathcal{E}. \mathcal{E} \in \text{valid-elections} \longrightarrow$ 
    extensional-continuation
     $(\varphi\text{-neutr valid-elections } \pi \circ \varphi\text{-neutr valid-elections } \pi') \text{ valid-elections } \mathcal{E} =$ 
     $(\varphi\text{-neutr valid-elections } \pi \circ \varphi\text{-neutr valid-elections } \pi') \mathcal{E}$ 
  by simp
  moreover have
     $\forall \mathcal{E}. \mathcal{E} \in \text{valid-elections} \longrightarrow$ 
     $(\varphi\text{-neutr valid-elections } \pi \circ \varphi\text{-neutr valid-elections } \pi') \mathcal{E} =$ 
    alternatives-rename  $\pi$  (alternatives-rename  $\pi'$   $\mathcal{E}$ )
  unfolding  $\varphi\text{-neutr.simps}$ 
  using valid-closed'
  by auto
  moreover have
     $\forall \mathcal{E}. \mathcal{E} \in \text{valid-elections}$ 

```

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

**interpretation**  $\psi\text{-neutr}_c\text{-act}$ : *group-action neutrality<sub>G</sub> UNIV  $\psi\text{-neutr}_c$*   
**proof** (*unfold group-action-def group-hom-def hom-def neutrality<sub>G</sub>-def group-hom-axioms-def*,

$\text{safe, (rule group-BijGroup)+)$   
**fix**  $\pi :: 'a \Rightarrow 'a$   
**assume**  $\pi \in \text{carrier (BijGroup UNIV)}$   
**hence** *bij  $\pi$*   
**unfolding** *BijGroup-def Bij-def*  
**by** *simp*  
**thus**  $\psi\text{-neutr}_c \pi \in \text{carrier (BijGroup UNIV)}$   
**unfolding**  *$\psi\text{-neutr}_c.\text{simps}$*   
**using** *rewrite-carrier*  
**by** *blast*  
**next**  
**fix**  
 $\pi :: 'a \Rightarrow 'a$  **and**  
 $\pi' :: 'a \Rightarrow 'a$   
**show**  $\psi\text{-neutr}_c (\pi \otimes \text{BijGroup UNIV } \pi') =$

$\psi\text{-neutr}_c \pi \otimes \text{BijGroup UNIV } \psi\text{-neutr}_c \pi'$   
**unfolding**  $\psi\text{-neutr}_c.\text{simps}$   
**by** *simp*  
**qed**

**interpretation**  $\psi\text{-neutr}_w\text{-act}$ : *group-action neutrality<sub>G</sub> UNIV  $\psi\text{-neutr}_w$*   
**proof** (*unfold group-action-def group-hom-def hom-def neutrality<sub>G</sub>-def group-hom-axioms-def*,

*safe, (rule group-BijGroup)+)*  
**show** *group-elem*:  
 $\bigwedge \pi. \pi \in \text{carrier (BijGroup UNIV)} \implies \psi\text{-neutr}_w \pi \in \text{carrier (BijGroup UNIV)}$   
**proof** –  
**fix**  $\pi :: 'c \Rightarrow 'c$   
**assume**  $\pi \in \text{carrier (BijGroup UNIV)}$   
**hence** *bij*  $\pi$   
**unfolding** *neutrality<sub>G</sub>-def BijGroup-def Bij-def*  
**by** *simp*  
**hence** *bij*  $(\psi\text{-neutr}_w \pi)$   
**unfolding** *neutrality<sub>G</sub>-def BijGroup-def Bij-def  $\psi\text{-neutr}_w.\text{simps}$*   
**using** *rel-rename-bij*  
**by** *blast*  
**thus**  $\psi\text{-neutr}_w \pi \in \text{carrier (BijGroup UNIV)}$   
**using** *rewrite-carrier*  
**by** *blast*  
**qed**

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

**lemma** *wf-result-neutrality-SCF*:  
*is-symmetry*  $(\lambda \mathcal{E}. \text{limit-set-SCF (alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV})$   
*(action-induced-equivariance (carrier neutrality<sub>G</sub>) valid-elections*  
*( $\varphi\text{-neutr valid-elections}$ ) (set-action  $\psi\text{-neutr}_c$ ))*  
**proof** (*unfold rewrite-equivariance, safe*)  
**fix**

$\pi :: 'a \Rightarrow 'a$  **and**  
 $A :: 'a$  *set* **and**  
 $V :: 'v$  *set* **and**  
 $p :: 'v \Rightarrow ('a \times 'a)$  *set* **and**  
 $r :: 'a$   
**assume**  
*carrier- $\pi$* :  $\pi \in$  *carrier neutrality<sub>G</sub>* **and**  
*prof*:  $(A, V, p) \in$  *valid-elections* **and**  
*neutr-valid-el*:  $\varphi$ -*neutr valid-elections*  $\pi (A, V, p) \in$  *valid-elections*  
{  
  **assume**  $r \in$  *limit-set-SCF* (*alternatives- $\mathcal{E}$*  ( $\varphi$ -*neutr valid-elections*  $\pi (A, V, p)$ )) *UNIV*  
  **thus**  $r \in$  *set-action*  $\psi$ -*neutr<sub>c</sub>*  $\pi$  (*limit-set-SCF* (*alternatives- $\mathcal{E}$*  ( $A, V, p$ )) *UNIV*)  
  **using** *carrier- $\pi$*  *prof* *neutr-valid-el*  
  **by** *auto*  
}  
{  
  **assume**  $r \in$  *set-action*  $\psi$ -*neutr<sub>c</sub>*  $\pi$  (*limit-set-SCF* (*alternatives- $\mathcal{E}$*  ( $A, V, p$ )) *UNIV*)  
  **thus**  $r \in$  *limit-set-SCF* (*alternatives- $\mathcal{E}$*  ( $\varphi$ -*neutr valid-elections*  $\pi (A, V, p)$ )) *UNIV*  
  **using** *carrier- $\pi$*  *prof* *neutr-valid-el*  
  **by** *simp*  
}  
**qed**

**lemma** *wf-result-neutrality-SWF*:

*is-symmetry* ( $\lambda \mathcal{E}. \text{limit-set-SWF} (\text{alternatives-}\mathcal{E} \ \mathcal{E}) \ \text{UNIV}$ )  
*(action-induced-equivariance (carrier neutrality<sub>G</sub>) valid-elections*  
*( $\varphi$ -neutr valid-elections) (set-action  $\psi$ -neutr<sub>w</sub>))*

**proof** (*unfold rewrite-equivariance voters- $\mathcal{E}$ .simps profile- $\mathcal{E}$ .simps set-action.simps, safe*)

**show** *lim-el- $\pi$* :

$\bigwedge \pi A V p r. \pi \in$  *carrier neutrality<sub>G</sub>*  $\implies (A, V, p) \in$  *valid-elections*  $\implies$   
 $\varphi$ -*neutr valid-elections*  $\pi (A, V, p) \in$  *valid-elections*  $\implies$   
 $r \in$  *limit-set-SWF* (*alternatives- $\mathcal{E}$*  ( $\varphi$ -*neutr valid-elections*  $\pi (A, V, p)$ ))

*UNIV*  $\implies$

$r \in \psi$ -*neutr<sub>w</sub>*  $\pi$  ‘ *limit-set-SWF* (*alternatives- $\mathcal{E}$*  ( $A, V, p$ )) *UNIV*

**proof** –

**fix**

$\pi :: 'c \Rightarrow 'c$  **and**  
 $A :: 'c$  *set* **and**  
 $V :: 'v$  *set* **and**  
 $p :: ('c, 'v)$  *Profile* **and**  
 $r :: 'c$  *rel*

**let** *?r-inv* =  $\psi$ -*neutr<sub>w</sub>* (*the-inv*  $\pi$ ) *r*

**assume**

*carrier- $\pi$* :  $\pi \in$  *carrier neutrality<sub>G</sub>* **and**  
*prof*:  $(A, V, p) \in$  *valid-elections* **and**



$\varphi$ -neutr valid-elections  $\pi (A, V, p) \in \text{valid-elections}$  **and**  
 $\text{lim-el: } r \in \text{limit-set-SWF (alternatives-}\mathcal{E} \text{ (}\varphi\text{-neutr valid-elections } \pi (A, V,$   
 $p))) \text{ UNIV}$   
**hence** *inv-carrier*: *the-inv*  $\pi \in \text{carrier neutrality}_{\mathcal{G}}$   
**unfolding** *neutrality<sub>G</sub>-def* *rewrite-carrier*  
**using** *bij-betw-the-inv-into*  
**by** *simp*  
**moreover have** *the-inv*  $\pi \circ \pi = \text{id}$   
**using** *carrier- $\pi$  universal-set-carrier-imp-bij-group bij-is-inj the-inv-f-f*  
**unfolding** *neutrality<sub>G</sub>-def*  
**by** *fastforce*  
**moreover have**  $\mathbf{1}_{\text{neutrality}_{\mathcal{G}}} = \text{id}$   
**unfolding** *neutrality<sub>G</sub>-def* *BijGroup-def*  
**by** *auto*  
**ultimately have** *the-inv*  $\pi \otimes \text{neutrality}_{\mathcal{G}} \pi = \mathbf{1}_{\text{neutrality}_{\mathcal{G}}}$   
**using** *carrier- $\pi$*   
**unfolding** *neutrality<sub>G</sub>-def*  
**using** *rewrite-mult-univ*  
**by** *metis*  
**hence** *inv-eq*: *inv* *neutrality<sub>G</sub>*  $\pi = \text{the-inv } \pi$   
**using** *carrier- $\pi$  inv-carrier  $\psi$ -neutr<sub>c</sub>.act.group-hom group.inv-closed group.inv-solve-right*  
*group.l-inv group-BijGroup group-hom.hom-one group-hom.one-closed*  
**unfolding** *neutrality<sub>G</sub>-def*  
**by** *metis*  
**have**  $r \in \text{limit-set-SWF } (\pi \text{ ' } A) \text{ UNIV}$   
**unfolding**  *$\varphi$ -neutr.simps*  
**using** *prof lim-el*  
**by** *simp*  
**hence** *lin*: *linear-order-on*  $(\pi \text{ ' } A) \text{ } r$   
**by** *auto*  
**have** *bij-inv*: *bij*  $(\text{the-inv } \pi)$   
**using** *carrier- $\pi$  bij-betw-the-inv-into universal-set-carrier-imp-bij-group*  
**unfolding** *neutrality<sub>G</sub>-def*  
**by** *blast*  
**hence**  $(\text{the-inv } \pi) \text{ ' } \pi \text{ ' } A = A$   
**using** *carrier- $\pi$  UNIV-I bij-betw-imp-surj universal-set-carrier-imp-bij-group*  
*f-the-inv-into-f-bij-betw image-f-inv-f surj-imp-inv-eq*  
**unfolding** *neutrality<sub>G</sub>-def*  
**by** *metis*  
**hence** *lin-inv*: *linear-order-on*  $A \text{ } ?r\text{-inv}$   
**using** *rel-rename-sound bij-inv lin bij-is-inj*  
**unfolding**  *$\psi$ -neutr<sub>w</sub>.simps linear-order-on-def preorder-on-def partial-order-on-def*  
**by** *metis*  
**hence**  $\forall a \ b. (a, b) \in ?r\text{-inv} \longrightarrow a \in A \wedge b \in A$   
**using** *linear-order-on-def partial-order-onD(1) refl-on-def*  
**by** *blast*  
**hence** *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 ... = *limit A ?r-inv*  
 by *blast*  
 hence  $?r\text{-inv} \in \text{limit-set-SWF } (\text{alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV}$   
 unfolding *limit-set-SWF.simps*  
 using *lin-inv UNIV-I fst-conv mem-Collect-eq alternatives-}\mathcal{E}.elim*  
     *iso-tuple-UNIV-I CollectI*  
 by (*metis (mono-tags, lifting)*)  
 moreover have  $r = \psi\text{-neutr}_w \pi \text{ ?r-inv}$   
 using *carrier-}\pi \text{ inv-eq inv-carrier iso-tuple-UNIV-I } \psi\text{-neutr}\_w\text{-act.orbit-sym-aux}*  
 by *metis*  
 ultimately show  $r \in \psi\text{-neutr}_w \pi \text{ ' limit-set-SWF } (\text{alternatives-}\mathcal{E} (A, V, p))$   
 UNIV  
     by *blast*  
 qed  
 fix  
      $\pi :: 'a \Rightarrow 'a$  and  
      $A :: 'a \text{ set}$  and  
      $V :: 'v \text{ set}$  and  
      $p :: ('a, 'v) \text{ Profile}$  and  
      $r :: 'a \text{ rel}$   
 let  $?r\text{-inv} = \psi\text{-neutr}_w (\text{the-inv } \pi) r$   
 assume  
     *carrier-}\pi: \pi \in \text{carrier neutrality}\_G* and  
     *prof: (A, V, p) \in \text{valid-elections}* and  
     *prof-}\pi: \varphi\text{-neutr valid-elections } \pi (A, V, p) \in \text{valid-elections}* and  
      $r \in \text{limit-set-SWF } (\text{alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV}$   
 hence  
      $r \in \text{limit-set-SWF } (\text{alternatives-}\mathcal{E} (\varphi\text{-neutr valid-elections } (\text{inv neutrality}_G \pi)$   
          $(\varphi\text{-neutr valid-elections } \pi (A, V, p)))) \text{ UNIV}$   
     using *\varphi-neutr-act.orbit-sym-aux*  
     by *metis*  
 moreover have *inv-group-elem: inv neutrality}\_G \pi \in \text{carrier neutrality}\_G*  
     using *carrier-}\pi \psi\text{-neutr}\_c\text{-act.group-hom}*  
         *group.inv-closed group-hom-def*  
     by *metis*  
 moreover have  
      $\varphi\text{-neutr valid-elections } (\text{inv neutrality}_G \pi)$   
      $(\varphi\text{-neutr valid-elections } \pi (A, V, p)) \in \text{valid-elections}$   
     using *prof \varphi-neutr-act.element-image inv-group-elem prof-}\pi*  
     by *metis*  
 ultimately have  
      $r \in \psi\text{-neutr}_w (\text{inv neutrality}_G \pi) \text{ '}$   
          $\text{limit-set-SWF } (\text{alternatives-}\mathcal{E} (\varphi\text{-neutr valid-elections } \pi (A, V, p)))$   
 UNIV  
     using *prof-}\pi \text{ lim-el-}\pi \text{ prod.collapse}*  
     by *metis*  
 thus  
      $\psi\text{-neutr}_w \pi r \in \text{limit-set-SWF } (\text{alternatives-}\mathcal{E} (\varphi\text{-neutr valid-elections } \pi (A,$

```

V, p))) UNIV
  using carrier- $\pi$   $\psi$ -neutrw-act.group-action-axioms
         $\psi$ -neutrw-act.inj-prop group-action.orbit-sym-aux
        inj-image-mem-iff inv-group-elem iso-tuple-UNIV-I
  by (metis (no-types, lifting))
qed

```

### 1.9.5 Homogeneity Lemmas

```

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

```

```

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

```

```

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

```

```

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

```

### 1.9.6 Reversal Symmetry Lemmas

```

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

```

```

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

```

```

lemma rev-rel-lin-ord:
  fixes

```

```

    A :: 'a set and
    r :: 'a rel
  assumes linear-order-on A r
  shows linear-order-on A (rev-rel r)
  using assms
  unfolding rev-rel.simps linear-order-on-def partial-order-on-def
    total-on-def antisym-def preorder-on-def refl-on-def trans-def
  by blast

interpretation reversalG-group: group reversalG
proof
  show 1 reversalG ∈ carrier reversalG
    unfolding reversalG-def
    by simp
next
  show carrier reversalG ⊆ Units reversalG
    unfolding reversalG-def Units-def
    using rev-rev-id
    by auto
next
  fix α :: 'a rel ⇒ 'a rel
  show α ⊗ reversalG 1 reversalG = α
    unfolding reversalG-def
    by auto
  assume α-elem: α ∈ carrier reversalG
  thus 1 reversalG ⊗ reversalG α = α
    unfolding reversalG-def
    by auto
  fix α' :: 'a rel ⇒ 'a rel
  assume α'-elem: α' ∈ carrier reversalG
  thus α ⊗ reversalG α' ∈ carrier reversalG
    using α-elem rev-rev-id
    unfolding reversalG-def
    by auto
  fix z :: 'a rel ⇒ 'a rel
  assume z ∈ carrier reversalG
  thus α ⊗ reversalG α' ⊗ reversalG z = α ⊗ reversalG (α' ⊗ reversalG z)
    using α-elem α'-elem
    unfolding reversalG-def
    by auto
qed

interpretation φ-rev-act: group-action reversalG valid-elections φ-rev valid-elections
proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def,
  safe, rule group-BijGroup)
  show car-el:
    ∧ π. π ∈ carrier reversalG ⇒ φ-rev valid-elections π ∈ carrier (BijGroup
  valid-elections)
  proof –

```

```

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

```

```

unfolding BijGroup-def Bij-def bij-betw-def
by auto
hence extensional-continuation
  ( $\varphi$ -rev valid-elections  $\pi \circ \varphi$ -rev valid-elections  $\pi'$ ) valid-elections =
  extensional-continuation (rel-app  $\pi \circ \text{rel-app } \pi'$ ) valid-elections
unfolding extensional-continuation.simps  $\varphi$ -rev.simps
by fastforce
also have
  extensional-continuation ( $\varphi$ -rev valid-elections  $\pi \circ \varphi$ -rev valid-elections  $\pi'$ )
valid-elections
  =  $\varphi$ -rev valid-elections  $\pi \otimes \text{BijGroup } \text{valid-elections } \varphi$ -rev valid-elections  $\pi'$ 
using car-el rewrite-mult rev rev'
by metis
finally show
   $\varphi$ -rev valid-elections ( $\pi \otimes \text{reversal}_G \pi'$ ) =
   $\varphi$ -rev valid-elections  $\pi \otimes \text{BijGroup } \text{valid-elections } \varphi$ -rev valid-elections  $\pi'$ 
using rewrite
by metis
qed

interpretation  $\psi$ -rev-act: group-action reversalG UNIV  $\psi$ -rev
proof (unfold group-action-def group-hom-def group-hom-axioms-def hom-def  $\psi$ -rev.simps,
  safe, rule group-BijGroup)
fix  $\pi :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$ 
show bij:  $\bigwedge \pi. \pi \in \text{carrier reversal}_G \implies \pi \in \text{carrier (BijGroup UNIV)}$ 
proof –
  fix  $\pi :: 'b \text{ rel} \Rightarrow 'b \text{ rel}$ 
assume  $\pi \in \text{carrier reversal}_G$ 
hence  $\pi \in \{id, \text{rev-rel}\}$ 
unfolding reversalG-def
by auto
hence bij  $\pi$ 
using rev-rev-id bij-id insertE o-bij singleton-iff
by metis
thus  $\pi \in \text{carrier (BijGroup UNIV)}$ 
using rewrite-carrier
by blast
qed
fix
   $\pi :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$  and
   $\pi' :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$ 
assume
  rev:  $\pi \in \text{carrier reversal}_G$  and
  rev':  $\pi' \in \text{carrier reversal}_G$ 
hence  $\pi \otimes \text{BijGroup UNIV } \pi' = \pi \circ \pi'$ 
using bij rewrite-mult-univ
by blast
also from rev rev' have  $\dots = \pi \otimes \text{reversal}_G \pi'$ 
unfolding reversalG-def

```

by *simp*  
 finally show  $\pi \otimes_{\text{reversal}_{\mathcal{G}}} \pi' = \pi \otimes_{\text{BijGroup UNIV}} \pi'$   
 by *simp*  
 qed

**lemma**  $\varphi$ - $\psi$ -rev-well-formed:

shows *is-symmetry* ( $\lambda \mathcal{E}. \text{limit-set-SWF } (\text{alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV}$ )  
 (action-induced-equivariance (carrier  $\text{reversal}_{\mathcal{G}}$ ) valid-elections  
 ( $\varphi$ -rev valid-elections) (set-action  $\psi$ -rev))

**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**

$\pi \in \text{carrier reversal}_{\mathcal{G}}$  **and**

$(A, V, p) \in \text{valid-elections}$

**moreover from this have cases:**  $\pi \in \{\text{id}, \text{rev-rel}\}$

**unfolding**  $\text{reversal}_{\mathcal{G}}\text{-def}$

**by** *auto*

**ultimately have**  $\text{eq-A: alternatives-}\mathcal{E} (\varphi\text{-rev valid-elections } \pi (A, V, p)) = A$

**by** *simp*

**have**

$\forall r \in \{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A \ r)\}. \exists r' \in \text{UNIV}.$

$\text{rev-rel } r = \text{limit } A (\text{rev-rel } r') \wedge$

$\text{rev-rel } r' \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A (\text{rev-rel } r'))$

**using**  $\text{rev-rel-limit[of } A] \text{ rev-rel-lin-ord[of } A]$

**by** *force*

**hence**

$\forall r \in \{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A \ r)\}.$

$\text{rev-rel } r \in$

$\{\text{limit } A (\text{rev-rel } r') \mid r'. \text{rev-rel } r' \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A$

$(\text{rev-rel } r'))\}$

**by** *blast*

**moreover have**

$\{\text{limit } A (\text{rev-rel } r') \mid r'. \text{rev-rel } r' \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A (\text{rev-rel } r'))\} \subseteq$

$\{\text{limit } A \ r \mid r. r \in \text{UNIV} \wedge \text{linear-order-on } A (\text{limit } A \ r)\}$

**by** *blast*

**ultimately have**  $\forall r \in \text{limit-set-SWF } A \text{ UNIV}. \text{rev-rel } r \in \text{limit-set-SWF } A \text{ UNIV}$

**unfolding**  $\text{limit-set-SWF.simps}$

**by** *blast*

**hence subset:**  $\forall r \in \text{limit-set-SWF } A \text{ UNIV}. \pi \ r \in \text{limit-set-SWF } A \text{ UNIV}$

**using** *cases*

**by** *fastforce*

**hence**  $\forall r \in \text{limit-set-SWF } A \text{ UNIV}. r \in \pi^{-1} \text{ limit-set-SWF } A \text{ UNIV}$

**using** *rev-rev-id comp-apply empty-iff id-apply image-eqI insert-iff cases*

```

    by metis
  hence  $\pi \text{ ' } \textit{limit-set-SWF } A \text{ UNIV} = \textit{limit-set-SWF } A \text{ UNIV}$ 
    using subset
    by blast
  hence  $\textit{set-action } \psi\text{-rev } \pi (\textit{limit-set-SWF } A \text{ UNIV}) = \textit{limit-set-SWF } A \text{ UNIV}$ 
    unfolding set-action.simps
    by simp
  also have
    ... =  $\textit{limit-set-SWF } (\textit{alternatives-}\mathcal{E} (\varphi\text{-rev valid-elections } \pi (A, V, p))) \text{ UNIV}$ 
    using eq-A
    by simp
  finally show
     $\textit{limit-set-SWF } (\textit{alternatives-}\mathcal{E} (\varphi\text{-rev valid-elections } \pi (A, V, p))) \text{ UNIV} =$ 
     $\textit{set-action } \psi\text{-rev } \pi (\textit{limit-set-SWF } (\textit{alternatives-}\mathcal{E} (A, V, p)) \text{ UNIV})$ 
    by simp
qed
end

```

## 1.10 Result-Dependent Voting Rule Properties

```

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

```

### 1.10.1 Properties Dependent on the Result Type

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

```

locale result-properties = result +
  fixes  $\psi\text{-neutr} :: ('a \Rightarrow 'a, 'b) \text{ binary-fun}$  and
     $\mathcal{E} :: ('a, 'v) \text{ Election}$ 
  assumes
    act-neutr:  $\textit{group-action neutrality}_G \text{ UNIV } \psi\text{-neutr}$  and
    well-formed-res-neutr:
      is-symmetry  $(\lambda \mathcal{E} :: ('a, 'v) \text{ Election}. \textit{limit-set } (\textit{alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV})$ 
      (action-induced-equivariance (carrier neutralityG)
        valid-elections ( $\varphi\text{-neutr valid-elections}$ ) (set-action  $\psi\text{-neutr}$ ))

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

```



### 1.10.2 Interpretations

**global-interpretation** *SCF-properties:*

*result-properties well-formed-SCF limit-set-SCF  $\psi$ -neutr<sub>c</sub>*

**unfolding** *result-properties-def result-properties-axioms-def*

**using** *wf-result-neutrality-SCF  $\psi$ -neutr<sub>c</sub>-act.group-action-axioms*  
*SCF-result.result-axioms*

**by** *blast*

**global-interpretation** *SWF-properties:*

*result-properties well-formed-SWF limit-set-SWF  $\psi$ -neutr<sub>w</sub>*

**unfolding** *result-properties-def result-properties-axioms-def*

**using** *wf-result-neutrality-SWF  $\psi$ -neutr<sub>w</sub>-act.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
     $f :: 'a \Rightarrow 'b::ord$  and
     $g :: 'a \Rightarrow 'b$  and
     $S :: 'a\ set$  and
     $x :: 'a$ 
  assumes  $\forall x \in S. f\ x = g\ x$ 
  shows  $is-arg-min\ f\ (\lambda s. s \in S)\ x = is-arg-min\ g\ (\lambda s. s \in S)\ x$ 
proof (unfold is-arg-min-def, cases  $x \notin S$ )
  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
     $y-in-S: y \in S$  and
     $g-y-lt-g-x: g y < g x$ 
  have  $f-eq-g-for-elems-in-S: \forall a. a \in S \longrightarrow f a = g a$ 
  using assms
  by simp
  hence  $g x = f x$ 
  using x-in-S
  by presburger
thus False
  using  $f-eq-g-for-elems-in-S$   $g-y-lt-g-x$   $not-y$   $y-in-S$ 
  by  $(metis (no-types))$ 
qed
thus ?thesis
  using x-in-S not-y
  by simp
qed
qed

```

**lemma** *list-cons-presv-finiteness:*

```

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

```

```

    a :: 'a and
    A' :: 'a set
  assume finite {a#l | 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 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
    elems-fin-then-set-fin: ∀ i::nat < length l. finite (!i) ⟹ finite (listset l) and
    fin-all-elems: ∀ i::nat < length (a#l). finite ((a#l)!i)
  hence finite a
  by auto
  moreover from fin-all-elems
  have ∀ i < length l. finite (!i)
  by auto
  hence finite (listset l)
  using elems-fin-then-set-fin
  by simp
  ultimately have finite {a'#l' | a' l'. a' ∈ a ∧ l' ∈ (listset l)}
  using list-cons-presv-finiteness

```

```

    by auto
  thus finite (listset (a#l))
    by (simp add: set-Cons-def)
qed

```

**lemma** *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  $\in$  listset []
  thus length l = length []
    by simp
next
  case (Cons a l)
  fix
    a :: 'a set and
    l :: 'a set list and
    m :: 'a list
  assume
     $\forall l'. l' \in \text{listset } l \longrightarrow \text{length } l' = \text{length } l$  and
    m  $\in$  listset (a#l)
  moreover have
     $\forall a' l'::('a \text{ set list}). \text{listset } (a'\#l') = \{b\#m \mid b \text{ m. } b \in a' \wedge m \in \text{listset } l'\}$ 
    by (simp add: set-Cons-def)
  ultimately show length m = length (a#l)
    using local.Cons
    by force
qed

```

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

```

  fixes l :: 'a set list
  shows  $\forall l' i::\text{nat}. l' \in \text{listset } l \wedge i < \text{length } l' \longrightarrow l'!i \in l!i$ 
proof (induct l, safe)
  case Nil
  fix
    l' :: 'a list and
    i :: nat
  assume
    l'  $\in$  listset [] and
    i < length l'
  thus l'!i  $\in$  []!i
    using Nil
    by simp
next
  case (Cons a l)
  fix
    a :: 'a set and

```

```

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

lemma all-ls-in-ls-set:
  fixes l :: 'a set list
  shows ∀ l'. length l' = length l ∧ (∀ i < length l'. l'!i ∈ !l i) → l' ∈ listset l
proof (induction l, safe)
  case Nil
  fix l' :: 'a list
  assume length l' = length []
  thus l' ∈ listset []
  by simp
next
  case (Cons a l)
  fix
    l :: 'a set list and
    l' :: 'a list and
    s :: 'a set
  assume
    all-ls-in-ls-set-induct:
    ∀ m. length m = length l ∧ (∀ i < length m. m!i ∈ !l i) → m ∈ listset l and
    len-eq: length l' = length (s#l) and
    elems-pos-in-cons-ls-pos: ∀ i < length l'. l'!i ∈ (s#l)!i
  then obtain t and x where
    l'-cons: l' = x#t
  using length-Suc-conv
  by metis
hence x ∈ s
  using elems-pos-in-cons-ls-pos
  by force
moreover have t ∈ listset l

```

```

using l'-cons all-ls-in-ls-set-induct len-eq diff-Suc-1 diff-Suc-eq-diff-pred
      elems-pos-in-cons-ls-pos length-Cons nth-Cons-Suc zero-less-diff
by metis
ultimately show  $l' \in \text{listset } (s\#l)$ 
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
  (-  $\lesssim$  - [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 Preference-List
  shows Relation.trans (pl- $\alpha$  l)
  unfolding Relation.trans-def pl- $\alpha$ -def
  by simp

lemma pl- $\alpha$ -lin-order:
  fixes
    A :: 'a set and
    r :: 'a rel
  assumes el: r  $\in$  pl- $\alpha$  'permutations-of-set A
  shows linear-order-on A r
proof (cases A = {})
case True
  thus ?thesis
    using assms
    unfolding pl- $\alpha$ -def is-less-preferred-than-l.simps
    by simp
next
case False
  thus ?thesis
proof (unfold linear-order-on-def total-on-def antisym-def
  partial-order-on-def preorder-on-def, safe)
  have A  $\neq$  {}
  using False
  by simp
  hence  $\forall l \in$  permutations-of-set A. l  $\neq$  []
  using assms permutations-of-setD(1)
  by force
  hence  $\forall a \in A. \forall l \in$  permutations-of-set A. a  $\lesssim_l$  a
  using is-less-preferred-than-l.simps
  unfolding permutations-of-set-def
  by simp
  hence  $\forall a \in A. \forall l \in$  permutations-of-set A. (a, a)  $\in$  pl- $\alpha$  l
  unfolding pl- $\alpha$ -def
  by simp
  hence  $\forall a \in A. (a, a) \in r$ 
  using el
  by auto
  moreover have r  $\subseteq A \times A$ 
  using el
  unfolding pl- $\alpha$ -def permutations-of-set-def
  by auto
end
end

```



```

ultimately show refl-on A r
  unfolding refl-on-def
  by simp
next
show Relation.trans r
  using el rel-trans
  by auto
next
fix
  x :: 'a and
  y :: 'a
assume
  x-rel-y: (x, y) ∈ r and
  y-rel-x: (y, x) ∈ r
have ∀ x y. ∀ l ∈ permutations-of-set A. x ≲l y ∧ y ≲l x ⟶ x = y
  using is-less-preferred-than-l.simps index-eq-index-conv nle-le
  unfolding permutations-of-set-def
  by metis
hence ∀ x y. ∀ l ∈ pl-α ' permutations-of-set A. (x, y) ∈ l ∧ (y, x) ∈ l ⟶ x
= y
  unfolding pl-α-def permutations-of-set-def antisym-on-def
  by blast
thus x = y
  using y-rel-x x-rel-y el
  by auto
next
fix
  x :: 'a and
  y :: 'a
assume
  x-in-A: x ∈ A and
  y-in-A: y ∈ A and
  x-neq-y: x ≠ y and
  not-y-x-rel: (y, x) ∉ r
have ∀ x y. ∀ l ∈ permutations-of-set A. x ∈ A ∧ y ∈ A ∧ x ≠ y ∧ (¬ y ≲l
x) ⟶ x ≲l y
  using is-less-preferred-than-l.simps
  unfolding permutations-of-set-def
  by auto
hence ∀ x y. ∀ l ∈ pl-α ' permutations-of-set A.
  x ∈ A ∧ y ∈ A ∧ x ≠ y ∧ (y, x) ∉ l ⟶ (x, y) ∈ l
  unfolding pl-α-def permutations-of-set-def
  by blast
thus (x, y) ∈ r
  using x-in-A y-in-A x-neq-y not-y-x-rel el
  by auto
qed
qed

```

```

lemma lin-order-pl-α:
  fixes
    r :: 'a rel and
    A :: 'a set
  assumes
    lin-order: linear-order-on A r and
    fin: finite A
  shows r ∈ 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 b. a ∈ ((underS r b) ∩ A) ∧ b ∈ ((underS r a) ∩ A) ⟶ False
    using lin-order
    unfolding underS-def linear-order-on-def partial-order-on-def antisym-def
    by auto
  hence ∀ a b c. a ∈ (underS r b) ∩ A ⟶ b ∈ (underS r c) ∩ A ⟶ a ∈ (underS
r c) ∩ A
    using lin-order CollectD CollectI transD IntE IntI
    unfolding underS-def linear-order-on-def partial-order-on-def preorder-on-def
    by (metis (mono-tags, lifting))
  hence ∀ a b. a ∈ (underS r b) ∩ A ⟶ (underS r a) ∩ A ⊂ (underS r b) ∩ A
    using antisym
    by blast
  hence mon: ∀ a b. a ∈ (underS r b) ∩ A ⟶ ?φ a < ?φ b
    using fin
    by (simp add: psubset-card-mono)
  moreover have total-underS:
    ∀ a b. a ∈ A ∧ b ∈ A ∧ a ≠ b ⟶ a ∈ ((underS r b) ∩ A) ∨ b ∈ ((underS r
a) ∩ 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 b. 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-sup-ord(2) linorder-le-less-linear
    unfolding underS-def
    by (metis (mono-tags, lifting))
  hence ?φ ' A ⊆ {0..card A}
    using atLeast0LessThan
    by blast
  moreover have card (?φ ' A) = 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** *bij*:  $\text{bij-betw } ? \varphi \text{ ' } A \{0 \dots \text{card } A\}$   
**using** *inj*  
**unfolding** *bij-betw-def*  
**by** *safe*  
**hence** *bij-inv*:  $\text{bij-betw } ? \text{inv } \{0 \dots \text{card } A\} \text{ ' } A$   
**using** *bij-betw-the-inv-into*  
**by** *metis*  
**hence**  $? \text{inv ' } \{0 \dots \text{card } A\} = A$   
**unfolding** *bij-betw-def*  
**by** *metis*  
**hence** *set-eq-A*:  $\text{set } ?l = A$   
**by** *simp*  
**moreover have** *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*:  $\forall a \in A. \text{index } ?l \text{ ' } a = \text{card } A - 1 - ? \varphi \text{ ' } a$   
**proof**  
**fix**  $a :: 'a$   
**assume** *a-in-A*:  $a \in A$   
**have**  $\forall xs. \forall i < \text{length } xs. (\text{rev } xs)!i = xs!(\text{length } xs - 1 - i)$   
**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** *card-A-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 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**  $\text{card } A - 1 - (\text{card } A - 1 - \text{card } (\text{underS } r \text{ ' } a \cap A)) = \text{card } (\text{underS } r \text{ ' } a \cap A)$   
**using** *in-bounds a-in-A*  
**by** *auto*  
**moreover have**  $? \text{inv } (\text{card } (\text{underS } r \text{ ' } a \cap A)) = a$   
**using** *a-in-A inj the-inv-into-f-f*  
**by** *fastforce*

```

ultimately have ?l!(card A - 1 - card (underS r a ∩ 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 ∩ 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-α ?l = r
proof
  show r ⊆ pl-α ?l
  proof (unfold pl-α-def is-less-preferred-than-l.simps, safe)
    fix
      a :: 'a and
      b :: 'a
    assume (a, b) ∈ r
    hence a ∈ A
      using lin-order
    unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
    by auto
    thus a ∈ set ?l
      using bij-inv set-eq-A
      by metis
  next
    fix
      a :: 'a and
      b :: 'a
    assume (a, b) ∈ r
    hence b ∈ A
      using lin-order
    unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
    by auto
    thus b ∈ set ?l
      using bij-inv set-eq-A
      by metis
  next
    fix
      a :: 'a and
      b :: 'a
    assume rel: (a, b) ∈ r
    hence el-A: a ∈ A ∧ b ∈ A
      using lin-order
    unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
    by auto
    moreover have a ∈ underS r b ∨ a = b
      using lin-order rel
    unfolding underS-def
    by simp
    ultimately have ?φ a ≤ ?φ b

```

```

    using mon le-eq-less-or-eq
    by auto
  thus index ?l b ≤ index ?l a
    using index-eq el-A diff-le-mono2
    by metis
qed
next
show pl-α ?l ⊆ r
proof (unfold pl-α-def is-less-preferred-than-l.simps, safe)
  fix
    a :: 'a and
    b :: 'a
  assume
    in-bnds-a: a ∈ set ?l and
    in-bnds-b: b ∈ set ?l and
    index-rel: index ?l b ≤ index ?l a
  have el-a: ?inv (index ?l a) ∈ A
    using bij-inv in-bnds-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) ∈ A
    using bij-inv in-bnds-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))
  ultimately have leq-diff: card A - 1 - (?φ b) ≤ card A - 1 - (?φ a)
    using index-rel index-eq in-bnds-a in-bnds-b set-eq-A
    by metis
  have ∀ a < card A. ?φ (?inv a) < card A
    using fin bij-inv bij
    unfolding bij-betw-def
    by fastforce
  hence ?φ b ≤ card A - 1 ∧ ?φ a ≤ card A - 1
    using in-bnds-a in-bnds-b fin
    by fastforce
  hence ?φ b ≥ ?φ a
    using fin leq-diff le-diff-iff'
    by blast
  hence cases: ?φ a < ?φ b ∨ ?φ a = ?φ b
    by auto
  have ∀ a b. a ∈ A ∧ b ∈ A ∧ ?φ a < ?φ b ⟶ a ∈ underS r b
    using mon total-underS antisym IntD1 order-less-not-sym
    by metis
  hence ?φ a < ?φ b ⟶ a ∈ underS r b

```

```

    using el-a el-b in-bnds-a in-bnds-b set-eq-A
    by blast
  hence cases-less:  $? \varphi \ a < ? \varphi \ b \longrightarrow (a, b) \in r$ 
    unfolding underS-def
    by simp
  have  $\forall \ a \ b. \ a \in A \wedge b \in A \wedge ? \varphi \ a = ? \varphi \ b \longrightarrow a = b$ 
    using mon total-underS antisym order-less-not-sym
    by metis
  hence  $? \varphi \ a = ? \varphi \ b \longrightarrow a = b$ 
    using el-a el-b in-bnds-a in-bnds-b set-eq-A
    by simp
  hence cases-eq:  $? \varphi \ a = ? \varphi \ b \longrightarrow (a, b) \in r$ 
    using lin-order el-a el-b in-bnds-a in-bnds-b set-eq-A
    unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
    by auto
  show  $(a, b) \in r$ 
    using cases cases-less cases-eq
    by auto
qed
qed
ultimately show  $r \in \text{pl-}\alpha \text{ 'permutations-of-set } A$ 
  by auto
qed

lemma index-helper:
  fixes
    xs :: 'x list and
    x :: 'x
  assumes
    fin-set-xs: finite (set xs) and
    dist-xs: distinct xs and
    x ∈ set xs
  shows index xs x = card {y ∈ set xs. index xs y < index xs x}
proof -
  have bij: bij-betw (index xs) (set xs) {0 ..< length xs}
    using assms bij-betw-index
    by blast
  hence card {y ∈ set xs. index xs y < index xs x}
    = card (index xs ' {y ∈ set xs. index xs y < index xs x})
    using CollectD bij-betw-same-card bij-betw-subset subsetI
    by (metis (no-types, lifting))
  also have index xs ' {y ∈ set xs. index xs y < index xs x}
    = {m | m. m ∈ index xs ' (set xs) ∧ m < index xs x}
    by blast
  also have {m | m. m ∈ index xs ' (set xs) ∧ m < index xs x} = {m | m. m <
index xs x}
    using bij assms atLeastLessThan-iff bot-nat-0.extremum
    index-image index-less-size-conv order-less-trans
    by metis

```

```

also have  $\text{card } \{m \mid m. m < \text{index } xs \ x\} = \text{index } xs \ x$ 
by simp
finally show ?thesis
by simp
qed

lemma pl- $\alpha$ -eq-imp-list-eq:
fixes
   $xs :: 'x \text{ list}$  and
   $ys :: 'x \text{ list}$ 
assumes
  fin-set-xs: finite (set  $xs$ ) and
  set-eq:  $\text{set } xs = \text{set } ys$  and
  dist-xs: distinct  $xs$  and
  dist-ys: distinct  $ys$  and
  pl- $\alpha$ -eq:  $\text{pl-}\alpha \ xs = \text{pl-}\alpha \ ys$ 
shows  $xs = ys$ 
proof (rule ccontr)
assume  $xs \neq ys$ 
moreover with this
have  $xs \neq [] \wedge ys \neq []$ 
using set-eq
by auto
ultimately obtain
   $i :: \text{nat}$  and
   $x :: 'x$  where
     $i < \text{length } xs$  and
     $xs[i] \neq ys[i]$  and
     $x = xs[i]$  and
   $x \in \text{set } xs$ 
using dist-xs dist-ys distinct-remdups-id
  length-remdups-card-conv nth-equalityI nth-mem set-eq
by metis
moreover with this
have neq-ind:  $\text{index } xs \ x \neq \text{index } ys \ x$ 
using dist-xs index-nth-id nth-index set-eq
by metis
ultimately have
   $\text{card } \{y \in \text{set } xs. \text{index } xs \ y < \text{index } xs \ x\} \neq \text{card } \{y \in \text{set } xs. \text{index } ys \ y <$ 
 $\text{index } ys \ x\}$ 
using dist-xs dist-ys set-eq index-helper fin-set-xs
by (metis (mono-tags))
then obtain  $y :: 'x$  where
   $y \in \text{set } xs$  and
   $y \neq x$  and
  neq-indices:
     $(\text{index } xs \ y < \text{index } xs \ x \wedge \text{index } ys \ y > \text{index } ys \ x) \vee$ 
     $(\text{index } ys \ y < \text{index } ys \ x \wedge \text{index } xs \ y > \text{index } xs \ x)$ 
using index-eq-index-conv not-less-iff-gr-or-eq set-eq

```

```

    by (metis (mono-tags, lifting))
  hence (is-less-preferred-than-l x xs y ∧ is-less-preferred-than-l y ys x)
        ∨ (is-less-preferred-than-l x ys y ∧ is-less-preferred-than-l y xs x)
    unfolding is-less-preferred-than-l.simps
    using y-in-set-xs less-imp-le-nat set-eq x-in-xs
    by blast
  hence ((x, y) ∈ pl-α xs ∧ (x, y) ∉ pl-α ys) ∨ ((x, y) ∈ pl-α ys ∧ (x, y) ∉ pl-α
xs)
    unfolding pl-α-def
    using is-less-preferred-than-l.simps y-neq-x neq-indices
        case-prod-conv linorder-not-less mem-Collect-eq
    by metis
  thus False
    using pl-α-eq
    by blast
qed

```

```

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

```

### 2.1.5 Limited Preference

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

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

**lemma** *limited-dest*:



```

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

lemma limit-equiv:
fixes
  A :: 'a set and
  l :: 'a list
assumes well-formed-l l
shows  $pl\text{-}\alpha \ (limit\text{-}l \ A \ l) = limit \ A \ (pl\text{-}\alpha \ l)$ 
using assms
proof (induction l)
case Nil
thus  $pl\text{-}\alpha \ (limit\text{-}l \ A \ []) = limit \ A \ (pl\text{-}\alpha \ [])$ 
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  $\implies pl\text{-}\alpha \ (limit\text{-}l \ A \ l) = limit \ A \ (pl\text{-}\alpha \ l)$  and
  wf-a-l: well-formed-l (a#l)
show  $pl\text{-}\alpha \ (limit\text{-}l \ A \ (a\#l)) = limit \ A \ (pl\text{-}\alpha \ (a\#l))$ 
using wf-imp-limit wf-a-l
proof (unfold limit-l.simps limit.simps, intro equalityI, safe)
fix
  b :: 'a and
  c :: 'a
assume
  b-less-c: (b, c)  $\in pl\text{-}\alpha \ (filter \ (\lambda a. a \in A) \ (a\#l))$ 
have limit-preference-list-assoc:  $pl\text{-}\alpha \ (limit\text{-}l \ A \ l) = limit \ A \ (pl\text{-}\alpha \ l)$ 
using wf-a-l wf-imp-limit
by simp
thus  $(b, c) \in pl\text{-}\alpha \ (a\#l)$ 
proof (unfold pl- $\alpha$ -def is-less-preferred-than-l.simps, safe)
show  $b \in set \ (a\#l)$ 
using b-less-c case-prodD filter-set mem-Collect-eq member-filter
is-less-preferred-than-l.simps

```

```

    unfolding pl- $\alpha$ -def
    by metis
next
show  $c \in \text{set } (a \# l)$ 
  using b-less-c case-prodD filter-set mem-Collect-eq member-filter
    is-less-preferred-than-l.simps
  unfolding pl- $\alpha$ -def
  by metis
next
have idx-set-eq:
 $\forall a' l' a''. (a' :: 'a) \lesssim_{l'} a'' =$ 
 $(a' \in \text{set } l' \wedge a'' \in \text{set } l' \wedge \text{index } l' a'' \leq \text{index } l' a')$ 
  using is-less-preferred-than-l.simps
  by blast
moreover from this
have  $\{(a', b'). a' \lesssim_{(\text{limit-}l A l)} b'\} =$ 
 $\{(a', a''). a' \in \text{set } (\text{limit-}l A l) \wedge a'' \in \text{set } (\text{limit-}l A l) \wedge$ 
 $\text{index } (\text{limit-}l A l) a'' \leq \text{index } (\text{limit-}l A l) a'\}$ 
  by presburger
moreover from this
have  $\{(a', b'). a' \lesssim_l b'\} =$ 
 $\{(a', a''). a' \in \text{set } l \wedge a'' \in \text{set } l \wedge \text{index } l a'' \leq \text{index } l a'\}$ 
  using is-less-preferred-than-l.simps
  by auto
ultimately have  $\{(a', b').$ 
 $a' \in \text{set } (\text{limit-}l A l) \wedge b' \in \text{set } (\text{limit-}l A l) \wedge$ 
 $\text{index } (\text{limit-}l A l) b' \leq \text{index } (\text{limit-}l A l) a'\} =$ 
 $\text{limit } A \{(a', b'). a' \in \text{set } l \wedge b' \in \text{set } l \wedge \text{index } l b' \leq \text{index } l a'\}$ 
  using pl- $\alpha$ -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- $\alpha$ -def
  by (metis (no-types))
moreover obtain
 $f :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \Rightarrow 'a$  and
 $g :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \Rightarrow 'a \text{ list}$  and
 $h :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \Rightarrow 'a$  where
 $\forall d s e. d \lesssim_s e \longrightarrow$ 
 $d = f e s d \wedge s = g e s d \wedge e = h e s d \wedge f e s d \in \text{set } (g e s d) \wedge$ 
 $\text{index } (g e s d) (h e s d) \leq \text{index } (g e s d) (f e s d) \wedge$ 
 $h e s d \in \text{set } (g e s d)$ 
  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**  $a \neq c \longrightarrow \text{index } (a\#l)\ c \leq \text{index } (a\#l)\ b$   
**proof** (*unfold filter.simps is-less-preferred-than-l.elims, safe*)  
**assume**  
 $\text{filter-}eq\text{-limit:}$   
 $\text{filter } (\lambda\ a.\ a \in A)\ l = \text{limit-}l\ A\ l$  **and**  
 $b\text{-}eq:$   
 $b = f\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b$   
**and**  
 $a\text{-}eq\text{-}g\text{-}filter:$   
 $(\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)$   
 $= g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b$   
**and**  
 $c\text{-}eq:$   
 $c = h\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)$   
**b and**  
 $f\text{-}elem\text{-}set\text{-}g\text{-}filter:$   
 $f\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b$   
 $\in \text{set } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
**and**  
 $h\text{-}elem\text{-}set\text{-}g\text{-}filter:$   
 $h\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b$   
 $\in \text{set } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
**and**  
 $idx\text{-}h\text{-}leq\text{-}f:$   
 $\text{index } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else } \text{filter } (\lambda\ a.\ a \in A)\ l))$

$b)$   
 $(h\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
 $\leq \text{index } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
 $(f\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l))$   
 $b)$  and  
 $a\text{-neq-}c: a \neq c$   
 $\{$   
 $\text{moreover assume}$   
 $\text{index } (\text{limit-}l\ A)\ c$   
 $\neq \text{index } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
 $(h\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
 $\text{ultimately have index } (a\#l)\ c \leq \text{index } (a\#l)\ b$   
 $\text{using add-le-cancel-right idx-imp index-Cons le-zero-eq nth-index set-ConsD}$   
 $\text{by metis}$   
 $\text{thus}$   
 $\text{index } (a\#l)\ c \leq \text{index } (a\#l)\ b$  and  
 $\text{index } (a\#l)\ c \leq \text{index } (a\#l)\ b$  and  
 $\text{index } (a\#l)\ c \leq \text{index } (a\#l)\ b$   
 $\text{by (clarify, clarify, clarify)}$   
 $\}$   
 $\{$   
 $\text{assume}$   
 $\text{index } (\text{limit-}l\ A)\ b$   
 $\neq \text{index } (g\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l)\ b)$   
 $(f\ c\ (\text{if } a \in A \text{ then } a\#\text{filter } (\lambda\ a.\ a \in A)\ l \text{ else filter } (\lambda\ a.\ a \in A)\ l))$   
 $b)$   
 $\text{thus index } (a\#l)\ c \leq \text{index } (a\#l)\ b$   
 $\text{using filter-eq-limit b-eq a-eq-g-filter c-eq f-elem-set-g-filter}$   
 $\text{h-elem-set-g-filter idx-h-leq-f a-neq-c add-le-cancel-right}$   
 $\text{idx-imp index-Cons le-zero-eq nth-index set-ConsD}$   
 $\text{by metis}$   
 $\}$   
 $\{$   
 $\text{assume}$   
 $\text{index } (\text{limit-}l\ A)\ c$   
 $\neq \text{index } (g\ c\ (\text{filter } (\lambda\ a.\ a \in A)\ (a\#l))\ b)\ (h\ c\ (\text{filter } (\lambda\ a.\ a \in A)\ (a\#l))\ b)$   
 $(a\#l))\ b)$   
 $\text{thus index } (a\#l)\ c \leq \text{index } (a\#l)\ b$   
 $\text{using filter-eq-limit b-eq a-eq-g-filter c-eq f-elem-set-g-filter}$   
 $\text{h-elem-set-g-filter idx-h-leq-f a-neq-c add-le-cancel-right}$   
 $\text{idx-imp index-Cons le-zero-eq nth-index set-ConsD}$   
 $\text{unfolding filter.simps}$   
 $\text{by metis}$   
 $\}$   
 $\{$   
 $\text{assume}$

```

      index (limit-l A l) b
      ≠ index (g c (if a ∈ A then a#filter (λ a. a ∈ A) l else filter (λ a. a ∈
A) l) b)
      (f c (if a ∈ A then a#filter (λ a. a ∈ A) l else filter (λ a. a ∈ A) l) b)
hence index (a#l) c ≤ index (a#l) b
      using filter-eq-limit b-eq a-eq-g-filter c-eq f-elem-set-g-filter
             h-elem-set-g-filter idx-h-leq-f a-neq-c add-le-cancel-right
             idx-imp index-Cons le-zero-eq nth-index set-ConsD
      by metis
thus
      index (a#l) c ≤ index (a#l) b and
      index (a#l) c ≤ index (a#l) b
      by (clarify, clarify)
    }
  {
    moreover assume
      index (limit-l A l) b
      ≠ index (g c (filter (λ a. a ∈ A) (a#l)) b) (f c (filter (λ a. a ∈ A)
(a#l)) b)
    ultimately show index (a#l) c ≤ index (a#l) b
      using idx-imp
      unfolding filter.simps
      by metis
  }
  {
    assume index (limit-l A l) c ≤ index (limit-l A l) b
    thus index (a#l) c ≤ index (a#l) b
      using filter-eq-limit b-eq a-eq-g-filter c-eq f-elem-set-g-filter
             h-elem-set-g-filter idx-h-leq-f a-neq-c add-le-cancel-right
             idx-imp index-Cons le-zero-eq nth-index set-ConsD wf-a-l
    unfolding distinct.simps
    by metis
  }
qed
thus index (a#l) c ≤ index (a#l) b
by force
qed
next
fix
  b :: 'a and
  c :: 'a
assume (b, c) ∈ pl-α (filter (λ a. a ∈ A) (a#l))
thus b ∈ A
  unfolding pl-α-def is-less-preferred-than-l.simps
  using case-prodD mem-Collect-eq set-filter
  by (metis (no-types, lifting))
next
fix
  b :: 'a and

```

```

    c :: 'a
  assume (b, c) ∈ pl-α (filter (λ a. a ∈ A) (a#l))
  thus c ∈ A
    unfolding pl-α-def
    using case-prodD filter-set is-less-preferred-than-l.simps
      mem-Collect-eq member-filter
    by metis
next
fix
  b :: 'a and
  c :: 'a
  assume
    b-less-c: (b, c) ∈ pl-α (a#l) and
    b-in-A: b ∈ A and
    c-in-A: c ∈ A
  show (b, c) ∈ pl-α (filter (λ a. a ∈ A) (a#l))
  proof (unfold pl-α-def is-less-preferred-than-l.simps, safe)
    show b ≲(filter (λ a. a ∈ A) (a#l)) c
    proof (unfold is-less-preferred-than-l.simps, safe)
      show b ∈ set (filter (λ a. a ∈ A) (a#l))
      using b-less-c b-in-A
      unfolding pl-α-def
      by fastforce
    next
      show c ∈ set (filter (λ a. a ∈ A) (a#l))
      using b-less-c c-in-A
      unfolding pl-α-def
      by fastforce
    next
      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 show
      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
      in-rel-Collect-case-prod-eq linorder-le-cases mem-Collect-eq not-one-le-zero
      by fastforce
  qed
qed
qed

```

qed

### 2.1.6 Auxiliary Definitions

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

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

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

theorem above-equiv:

```



```

fixes
   $l :: 'a$  Preference-List and
   $a :: 'a$ 
shows  $\text{set } (\text{above-}l\ l\ a) = \text{above } (pl-\alpha\ l)\ a$ 
proof (safe)
  fix  $b :: 'a$ 
  assume  $b \in \text{set } (\text{above-}l\ l\ a)$ 
  hence  $\text{index } l\ b \leq \text{index } l\ a$ 
    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  $b \in \text{above } (pl-\alpha\ l)\ a$ 
    using less-preferred-l-rel-equiv pref-imp-in-above
    by metis
next
  fix  $b :: 'a$ 
  assume  $b \in \text{above } (pl-\alpha\ l)\ a$ 
  hence  $a \lesssim_l b$ 
    using pref-imp-in-above less-preferred-l-rel-equiv
    by metis
  thus  $b \in \text{set } (\text{above-}l\ l\ a)$ 
    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  $\text{rank-}l\ l\ a = \text{rank } (pl-\alpha\ l)\ a$ 
proof (unfold rank-l.simps rank.simps, cases a ∈ set l)
  case True
  moreover have  $\text{above } (pl-\alpha\ l)\ a = \text{set } (\text{above-}l\ l\ a)$ 
    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  $\text{card } (\text{set } (\text{above-}l\ l\ a)) = \text{length } (\text{above-}l\ l\ a)$ 

```

```

    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 pl-α-def linear-order-on-l-def linear-order-on-def refl-on-l-def
    Relation.trans-def preorder-on-l-def partial-order-on-l-def partial-order-on-def
    total-on-l-def preorder-on-def refl-on-def antisym-def total-on-def
    is-less-preferred-than-l.simps
  by auto

```

### 2.1.8 First Occurrence Indices

```

lemma pos-in-list-yields-rank:
  fixes
    l :: 'a Preference-List and
    a :: 'a and
    n :: nat
  assumes
    ∀ (j::nat) ≤ n. l!j ≠ a and
    l!(n - 1) = a
  shows rank-l l a = n
  using assms
proof (induction l arbitrary: n)
  case Nil
  thus ?case
    by simp
next
fix

```

```

    l :: 'a Preference-List and
    a :: 'a
  case (Cons a l)
  thus ?case
    by simp
qed

```

```

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

```

```

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

```

```

lemma rel-of-pref-pred-for-set-eq-list-to-rel:
  fixes l :: 'a Preference-List
  shows relation-of (λ y z. y ≲l z) (set l) = pl-α l

```

```

proof (unfold relation-of-def, safe)
  fix
     $a :: 'a$  and
     $b :: 'a$ 
  assume  $a \lesssim_l b$ 
  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
     $a :: 'a$  and
     $b :: 'a$ 
  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**  $'a\ Election-List = 'a\ set \times 'a\ Profile-List$

Abstraction from profile list to profile.

```

fun pl-to-pr- $\alpha :: 'a\ Profile-List \Rightarrow ('a, nat)\ Profile$  where
  pl-to-pr- $\alpha\ pl = (\lambda\ n.\ if\ (n < length\ pl \wedge n \geq 0)$ 
    then (map (Preference-List.pl- $\alpha$ ) pl)!n
    else {})

```

```

lemma prof-abstr-presv-size:
  fixes  $p :: 'a \text{ Profile-List}$ 
  shows  $\text{length } p = \text{length } (\text{to-list } \{0 \dots \text{length } p\} \text{ (pl-to-pr-}\alpha \text{ } p))$ 
  by simp

```

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

```

definition profile-l ::  $'a \text{ set} \Rightarrow 'a \text{ Profile-List} \Rightarrow \text{bool}$  where
  profile-l  $A \ p \equiv \forall \ i < \text{length } p. \text{ballot-on } A \ (p!i)$ 

```

```

lemma refinement:
  fixes
     $A :: 'a \text{ set}$  and
     $p :: 'a \text{ Profile-List}$ 
  assumes profile-l  $A \ p$ 
  shows  $\text{profile } \{0 \dots \text{length } p\} \ A \ (\text{pl-to-pr-}\alpha \ p)$ 
proof (unfold profile-def, safe)
  fix  $i :: \text{nat}$ 
  assume in-range:  $i \in \{0 \dots \text{length } p\}$ 
  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  $A \ (\text{pl-to-pr-}\alpha \ p \ i)$ 
    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 –
  assume
    ex:  $\bigwedge \text{ord. linear-order-on } X \text{ ord} \implies ?thesis$ 
  obtain l :: 'x list where
    set-l: set l = X
  using finite-list assms
  by blast
  let ?r = pl- $\alpha$  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
  thus ?thesis
    using ex
    by blast
qed

```

```

typedef 'a Ordered-Preference =
  {p :: 'a::finite Preference-Relation. linear-order-on (UNIV::'a set) p}
morphisms ord2pref pref2ord
proof (unfold mem-Collect-eq)
  have finite (UNIV::'a set)
    by simp
  then obtain p :: 'a Preference-Relation where
    linear-order-on (UNIV::'a set) p
  using fin-ordered
  by metis
  thus  $\exists p::'a \text{ Preference-Relation. linear-order } p$ 
    by blast
qed

```

```

instance Ordered-Preference :: (finite) finite
proof
  have (UNIV::'a Ordered-Preference set) =

```

```

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

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

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

end

```

## 2.4 Alternative Election Type

```

theory Quotient-Type-Election
  imports Profile
begin

lemma election-equality-equiv:
  election-equality E E and
  election-equality E E'  $\implies$  election-equality E' E and
  election-equality E E'  $\implies$  election-equality E' F  $\implies$  election-equality E F
proof –
  have  $\forall E. E = (\text{fst } E, \text{fst } (\text{snd } E), \text{snd } (\text{snd } E))$ 
    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) ElectionQ =
  'a set  $\times$  'v set  $\times$  ('a, 'v) Profile / election-equality
  unfolding equivp-reflp-symp-transp reflp-def symp-def transp-def
  using election-equality-equiv
  by simp

fun fstQ :: ('a, 'v) ElectionQ  $\Rightarrow$  'a set where
  fstQ E = Product-Type.fst (rep-ElectionQ E)

fun sndQ :: ('a, 'v) ElectionQ  $\Rightarrow$  'v set  $\times$  ('a, 'v) Profile where
  sndQ E = Product-Type.snd (rep-ElectionQ E)

abbreviation alternatives- $\mathcal{E}_Q$  :: ('a, 'v) ElectionQ  $\Rightarrow$  'a set where
  alternatives- $\mathcal{E}_Q$  E  $\equiv$  fstQ E

abbreviation voters- $\mathcal{E}_Q$  :: ('a, 'v) ElectionQ  $\Rightarrow$  'v set where
  voters- $\mathcal{E}_Q$  E  $\equiv$  Product-Type.fst (sndQ E)

abbreviation profile- $\mathcal{E}_Q$  :: ('a, 'v) ElectionQ  $\Rightarrow$  ('a, 'v) Profile where
  profile- $\mathcal{E}_Q$  E  $\equiv$  Product-Type.snd (sndQ 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 card s  $\neq$  1. Note that "undefined = undefined" is the only
provable equality for undefined.
```

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

```
fun  $\pi_Q$  :: ('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  $\forall y \in r^{-1}\{x\}. (x, y) \in r$ 
    unfolding Image-def
    by simp
  hence func-eq-x:  $\{f y \mid y. y \in r^{-1}\{x\}\} = \{f x \mid y. y \in r^{-1}\{x\}\}$ 
    using assms
    unfolding congruent-def
    by fastforce
  assume
     $t \in s // r$  and
     $x \text{ in } t: x \in t$ 
  moreover from this have  $r^{-1}\{x\} \in s // r$ 
    using assms quotient-eq-iff equiv-class-eq-iff quotientI
    by metis
  ultimately have r-img-elem-x-eq-t:  $r^{-1}\{x\} = t$ 
    using assms quotient-eq-iff Image-singleton-iff
    by metis
  hence  $\{f x \mid y. y \in r^{-1}\{x\}\} = \{f x\}$ 
    using x-in-t
    by blast
  hence  $f ` t = \{f x\}$ 
    using Setcompr-eq-image r-img-elem-x-eq-t func-eq-x

```

```

    by metis
  thus  $\pi_Q f t = f x$ 
    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
     $\forall A \in s \ / \ r. \ \pi_Q\ induced\text{-}fun\ A = f\ A$ 
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  $\pi_Q\ induced\text{-}fun\ A = induced\text{-}fun\ a$ 
    using pass-to-quotient assms
    by blast
  thus  $\pi_Q\ induced\text{-}fun\ A = f\ A$ 
    using A-eq-rel-class-r-a
    unfolding induced-fun-def
    by simp
qed

```

### 3.1.3 Equivalence Relations

**lemma** *equiv-rel-restr*:

```

fixes
   $s :: 'x \text{ set}$  and
   $t :: 'x \text{ set}$  and
   $r :: 'x \text{ rel}$ 
assumes
   $\text{equiv } s \text{ } r$  and
   $t \subseteq s$ 
shows  $\text{equiv } t \text{ } (\text{Restr } r \text{ } t)$ 
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  $\text{sym } (\text{Restr } r \text{ } t)$ 
    using assms
    unfolding equiv-def sym-def
    by blast
next
  show  $\text{Relation.trans } (\text{Restr } r \text{ } t)$ 
    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  $\text{group-action } m \text{ } s \text{ } \varphi$ 
shows  $\text{equiv } s \text{ } (\text{action-induced-rel } (\text{carrier } m) \text{ } s \text{ } \varphi)$ 
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  $\varphi \text{ } 1 \text{ } m \text{ } y = y$ 
    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 ∈ s and
    carrier-g: g ∈ carrier m
  hence y = φ (inv m g) (φ g y)
    using assms
    by (simp add: group-action.orbit-sym-aux)
  thus ∃ h ∈ carrier m. φ h (φ g y) = y
    using assms carrier-g group.inv-closed group-action.group-hom group-hom.axioms(1)
    by metis
next
fix
  y :: 'y and
  g :: 'x and
  h :: 'x
  assume
    y-in-s: y ∈ s and
    carrier-g: g ∈ carrier m and
    carrier-h: h ∈ carrier m
  hence φ (h ⊗ m g) y = φ h (φ g y)
    using assms
    by (simp add: group-action.composition-rule)
  thus ∃ f ∈ carrier m. φ f y = φ h (φ g y)
    using assms carrier-g carrier-h group-action.group-hom
      group-hom.axioms(1) monoid.m-closed
    unfolding group-def
    by metis
qed

end

```

## 3.2 Quotients of Equivalence Relations on Election Sets

```

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

```

### 3.2.1 Auxiliary Lemmas

```

lemma obtain-partition:

```

```

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

```

$\text{fin-}X$ : *finite*  $X$  **and**  
 $\text{card-}X$ :  $\text{sum } N \ Y = \text{card } X$  **and**  
 $\text{hyp}$ :  
 $\bigwedge Y \ (X :: 'x \text{ set}).$   
 $x = \text{card } Y \implies$   
 $\text{finite } X \implies$   
 $\text{finite } Y \implies$   
 $\text{sum } N \ Y = \text{card } X \implies$   
 $\exists \mathcal{X}.$   
 $X = \bigcup \{ \mathcal{X} \ i \mid i. i \in Y \} \wedge$   
 $(\forall i \in Y. \text{card } (\mathcal{X} \ i) = N \ i) \wedge$   
 $(\forall i \ j. i \neq j \longrightarrow i \in Y \wedge j \in Y \longrightarrow \mathcal{X} \ i \cap \mathcal{X} \ j = \{\})$

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

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

### 3.2.2 Anonymity Quotient: Grid

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

— Here, we count the occurrences of a ballot per election in a set of elections for which the occurrences of the ballot per election coincide for all elections in the set.

**fun** *vote-count<sub>Q</sub>* :: *'a Preference-Relation*  $\Rightarrow$  (*'a, 'v Election set*  $\Rightarrow$  *nat* **where**  
*vote-count<sub>Q</sub> p =  $\pi_Q$  (vote-count p)*

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



$\text{anonymity-class } X = (\chi \ p. \text{ vote-count}_{\mathcal{Q}} (\text{ord2pref } p) \ X)$

**lemma** *anon-rel-equiv*:

*equiv* (*elections-A UNIV*) (*anonymity<sub>R</sub> (elections-A UNIV)*)

**proof** –

**have** *subset*: *elections-A UNIV*  $\subseteq$  *valid-elections*

**by** *simp*

**have** *equiv valid-elections* (*anonymity<sub>R</sub> valid-elections*)

**using** *rel-ind-by-group-act-equiv*[*of anonymity<sub>G</sub> valid-elections  $\varphi$ -anon valid-elections*]  
*rel-ind-by-coinciding-action-on-subset-eq-restr*

**by** (*simp add: anonymous-group-action.group-action-axioms*)

**moreover have**

$\forall \pi \in \text{carrier anonymity}_{\mathcal{G}}.$

$\forall E \in \text{elections-A UNIV}.$

$\varphi\text{-anon} (\text{elections-A UNIV}) \ \pi \ E = \varphi\text{-anon valid-elections } \pi \ E$

**using** *subset*

**unfolding**  *$\varphi$ -anon.simps*

**by** *simp*

**ultimately show** *?thesis*

**using** *subset equiv-rel-restr rel-ind-by-coinciding-action-on-subset-eq-restr*[*of*  
*elections-A UNIV valid-elections carrier anonymity<sub>G</sub>*  
 *$\varphi$ -anon (elections-A UNIV)*]

**unfolding** *anonymity<sub>R</sub>.simps*

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

**qed**

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

**theorem** *anonymity<sub>Q</sub>-isomorphism*:

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

**shows** *bij-betw* (*anonymity-class::('a::finite, 'v) Election set  $\Rightarrow$  nat<sup>\*</sup>('a Ordered-Preference)*)  
*(anonymity<sub>Q</sub> (UNIV::'a set)) (UNIV::(nat<sup>\*</sup>('a Ordered-Preference))*

*set)*

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

**fix**

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

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

**assume**

*class-X*:  $X \in \text{anonymity}_{\mathcal{Q}} \text{ UNIV}$  **and**

*class-Y*:  $Y \in \text{anonymity}_{\mathcal{Q}} \text{ UNIV}$  **and**

*eq-vec*: *anonymity-class*  $X = \text{anonymity-class } Y$

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

by *simp*  
 hence  $\forall (E, E') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}). \text{finite} (\text{voters-}\mathcal{E} E)$   
 by *simp*  
 moreover have *subset: elections-}\mathcal{A} \text{ UNIV} \subseteq \text{valid-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. (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}\_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})]*  
*vote-count-invar anon-rel-equiv*  
 unfolding *anonymity}\_{\mathcal{Q}}.\text{sims anonymity}\_{\mathcal{R}}.\text{sims vote-count}\_{\mathcal{Q}}.\text{sims}*  
 by *metis*  
 moreover from *anon-rel-equiv*  
 obtain  
 $E :: ('a, 'v) \text{ Election}$  and  
 $E' :: ('a, 'v) \text{ Election}$  where  
 $E\text{-in-}X: E \in X$  and  
 $E'\text{-in-}Y: E' \in Y$   
 using *class-X class-Y equiv-Eps-in*  
 unfolding *anonymity}\_{\mathcal{Q}}.\text{sims}*  
 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.sims*  
 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}\_{\mathcal{Q}}.\text{sims}*  
 using *in-quotient-imp-subset*  
 by *blast*

hence *eq-alt*s: *alternatives- $\mathcal{E}$*   $E = UNIV \wedge \text{alternatives-}\mathcal{E} \ E' = UNIV$   
 using *E-in-X E'-in-Y*  
 unfolding *elections- $\mathcal{A}$ .sims*  
 by *blast*  
 with *subset-fixed-alt*s **have** *eq-complement*:  
 $\forall p \in UNIV - \{p. \text{linear-order-on } (UNIV::'a \text{ set}) \ p\}.$   
 $\{v \in \text{voters-}\mathcal{E} \ E. \text{profile-}\mathcal{E} \ E \ v = p\} = \{\} \wedge \{v \in \text{voters-}\mathcal{E} \ E'. \text{profile-}\mathcal{E} \ E' \ v$   
 $= p\} = \{\}$   
 using *E-in-X E'-in-Y*  
 unfolding *elections- $\mathcal{A}$ .sims valid-elections-def profile-def*  
 by *auto*  
 hence  $\forall p \in UNIV - \{p. \text{linear-order-on } (UNIV::'a \text{ set}) \ p\}.$   
 $\text{vote-count } p \ E = 0 \wedge \text{vote-count } p \ E' = 0$   
 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-alt*s *E-in-X E'-in-Y*  
**have** *finite* (*voters- $\mathcal{E}$*   $E$ )  $\wedge$  *finite* (*voters- $\mathcal{E}$*   $E'$ )  
 unfolding *elections- $\mathcal{A}$ .sims*  
 by *blast*  
 moreover from *subset-fixed-alt*s *E-in-X E'-in-Y*  
**have**  $(E, E') \in (\text{elections-}\mathcal{A} \ UNIV) \times (\text{elections-}\mathcal{A} \ UNIV)$   
 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**  $(E, E') \in \text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \ UNIV)$   
 using *eq-alt*s *vote-count-anon-rel*  
 by *metis*  
 hence *anonymity $_{\mathcal{R}}$*  (*elections- $\mathcal{A}$*   $UNIV$ ) “ $\{E\} =$   
 $\text{anonymity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \ UNIV) \text{ “}\{E'\}$   
 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 $_{\mathcal{Q}}$ .sims*  
 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 $_{\mathcal{Q}}$ .sims*  
 by (*metis* (*no-types*, *lifting*))  
 finally **show**  $X = Y$   
 by *simp*  
 next  
**have**  $(UNIV::(\text{nat}, 'a \text{ Ordered-Preference}) \text{ vec set})) \subseteq$

```

    (anonymity-class::('a, 'v) Election set  $\Rightarrow$  (nat, 'a Ordered-Preference) vec) ‘
    anonymityQ 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  $\leq$  sum ( $\lambda$  i. x$i) 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. card (X' i) = x$i and
    partition': V =  $\bigcup$  {X' i | i. i  $\in$  UNIV} and
    disjoint':  $\forall$  i j. i  $\neq$  j  $\longrightarrow$  X' i  $\cap$  X' j = {}
    using obtain-partition[of V UNIV ($) x]
    by auto
  obtain X :: 'a Preference-Relation  $\Rightarrow$  'v set where
    def-X: X = ( $\lambda$  i. if (i  $\in$  {i. linear-order i}) then X' (pref2ord i) 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 | i. i  $\in$  UNIV} =
      {X i | i. i  $\in$  {i. linear-order i}}  $\cup$  {X i | i. i  $\in$  UNIV - {i. linear-order
i}}
    by blast
  ultimately have
    {X i | i. i  $\in$  UNIV} = {X' (pref2ord i) | i. i  $\in$  {i. linear-order i}}  $\vee$ 
      {X i | i. i  $\in$  UNIV} = {X' (pref2ord i) | i. i  $\in$  {i. linear-order i}}  $\cup$  {}
    by auto
  also have {X' (pref2ord i) | i. i  $\in$  {i. linear-order i}} = {X' i | i. i  $\in$  UNIV}
    using iso-tuple-UNIV-I pref2ord-cases
    by metis
  finally have
    {X i | i. i  $\in$  UNIV} = {X' i | i. i  $\in$  UNIV}  $\vee$ 

```

$\{X\ i \mid i. i \in UNIV\} = \{X'\ i \mid i. i \in UNIV\} \cup \{\{\}\}$   
 by *simp*  
 hence  $\bigcup \{X\ i \mid i. i \in UNIV\} = \bigcup \{X'\ i \mid i. i \in UNIV\}$   
 using *Sup-union-distrib ccpo-Sup-singleton sup-bot.right-neutral*  
 by (*metis (no-types, lifting)*)  
 hence *partition*:  $V = \bigcup \{X\ i \mid i. i \in UNIV\}$   
 using *partition'*  
 by *simp*  
 moreover have  $\forall\ i\ j. i \neq j \longrightarrow X\ i \cap X\ j = \{\}$   
 using *disjoint' def-X pref2ord-inject*  
 by *auto*  
 ultimately have  $\forall\ v \in V. \exists!\ i. v \in X\ i$   
 by *auto*  
 then obtain  $p' :: 'v \Rightarrow 'a\ Preference-Relation$  where  
 $p-X: \forall\ v \in V. v \in X\ (p'\ v)$  and  
 $p-disj: \forall\ v \in V. \forall\ i. i \neq p'\ v \longrightarrow v \notin X\ i$   
 by *metis*  
 then obtain  $p :: 'v \Rightarrow 'a\ Preference-Relation$  where  
 $p-def: p = (\lambda\ v. \text{if } v \in V \text{ then } p'\ v \text{ else } \{\})$   
 by *simp*  
 hence *lin-ord*:  $\forall\ v \in V. \text{linear-order } (p\ v)$   
 using *def-X p-X p-disj*  
 by *fastforce*  
 hence *valid*:  $(UNIV, V, p) \in \text{elections-}\mathcal{A}\ UNIV$   
 using *fin-V*  
 unfolding *p-def elections-}\mathcal{A}.simps valid-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 *anon-rel-vote-count[of (UNIV, V, p) - elections-}\mathcal{A}\ UNIV]*  
 $\text{fin-V}$   
 by *simp*  
 moreover have  $(UNIV, V, p) \in \text{anonymity}_{\mathcal{R}}\ (\text{elections-}\mathcal{A}\ UNIV) \text{ “ } \{(UNIV,$   
 $V, p)\}$   
 using *anon-rel-equiv valid*  
 unfolding *Image-def equiv-def refl-on-def*  
 by *blast*  
 ultimately have *eq-vote-count*:  
 $\forall\ i. \text{vote-count } i\ \text{“ } (\text{anonymity}_{\mathcal{R}}\ (\text{elections-}\mathcal{A}\ UNIV) \text{ “ } \{(UNIV, V, p)\}) =$   
 $\{\text{vote-count } i\ (UNIV, V, p)\}$   
 by *blast*  
 have  $\forall\ i. \forall\ v \in V. p\ v = i \longleftrightarrow v \in X\ i$   
 using *p-X p-disj*  
 unfolding *p-def*  
 by *metis*  
 hence  $\forall\ i. \{v \in V. p\ v = i\} = \{v \in V. v \in X\ i\}$   
 by *blast*  
 moreover have  $\forall\ i. X\ i \subseteq V$   
 using *partition*

by *blast*  
 ultimately have *rewr-preimg*:  $\forall i. \{v \in V. p \ v = i\} = X \ i$   
 by *auto*  
 hence  $\forall i \in \{i. \text{linear-order } i\}. \text{vote-count } i \ (UNIV, V, p) = x\$(\text{pref2ord } i)$   
 using *def-X card'*  
 by *simp*  
 hence  $\forall i \in \{i. \text{linear-order } i\}.$   
 $\text{vote-count } i \ ' (anonymity_{\mathcal{R}} \ (\text{elections-}\mathcal{A} \ UNIV) \ \{\{(UNIV, V, p)\}\}) =$   
 $\{x\$(\text{pref2ord } i)\}$   
 using *eq-vote-count*  
 by *metis*  
 hence  
 $\forall i \in \{i. \text{linear-order } i\}.$   
 $\text{vote-count}_{\mathcal{Q}} \ i \ (anonymity_{\mathcal{R}} \ (\text{elections-}\mathcal{A} \ UNIV) \ \{\{(UNIV, V, p)\}\}) =$   
 $x\$(\text{pref2ord } i)$   
 unfolding *vote-count<sub>Q</sub>.simps*  *$\pi_{\mathcal{Q}}$ .simps* *singleton-set.simps*  
 using *is-singleton-altdef* *singleton-set-def-if-card-one*  
 by *fastforce*  
 hence  $\forall i. \text{vote-count}_{\mathcal{Q}} \ (\text{ord2pref } i) \ (anonymity_{\mathcal{R}} \ (\text{elections-}\mathcal{A} \ UNIV) \ \{\{(UNIV, V, p)\}\})$   
 $= x\$i$   
 using *ord2pref ord2pref-inverse*  
 by *metis*  
 hence *anonymity-class*  $(anonymity_{\mathcal{R}} \ (\text{elections-}\mathcal{A} \ UNIV) \ \{\{(UNIV, V, p)\}\})$   
 $= x$   
 using *anonymity-class.simps* *vec-lambda-unique*  
 by *(metis (no-types, lifting))*  
 moreover have  
 $anonymity_{\mathcal{R}} \ (\text{elections-}\mathcal{A} \ UNIV) \ \{\{(UNIV, V, p)\}\} \in anonymity_{\mathcal{Q}} \ UNIV$   
 using *valid*  
 unfolding *anonymity<sub>Q</sub>.simps* *quotient-def*  
 by *blast*  
 ultimately show  
 $x \in (\lambda X::('a, 'v) \text{ Election set}). \chi \ p. \text{vote-count}_{\mathcal{Q}} \ (\text{ord2pref } p) \ X) \ ' anonymity_{\mathcal{Q}}$   
 $UNIV$   
 using *anonymity-class.elims*  
 by *blast*  
 qed  
 thus  $(anonymity\text{-}class::('a, 'v) \text{ Election set} \Rightarrow (nat, 'a \text{ Ordered-Preference}) \text{ vec})$   
 $,$   
 $anonymity_{\mathcal{Q}} \ UNIV = (UNIV::((nat, 'a \text{ Ordered-Preference}) \text{ vec set}))$   
 by *blast*  
 qed

### 3.2.3 Homogeneity Quotient: Simplex

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

then (Fract (vote-count  $r$   $E$ ) (card (voters- $\mathcal{E}$   $E$ ))) else 0)

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

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

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

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

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

**fun** rat-vector :: real $^b$   $\Rightarrow$  rat $^b$  **where**  
 rat-vector  $v$  = ( $\chi$   $p$ . the-inv of-rat ( $v\$p$ ))

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

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

The rational points in the simplex.

**definition** vote-simplex :: (rat $^b$ ) set **where**  
 vote-simplex  $\equiv \text{insert } 0 \ (\text{rat-vector-set } (\text{convex hull } (\text{standard-basis :: (real}^b\text{) set})))$

## Auxiliary Lemmas

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

**fixes**

$X :: (\text{real}^b)$  set **and**

$x :: \text{real}^b$

**assumes**  $\exists f :: (\text{real}^b) \Rightarrow \text{real}$ .

$\text{sum } f \ X = 1 \wedge (\forall x \in X. f \ x \geq 0) \wedge x = \text{sum } (\lambda x. (f \ x) *_{\mathbb{R}} x) \ X$

**shows**  $x \in \text{convex hull } X$

**using** *assms*

**proof** (induction card  $X$  arbitrary:  $X \ x$ )

**case** 0

**fix**

$X :: (\text{real}^b)$  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  $(\text{Suc } n)$ 
fix
   $X :: (\text{real}^b) \text{ set}$  and
   $x :: \text{real}^b$  and
   $n :: \text{nat}$ 
assume
  card: Suc n = 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
  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
  sum: sum f X = 1 and
  nonneg:  $\forall x \in X. 0 \leq f x$  and
  x-sum:  $x = (\sum x \in X. f x *_R x)$ 
  by blast
have  $\text{card } X > 0$ 
  using card
  by linarith
hence fin: 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 finite.emptyI sum.insert sum.neutral
  by (metis (no-types, opaque-lifting))
hence  $n = 0 \longrightarrow (\exists y. X = \{y\} \wedge x = y)$ 
  using x-sum
  by fastforce
hence  $n = 0 \longrightarrow x \in X$ 
  by blast
moreover have  $n > 0 \longrightarrow x \in \text{convex hull } X$ 

```



```

proof (safe)
  assume  $0 < n$ 
  hence card-X-gt-1:  $\text{card } X > 1$ 
    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-1
    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
    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**  $\forall x::(\text{real}^b). \forall y. \forall u v.$

$sum ((\$) (u *_R x + v *_R y)) UNIV = sum ((\$) (u *_R x)) UNIV + sum ((\$) (v *_R y)) UNIV$   
**by** *blast*  
**moreover have**  $\forall x u. sum ((\$) (u *_R x)) UNIV = u *_R (sum ((\$) x) UNIV)$   
**using** *scaleR-right.sum sum.cong vector-scaleR-component*  
**by** *(metis (mono-tags, lifting))*  
**ultimately have**  $\forall x::(real^b). \forall y. \forall u v.$   
 $sum ((\$) (u *_R x + v *_R y)) UNIV = u *_R (sum ((\$) x) UNIV) + v *_R (sum ((\$) y) UNIV)$   
**by** *(metis (no-types))*  
**moreover have**  $\forall x \in ?simplex. sum ((\$) x) UNIV = 1$   
**by** *simp*  
**ultimately have**  
 $\forall x \in ?simplex. \forall y \in ?simplex. \forall u v. sum ((\$) (u *_R x + v *_R y)) UNIV =$   
 $u *_R 1 + v *_R 1$   
**by** *(metis (no-types, lifting))*  
**hence**  $\forall x \in ?simplex. \forall y \in ?simplex. \forall u v. sum ((\$) (u *_R x + v *_R y)) UNIV = u + v$   
**by** *simp*  
**moreover have**  
 $\forall x \in ?simplex. \forall y \in ?simplex. \forall u \geq 0. \forall v \geq 0.$   
 $u + v = 1 \longrightarrow (\forall i. (u *_R x + v *_R y)\$i \geq 0)$   
**by** *simp*  
**ultimately have** *simplex-convex*:  
 $\forall x \in ?simplex. \forall y \in ?simplex. \forall u \geq 0. \forall v \geq 0.$   
 $u + v = 1 \longrightarrow u *_R x + v *_R y \in ?simplex$   
**by** *simp*  
**have** *entries*:  $\forall v::(real^b) \in standard-basis. \exists b. v\$b = 1 \wedge (\forall c. c \neq b \longrightarrow v\$c = 0)$   
**unfolding** *standard-basis-def*  
**by** *simp*  
**then obtain** *one* ::  $real^b \Rightarrow 'b$  **where**  
*def*:  $\forall v \in standard-basis. v\$(one\ v) = 1 \wedge (\forall i \neq one\ v. v\$i = 0)$   
**by** *metis*  
**hence**  $\forall v::(real^b) \in standard-basis. \forall b. v\$b = 0 \vee v\$b = 1$   
**by** *metis*  
**hence** *geq-0*:  $\forall v::(real^b) \in standard-basis. \forall b. v\$b \geq 0$   
**using** *dual-order.refl zero-less-one-class.zero-le-one*  
**by** *metis*  
**moreover have**  $\forall v::(real^b) \in standard-basis.$   
 $sum ((\$) v) UNIV = sum ((\$) v) (UNIV - \{one\ v\}) + v\$(one\ v)$   
**unfolding** *def*  
**using** *add commute finite insert-UNIV sum.insert-remove*  
**by** *metis*  
**moreover have**  $\forall v \in standard-basis. sum ((\$) v) (UNIV - \{one\ v\}) + v\$(one\ v) = 1$   
**using** *def*  
**by** *simp*  
**ultimately have** *standard-basis*  $\subseteq ?simplex$

by force  
 with simplex-convex  
 have ?simplex  $\in$   
 $\{t. (\forall x \in t. \forall y \in t. \forall u \geq 0. \forall v \geq 0. u + v = 1 \longrightarrow u *_R x + v *_R y \in$   
 $t) \wedge \text{standard-basis} \subseteq t\}$   
 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  $\{v. (\forall i. 0 \leq v \$ i) \wedge \text{sum } ((\$) v) \text{ UNIV} = 1\} \subseteq$   
 $\bigcap \{t. (\forall x \in t. \forall y \in t. \forall u \geq 0. \forall v \geq 0. u + v = 1 \longrightarrow u *_R x + v *_R$   
 $y \in t) \wedge (\text{standard-basis}::(\text{real}^\sim b) \text{ set}) \subseteq t\}$   
 proof (intro subsetI)  
 fix  
 $x :: \text{real}^\sim b$  and  
 $X :: (\text{real}^\sim b) \text{ set}$   
 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  $:: (\text{real}^\sim b) \Rightarrow 'b$  where  
 ind-1:  $\forall v \in \text{standard-basis}. v \$ (\text{ind } v) = 1$  and  
 ind-0:  $\forall v \in \text{standard-basis}. \forall b \neq (\text{ind } v). v \$ b = 0$   
 by metis  
 hence  $\forall v v'. v \in \text{standard-basis} \wedge v' \in \text{standard-basis} \longrightarrow \text{ind } v = \text{ind } v'$   
 $\longrightarrow (\forall b. v \$ b = v' \$ b)$   
 by metis  
 hence inj-ind:  
 $\forall v v'. v \in \text{standard-basis} \wedge v' \in \text{standard-basis} \longrightarrow \text{ind } v = \text{ind } v' \longrightarrow v =$   
 $v'$   
 unfolding vec-eq-iff  
 by simp  
 hence inj-on ind standard-basis  
 unfolding inj-on-def  
 by blast  
 hence bij: bij-betw ind standard-basis (ind ` standard-basis)  
 unfolding bij-betw-def  
 by simp  
 obtain ind-inv  $:: 'b \Rightarrow (\text{real}^\sim b)$  where  
 char-vec:  $\text{ind-inv} = (\lambda b. (\chi i. \text{if } i = b \text{ then } 1 \text{ else } 0))$   
 by blast  
 hence in-basis:  $\forall b. \text{ind-inv } b \in \text{standard-basis}$   
 unfolding standard-basis-def  
 by simp  
 moreover from this

**have** *ind-inv-map*:  $\forall b. \text{ind} (\text{ind-inv } b) = b$   
**using** *char-vec ind-0 ind-1 axis-def axis-nth zero-neq-one*  
**by** *metis*  
**ultimately have**  $\forall b. \exists v. v \in \text{standard-basis} \wedge b = \text{ind } v$   
**by** *metis*  
**hence** *univ*:  $\text{ind} \text{ ` standard-basis} = \text{UNIV}$   
**by** *blast*  
**have** *bij-inv*: *bij-betw ind-inv UNIV standard-basis*  
**using** *ind-inv-map bij bij-betw-byWitness[of UNIV ind] in-basis inj-ind*  
**unfolding** *image-subset-iff*  
**by** *simp*  
**obtain**  $f :: (\text{real}^b) \Rightarrow \text{real}$  **where**  
 $\text{def: } f = (\lambda v. \text{if } v \in \text{standard-basis} \text{ then } x\$(\text{ind } v) \text{ else } 0)$   
**by** *blast*  
**hence**  $\text{sum } f \text{ standard-basis} = \text{sum } (\lambda v. x\$(\text{ind } v)) \text{ standard-basis}$   
**by** *simp*  
**also have**  $\text{sum } (\lambda v. x\$(\text{ind } v)) \text{ standard-basis} = \text{sum } ((\$) x \circ \text{ind}) \text{ standard-basis}$   
**unfolding** *comp-def*  
**by** *simp*  
**also have**  $\dots = \text{sum } ((\$) x) (\text{ind} \text{ ` standard-basis})$   
**using** *sum-comp[of ind standard-basis ind ` standard-basis (\$) x] bij*  
**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-1*:  $\text{sum } f \text{ standard-basis} = 1$   
**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-1 ind-0*  
**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  $\text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis}$   
      $= \text{sum } (\lambda v. (x\$(\text{ind } v)) *_{\mathbb{R}} v) \text{ standard-basis}$   
 unfolding *def*  
 by *simp*  
 ultimately have  $\text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis}$   
      $= \text{sum } (\lambda v. (\chi i. \text{if } i = \text{ind } v \text{ then } x\$(\text{ind } v) \text{ else } 0)) \text{ standard-basis}$   
 by *force*  
 also have ...  $= \text{sum } (\lambda b. (\chi i. \text{if } i = \text{ind } (\text{ind-inv } b) \text{ then } x\$(\text{ind } (\text{ind-inv } b))$   
 else 0)) *UNIV*  
 using *bij-inv sum-comp*  
 unfolding *comp-def*  
 by *blast*  
 also have ...  $= \text{sum } (\lambda b. (\chi i. \text{if } i = b \text{ then } x\$b \text{ else } 0)) \text{ UNIV}$   
 using *ind-inv-map*  
 by *presburger*  
 finally have  $\text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis}$   
      $= \text{sum } (\lambda b. (\chi i. \text{if } i = b \text{ then } x\$b \text{ else } 0)) \text{ UNIV}$   
 by *simp*  
 moreover have  $\forall b. (\text{sum } (\lambda b'. (\chi i. \text{if } i = b \text{ then } x\$b \text{ else } 0)) \text{ UNIV})\$b$   
      $= \text{sum } (\lambda b'. (\chi i. \text{if } i = b' \text{ then } x\$b' \text{ else } 0))\$b \text{ UNIV}$   
 using *sum-component*  
 by *blast*  
 moreover have  $\forall b. (\lambda b'. (\chi i. \text{if } i = b' \text{ then } x\$b' \text{ else } 0))\$b$   
      $= (\lambda b'. \text{if } b' = b \text{ then } x\$b \text{ else } 0)$   
 by *force*  
 moreover have  $\forall b. \text{sum } (\lambda b'. \text{if } b' = b \text{ then } x\$b \text{ else } 0) \text{ UNIV}$   
      $= x\$b + \text{sum } (\lambda b'. 0) (\text{UNIV} - \{b\})$   
 by *simp*  
 ultimately have  $\forall b. (\text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis})\$b = x\$b$   
 by *simp*  
 hence  $\text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis} = x$   
 unfolding *vec-eq-iff*  
 by *simp*  
 hence  $\exists f::(\text{real}^b) \Rightarrow \text{real}.$   
      $\text{sum } f \text{ standard-basis} = 1 \wedge (\forall x \in \text{standard-basis}. f x \geq 0)$   
      $\wedge x = \text{sum } (\lambda x. (f x) *_{\mathbb{R}} x) \text{ standard-basis}$   
 using *sum-1 nonneg*  
 by *blast*  
 hence  $x \in \text{convex hull } (\text{standard-basis}::(\text{real}^b \text{ set}))$   
 using *convex-combination-in-convex-hull*  
 by *blast*  
 thus  $x \in \bigcap \{t. (\forall x \in t. \forall y \in t. \forall u \geq 0. \forall v \geq 0. u + v = 1 \longrightarrow u *_{\mathbb{R}} x$   
 +  $v *_{\mathbb{R}} y \in t)$   
      $\wedge (\text{standard-basis}::(\text{real}^b \text{ set})) \subseteq t\}$   
 unfolding *convex-def hull-def*  
 by *blast*  
 qed  
 qed

```

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

lemma anonymity-homogeneity-is-equivalence:
  fixes  $X :: ('a, 'v) \text{ Election set}$ 
  assumes  $\forall E \in X. \text{finite } (\text{voters-}\mathcal{E} \ E)$ 
  shows  $\text{equiv } X \ (\text{anonymity-homogeneity}_{\mathcal{R}} \ X)$ 
proof (unfold equiv-def, safe)
  show  $\text{refl-on } X \ (\text{anonymity-homogeneity}_{\mathcal{R}} \ X)$ 
    unfolding refl-on-def anonymity-homogeneityR.simps
    by blast
next
  show  $\text{sym } (\text{anonymity-homogeneity}_{\mathcal{R}} \ X)$ 
    unfolding sym-def anonymity-homogeneityR.simps
    using sup-commute
    by simp
next
  show  $\text{Relation.trans } (\text{anonymity-homogeneity}_{\mathcal{R}} \ X)$ 
proof
  fix
     $E :: ('a, 'v) \text{ Election}$  and
     $E' :: ('a, 'v) \text{ Election}$  and
     $F :: ('a, 'v) \text{ Election}$ 
  assume
     $\text{rel}: (E, E') \in \text{anonymity-homogeneity}_{\mathcal{R}} \ X$  and
     $\text{rel}': (E', F) \in \text{anonymity-homogeneity}_{\mathcal{R}} \ X$ 
  hence  $\text{fin}: \text{finite } (\text{voters-}\mathcal{E} \ E')$ 
    unfolding anonymity-homogeneityR.simps
    using assms
    by fastforce
  from  $\text{rel rel'}$  have eq-frac:
     $(\forall r. \text{vote-fraction } r \ E = \text{vote-fraction } r \ E') \wedge$ 
     $(\forall r. \text{vote-fraction } r \ E' = \text{vote-fraction } r \ F)$ 
    unfolding anonymity-homogeneityR.simps
    by blast
  hence  $\forall r. \text{vote-fraction } r \ E = \text{vote-fraction } r \ F$ 
    by metis
  thus  $(E, F) \in \text{anonymity-homogeneity}_{\mathcal{R}} \ X$ 
    using  $\text{rel rel'}$  snd-conv
    unfolding anonymity-homogeneityR.simps
    by blast

```

qed  
qed

**lemma** *fract-distr*:

**fixes**

$A :: 'x \text{ set}$  **and**  
 $f :: 'x \Rightarrow \text{int}$  **and**  
 $b :: \text{int}$

**assumes**

*finite*  $A$  **and**  
 $b \neq 0$

**shows**  $\text{sum } (\lambda a. \text{Fract } (f a) b) A = \text{Fract } (\text{sum } f A) b$

**using** *assms*

**proof** (*induction card A arbitrary: A f b*)

**case** 0

**fix**

$A :: 'x \text{ set}$  **and**  
 $f :: 'x \Rightarrow \text{int}$  **and**  
 $b :: \text{int}$

**assume**

$0 = \text{card } A$  **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*:  $\text{Suc } n = \text{card } A$  **and**  
*fin-A*: *finite*  $A$  **and**  
*b-non-zero*:  $b \neq 0$  **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*:  $c \in A$

**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 ... = Fract (sum f (A - {c})) b + Fract (f c) b
    using hyp card-A fin-A b-non-zero c-in-A Diff-empty card-Diff-singleton
      diff-Suc-1 finite-Diff-insert
    by metis
  also have ... = Fract (sum f (A - {c}) + f c) b
    using c-in-A b-non-zero fract-distr-helper
    by metis
  also have ... = Fract (sum f A) b
    using c-in-A fin-A
    by (simp add: sum-diff1)
  finally show ( $\sum a \in A. \text{Fract } (f a) b = \text{Fract } (\text{sum } f A) b$ )
    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<sub>Q</sub>-isomorphism:*

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

**shows**

*bij-betw (anonymity-homogeneity-class::('a::finite, 'v) Election set  $\Rightarrow$   
 rat $\sim$ ('a Ordered-Preference)) (anonymity-homogeneity<sub>Q</sub> (UNIV::'a set))  
 (vote-simplex :: (rat $\sim$ ('a Ordered-Preference)) set)*

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

**fix**

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

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

**assume**

*class-X: X  $\in$  anonymity-homogeneity<sub>Q</sub> UNIV and*

*class-Y: Y  $\in$  anonymity-homogeneity<sub>Q</sub> UNIV and*

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

**have** *equiv: equiv (elections-A UNIV) (anonymity-homogeneity<sub>R</sub> (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 $_{\mathcal{Q}}$ .simps
by blast
then obtain  $E :: ('a, 'v)$  Election and
       $E' :: ('a, 'v)$  Election where
       $E$ -in- $X$ :  $E \in X$  and
       $E'$ -in- $Y$ :  $E' \in Y$ 
by blast
hence class- $X$ - $E$ : anonymity-homogeneity $_{\mathcal{R}}$  (elections- $\mathcal{A}$  UNIV) “ $\{E\} = X$ ”
using class- $X$  equiv Image-singleton-iff equiv-class-eq quotient $E$ 
unfolding anonymity-homogeneity $_{\mathcal{Q}}$ .simps
by (metis (no-types, opaque-lifting))
hence  $\forall F \in X. (E, F) \in$  anonymity-homogeneity $_{\mathcal{R}}$  (elections- $\mathcal{A}$  UNIV)
unfolding Image-def
by blast
hence  $\forall F \in X. \forall p. \text{vote-fraction } p \ F = \text{vote-fraction } p \ E$ 
unfolding anonymity-homogeneity $_{\mathcal{R}}$ .simps
by fastforce
hence  $\forall p. \text{vote-fraction } p \ \text{' } X = \{\text{vote-fraction } p \ E\}$ 
using  $E$ -in- $X$ 
by blast
hence  $\forall p. \text{vote-fraction}_{\mathcal{Q}} \ p \ X = \text{vote-fraction } p \ E$ 
using is-singletonI singleton-set-def-if-card-one the-elem-eq
unfolding is-singleton-altdef vote-fraction $_{\mathcal{Q}}$ .simps  $\pi_{\mathcal{Q}}$ .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 quotient $E$ 
unfolding anonymity-homogeneity $_{\mathcal{Q}}$ .simps
by (metis (no-types, opaque-lifting))
hence  $\forall F \in Y. (E', F) \in$  anonymity-homogeneity $_{\mathcal{R}}$  (elections- $\mathcal{A}$  UNIV)
unfolding Image-def
by blast
hence  $\forall F \in Y. \forall p. \text{vote-fraction } p \ E' = \text{vote-fraction } p \ F$ 
unfolding anonymity-homogeneity $_{\mathcal{R}}$ .simps
by blast
hence  $\forall p. \text{vote-fraction } p \ \text{' } Y = \{\text{vote-fraction } p \ E'\}$ 
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-A.simps valid-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  $(E, E') \in \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-A UNIV})$ 
using subset E-in-X E'-in-Y elections-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  $(\text{anonymity-homogeneity-class}::('a, 'v) \text{ Election set} \Rightarrow \text{rat}^{\sim}('a \text{ Ordered-Preference}))$ 
 $\quad \text{'anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV} = \text{vote-simplex}$ 
proof (unfold vote-simplex-def, safe)
fix  $X :: ('a, 'v) \text{ Election set}$ 
assume
 $\text{quot: } X \in \text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV and}$ 
 $\text{not-simplex: anonymity-homogeneity-class } X \notin \text{rat-vector-set } (\text{convex hull}$ 
 $\text{standard-basis})$ 
have equiv-rel:
 $\text{equiv } (\text{elections-A UNIV}) (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-A UNIV}))$ 
using anonymity-homogeneity-is-equivalence[of elections-A UNIV] elections-A.simps
by blast
then obtain  $E :: ('a, 'v) \text{ Election where}$ 
 $E\text{-in-}X: E \in X \text{ and}$ 
 $X = \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-A UNIV}) \text{ `` } \{E\}$ 
using quot anonymity-homogeneity $_{\mathcal{Q}}$ .simps equiv-Eps-in proj-Eps
unfolding proj-def
by metis

```

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

```

    using eq-card
    by fastforce
  hence finite (voters- $\mathcal{E}$   $E$ )  $\wedge$  voters- $\mathcal{E}$   $E \neq \{\}$   $\longrightarrow$ 
    sum ( $\lambda p.$  Fract (vote-count  $p$   $E$ ) ?sum) UNIV = Fract ?sum ?sum
    using fract-distr[of UNIV ?sum  $\lambda p.$  int (vote-count  $p$   $E$ )]
      card-0-eq eq-card finite-class.finite-UNIV
      of-nat-eq-0-iff of-nat-sum sum.cong
    by (metis (no-types, lifting))
  moreover have finite (voters- $\mathcal{E}$   $E$ )  $\wedge$  voters- $\mathcal{E}$   $E \neq \{\}$   $\longrightarrow$  Fract ?sum ?sum
= 1
    using gt-0 One-rat-def eq-rat(1)[of ?sum 1 ?sum 1]
    by linarith
  ultimately have sum-1:
    finite (voters- $\mathcal{E}$   $E$ )  $\wedge$  voters- $\mathcal{E}$   $E \neq \{\}$   $\longrightarrow$  sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) UNIV
= 1
    by presburger
  have inv-of-rat:  $\forall x \in \mathbb{Q}.$  the-inv of-rat (of-rat  $x$ ) =  $x$ 
    unfolding Rats-def
    using the-inv-f-f injI of-rat-eq-iff
    by metis
  have  $E \in \text{elections-}\mathcal{A}$  UNIV
    using quot  $E$ -in- $X$  equiv-class-eq-iff equiv-rel rel
    unfolding anonymity-homogeneity $_{\mathbb{Q}}$ .simps quotient-def
    by fastforce
  hence  $\forall v \in \text{voters-}\mathcal{E}$   $E.$  linear-order (profile- $\mathcal{E}$   $E$   $v$ )
    unfolding elections- $\mathcal{A}$ .simps valid-elections-def profile-def
    by fastforce
  hence  $\forall p.$   $\neg$  linear-order  $p \longrightarrow$  vote-count  $p$   $E = 0$ 
    unfolding vote-count.simps
    using card.infinite card-0-eq
    by blast
  hence  $\forall p.$   $\neg$  linear-order  $p \longrightarrow$  vote-fraction  $p$   $E = 0$ 
    using rat-number-collapse
    by simp
  moreover have sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) UNIV =
    sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) { $p.$  linear-order  $p$ } +
    sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) (UNIV - { $p.$  linear-order  $p$ })
    using finite CollectD Collect-mono UNIV-I add commute sum.subset-diff
top-set-def
    by metis
  ultimately have sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) UNIV =
    sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) { $p.$  linear-order  $p$ }
    by simp
  moreover have bij-betw ord2pref UNIV { $p.$  linear-order  $p$ }
    using inj-def ord2pref-inject range-ord2pref
    unfolding bij-betw-def
    by blast
  ultimately have
    sum ( $\lambda p.$  vote-fraction  $p$   $E$ ) UNIV = sum ( $\lambda p.$  vote-fraction (ord2pref  $p$ )  $E$ )

```

$UNIV$   
**using** *comp-def*[*of*  $\lambda p. \text{vote-fraction } p \ E \ \text{ord2pref}$ ]  
 $\text{sum-comp}[\text{of } \text{ord2pref } UNIV \ \{p. \text{linear-order } p\} \ \lambda p. \text{vote-fraction } p \ E]$   
**by** *auto*  
**hence**  $\text{finite } (\text{voters-}\mathcal{E} \ E) \wedge \text{voters-}\mathcal{E} \ E \neq \{\} \longrightarrow$   
 $\text{sum } (\lambda p. \text{vote-fraction } (\text{ord2pref } p) \ E) \ UNIV = 1$   
**using** *sum-1*  
**by** *presburger*  
**hence**  $\text{finite } (\text{voters-}\mathcal{E} \ E) \wedge \text{voters-}\mathcal{E} \ E \neq \{\} \longrightarrow$   
 $\text{sum } (\lambda p. \text{real-of-rat } (\text{vote-fraction } (\text{ord2pref } p) \ E)) \ UNIV = 1$   
**using** *of-rat-1 of-rat-sum*  
**by** *metis*  
**with** *zero-case*  
**have**  $(\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) = 0 \vee$   
 $\text{sum } (\lambda p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \ UNIV = 1$   
**using** *repr*  
**by** *force*  
**hence**  $(\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) = 0 \vee$   
 $((\forall \ p. (\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \$ p \geq 0) \wedge$   
 $\text{sum } ((\$) (\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X))) \ UNIV = 1)$   
**using** *geq-0*  
**by** *force*  
**moreover have**  $\text{rat-entries: } \forall \ p. (\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \$ p \in \mathbb{Q}$   
**by** *simp*  
**ultimately have** *simplex-el:*  
 $(\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \in$   
 $\{x \in \text{insert } 0 \ (\text{convex hull standard-basis}). \forall \ i. x \$ i \in \mathbb{Q}\}$   
**using** *standard-simplex-rewrite*  
**by** *blast*  
**moreover have**  
 $\forall \ p. (\text{rat-vector } (\chi \ p. \text{of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X))) \$ p$   
 $= \text{the-inv real-of-rat } ((\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \$ p)$   
**unfolding** *rat-vector.simps*  
**using** *vec-lambda-beta*  
**by** *blast*  
**moreover have**  
 $\forall \ p. \text{the-inv real-of-rat } ((\chi \ p. \text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) \$ p)$   
 $=$   
 $\text{the-inv real-of-rat } (\text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X))$   
**by** *simp*  
**moreover have**  
 $\forall \ p. \text{the-inv real-of-rat } (\text{real-of-rat } (\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X)) =$   
 $\text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X$   
**using** *rat-entries inv-of-rat Rats-eq-range-nat-to-rat-surj surj-nat-to-rat-surj*  
**by** *blast*  
**moreover have**  $\forall \ p. \text{vote-fraction}_{\mathbb{Q}} (\text{ord2pref } p) \ X = (\text{anonymity-homogeneity-class } X) \$ p$   
**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**  $\exists x \in \{x \in \text{insert } 0 (\text{convex hull standard-basis}). \forall i. x \$ i \in \mathbb{Q}\}.$   
 $\text{rat-vector } x = \text{anonymity-homogeneity-class } X$   
**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*:  
 $(UNIV, \{\}, \lambda v. \{\}) \in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ } UNIV)) \text{ “ } \{(UNIV, \{\}, \lambda v. \{\})\}$   
**unfolding** *anonymity-homogeneity<sub>R</sub>.simps Image-def elections- $\mathcal{A}$ .simps*  
*valid-elections-def profile-def*  
**by** *simp*  
**have** *in-els*:  $(UNIV, \{\}, \lambda v. \{\}) \in \text{elections-}\mathcal{A} \text{ } UNIV$   
**unfolding** *elections- $\mathcal{A}$ .simps valid-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** *anonymity-homogeneity<sub>R</sub>.simps*  
**by** *auto*  
**moreover have**  
 $\forall E \in (\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ } UNIV)) \text{ “ } \{(UNIV, \{\}, (\lambda v. \{\}))\}.$   
*finite (voters- $\mathcal{E}$  E)*  
**unfolding** *Image-def anonymity-homogeneity<sub>R</sub>.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**  $\forall r. 0 \in$   
 $\text{vote-fraction } r \text{ ‘ (anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV,$   
 $\{\}, (\lambda v. \{\})\}\}$   
**using** *non-empty image-eqI*  
**by** (*metis (mono-tags, lifting)*)  
**hence**  $\forall r. \{0\} \subseteq \text{vote-fraction } r \text{ ‘}$   
 $(\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\}$   
**by** *blast*  
**moreover have**  $\forall r. \{0\} \supseteq \text{vote-fraction } r \text{ ‘}$   
 $(\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\}$   
**using** *all-zero*  
**by** *blast*  
**ultimately have**  $\forall r.$   
 $\text{vote-fraction } r \text{ ‘ (anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV,$   
 $\{\}, \lambda v. \{\})\}\} = \{0\}$   
**by** *blast*  
**hence**  
 $\forall r.$   
 $\text{card (vote-fraction } r$   
 $\text{ ‘ (anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\})$   
 $= 1$   
 $\wedge \text{the-inv } (\lambda x. \{x\})$   
 $(\text{vote-fraction } r \text{ ‘}$   
 $(\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\})$   
 $= 0$   
**using** *is-singletonI singleton-insert-inj-eq' singleton-set-def-if-card-one*  
**unfolding** *is-singleton-altdef singleton-set.simps*  
**by** *metis*  
**hence**  
 $\forall r. \text{vote-fraction}_{\mathcal{Q}} r$   
 $(\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\} =$   
 $0$   
**unfolding** *vote-fraction<sub>Q</sub>.simps  $\pi_{\mathcal{Q}}$ .simps singleton-set.simps*  
**by** *metis*  
**hence**  $\forall r::('a \text{ Ordered-Preference}). \text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } r)$   
 $(\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\}$   
 $= 0$   
**by** *metis*  
**hence**  $\forall r::('a \text{ Ordered-Preference}).$   
 $(\text{anonymity-homogeneity-class } ((\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV))$   
 $\text{ ‘ ‘}\{(UNIV, \{\}, \lambda v. \{\})\}\}))\$r = 0$   
**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**  $\forall r::('a \text{ Ordered-Preference}).$   
 $(\text{anonymity-homogeneity-class}$   
 $((\text{anonymity-homogeneity}_{\mathcal{R}} \text{ (elections-}\mathcal{A} \text{ UNIV)) ‘ ‘}\{(UNIV, \{\}, \lambda v.$



```

{})))))$r
  = (0::(rat ^('a Ordered-Preference)))$r
  by (metis (no-types))
hence anonymity-homogeneity-class
  ((anonymity-homogeneityR (elections- $\mathcal{A}$  UNIV) “ {(UNIV, {},  $\lambda$  v. {})}))
  = (0::(rat ^('a Ordered-Preference)))
  using vec-eq-iff
  by blast
moreover have
  (anonymity-homogeneityR (elections- $\mathcal{A}$  UNIV) “ {(UNIV, {},  $\lambda$  v. {})}))
     $\in$  anonymity-homogeneityQ UNIV
  unfolding anonymity-homogeneityQ.simps quotient-def
  using in-els
  by blast
ultimately show (0::(rat ^('a Ordered-Preference)))
   $\in$  anonymity-homogeneity-class ‘ anonymity-homogeneityQ UNIV
  using image-eqI
  by (metis (no-types))
next
fix x :: rat ^('a Ordered-Preference)
assume x  $\in$  rat-vector-set (convex hull standard-basis)
— The following converts a rational vector  $x$  to real vector  $x'$ .
then obtain x' :: real ^('a Ordered-Preference) where
  conv: x'  $\in$  convex hull standard-basis and
  inv:  $\forall$  p. x$p = the-inv real-of-rat (x'$p) and
  rat:  $\forall$  p. x'$p  $\in$  Q
  unfolding rat-vector-set.simps rat-vector.simps
  by force
hence convex: ( $\forall$  p. 0  $\leq$  x'$p)  $\wedge$  sum (( $\$$ ) x') UNIV = 1
  using standard-simplex-rewrite
  by blast
have map:  $\forall$  p. 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.  $\exists$  fract. Fract (fst fract) (snd fract) = x$p  $\wedge$  0 < snd fract
  using quotient-of-unique
  by metis
then obtain fraction' :: 'a Ordered-Preference  $\Rightarrow$  (int  $\times$  int) where
   $\forall$  p. x$p = Fract (fst (fraction' p)) (snd (fraction' p)) and
  pos':  $\forall$  p. 0 < snd (fraction' p)
  by metis
with map
have fract':  $\forall$  p. x'$p = (fst (fraction' p)) / (snd (fraction' p))
  using div-by-0 divide-less-cancel of-int-0 of-int-pos of-rat-rat
  by metis
with convex
have  $\forall$  p. (fst (fraction' p)) / (snd (fraction' p))  $\geq$  0
  by fastforce

```

```

with pos'
have  $\forall p. \text{fst} (\text{fraction}' p) \geq 0$ 
  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  $\forall p. \exists (n::\text{nat}) (m::\text{nat}). \text{fst} (\text{fraction}' p) = n \wedge \text{snd} (\text{fraction}' p) = m$ 
  using Nats-cases
  by metis
hence  $\forall p. \exists m::\text{nat} \times \text{nat}. \text{fst} (\text{fraction}' p) = \text{int} (\text{fst } m) \wedge \text{snd} (\text{fraction}' p)$ 
 $= \text{int} (\text{snd } m)$ 
  by simp
then obtain fraction :: 'a Ordered-Preference  $\Rightarrow (\text{nat} \times \text{nat})$  where
  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 ( $\lambda p. \text{snd} (\text{fraction } p)$ ) UNIV
have fin: finite (UNIV::('a Ordered-Preference set))
  by simp
hence finite {snd (fraction p) | p. p  $\in$  UNIV}
  using finite-Atleast-Atmost-nat
  by simp
have pos-prod: ?prod > 0
  using pos
  by simp
hence  $\forall p. ?\text{prod} \bmod (\text{snd} (\text{fraction } p)) = 0$ 
  using pos finite UNIV-I bits-mod-0 mod-prod-eq mod-self prod-zero
  by (metis (mono-tags, lifting))
hence div:  $\forall p. (?\text{prod} \text{ div } (\text{snd} (\text{fraction } p))) * (\text{snd} (\text{fraction } p)) = ?\text{prod}$ 
  using add commute add-0 div-mult-mod-eq
  by metis
obtain voter-amount :: 'a Ordered-Preference  $\Rightarrow \text{nat}$  where
  def: voter-amount = ( $\lambda p. (\text{fst} (\text{fraction } p)) * (?\text{prod} \text{ div } (\text{snd} (\text{fraction } p)))$ )
  by blast
have rewrite-div:  $\forall p. ?\text{prod} \text{ div } (\text{snd} (\text{fraction } p)) = ?\text{prod} / (\text{snd} (\text{fraction } p))$ 
  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 ( $\lambda p. (\text{fst} (\text{fraction } p)) * (?\text{prod} / (\text{snd} (\text{fraction } p)))$ ) 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:  $V = \bigcup \{part\ p \mid p. p \in UNIV\}$  and
  disjoint:  $\forall\ p\ p'. p \neq p' \longrightarrow part\ p \cap part\ p' = \{\}$  and
  card:  $\forall\ p. card\ (part\ p) = voter-amount\ p$ 
  using obtain-partition[of V UNIV voter-amount]
  by auto
hence exactly-one-prof:  $\forall\ v \in V. \exists!p. v \in part\ p$ 
  by blast
then obtain prof' :: 'v ⇒ 'a Ordered-Preference where
  maps-to-prof':  $\forall\ v \in V. v \in part\ (prof'\ v)$ 
  by metis
then obtain prof :: 'v ⇒ 'a Preference-Relation where
  prof:  $prof = (\lambda\ v. \text{if } v \in V \text{ then } ord2pref\ (prof'\ v) \text{ else } \{\})$ 
  by blast
hence election:  $(UNIV, V, prof) \in elections-A\ UNIV$ 
  unfolding elections-A.simps valid-elections-def profile-def
  using fin-V ord2pref
  by auto
have  $\forall\ p. \{v \in V. prof'\ v = p\} = \{v \in V. v \in part\ p\}$ 
  using maps-to-prof' exactly-one-prof
  by blast
hence  $\forall\ p. \{v \in V. prof'\ v = p\} = part\ p$ 
  using partition
  by fastforce
hence  $\forall\ p. card\ \{v \in V. prof'\ v = p\} = voter-amount\ p$ 
  using card
  by presburger
moreover have  $\forall\ p. \forall\ v. (v \in \{v \in V. prof'\ v = p\}) = (v \in \{v \in V. prof\ v$ 
= (ord2pref p))
  using prof
  by (simp add: ord2pref-inject)
ultimately have  $\forall\ p. card\ \{v \in V. prof\ v = (ord2pref\ p)\} = voter-amount\ p$ 
  by simp
hence  $\forall\ p::'a\ Ordered-Preference.$ 
  vote-fraction (ord2pref p) (UNIV, V, prof) = Fract (voter-amount p) (card

```

$V)$   
**using** *rat-number-collapse fin-V*  
**by** *simp*  
**moreover have**  $\forall p. \text{Fract} (\text{voter-amount } p) (\text{card } V) = (\text{voter-amount } p) /$   
 $(\text{card } V)$   
**unfolding** *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** *card def card-V-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** *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*:  
 $\forall p :: 'a \text{ Ordered-Preference. vote-fraction } (\text{ord2pref } p) (UNIV, V, \text{prof}) =$   
 $x' \$ p$   
**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 $_{\mathcal{R}}$ .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**  $\forall x \in \mathbb{Q}. \forall y. \text{complex-of-rat } y = \text{complex-of-real } x \longrightarrow y = \text{the-inv}$   
 $\text{real-of-rat } x$

```

    using injI of-rat-eq-iff the-inv-f-f
    by metis
  with vec-entries-match-E-vote-fraction
  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 $_{\mathcal{R}}$ .simps election
    by blast
  ultimately have  $\forall p. \text{vote-fraction } (\text{ord2pref } p) \text{ “ } \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\} \supseteq \{x\$p\}$ 
    using image-insert insert-iff mk-disjoint-insert singletonD subsetI
    by (metis (no-types, lifting))
  with all-eq-vec
  have  $\forall p. \text{vote-fraction } (\text{ord2pref } p) \text{ “ } \text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\} = \{x\$p\}$ 
    by blast
  hence  $\forall p. \text{vote-fraction}_{\mathcal{Q}} (\text{ord2pref } p)$ 
     $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\}) = x\$p$ 
    using is-singletonI singleton-inject singleton-set-def-if-card-one
    unfolding is-singleton-altdef vote-fraction $_{\mathcal{Q}}$ .simps  $\pi_{\mathcal{Q}}$ .simps
    by metis
  hence  $x = \text{anonymity-homogeneity-class}$ 
     $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV}) \text{ “ } \{(UNIV, V, \text{prof})\})$ 
    unfolding anonymity-homogeneity-class.simps
    using vec-lambda-unique
    by (metis (no-types, lifting))
  moreover have  $(\text{anonymity-homogeneity}_{\mathcal{R}} (\text{elections-}\mathcal{A} \text{ UNIV})) \text{ “ } \{(UNIV, V, \text{prof})\}$ 
     $\in \text{anonymity-homogeneity}_{\mathcal{Q}} \text{ UNIV}$ 
    unfolding anonymity-homogeneity $_{\mathcal{Q}}$ .simps quotient-def
    using election
    by blast
  ultimately show
     $x \in (\text{anonymity-homogeneity-class} :: ('a, 'v) \text{ Election set} \Rightarrow \text{rat}^{\sim}('a \text{ Ordered-Preference}))$ 
    ‘anonymity-homogeneity $_{\mathcal{Q}}$  UNIV
    by blast
qed
qed
end

```

## Chapter 4

# Component Types

### 4.1 Distance

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

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

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

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

#### 4.1.1 Definition

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

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

```
fun tup :: 'a Distance  $\Rightarrow$  ('a * 'a  $\Rightarrow$  ereal) where
  tup d = ( $\lambda$  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**  
 $f :: 'a \Rightarrow \text{int}$  **and**  
 $g :: 'a \Rightarrow \text{int}$   
**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 :: 'a  $\Rightarrow$  int and
    g :: 'a  $\Rightarrow$  int
  shows  $(\sum a \in A. f\ a) + (\sum a \in A. g\ a) = (\sum a \in A. f\ a + g\ a)$ 
  using sum.distrib
  by metis

```

```

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

```

```

lemma uneq-ereal:
  fixes
    x :: int and
    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 :: 'a Vote and
        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 :: 'a Vote and
        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  $\sum a \in A. abs (int (rank x a) - int (rank y a))$   
 else  $\infty$ )

**lemma** *spearman-case-inf*:  
**fixes**  
 $x :: 'a \text{ Vote}$  **and**  
 $y :: 'a \text{ Vote}$   
**assumes**  $alts\text{-}\mathcal{V} x \neq alts\text{-}\mathcal{V} y$   
**shows**  $spearman x y = \infty$   
**using** *assms*  
**by** (induction rule: *spearman.induct, simp*)

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

#### 4.1.7 Properties

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

##### Definitions

**fun** *total-invariance<sub>D</sub>* ::  $'x \text{ Distance} \Rightarrow 'x \text{ rel} \Rightarrow \text{bool}$  **where**  
*total-invariance<sub>D</sub>*  $d \text{ rel} = is\text{-}symmetry (tup d) (Invariance (product rel))$

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

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

```

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 $\mathcal{G}$ ) X ( $\varphi$ -neutr 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 $\mathcal{G}$ ) X ( $\varphi$ -rev X)

definition distance-homogeneity' :: ('a, 'v::linorder) Election set
   $\Rightarrow$  ('a, 'v) Election Distance  $\Rightarrow$  bool where
  distance-homogeneity' X d = 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 = 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 (safe)
  fix
    a :: 'x and
    b :: 'x and
    x :: 'x and
    y :: 'x
  assume
    inv: total-invariance $\mathcal{D}$  d r and
    (a, b)  $\in$  r and
    (x, y)  $\in$  r
  hence rel: ((a, x), (b, y))  $\in$  product r
  by simp
  hence tup d (a, x) = tup d (b, y)
  using inv
  unfolding total-invariance $\mathcal{D}$ .simps is-symmetry.simps
  by simp
  thus d a x = d b y
  by simp
next
  show  $\forall$  (x, y)  $\in$  r.  $\forall$  (a, b)  $\in$  r. d a x = d b y  $\implies$  total-invariance $\mathcal{D}$  d r
  proof (unfold total-invariance $\mathcal{D}$ .simps is-symmetry.simps product.simps, safe)
  fix
    a :: 'x and
    b :: 'x and
    x :: 'x and
    y :: 'x

```

```

assume
   $\forall (x, y) \in r. \forall (a, b) \in r. d\ a\ x = d\ b\ y$  and
   $(fst\ (x, a), fst\ (y, b)) \in r$  and
   $(snd\ (x, a), snd\ (y, b)) \in r$ 
hence  $d\ x\ a = d\ y\ b$ 
  by auto
thus  $tup\ d\ (x, a) = tup\ d\ (y, b)$ 
  by simp
qed
qed

lemma rewrite-invarianceD:
fixes
   $d :: 'y\ Distance$  and
   $X :: 'x\ set$  and
   $Y :: 'y\ set$  and
   $\varphi :: ('x, 'y)\ binary\_fun$ 
shows  $invariance_{\mathcal{D}}\ d\ X\ Y\ \varphi = (\forall x \in X. \forall y \in Y. \forall z \in Y. d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z))$ 
proof (safe)
fix
   $x :: 'x$  and
   $y :: 'y$  and
   $z :: 'y$ 
assume
   $x \in X$  and
   $y \in Y$  and
   $z \in Y$  and
   $invariance_{\mathcal{D}}\ d\ X\ Y\ \varphi$ 
thus  $d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z)$ 
  by fastforce
next
show  $\forall x \in X. \forall y \in Y. \forall z \in Y. d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z) \implies invariance_{\mathcal{D}}\ d\ X\ Y\ \varphi$ 
proof (unfold invarianceD.sims is-symmetry.sims equivariance.sims, safe)
fix
   $x :: 'x$  and
   $a :: 'y$  and
   $b :: 'y$ 
assume
   $\forall x \in X. \forall y \in Y. \forall z \in Y. d\ y\ z = d\ (\varphi\ x\ y)\ (\varphi\ x\ z)$  and
   $x \in X$  and
   $a \in Y$  and
   $b \in Y$ 
hence  $d\ a\ b = d\ (\varphi\ x\ a)\ (\varphi\ x\ b)$ 
  by blast
thus  $tup\ d\ (a, b) = tup\ d\ (\varphi\ x\ a, \varphi\ x\ b)$ 
  by simp
qed

```

qed

**lemma** *invar-dist-image*:

**fixes**

$d :: 'y$  Distance **and**

$G :: 'x$  monoid **and**

$Y :: 'y$  set **and**

$Y' :: 'y$  set **and**

$\varphi :: ('x, 'y)$  binary-fun **and**

$y :: 'y$  **and**

$g :: 'x$

**assumes**

*invar-d*: *invariance<sub>D</sub>*  $d$  (carrier  $G$ )  $Y$   $\varphi$  **and**

*Y'-in-Y*:  $Y' \subseteq Y$  **and**

*action- $\varphi$* : group-action  $G$   $Y$   $\varphi$  **and**

*g-carrier*:  $g \in \text{carrier } G$  **and**

*y-in-Y*:  $y \in Y$

**shows**  $d (\varphi g y) \text{ ' } (\varphi g) \text{ ' } Y' = d y \text{ ' } Y'$

**proof** (*safe*)

**fix**  $y' :: 'y$

**assume** *y'-in-Y'*:  $y' \in Y'$

**hence**  $((y, y'), ((\varphi g y), (\varphi g y')))) \in \text{equivariance (carrier } G) Y \varphi$

**using** *Y'-in-Y y-in-Y g-carrier*

**unfolding** *equivariance.simps*

**by** *blast*

**hence** *eq-dist*:  $\text{tup } d ((\varphi g y), (\varphi g y')) = \text{tup } d (y, y')$

**using** *invar-d*

**unfolding** *invariance<sub>D</sub>.simps*

**by** *fastforce*

**thus**  $d (\varphi g y) (\varphi g y') \in d y \text{ ' } Y'$

**using** *y'-in-Y'*

**by** *simp*

**have**  $\varphi g y' \in \varphi g \text{ ' } Y'$

**using** *y'-in-Y'*

**by** *simp*

**thus**  $d y y' \in d (\varphi g y) \text{ ' } \varphi g \text{ ' } Y'$

**using** *eq-dist*

**by** (*simp add: rev-image-eqI*)

qed

**lemma** *swap-neutral*: *invariance<sub>D</sub>* swap (carrier neutrality<sub>G</sub>)

*UNIV*  $(\lambda \pi (A, q). (\pi \text{ ' } A, \text{rel-rename } \pi q))$

**proof** (*unfold rewrite-invariance<sub>D</sub>, safe*)

**fix**

$\pi :: 'a \Rightarrow 'a$  **and**

$A :: 'a$  set **and**

$q :: 'a$  rel **and**

$A' :: 'a$  set **and**

$q' :: 'a$  rel

```

assume  $\pi \in \text{carrier neutrality}_G$ 
hence  $\text{bij}$ :  $\text{bij } \pi$ 
  unfolding  $\text{neutrality}_G\text{-def}$ 
  using  $\text{rewrite-carrier}$ 
  by  $\text{blast}$ 
show  $\text{swap } (A, q) (A', q') = \text{swap } (\pi \text{ ` } A, \text{rel-rename } \pi \text{ } q) (\pi \text{ ` } A', \text{rel-rename } \pi$ 
 $q')$ 
proof ( $\text{cases } A = A'$ )
  let  $?f = (\lambda (a, b). (\pi \text{ } a, \pi \text{ } b))$ 
  let  $?swap\text{-set} = \{(a, b) \in A \times A. a \neq b \wedge \text{neg-ord } q \text{ } q' \text{ } a \text{ } b\}$ 
  let  $?swap\text{-set}' =$ 
 $\{(a, b) \in \pi \text{ ` } A \times \pi \text{ ` } A. a \neq b \wedge \text{neg-ord } (\text{rel-rename } \pi \text{ } q) (\text{rel-rename } \pi \text{ } q')$ 
 $a \text{ } b\}$ 
  let  $?rel = \{(a, b) \in A \times A. a \neq b \wedge \text{neg-ord } q \text{ } q' \text{ } a \text{ } b\}$ 
  case  $\text{True}$ 
  hence  $\pi \text{ ` } A = \pi \text{ ` } A'$ 
  by  $\text{simp}$ 
hence  $\text{swap } (\pi \text{ ` } A, \text{rel-rename } \pi \text{ } q) (\pi \text{ ` } A', \text{rel-rename } \pi \text{ } q') = \text{card } ?swap\text{-set}'$ 
  by  $\text{simp}$ 
moreover have  $\text{bij-betw } ?f \text{ } ?swap\text{-set} \text{ } ?swap\text{-set}'$ 
proof ( $\text{unfold } \text{bij-betw-def } \text{inj-on-def}, \text{intro } \text{conjI } \text{impI } \text{ballI}$ )
  fix
     $x :: 'a \times 'a$  and
     $y :: 'a \times 'a$ 
  assume
     $x \in ?swap\text{-set}$  and
     $y \in ?swap\text{-set}$  and
     $?f \text{ } x = ?f \text{ } y$ 
  hence
     $\pi (fst \text{ } x) = \pi (fst \text{ } y)$  and
     $\pi (snd \text{ } x) = \pi (snd \text{ } y)$ 
  by  $\text{auto}$ 
hence
     $fst \text{ } x = fst \text{ } y$  and
     $snd \text{ } x = snd \text{ } y$ 
  using  $\text{bij } \text{bij-pointE}$ 
  by ( $\text{metis}, \text{metis}$ )
thus  $x = y$ 
  using  $\text{prod.expand}$ 
  by  $\text{metis}$ 
next
show  $?f \text{ ` } ?swap\text{-set} = ?swap\text{-set}'$ 
proof
  have  $\forall a \text{ } b. (a, b) \in A \times A \longrightarrow (\pi \text{ } a, \pi \text{ } b) \in \pi \text{ ` } A \times \pi \text{ ` } A$ 
  by  $\text{simp}$ 
moreover have  $\forall a \text{ } b. a \neq b \longrightarrow \pi \text{ } a \neq \pi \text{ } b$ 
  using  $\text{bij } \text{bij-pointE}$ 
  by  $\text{metis}$ 
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  $?f \text{ ' } ?\text{swap-set} \subseteq ?\text{swap-set}'$ 
      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 bij-is-inj the-inv-f-f
      by fastforce
      moreover have  $\forall a b. (a, b) \in (\text{rel-rename } \pi \ q') \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi$ 
 $\pi \ b) \in q'$ 
      unfolding rel-rename.simps
      using bij 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  $\forall a b. (a, b) \in \pi \text{ ' } A \times \pi \text{ ' } A \longrightarrow (\text{the-inv } \pi \ a, \text{the-inv } \pi$ 
 $b) \in A \times A$ 
      using bij 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 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
      by fastforce
      ultimately show  $?\text{swap-set}' \subseteq ?f \text{ ' } ?\text{swap-set}$ 
      by blast
    qed
  qed
  moreover have  $\text{card } ?\text{swap-set} = \text{swap } (A, q) \ (A', q')$ 
  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 bij-is-inj inj-image-eq-iff
  by metis
  hence  $\text{swap } (A, q) \ (A', q') = \infty \wedge$ 
 $\text{swap } (\pi \text{ ' } A, \text{rel-rename } \pi \ q) \ (\pi \text{ ' } A', \text{rel-rename } \pi \ q') = \infty$ 
  using False

```

```

      by simp
    thus ?thesis
      by simp
  qed
qed
end

```

## 4.2 Votewise Distance

```

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

```

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

### 4.2.1 Definition

```

fun votewise-distance :: 'a Vote Distance  $\Rightarrow$  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-map2-permute:
fixes
   $d$  :: 'a Vote Distance and
   $n$  :: Norm and
   $A$  :: 'a set and
   $A'$  :: 'a set and
   $\varphi$  :: nat  $\Rightarrow$  nat and
   $p$  :: ('a Preference-Relation) list and
   $p'$  :: ('a Preference-Relation) list
assumes
  perm:  $\varphi$  permutes  $\{0 \dots \text{length } p\}$  and
  len-eq:  $\text{length } p = \text{length } p'$  and
  sym-n: symmetry  $n$ 
shows  $n$  (map2 ( $\lambda$   $q$   $q'$ .  $d$  ( $A$ ,  $q$ ) ( $A'$ ,  $q'$ ))  $p$   $p'$ )
      =  $n$  (map2 ( $\lambda$   $q$   $q'$ .  $d$  ( $A$ ,  $q$ ) ( $A'$ ,  $q'$ )) (permute-list  $\varphi$   $p$ ) (permute-list  $\varphi$   $p'$ ))

```



**proof** –  
**let**  $?z = \text{zip } p \ p'$  **and**  
 $?lt-len = \lambda i. \{..< \text{length } i\}$  **and**  
 $?c-prod = \text{case-prod } (\lambda q \ q'. d \ (A, q) \ (A', q'))$   
**let**  $?listpi = \lambda q. \text{permute-list } \varphi \ q$   
**let**  $?q = ?listpi \ p$  **and**  
 $?q' = ?listpi \ p'$   
**have**  $listpi-sym: \forall \ l. (\text{length } l = \text{length } p \longrightarrow ?listpi \ l <\sim\sim> l)$   
**using**  $mset-permute-list \ \text{perm} \ \text{atLeast-upt}$   
**by**  $simp$   
**moreover** **have**  $\text{length } (\text{map2 } (\lambda x \ y. d \ (A, x) \ (A', y)) \ p \ p') = \text{length } p$   
**using**  $len-eq$   
**by**  $simp$   
**ultimately** **have**  $(\text{map2 } (\lambda q \ q'. d \ (A, q) \ (A', q')) \ p \ p')$   
 $<\sim\sim> (?listpi \ (\text{map2 } (\lambda x \ y. d \ (A, x) \ (A', y)) \ p \ p'))$   
**by**  $metis$   
**hence**  $n \ (\text{map2 } (\lambda q \ q'. d \ (A, q) \ (A', q')) \ p \ p')$   
 $= n \ (?listpi \ (\text{map2 } (\lambda x \ y. d \ (A, x) \ (A', y)) \ p \ p'))$   
**using**  $sym-n$   
**unfolding**  $symmetry-def$   
**by**  $blast$   
**also** **have**  $\dots = n \ (\text{map } (\text{case-prod } (\lambda x \ y. d \ (A, x) \ (A', y)))$   
 $(?listpi \ (\text{zip } p \ p')))$   
**using**  $permute-list-map[of \ \varphi \ ?z \ ?c-prod] \ \text{perm} \ len-eq \ \text{atLeast-upt}$   
**by**  $simp$   
**also** **have**  $\dots = n \ (\text{map2 } (\lambda x \ y. d \ (A, x) \ (A', y)) \ (?listpi \ p) \ (?listpi \ p'))$   
**using**  $len-eq \ \text{perm} \ \text{atLeast-upt}$   
**by**  $(simp \ add: \ \text{permute-list-zip})$   
**finally** **show**  $?thesis$   
**by**  $simp$   
**qed**

**lemma** *permute-invariant-under-map:*

**fixes**  
 $l :: 'a \ \text{list}$  **and**  
 $ls :: 'a \ \text{list}$   
**assumes**  $l <\sim\sim> ls$   
**shows**  $\text{map } f \ l <\sim\sim> \text{map } f \ ls$   
**using**  $assms$   
**by**  $simp$

**lemma** *linorder-rank-injective:*

**fixes**  
 $V :: 'v::\text{linorder} \ \text{set}$  **and**  
 $v :: 'v$  **and**  
 $v' :: 'v$   
**assumes**  
 $v\text{-in-}V: v \in V$  **and**  
 $v'\text{-in-}V: v' \in V$  **and**

```

    v'-neq-v: v' ≠ v and
    fin-V: finite V
  shows card {x ∈ V. x < v} ≠ card {x ∈ V. x < v'}
proof -
  have v < v' ∨ v' < v
    using v'-neq-v linorder-less-linear
  by metis
  hence {x ∈ V. x < v} ⊂ {x ∈ V. x < v'} ∨ {x ∈ V. x < v'} ⊂ {x ∈ V. x < v}
    using v-in-V v'-in-V dual-order.strict-trans
  by blast
  thus ?thesis
    using assms sorted-list-of-set-nth-equals-card
    by (metis (full-types))
qed

```

**lemma** *permute-invariant-under-coinciding-funs:*

```

fixes
  l :: 'v list and
  π-1 :: nat ⇒ nat and
  π-2 :: nat ⇒ nat
assumes ∀ i < length l. π-1 i = π-2 i
shows permute-list π-1 l = permute-list π-2 l
using assms
unfolding permute-list-def
by simp

```

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

```

fixes
  d :: 'a Vote Distance and
  n :: Norm
assumes symmetry n
shows distance-anonymity (votewise-distance d n)
proof (unfold distance-anonymity-def, safe)
  fix
    A :: 'a set and
    A' :: 'a set and
    V :: 'v::linorder set and
    V' :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile and
    π :: 'v ⇒ 'v
  let ?rn1 = rename π (A, V, p) and
      ?rn2 = rename π (A', V', p') and
      ?rn-V = π ' V and
      ?rn-V' = π ' V' and
      ?rn-p = p ∘ (the-inv π) and
      ?rn-p' = p' ∘ (the-inv π) 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*.  
 $?perm-total = (\lambda i. (if (i < ?len)$   
      $then card (\{v \in ?rn-V. v < \pi (?sl-V!i)\})$   
      $else i))$   
**assume** *bij*: *bij*  $\pi$   
**show** *votewise-distance*  $d\ n\ (A, V, p)\ (A', V', p') = votewise-distance\ d\ n\ ?rn1$   
 $?rn2$   
**proof** —  
   **have** *rn-A-eq-A*: *fst*  $?rn1 = A$   
     **by** *simp*  
   **have** *rn-A'-eq-A'*: *fst*  $?rn2 = A'$   
     **by** *simp*  
   **have** *rn-V-eq-pi-V*: *fst* (*snd*  $?rn1$ ) =  $?rn-V$   
     **by** *simp*  
   **have** *rn-V'-eq-pi-V'*: *fst* (*snd*  $?rn2$ ) =  $?rn-V'$   
     **by** *simp*  
   **have** *rn-p-eq-pi-p*: *snd* (*snd*  $?rn1$ ) =  $?rn-p$   
     **by** *simp*  
   **have** *rn-p'-eq-pi-p'*: *snd* (*snd*  $?rn2$ ) =  $?rn-p'$   
     **by** *simp*  
   **show** *?thesis*  
   **proof** (*cases finite*  $V \wedge V = V' \wedge (V \neq \{\} \vee A = A')$ )  
     **case** *False*  
     — Case: Both distances are infinite.  
     **hence** *inf-dist*: *votewise-distance*  $d\ n\ (A, V, p)\ (A', V', p') = \infty$   
       **by** *auto*  
     **moreover** **have** *infinite*  $V \implies infinite\ ?rn-V$   
       **using** *False bij bij-betw-finite bij-betw-subset False subset-UNIV*  
       **by** *metis*  
     **moreover** **have**  $V \neq V' \implies ?rn-V \neq ?rn-V'$   
       **using** *bij bij-def inj-image-mem-iff subsetI subset-antisym*  
       **by** *metis*  
     **moreover** **have**  $V = \{\} \implies ?rn-V = \{\}$   
       **using** *bij*  
       **by** *simp*  
     **ultimately** **have** *inf-dist-rewrite*: *votewise-distance*  $d\ n\ ?rn1\ ?rn2 = \infty$   
       **using** *False*  
       **by** *auto*  
     **thus** *votewise-distance*  $d\ n\ (A, V, p)\ (A', V', p') = votewise-distance\ d\ n$   
 $?rn1\ ?rn2$   
       **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 to-list-permutes-under-bij
  unfolding comp-def
  by (metis (no-types))
hence len-V-rn-V-eq: ?len = length (to-list ?rn-V ?rn-p)
  by simp
hence permute-list ?perm (to-list ?rn-V ?rn-p)
  = permute-list ?perm-total (to-list ?rn-V ?rn-p)
  using permute-invariant-under-coinciding-funs[of (to-list ?rn-V ?rn-p)]
    perm-funs-coincide
  by presburger
hence rn-list-perm-list-V: (to-list V p) = permute-list ?perm-total (to-list
?rn-V ?rn-p)
  using rn-V-permutes
  by metis

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

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

  have  $\forall i j. (i < ?len \wedge j < ?len \wedge i \neq j$ 
     $\longrightarrow \pi ((\text{sorted-list-of-set } V)!i) \neq \pi ((\text{sorted-list-of-set } V)!j))$ 
    using bij bij-pointE True nth-eq-iff-index-eq length-map
      sorted-list-of-set.distinct-sorted-key-list-of-set to-list.elims
    by (metis (mono-tags, opaque-lifting))
  moreover have in-bnds-imp-img-el:  $\forall i. i < ?len \longrightarrow \pi ((\text{sorted-list-of-set } V)!i) \in \pi ' V$ 
    using True image-eqI nth-mem sorted-list-of-set(1) to-list.simps length-map

```

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

```

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

```

fixes
  d :: 'a Vote Distance and
  n :: Norm
defines vote-action ≡ (λ π (A, q). (π 'A, rel-rename π q))
assumes invar: invarianceD d (carrier neutralityG) UNIV vote-action
shows distance-neutrality valid-elections (votewise-distance d n)
proof (unfold distance-neutrality.simps rewrite-invarianceD, safe)
fix
  A :: 'a set and
  A' :: 'a set and
  V :: 'v::linorder set and
  V' :: 'v set and
  p :: ('a, 'v) Profile and
  p' :: ('a, 'v) Profile and
  π :: 'a ⇒ 'a
assume
  carrier: π ∈ carrier neutralityG and
  valid: (A, V, p) ∈ valid-elections and
  valid': (A', V', p') ∈ valid-elections
hence bij: bij π

```

```

unfolding neutralityG-def
using rewrite-carrier
by blast
thus votewise-distance  $d\ n\ (A, V, p)\ (A', V', p') =$ 
  votewise-distance  $d\ n$ 
    ( $\varphi$ -neutr valid-elections  $\pi\ (A, V, p)$ ) ( $\varphi$ -neutr valid-elections  $\pi\ (A', V',$ 
 $p')$ )
proof (cases finite  $V \wedge V = V' \wedge (V \neq \{\} \vee A = A')$ )
  case True
    hence finite  $V \wedge V = V' \wedge (V \neq \{\} \vee \pi\ 'A = \pi\ 'A')$ 
    by metis
    hence votewise-distance  $d\ n$ 
      ( $\varphi$ -neutr valid-elections  $\pi\ (A, V, p)$ ) ( $\varphi$ -neutr valid-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 map2-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 rewrite-invarianceD[of  $d$  carrier neutralityG UNIV vote-action]
        invar carrier UNIV-I case-prod-conv
      unfolding vote-action-def
      by (metis (no-types, lifting))
    finally have votewise-distance  $d\ n$ 
      ( $\varphi$ -neutr valid-elections  $\pi\ (A, V, p)$ ) ( $\varphi$ -neutr valid-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 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  $\neg (\text{finite}\ V \wedge V = V' \wedge (V \neq \{\} \vee \pi\ 'A = \pi\ 'A'))$ 

```

```

    using bij bij-is-inj inj-image-eq-iff
    by metis
  hence votewise-distance d n
    ( $\varphi$ -neutr valid-elections  $\pi$  ( $A$ ,  $V$ ,  $p$ )) ( $\varphi$ -neutr valid-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  $p$   $v =$  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-topC' :: 'a  $\Rightarrow$  ('a, 'v) Consensus where
  equal-topC' a ( $A$ ,  $V$ ,  $p$ ) = ( $a \in A \wedge (\forall v \in V. \text{above } (p\ v) a = \{a\})$ )

```



**fun** *equal-top<sub>C</sub>* :: ('a, 'v) *Consensus* **where**  
*equal-top<sub>C</sub>* c = ( $\exists$  a. *equal-top<sub>C</sub>'* a c)

Equal votes.

**fun** *equal-vote<sub>C</sub>'* :: 'a *Preference-Relation*  $\Rightarrow$  ('a, 'v) *Consensus* **where**  
*equal-vote<sub>C</sub>'* r (A, V, p) = ( $\forall$  v  $\in$  V. (p v) = r)

**fun** *equal-vote<sub>C</sub>* :: ('a, 'v) *Consensus* **where**  
*equal-vote<sub>C</sub>* c = ( $\exists$  r. *equal-vote<sub>C</sub>'* r c)

Unanimity condition.

**fun** *unanimity<sub>C</sub>* :: ('a, 'v) *Consensus* **where**  
*unanimity<sub>C</sub>* c = (*nonempty-set<sub>C</sub>* c  $\wedge$  *nonempty-profile<sub>C</sub>* c  $\wedge$  *equal-top<sub>C</sub>* c)

Strong unanimity condition.

**fun** *strong-unanimity<sub>C</sub>* :: ('a, 'v) *Consensus* **where**  
*strong-unanimity<sub>C</sub>* c = (*nonempty-set<sub>C</sub>* c  $\wedge$  *nonempty-profile<sub>C</sub>* c  $\wedge$  *equal-vote<sub>C</sub>* c)

### 4.3.3 Properties

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

**fun** *consensus-neutrality* :: ('a, 'v) *Election set*  $\Rightarrow$  ('a, 'v) *Consensus*  $\Rightarrow$  *bool* **where**  
*consensus-neutrality* X c = *is-symmetry* c (*Invariance* (*neutrality<sub>R</sub>* X))

### 4.3.4 Auxiliary Lemmas

**lemma** *cons-anon-conj*:

**fixes**

*c1* :: ('a, 'v) *Consensus* **and**

*c2* :: ('a, 'v) *Consensus*

**assumes**

*anon1*: *consensus-anonymity* *c1* **and**

*anon2*: *consensus-anonymity* *c2*

**shows** *consensus-anonymity* ( $\lambda e. c1 e \wedge c2 e$ )

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

**fix**

*A* :: 'a *set* **and**

*A'* :: 'a *set* **and**

*V* :: 'v *set* **and**

*V'* :: 'v *set* **and**

*p* :: ('a, 'v) *Profile* **and**

```

  q :: ('a, 'v) Profile and
  π :: 'v ⇒ 'v
assume
  bij: bij π and
  prof: profile V A p and
  renamed: rename π (A, V, p) = (A', V', q) and
  c1: c1 (A, V, p) and
  c2: c2 (A, V, p)
hence profile V' A' q
  using rename-sound renamed bij fst-conv rename.simps
  by metis
thus c1 (A', V', q) ∧ c2 (A', V', q)
  using bij renamed c1 c2 assms prof
  unfolding consensus-anonymity-def
  by auto
qed

theorem cons-conjunction-invariant:
  fixes
    C :: ('a, 'v) Consensus set and
    rel :: ('a, 'v) Election rel
  defines C ≡ (λ E. (∀ C' ∈ C. C' E))
  assumes ∧ C'. C' ∈ C ⇒ is-symmetry C' (Invariance rel)
  shows is-symmetry C (Invariance rel)
proof (unfold is-symmetry.simps, intro allI impI)
  fix
    E :: ('a, 'v) Election and
    E' :: ('a, 'v) Election
  assume (E, E') ∈ rel
  hence ∀ C' ∈ 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 set and
    A' :: 'a set and
    V :: 'v set and
    V' :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and
    π :: 'v ⇒ 'v
  assumes

```

```

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

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

```

by *fastforce*  
 ultimately have  $b' x (A', V', q)$   
 using *all-cond-anon bij prof-p renamed*  
 unfolding *consensus-anonymity-def*  
 by *auto*  
 hence  $\exists x. b' x (A', V', q)$   
 by *metis*  
 thus  $b (A', V', q)$   
 using *general-cond-b*  
 by *simp*  
 qed

### 4.3.5 Theorems

#### Anonymity

**lemma** *nonempty-set-cons-anonymous: consensus-anonymity nonempty-set<sub>C</sub>*  
 unfolding *consensus-anonymity-def*  
 by *simp*

**lemma** *nonempty-profile-cons-anonymous: consensus-anonymity nonempty-profile<sub>C</sub>*  
**proof** (*unfold consensus-anonymity-def Let-def, clarify*)

fix  
 $A :: 'a \text{ set}$  and  
 $A' :: 'a \text{ set}$  and  
 $V :: 'v \text{ set}$  and  
 $V' :: 'v \text{ set}$  and  
 $p :: ('a, 'v) \text{ Profile}$  and  
 $q :: ('a, 'v) \text{ Profile}$  and  
 $\pi :: 'v \Rightarrow 'v$   
 assume  
 $\text{bij: } \text{bij } \pi$  and  
 $\text{prof-p: } \text{profile } V \ A \ p$  and  
 $\text{renamed: } \text{rename } \pi \ (A, V, p) = (A', V', q)$  and  
 $\text{not-empty-p: } \text{nonempty-profile}_C \ (A, V, p)$   
 have  $\text{card } V = \text{card } V'$   
 using *renamed bij rename.simps Pair-inject*  
 $\text{bij-betw-same-card bij-betw-subset top-greatest}$   
 by (*metis (mono-tags, lifting)*)  
 thus  $\text{nonempty-profile}_C \ (A', V', q)$   
 using *not-empty-p length-0-conv renamed*  
 unfolding *nonempty-profile<sub>C</sub>.simps*  
 by *auto*  
 qed

**lemma** *equal-top-cons'-anonymous:*  
 fixes  $a :: 'a$   
 shows *consensus-anonymity (equal-top<sub>C</sub>' a)*  
**proof** (*unfold consensus-anonymity-def Let-def, clarify*)  
 fix

```

  A :: 'a set and
  A' :: 'a set and
  V :: 'v set and
  V' :: 'v set and
  p :: ('a, 'v) Profile and
  q :: ('a, 'v) Profile and
  π :: 'v ⇒ 'v
assume
  bij: bij π and
  prof-p: profile V A p and
  renamed: rename π (A, V, p) = (A', V', q) and
  top-cons-a: equal-topC' a (A, V, p)
have ∀ v' ∈ V'. q v' = p ((the-inv π) v')
  using renamed
  by auto
moreover have ∀ v' ∈ V'. (the-inv π) v' ∈ V
  using bij renamed rename.simps bij-is-inj
  f-the-inv-into-f-bij-betw inj-image-mem-iff
  by fastforce
moreover have winner: ∀ v ∈ V. above (p v) a = {a}
  using top-cons-a
  by simp
ultimately have ∀ v' ∈ V'. above (q v') a = {a}
  by simp
moreover have a ∈ A
  using top-cons-a
  by simp
ultimately show equal-topC' a (A', V', q)
  using renamed
  unfolding equal-topC'.simps
  by simp
qed

lemma eq-top-cons-anon: consensus-anonymity equal-topC
  using equal-top-cons'-anonymous
  ex-anon-cons-imp-cons-anonymous[of equal-topC equal-topC]
  by fastforce

lemma eq-vote-cons'-anonymous:
  fixes r :: 'a Preference-Relation
  shows consensus-anonymity (equal-voteC' r)
proof (unfold consensus-anonymity-def Let-def, clarify)
  fix
    A :: 'a set and
    A' :: 'a set and
    V :: 'v set and
    V' :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and

```

```

   $\pi :: 'v \Rightarrow 'v$ 
assume
  bij: bij  $\pi$  and
  prof-p: profile  $V A p$  and
  renamed: rename  $\pi (A, V, p) = (A', V', q)$  and
  eq-vote: equal-voteC'  $r (A, V, p)$ 
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 renamed rename.simps bij-is-inj
  f-the-inv-into-f-bij-betw inj-image-mem-iff
by fastforce
moreover have winner:  $\forall v \in V. p v = r$ 
using eq-vote
by simp
ultimately have  $\forall v' \in V'. q v' = r$ 
by simp
thus equal-voteC'  $r (A', V', q)$ 
unfolding equal-voteC'.simps
by metis
qed

lemma eq-vote-cons-anonymous: consensus-anonymity equal-voteC
unfolding equal-voteC.simps
using eq-vote-cons'-anonymous ex-anon-cons-imp-cons-anonymous
by blast

```

## Neutrality

```

lemma nonempty-setC-neutral: consensus-neutrality valid-elections nonempty-setC
unfolding valid-elections-def
by auto

```

```

lemma nonempty-profileC-neutral: consensus-neutrality valid-elections nonempty-profileC
unfolding valid-elections-def
by auto

```

```

lemma equal-voteC-neutral: consensus-neutrality valid-elections equal-voteC
proof (unfold valid-elections-def consensus-neutrality.simps is-symmetry.simps,
  intro allI impI,
  unfold split-paired-all neutralityR.simps action-induced-rel.simps
  voters- $\mathcal{E}$ .simps alternatives- $\mathcal{E}$ .simps profile- $\mathcal{E}$ .simps  $\varphi$ -neutr.simps
  extensional-continuation.simps equal-voteC.simps equal-voteC'.simps
  alternatives-rename.simps case-prod-unfold mem-Collect-eq fst-conv
  snd-conv mem-Sigma-iff conj-assoc If-def simp-thms, safe)
fix
   $A :: 'a \text{ set}$  and

```

$A' :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $V' :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$  **and**  
 $\pi :: 'a \Rightarrow 'a$  **and**  
 $r :: 'a \text{ rel}$   
**assume**  
*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** *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 valid-elections strong-unanimityC
  using nonempty-setC-neutral equal-voteC-neutral nonempty-profileC-neutral
    cons-conjunction-invariant[of
      {nonempty-setC, nonempty-profileC, equal-voteC} neutralityR valid-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**  $\text{fun}_{\mathcal{E}} :: ('v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow 'r) \Rightarrow (('a, 'v) \text{ Election} \Rightarrow 'r)$   
**where**  
 $\text{fun}_{\mathcal{E}} m = (\lambda E. m (\text{voters-}\mathcal{E} E) (\text{alternatives-}\mathcal{E} E) (\text{profile-}\mathcal{E} E))$

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

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

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

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

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

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

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

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* = *is-symmetry* (*fun<sub>E</sub>* *m*) (*Invariance* (*anonymity<sub>R</sub>* *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<sub>E</sub>* *m*) (*Invariance* (*homogeneity<sub>R</sub>* *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, 'b Result) Electoral-Module

$\Rightarrow$  bool **where**

*homogeneity'* *X m* = *is-symmetry* (*fun<sub>E</sub>* *m*) (*Invariance* (*homogeneity<sub>R</sub>* *X*))

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

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

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

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

**assume** *rel*: (*E*, *E'*)  $\in$  *anonymity<sub>R</sub>* *X*

**hence** *E*  $\in$  *X*  $\wedge$  *E'*  $\in$  *X*

```

    unfolding anonymity $\mathcal{R}$ .simps action-induced-rel.simps
  by blast
moreover with this
  have fin: finite (voters- $\mathcal{E}$   $E$ )  $\wedge$  finite (voters- $\mathcal{E}$   $E'$ )
  using assms
  by simp
moreover with this
  have  $\forall r. \text{vote-count } r \ E = 1 * (\text{vote-count } r \ E')$ 
  using anon-rel-vote-count rel mult-1
  by metis
moreover with fin
  have alternatives- $\mathcal{E}$   $E = \text{alternatives-}\mathcal{E} \ E'$ 
  using anon-rel-vote-count rel
  by blast
ultimately show  $\text{fun}_{\mathcal{E}} \ m \ E = \text{fun}_{\mathcal{E}} \ m \ E'$ 
  using assms zero-less-one
  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  $X \ m = \text{is-symmetry} (\text{fun}_{\mathcal{E}} \ m)$ 
  (action-induced-equivariance (carrier neutrality $\mathcal{G}$ )  $X$  ( $\varphi$ -neutr  $X$ ) (result-action
 $\psi$ -neutr))

```

### 4.4.4 Reversal Symmetry of Social Welfare Rules

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

```

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

```

### 4.4.5 Social Choice Modules

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

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

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

"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**  $\text{indep-of-alt} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a$   
 $\Rightarrow \text{bool}$  **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**  
 $e :: 'a \text{ set}$  **and**  
 $r :: 'a \text{ set}$  **and**  
 $d :: 'a \text{ set}$

**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**

$\text{SCF-result.electoral-module } m$  **and**  
 $\text{profile } V \ A \ p$

**shows**  $\text{well-formed-SCF } A \ (m \ V \ A \ p)$

**using** *assms*

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

**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**

$\text{SCF-result.electoral-module } m$  **and**  
 $\text{profile } V \ A \ p$

**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**  $a :: 'a$

**assume**  $a \in \text{elect } m \ V \ A \ p$

**moreover have**

$\forall p'. \text{set-equals-partition } A \ p' \longrightarrow$   
 $(\exists E \ R \ D. p' = (E, R, D) \wedge E \cup R \cup D = A)$

**by** *simp*

**moreover have**  $\text{set-equals-partition } A \ (m \ V \ A \ p)$

```

    using assms
    unfolding SCF-result.electoral-module.simps
    by simp
    ultimately show  $a \in A$ 
    using UnI1 fstI
    by (metis (no-types))
next
  fix  $a :: 'a$ 
  assume  $a \in \text{reject } m \ V \ A \ p$ 
  moreover have
     $\forall p'. \text{set-equals-partition } A \ p' \longrightarrow$ 
     $(\exists E \ R \ D. p' = (E, R, D) \wedge E \cup R \cup D = A)$ 
    by simp
  moreover have set-equals-partition  $A \ (m \ V \ A \ p)$ 
    using assms
    unfolding SCF-result.electoral-module.simps
    by simp
  ultimately show  $a \in A$ 
    using UnI1 fstI sndI subsetD sup-ge2
    by metis
next
  fix  $a :: 'a$ 
  assume  $a \in \text{defer } m \ V \ A \ p$ 
  moreover have
     $\forall p'. \text{set-equals-partition } A \ p' \longrightarrow$ 
     $(\exists E \ R \ D. p' = (E, R, D) \wedge E \cup R \cup D = A)$ 
    by simp
  moreover have set-equals-partition  $A \ (m \ V \ A \ p)$ 
    using assms
    unfolding SCF-result.electoral-module.simps
    by simp
  ultimately show  $a \in A$ 
    using sndI subsetD sup-ge2
    by metis
next
  fix  $a :: 'a$ 
  assume
     $a \in A$  and
     $a \notin \text{defer } m \ V \ A \ p$  and
     $a \notin \text{reject } m \ V \ A \ p$ 
  moreover have
     $\forall p'. \text{set-equals-partition } A \ p' \longrightarrow$ 
     $(\exists E \ R \ D. p' = (E, R, D) \wedge E \cup R \cup D = A)$ 
    by simp
  moreover have set-equals-partition  $A \ (m \ V \ A \ p)$ 
    using assms
    unfolding SCF-result.electoral-module.simps
    by simp
  ultimately 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 Result) Electoral-Module and
    A :: 'a set and
    p :: ('a, 'v) Profile and
    V :: 'v set
  assumes
    SCF-result.electoral-module m and
    profile V A p
  shows
    (elect m V A p)  $\cap$  (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) = {}
proof (safe)
  fix a :: 'a
  assume
    a  $\in$  elect m V A p and
    a  $\in$  reject m V A p
  moreover have well-formed-SCF A (m V A p)
  using assms
  unfolding SCF-result.electoral-module.simps
  by metis
  ultimately show a  $\in$  {}
  using prod.exhaust-sel DiffE UnCI result-imp-rej
  by (metis (no-types))
next
  fix a :: 'a
  assume
    elect-a: a  $\in$  elect m V A p and
    defer-a: a  $\in$  defer m V A p
  have disj:
     $\forall p'. \text{disjoint3 } p' \longrightarrow$ 
     $(\exists B\ C\ D. p' = (B, C, D) \wedge B \cap C = \{\} \wedge B \cap D = \{\} \wedge C \cap D = \{\})$ 
  by simp
  have well-formed-SCF A (m V A p)
  using assms
  unfolding SCF-result.electoral-module.simps
  by metis
  hence disjoint3 (m V A p)
  by simp
  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)) ∧
      e (m V A p) ∩ r (m V A p) = {} ∧
      e (m V A p) ∩ d (m V A p) = {} ∧
      r (m V A p) ∩ d (m V A p) = {}
    using elect-a defer-a disj
    by metis
  hence ((elect m V A p) ∩ (reject m V A p) = {}) ∧
        ((elect m V A p) ∩ (defer m V A p) = {}) ∧
        ((reject m V A p) ∩ (defer m V A p) = {})
    using eq-snd-iff fstI
    by metis
  thus a ∈ {}
    using elect-a defer-a disjoint-iff-not-equal
    by (metis (no-types))
next
fix a :: 'a
assume
  a ∈ reject m V A p and
  a ∈ defer m V A p
moreover have well-formed-SCF A (m V A p)
  using assms
  unfolding SCF-result.electoral-module.simps
  by simp
ultimately show a ∈ {}
  using prod.exhaust-sel DiffE UnCI result-imp-rej
  by (metis (no-types))
qed

```

```

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

```

```

lemma reject-in-alts:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    SCF-result.electoral-module m and

```

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

**lemma** *defer-in-alts:*

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

**lemma** *def-presv-prof:*

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

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

**lemma** *upper-card-bounds-for-result:*

**fixes**  
*m :: ('a, 'v, 'a Result) Electoral-Module* **and**  
*A :: 'a set* **and**  
*V :: 'v set* **and**  
*p :: ('a, 'v) Profile*  
**assumes**  
*SCF-result.electoral-module m* **and**  
*profile V A p* **and**  
*finite A*  
**shows**  
*upper-card-bound-for-elect: card (elect m V A p)  $\leq$  card A* **and**  
*upper-card-bound-for-reject: card (reject m V A p)  $\leq$  card A* **and**  
*upper-card-bound-for-defer: card (defer m V A p)  $\leq$  card A*

**proof** –

**show** *card (elect m V A p)  $\leq$  card A*  
**using** *assms card-mono elect-in-alts*

```

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

lemma reject-not-elec-or-def:
  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  $\text{reject } m \ V \ A \ p = A - (\text{elect } m \ V \ A \ p) - (\text{defer } m \ V \ A \ p)$ 
proof -
  have well-formed-SCF A (m V A p)
  using assms
  unfolding SCF-result.electoral-module.simps
  by simp
  hence  $(\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p) = A$ 
  using assms result-presv-alts
  by simp
  moreover have
     $(\text{elect } m \ V \ A \ p) \cap (\text{reject } m \ V \ A \ p) = \{\}$   $\wedge$   $(\text{reject } m \ V \ A \ p) \cap (\text{defer } m \ V \ A \ p) = \{\}$ 
  using assms result-disj
  by blast
  ultimately show ?thesis
  by blast
qed

lemma elec-and-def-not-rej:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  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  $\text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p = A - (\text{reject } m \ V \ A \ p)$ 
proof -

```

**have**  $(elect\ m\ V\ A\ p) \cup (reject\ m\ V\ A\ p) \cup (defer\ m\ V\ A\ p) = A$   
**using** *assms result-presv-alts*  
**by** *blast*  
**moreover have**  
 $(elect\ m\ V\ A\ p) \cap (reject\ m\ V\ A\ p) = \{\}$   $\wedge$   $(reject\ m\ V\ A\ p) \cap (defer\ m\ V\ A\ p) = \{\}$   
**using** *assms result-disj*  
**by** *blast*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

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

**fixes**  
 $m :: ('a, 'v, 'a\ Result)\ Electoral\ Module$  **and**  
 $A :: 'a\ set$  **and**  
 $p :: ('a, 'v)\ Profile$   
**assumes**  
 $SCF\ result.electoral\ module\ m$  **and**  
 $profile\ V\ A\ p$   
**shows**  $defer\ m\ V\ A\ p = A - (elect\ m\ V\ A\ p) - (reject\ m\ V\ A\ p)$   
**proof** –  
**have**  $well\ formed\ SCF\ A\ (m\ V\ A\ p)$   
**using** *assms*  
**unfolding**  $SCF\ result.electoral\ module.simps$   
**by** *simp*  
**hence**  $(elect\ m\ V\ A\ p) \cup (reject\ m\ V\ A\ p) \cup (defer\ m\ V\ A\ p) = A$   
**using** *assms result-presv-alts*  
**by** *simp*  
**moreover have**  
 $(elect\ m\ V\ A\ p) \cap (defer\ m\ V\ A\ p) = \{\}$   $\wedge$   $(reject\ m\ V\ A\ p) \cap (defer\ m\ V\ A\ p) = \{\}$   
**using** *assms result-disj*  
**by** *blast*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *electoral-mod-defer-elem:*

**fixes**  
 $m :: ('a, 'v, 'a\ Result)\ Electoral\ Module$  **and**  
 $A :: 'a\ set$  **and**  
 $V :: 'v\ set$  **and**  
 $p :: ('a, 'v)\ Profile$  **and**  
 $a :: 'a$   
**assumes**  
 $SCF\ result.electoral\ module\ m$  **and**  
 $profile\ V\ A\ p$  **and**  
 $a \in A$  **and**

```

    a ∉ elect m V A p and
    a ∉ reject m V A p
  shows a ∈ defer m V A p
  using DiffI assms reject-not-elec-or-def
  by metis

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

```

```

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

```

```

lemma not-rej-imp-elec-or-defer:
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-result.electoral-module m and
  profile V A p and
   $a \in A$  and
   $a \notin \text{reject } m \ V \ A \ p$ 
shows  $a \in \text{elect } m \ V \ A \ p \vee a \in \text{defer } m \ V \ A \ p$ 
using assms electoral-mod-defer-elem
by metis

```

```

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

```

```

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

```

*prof-q: profile V A q*  
**shows**  $m \ V \ A \ p = m \ V \ A \ q$   
**proof** –  
**have** *elected-in-A: elect m V A q  $\subseteq$  A*  
**using** *elect-in-alts mod-m prof-q*  
**by** *metis*  
**have** *rejected-in-A: reject m V A q  $\subseteq$  A*  
**using** *reject-in-alts mod-m prof-q*  
**by** *metis*  
**have** *deferred-in-A: defer m V A q  $\subseteq$  A*  
**using** *defer-in-alts mod-m prof-q*  
**by** *metis*  
**have**  $\forall a \in \text{elect } m \ V \ A \ p. a \in \text{elect } m \ V \ A \ q$   
**using** *elect-in-alts eq prof-contains-result-def mod-m prof-p in-mono*  
**by** *metis*  
**moreover have**  $\forall a \in \text{elect } m \ V \ A \ q. a \in \text{elect } m \ V \ A \ p$   
**proof**  
**fix**  $a :: 'a$   
**assume** *q-elect-a:  $a \in \text{elect } m \ V \ A \ q$*   
**hence**  $a \in A$   
**using** *elected-in-A*  
**by** *blast*  
**moreover have**  $a \notin \text{defer } m \ V \ A \ q$   
**using** *q-elect-a prof-q mod-m result-disj*  
**by** *blast*  
**moreover have**  $a \notin \text{reject } m \ V \ A \ q$   
**using** *q-elect-a disjoint-iff-not-equal prof-q mod-m result-disj*  
**by** *metis*  
**ultimately show**  $a \in \text{elect } m \ V \ A \ p$   
**using** *electoral-mod-defer-elem eq prof-contains-result-def*  
**by** *metis*  
**qed**  
**moreover have**  $\forall a \in \text{reject } m \ V \ A \ p. a \in \text{reject } m \ V \ A \ q$   
**using** *reject-in-alts eq prof-contains-result-def mod-m prof-p subset-iff*  
**by** *(metis (no-types, lifting))*  
**moreover have**  $\forall a \in \text{reject } m \ V \ A \ q. a \in \text{reject } m \ V \ A \ p$   
**proof**  
**fix**  $a :: 'a$   
**assume** *q-rejects-a:  $a \in \text{reject } m \ V \ A \ q$*   
**hence**  $a \in A$   
**using** *rejected-in-A*  
**by** *blast*  
**moreover have**  $a \notin \text{defer } m \ V \ A \ q$   
**using** *q-rejects-a prof-q mod-m result-disj*  
**by** *blast*  
**moreover have**  $a \notin \text{elect } m \ V \ A \ q$   
**using** *q-rejects-a disjoint-iff-not-equal prof-q mod-m result-disj*  
**by** *metis*  
**ultimately show**  $a \in \text{reject } m \ V \ A \ p$

```

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

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

```



```

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

```

#### 4.4.8 Non-Blocking

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

**definition** *non-blocking* :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$  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 (safe)
  fix a :: 'a
  assume elect-a: a ∈ elect m V A p
  have SCF-result.electoral-module m  $\longrightarrow$  elect m V A p  $\subseteq$  A
    using prof elect-in-alts
    by blast
  hence elect m V A p  $\subseteq$  A
    using electing
    unfolding electing-def
    by metis

```

```

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

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

```

#### 4.4.10 Properties

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

**definition** *non-electing*  $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow \text{bool}$  **where**  
*non-electing m*  $\equiv$

$SCF\text{-result.electoral-module } m \wedge (\forall A V p. \text{profile } V A p \longrightarrow \text{elect } m V A p = \{\})$

**lemma** *single-rej-decr-def-card*:

**fixes**

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

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

$SCF\text{-result.electoral-module } m \wedge (\forall V A q. \text{profile } V A q \longrightarrow \text{elect } m V A q = \{\})$

**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-2*:

**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*:

```

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*  $m \equiv$   
 $\text{SCF}$ -result.electoral-module  $m \wedge \text{non-electing } m \wedge \text{defers } 1 m$

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*  $m \equiv$   
 $\text{SCF}$ -result.electoral-module  $m \wedge$   
 $(\forall A V p. \text{profile } V A p \wedge \text{card } A > 1 \longrightarrow \text{card } (\text{reject } m V A p) \geq 1)$

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

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

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-result.electoral-module* m  $\wedge$   
 $(\forall A V p q a.$   
 $(a \in \text{defer } m \ V \ A \ p \wedge \text{lifted } V \ A \ p \ q \ a) \longrightarrow a \in \text{defer } m \ V \ A \ q)$

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 set **and**

*A'* :: 'a set **and**

*V* :: 'v set **and**

*V'* :: 'v set **and**

*p* :: ('a, 'v) Profile **and**

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

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* ::  $(\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module} \Rightarrow \text{bool}$   
**where**

$\text{invariant-monotonicity } m \equiv$   
 $\text{SCF-result.electoral-module } m \wedge$   
 $(\forall A V p q a. (a \in \text{elect } m V A p \wedge \text{lifted } V A p q a) \longrightarrow$   
 $(\text{elect } m V A q = \text{elect } m V A p \vee \text{elect } m V A q = \{a\}))$

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  $\equiv$   
*SCF-result.electoral-module* m  $\wedge$  *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 Result) Electoral-Module **and**

*A* :: 'a set **and**

*V* :: 'v set **and**

*p* :: ('a, 'v) Profile **and**

*a* :: 'a

**assumes**

*ccomp*: *condorcet-compatibility* m **and**

*dd*: *defer-deciding* m **and**

*winner*: *condorcet-winner* V A p a

**shows** *defer* m V A p = {a}

**proof** (rule *ccontr*)

**assume** *not-w*: *defer* m V A p  $\neq$  {a}

**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}. *wins* V a p b)

**using** *winner*

**by** *auto*

**hence** *card* (*defer* m V A p) = 1

**using** *Suc-leI card-gt-0-iff def-one equals0D*

**unfolding** *One-nat-def defers-def*

**by** *metis*

**hence**  $\exists$  b  $\in$  A. *defer* m V A p = {b}

**using** *card-1-singletonE dd defer-in-alts insert-subset c-win*

**unfolding** *defer-deciding-def*

**by** *metis*

**hence**  $\exists$  b  $\in$  A. b  $\neq$  a  $\wedge$  *defer* m V A p = {b}

**using** *not-w*

**by** *metis*

**hence** *not-in-defer*: a  $\notin$  *defer* m V A p

**by** *auto*

**have** *non-electing* m

**using** *dd*

**unfolding** *defer-deciding-def*

**by** *simp*

**hence** a  $\notin$  *elect* m V A p

```

    using c-win equals0D
    unfolding non-electing-def
    by simp
  hence  $a \in \text{reject } m \ V \ A \ p$ 
    using not-in-defer ccomp c-win electoral-mod-defer-elem
    unfolding condorcet-compatibility-def
    by metis
  moreover have  $a \notin \text{reject } m \ V \ A \ p$ 
    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  $SCF\text{-result.electoral-module } m$ 
    using dd
    unfolding defer-deciding-def
    by metis
next
fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 
  assume c-winner: condorcet-winner  $V \ A \ p \ a$ 
  hence elect-empty:  $\text{elect } m \ V \ A \ p = \{\}$ 
    using dd
    unfolding defer-deciding-def non-electing-def
    by simp
  have cond-winner-a:  $\{a\} = \{c \in A. \text{condorcet-winner } V \ A \ p \ c\}$ 
    using cond-winner-unique c-winner
    by metis
  have defer-a:  $\text{defer } m \ V \ A \ p = \{a\}$ 
    using c-winner dd ccomp ccomp-and-dd-imp-def-only-winner
    by simp
  hence  $\text{reject } m \ V \ A \ p = A - \text{defer } m \ V \ A \ p$ 
    using Diff-empty dd reject-not-elec-or-def c-winner elect-empty
    unfolding defer-deciding-def condorcet-winner.simps
    by metis
  hence  $m \ V \ A \ p = (\{\}, A - \text{defer } m \ V \ A \ p, \{a\})$ 
    using elect-empty defer-a elect-rej-def-combination

```



```

    by metis
  thus  $m \ V \ A \ p = (\{\}, A - \text{defer } m \ V \ A \ p, \{c \in A. \text{condorcet-winner } V \ A \ p \ c\})$ 
    using cond-winner-a
    by simp
qed

```

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

```

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

```

```

    by metis
  thus  $\exists B \subseteq A.$ 
    ( $\forall a \in B.$ 
      indep-of-alt  $n \ V \ A \ a \wedge (\forall p. \text{profile } V \ A \ p \longrightarrow a \in \text{reject } n \ V \ A \ p)) \wedge$ 
      ( $\forall a \in A - B.$ 
        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  $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow \text{bool}$   
**where**

```

condorcet-consistency  $m \equiv$ 
  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  $m =$ 
    (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

```

**next**

```

fix
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 
assume
  condorcet-consistency  $m$  and
  condorcet-winner  $V \ A \ p \ a$ 
thus  $m \ V \ A \ p = (\{a\}, A - \text{elect } m \ V \ A \ p, \{\})$ 

```

```

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

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

```

```

show  $m \ V \ A \ p = (\{a\}, A - \text{elect } m \ V \ A \ p, \{\})$ 
  using cc c-win fst-conv
  by metis
qed

```

### (Weak) Monotonicity

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

```

definition monotonicity :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$  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
     $m :: ('a, 'v, 'r) \text{ Electoral-Module}$  and
     $r :: ('a, 'v) \text{ Election rel}$ 
  shows  $(\text{is-symmetry } (\text{fun}_{\mathcal{E}} \ m) \ (\text{Invariance } r)) = (\text{fun}_{\mathcal{E}} \ m \ \text{respects } r)$ 
  unfolding is-symmetry.simps congruent-def
  by blast

```

```

lemma invariance-is-congruence':
  fixes
     $f :: 'x \Rightarrow 'y$  and
     $r :: 'x \text{ rel}$ 
  shows  $(\text{is-symmetry } f \ (\text{Invariance } r)) = (f \ \text{respects } r)$ 
  unfolding is-symmetry.simps congruent-def
  by blast

```

```

theorem pass-to-election-quotient:
  fixes
     $m :: ('a, 'v, 'r) \text{ Electoral-Module}$  and
     $r :: ('a, 'v) \text{ Election rel}$  and
     $X :: ('a, 'v) \text{ Election set}$ 
  assumes
    equiv  $X \ r$  and

```

```

    is-symmetry (funE m) (Invariance r)
  shows  $\forall A \in X // r. \forall E \in A. \pi_Q (fun_E m) A = fun_E m E$ 
  using invariance-is-congruence pass-to-quotient assms
  by blast
end

```

## 4.6 Evaluation Function

```

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

```

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

### 4.6.1 Definition

```

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

```

### 4.6.2 Property

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

```

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

```

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

```

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

```

### 4.6.3 Theorems

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

```

theorem cond-winner-imp-max-eval-val:
  fixes

```

```

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

```

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

**theorem** *non-cond-winner-not-max-eval:*  
**fixes**  
 e :: ('a, 'v) Evaluation-Function **and**  
 A :: 'a set **and**  
 V :: 'v set **and**

```

  p :: ('a, 'v) Profile and
  a :: 'a and
  b :: 'a
assumes
  rating: condorcet-rating e and
  f-prof: finite-profile V A p and
  winner: condorcet-winner V A p a and
  lin-A: b ∈ 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 e V a A p = 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*  $\Rightarrow$  ('a, 'v) *Electoral-Set* **where**  
*elimination-set*  $e \ t \ r \ V \ A \ p = \{a \in A . r \ (e \ V \ a \ A \ p) \ t\}$

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

#### 4.7.2 Social Choice Definitions

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

#### 4.7.3 Common 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*  $e \ V \ A \ p =$   
*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*  $e \ V \ A \ p =$   
*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) *Evaluation-Function* **and**  
 *t* :: *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 Only participating voters impact the result

**lemma** *voters-determine-elim-mod*[simp]:  
**fixes**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **and**  
 $t :: \text{Threshold-Value}$  **and**  
 $r :: \text{Threshold-Relation}$   
**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**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$   
**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*  $e\ t\ r\ V\ A\ p = \text{elimination-set}\ e\ t\ r\ V\ A\ p'$   
**unfolding** *elimination-set.simps*  
**by** *presburger*  
**thus** (*if* *elimination-set*  $e\ t\ r\ V\ A\ p \neq A$   
*then*  $(\{\}, \text{elimination-set}\ e\ t\ r\ V\ A\ p, A - \text{elimination-set}\ e\ t\ r\ V\ A\ p)$   
*else*  $(\{\}, \{\}, A) =$   
*(if* *elimination-set*  $e\ t\ r\ V\ A\ p' \neq A$   
*then*  $(\{\}, \text{elimination-set}\ e\ t\ r\ V\ A\ p', A - \text{elimination-set}\ e\ t\ r\ V\ A\ p')$   
*else*  $(\{\}, \{\}, A)$ )  
**by** *presburger*  
**qed**

**lemma** *voters-determine-less-elim*[simp]:  
**fixes**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **and**  
 $t :: \text{Threshold-Value}$   
**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**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **and**  
 $t :: \text{Threshold-Value}$   
**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 :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile
  assume coinciding: ∀ v ∈ 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  $\text{less-eliminator } e \ (\text{Max } \{e \ V \ x \ A \ p \mid x. x \in A\}) \ V \ A \ p =$ 
     $\text{less-eliminator } e \ (\text{Max } \{e \ V \ x \ A \ p' \mid x. x \in A\}) \ V \ A \ p'$ 
  using coinciding assms voters-determine-less-elim
  unfolding voters-determine-election.simps
  by (metis (no-types, lifting))
qed

lemma voters-determine-min-elim[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  assumes voters-determine-evaluation e
  shows voters-determine-election (min-eliminator e)
proof (unfold min-eliminator.simps voters-determine-election.simps, safe)
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile
  assume
    coinciding: ∀ v ∈ 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  $\text{leq-eliminator } e \ (\text{Min } \{e \ V \ x \ A \ p \mid x. x \in A\}) \ V \ A \ p =$ 
     $\text{leq-eliminator } e \ (\text{Min } \{e \ V \ x \ A \ p' \mid x. x \in A\}) \ V \ A \ p'$ 
  using coinciding assms voters-determine-leq-elim

```

```

    unfolding voters-determine-election.simps
    by (metis (no-types, lifting))
qed

lemma voters-determine-less-avg-elim[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  assumes voters-determine-evaluation e
  shows voters-determine-election (less-average-eliminator e)
proof (unfold less-average-eliminator.simps voters-determine-election.simps, safe)
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile
  assume coinciding:  $\forall v \in V. p\ v = p'\ v$ 
  hence  $\forall x \in A. e\ V\ x\ A\ p = e\ V\ x\ A\ p'$ 
    using assms
    unfolding voters-determine-election.simps
    by simp
  hence  $average\ e\ V\ A\ p = average\ e\ V\ A\ p'$ 
    unfolding average.simps
    by auto
  thus less-eliminator e (average e V A p) V A p =
    less-eliminator e (average e V A p') V A p'
    using coinciding assms voters-determine-less-elim
    unfolding voters-determine-election.simps
    by (metis (no-types, lifting))
qed

lemma voters-determine-leq-avg-elim[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  assumes voters-determine-evaluation e
  shows voters-determine-election (leq-average-eliminator e)
proof (unfold leq-average-eliminator.simps voters-determine-election.simps, safe)
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    p' :: ('a, 'v) Profile
  assume coinciding:  $\forall v \in V. p\ v = p'\ v$ 
  hence  $\forall x \in A. e\ V\ x\ A\ p = e\ V\ x\ A\ p'$ 
    using assms
    unfolding voters-determine-election.simps
    by simp
  hence  $average\ e\ V\ A\ p = average\ e\ V\ A\ p'$ 
    unfolding average.simps
    by auto
  thus leq-eliminator e (average e V A p) V A p =
    leq-eliminator e (average e V A p') V A p'

```

```

    using coinciding assms voters-determine-leq-elim
    unfolding voters-determine-election.simps
    by (metis (no-types, lifting))
qed

```

#### 4.7.6 Non-Blocking

```

lemma elim-mod-non-blocking:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    t :: Threshold-Value and
    r :: Threshold-Relation
  shows non-blocking (elimination-module e t r)
  unfolding non-blocking-def
  by auto

```

```

lemma less-elim-non-blocking:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    t :: Threshold-Value
  shows non-blocking (less-eliminator e t)
  unfolding less-eliminator.simps
  using elim-mod-non-blocking
  by auto

```

```

lemma leq-elim-non-blocking:
  fixes
    e :: ('a, 'v) Evaluation-Function and
    t :: Threshold-Value
  shows non-blocking (leq-eliminator e t)
  unfolding leq-eliminator.simps
  using elim-mod-non-blocking
  by auto

```

```

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

```

```

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

```

```

lemma less-avg-elim-non-blocking:

```

**fixes**  $e :: ('a, 'v) \text{ Evaluation-Function}$   
**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**  $e :: ('a, 'v) \text{ Evaluation-Function}$   
**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**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **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**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **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**  
 $e :: ('a, 'v) \text{ Evaluation-Function}$  **and**  
 $t :: \text{Threshold-Value}$   
**shows** *non-electing* (*leq-eliminator*  $e$   $t$ )  
**unfolding** *non-electing-def*  
**by** *force*

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

**lemma** *min-elim-non-electing*:  
**fixes**  $e :: ('a, 'v) \text{ Evaluation-Function}$

**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-win*: *condorcet-winner* *V* *A* *p* *a* **and**  
*rej-a*: *a* ∈ *reject* (*max-eliminator* *e*) *V* *A* *p*  
**have** *e* *V* *a* *A* *p* = *Max* {*e* *V* *b* *A* *p* | *b*. *b* ∈ *A*}  
**using** *c-win cond-winner-imp-max-eval-val assms*  
**by** *fastforce*  
**hence** *a* ∉ *reject* (*max-eliminator* *e*) *V* *A* *p*  
**by** *simp*  
**thus** *False*  
**using** *rej-a*  
**by** *linarith*  
**next**  
**fix**  
*A* :: 'a *set* **and**  
*V* :: 'v *set* **and**  
*p* :: ('a, 'v) *Profile* **and**

```

  a :: 'a
assume a ∈ elect (max-eliminator e) V A p
moreover have a ∉ elect (max-eliminator e) V A p
  by simp
ultimately show False
  by linarith
next
fix
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  a :: 'a and
  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 condorcet-winner.elims(2) empty-iff max-elim-non-electing
  unfolding non-electing-def
  by metis
qed

```

If the used evaluation function is Condorcet rating, max-eliminator is defer-Condorcet-consistent.

```

theorem cr-eval-imp-dcc-max-elim[simp]:
  fixes e :: ('a, 'v) Evaluation-Function
  assumes condorcet-rating e
  shows defer-condorcet-consistency (max-eliminator e)
proof (unfold defer-condorcet-consistency-def, safe)
  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  $e \ V \ a \ A \ p = \text{Max } \{e \ V \ x \ A \ p \mid x. x \in A\}$ 
  using winner assms cond-winner-imp-max-eval-val
  by fastforce
hence  $\forall b \in A. b \neq a \longleftrightarrow b \in \{c \in A. e \ V \ c \ A \ p < \text{Max } \{e \ V \ b \ A \ p \mid b. b \in A\}\}$ 
  using winner assms mem-Collect-eq linorder-neq-iff
  unfolding condorcet-rating-def
  by (metis (mono-tags, lifting))
hence elim-set:  $(\text{elimination-set } e \ ?trsh \ (<) \ V \ A \ p) = A - \{a\}$ 
  unfolding elimination-set.simps
  by blast
case True
hence
   $\text{max-eliminator } e \ V \ A \ p =$ 
    ( $\{\},$ 
       $(\text{elimination-set } e \ ?trsh \ (<) \ V \ A \ p),$ 
       $A - (\text{elimination-set } e \ ?trsh \ (<) \ V \ A \ p)$ )
  by simp
also have  $\dots = (\{\}, A - \{a\}, \{a\})$ 
  using elim-set winner
  by auto
also have  $\dots = (\{\}, A - \text{defer } (\text{max-eliminator } e) \ V \ A \ p, \{a\})$ 
  using calculation
  by simp
also have
   $\dots = (\{\},$ 
     $A - \text{defer } (\text{max-eliminator } e) \ V \ A \ p,$ 
     $\{b \in A. \text{condorcet-winner } V \ A \ p \ b\})$ 
  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*  $\equiv$

*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*  $\equiv$

*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') =
    (e  $\cup$  e',
     A - (e  $\cup$  e'  $\cup$  d  $\cup$  d'),
     (d  $\cup$  d') - (e  $\cup$  e'))
```

### 4.9.2 Auxiliary Lemma

```
lemma max-agg-rej-set:
  fixes
    A :: 'a set and
    e :: 'a set and
    e' :: 'a set and
    d :: 'a set and
    d' :: 'a set and
    r :: 'a set and
    r' :: 'a set and
    a :: 'a
  assumes
    wf-first-mod: well-formed-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 :: 'a set and
  e :: 'a set and
  e' :: 'a set and
  d :: 'a set and
  d' :: 'a set and
  r :: 'a set and
  r' :: 'a set and
  a :: 'a
assume
  e' ∪ r' ∪ d' = e ∪ r ∪ d and
  a ∉ d and
  a ∉ r and
  a ∈ e'
thus a ∈ e
  by auto
next
fix
  A :: 'a set and
  e :: 'a set and
  e' :: 'a set and
  d :: 'a set and
  d' :: 'a set and
  r :: 'a set and
  r' :: 'a set and
  a :: 'a
assume
  e' ∪ r' ∪ d' = e ∪ r ∪ d and
  a ∉ d and
  a ∉ r and
  a ∈ d'
thus a ∈ e
  by auto
qed

```

### 4.9.4 Properties

The max-aggregator is conservative.

**theorem** *max-agg-consv[simp]: agg-conservative max-aggregator*

**proof** (*unfold agg-conservative-def, safe*)

```

show aggregator max-aggregator
  using max-agg-sound
  by metis
next

```

```

fix
   $A :: 'a \text{ set}$  and
   $e :: 'a \text{ set}$  and
   $e' :: 'a \text{ set}$  and
   $d :: 'a \text{ set}$  and
   $d' :: 'a \text{ set}$  and
   $r :: 'a \text{ set}$  and
   $r' :: 'a \text{ set}$  and
   $a :: 'a$ 
assume
  elect-a:  $a \in \text{elect-}r \text{ (max-aggregator } A \text{ (} e, r, d \text{) (} e', r', d' \text{))}$  and
  a-not-in-e':  $a \notin e'$ 
have  $a \in e \cup e'$ 
  using elect-a
  by simp
thus  $a \in e$ 
  using a-not-in-e'
  by simp
next
fix
   $A :: 'a \text{ set}$  and
   $e :: 'a \text{ set}$  and
   $e' :: 'a \text{ set}$  and
   $d :: 'a \text{ set}$  and
   $d' :: 'a \text{ set}$  and
   $r :: 'a \text{ set}$  and
   $r' :: 'a \text{ set}$  and
   $a :: 'a$ 
assume
  wf-result: well-formed-SCF  $A \text{ (} e', r', d' \text{)}$  and
  reject-a:  $a \in \text{reject-}r \text{ (max-aggregator } A \text{ (} e, r, d \text{) (} e', r', d' \text{))}$  and
  a-not-in-r':  $a \notin r'$ 
have  $a \in r \cup r'$ 
  using wf-result reject-a
  by force
thus  $a \in r$ 
  using a-not-in-r'
  by simp
next
fix
   $A :: 'a \text{ set}$  and
   $e :: 'a \text{ set}$  and
   $e' :: 'a \text{ set}$  and
   $d :: 'a \text{ set}$  and
   $d' :: 'a \text{ set}$  and
   $r :: 'a \text{ set}$  and
   $r' :: 'a \text{ set}$  and
   $a :: 'a$ 
assume

```

```

    defer-a:  $a \in \text{defer-}r \text{ (max-aggregator } A \text{ (} e, r, d \text{) (} e', r', d' \text{))}$  and
    a-not-in-d':  $a \notin d'$ 
have  $a \in d \cup d'$ 
    using defer-a
    by force
thus  $a \in d$ 
    using a-not-in-d'
    by simp
qed

```

The max-aggregator is commutative.

```

theorem max-agg-comm[simp]: agg-commutative max-aggregator
  unfolding agg-commutative-def
  by auto
end

```

## 4.10 Termination Condition

```

theory Termination-Condition
  imports Social-Choice-Types/Result
begin

```

The termination condition is used in loops. It decides whether or not to terminate the loop after each iteration, depending on the current state of the loop.

### 4.10.1 Definition

```

type-synonym 'r Termination-Condition = 'r Result  $\Rightarrow$  bool
end

```

## 4.11 Defer Equal Condition

```

theory Defer-Equal-Condition
  imports Termination-Condition
begin

```

This is a family of termination conditions. For a natural number  $n$ , the according defer-equal condition is true if and only if the given result's defer-set contains exactly  $n$  elements.

#### 4.11.1 Definition

```
fun defer-equal-condition :: nat  $\Rightarrow$  'a Termination-Condition where  
  defer-equal-condition n (e, r, d) = (card d = n)  
end
```

## Chapter 5

# Basic Modules

### 5.1 Defer Module

```
theory Defer-Module
  imports Component-Types/Electoral-Module
begin
```

The defer module is not concerned about the voter's ballots, and simply defers all alternatives. It is primarily used for defining an empty loop.

#### 5.1.1 Definition

```
fun defer-module :: ('a, 'v, 'a Result) Electoral-Module where
  defer-module V A p = ({}, {}, A)
```

#### 5.1.2 Soundness

```
theorem def-mod-sound[simp]: 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*)

```

  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**  $(\text{'a}, \text{'v}, \text{'r})$  *Consensus-Class* =  $(\text{'a}, \text{'v})$  *Consensus*  $\times$   $(\text{'a}, \text{'v}, \text{'r})$  *Electoral-Module*

**fun** *consensus-K* ::  $(\text{'a}, \text{'v}, \text{'r})$  *Consensus-Class*  $\Rightarrow$   $(\text{'a}, \text{'v})$  *Consensus*  
**where** *consensus-K* *K* = *fst K*

**fun** *rule-K* ::  $(\text{'a}, \text{'v}, \text{'r})$  *Consensus-Class*  $\Rightarrow$   $(\text{'a}, \text{'v}, \text{'r})$  *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** *K<sub>E</sub>* ::  $(\text{'a}, \text{'v}, \text{'r})$  *Result* *Consensus-Class*  $\Rightarrow$   $\text{'r}$   $\Rightarrow$   $(\text{'a}, \text{'v})$  *Election set* **where**  
*K<sub>E</sub>* *K* *w* =  
 $\{(A, V, p) \mid A \ V \ p. (\text{consensus-K } K) (A, V, p) \wedge \text{finite-profile } V \ A \ p$   
 $\wedge \text{elect } (\text{rule-K } K) \ V \ A \ p = \{w\}\}$

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

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* ::  $(\text{'a}, \text{'v})$  *Consensus*  $\Rightarrow$   $(\text{'a}, \text{'v}, \text{'r})$  *Electoral-Module*  $\Rightarrow$  *bool*  
**where**  
*well-formed* *c* *m*  $\equiv$   
 $\forall \ A \ V \ V' \ p \ p'. \text{profile } V \ A \ p \wedge \text{profile } V' \ A \ p' \wedge c \ (A, V, p) \wedge c \ (A, V', p')$   
 $\longrightarrow$   
 $m \ V \ A \ p = m \ V' \ A \ p'$

A sensible social choice rule for a given arbitrary consensus and social choice rule  $r$  is the one that chooses the result of  $r$  for all consensus elections and defers all candidates otherwise.

```
fun consensus-choice :: ('a, 'v) Consensus  $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module
   $\Rightarrow$  ('a, 'v, 'a Result) Consensus-Class where
  consensus-choice c m =
    (let
      w = ( $\lambda$  V A p. if c (A, V, p) then m V A p else defer-module V A p)
    in (c, w))
```

### 5.3.3 Auxiliary Lemmas

**lemma** unanimity'-consensus-imp-elect-fst-mod-well-formed:

**fixes** a :: 'a

**shows** well-formed ( $\lambda$  c. nonempty-set<sub>C</sub> c  $\wedge$  nonempty-profile<sub>C</sub> c  $\wedge$  equal-top<sub>C</sub>' a c)

*elect-first-module*

**proof** (unfold well-formed-def, safe)

**fix**

a :: 'a **and**

A :: 'a set **and**

V :: 'v::wellorder set **and**

V' :: 'v set **and**

p :: ('a, 'v) Profile **and**

p' :: ('a, 'v) Profile

**let** ?cond =  $\lambda$  c. nonempty-set<sub>C</sub> c  $\wedge$  nonempty-profile<sub>C</sub> c  $\wedge$  equal-top<sub>C</sub>' a c

**assume**

prof-p: profile V A p **and**

prof-p': profile V' A p' **and**

eq-top-p: equal-top<sub>C</sub>' a (A, V, p) **and**

eq-top-p': equal-top<sub>C</sub>' a (A, V', p') **and**

not-empty-A: nonempty-set<sub>C</sub> (A, V, p) **and**

not-empty-A': nonempty-set<sub>C</sub> (A, V', p') **and**

not-empty-p: nonempty-profile<sub>C</sub> (A, V, p) **and**

not-empty-p': nonempty-profile<sub>C</sub> (A, V', p')

**hence**

cond-Ap: ?cond (A, V, p) **and**

cond-Ap': ?cond (A, V', p')

**by** simp-all

**have**  $\forall a' \in A. ((\text{above } (p \text{ (least } V)) \ a' = \{a'\}) = (\text{above } (p' \text{ (least } V')) \ a' = \{a'\}))$

**proof**

**fix** a' :: 'a

**assume** a'-in-A: a'  $\in$  A

**show** (above (p (least V)) a' = {a'}) = (above (p' (least V')) a' = {a'})

**proof** (cases)

**assume** a' = a

**thus** ?thesis

**using** cond-Ap cond-Ap' Collect-mem-eq LeastI empty-Collect-eq equal-top<sub>C</sub>'.simps

```

      nonempty-profileC.simps least.simps
    by (metis (no-types, lifting))
  next
    assume a'-neq-a: a' ≠ a
    have non-empty: V ≠ {} ∧ V' ≠ {}
      using not-empty-p not-empty-p'
      by simp
    hence A ≠ {} ∧ linear-order-on A (p (least V))
      ∧ linear-order-on A (p' (least V'))
      using not-empty-A not-empty-A' prof-p prof-p'
      a'-in-A card.remove enumerate.simps(1)
      enumerate-in-set finite-enumerate-in-set
      least.elims all-not-in-conv
      zero-less-Suc
    unfolding profile-def
    by metis
    hence (a ∈ above (p (least V)) a' ∨ a' ∈ above (p (least V)) a) ∧
      (a ∈ above (p' (least V')) a' ∨ a' ∈ above (p' (least V')) a)
      using a'-in-A a'-neq-a eq-top-p
    unfolding above-def linear-order-on-def total-on-def
    by auto
    hence (above (p (least V)) a = {a} ∧ above (p (least V)) a' = {a'}) → a =
a') ∧
      (above (p' (least V')) a = {a} ∧ above (p' (least V')) a' = {a'}) → a
= a')
      by auto
    thus ?thesis
      using bot-nat-0.not-eq-extremum card-0-eq cond-Ap cond-Ap'
      enumerate.simps(1) enumerate-in-set equal-topC'.simps
      finite-enumerate-in-set non-empty least.simps
      by metis
  qed
qed
thus elect-first-module V A p = elect-first-module V' A p'
  by auto
qed

```

**lemma** *strong-unanimity'consensus-imp-elect-fst-mod-completely-determined:*

**fixes**  $r :: 'a$  *Preference-Relation*

**shows** *well-formed*

( $\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C' r c$ ) *elect-first-module*

**proof** (*unfold well-formed-def, clarify*)

**fix**

$a :: 'a$  **and**

$A :: 'a$  *set* **and**

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

$V' :: 'v$  *set* **and**

$p :: ('a, 'v)$  *Profile* **and**

$p' :: ('a, 'v)$  *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-vote-p:  $\text{equal-vote}_C' r (A, V, p)$  and
  eq-vote-p':  $\text{equal-vote}_C' r (A, V', p')$  and
  not-empty-A:  $\text{nonempty-set}_C (A, V, p)$  and
  not-empty-A':  $\text{nonempty-set}_C (A, V', p')$  and
  not-empty-p:  $\text{nonempty-profile}_C (A, V, p)$  and
  not-empty-p':  $\text{nonempty-profile}_C (A, V', p')$ 
hence
  cond-Ap: ?cond  $(A, V, p)$  and
  cond-Ap': ?cond  $(A, V', p')$ 
by simp-all
have  $p (\text{least } V) = r \wedge p' (\text{least } V') = r$ 
using eq-vote-p eq-vote-p' not-empty-p not-empty-p'
  bot-nat-0.not-eq-extremum card-0-eq enumerate.simps(1)
  enumerate-in-set equal-vote_C'.simps finite-enumerate-in-set
  nonempty-profile_C.simps least.elims
by (metis (no-types, lifting))
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-well-formed:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  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
  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- $\mathcal{K} C \subseteq \text{valid-elections}$ 
proof
  fix  $E :: ('a, 'v) \text{ Election}$ 
  assume  $E \in \text{elections-}\mathcal{K} C$ 
  hence fun $_{\mathcal{E}}$  profile  $E$ 
    unfolding  $\mathcal{K}_{\mathcal{E}}.simps$ 
    by force
  thus  $E \in \text{valid-elections}$ 
    unfolding valid-elections-def
    by simp
qed

lemma cons-domain-finite:
  fixes  $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
  shows

```

*finite: elections- $\mathcal{K}$   $C \subseteq \text{finite-elections}$  and*  
*finite-voters: elections- $\mathcal{K}$   $C \subseteq \text{finite-elections-}\mathcal{V}$*   
**proof** –  
**have**  $\forall E \in \text{elections-}\mathcal{K} \ C. \text{fun}_{\mathcal{E}} \text{ profile } E \wedge \text{finite} (\text{alternatives-}\mathcal{E} \ E) \wedge \text{finite} (\text{voters-}\mathcal{E} \ E)$   
**unfolding**  $\mathcal{K}_{\mathcal{E}}.\text{sims}$   
**by force**  
**thus**  $\text{elections-}\mathcal{K} \ C \subseteq \text{finite-elections}$   
**unfolding**  $\text{finite-elections-def fun}_{\mathcal{E}}.\text{sims}$   
**by blast**  
**thus**  $\text{elections-}\mathcal{K} \ C \subseteq \text{finite-elections-}\mathcal{V}$   
**unfolding**  $\text{finite-elections-def finite-elections-}\mathcal{V}\text{-def}$   
**by blast**  
**qed**

### 5.3.4 Consensus Rules

**definition**  $\text{non-empty-set} :: ('a, 'v, 'r) \text{Consensus-Class} \Rightarrow \text{bool}$  **where**  
 $\text{non-empty-set } c \equiv \exists K. \text{consensus-}\mathcal{K} \ c \ K$

Unanimity condition.

**definition**  $\text{unanimity} :: ('a, 'v::\text{wellorder}, 'a \text{Result}) \text{Consensus-Class}$  **where**  
 $\text{unanimity} = \text{consensus-choice unanimity}_{\mathcal{C}} \text{ elect-first-module}$

Strong unanimity condition.

**definition**  $\text{strong-unanimity} :: ('a, 'v::\text{wellorder}, 'a \text{Result}) \text{Consensus-Class}$  **where**  
 $\text{strong-unanimity} = \text{consensus-choice strong-unanimity}_{\mathcal{C}} \text{ elect-first-module}$

### 5.3.5 Properties

**definition**  $\text{consensus-rule-anonymity} :: ('a, 'v, 'r) \text{Consensus-Class} \Rightarrow \text{bool}$  **where**  
 $\text{consensus-rule-anonymity } c \equiv$   
 $(\forall A \ V \ p \ \pi::('v \Rightarrow 'v)).$   
 $\text{bij } \pi \longrightarrow$   
 $(\text{let } (A', V', q) = (\text{rename } \pi \ (A, V, p)) \text{ in}$   
 $\text{profile } V \ A \ p \longrightarrow \text{profile } V' \ A' \ q$   
 $\longrightarrow \text{consensus-}\mathcal{K} \ c \ (A, V, p)$   
 $\longrightarrow (\text{consensus-}\mathcal{K} \ c \ (A', V', q) \wedge (\text{rule-}\mathcal{K} \ c \ V \ A \ p = \text{rule-}\mathcal{K} \ c \ V' \ A' \ q))))$

**fun**  $\text{consensus-rule-anonymity}' :: ('a, 'v) \text{Election set} \Rightarrow ('a, 'v, 'r \text{Result}) \text{Consensus-Class}$   
 $\Rightarrow \text{bool}$  **where**  
 $\text{consensus-rule-anonymity}' \ X \ C =$   
 $\text{is-symmetry} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{Invariance} (\text{anonymity}_{\mathcal{R}} \ X))$

**fun** (**in**  $\text{result-properties}$ )  $\text{consensus-rule-neutrality} :: ('a, 'v) \text{Election set}$   
 $\Rightarrow ('a, 'v, 'b \text{Result}) \text{Consensus-Class} \Rightarrow \text{bool}$  **where**  
 $\text{consensus-rule-neutrality } X \ C = \text{is-symmetry} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$   
 $(\text{action-induced-equivariance} (\text{carrier neutrality}_{\mathcal{G}}) \ X \ (\varphi\text{-neutr } X) (\text{set-action } \psi\text{-neutr}))$

```

fun consensus-rule-reversal-symmetry :: ('a, 'v) Election set
  ⇒ ('a, 'v, 'a rel Result) Consensus-Class ⇒ bool where
  consensus-rule-reversal-symmetry X C = is-symmetry (elect-r ∘ funε (rule- $\mathcal{K}$  C))
    (action-induced-equivariance (carrier reversalG) X (φ-rev X) (set-action ψ-rev))

```

### 5.3.6 Inference Rules

**lemma** consensus-choice-equivar:

```

fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  c :: ('a, 'v) Consensus and
  G :: 'x set and
  X :: ('a, 'v) Election set and
  φ :: ('x, ('a, 'v) Election) binary-fun and
  ψ :: ('x, 'a) binary-fun and
  f :: 'a Result ⇒ 'a set
defines equivar ≡ action-induced-equivariance G X φ (set-action ψ)
assumes
  equivar-m: is-symmetry (f ∘ funε m) equivar and
  equivar-defer: is-symmetry (f ∘ funε defer-module) equivar and
  — This could be generalized to arbitrary modules instead of defer-module.
  invar-cons: is-symmetry c (Invariance (action-induced-rel G X φ))
shows is-symmetry (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m)))
  (action-induced-equivariance G X φ (set-action ψ))
proof (unfold rewrite-equivariance, intro ballI impI)
fix
  E :: ('a, 'v) Election and
  g :: 'x
assume
  g-in-G: g ∈ G and
  E-in-X: E ∈ X and
  φ-g-E-in-X: φ g E ∈ X
show (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) (φ g E) =
  set-action ψ g ((f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) E)
proof (cases c E)
case True
hence c (φ g E)
  using invar-cons rewrite-invar-ind-by-act g-in-G φ-g-E-in-X E-in-X
  by metis
hence (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) (φ g E) = (f ∘ funε m) (φ
g E)
  by simp
also have (f ∘ funε m) (φ g E) =
  set-action ψ g ((f ∘ funε m) E)
  using equivar-m E-in-X φ-g-E-in-X g-in-G rewrite-equivariance
  unfolding equivar-def
  by (metis (mono-tags, lifting))
also have (f ∘ funε m) E =

```

```

    (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) E
  using True E-in-X g-in-G invar-cons
  by simp
  finally show ?thesis
  by simp
next
case False
hence ¬ c (φ g E)
  using invar-cons rewrite-invar-ind-by-act g-in-G φ-g-E-in-X E-in-X
  by metis
hence (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) (φ g E) =
  (f ∘ funε defer-module) (φ g E)
  by simp
also have (f ∘ funε defer-module) (φ g E) =
  set-action ψ g ((f ∘ funε defer-module) E)
  using equivar-defer E-in-X g-in-G φ-g-E-in-X rewrite-equivariance
  unfolding equivar-def
  by (metis (mono-tags, lifting))
also have (f ∘ funε defer-module) E =
  (f ∘ funε (rule- $\mathcal{K}$  (consensus-choice c m))) E
  using False E-in-X g-in-G invar-cons
  by simp
  finally show ?thesis
  by simp
qed
qed

lemma consensus-choice-anonymous:
  fixes
    α :: ('a, 'v) Consensus and
    β :: ('a, 'v) Consensus and
    m :: ('a, 'v, 'a Result) Electoral-Module and
    β' :: 'b ⇒ ('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 set and
    A' :: 'a set and
    V :: 'v set and
    V' :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and
    π :: 'v ⇒ 'v
  assume

```



**bij**: *bij*  $\pi$  **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**: *consensus- $\mathcal{K}$*  (*consensus-choice*  $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ m$ )  $(A,\ V,$   
 $p)$   
**hence**  $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ (A,\ V,\ p)$   
**by** *simp*  
**hence**  
**alpha-Ap**:  $\alpha\ (A,\ V,\ p)$  **and**  
**beta-Ap**:  $\beta\ (A,\ V,\ p)$   
**by** *simp-all*  
**have** *alpha-A-perm-p*:  $\alpha\ (A',\ V',\ q)$   
**using** *anon-cons-cond* *alpha-Ap* *bij* *prof-p* *prof-q* *renamed*  
**unfolding** *consensus-anonymity-def*  
**by** *fastforce*  
**moreover** **have**  $\beta\ (A',\ V',\ q)$   
**using** *beta'-anon* *beta-Ap* *beta-sat* *ex-anon-cons-imp-cons-anonymous* [*of*  $\beta\ \beta'$ ]  
*bij*  
*prof-p* *renamed* *beta'-anon* *cons-anon-invariant* [*of*  $\beta$ ]  
**unfolding** *consensus-anonymity-def*  
**by** *blast*  
**ultimately show** *em-cond-perm*:  
*consensus- $\mathcal{K}$*  (*consensus-choice*  $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ m$ )  $(A',\ V',\ q)$   
**using** *beta-Ap* *beta-sat* *ex-anon-cons-imp-cons-anonymous* *bij*  
*prof-p* *prof-q*  
**by** *simp*  
**have**  $\exists\ x.\ \beta'\ x\ (A,\ V,\ p)$   
**using** *beta-Ap* *beta-sat*  
**by** *simp*  
**then obtain**  $x$  **where**  
*beta'-x-Ap*:  $\beta'\ x\ (A,\ V,\ p)$   
**by** *metis*  
**hence** *beta'-x-A-perm-p*:  $\beta'\ x\ (A',\ V',\ q)$   
**using** *beta'-anon* *bij* *prof-p* *renamed*  
*cons-anon-invariant* *prof-q*  
**unfolding** *consensus-anonymity-def*  
**by** *blast*  
**have**  $m\ V\ A\ p = m\ V'\ A'\ q$   
**using** *alpha-Ap* *alpha-A-perm-p* *beta'-x-Ap* *beta'-x-A-perm-p*  
*conditions-univ* *prof-p* *prof-q* *rename.simps* *prod.inject* *renamed*  
**unfolding** *well-formed-def*  
**by** *metis*  
**thus** *rule- $\mathcal{K}$*  (*consensus-choice*  $(\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ m$ )  $V\ A\ p =$   
 $\text{rule-}\mathcal{K}\ (\text{consensus-choice}\ (\lambda\ E.\ \alpha\ E \wedge \beta\ E)\ m)\ V'\ A'\ q$   
**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<sub>C</sub>* = ( $\lambda c. \exists a. \text{equal-top}_C' a c$ )

**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<sub>C</sub>]*

*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<sub>C</sub> elect-first-module*

**using** *unanimity<sub>C</sub>.simps*

**by** *metis*

**ultimately show** *consensus-rule-anonymity (consensus-choice unanimity<sub>C</sub> 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<sub>C</sub>* = ( $\lambda c. \exists v. \text{equal-vote}_C' v c$ )

**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<sub>C</sub>]*

*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<sub>C</sub> elect-first-module*

**using** *strong-unanimity<sub>C</sub>.elims(2, 3)*

**by** *metis*

**ultimately show**

$\text{consensus-rule-anonymity } (\text{consensus-choice strong-unanimity}_C \text{ elect-first-module})$   
 by (metis (no-types))  
 qed

## Neutrality

**lemma** *defer-winners-equivariant:*

**fixes**  
 $G :: 'x \text{ set and}$   
 $X :: ('a, 'v) \text{ Election set and}$   
 $\varphi :: ('x, ('a, 'v) \text{ Election}) \text{ binary-fun and}$   
 $\psi :: ('x, 'a) \text{ binary-fun}$   
**shows**  $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_E \text{ defer-module})$   
 $(\text{action-induced-equivariance } G \ X \ \varphi \ (\text{set-action } \psi))$   
**using** *rewrite-equivariance*  
**by** *fastforce*

**lemma** *elect-first-winners-neutral: is-symmetry*  $(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module})$   
 $(\text{action-induced-equivariance } (\text{carrier neutrality}_G))$   
 $\text{valid-elections } (\varphi\text{-neutr valid-elections}) \ (\text{set-action } \psi\text{-neutr}_C)$

**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**  
 $\text{bij: } \pi \in \text{carrier neutrality}_G \text{ and}$   
 $\text{valid: } (A, V, p) \in \text{valid-elections}$

**hence**  $\text{bijective-}\pi$ :  $\text{bij } \pi$   
**unfolding** *neutrality<sub>G</sub>-def*  
**using** *rewrite-carrier*  
**by** *blast*

**hence**  $\text{inv: } \forall a. a = \pi \ (\text{the-inv } \pi \ a)$   
**by** (*simp add: f-the-inv-into-f-bij-betw*)

**from** *bij valid* **have**

$(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module}) \ (\varphi\text{-neutr valid-elections } \pi \ (A, V, p)) =$   
 $\{a \in \pi \text{ ' } A. \text{ above } (\text{rel-rename } \pi \ (p \ (\text{least } V))) \ a = \{a\}\}$

**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:*

$(\text{elect-r} \circ \text{fun}_E \text{ elect-first-module}) \ (\varphi\text{-neutr valid-elections } \pi \ (A, V, p)) =$   
 $\{a \in \pi \text{ ' } A. \{b. (a, b) \in \{(\pi \ a, \pi \ b) \mid a \ b. (a, b) \in p \ (\text{least } V)\}\} = \{a\}\}$

**by** *simp*

**have**  $\forall a \in \pi \text{ ' } A. \{b. (a, b) \in \{(\pi \ x, \pi \ y) \mid x \ y. (x, y) \in p \ (\text{least } V)\}\} =$

$\{\pi b \mid b. (a, \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p (least V)\}\}$   
**by** *blast*  
**moreover have**  $\forall a \in \pi ' A.$   
 $\{\pi b \mid b. (a, \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p (least V)\}\} =$   
 $\{\pi b \mid b. (\pi (the-inv \pi a), \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p (least V)\}\}$   
**using** *bijjective- $\pi$*   
**by** (*simp add: f-the-inv-into-f-bij-betw*)  
**moreover have**  $\forall a \in \pi ' A. \forall b.$   
 $((\pi (the-inv \pi a), \pi b) \in \{(\pi x, \pi y) \mid x y. (x, y) \in p (least V)\}) =$   
 $((the-inv \pi a, b) \in \{(x, y) \mid x y. (x, y) \in p (least V)\})$   
**using** *bijjective- $\pi$  rel-rename-helper[*of*  $\pi$ ]*  
**by** *auto*  
**moreover have**  $\{(x, y) \mid x y. (x, y) \in p (least V)\} = p (least V)$   
**by** *simp*  
**ultimately have**  
 $\forall a \in \pi ' A. (\{b. (a, b) \in \{(\pi a, \pi b) \mid a b. (a, b) \in p (least V)\}\} = \{a\}) =$   
 $(\{\pi b \mid b. (the-inv \pi a, b) \in p (least V)\} = \{a\})$   
**by** *force*  
**hence**  $\{a \in \pi ' A. \{b. (a, b) \in \{(\pi a, \pi b) \mid a b. (a, b) \in p (least V)\}\} = \{a\}\}$   
 $=$   
 $\{a \in \pi ' A. \{\pi b \mid b. (the-inv \pi a, b) \in p (least V)\} = \{a\}\}$   
**by** *auto*  
**hence** (*elect-r*  $\circ$  *fun $\mathcal{E}$  elect-first-module*) ( *$\varphi$ -neutr valid-elections*  $\pi (A, V, p)$ ) =  
 $\{a \in \pi ' A. \{\pi b \mid b. (the-inv \pi a, b) \in p (least V)\} = \{a\}\}$   
**using** *elect-simp*  
**by** *simp*  
**also have**  $\{a \in \pi ' A. \{\pi b \mid b. (the-inv \pi a, b) \in p (least V)\} = \{a\}\} =$   
 $\{\pi a \mid a. a \in A \wedge \{\pi b \mid b. (a, b) \in p (least V)\} = \{\pi a\}\}$   
**using** *bijjective- $\pi$  inv bij-is-inj the-inv-f-f*  
**by** *fastforce*  
**also have**  $\{\pi a \mid a. a \in A \wedge \{\pi b \mid b. (a, b) \in p (least V)\} = \{\pi a\}\} =$   
 $\pi ' \{a \in A. \{\pi b \mid b. (a, b) \in p (least V)\} = \{\pi a\}\}$   
**by** *blast*  
**also have**  $\pi ' \{a \in A. \{\pi b \mid b. (a, b) \in p (least V)\} = \{\pi a\}\} =$   
 $\pi ' \{a \in A. \pi ' \{b \mid b. (a, b) \in p (least V)\} = \pi ' \{a\}\}$   
**by** *blast*  
**finally have**  
 $(elect-r \circ fun_{\mathcal{E}} elect-first-module) (\varphi-neutr valid-elections \pi (A, V, p)) =$   
 $\pi ' \{a \in A. \pi ' (above (p (least V)) a) = \pi ' \{a\}\}$   
**unfolding** *above-def*  
**by** *simp*  
**moreover have**  
 $\forall a. (\pi ' (above (p (least V)) a) = \pi ' \{a\}) =$   
 $(the-inv \pi ' \pi ' above (p (least V)) a = the-inv \pi ' \pi ' \{a\})$   
**using**  *$\langle$ bij  $\pi \rangle$  bij-betw-the-inv-into bij-def inj-image-eq-iff*  
**by** *metis*  
**moreover have**  $\forall a. (the-inv \pi ' \pi ' above (p (least V)) a = the-inv \pi ' \pi ' \{a\}) =$   
 $(above (p (least V)) a = \{a\})$

**using** *bijjective- $\pi$  bij-betw-imp-inj-on bij-betw-the-inv-into inj-image-eq-iff*  
**by** *metis*  
**ultimately have**  $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{elect-first-module}) (\varphi\text{-neutr valid-elections } \pi (A, V, p)) =$   
 $\pi \text{ ' } \{a \in A. \text{above } (p \text{ (least } V)) \ a = \{a\}\}$   
**by** *presburger*  
**moreover have**  $\text{elect elect-first-module } V \ A \ p = \{a \in A. \text{above } (p \text{ (least } V)) \ a = \{a\}\}$   
**by** *simp*  
**moreover have** *set-action  $\psi\text{-neutr}_c \pi$*   
 $((\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{elect-first-module}) (A, V, p)) =$   
 $\pi \text{ ' } (\text{elect elect-first-module } V \ A \ p)$   
**by** *auto*  
**ultimately show**  
 $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{elect-first-module}) (\varphi\text{-neutr valid-elections } \pi (A, V, p)) =$   
 $\text{set-action } \psi\text{-neutr}_c \pi$   
 $((\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{elect-first-module}) (A, V, p))$   
**by** *blast*  
**qed**

**lemma** *strong-unanimity-neutral:*

**defines**  $\text{domain} \equiv \text{valid-elections} \cap \text{Collect strong-unanimity}_{\mathcal{C}}$   
 — 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:  $\forall \pi. \forall E \in \text{domain}. \varphi\text{-neutr domain } \pi \ E = \varphi\text{-neutr valid-elections } \pi \ E$*   
**unfolding** *domain-def  $\varphi\text{-neutr.simps}$*   
**by** *auto*  
**have** *consensus-neutrality domain strong-unanimity $_{\mathcal{C}}$*   
**using** *strong-unanimity $_{\mathcal{C}}$ -neutral invar-under-subset-rel*  
**unfolding** *domain-def*  
**by** *simp*  
**hence** *is-symmetry strong-unanimity $_{\mathcal{C}}$*   
 $(\text{Invariance } (\text{action-induced-rel } (\text{carrier neutrality}_{\mathcal{G}}) \text{domain } (\varphi\text{-neutr valid-elections})))$   
**unfolding** *consensus-neutrality.simps neutrality $_{\mathcal{R}}$ .simps*  
**using** *coincides coinciding-actions-ind-equal-rel*  
**by** *metis*  
**moreover have** *is-symmetry  $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} \text{elect-first-module})$*   
 $(\text{action-induced-equivariance } (\text{carrier neutrality}_{\mathcal{G}})$   
 $\text{domain } (\varphi\text{-neutr valid-elections}) (\text{set-action } \psi\text{-neutr}_c))$   
**using** *elect-first-winners-neutral*  
**unfolding** *domain-def action-induced-equivariance-def*  
**using** *equivar-under-subset*  
**by** *blast*  
**ultimately have** *is-symmetry  $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity}))$*   
 $(\text{action-induced-equivariance } (\text{carrier neutrality}_{\mathcal{G}}) \text{domain}$   
 $(\varphi\text{-neutr valid-elections}) (\text{set-action } \psi\text{-neutr}_c))$

```

using defer-winners-equivariant[of
  carrier neutralityG domain  $\varphi$ -neutr valid-elections  $\psi$ -neutrc]
  consensus-choice-equivar[of
    elect-r elect-first-module carrier neutralityG domain
     $\varphi$ -neutr valid-elections  $\psi$ -neutrc strong-unanimityC]
unfolding strong-unanimity-def
by metis
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  $\subseteq$  valid-elections  $\cap$  Collect strong-unanimityC
  unfolding valid-elections-def  $\mathcal{K}_{\mathcal{E}}$ .simps strong-unanimity-def
  by force
  moreover from this have coincide:
     $\forall \pi. \forall E \in \text{elections-}\mathcal{K} \text{ strong-unanimity.}$ 
     $\varphi\text{-neutr (valid-elections} \cap \text{Collect strong-unanimity}_C) \pi E =$ 
     $\varphi\text{-neutr (elections-}\mathcal{K} \text{ strong-unanimity)} \pi E$ 
  unfolding  $\varphi\text{-neutr.simps}$ 
  using extensional-continuation-subset
  by (metis (no-types, lifting))
  ultimately have
    is-symmetry (elect-r  $\circ$  fun $\mathcal{E}$  (rule- $\mathcal{K}$  strong-unanimity))
    (action-induced-equivariance (carrier neutralityG) (elections- $\mathcal{K}$  strong-unanimity)
      ( $\varphi\text{-neutr (valid-elections} \cap \text{Collect strong-unanimity}_C)$ ) (set-action  $\psi\text{-neutr}_c$ ))
  using strong-unanimity-neutral
    equivar-under-subset[of
      elect-r  $\circ$  fun $\mathcal{E}$  (rule- $\mathcal{K}$  strong-unanimity)
      valid-elections  $\cap$  Collect strong-unanimityC
      {( $\varphi\text{-neutr (valid-elections} \cap \text{Collect strong-unanimity}_C)$ )  $g$ , set-action
         $\psi\text{-neutr}_c g$ ) |  $g$ .
         $g \in \text{carrier neutrality}_G$ } 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
       $\varphi\text{-neutr (elections-}\mathcal{K} \text{ strong-unanimity)}$ 
       $\varphi\text{-neutr (valid-elections} \cap \text{Collect strong-unanimity}_C)$ 
      elect-r  $\circ$  fun $\mathcal{E}$  (rule- $\mathcal{K}$  strong-unanimity) set-action  $\psi\text{-neutr}_c$ ]
  by (metis (no-types))
qed

```

**lemma** *strong-unanimity-closed-under-neutrality: closed-restricted-rel*  
*(neutrality<sub>R</sub> valid-elections) valid-elections (elections-K strong-unanimity)*

**proof** *(unfold closed-restricted-rel.simps restricted-rel.simps neutrality<sub>R</sub>.simps*  
*action-induced-rel.simps elections-K.simps, safe)*

**fix**  
*A :: 'a set and*  
*V :: 'b set and*  
*p :: ('a, 'b) Profile and*  
*A' :: 'a set and*  
*V' :: 'b set and*  
*p' :: ('a, 'b) Profile and*  
*π :: 'a ⇒ 'a and*  
*a :: 'a*

**assume**  
*prof: (A, V, p) ∈ valid-elections and*  
*cons: (A, V, p) ∈ K<sub>E</sub> strong-unanimity a and*  
*bij: π ∈ carrier neutrality<sub>G</sub> and*  
*img: φ-neutr valid-elections π (A, V, p) = (A', V', p')*

**hence** *fin: (A, V, p) ∈ finite-elections*  
*unfolding K<sub>E</sub>.simps finite-elections-def*  
*by simp*

**hence** *valid': (A', V', p') ∈ valid-elections*  
*using bij img φ-neutr-act.group-action-axioms group-action.element-image prof*  
*unfolding finite-elections-def*  
*by (metis (mono-tags, lifting))*

**moreover** *have V' = V ∧ A' = π ' A*  
*using img fin alternatives-rewrite.elims fstI prof sndI*  
*unfolding extensional-continuation.simps φ-neutr.simps alternatives-E.simps*  
*voters-E.simps*  
*by (metis (no-types, lifting))*

**ultimately** *have prof': finite-profile V' A' p'*  
*using fin bij CollectD finite-imageI fst-eqD snd-eqD*  
*unfolding finite-elections-def valid-elections-def alternatives-E.simps*  
*voters-E.simps profile-E.simps*  
*by (metis (no-types, lifting))*

**let** *?domain = valid-elections ∩ Collect strong-unanimity<sub>C</sub>*  
**have** *((A, V, p), (A', V', p')) ∈ neutrality<sub>R</sub> valid-elections*  
*using bij img fin valid'*  
*unfolding neutrality<sub>R</sub>.simps action-induced-rel.simps*  
*finite-elections-def valid-elections-def*  
*by blast*

**moreover** *have unanimous: (A, V, p) ∈ ?domain*  
*using cons fin*  
*unfolding K<sub>E</sub>.simps strong-unanimity-def valid-elections-def*  
*by simp*

**ultimately** *have unanimous': (A', V', p') ∈ ?domain*  
*using strong-unanimity<sub>C</sub>-neutral*  
*by force*

**have** *rewrite*:  $\forall \pi \in \text{carrier neutrality}_{\mathcal{G}}$ .  
 $\varphi\text{-neutr } ?\text{domain } \pi (A, V, p) \in ?\text{domain} \longrightarrow$   
 $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (\varphi\text{-neutr } ?\text{domain } \pi (A, V, p))$   
 $=$   
 $\text{set-action } \psi\text{-neutr}_c \pi ((\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (A, V,$   
 $p))$   
**using** *strong-unanimity-neutral unanimous*  
 $\text{rewrite-equivariance[of}$   
 $\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})$   
 $\text{carrier neutrality}_{\mathcal{G}} ?\text{domain}$   
 $\varphi\text{-neutr } ?\text{domain set-action } \psi\text{-neutr}_c]$   
**unfolding** *SCF-properties.consensus-rule-neutrality.simps*  
**by** *blast*  
**have** *img'*:  $\varphi\text{-neutr } ?\text{domain } \pi (A, V, p) = (A', V', p')$   
**using** *img unanimous*  
**by** *simp*  
**hence**  $\text{elect} (\text{rule-}\mathcal{K} \text{ strong-unanimity}) V' A' p' =$   
 $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (\varphi\text{-neutr } ?\text{domain } \pi (A, V, p))$   
**by** *simp*  
**also have**  $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (\varphi\text{-neutr } ?\text{domain } \pi (A, V,$   
 $p)) =$   
 $\text{set-action } \psi\text{-neutr}_c \pi$   
 $((\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (A, V, p))$   
**using** *bij img' unanimous' rewrite*  
**by** *metis*  
**also have**  $(\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ strong-unanimity})) (A, V, p) = \{a\}$   
**using** *cons*  
**unfolding** *K<sub>E</sub>.simps*  
**by** *simp*  
**finally have**  $\text{elect} (\text{rule-}\mathcal{K} \text{ strong-unanimity}) V' A' p' = \{\psi\text{-neutr}_c \pi a\}$   
**by** *simp*  
**hence**  $(A', V', p') \in \mathcal{K}_{\mathcal{E}} \text{ strong-unanimity } (\psi\text{-neutr}_c \pi a)$   
**unfolding** *K<sub>E</sub>.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-K.simps*  
**using** *cons*  
**by** *blast*  
**moreover have**  $\exists \pi \in \text{carrier neutrality}_{\mathcal{G}}. \varphi\text{-neutr valid-elections } \pi (A, V, p)$   
 $= (A', V', p')$   
**using** *img bij*  
**unfolding** *neutrality<sub>G</sub>-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 ' (K<sub>E</sub> K w))

**fun** (in *result*) *R<sub>W</sub>* :: ('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**  
*R<sub>W</sub>* d K V A p = arg-min-set (score d K (A, V, p)) (limit-set A UNIV)

**fun** (in *result*) *distance-R* :: ('a, 'v) *Election Distance*  $\Rightarrow$  ('a, 'v, 'r *Result*) *Consensus-Class*  $\Rightarrow$   
('a, 'v, 'r *Result*) *Electoral-Module* **where**  
*distance-R* d K V A p = (*R<sub>W</sub>* d K V A p, (limit-set A UNIV) - *R<sub>W</sub>* 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** *all-profiles* :: ('a, 'v) set  $\Rightarrow$  ('a, 'v) Profile set **where**  
*all-profiles* V A =  
 (if (infinite A  $\vee$  infinite V)  
 then {} else {p. p ' V  $\subseteq$  (pl- $\alpha$  ' permutations-of-set A)})

**fun** *K<sub>E</sub>-std* :: ('a, 'v, 'r Result) Consensus-Class  $\Rightarrow$  'r  $\Rightarrow$  'a set  $\Rightarrow$  'v set  
 $\Rightarrow$  ('a, 'v) Election set **where**  
*K<sub>E</sub>-std* K w A V =  
 ( $\lambda$  p. (A, V, p)) ' (Set.filter  
 ( $\lambda$  p. (consensus-K K) (A, V, p)  $\wedge$  elect (rule-K K) V A p =  
 {w})  
 (all-profiles V A))

Returns those consensus elections on a given alternative and voter set from a given consensus that are mapped to the given unique winner by a given consensus rule.

**fun** *score-std* :: ('a, 'v) Election Distance  $\Rightarrow$  ('a, 'v, 'r Result) Consensus-Class  $\Rightarrow$   
 ('a, 'v) Election  $\Rightarrow$  'r  $\Rightarrow$  ereal **where**  
*score-std* d K E w =  
 (if *K<sub>E</sub>-std* K w (alternatives-E E) (voters-E E) = {}  
 then  $\infty$  else Min (d E ' (*K<sub>E</sub>-std* K w (alternatives-E E) (voters-E E))))

**fun** (in result) *R<sub>W</sub>-std* :: ('a, 'v) Election Distance  $\Rightarrow$  ('a, 'v, 'r Result) Consensus-Class  $\Rightarrow$   
 'v set  $\Rightarrow$  'a set  $\Rightarrow$  ('a, 'v) Profile  $\Rightarrow$  'r set **where**  
*R<sub>W</sub>-std* d K V A p = arg-min-set (score-std d K (A, V, p)) (limit-set A UNIV)

**fun** (in result) *distance-R-std* :: ('a, 'v) Election Distance  $\Rightarrow$   
 ('a, 'v, 'r Result) Consensus-Class  $\Rightarrow$  ('a, 'v, 'r Result) Electoral-Module  
**where**  
*distance-R-std* d K V A p = (*R<sub>W</sub>-std* d K V A p, (limit-set A UNIV) - *R<sub>W</sub>-std*  
 d K V A p, {})

### 5.4.3 Auxiliary Lemmas

**lemma** *fin-K<sub>E</sub>*:  
**fixes** C :: ('a, 'v, 'r Result) Consensus-Class  
**shows** elections-K C  $\subseteq$  finite-elections  
**proof**  
**fix** E :: ('a, 'v) Election  
**assume** E  $\in$  elections-K C  
**hence** finite-election E  
**unfolding** *K<sub>E</sub>.sims*  
**by** force

```

thus  $E \in \text{finite-elections}$ 
  unfolding finite-elections-def
  by simp
qed

```

```

lemma univ- $\mathcal{K}_E$ :
  fixes  $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
  shows  $\text{elections-}\mathcal{K} \ C \subseteq \text{UNIV}$ 
  by simp

```

```

lemma list-cons-presv-finiteness:

```

```

  fixes
     $A :: 'a \text{ set}$  and
     $S :: 'a \text{ list set}$ 
  assumes
    fin-A:  $\text{finite } A$  and
    fin-B:  $\text{finite } S$ 
  shows  $\text{finite } \{a \# l \mid a \ l. a \in A \wedge l \in S\}$ 
proof –
  let  $?P = \lambda A. \text{finite } \{a \# l \mid a \ l. a \in A \wedge l \in S\}$ 
  have  $\bigwedge a \ A'. \text{finite } A' \implies a \notin A' \implies ?P \ A' \implies ?P \ (\text{insert } a \ A')$ 
  proof –
    fix
       $a :: 'a$  and
       $A' :: 'a \text{ set}$ 
    assume
      fin:  $\text{finite } A'$  and
      not-in:  $a \notin A'$  and
      fin-set:  $\text{finite } \{a \# l \mid a \ l. a \in A' \wedge l \in S\}$ 
    have  $\{a' \# l \mid a' \ l. a' \in \text{insert } a \ A' \wedge l \in S\}$ 
       $= \{a \# l \mid a \ l. a \in A' \wedge l \in S\} \cup \{a \# l \mid l. l \in S\}$ 
    by auto
    moreover have  $\text{finite } \{a \# l \mid l. l \in S\}$ 
    using fin-B
    by simp
    ultimately have  $\text{finite } \{a' \# l \mid a' \ l. a' \in \text{insert } a \ A' \wedge l \in S\}$ 
    using fin-set
    by simp
    thus  $?P \ (\text{insert } a \ A')$ 
    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  $l :: 'a \text{ set list}$ 
assumes  $\forall i :: \text{nat}. i < \text{length } l \longrightarrow \text{finite } (l!i)$ 
shows  $\text{finite } (\text{listset } l)$ 
using assms
proof (induct  $l$ )
  case Nil
  show  $\text{finite } (\text{listset } [])$ 
    by simp
next
  case (Cons  $a \ l$ )
  fix
     $a :: 'a \text{ set}$  and
     $l :: 'a \text{ set list}$ 
  assume
    elems-fin-then-set-fin:  $\forall i :: \text{nat} < \text{length } l. \text{finite } (l!i) \implies \text{finite } (\text{listset } l)$  and
    fin-all-elems:  $\forall i :: \text{nat} < \text{length } (a\#l). \text{finite } ((a\#l)!i)$ 
  hence  $\text{finite } a$ 
    by auto
  moreover from fin-all-elems
  have  $\forall i < \text{length } l. \text{finite } (l!i)$ 
    by auto
  hence  $\text{finite } (\text{listset } l)$ 
    using elems-fin-then-set-fin
    by simp
  ultimately have  $\text{finite } \{a'\#l' \mid a' \ l'. a' \in a \wedge l' \in (\text{listset } l)\}$ 
    using list-cons-presv-finiteness
    by auto
  thus  $\text{finite } (\text{listset } (a\#l))$ 
    by (simp add: set-Cons-def)
qed

```

**lemma** *ls-entries-empty-imp-ls-set-empty*:

```

fixes  $l :: 'a \text{ set list}$ 
assumes
   $0 < \text{length } l$  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
     $a :: 'a \text{ set}$  and
     $l :: 'a \text{ set list}$  and
     $l' :: 'a \text{ list}$ 
  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  $l :: 'a \text{ set list}$ 
  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)
  fix
     $a :: 'a \text{ set}$  and
     $l :: 'a \text{ set list}$  and
     $l' :: 'a \text{ list}$ 
  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 all-ls-elems-in-ls-set:
  fixes  $l :: 'a \text{ set list}$ 
  shows  $\forall l' i :: \text{nat}. l' \in \text{listset } l \wedge i < \text{length } l' \longrightarrow l!i \in l!i$ 
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)
fix
  a :: 'a set and
  l :: 'a set list and
  l' :: 'a list and
  i :: nat
assume elems-in-set-then-elems-pos:
   $\forall l' i :: \text{nat}. l' \in \text{listset } l \wedge i < \text{length } l' \longrightarrow l'!i \in l!i$  and
  l-prime-in-set-a-l:  $l' \in \text{listset } (a \# l)$  and
  i-lt-len-l-prime:  $i < \text{length } l'$ 
have  $l' \in \text{set-Cons } a (\text{listset } l)$ 
  using l-prime-in-set-a-l
  by simp
hence  $l' \in \{m. \exists b m'. m = b \# m' \wedge b \in a \wedge m' \in (\text{listset } l)\}$ 
  unfolding set-Cons-def
  by simp
hence  $\exists b m. l' = b \# m \wedge b \in a \wedge m \in (\text{listset } l)$ 
  by simp
thus  $l'!i \in (a \# l)!i$ 
  using elems-in-set-then-elems-pos i-lt-len-l-prime nth-Cons-Suc
  Suc-less-eq gr0-conv-Suc length-Cons nth-non-equal-first-eq
  by metis
qed

```

lemma *fin-all-profs*:

```

fixes
  A :: 'a set and
  V :: 'v set and
  x :: 'a Preference-Relation
assumes
  fin-A: finite A and
  fin-V: finite V
shows finite (all-profiles V A  $\cap \{p. \forall v. v \notin V \longrightarrow p v = x\}$ )
proof (cases A = {})
let ?profs = all-profiles V A  $\cap \{p. \forall v. v \notin V \longrightarrow p v = x\}$ 
case True
hence permutations-of-set A = {}
  unfolding permutations-of-set-def
  by fastforce
hence pl- $\alpha$  ' permutations-of-set A = {}
  unfolding pl- $\alpha$ -def
  by simp
hence  $\forall p \in \text{all-profiles } V A. \forall v. v \in V \longrightarrow p v = \{ \}$ 
  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 ?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 = (all-profiles V A  $\cap \{p. \forall v. v \notin V \longrightarrow p v = x\}$ )
  case False
  from fin-V obtain ord where linear-order-on V ord
    using finite-list lin-ord-equiv lin-order-equiv-list-of-alts
    by metis
  then obtain list-V where
    len: length list-V = card V and
    pl: ord = pl- $\alpha$  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 =  $\lambda p::('a, 'v)$  Profile. map p list-V
  have  $\forall p \in \text{all-profiles } V A. \forall v \in V. p v \in (\text{pl-}\alpha \text{ ' } \text{permutations-of-set } A)$ 
    by (simp add: image-subset-iff)
  hence  $\forall p \in \text{all-profiles } V A. (\forall v \in V. \text{linear-order-on } A (p v))$ 
    using pl- $\alpha$ -lin-order fin-A False
    by metis
  moreover have  $\forall p \in ?profs. \forall i < \text{length } (?map p). (?map p)!i = p (\text{list-V}!i)$ 
    by simp
  moreover have  $\forall i < \text{length list-V}. \text{list-V}!i \in V$ 
    using perm nth-mem permutations-of-setD(1)
    by metis
  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:  $?map \text{ ' } ?profs \subseteq \{xs. \text{length } xs = \text{card } V \wedge$ 
     $(\forall i < \text{length } xs. \text{linear-order-on } A (xs!i))\}$ 
    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 finite {xs. length xs = card V ∧ (∀ i < length xs. linear-order-on
A (xs!i))}
  proof -
    have finite {r. linear-order-on A r}
    using fin-A
    unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
    by simp
    hence finSupset: ∀ n. finite {xs. length xs = n ∧ set xs ⊆ {r. linear-order-on
A r}}
    using Collect-mono finite-lists-length-eq rev-finite-subset
    by (metis (no-types, lifting))
    have ∀ l ∈ {xs. length xs = card V ∧
      (∀ i < length xs. linear-order-on A (xs!i))}.
      set l ⊆ {r. linear-order-on A r}
    using in-set-conv-nth mem-Collect-eq subsetI
    by (metis (no-types, lifting))
    hence {xs. length xs = card V ∧
      (∀ i < length xs. linear-order-on A (xs!i))}
      ⊆ {xs. length xs = card V ∧ set xs ⊆ {r. linear-order-on A r}}
    by blast
    thus ?thesis
    using finSupset rev-finite-subset
    by blast
  qed
  moreover have ∀ f X Y. inj-on f X ∧ finite Y ∧ f ` X ⊆ Y ⟶ finite X
    using finite-imageD finite-subset
    by metis
  ultimately show finite ?profs
    using subset
    by blast
qed

lemma profile-permutation-set:
  fixes
    A :: 'a set and
    V :: 'v set
  shows all-profiles V A =
    {p' :: ('a, 'v) Profile. finite-profile V A p'}
proof (cases finite A ∧ finite V ∧ A ≠ {})
  case True
  assume finite A ∧ finite V ∧ A ≠ {}
  hence
    fin-A: finite A and
    fin-V: finite V and
    non-empty: A ≠ {}
  by safe
  show all-profiles V A = {p'. finite-profile V A p'}

```



```

proof
  show all-profiles  $V \ A \subseteq \{p'. \text{finite-profile } V \ A \ p'\}$ 
proof (standard, clarify)
  fix  $p' :: 'v \Rightarrow 'a \text{ Preference-Relation}$ 
  assume subset:  $p' \in \text{all-profiles } V \ A$ 
  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\text{-lin-order non-empty}
    by metis
  thus finite-profile  $V \ A \ p'$ 
    unfolding profile-def
    using fin-A fin-V
    by blast
qed
next
  show  $\{p'. \text{finite-profile } V \ A \ p'\} \subseteq \text{all-profiles } V \ A$ 
proof (standard, clarify)
  fix  $p' :: ('a, 'v) \text{ Profile}$ 
  assume prof: profile  $V \ A \ p'$ 
  have  $p' \in \{p. p \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\}$ 
    using fin-A lin-order-pl-}\alpha \text{ prof}
    unfolding profile-def
    by blast
  thus  $p' \in \text{all-profiles } V \ A$ 
    using fin-A fin-V
    unfolding all-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  $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \implies \text{permutations-of-set } A = \{\emptyset\}$ 
    unfolding permutations-of-set-def
    by fastforce
  hence pl-empty:  $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \implies \text{pl-}\alpha \text{ ' permutations-of-set } A$ 
     $= \{\{\}\}$ 
    unfolding pl-}\alpha\text{-def}
    by simp
  hence  $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \implies$ 
     $\forall \pi \in \{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\}. \forall v \in V. \pi \ v = \{\}$ 
    by fastforce
  hence  $\text{finite } A \wedge \text{finite } V \wedge A = \{\} \implies$ 
     $\{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\} = \{\pi. \forall v \in V. \pi \ v = \{\}\}$ 
    using image-subset-iff singletonD singletonI pl-empty
    by fastforce
  moreover have  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$ 

```

$\implies \text{all-profiles } V A = \{\pi. \pi \text{ ' } V \subseteq (\text{pl-}\alpha \text{ ' permutations-of-set } A)\}$   
 by *simp*  
**ultimately have** *all-prof-eq*:  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$   
 $\implies \text{all-profiles } V A = \{\pi. \forall v \in V. \pi v = \{\}\}$   
 by *simp*  
**have**  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$   
 $\implies \forall p' \in \{p'. \text{finite-profile } V A p' \wedge (\forall v'. v' \notin V \longrightarrow p' v' = \{\})\}.$   
 $(\forall v \in V. \text{linear-order-on } \{\} (p' v))$   
 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**  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$   
 $\implies \forall p' \in \{p'. \text{finite-profile } V A p' \wedge (\forall v'. v' \notin V \longrightarrow p' v' = \{\})\}.$   
 $\forall v. p' v = \{\}$   
 by *blast*  
**hence**  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$   
 $\implies \{p'. \text{finite-profile } V A p'\} = \{p'. \forall v \in V. p' v = \{\}\}$   
 using *lin-ord-not-empty linear-order-on-empty*  
 unfolding *profile-def*  
 by (*metis* (*no-types*, *opaque-lifting*))  
**hence**  $\text{finite } A \wedge \text{finite } V \wedge A = \{\}$   
 $\implies \text{all-profiles } V A = \{p'. \text{finite-profile } V A p'\}$   
 using *all-prof-eq*  
 by *simp*  
**moreover have**  $\text{infinite } A \vee \text{infinite } V \implies \text{all-profiles } V A = \{\}$   
 by *simp*  
**moreover have**  $\text{infinite } A \vee \text{infinite } V \implies$   
 $\{p'. \text{finite-profile } V A p' \wedge (\forall v'. v' \notin V \longrightarrow p' v' = \{\})\} = \{\}$   
 by *auto*  
**moreover have**  $\text{infinite } A \vee \text{infinite } V \vee A = \{\}$   
 using *not-fin-empty*  
 by *simp*  
**ultimately show**  $\text{all-profiles } V A = \{p'. \text{finite-profile } V A p'\}$   
 by *blast*  
 qed

#### 5.4.4 Soundness

**lemma** (*in result*) *R-sound*:

**fixes**

$K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$  **and**

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

**shows** *electoral-module* (*distance-R d K*)

**proof** (*unfold electoral-module.simps, safe*)

**fix**

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

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

```

  p :: ('a, 'v) Profile
have  $\mathcal{R}_W d K V A p \subseteq (\text{limit-set } A \text{ UNIV})$ 
  using  $\mathcal{R}_W.\text{simps arg-min-subset}$ 
  by metis
hence set-equals-partition ( $\text{limit-set } A \text{ UNIV}$ ) ( $\text{distance-}\mathcal{R} d K V A p$ )
  by auto
moreover have disjoint3 ( $\text{distance-}\mathcal{R} d K V A p$ )
  by simp
ultimately show well-formed A ( $\text{distance-}\mathcal{R} d K V A p$ )
  using result-axioms
  unfolding result-def
  by simp
qed

```

### 5.4.5 Inference Rules

```

lemma is-arg-min-equal:
  fixes
    f :: 'a  $\Rightarrow$  'b::ord and
    g :: 'a  $\Rightarrow$  'b and
    S :: 'a set and
    x :: 'a
  assumes  $\forall x \in S. f x = g x$ 
  shows is-arg-min f ( $\lambda s. s \in S$ )  $x = \text{is-arg-min } g (\lambda s. s \in S) x$ 
proof (unfold is-arg-min-def, cases  $x \in S$ )
  case False
  thus  $(x \in S \wedge (\nexists y. y \in S \wedge f y < f x)) = (x \in S \wedge (\nexists y. y \in S \wedge g y < g x))$ 
    by simp
next
  case x-in-S: True
  thus  $(x \in S \wedge (\nexists y. y \in S \wedge f y < f x)) = (x \in S \wedge (\nexists y. y \in S \wedge g y < g x))$ 
  proof (cases  $\exists y. (\lambda s. s \in S) y \wedge f y < f x$ )
    case y: True
    then obtain  $y :: 'a$  where
       $(\lambda s. s \in S) y \wedge f y < f x$ 
    by metis
    hence  $(\lambda s. s \in S) y \wedge g y < g x$ 
    using x-in-S assms
    by metis
    thus ?thesis
    using y
    by metis
  next
    case not-y: False
    have  $\neg (\exists y. (\lambda s. s \in S) y \wedge g y < g x)$ 
    proof (safe)
      fix  $y :: 'a$ 
      assume
        y-in-S:  $y \in S$  and

```

```

    g-y-lt-g-x:  $g\ y < g\ x$ 
  have f-eq-g-for-elems-in-S:  $\forall\ a.\ a \in S \longrightarrow f\ a = g\ a$ 
    using assms
    by simp
  hence  $g\ x = f\ x$ 
    using x-in-S
    by presburger
  thus False
    using f-eq-g-for-elems-in-S g-y-lt-g-x not-y y-in-S
    by (metis (no-types))
qed
thus ?thesis
  using x-in-S not-y
  by simp
qed
qed

```

**lemma** (*in result*) *standard-distance-imp-equal-score*:

```

  fixes
    d :: ('a, 'v) Election Distance and
    K :: ('a, 'v, 'r Result) Consensus-Class and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    w :: 'r
  assumes
    irr-non-V: voters-determine-distance d and
    std: standard d
  shows score d K (A, V, p) w = score-std d K (A, V, p) w
proof -
  have profile-perm-set:
    all-profiles V A =
      {p' :: ('a, 'v) Profile. finite-profile V A p'}
    using profile-permutation-set
    by metis
  hence eq-intersect:  $\mathcal{K}_{\mathcal{E}}\text{-std}\ K\ w\ A\ V =$ 
     $\mathcal{K}_{\mathcal{E}}\ K\ w \cap \text{Pair}\ A\ \text{'Pair}\ V\ \text{'}\{p' :: ('a, 'v)\ \text{Profile. finite-profile}\ V\ A\ p'\}$ 
    by force
  have inf-eq-inf-for-std-cons:
     $\text{Inf}\ (d\ (A, V, p)\ \text{'}\ (\mathcal{K}_{\mathcal{E}}\ K\ w)) =$ 
     $\text{Inf}\ (d\ (A, V, p)\ \text{'}\ (\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ 
     $\text{Pair}\ A\ \text{'Pair}\ V\ \text{'}\{p' :: ('a, 'v)\ \text{Profile. finite-profile}\ V\ A\ p'\}))$ 
proof -
  have  $(\mathcal{K}_{\mathcal{E}}\ K\ w \cap \text{Pair}\ A\ \text{'Pair}\ V\ \text{'}\{p' :: ('a, 'v)\ \text{Profile. finite-profile}\ V\ A\ p'\})$ 
     $\subseteq (\mathcal{K}_{\mathcal{E}}\ K\ w)$ 
    by simp
  hence  $\text{Inf}\ (d\ (A, V, p)\ \text{'}\ (\mathcal{K}_{\mathcal{E}}\ K\ w)) \leq$ 
     $\text{Inf}\ (d\ (A, V, p)\ \text{'}\ (\mathcal{K}_{\mathcal{E}}\ K\ w \cap$ 
     $\text{Pair}\ A\ \text{'Pair}\ V\ \text{'}\{p' :: ('a, 'v)\ \text{Profile. finite-profile}\ V\ A\ p'\}))$ 

```

```

    using INF-superset-mono dual-order.refl
    by metis
  moreover have  $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w)) \geq$ 
     $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w \cap$ 
       $\text{Pair } A \ ' \text{Pair } V \ ' \{p' :: ('a, 'v) \text{Profile. finite-profile } V \ A \ p'\}))$ 
  proof (rule INF-greatest)
    let ?inf =  $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\}))$ 
    let ?compl =  $(\mathcal{K}_\mathcal{E} \ K \ w) - (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})$ 
    fix i :: ('a, 'v) Election
    assume el:  $i \in \mathcal{K}_\mathcal{E} \ K \ w$ 
    have in-intersect:  $i \in (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})$ 
  p'})
     $\implies ?inf \leq d \ (A, V, p) \ i$ 
    using Complete-Lattices.complete-lattice-class.INF-lower
    by metis
  have  $i \in ?compl \implies (V \neq \text{fst } (\text{snd } i) \vee A \neq \text{fst } i \vee \neg \text{finite-profile } V \ A \ (\text{snd } (\text{snd } i)))$ 
    by fastforce
  moreover have  $V \neq \text{fst } (\text{snd } i) \implies d \ (A, V, p) \ i = \infty$ 
    using std.prod.collapse
    unfolding standard-def
    by metis
  moreover have  $A \neq \text{fst } i \implies d \ (A, V, p) \ i = \infty$ 
    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
  ultimately have
     $i \in ?compl \implies \text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})) \leq d \ (A, V, p) \ i$ 
    using ereal-less-eq
    by metis
  thus  $\text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\})) \leq d \ (A, V, p) \ i$ 
    using in-intersect el
    by blast
qed
ultimately show
 $\text{Inf } (d \ (A, V, p) \ ' \mathcal{K}_\mathcal{E} \ K \ w) = \text{Inf } (d \ (A, V, p) \ ' (\mathcal{K}_\mathcal{E} \ K \ w \cap \text{Pair } A \ ' \text{Pair } V \ ' \{p'. \text{finite-profile } V \ A \ p'\}))$ 

```

$(\mathcal{K}_{\mathcal{E}} K w \cap \text{Pair } A \text{ 'Pair } V \text{ ' } \{p'. \text{finite-profile } V A p'\}))$   
 by *simp*  
**qed**  
**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**  $\text{score-std } d K (A, V, p) w = \infty$   
 using *True*  
 unfolding *Let-def*  
 by *simp*  
**finally show** *?thesis*  
 by *simp*  
**next**  
**case** *False*  
**hence** *fin*:  $\text{finite } A \wedge \text{finite } V$   
 using *eq-intersect*  
 by *blast*  
**have**  $\text{finite } (d (A, V, p) \text{ ' } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V))$   
**proof** –  
**have**  $\mathcal{K}_{\mathcal{E}}\text{-std } K w A V = (\mathcal{K}_{\mathcal{E}} K w) \cap \{(A, V, p') \mid p'. \text{finite-profile } V A p'\}$   
 using *eq-intersect*  
 by *blast*  
**hence** *subset*:  $d (A, V, p) \text{ ' } (\mathcal{K}_{\mathcal{E}}\text{-std } K w A V) \subseteq d (A, V, p) \text{ ' } \{(A, V, p') \mid p'. \text{finite-profile } V A p'\}$   
 by *blast*  
**let**  $?finite\text{-prof} = \lambda p' v. (\text{if } (v \in V) \text{ then } p' v \text{ else } \{\})$   
**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  $\forall (A', V', p') \in \{(A', V', p'). A' = A \wedge V' = V \wedge \text{finite-profile } V A p'\}.$

$$d(A, V, p)(A', V', p') \in$$

$$d(A, V, p) \text{ ' } \{(A, V, p') \mid p'. \text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$$

by *fastforce*

hence *subset-2*:  $d(A, V, p) \text{ ' } \{(A, V, p') \mid p'. \text{finite-profile } V A p'\}$

$$\subseteq d(A, V, p) \text{ ' } \{(A, V, p') \mid p'. \text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$$

by *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  $\{(A, V, p') \mid p'. \text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$

$$\subseteq \{(A, V, p') \mid p'. p' \in \{p'. (\forall v \in V. \text{linear-order-on } A(p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}\}$$

by *blast*

moreover have *finite*  $\{(A, V, p') \mid p'. p' \in \{p'. (\forall v \in V. \text{linear-order-on } A(p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}\}$

proof –

have  $\{p'. (\forall v \in V. \text{linear-order-on } A(p' v)) \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$

$$\subseteq \text{all-profiles } V A \cap \{p. \forall v. v \notin V \longrightarrow p v = \{\}\}$$

using *lin-order-pl-α fin*

by *fastforce*

moreover have *finite*  $(\text{all-profiles } V A \cap \{p. \forall v. v \notin V \longrightarrow p v = \{\}\})$

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*

thus *?thesis*

by *simp*

qed

ultimately have *finite*  $\{(A, V, p') \mid p'. \text{finite-profile } V A p' \wedge (\forall v. v \notin V \longrightarrow p' v = \{\})\}$

```

    using rev-finite-subset
  by simp
hence finite (d (A, V, p) ‘ {(A, V, p') | p'. finite-profile V A p'
               ∧ (∀ v. v ∉ V ⟶ p' v = {})})
  by simp
hence finite (d (A, V, p) ‘ {(A, V, p') | p'. finite-profile V A p'})
  using subset-2 rev-finite-subset
  by simp
thus ?thesis
  using subset rev-finite-subset
  by blast
qed
moreover have d (A, V, p) ‘ (Kℰ-std K w A V) ≠ {}
  using False
  by simp
ultimately have Inf (d (A, V, p) ‘ (Kℰ-std K w A V)) = Min (d (A, V, p)
‘ (Kℰ-std K w A V))
  using Min-Inf False
  by metis
also have ... = score-std d K (A, V, p) w
  using False
  by simp
also have Inf (d (A, V, p) ‘ (Kℰ-std K w A V)) =
  Inf (d (A, V, p) ‘ (Kℰ K w ∩
    Pair A ‘ Pair V ‘ {p'. finite-profile V A p'}))
  using eq-intersect
  by simp
ultimately show ?thesis
  by simp
qed
finally show score d K (A, V, p) w = score-std d K (A, V, p) w
  by simp
qed

```

**lemma** (in result) *anonymous-distance-and-consensus-imp-rule-anonymity*:

```

fixes
  d :: ('a, 'v) Election Distance and
  K :: ('a, 'v, 'r Result) Consensus-Class
assumes
  d-anon: distance-anonymity d and
  K-anon: consensus-rule-anonymity K
shows anonymity (distance-ℛ d K)
proof (unfold anonymity-def Let-def, safe)
  show electoral-module (distance-ℛ d K)
    using ℛ-sound
    by metis
next
fix
  A :: 'a set and

```



$A' :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $V' :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $q :: ('a, 'v) \text{ Profile}$  **and**  
 $\pi :: 'v \Rightarrow 'v$   
**assume**  
 $\text{fin-}A$ :  $\text{finite } A$  **and**  
 $\text{fin-}V$ :  $\text{finite } V$  **and**  
 $\text{profile-}p$ :  $\text{profile } V \ A \ p$  **and**  
 $\text{profile-}q$ :  $\text{profile } V' \ A' \ q$  **and**  
 $\text{bij}$ :  $\text{bij } \pi$  **and**  
 $\text{renamed}$ :  $\text{rename } \pi \ (A, V, p) = (A', V', q)$   
**have**  $A = A'$   
**using**  $\text{bij renamed}$   
**by**  $\text{simp}$   
**hence**  $\text{eq-univ}$ :  $\text{limit-set } A \ \text{UNIV} = \text{limit-set } A' \ \text{UNIV}$   
**by**  $\text{simp}$   
**hence**  $\mathcal{R}_W \ d \ K \ V \ A \ p = \mathcal{R}_W \ d \ K \ V' \ A' \ q$   
**proof** –  
**have**  $\text{dist-rename-inv}$ :  
 $\forall E :: ('a, 'v) \text{ Election. } d \ (A, V, p) \ E = d \ (A', V', q) \ (\text{rename } \pi \ E)$   
**using**  $d\text{-anon bij renamed surj-pair}$   
**unfolding**  $\text{distance-anonymity-def}$   
**by**  $\text{metis}$   
**hence**  $\forall S :: ('a, 'v) \text{ Election set.}$   
 $(d \ (A, V, p) \ 'S) \subseteq (d \ (A', V', q) \ '(\text{rename } \pi \ 'S))$   
**by**  $\text{blast}$   
**moreover have**  $\forall S :: ('a, 'v) \text{ Election set.}$   
 $((d \ (A', V', q) \ '(\text{rename } \pi \ 'S)) \subseteq (d \ (A, V, p) \ 'S))$   
**proof** ( $\text{clarify}$ )  
**fix**  
 $S :: ('a, 'v) \text{ Election set}$  **and**  
 $X :: 'a \text{ set}$  **and**  
 $X' :: 'a \text{ set}$  **and**  
 $Y :: 'v \text{ set}$  **and**  
 $Y' :: 'v \text{ set}$  **and**  
 $z :: ('a, 'v) \text{ Profile}$  **and**  
 $z' :: ('a, 'v) \text{ Profile}$   
**assume**  
 $(X', Y', z') = \text{rename } \pi \ (X, Y, z)$  **and**  
 $\text{el}: (X, Y, z) \in S$   
**hence**  $d \ (A', V', q) \ (X', Y', z') = d \ (A, V, p) \ (X, Y, z)$   
**using**  $\text{dist-rename-inv}$   
**by**  $\text{simp}$   
**thus**  $d \ (A', V', q) \ (X', Y', z') \in d \ (A, V, p) \ 'S$   
**using**  $\text{el}$   
**by**  $\text{simp}$   
**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**  
 $A :: 'a \text{ set}$  **and**  
 $A' :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $V' :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$   
**assume**  
 $\text{renamed}: (A', V', p') = \text{rename } \pi (A, V, p)$  **and**  
 $\text{consensus}: (A, V, p) \in \mathcal{K}_{\mathcal{E}} K w$   
**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*  
**hence** *fin-img*:  $\text{finite-profile } V' A' p'$   
**using** *renamed* *bij* *rename.simps* *fst-conv* *rename-finite*  
**by** *metis*  
**hence** *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* *bij* *cons*  
**unfolding** *consensus-rule-anonymity-def* *Let-def*  
**by** *simp*  
**hence** *elect*  $(\text{rule-}\mathcal{K} K) V' A' p' = \{w\}$   
**using** *cons*  
**by** *simp*  
**thus**  $(A', V', p') \in \mathcal{K}_{\mathcal{E}} K w$   
**using** *cons-img* *fin-img*  
**by** *simp*  
**qed**  
**moreover have**  $\forall w. (\mathcal{K}_{\mathcal{E}} K w) \subseteq \text{rename } \pi \text{ ' } (\mathcal{K}_{\mathcal{E}} K w)$   
**proof** (*clarify*)  
**fix**  
 $w :: 'r$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**assume**  $\text{consensus}: (A, V, p) \in \mathcal{K}_{\mathcal{E}} K w$   
**let**  $?inv = \text{rename } (\text{the-inv } \pi) (A, V, p)$   
**have** *inv-inv-id*:  $\text{the-inv } (\text{the-inv } \pi) = \pi$   
**using** *the-inv-f-f* *bij* *bij-betw-imp-inj-on* *bij-betw-imp-surj*  
 $\text{inj-on-the-inv-into}$  *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 bij 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 bij the-inv-f-f bij-betw-def image-inv-into-cancel
      surj-imp-inv-eq top-greatest
    by (metis (no-types, opaque-lifting))
  ultimately have preimg:  $\text{rename } \pi \text{ ?inv} = (A, V, p)$ 
    unfolding Let-def
    by simp
  moreover have  $\text{?inv} \in \mathcal{K}_\mathcal{E} K w$ 
  proof -
    have 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}
      using consensus
      by simp
    moreover have bij-inv:  $\text{bij } (\text{the-inv } \pi)$ 
      using bij bij-betw-the-inv-into
      by metis
    moreover have fin-preimg:  $\text{finite-profile } (\text{fst } (\text{snd ?inv})) (\text{fst ?inv}) (\text{snd } (\text{snd ?inv}))$ 
      using bij-inv rename.simps fst-conv rename-finite cons
      by fastforce
    ultimately have cons-preimg:
       $\text{consensus-}\mathcal{K} K \text{ ?inv} \wedge$ 
       $(\text{rule-}\mathcal{K} K V A p = \text{rule-}\mathcal{K} K (\text{fst } (\text{snd ?inv})) (\text{fst ?inv}) (\text{snd } (\text{snd ?inv})))$ 
      using K-anon renamed bij cons
      unfolding consensus-rule-anonymity-def Let-def
      by simp
    hence elect  $(\text{rule-}\mathcal{K} K) (\text{fst } (\text{snd ?inv})) (\text{fst ?inv}) (\text{snd } (\text{snd ?inv})) = \{w\}$ 
      using cons
      by simp
    thus ?thesis
      using cons-preimg fin-preimg
      by simp
  qed
  ultimately show  $(A, V, p) \in \text{rename } \pi \text{ ' } \mathcal{K}_\mathcal{E} K w$ 
    using 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 arg-min-set (score d K (A, V, p)) (limit-set A UNIV)
      = arg-min-set (score d K (A', V', q)) (limit-set A' UNIV)
      using eq-univ
      by presburger
thus  $\mathcal{R}_{\mathcal{W}} d K V A p = \mathcal{R}_{\mathcal{W}} d K V' A' q$ 
      by simp
qed
thus distance- $\mathcal{R}$  d K V A p = distance- $\mathcal{R}$  d K V' A' q
      using eq-univ
      by simp
qed
end

```

## 5.5 Votewise Distance Rationalization

```

theory Votewise-Distance-Rationalization
  imports Distance-Rationalization
           Votewise-Distance
begin

```

A votewise distance rationalization of a voting rule is its distance rationalization with a distance function that depends on the submitted votes in a simple and a transparent manner by using a distance on individual orders and combining the components with a norm on  $\mathbb{R}$  to  $\mathbb{N}$ .

### 5.5.1 Common Rationalizations

```

fun swap- $\mathcal{R}$  :: ('a, 'v::linorder, 'a Result) Consensus-Class  $\Rightarrow$ 
           ('a, 'v, 'a Result) Electoral-Module where
  swap- $\mathcal{R}$  K = SCF-result.distance- $\mathcal{R}$  (votewise-distance swap l-one) K

```

### 5.5.2 Theorems

```

lemma votewise-non-voters-irrelevant:
  fixes
    d :: 'a Vote Distance and
    N :: Norm
  shows voters-determine-distance (votewise-distance d N)
proof (unfold voters-determine-distance-def, clarify)
  fix
    A :: 'a set and
    V :: 'v::linorder set and
    p :: ('a, 'v) Profile and
    A' :: 'a set and
    V' :: 'v set and

```

```

    p' :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile
  assume coincide:  $\forall v \in V. p\ v = q\ v$ 
  have  $\forall i < \text{length } (\text{sorted-list-of-set } V). (\text{sorted-list-of-set } V)!i \in V$ 
    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 set and
    V :: 'v::linorder set and
    p :: ('a, 'v) Profile and
    A' :: 'a set and
    V' :: 'v set and
    p' :: ('a, 'v) Profile
  assume assms:  $V \neq V' \vee A \neq A'$ 
  let ?l =  $(\lambda\ l1\ l2. (\text{map2 } (\lambda\ q\ q'. \text{swap } (A, q)\ (A', q'))\ l1\ l2))$ 
  have  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \implies \forall q\ q'. \text{swap } (A, q)\ (A', q')$ 
  =  $\infty$ 
    by simp
  hence  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \implies$ 
     $\forall l1\ l2. (l1 \neq [] \wedge l2 \neq [] \longrightarrow (\forall i < \text{length } (?l\ l1\ l2). (?l\ l1\ l2)!i = \infty))$ 
    by simp
  moreover have  $V = V' \wedge V \neq \{\} \wedge \text{finite } V \implies (\text{to-list } V\ p) \neq [] \wedge (\text{to-list } V'\ p') \neq []$ 
    using card-eq-0-iff length-map list.size(3) to-list.simps
      sorted-list-of-set.length-sorted-key-list-of-set
    by metis
  moreover have  $\forall l. (\exists i < \text{length } l. ll i = \infty) \longrightarrow l\text{-one } l = \infty$ 
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 abs-ereal.simps(3) finite-lessThan lessThan-iff*  
 by *metis*  
 thus *l-one*  $l = \infty$   
 by *auto*  
 qed  
 ultimately have  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \implies$   
 $\text{l-one } (?l \text{ (to-list } V \text{ } p) \text{ (to-list } V' \text{ } p)) = \infty$   
 using *length-greater-0-conv map-is-Nil-conv zip-eq-Nil-iff*  
 by *metis*  
 hence  $A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V \implies$   
 $\text{votewise-distance swap l-one } (A, V, p) (A', V', p') = \infty$   
 by *simp*  
 moreover have  $V \neq V' \implies \text{votewise-distance swap l-one } (A, V, p) (A', V',$   
 $p') = \infty$   
 by *simp*  
 moreover have  $A \neq A' \wedge V = \{\} \implies \text{votewise-distance swap l-one } (A, V, p)$   
 $(A', V', p') = \infty$   
 by *simp*  
 moreover have *infinite*  $V \implies \text{votewise-distance swap l-one } (A, V, p) (A', V',$   
 $p') = \infty$   
 by *simp*  
 moreover have  $(A \neq A' \wedge V = V' \wedge V \neq \{\} \wedge \text{finite } V) \vee \text{infinite } V \vee (A \neq$   
 $A' \wedge V = \{\}) \vee V \neq V'$   
 using *assms*  
 by *blast*  
 ultimately show  $\text{votewise-distance swap l-one } (A, V, p) (A', V', p') = \infty$   
 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*:  $(\text{score}::('a, 'v::\text{linorder}) \text{ score-type}) (\text{votewise-distance}$   
 $\text{swap l-one}) =$   
 $\text{score-std } (\text{votewise-distance swap l-one})$   
 using *votewise-non-voters-irrelevant swap-standard*

```

      SCF-result.standard-distance-imp-equal-score
    by fast

lemma swap- $\mathcal{R}$ -code[code]: swap- $\mathcal{R}$  =
  (SCF-result.distance- $\mathcal{R}$ -std::('a, 'v::linorder) dist-rat-std-type))
  (votewise-distance swap l-one)

proof -
  from equal-score-swap
  have
     $\forall K E a. (score::('a, 'v::linorder) score-type))$ 
    (votewise-distance swap l-one)  $K E a =$ 
    score-std (votewise-distance swap l-one)  $K E a$ 
  by metis
  hence  $\forall K V A p. (SCF-result.\mathcal{R}_V::('a, 'v::linorder) dist-rat-type))$ 
    (votewise-distance swap l-one)  $K V A p =$ 
    SCF-result. $\mathcal{R}_V$ -std
    (votewise-distance swap l-one)  $K V A p$ 
  by (simp add: equal-score-swap)
  hence  $\forall K V A p. (SCF-result.distance-\mathcal{R}::('a, 'v::linorder) dist-type))$ 
    (votewise-distance swap l-one)  $K V A p$ 
    = SCF-result.distance- $\mathcal{R}$ -std
    (votewise-distance swap l-one)  $K V A p$ 
  by fastforce
  thus ?thesis
    unfolding swap- $\mathcal{R}$ .simps
  by blast
qed

end

```

## 5.6 Symmetry in Distance-Rationalizable Rules

```

theory Distance-Rationalization-Symmetry
  imports Distance-Rationalization
begin

```

### 5.6.1 Minimizer function

```

fun distance-infimum :: 'x Distance  $\Rightarrow$  'x set  $\Rightarrow$  'x  $\Rightarrow$  ereal where
  distance-infimum  $d X a = \text{Inf } (d a \text{ ' } X)$ 

fun closest-preimg-distance :: ('x  $\Rightarrow$  'y)  $\Rightarrow$  'x set  $\Rightarrow$  'x Distance  $\Rightarrow$  'x  $\Rightarrow$  'y  $\Rightarrow$  ereal
where
  closest-preimg-distance  $f \text{ domain}_f d x y = \text{distance-infimum } d (\text{preimg } f \text{ domain}_f y) x$ 

fun minimizer :: ('x  $\Rightarrow$  'y)  $\Rightarrow$  'x set  $\Rightarrow$  'x Distance  $\Rightarrow$  'y set  $\Rightarrow$  'x  $\Rightarrow$  'y set where

```

$\text{minimizer } f \text{ domain}_f d \ Y \ x = \text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f d \ x) \ Y$

## Auxiliary Lemmas

**lemma** *rewrite-arg-min-set:*

**fixes**

$f :: 'x \Rightarrow 'y::\text{linorder}$  **and**

$X :: 'x \text{ set}$

**shows**  $\text{arg-min-set } f \ X = \bigcup (\text{preimg } f \ X \ ' \ \{y \in (f \ ' \ X). \ \forall \ z \in f \ ' \ X. \ y \leq z\})$

**proof** (*safe*)

**fix**  $x :: 'x$

**assume**  $\text{arg-min}: x \in \text{arg-min-set } f \ X$

**hence**  $\text{is-arg-min } f \ (\lambda \ a. \ a \in X) \ x$

**by** *simp*

**hence**  $\forall \ x' \in X. \ f \ x' \geq f \ x$

**by** (*simp add: is-arg-min-linorder*)

**hence**  $\forall \ z \in f \ ' \ X. \ f \ x \leq z$

**by** *blast*

**moreover have**  $f \ x \in f \ ' \ X$

**using** *arg-min*

**by** (*simp add: is-arg-min-linorder*)

**ultimately have**  $f \ x \in \{y \in f \ ' \ X. \ \forall \ z \in f \ ' \ X. \ y \leq z\}$

**by** *blast*

**moreover have**  $x \in \text{preimg } f \ X \ (f \ x)$

**using** *arg-min*

**by** (*simp add: is-arg-min-linorder*)

**ultimately show**  $x \in \bigcup (\text{preimg } f \ X \ ' \ \{y \in (f \ ' \ X). \ \forall \ z \in f \ ' \ X. \ y \leq z\})$

**by** *blast*

**next**

**fix**

$x :: 'x$  **and**

$x' :: 'x$  **and**

$b :: 'x$

**assume**

*same-img*:  $x \in \text{preimg } f \ X \ (f \ x')$  **and**

*min*:  $\forall \ z \in f \ ' \ X. \ f \ x' \leq z$

**hence**  $f \ x = f \ x'$

**by** *simp*

**hence**  $\forall \ z \in f \ ' \ X. \ f \ x \leq z$

**using** *min*

**by** *simp*

**moreover have**  $x \in X$

**using** *same-img*

**by** *simp*

**ultimately show**  $x \in \text{arg-min-set } f \ X$

**by** (*simp add: is-arg-min-linorder*)

**qed**



## Equivariance

**lemma** *restr-induced-rel*:

**fixes**

$X :: 'x \text{ set}$  **and**

$Y :: 'y \text{ set}$  **and**

$Y' :: 'y \text{ set}$  **and**

$\varphi :: ('x, 'y) \text{ binary-fun}$

**assumes**  $Y' \subseteq Y$

**shows**  $\text{Restr } (\text{action-induced-rel } X \ Y \ \varphi) \ Y' = \text{action-induced-rel } X \ Y' \ \varphi$

**using** *assms*

**by** *auto*

**theorem** *group-act-invar-dist-and-equivar-f-imp-equivar-minimizer*:

**fixes**

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

$\text{domain}_f :: 'x \text{ set}$  **and**

$d :: 'x \text{ Distance}$  **and**

$\text{valid-img} :: 'x \Rightarrow 'y \text{ set}$  **and**

$X :: 'x \text{ set}$  **and**

$G :: 'z \text{ monoid}$  **and**

$\varphi :: ('z, 'x) \text{ binary-fun}$  **and**

$\psi :: ('z, 'y) \text{ binary-fun}$

**defines**  $\text{equivar-prop-set-valued} \equiv \text{action-induced-equivariance } (\text{carrier } G) \ X \ \varphi$   
 $(\text{set-action } \psi)$

**assumes**

*action- $\varphi$* :  $\text{group-action } G \ X \ \varphi$  **and**

*group-act-res*:  $\text{group-action } G \ \text{UNIV} \ \psi$  **and**

*dom-in-X*:  $\text{domain}_f \subseteq X$  **and**

*closed-domain*:

$\text{closed-restricted-rel } (\text{action-induced-rel } (\text{carrier } G) \ X \ \varphi) \ X \ \text{domain}_f$  **and**

*equivar-img*:  $\text{is-symmetry } \text{valid-img} \ \text{equivar-prop-set-valued}$  **and**

*invar-d*:  $\text{invariance}_{\mathcal{D}} \ d \ (\text{carrier } G) \ X \ \varphi$  **and**

*equivar-f*:  $\text{is-symmetry } f \ (\text{action-induced-equivariance } (\text{carrier } G) \ \text{domain}_f \ \varphi$

$\psi)$

**shows**  $\text{is-symmetry } (\lambda x. \text{minimizer } f \ \text{domain}_f \ d \ (\text{valid-img } x) \ x) \ \text{equivar-prop-set-valued}$

**proof** (*unfold action-induced-equivariance-def equivar-prop-set-valued-def is-symmetry.simps*  
*set-action.simps minimizer.simps, clarify*)

**fix**

$x :: 'x$  **and**

$g :: 'z$

**assume**

*group-elem*:  $g \in \text{carrier } G$  **and**

*x-in-X*:  $x \in X$  **and**

*img-X*:  $\varphi \ g \ x \in X$

**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*:  
 $\forall y. d\ x\ ' (preimg\ f\ domain_f\ y) = d\ ?x'\ ' (\varphi\ g\ ' (preimg\ f\ domain_f\ y))$   
 using *x-in-X group-elem invar-dist-image invar-d action-φ*  
 by *metis*  
 have  $\forall y. preimg\ f\ domain_f\ (\psi\ g\ y) = (\varphi\ g)\ ' (preimg\ f\ domain_f\ y)$   
 using *group-act-equivar-f-imp-equivar-preimg[of G X φ ψ domain\_f f g] assms*  
*group-elem*  
 by *blast*  
 hence  $\forall y. d\ ?x'\ ' preimg\ f\ domain_f\ (\psi\ g\ y) = d\ ?x'\ ' (\varphi\ g)\ ' (preimg\ f\ domain_f\ y)$   
 by *presburger*  
 hence  $\forall y. Inf\ (d\ ?x'\ ' preimg\ f\ domain_f\ (\psi\ g\ y)) = Inf\ (d\ x\ ' preimg\ f\ domain_f\ y)$   
 using *invar-dist-img*  
 by *metis*  
 hence  $\forall y. distance\_infimum\ d\ (preimg\ f\ domain_f\ (\psi\ g\ y))\ ?x'$   
 $\quad = distance\_infimum\ d\ (preimg\ f\ domain_f\ y)\ x$   
 by *simp*  
 hence  $\forall y. closest\_preimg\_distance\ f\ domain_f\ d\ ?x'\ (\psi\ g\ y) =$   
 $\quad closest\_preimg\_distance\ f\ domain_f\ d\ x\ y$   
 by *simp*  
 hence *comp*:  $closest\_preimg\_distance\ f\ domain_f\ d\ x$   
 $\quad = (closest\_preimg\_distance\ f\ domain_f\ d\ ?x') \circ (\psi\ g)$   
 by *auto*  
 hence  $\forall Y\ \alpha. preimg\ ?c'\ (\psi\ g\ ' Y)\ \alpha = \psi\ g\ ' preimg\ ?c\ Y\ \alpha$   
 using *preimg-comp*  
 by *auto*  
 hence  $\forall Y\ A. \{preimg\ ?c'\ (\psi\ g\ ' Y)\ \alpha \mid \alpha. \alpha \in A\} = \{\psi\ g\ ' preimg\ ?c\ Y\ \alpha \mid$   
 $\alpha. \alpha \in A\}$   
 by *simp*  
 moreover have  $\forall Y\ A. \{\psi\ g\ ' preimg\ ?c\ Y\ \alpha \mid \alpha. \alpha \in A\} = \{\psi\ g\ ' \beta \mid \beta. \beta \in$   
 $preimg\ ?c\ Y\ ' A\}$   
 by *blast*  
 moreover have  $\forall Y\ A. preimg\ ?c'\ (\psi\ g\ ' Y)\ ' A = \{preimg\ ?c'\ (\psi\ g\ ' Y)\ \alpha \mid$   
 $\alpha. \alpha \in A\}$   
 by *blast*  
 ultimately have  
 $\forall Y\ A. preimg\ ?c'\ (\psi\ g\ ' Y)\ ' A = \{\psi\ g\ ' \alpha \mid \alpha. \alpha \in preimg\ ?c\ Y\ ' A\}$   
 by *simp*  
 hence  $\forall Y\ A. \bigcup (preimg\ ?c'\ (\psi\ g\ ' Y)\ ' A) = \bigcup \{\psi\ g\ ' \alpha \mid \alpha. \alpha \in preimg\ ?c$   
 $Y\ ' A\}$   
 by *simp*  
 moreover have  $\forall Y\ A. \bigcup \{\psi\ g\ ' \alpha \mid \alpha. \alpha \in preimg\ ?c\ Y\ ' A\} = \psi\ g\ ' \bigcup$   
 $(preimg\ ?c\ Y\ ' A)$   
 by *blast*  
 ultimately have *eq-preimg-unions*:  
 $\forall Y\ A. \bigcup (preimg\ ?c'\ (\psi\ g\ ' Y)\ ' A) = \psi\ g\ ' \bigcup (preimg\ ?c\ Y\ ' A)$   
 by *simp*

**have**  $\forall Y. ?c' \text{ ' } \psi \text{ } g \text{ ' } Y = ?c \text{ ' } Y$   
**using** *comp*  
**unfolding** *image-comp*  
**by** *simp*  
**hence**  $\forall Y. \{\alpha \in ?c \text{ ' } Y. \forall \beta \in ?c \text{ ' } Y. \alpha \leq \beta\} =$   
 $\{\alpha \in ?c' \text{ ' } \psi \text{ } g \text{ ' } Y. \forall \beta \in ?c' \text{ ' } \psi \text{ } g \text{ ' } Y. \alpha \leq \beta\}$   
**by** *simp*  
**hence**  
 $\forall Y. \text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f \text{ } d \text{ } ?x') (\psi \text{ } g \text{ ' } Y)$   
 $= (\psi \text{ } g) \text{ ' } (\text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f \text{ } d \text{ } x) \text{ } Y)$   
**using** *rewrite-arg-min-set[of ?c'] rewrite-arg-min-set[of ?c] eq-preimg-unions*  
**by** *presburger*  
**moreover have** *valid-img*  $(\varphi \text{ } g \text{ } x) = \psi \text{ } g \text{ ' } \text{valid-img } x$   
**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**  
 $\text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f \text{ } d \text{ } (\varphi \text{ } g \text{ } x)) (\text{valid-img } (\varphi \text{ } g \text{ } x))$   
 $= \psi \text{ } g \text{ ' } \text{arg-min-set } (\text{closest-preimg-distance } f \text{ domain}_f \text{ } d \text{ } x) (\text{valid-img } x)$   
**by** *presburger*  
**qed**

## Invariance

**lemma** *closest-dist-invar-under-refl-rel-and-tot-invar-dist:*

**fixes**

$f :: 'x \Rightarrow 'y$  **and**  
 $\text{domain}_f :: 'x \text{ set}$  **and**  
 $d :: 'x \text{ Distance}$  **and**  
 $\text{rel} :: 'x \text{ rel}$

**assumes**

$r\text{-refl} :: \text{refl-on } \text{domain}_f \text{ } (\text{Restr } \text{rel } \text{domain}_f)$  **and**  
 $\text{tot-invar-d} :: \text{total-invariance}_{\mathcal{D}} \text{ } d \text{ } \text{rel}$

**shows** *is-symmetry*  $(\text{closest-preimg-distance } f \text{ domain}_f \text{ } d)$   $(\text{Invariance } \text{rel})$

**proof**  $(\text{unfold } \text{is-symmetry.simps, intro allI impI ext})$

**fix**

$a :: 'x$  **and**  
 $b :: 'x$  **and**  
 $y :: 'y$

**assume**  $\text{rel} : (a, b) \in \text{rel}$

**have**  $\forall c \in \text{domain}_f. (c, c) \in \text{rel}$

**using** *r-refl*

**unfolding** *refl-on-def*

**by** *simp*

**hence**  $\forall c \in \text{domain}_f. d \text{ } a \text{ } c = d \text{ } b \text{ } c$

**using** *rel tot-invar-d*

**unfolding** *rewrite-total-invariance<sub>D</sub>*

**by** *blast*

**thus**  $\text{closest-preimg-distance } f \text{ domain}_f \text{ } d \text{ } a \text{ } y = \text{closest-preimg-distance } f \text{ domain}_f$

*d b y*  
 by *simp*  
 qed

**lemma** *refl-rel-and-tot-invar-dist-imp-invar-minimizer:*

**fixes**  
*f* :: '*x*  $\Rightarrow$  '*y* **and**  
*domain<sub>f</sub>* :: '*x* set **and**  
*d* :: '*x* Distance **and**  
*rel* :: '*x* rel **and**  
*img* :: '*y* set  
**assumes**  
*r-refl*: *refl-on domain<sub>f</sub> (Restr rel domain<sub>f</sub>)* **and**  
*tot-invar-d*: *total-invariance<sub>D</sub> d rel*  
**shows** *is-symmetry (minimizer f domain<sub>f</sub> d img) (Invariance rel)*  
**proof** –  
**have** *is-symmetry (closest-preimg-distance f domain<sub>f</sub> d) (Invariance rel)*  
 using *r-refl tot-invar-d closest-dist-invar-under-refl-rel-and-tot-invar-dist*  
 by *simp*  
**moreover have** *minimizer f domain<sub>f</sub> d img =*  
*( $\lambda x. \text{arg-min-set } x \text{ img}) \circ (\text{closest-preimg-distance } f \text{ domain}_f d)$*   
 unfolding *comp-def*  
 by *auto*  
**ultimately show** *?thesis*  
 using *invar-comp*  
 by *simp*  
 qed

**theorem** *group-act-invar-dist-and-invar-f-imp-invar-minimizer:*

**fixes**  
*f* :: '*x*  $\Rightarrow$  '*y* **and**  
*domain<sub>f</sub>* :: '*x* set **and**  
*d* :: '*x* Distance **and**  
*img* :: '*y* set **and**  
*X* :: '*x* set **and**  
*G* :: '*z* monoid **and**  
 $\varphi$  :: ('*z*, '*x*) binary-fun  
**defines**  
*rel*  $\equiv$  *action-induced-rel (carrier G) X  $\varphi$*  **and**  
*rel'*  $\equiv$  *action-induced-rel (carrier G) domain<sub>f</sub>  $\varphi$*   
**assumes**  
*action- $\varphi$* : *group-action G X  $\varphi$*  **and**  
*domain<sub>f</sub>*  $\subseteq$  *X* **and**  
*closed-domain*: *closed-restricted-rel rel X domain<sub>f</sub>* **and**  
  
*invar-d*: *invariance<sub>D</sub> d (carrier G) X  $\varphi$*  **and**  
*invar-f*: *is-symmetry f (Invariance rel')*  
**shows** *is-symmetry (minimizer f domain<sub>f</sub> d img) (Invariance rel)*  
**proof** –

**let**  
 $?ψ = λ g. id$  **and**  
 $?img = λ x. img$   
**have** *is-symmetry*  $f$  (*action-induced-equivariance* (*carrier*  $G$ ) *domain<sub>f</sub>*  $φ$   $?ψ$ )  
**using** *invar-f rewrite-invar-as-equivar*  
**unfolding** *rel'-def*  
**by** *blast*  
**moreover have** *group-action*  $G$  *UNIV*  $?ψ$   
**using** *const-id-is-group-act action-φ*  
**unfolding** *group-action-def group-hom-def*  
**by** *blast*  
**moreover have** *is-symmetry*  $?img$  (*action-induced-equivariance* (*carrier*  $G$ )  $X$   $φ$   
(*set-action*  $?ψ$ ))  
**unfolding** *action-induced-equivariance-def*  
**by** *fastforce*  
**ultimately have**  
*is-symmetry* ( $λ x. minimizer f domain_f d (?img x) x$   
(*action-induced-equivariance* (*carrier*  $G$ )  $X$   $φ$  (*set-action*  $?ψ$ ))  
**using** *assms group-act-invar-dist-and-equivar-f-imp-equivar-minimizer* [*of*  
 $G$   $X$   $φ$   $?ψ$  *domain<sub>f</sub>*  $?img$   $d$   $f$ ]  
**by** *blast*  
**hence** *is-symmetry* (*minimizer*  $f$  *domain<sub>f</sub>*  $d$  *img*)  
(*action-induced-equivariance* (*carrier*  $G$ )  $X$   $φ$  (*set-action*  $?ψ$ ))  
**by** *blast*  
**thus** *?thesis*  
**unfolding** *rel-def set-action.simps*  
**using** *rewrite-invar-as-equivar image-id*  
**by** *metis*  
**qed**

## 5.6.2 Distance Rationalization as Minimizer

**lemma**  $\mathcal{K}_{\mathcal{E}}$ -*is-preimg*:

**fixes**

$d :: ('a, 'v)$  *Election Distance* **and**  
 $C :: ('a, 'v, 'r)$  *Result Consensus-Class* **and**  
 $E :: ('a, 'v)$  *Election* **and**  
 $w :: 'r$

**shows** *preimg* (*elect-r*  $\circ$  *fun<sub>E</sub>* (*rule-K*  $C$ )) (*elections-K*  $C$ )  $\{w\} = \mathcal{K}_{\mathcal{E}} C w$

**proof** –

**have** *preimg* (*elect-r*  $\circ$  *fun<sub>E</sub>* (*rule-K*  $C$ )) (*elections-K*  $C$ )  $\{w\} =$

$\{E \in \text{elections-K } C. (\text{elect-r} \circ \text{fun}_E (\text{rule-K } C)) E = \{w\}\}$

**by** *simp*

**also have**  $\{E \in \text{elections-K } C. (\text{elect-r} \circ \text{fun}_E (\text{rule-K } C)) E = \{w\}\} =$

$\{E \in \text{elections-K } C. \text{elect} (\text{rule-K } C) (\text{voters-}\mathcal{E} E) (\text{alternatives-}\mathcal{E} E) (\text{profile-}\mathcal{E} E) = \{w\}\}$

**by** *simp*

**also have**

$\{E \in \text{elections-K } C. \text{elect} (\text{rule-K } C) (\text{voters-}\mathcal{E} E) (\text{alternatives-}\mathcal{E} E) (\text{profile-}\mathcal{E} E) = \{w\}\}$

$E) = \{w\}\} =$   
 $\text{elections-}\mathcal{K} \ C \cap \{E. \text{elect} (\text{rule-}\mathcal{K} \ C) (\text{voters-}\mathcal{E} \ E) (\text{alternatives-}\mathcal{E} \ E) (\text{profile-}\mathcal{E} \ E) = \{w\}\}$   
**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**  
 $d :: ('a, 'v) \text{ Election Distance}$  **and**  
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$  **and**  
 $E :: ('a, 'v) \text{ Election}$  **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**  $\mathcal{K}_{\mathcal{E}}\text{-is-preimg}$   
**by** *metis*  
**also have**  $\text{Inf} (d \ E \ ' (\text{preimg} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \{w\}))$

```

      = closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C)
d E {w}
  by simp
  finally show ?thesis
    by simp
qed

lemma (in result)  $\mathcal{R}_W$ -is-minimizer:
  fixes
    d :: ('a, 'v) Election Distance and
    C :: ('a, 'v, 'r Result) Consensus-Class
  shows funE ( $\mathcal{R}_W$  d C) =
    ( $\lambda$  E.  $\bigcup$  (minimizer (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C) d
      (singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV)) E))

proof
  fix E :: ('a, 'v) Election
  let ?min = (minimizer (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C) d
    (singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV)) E)
  have ?min = arg-min-set
    (closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C) d
    E)
    (singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV))
  by simp
  also have
    ... = singleton-set-system (arg-min-set (score d C E) (limit-set (alternatives- $\mathcal{E}$ 
    E) UNIV))
  proof (safe)
    fix R :: 'r set
    assume
      min: R  $\in$  arg-min-set
      (closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C)
      d E)
      (singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV))
    hence R  $\in$  singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV)
      using arg-min-subset subsetD
      by (metis (no-types, lifting))
    then obtain r :: 'r where
      res-singleton: R = {r} and
      r-in-lim-set: r  $\in$  limit-set (alternatives- $\mathcal{E}$  E) UNIV
    by auto
    have  $\nexists$  R'. R'  $\in$  singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E) UNIV)  $\wedge$ 
      closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C) d E
      R'
      < closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C)
      d E R
      using min arg-min-set.simps is-arg-min-def CollectD
      by (metis (mono-tags, lifting))
    hence  $\nexists$  r'. r'  $\in$  limit-set (alternatives- $\mathcal{E}$  E) UNIV  $\wedge$ 
      closest-preimg-distance (elect-r o funE (rule- $\mathcal{K}$  C)) (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*  
**hence**  $\nexists \ r'. \ r' \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV \wedge \text{score } d \ C \ E \ r' < \text{score}$   
 $d \ C \ E \ r$   
**using** *score-is-closest-preimg-dist*  
**by** *metis*  
**hence**  $r \in \text{arg-min-set } (\text{score } d \ C \ E) (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$   
**using** *r-in-lim-set arg-min-set.simps is-arg-min-def CollectI*  
**by** *metis*  
**thus**  $R \in \text{singleton-set-system } (\text{arg-min-set } (\text{score } d \ C \ E) (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV))$   
**using** *res-singleton*  
**by** *simp*  
**next**  
**fix**  $R :: 'r \text{ set}$   
**assume**  
 $R \in \text{singleton-set-system } (\text{arg-min-set } (\text{score } d \ C \ E) (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV))$   
**then obtain**  $r :: 'r$  **where**  
*res-singleton:*  $R = \{r\}$  **and**  
*r-min-lim-set:*  $r \in \text{arg-min-set } (\text{score } d \ C \ E) (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$   
**by** *auto*  
**hence**  $\nexists \ r'. \ r' \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV \wedge \text{score } d \ C \ E \ r' < \text{score}$   
 $d \ C \ E \ r$   
**using** *CollectD arg-min-set.simps is-arg-min-def*  
**by** *metis*  
**hence**  $\nexists \ r'. \ r' \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV \wedge$   
 $\text{closest-preimg-distance } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E$   
 $\{r'\}$   
 $< \text{closest-preimg-distance } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C)$   
 $d \ E \ \{r\}$   
**using** *score-is-closest-preimg-dist*  
**by** *metis*  
**moreover have**  $\forall \ R' \in \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV).$   
 $\exists \ r' \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV. \ R' = \{r'\}$   
**by** *auto*  
**ultimately have**  $\nexists \ R'. \ R' \in \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV) \wedge$   
 $\text{closest-preimg-distance } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E \ R'$   
 $< \text{closest-preimg-distance } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d \ E$   
 $R$   
**using** *res-singleton*  
**by** *auto*  
**moreover have**  $R \in \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$



using *r-min-lim-set res-singleton arg-min-subset*  
 by *fastforce*  
 ultimately show  $R \in \text{arg-min-set}$   
     (*closest-preimg-distance* (*elect-r*  $\circ$  *fun<sub>E</sub>* (*rule-K C*)) (*elections-K C*)  
*d E*)  
     (*singleton-set-system* (*limit-set* (*alternatives-E E*) *UNIV*))  
 using *arg-min-set.simps is-arg-min-def CollectI*  
 by (*metis* (*mono-tags, lifting*))  
 qed  
 also have (*arg-min-set* (*score d C E*) (*limit-set* (*alternatives-E E*) *UNIV*)) =  
*fun<sub>E</sub>* (*R<sub>W</sub>* *d C*) *E*  
 by *simp*  
 finally have  $\bigcup ?min = \bigcup (\text{singleton-set-system } (\text{fun}_E (\mathcal{R}_W \ d \ C) \ E))$   
 by *presburger*  
 thus  $\text{fun}_E (\mathcal{R}_W \ d \ C) \ E = \bigcup ?min$   
 using *un-left-inv-singleton-set-system*  
 by *auto*  
 qed

## Invariance

**theorem** (*in result*) *tot-invar-dist-imp-invar-dr-rule*:

fixes  
   *d* :: (*'a*, *'v*) *Election Distance* and  
   *C* :: (*'a*, *'v*, *'r Result*) *Consensus-Class* and  
   *rel* :: (*'a*, *'v*) *Election rel*  
 assumes  
   *r-refl*: *refl-on* (*elections-K C*) (*Restr rel* (*elections-K C*)) and  
   *tot-invar-d*: *total-invariance<sub>D</sub>* *d rel* and  
   *invar-res*: *is-symmetry* ( $\lambda E. \text{limit-set } (\text{alternatives-E } E) \text{ UNIV}$ ) (*Invariance rel*)  
 shows *is-symmetry* ( $\text{fun}_E (\text{distance-}\mathcal{R} \ d \ C)$ ) (*Invariance rel*)  
 proof –  
   let  $?min = \lambda E. \bigcup \circ (\text{minimizer } (\text{elect-r} \circ \text{fun}_E (\text{rule-K } C)) (\text{elections-K } C) \ d$   
     (*singleton-set-system* (*limit-set* (*alternatives-E E*)  
   *UNIV*)))  
   have  $\forall E. \text{is-symmetry } (?min \ E) \ (\text{Invariance rel})$   
   using *r-refl tot-invar-d invar-comp*  
     *refl-rel-and-tot-invar-dist-imp-invar-minimizer*[of  
       *elections-K C rel d elect-r*  $\circ \text{fun}_E (\text{rule-K } C)$ ]  
   by *blast*  
 moreover have *is-symmetry*  $?min \ (\text{Invariance rel})$   
 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 ( $\lambda E. ?min \ E \ E$ ) =  $\text{fun}_E (\mathcal{R}_W \ d \ C)$   
 using *R<sub>W</sub>-is-minimizer*

**unfolding** *comp-def fun<sub>E</sub>.sims*  
**by** *metis*  
**finally have** *invar- $\mathcal{R}_W$ : is-symmetry (fun<sub>E</sub> ( $\mathcal{R}_W$  d C)) (Invariance rel)*  
**by** *simp*  
**hence** *is-symmetry ( $\lambda$  E. limit-set (alternatives- $\mathcal{E}$  E) UNIV - fun<sub>E</sub> ( $\mathcal{R}_W$  d C)*  
*E) (Invariance rel)*  
**using** *invar-res*  
**by** *fastforce*  
**thus** *is-symmetry (fun<sub>E</sub> (distance- $\mathcal{R}$  d C)) (Invariance rel)*  
**using** *invar- $\mathcal{R}_W$*   
**by** *auto*  
**qed**

**theorem** (*in result*) *invar-dist-cons-imp-invar-dr-rule:*  
**fixes**  
*d :: ('a, 'v) Election Distance and*  
*C :: ('a, 'v, 'r Result) Consensus-Class and*  
*G :: 'x monoid and*  
 *$\varphi$  :: ('x, ('a, 'v) Election) binary-fun and*  
*B :: ('a, 'v) Election set*  
**defines**  
*rel  $\equiv$  action-induced-rel (carrier G) B  $\varphi$  and*  
*rel'  $\equiv$  action-induced-rel (carrier G) (elections- $\mathcal{K}$  C)  $\varphi$*   
**assumes**  
*action- $\varphi$ : group-action G B  $\varphi$  and*  
*consensus-C-in-B: elections- $\mathcal{K}$  C  $\subseteq$  B and*  
*closed-domain:*  
*closed-restricted-rel rel B (elections- $\mathcal{K}$  C) and*  
*invar-res: is-symmetry ( $\lambda$  E. limit-set (alternatives- $\mathcal{E}$  E) UNIV) (Invariance*  
*rel) and*  
*invar-d: invariance<sub>D</sub> d (carrier G) B  $\varphi$  and*  
*invar-C-winners: is-symmetry (elect-r  $\circ$  fun<sub>E</sub> (rule- $\mathcal{K}$  C)) (Invariance rel')*  
**shows** *is-symmetry (fun<sub>E</sub> (distance- $\mathcal{R}$  d C)) (Invariance rel)*  
**proof** –  
**let** *?min =  $\lambda$  E.  $\bigcup \circ$  (minimizer (elect-r  $\circ$  fun<sub>E</sub> (rule- $\mathcal{K}$  C)) (elections- $\mathcal{K}$  C) d*  
*(singleton-set-system (limit-set (alternatives- $\mathcal{E}$  E)*  
*UNIV)))*  
**have**  *$\forall$  E. is-symmetry (?min E) (Invariance rel)*  
**using** *action- $\varphi$  closed-domain consensus-C-in-B invar-d invar-C-winners*  
*group-act-invar-dist-and-invar-f-imp-invar-minimizer rel-def*  
*rel'-def invar-comp*  
**by** (*metis (no-types, lifting)*)  
**moreover have** *is-symmetry ?min (Invariance rel)*  
**using** *invar-res*  
**by** *auto*  
**ultimately have** *is-symmetry ( $\lambda$  E. ?min E E) (Invariance rel)*  
**using** *invar-parameterized-fun[of ?min rel]*  
**by** *blast*  
**also have** *( $\lambda$  E. ?min E E) = fun<sub>E</sub> ( $\mathcal{R}_W$  d C)*

**using**  $\mathcal{R}_{\mathcal{W}}$ -is-minimizer  
**unfolding**  $\text{comp-def fun}_{\mathcal{E}}.\text{sims}$   
**by** *metis*  
**finally have**  $\text{invar-}\mathcal{R}_{\mathcal{W}}$ : *is-symmetry* ( $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C)$ ) (*Invariance rel*)  
**by** *simp*  
**hence** *is-symmetry* ( $\lambda E. \text{limit-set} (\text{alternatives-}\mathcal{E} E) \text{UNIV} -$   
 $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) E$ ) (*Invariance rel*)  
**using** *invar-res*  
**by** *fastforce*  
**thus** *is-symmetry* ( $\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} d C)$ ) (*Invariance rel*)  
**using**  $\text{invar-}\mathcal{R}_{\mathcal{W}}$   
**by** *simp*  
**qed**

## Equivariance

**theorem** (*in result*) *invar-dist-equivar-cons-imp-equivar-dr-rule*:

**fixes**

$d :: ('a, 'v) \text{ Election Distance}$  **and**  
 $C :: ('a, 'v, 'r) \text{ Consensus-Class}$  **and**  
 $G :: 'x \text{ monoid}$  **and**  
 $\varphi :: ('x, ('a, 'v) \text{ Election}) \text{ binary-fun}$  **and**  
 $\psi :: ('x, 'r) \text{ binary-fun}$  **and**  
 $B :: ('a, 'v) \text{ Election set}$

**defines**

$\text{rel} \equiv \text{action-induced-rel} (\text{carrier } G) B \varphi$  **and**  
 $\text{rel}' \equiv \text{action-induced-rel} (\text{carrier } G) (\text{elections-}\mathcal{K} C) \varphi$  **and**  
 $\text{equivar-prop} \equiv$   
 $\text{action-induced-equivariance} (\text{carrier } G) (\text{elections-}\mathcal{K} C) \varphi (\text{set-action } \psi)$  **and**  
 $\text{equivar-prop-global-set-valued} \equiv$   
 $\text{action-induced-equivariance} (\text{carrier } G) B \varphi (\text{set-action } \psi)$  **and**  
 $\text{equivar-prop-global-result-valued} \equiv$   
 $\text{action-induced-equivariance} (\text{carrier } G) B \varphi (\text{result-action } \psi)$

**assumes**

$\text{action-}\varphi$ : *group-action*  $G B \varphi$  **and**  
 $\text{group-act-res}$ : *group-action*  $G \text{UNIV } \psi$  **and**  
 $\text{cons-elect-set}$ :  $\text{elections-}\mathcal{K} C \subseteq B$  **and**  
 $\text{closed-domain}$ : *closed-restricted-rel*  $\text{rel } B (\text{elections-}\mathcal{K} C)$  **and**  
 $\text{equivar-res}$ :  
 $\text{is-symmetry} (\lambda E. \text{limit-set} (\text{alternatives-}\mathcal{E} E) \text{UNIV}) \text{equivar-prop-global-set-valued}$

**and**

$\text{invar-d}$ : *invariance<sub>D</sub>*  $d (\text{carrier } G) B \varphi$  **and**  
 $\text{equivar-C-winners}$ : *is-symmetry* ( $\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)$ ) *equivar-prop*  
**shows** *is-symmetry* ( $\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} d C)$ ) *equivar-prop-global-result-valued*

**proof** –

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

$E$

**let**  $?min = \lambda E. \bigcup \circ (\text{minimizer} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) d$

$(\text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E))$   
 $\text{UNIV}))$   
**let**  $?\psi' = \text{set-action } (\text{set-action } \psi)$   
**let**  $?\text{equivar-prop-global-set-valued}' = \text{action-induced-equivariance } (\text{carrier } G) \ B$   
 $\varphi \ ?\psi'$   
**have**  $\forall \ E \ g. \ g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$   
 $\text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}) =$   
 $\{\{r\} \mid r. \ r \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}\}$   
**by** *simp*  
**moreover have**  
 $\forall \ E \ g. \ g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$   
 $\text{limit-set } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV} = \psi \ g \ ' (\text{limit-set } (\text{alternatives-}\mathcal{E}$   
 $E) \ \text{UNIV})$   
**using** *equivar-res action- $\varphi$  group-action.element-image*  
**unfolding** *equivar-prop-global-set-valued-def action-induced-equivariance-def*  
**by** *fastforce*  
**ultimately have**  $\forall \ E \ g. \ g \in \text{carrier } G \longrightarrow E \in B \longrightarrow$   
 $\text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ (\varphi \ g \ E)) \ \text{UNIV}) =$   
 $\{\{r\} \mid r. \ r \in \psi \ g \ ' (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})\}$   
**by** *simp*  
**moreover have**  $\forall \ E \ g. \ \{\{r\} \mid r. \ r \in \psi \ g \ ' (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})\}$   
 $= \{\psi \ g \ ' \{r\} \mid r. \ r \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}\}$   
**by** *blast*  
**moreover have**  $\forall \ E \ g. \ \{\psi \ g \ ' \{r\} \mid r. \ r \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}\}$   
 $=$   
 $?\psi' \ g \ \{\{r\} \mid r. \ r \in \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}\}$   
**unfolding** *set-action.simps*  
**by** *blast*  
**ultimately have** *is-symmetry*  $(\lambda \ E. \ \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E}$   
 $E) \ \text{UNIV}))$   
 $?\text{equivar-prop-global-set-valued}'$   
**using** *rewrite-equivariance[of*  
 $\lambda \ E. \ \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ \text{carrier } G$   
 $B \ \varphi \ ?\psi']$   
**by** *force*  
**moreover have** *group-action*  $G \ \text{UNIV} \ (\text{set-action } \psi)$   
**unfolding** *set-action.simps*  
**using** *group-act-induces-set-group-act[of*  $G \ \text{UNIV} \ \psi]$  *group-act-res*  
**by** *simp*  
**ultimately have** *is-symmetry*  $?\text{min-}E \ ?\text{equivar-prop-global-set-valued}'$   
**using** *action- $\varphi$  invar-d cons-elect-set closed-domain equivar-C-winners*  
 $\text{group-act-invar-dist-and-equivar-f-imp-equivar-minimizer[of}$   
 $G \ B \ \varphi \ \text{set-action } \psi \ \text{elections-}\mathcal{K} \ C$   
 $\lambda \ E. \ \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$   
 $d \ \text{elect-r} \circ \text{fun}_{\mathcal{E}} \ (\text{rule-}\mathcal{K} \ C)]$   
**unfolding** *rel'-def rel-def equivar-prop-def*  
**by** *metis*  
**moreover have** *is-symmetry*  $\bigcup \ (\text{action-induced-equivariance } (\text{carrier } G) \ \text{UNIV}$   
 $?\psi' \ (\text{set-action } \psi))$

**using** *equivar-union-under-img-act*[*of carrier G ψ*]  
**by** *simp*  
**ultimately have** *is-symmetry*  $(\bigcup \circ ?min-E)$  *equivar-prop-global-set-valued*  
**unfolding** *equivar-prop-global-set-valued-def*  
**using** *equivar-ind-by-act-comp*[*of ?min-E B UNIV*]  
**by** *simp*  
**moreover have**  $(\lambda E. ?min E E) = \bigcup \circ ?min-E$   
**unfolding** *comp-def*  
**by** *simp*  
**ultimately have** *is-symmetry*  $(\lambda E. ?min E E)$  *equivar-prop-global-set-valued*  
**by** *simp*  
**moreover have**  $(\lambda E. ?min E E) = fun_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C)$   
**using**  *$\mathcal{R}_{\mathcal{W}}$ -is-minimizer*  
**unfolding** *comp-def fun<sub>ε</sub>.simps*  
**by** *metis*  
**ultimately have** *equivar- $\mathcal{R}_{\mathcal{W}}$ : is-symmetry*  $(fun_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C))$  *equivar-prop-global-set-valued*  
**by** *simp*  
**moreover have**  $\forall g \in carrier G. bij (\psi g)$   
**using** *group-act-res*  
**unfolding** *bij-betw-def*  
**by** *(simp add: group-action.inj-prop group-action.surj-prop)*  
**ultimately have**  
*is-symmetry*  $(\lambda E. limit-set (alternatives-\mathcal{E} E) UNIV - fun_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) E)$   
*equivar-prop-global-set-valued*  
**using** *equivar-res equivar-set-minus*  
**unfolding** *equivar-prop-global-set-valued-def action-induced-equivariance-def set-action.simps*  
**by** *blast*  
**thus** *is-symmetry*  $(fun_{\mathcal{E}} (distance-\mathcal{R} d C))$  *equivar-prop-global-result-valued*  
**using** *equivar- $\mathcal{R}_{\mathcal{W}}$*   
**unfolding** *equivar-prop-global-result-valued-def equivar-prop-global-set-valued-def*  
*rewrite-equivariance*  
**by** *simp*  
**qed**

### 5.6.3 Symmetry Property Inference Rules

**theorem** (*in result*) *anon-dist-and-cons-imp-anon-dr*:  
**fixes**  
 $d :: ('a, 'v) Election Distance$  **and**  
 $C :: ('a, 'v, 'r Result) Consensus-Class$   
**assumes**  
*anon-d: distance-anonymity' valid-elections d* **and**  
*anon-C: consensus-rule-anonymity' (elections- $\mathcal{K}$  C) C* **and**  
*closed-C: closed-restricted-rel (anonymity<sub>R</sub> valid-elections) valid-elections (elections- $\mathcal{K}$  C)*  
**shows** *anonymity' valid-elections (distance- $\mathcal{R}$  d C)*  
**proof** –  
**have**  $\forall \pi. \forall E \in elections-\mathcal{K} C. \varphi\text{-anon} (elections-\mathcal{K} C) \pi E = \varphi\text{-anon valid-elections}$   
 $\pi E$

**using** *cons-domain-valid extensional-continuation-subset*  
**unfolding**  $\varphi$ -anon.simps  
**by** *metis*  
**hence** *action-induced-rel* (*carrier anonymity<sub>G</sub>*) (*elections-K C*) ( $\varphi$ -anon *valid-elections*)  
 =  
*action-induced-rel* (*carrier anonymity<sub>G</sub>*) (*elections-K C*) ( $\varphi$ -anon (*elections-K C*))  
**using** *coinciding-actions-ind-equal-rel*[of *carrier anonymity<sub>G</sub>* *elections-K C*]  
**by** *metis*  
**hence** *is-symmetry* (*elect-r*  $\circ$  *fun<sub>E</sub>* (*rule-K C*))  
 (*Invariance* (*action-induced-rel*  
 (*carrier anonymity<sub>G</sub>*) (*elections-K C*) ( $\varphi$ -anon *valid-elections*)))  
**using** *anon-C*  
**unfolding** *consensus-rule-anonymity'.simps* *anonymity<sub>R</sub>.simps*  
**by** *presburger*  
**thus** *?thesis*  
**using** *cons-domain-valid*[of *C*] *assms anonymous-group-action.group-action-axioms*  
*well-formed-res-anon invar-dist-cons-imp-invar-dr-rule*[of *anonymity<sub>G</sub>*]  
**unfolding** *distance-anonymity'.simps* *anonymity<sub>R</sub>.simps* *anonymity'.simps*  
*consensus-rule-anonymity'.simps*  
**by** *blast*  
**qed**

**theorem** (**in** *result-properties*) *neutr-dist-and-cons-imp-neutr-dr*:  
**fixes**  
*d* :: ('a, 'v) *Election Distance* **and**  
*C* :: ('a, 'v, 'b *Result*) *Consensus-Class*  
**assumes**  
*neutr-d*: *distance-neutrality valid-elections d* **and**  
*neutr-C*: *consensus-rule-neutrality* (*elections-K C*) *C* **and**  
*closed-C*:  
*closed-restricted-rel* (*neutrality<sub>R</sub>* *valid-elections*) *valid-elections* (*elections-K C*)  
**shows** *neutrality valid-elections* (*distance-R d C*)  
**proof** –  
**have**  $\forall \pi. \forall E \in \text{elections-K } C. \varphi\text{-neutr } \text{valid-elections } \pi E = \varphi\text{-neutr } (\text{elections-K } C) \pi E$   
**using** *cons-domain-valid extensional-continuation-subset*  
**unfolding**  $\varphi$ -neutr.simps  
**by** *metis*  
**hence** *is-symmetry* (*elect-r*  $\circ$  *fun<sub>E</sub>* (*rule-K C*))  
 (*action-induced-equivariance* (*carrier neutrality<sub>G</sub>*) (*elections-K C*)  
 ( $\varphi$ -neutr *valid-elections*) (*set-action*  $\psi$ -neutr))  
**using** *neutr-C equivar-ind-by-act-coincide*[of *carrier neutrality<sub>G</sub>*]  
**unfolding** *consensus-rule-neutrality.simps*  
**by** (*metis* (*no-types*, *lifting*))  
**thus** *?thesis*  
**using** *neutr-d closed-C*  $\varphi$ -neutr-act.group-action-axioms *well-formed-res-neutr*  
*act-neutr*  
*cons-domain-valid*[of *C*] *invar-dist-equivar-cons-imp-equivar-dr-rule*[of

*neutralityg*  
*valid-elections  $\varphi$ -neutr valid-elections]*  
 by *simp*  
 qed

**theorem** *reversal-sym-dist-and-cons-imp-reversal-sym-dr:*  
 fixes  
*d :: ('a, 'c) Election Distance and*  
*C :: ('a, 'c, 'a rel Result) Consensus-Class*  
 assumes  
*rev-sym-d: distance-reversal-symmetry valid-elections d and*  
*rev-sym-C: consensus-rule-reversal-symmetry (elections- $\mathcal{K}$  C) C and*  
*closed-C: closed-restricted-rel (reversal $_{\mathcal{R}}$  valid-elections) valid-elections (elections- $\mathcal{K}$  C)*  
 shows *reversal-symmetry valid-elections (SWF-result.distance- $\mathcal{R}$  d C)*  
**proof** –  
 have  $\forall \pi. \forall E \in \text{elections-}\mathcal{K} \ C. \varphi\text{-rev valid-elections } \pi \ E = \varphi\text{-rev (elections-}\mathcal{K} \ C) \ \pi \ E$   
 using *cons-domain-valid extensional-continuation-subset*  
 unfolding  *$\varphi$ -rev.simps*  
 by *metis*  
 hence *is-symmetry (elect-r  $\circ$  fun $_{\mathcal{E}}$  (rule- $\mathcal{K}$  C))*  
   *(action-induced-equivariance (carrier reversal $_{\mathcal{G}}$ ) (elections- $\mathcal{K}$  C))*  
   *( $\varphi$ -rev valid-elections) (set-action  $\psi$ -rev))*  
 using *rev-sym-C equivar-ind-by-act-coincide[of carrier reversal $_{\mathcal{G}}$ ]*  
 unfolding *consensus-rule-reversal-symmetry.simps*  
 by *(metis (no-types, lifting))*  
 thus *?thesis*  
 using *cons-domain-valid rev-sym-d closed-C  $\varphi$ -rev-act.group-action-axioms*  
    *$\psi$ -rev-act.group-action-axioms  $\varphi$ - $\psi$ -rev-well-formed*  
   *SWF-result.invar-dist-equivar-cons-imp-equivar-dr-rule[of*  
   *reversal $_{\mathcal{G}}$  valid-elections  $\varphi$ -rev valid-elections  $\psi$ -rev C d]*  
 unfolding *distance-reversal-symmetry.simps reversal-symmetry-def reversal $_{\mathcal{R}}$ .simps*  
 by *metis*  
 qed

**theorem** *(in result) tot-hom-dist-imp-hom-dr:*  
 fixes  
*d :: ('a, nat) Election Distance and*  
*C :: ('a, nat, 'r Result) Consensus-Class*  
 assumes *distance-homogeneity finite-elections- $\mathcal{V}$  d*  
 shows *homogeneity finite-elections- $\mathcal{V}$  (distance- $\mathcal{R}$  d C)*  
**proof** –  
 have *Restr (homogeneity $_{\mathcal{R}}$  finite-elections- $\mathcal{V})$*  *(elections- $\mathcal{K}$  C) = homogeneity $_{\mathcal{R}}$*   
*(elections- $\mathcal{K}$  C)*  
 using *cons-domain-finite[of C]*  
 unfolding *homogeneity $_{\mathcal{R}}$ .simps finite-elections- $\mathcal{V}$ -def*  
 by *blast*  
 hence *refl-on (elections- $\mathcal{K}$  C) (Restr (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* ( $\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ )  
*(Invariance (homogeneity $_{\mathcal{R}}$  finite-elections- $\mathcal{V}$ ))*  
**using** *well-formed-res-homogeneity*  
**by** *simp*  
**ultimately show** *?thesis*  
**using** *assms tot-invar-dist-imp-invar-dr-rule [of  $C$  homogeneity $_{\mathcal{R}}$  finite-elections- $\mathcal{V}$ ]*  
 $d$ ]  
**unfolding** *distance-homogeneity-def homogeneity.simps*  
**by** *metis*  
**qed**

**theorem** (**in** *result*) *tot-hom-dist-imp-hom-dr'*:  
**fixes**  
 $d :: ('a, 'v::\text{linorder}) \text{ Election Distance}$  **and**  
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$   
**assumes** *distance-homogeneity' finite-elections- $\mathcal{V}$   $d$*   
**shows** *homogeneity' finite-elections- $\mathcal{V}$  (distance- $\mathcal{R}$   $d$   $C$ )*  
**proof** –  
**have** *Restr (homogeneity $_{\mathcal{R}}$ ' finite-elections- $\mathcal{V}$ ) (elections- $\mathcal{K}$   $C$ )*  
 $= \text{homogeneity}_{\mathcal{R}}' (\text{elections-}\mathcal{K} \ C)$   
**using** *cons-domain-finite*  
**unfolding** *homogeneity $_{\mathcal{R}}$ '.simps finite-elections- $\mathcal{V}$ -def*  
**by** *blast*  
**hence** *refl-on (elections- $\mathcal{K}$   $C$ ) (Restr (homogeneity $_{\mathcal{R}}$ ' finite-elections- $\mathcal{V}$ ) (elections- $\mathcal{K}$   $C$ ))*  
 $C$ ))  
**using** *refl-homogeneity $_{\mathcal{R}}$ '[of elections- $\mathcal{K}$   $C$ ] cons-domain-finite[of  $C$ ]*  
**by** *presburger*  
**moreover have**  
*is-symmetry* ( $\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV$ )  
*(Invariance (homogeneity $_{\mathcal{R}}$ ' finite-elections- $\mathcal{V}$ ))*  
**using** *well-formed-res-homogeneity'*  
**by** *simp*  
**ultimately show** *?thesis*  
**using** *assms tot-invar-dist-imp-invar-dr-rule*  
**unfolding** *distance-homogeneity'-def homogeneity'.simps*  
**by** *blast*  
**qed**

#### 5.6.4 Further Properties

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



end

## 5.7 Distance Rationalization on Election Quotients

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

### 5.7.1 Quotient Distances

```
fun distance_Q :: 'x Distance  $\Rightarrow$  'x set Distance where
  distance_Q d A B = (if (A = {}  $\wedge$  B = {}) then 0 else
    (if (A = {}  $\vee$  B = {}) then  $\infty$  else
       $\pi_Q$  (tup d) (A  $\times$  B)))
```

```
fun relation-paths :: 'x rel  $\Rightarrow$  'x list set where
  relation-paths r = {p.  $\exists$  k. (length p = 2 * k  $\wedge$  ( $\forall$  i < k. (p!(2 * i), p!(2 * i + 1))  $\in$  r))}
```

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

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

```
fun quotient-dist :: 'x rel  $\Rightarrow$  'x Distance  $\Rightarrow$  'x set Distance where
  quotient-dist r d A B = Inf ( $\bigcup$  { {path-length p d | p. p  $\in$  admissible-paths r A B} })
```

```
fun distance-infimum_Q :: 'x Distance  $\Rightarrow$  'x set Distance where
  distance-infimum_Q d A B = Inf {d a b | a b. a  $\in$  A  $\wedge$  b  $\in$  B}
```

```
fun simple :: 'x rel  $\Rightarrow$  'x set  $\Rightarrow$  'x Distance  $\Rightarrow$  bool where
  simple r X d =
    ( $\forall$  A  $\in$  X // r. ( $\exists$  a  $\in$  A.  $\forall$  B  $\in$  X // r. distance-infimum_Q d A B = Inf {d a 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
```

$$\text{product}' r = \{(p_1, p_2). ((fst\ p_1, fst\ p_2) \in r \wedge snd\ p_1 = snd\ p_2) \\ \vee ((snd\ p_1, snd\ p_2) \in r \wedge fst\ p_1 = fst\ p_2)\}$$

## Auxiliary Lemmas

**lemma** *tot-dist-invariance-is-congruence*:

**fixes**

$d :: 'x$  *Distance* **and**

$r :: 'x$  *rel*

**shows**  $(total\_invariance_{\mathcal{D}}\ d\ r) = (tup\ d\ respects\ (product\ r))$

**unfolding**  $total\_invariance_{\mathcal{D}}.simps\ is\_symmetry.simps\ congruent\_def$

**by** *blast*

**lemma** *product-helper*:

**fixes**

$r :: 'x$  *rel* **and**

$X :: 'x$  *set*

**shows**

*trans-imp*:  $Relation.trans\ r \implies Relation.trans\ (product\ r)$  **and**

*refl-imp*:  $refl\_on\ X\ r \implies refl\_on\ (X \times X)\ (product\ r)$  **and**

*sym*:  $sym\_on\ X\ r \implies sym\_on\ (X \times X)\ (product\ r)$

**unfolding**  $Relation.trans\_def\ refl\_on\_def\ sym\_on\_def\ product.simps$

**by** *auto*

**theorem** *dist-pass-to-quotient*:

**fixes**

$d :: 'x$  *Distance* **and**

$r :: 'x$  *rel* **and**

$X :: 'x$  *set*

**assumes**

*equiv-X-r*:  $equiv\ X\ r$  **and**

*tot-inv-dist-d-r*:  $total\_invariance_{\mathcal{D}}\ d\ r$

**shows**  $\forall\ A\ B. A \in X\ /\!/\ r \wedge B \in X\ /\!/\ r \longrightarrow (\forall\ a\ b. a \in A \wedge b \in B \longrightarrow distance_Q\ d\ A\ B = d\ a\ b)$

**proof** (*safe*)

**fix**

$A :: 'x$  *set* **and**

$B :: 'x$  *set* **and**

$a :: 'x$  **and**

$b :: 'x$

**assume**

*a-in-A*:  $a \in A$  **and**

$A \in X\ /\!/\ r$

**moreover with** *equiv-X-r quotient-eq-iff*

**have**  $(a, a) \in r$

**by** *metis*

**moreover with** *equiv-X-r*

**have** *a-in-X*:  $a \in X$

**using** *equiv-class-eq-iff*

by *metis*  
 ultimately have  $A\text{-eq-}r\text{-}a: A = r `` \{a\}$   
 using *equiv-X-r quotient-eq-iff quotientI*  
 by *fast*  
 assume  
 $b\text{-in-}B: b \in B$  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  $b\text{-in-}X: b \in X$   
 using *equiv-class-eq-iff*  
 by *metis*  
 ultimately have  $B\text{-eq-}r\text{-}b: B = r `` \{b\}$   
 using *equiv-X-r quotient-eq-iff quotientI*  
 by *fast*  
 from  $A\text{-eq-}r\text{-}a$   $B\text{-eq-}r\text{-}b$   $a\text{-in-}X$   $b\text{-in-}X$   
 have  $A \times B \in (X \times X) // (\text{product } r)$   
 unfolding *quotient-def*  
 by *fastforce*  
 moreover have *equiv*  $(X \times X)$  (*product r*)  
 using *equiv-X-r product-helper UNIV-Times-UNIV equivE equivI*  
 by *metis*  
 moreover have *tup d respects* (*product r*)  
 using *tot-inv-dist-d-r tot-dist-invariance-is-congruence*  
 by *metis*  
 ultimately show  $\text{distance}_Q d A B = d a b$   
 unfolding *distance\_Q.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  
 $\text{length } p = 2 * k$  and

```

    rel:  $\forall i < k. (p!(2 * i), p!(2 * i + 1)) \in r$ 
  by auto
moreover obtain  $k' :: nat$  where
  i-cases:  $i = 2 * k' \vee i = 2 * k' + 1$ 
  using diff-Suc-1 even-Suc oddE odd-two-times-div-two-nat
  by metis
moreover assume  $i < length\ p$ 
ultimately have  $k' < k$ 
  by linarith
thus  $p!i \in X$ 
  using assms rel i-cases
  by blast
qed

lemma admissible-path-len:
  fixes
    d :: 'x Distance and
    r :: 'x rel and
    X :: 'x set and
    a :: 'x and
    b :: 'x and
    p :: 'x list
  assumes refl-on X r
  shows triangle-ineq X d  $\wedge$   $p \in relation-paths\ r \wedge total-invariance_{\mathcal{D}}\ d\ r \wedge$ 
     $a \in X \wedge b \in X \longrightarrow path-length\ (a\#\!p@[b])\ d \geq d\ a\ b$ 
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:  $total-invariance_{\mathcal{D}}\ d\ r$  and
    hyp:  $\bigwedge a\ b. triangle-ineq\ X\ d \Longrightarrow xs \in relation-paths\ r \Longrightarrow total-invariance_{\mathcal{D}}\ d$ 
  r  $\Longrightarrow$ 
     $a \in X \Longrightarrow b \in X \Longrightarrow d\ a\ b \leq path-length\ (a\#\!xs@[b])\ d$ 
  then obtain  $k :: nat$  where
    len:  $length\ (x\#\!y\#\!xs) = 2 * k$ 
    by auto
  moreover have  $\forall i < k - 1. (xs!(2 * i), xs!(2 * i + 1)) =$ 
     $((x\#\!y\#\!xs)!(2 * (i + 1)), (x\#\!y\#\!xs)!(2 * (i + 1) + 1))$ 

```

by *simp*  
 ultimately have  $\forall i < k - 1. (xs!(2 * i), xs!(2 * i + 1)) \in r$   
 using *rel less-diff-conv*  
 unfolding *relation-paths.simps*  
 by *fastforce*  
 moreover have  $length\ xs = 2 * (k - 1)$   
 using *len*  
 by *simp*  
 ultimately have  $xs \in relation-paths\ r$   
 by *simp*  
 hence  $\forall x\ y. x \in X \wedge y \in X \longrightarrow d\ x\ y \leq path-length\ (x\#\!xs@[y])\ d$   
 using *ineq invar hyp*  
 by *blast*  
 moreover have  $path-length\ (a\#\!(x\#y\#xs)\@[b])\ d = d\ a\ x + path-length\ (y\#xs@[b])\ d$   
 d  
 by *simp*  
 moreover have  $x-rel-y: (x, y) \in r$   
 using *rel*  
 unfolding *relation-paths.simps*  
 by *fastforce*  
 ultimately have  $path-length\ (a\#\!(x\#y\#xs)\@[b])\ d \geq d\ a\ x + d\ y\ b$   
 using *assms add-left-mono assms refl-onD2 b-in-X*  
 unfolding *refl-on-def*  
 by *metis*  
 moreover have  $d\ a\ x + d\ y\ b = d\ a\ x + d\ x\ b$   
 using *invar x-rel-y rewrite-total-invariance<sub>D</sub> assms b-in-X*  
 unfolding *refl-on-def*  
 by *fastforce*  
 moreover have  $d\ a\ x + d\ x\ b \geq d\ a\ b$   
 using *a-in-X b-in-X x-rel-y assms ineq*  
 unfolding *refl-on-def triangle-ineq-def*  
 by *auto*  
 ultimately show  $d\ a\ b \leq path-length\ (a\#\!(x\#y\#xs)\@[b])\ d$   
 by *simp*  
 qed

**lemma** *quotient-dist-coincides-with-dist<sub>Q</sub>*:

fixes  
 $d :: 'x\ Distance$  and  
 $r :: 'x\ rel$  and  
 $X :: 'x\ set$   
 assumes  
 $equiv: equiv\ X\ r$  and  
 $tri: triangle-ineq\ X\ d$  and  
 $invar: total-invariance_{\mathcal{D}}\ d\ r$   
 shows  $\forall A \in X // r. \forall B \in X // r. quotient-dist\ r\ d\ A\ B = distance_Q\ d\ A\ B$   
 proof (clarify)  
 fix  
 $A :: 'x\ set$  and

```

    B :: 'x set
  assume
    A-in-quot-X: A ∈ X // r and
    B-in-quot-X: B ∈ X // r
  then obtain
    a :: 'x and
    b :: 'x where
      el: a ∈ A ∧ b ∈ B and
      def-dist: distanceQ d A B = d a b
    using dist-pass-to-quotient assms in-quotient-imp-non-empty ex-in-conv
    by (metis (full-types))
  hence equiv-class: A = r “ {a} ∧ B = r “ {b}
    using A-in-quot-X B-in-quot-X assms equiv-class-eq-iff equiv-class-self
    quotientI quotient-eq-iff
    by meson
  have subset-X: r ⊆ X × X ∧ A ⊆ X ∧ B ⊆ X
    using assms A-in-quot-X B-in-quot-X equiv-def refl-on-def Union-quotient
  Union-upper
    by metis
  have ∀ p ∈ admissible-paths r A B.
    (∃ p' x y. x ∈ A ∧ y ∈ B ∧ p' ∈ relation-paths r ∧ p = x#p'@[y])
    unfolding admissible-paths.simps
    by blast
  moreover have ∀ x y. x ∈ A ∧ y ∈ B ⟶ d x y = d a b
    using invar equiv-class
    by auto
  moreover have refl-on X r
    using equiv equiv-def
    by blast
  ultimately have ∀ p. p ∈ admissible-paths r A B ⟶ path-length p d ≥ d a b
    using admissible-path-len[of X r d] tri subset-X el invar in-mono
    by metis
  hence ∀ l. l ∈ ⋃ { {path-length p d | p. p ∈ admissible-paths r A B} } ⟶ l ≥
    d a b
    by blast
  hence geg: quotient-dist r d A B ≥ d a b
    unfolding quotient-dist.simps[of r d A B] le-Inf-iff
    by simp
  with el def-dist
  have geg: quotient-dist r d A B ≥ distanceQ d A B
    by presburger
  have [a, b] ∈ 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 ≤ d a b
    using quotient-dist.simps[of r d A B] CollectI Inf-lower ccpo-Sup-singleton
    by (metis (mono-tags, lifting))

```

```

thus quotient-dist  $r$   $d$   $A$   $B$  = distanceQ  $d$   $A$   $B$ 
using geq def-dist nle-le
by metis
qed

lemma inf-dist-coincides-with-distQ:
fixes
   $d :: 'x$  Distance and
   $r :: 'x$  rel and
   $X :: 'x$  set
assumes
  equiv-X-r: equiv  $X$   $r$  and
  tot-inv-d-r: total-invarianceD  $d$   $r$ 
shows  $\forall A \in X // r. \forall B \in X // r. \text{distance-infimum}_Q d A B = \text{distance}_Q d A B$ 
proof (clarify)
fix
   $A :: 'x$  set and
   $B :: 'x$  set
assume
  A-in-quot-X:  $A \in X // r$  and
  B-in-quot-X:  $B \in X // r$ 
then obtain
   $a :: 'x$  and
   $b :: 'x$  where
     $el$ :  $a \in A \wedge b \in B$  and
    def-dist: distanceQ  $d$   $A$   $B$  =  $d$   $a$   $b$ 
    using dist-pass-to-quotient equiv-X-r tot-inv-d-r in-quotient-imp-non-empty
ex-in-conv
    by (metis (full-types))
from def-dist equiv-X-r tot-inv-d-r
have  $\forall x y. x \in A \wedge y \in B \longrightarrow d x y = d a b$ 
    using dist-pass-to-quotient A-in-quot-X B-in-quot-X
    by force
hence  $\{d x y \mid x y. x \in A \wedge y \in B\} = \{d a b\}$ 
    using  $el$ 
    by blast
thus distance-infimumQ  $d$   $A$   $B$  = distanceQ  $d$   $A$   $B$ 
    unfolding distance-infimumQ.simps
    using def-dist
    by simp
qed

lemma inf-helper:
fixes
   $A :: 'x$  set and
   $B :: 'x$  set and
   $d :: 'x$  Distance
shows  $\text{Inf } \{d a b \mid a b. a \in A \wedge b \in B\} = \text{Inf } \{\text{Inf } \{d a b \mid b. b \in B\} \mid a. a \in A\}$ 

```

$A\}$   
**proof** –  
**have**  $\forall a b. a \in A \wedge b \in B \longrightarrow \text{Inf } \{d a b \mid b. b \in B\} \leq d a b$   
**using** *INF-lower Setcompr-eq-image*  
**by** *metis*  
**hence**  $\forall \alpha \in \{d a b \mid a b. a \in A \wedge b \in B\}. \exists \beta \in \{\text{Inf } \{d a b \mid b. b \in B\} \mid a. a \in A\}. \beta \leq \alpha$   
**by** *blast*  
**hence**  $\text{Inf } \{\text{Inf } \{d a b \mid b. b \in B\} \mid a. a \in A\} \leq \text{Inf } \{d a b \mid a b. a \in A \wedge b \in B\}$   
**using** *Inf-mono*  
**by** (*metis (no-types, lifting)*)  
**moreover have**  $\neg (\text{Inf } \{\text{Inf } \{d a b \mid b. b \in B\} \mid a. a \in A\} < \text{Inf } \{d a b \mid a b. a \in A \wedge b \in B\})$   
**proof** (*rule ccontr, safe*)  
**assume**  $\text{Inf } \{\text{Inf } \{d a b \mid b. b \in B\} \mid a. a \in A\} < \text{Inf } \{d a b \mid a b. a \in A \wedge b \in B\}$   
**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**  $a :: 'x \text{ where}$   
 $a\text{-in-}A: a \in A$  **and**  
 $\alpha = \text{Inf } \{d a b \mid b. b \in B\}$   
**by** *blast*  
**with** *less*  
**have** *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  $\text{simple} \ (\text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi) \ Y \ d$ 
proof ( $\text{unfold simple.simps, safe}$ )
  fix  $A :: 'y \text{ set}$ 
  assume  $\text{class}_Y$ :  $A \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi$ 
  have  $\text{equiv-rel}$ :  $\text{equiv } Y \ (\text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi)$ 
    using  $\text{assms rel-ind-by-group-act-equiv}$ 
    by blast
  with  $\text{class}_Y$  obtain  $a :: 'y$  where
     $a \text{-in-} A$ :  $a \in A$ 
    using  $\text{equiv-Eps-in}$ 
    by blast
  have  $\text{subset}$ :  $\forall B \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi. B \subseteq Y$ 
    using  $\text{equiv-rel in-quotient-imp-subset}$ 
    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  $\text{class}_Y$ 
    by blast
  hence  $\text{eq-dist}$ :
     $\forall B \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi.$ 
     $\forall B' \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi.$ 
     $\forall b \in B. \forall c \in B'. \forall g \in \text{carrier } G.$ 
     $d \ (\varphi \ g \ c) \ (\varphi \ g \ b) = d \ c \ b$ 
    using  $\text{invar rewrite-invariance}_{\mathcal{D}} \ \text{class}_Y$ 
    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  $\text{action-induced-rel.simps}$ 
    using  $\text{group-action.element-image action-}\varphi$ 
    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  $\text{Image-def}$ 
    by blast
  moreover have  $\text{equiv-class}$ :
     $\forall B. B \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi \longrightarrow$ 
     $(\forall b \in B. B = \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi \text{ “ } \{b\})$ 
    using  $\text{equiv-class-eq-iff equiv-rel insertI1 quotientI quotient-eq-iff rev-ImageI}$ 
    by meson
  ultimately have  $\text{closed-class}$ :
     $\forall B \in Y // \text{action-induced-rel} \ (\text{carrier } G) \ Y \ \varphi. \forall b \in B. \forall g \in \text{carrier } G. \varphi$ 
 $g \ b \in B$ 
    using  $\text{equiv-rel subset}$ 
    by blast
  with  $\text{eq-dist class}_Y$ 

```

```

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 classY 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 Distance and
    r :: 'x rel and
    X :: 'x set
  assumes
    equiv-on-X: equiv X r and
    invar: total-invarianceD d r
  shows simple r X d
proof (unfold simple.simps, safe)
  fix A :: 'x set
  assume A-quot-X: A ∈ X // r
  then obtain a :: 'x where

```

$a\text{-in-}A: a \in A$   
**using** *equiv-on-X equiv-Eps-in*  
**by** *blast*  
**have**  $\forall a \in A. A = r \text{ `` } \{a\}$   
**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-invariance<sub>D</sub>*  
**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 Quotient Consensus and Results

**fun** *elections- $\mathcal{K}_{\mathcal{Q}}$*  ::  $(\text{'a}, \text{'v}) \text{ Election rel} \Rightarrow (\text{'a}, \text{'v}, \text{'r Result}) \text{ Consensus-Class} \Rightarrow$   
 $(\text{'a}, \text{'v}) \text{ Election set set}$  **where**  
 $\text{elections-}\mathcal{K}_{\mathcal{Q}} r C = (\text{elections-}\mathcal{K} C) // r$

**fun** **(in result)** *limit-set<sub>Q</sub>* ::  $(\text{'a}, \text{'v}) \text{ Election set} \Rightarrow \text{'r set} \Rightarrow \text{'r set}$  **where**  
 $\text{limit-set}_{\mathcal{Q}} X \text{ res} = \bigcap \{\text{limit-set } (\text{alternatives-}\mathcal{E} E) \text{ res} \mid E. E \in X\}$

### Auxiliary Lemmas

**lemma** *closed-under-equiv-rel-subset*:  
**fixes**  
 $X :: \text{'x set}$  **and**  
 $Y :: \text{'x set}$  **and**  
 $Z :: \text{'x set}$  **and**

```

    r :: 'x rel
  assumes
    equiv X r and
    Y ⊆ X and
    Z ⊆ X and
    Z ∈ Y // r and
    closed-restricted-rel r X Y
  shows Z ⊆ Y
proof (safe)
  fix z :: 'x
  assume z ∈ Z
  then obtain y :: 'x where
    y ∈ 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-set-invar*:

```

  fixes
    d :: ('a, 'v) Election Distance and
    r :: ('a, 'v) Election rel and
    C :: ('a, 'v, 'r Result) Consensus-Class and
    X :: ('a, 'v) Election set and
    A :: ('a, 'v) Election set
  assumes
    quot-class: A ∈ X // r and
    equiv-rel: equiv X r and
    cons-subset: elections- $\mathcal{K}$  C ⊆ X and
    invar-res: is-symmetry (λ E. limit-set (alternatives- $\mathcal{E}$  E) UNIV) (Invariance r)
  shows ∀ a ∈ A. limit-set (alternatives- $\mathcal{E}$  a) UNIV = limit-setQ A UNIV
proof
  fix a :: ('a, 'v) Election
  assume a-in-A: a ∈ A
  hence ∀ b ∈ A. (a, b) ∈ r
  using quot-class equiv-rel quotient-eq-iff
  by metis
  hence ∀ b ∈ A. limit-set (alternatives- $\mathcal{E}$  b) UNIV = limit-set (alternatives- $\mathcal{E}$  a)

```

```

UNIV
  using invar-res
  unfolding is-symmetry.simps
  by (metis (mono-tags, lifting))
  hence limit-setQ A UNIV =  $\bigcap \{ \text{limit-set } (\text{alternatives-}\mathcal{E} \ a) \ UNIV \}$ 
  unfolding limit-setQ.simps
  using a-in-A
  by blast
  thus limit-set (alternatives- $\mathcal{E}$  a) UNIV = limit-setQ A UNIV
  by simp
qed

lemma (in result) preimg-invar:
  fixes
    f :: 'x  $\Rightarrow$  'y and
    domainf :: 'x set and
    d :: 'x Distance and
    r :: 'x rel and
    X :: 'x set
  assumes
    equiv-rel: equiv X r and
    cons-subset: domainf  $\subseteq$  X and
    closed-domain: closed-restricted-rel r X domainf and
    invar-f: is-symmetry f (Invariance (Restr r domainf))
  shows  $\forall y. (\text{preimg } f \text{ domain}_f y) // r = \text{preimg } (\pi_Q f) (\text{domain}_f // r) y$ 
  proof (safe)
    fix
      A :: 'x set and
      y :: 'y
    assume preimg-quot:  $A \in \text{preimg } f \text{ domain}_f y // r$ 
    hence A-in-dom:  $A \in \text{domain}_f // r$ 
      unfolding preimg.simps quotient-def
      by blast
    obtain x :: 'x where
      x  $\in$  preimg f domainf y and
      A-eq-img-singleton-r:  $A = r \text{ `` } \{x\}$ 
      using equiv-rel preimg-quot quotientE
      unfolding quotient-def
      by blast
    hence x-in-dom-and-f-x-y:  $x \in \text{domain}_f \wedge f x = y$ 
      unfolding preimg.simps
      by blast
    moreover have  $r \text{ `` } \{x\} \subseteq X$ 
      using equiv-rel equiv-type
      by fastforce
    ultimately have  $r \text{ `` } \{x\} \subseteq \text{domain}_f$ 
      using closed-domain A-eq-img-singleton-r A-in-dom
      by fastforce
    hence  $\forall x' \in r \text{ `` } \{x\}. (x, x') \in \text{Restr } r \text{ domain}_f$ 

```

```

    using x-in-dom-and-f-x-y in-mono
    by blast
  hence  $\forall x' \in r \text{ `` } \{x\}. f x' = y$ 
    using invar-f x-in-dom-and-f-x-y
    unfolding is-symmetry.simps
    by metis
  moreover have  $x \in A$ 
    using equiv-rel cons-subset equiv-class-self in-mono
      A-eq-img-singleton-r x-in-dom-and-f-x-y
    by metis
  ultimately have  $f \text{ ` } A = \{y\}$ 
    using A-eq-img-singleton-r
    by auto
  hence  $\pi_Q f A = y$ 
    unfolding  $\pi_Q$ .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:  $A = r \text{ `` } \{x\}$ 
    using A-in-dom-rel-r equiv-rel cons-subset equiv-class-self in-mono quotientE
    by metis
  ultimately have  $\forall x' \in A. (x, x') \in \text{Restr } r \text{ domain}_f$ 
    by blast
  hence  $\forall x' \in A. f x' = f x$ 
    using invar-f
    by fastforce
  hence  $f \text{ ` } A = \{f x\}$ 
    using x-in-A

```

```

    by blast
  hence  $\pi_Q f A = f x$ 
    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-eq-r-img-single-x
    unfolding 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  $y \in \text{minimizer } f \text{ domain}_f d Y x =$ 
     $(y \in Y \wedge (\forall y' \in Y. \text{Inf } (d x \text{ ` } (\text{preimg } f \text{ domain}_f y)) \leq \text{Inf } (d x \text{ ` } (\text{preimg } f$ 
 $\text{domain}_f y'))))$ 
    unfolding is-arg-min-def minimizer.simps arg-min-set.simps
    by auto

lemma rewr-singleton-set-system-union:
  fixes
     $Y :: 'x \text{ set set}$  and
     $X :: 'x \text{ set}$ 
  assumes  $Y \subseteq \text{singleton-set-system } X$ 
  shows
    singleton-set-union:  $x \in \bigcup Y \longleftrightarrow \{x\} \in Y$  and
    obtain-singleton:  $A \in \text{singleton-set-system } X \longleftrightarrow (\exists x \in X. A = \{x\})$ 
    unfolding singleton-set-system.simps
    using assms
    by auto

lemma union-inf:

```

```

fixes  $X :: \text{ereal set set}$ 
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 INF-lower2 Inf-lower Setcompr-eq-image
    by metis
  hence  $\forall x \in \bigcup X. ?inf \leq x$ 
    by simp
  hence le:  $?inf \leq \text{Inf } (\bigcup X)$ 
    using Inf-greatest
    by blast
  have  $\forall A \in X. \text{Inf } (\bigcup X) \leq \text{Inf } A$ 
    using 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 Quotient 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
   $\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A = \bigcup (\text{minimizer } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r$ 
     $C))$ 
     $(\text{distance-infimum}_{\mathcal{Q}} \ d) (\text{singleton-set-system } (\text{limit-set}_{\mathcal{Q}} \ A$ 
     $\text{UNIV})) \ A)$ 

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
   $\text{distance-}\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A =$ 
   $(\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A, \pi_{\mathcal{Q}} (\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \text{ UNIV}) \ A - \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$ 
   $\{\})$ 

```

Hadjibeyli and Wilson 2016 4.17

```

theorem (in result) invar-dr-simple-dist-imp-quotient-dr-winners:
  fixes
     $d :: ('a, 'v) \text{ Election Distance}$  and
     $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$  and
     $r :: ('a, 'v) \text{ Election rel}$  and
     $X :: ('a, 'v) \text{ Election set}$  and
     $A :: ('a, 'v) \text{ Election set}$ 
  assumes
    simple: simple  $r \ X \ d$  and

```



*closed-domain*: *closed-restricted-rel*  $r$   $X$  (*elections- $\mathcal{K}$*   $C$ ) **and**  
*invar-res*: *is-symmetry*  $(\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ UNIV)$  (*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**  $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A = \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$   
**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**  $\forall y'.$   
 $\bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r) \supseteq$   
 $\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'$   
**unfolding** *quotient-def*  
**by** *blast*  
**moreover have**  $\forall y'.$   
 $\bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r) \subseteq$   
 $\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'$   
**proof** (*intro allI subsetI*)  
**fix**  
 $Y' :: 'r \text{ set}$  **and**  
 $E :: ('a, 'v) \text{ Election}$   
**assume**  $E \in \bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ Y' // r)$   
**then obtain**  $B :: ('a, 'v) \text{ Election set}$  **where**  
 $E\text{-in-}B: E \in B$  **and**  
 $B \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ Y' // r$   
**by** *blast*  
**then obtain**  $E' :: ('a, 'v) \text{ Election}$  **where**  
 $B = r \text{ “ } \{E\}$  **and**  
 $\text{map-to-}Y': E' \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ Y'$   
**using** *quotientE*  
**by** *blast*  
**hence** *in-restr-rel*:  $(E', E) \in r \cap (\text{elections-}\mathcal{K} \ C) \times X$   
**using** *E-in-B equiv-rel*  
**unfolding** *preimg.simps equiv-def refl-on-def*  
**by** *blast*  
**hence**  $E \in \text{elections-}\mathcal{K} \ C$   
**using** *closed-domain*  
**unfolding** *closed-restricted-rel.simps restricted-rel.simps Image-def*

by *blast*  
 hence *rel-cons-els*:  $(E', E) \in \text{Restr } r \text{ (elections-}\mathcal{K} \text{ } C)$   
 using *in-restr-rel*  
 by *blast*  
 hence  $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) E = (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) E'$   
 using *invar-C*  
 unfolding *is-symmetry.simps*  
 by *blast*  
 hence  $(\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) E = Y'$   
 using *map-to-Y'*  
 by *simp*  
 thus  $E \in \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) (\text{elections-}\mathcal{K} \text{ } C) Y'$   
 unfolding *preimg.simps*  
 using *rel-cons-els*  
 by *blast*  
 qed  
 ultimately have *preimg-partition*:  $\forall y'.$   

$$\bigcup (\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) (\text{elections-}\mathcal{K} \text{ } C) y' // r) =$$

$$\text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) (\text{elections-}\mathcal{K} \text{ } C) y'$$
 by *blast*  
 have *quot-classes-subset*:  $(\text{elections-}\mathcal{K} \text{ } C) // r \subseteq X // r$   
 using *cons-subset*  
 unfolding *quotient-def*  
 by *blast*  
 obtain  $a :: ('a, 'v) \text{ Election where}$   
 $a\text{-in-}A: a \in A$  and  
 $a\text{-def-inf-dist}: \forall B \in X // r. \text{distance-infimum}_{\mathcal{Q}} d A B = \text{Inf } \{d a b \mid b. b \in B\}$   
 by *blast*  
 using *simple quot-class*  
 unfolding *simple.simps*  
 by *blast*  
 hence *inf-dist-preimg-sets*:  

$$\forall y' B. B \in \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) y' \longrightarrow$$

$$\text{distance-infimum}_{\mathcal{Q}} d A B = \text{Inf } \{d a b \mid b. b \in B\}$$
 using *preimg-img-imp-cls quot-classes-subset*  
 by *blast*  
 have *valid-res-eq*:  $\text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \text{ } a) \text{ UNIV}) =$   
 $\text{singleton-set-system } (\text{limit-set}_{\mathcal{Q}} A \text{ UNIV})$   
 using *invar-res a-in-A quot-class cons-subset equiv-rel limit-set-invar*  
 by *metis*  
 have *inf-le-iff*:  $\forall x.$   

$$(\forall y' \in \text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} \text{ } a) \text{ UNIV}).$$

$$\text{Inf } (d a ' \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) (\text{elections-}\mathcal{K} \text{ } C) \{x\})$$

$$\leq \text{Inf } (d a ' \text{preimg } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)) (\text{elections-}\mathcal{K} \text{ } C) y')$$

$$= (\forall y' \in \text{singleton-set-system } (\text{limit-set}_{\mathcal{Q}} A \text{ UNIV}).$$

$$\text{Inf } (\text{distance-infimum}_{\mathcal{Q}} d A ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)))$$

$$(\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) \{x\})$$

$$\leq \text{Inf } (\text{distance-infimum}_{\mathcal{Q}} d A ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \text{ } C)))$$

$$(\text{elections-}\mathcal{K}_{\mathcal{Q}} r C) y'))$$

**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}}.simps*  
**using** *preimg-invar closed-domain cons-subset equiv-rel invar-C*  
**by** *blast*  
**ultimately have**  

$$\forall y'. \text{Inf } (\text{distance-infimum}_{\mathcal{Q}} \ d \ A \ ' \text{preimg } (\pi_{\mathcal{Q}} (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))))$$

```

      (elections- $\mathcal{K}_Q$   $r$   $C$ )  $y'$ )
    = Inf {Inf { $d$   $a$   $b$  |  $b$ .  $b \in B$ }
      |  $B$ .  $B \in \text{preimg} (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y' // r$ }
    by simp
  thus ?thesis
    using valid-res-eq rewrite-inf-dist preimg-partition-dist
    by presburger
qed
from a-in-A
have  $\pi_Q (\text{fun}_E (\mathcal{R}_W \ d \ C)) \ A = \text{fun}_E (\mathcal{R}_W \ d \ C) \ a$ 
  using invar-dr equiv-rel quot-class pass-to-quotient invariance-is-congruence
  by blast
moreover have  $\forall \ x. x \in \text{fun}_E (\mathcal{R}_W \ d \ C) \ a \longleftrightarrow x \in \mathcal{R}_Q \ r \ d \ C \ A$ 
proof
  fix  $x :: 'r$ 
  have  $(x \in \text{fun}_E (\mathcal{R}_W \ d \ C) \ a) =$ 
     $(x \in \bigcup (\text{minimizer} (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ d$ 
       $(\text{singleton-set-system} (\text{limit-set} (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV})) \ a))$ 
    using  $\mathcal{R}_W$ -is-minimizer
    by metis
  also have ... =  $(\{x\} \in \text{minimizer} (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C)$ 
 $d$ 
     $(\text{singleton-set-system} (\text{limit-set} (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV})) \ a)$ 
    using singleton-set-union
    unfolding minimizer.simps arg-min-set.simps is-arg-min-def
    by auto
  also have ... =  $(\{x\} \in \text{singleton-set-system} (\text{limit-set} (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV})$ 
 $\wedge$ 
     $(\forall \ y' \in \text{singleton-set-system} (\text{limit-set} (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV}).$ 
     $\text{Inf} (d \ a \ \text{'preimg} (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ \{x\}) \leq$ 
     $\text{Inf} (d \ a \ \text{'preimg} (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)) (\text{elections-}\mathcal{K} \ C) \ y'))$ 
    using minimizer-helper
    by (metis (no-types, lifting))
  also have ... =  $(\{x\} \in \text{singleton-set-system} (\text{limit-set}_Q \ A \ \text{UNIV}) \wedge$ 
     $(\forall \ y' \in \text{singleton-set-system} (\text{limit-set}_Q \ A \ \text{UNIV}).$ 
     $\text{Inf} (\text{distance-infimum}_Q \ d \ A \ \text{'preimg} (\pi_Q (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)))$ 
     $(\text{elections-}\mathcal{K}_Q \ r \ C) \ \{x\})$ 
     $\leq \text{Inf} (\text{distance-infimum}_Q \ d \ A \ \text{'preimg} (\pi_Q (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C)))$ 
     $(\text{elections-}\mathcal{K}_Q \ r \ C) \ y'))$ 
    using valid-res-eq inf-le-iff
    by blast
  also have ... =
     $(\{x\} \in \text{minimizer} (\pi_Q (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_Q \ r \ C)$ 
     $(\text{distance-infimum}_Q \ d) (\text{singleton-set-system} (\text{limit-set}_Q \ A$ 
     $\text{UNIV})) \ A)$ 
    using minimizer-helper
    by (metis (no-types, lifting))
  also have ... =  $(x \in \bigcup (\text{minimizer} (\pi_Q (\text{elect-}r \circ \text{fun}_E (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_Q$ 
 $r \ C)$ 

```

```

(distance-infimumQ d) (singleton-set-system (limit-setQ A
UNIV)) A))
  using singleton-set-union
  unfolding minimizer.simps arg-min-set.simps is-arg-min-def
  by auto
  finally show (x ∈ funE (RW d C) a) = (x ∈ RQ r d C A)
  unfolding RQ.simps
  by blast
qed
ultimately show πQ (funE (RW d C)) A = RQ r d C A
  by blast
qed

theorem (in result) invar-dr-simple-dist-imp-quotient-dr:
  fixes
    d :: ('a, 'v) Election Distance and
    C :: ('a, 'v, 'r Result) Consensus-Class and
    r :: ('a, 'v) Election rel and
    X :: ('a, 'v) Election set and
    A :: ('a, 'v) Election set
  assumes
    simple: simple r X d and
    closed-domain: closed-restricted-rel r X (elections-K C) and
    invar-res: is-symmetry (λ E. limit-set (alternatives-E E) UNIV) (Invariance r)
  and
    invar-C: is-symmetry (elect-r ∘ funE (rule-K C)) (Invariance (Restr r (elections-K
C))) and
    invar-dr: is-symmetry (funE (RW d C)) (Invariance r) and
    quot-class: A ∈ X // r and
    equiv-rel: equiv X r and
    cons-subset: elections-K C ⊆ X
  shows πQ (funE (distance-R d C)) A = distance-RQ r d C A
proof -
  have ∀ E. funE (distance-R d C) E =
    (funE (RW d C) E, limit-set (alternatives-E E) UNIV - funE (RW d C)
E, {})
  by simp
  moreover have ∀ E ∈ A. funE (RW d C) E = πQ (funE (RW d C)) A
  using invar-dr invariance-is-congruence pass-to-quotient quot-class equiv-rel
  by blast
  moreover have πQ (funE (RW d C)) A = RQ r d C A
  using invar-dr-simple-dist-imp-quotient-dr-winners assms
  by blast
  moreover have
    ∀ E ∈ A. limit-set (alternatives-E E) UNIV = πQ (λ E. limit-set (alternatives-E
E) UNIV) A
  using invar-res invariance-is-congruence' pass-to-quotient quot-class equiv-rel
  by blast
  ultimately have all-eq:

```

```

    ∀ E ∈ A. funℰ (distance-ℛ d C) E =
      (ℛQ r d C A, πQ (λ E. limit-set (alternatives-ℰ E) UNIV) A - ℛQ r d C
A, {}))
    by fastforce
    hence {(ℛQ r d C A, πQ (λ E. limit-set (alternatives-ℰ E) UNIV) A - ℛQ r
d C A, {}))} ⊇
      funℰ (distance-ℛ d C) ‘ A
    by blast
    moreover have A ≠ {}
    using quot-class equiv-rel in-quotient-imp-non-empty
    by metis
    ultimately have single-img:
      {(ℛQ r d C A, πQ (λ E. limit-set (alternatives-ℰ E) UNIV) A - ℛQ r d C A,
{}))} =
      funℰ (distance-ℛ d C) ‘ A
    using empty-is-image subset-singletonD
    by (metis (no-types, lifting))
    moreover from this
    have card (funℰ (distance-ℛ d C) ‘ A) = 1
    using is-singleton-altdef is-singletonI
    by (metis (no-types, lifting))
    moreover from this single-img
    have the-inv (λ x. {x}) (funℰ (distance-ℛ d C) ‘ A) =
      (ℛQ r d C A, πQ (λ E. limit-set (alternatives-ℰ E) UNIV) A - ℛQ 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-ℛQ.simps
    using πQ.simps[of funℰ (distance-ℛ d C)] singleton-set.simps[of funℰ (distance-ℛ
d C) ‘ A]
    by presburger
qed

end

```

## 5.8 Result and Property Locale Code Generation

```

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

```

Lemmas stating the explicit instantiations of interpreted abstract functions from locales.

**lemma** *electoral-module-SCF-code-lemma*:

**fixes**  $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**shows**  $SCF\text{-result.electoral-module } m = (\forall A V p. \text{profile } V A p \longrightarrow \text{well-formed-SCF } A (m V A p))$   
**unfolding**  $SCF\text{-result.electoral-module.simps}$   
**by** *safe*

**lemma**  $\mathcal{R}_{\mathcal{W}}\text{-SCF-code-lemma:}$   
**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**  $SCF\text{-result.}\mathcal{R}_{\mathcal{W}} d K V A p = \text{arg-min-set } (\text{score } d K (A, V, p)) (\text{limit-set-SCF } A \text{ UNIV})$   
**unfolding**  $SCF\text{-result.}\mathcal{R}_{\mathcal{W}}.simps$   
**by** *safe*

**lemma**  $\text{distance-}\mathcal{R}\text{-SCF-code-lemma:}$   
**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**  $SCF\text{-result.distance-}\mathcal{R} d K V A p =$   
 $(SCF\text{-result.}\mathcal{R}_{\mathcal{W}} d K V A p, (\text{limit-set-SCF } A \text{ UNIV}) - SCF\text{-result.}\mathcal{R}_{\mathcal{W}} d$   
 $K V A p, \{\})$   
**unfolding**  $SCF\text{-result.distance-}\mathcal{R}.simps$   
**by** *safe*

**lemma**  $\mathcal{R}_{\mathcal{W}}\text{-std-SCF-code-lemma:}$   
**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**  $SCF\text{-result.}\mathcal{R}_{\mathcal{W}}\text{-std } d K V A p =$   
 $\text{arg-min-set } (\text{score-std } d K (A, V, p)) (\text{limit-set-SCF } A \text{ UNIV})$   
**unfolding**  $SCF\text{-result.}\mathcal{R}_{\mathcal{W}}\text{-std}.simps$   
**by** *safe*

**lemma**  $\text{distance-}\mathcal{R}\text{-std-SCF-code-lemma:}$   
**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-R-std d K V A p =
  (SCF-result.RW-std d K V A p, (limit-set-SCF A UNIV) - SCF-result.RW-std
d K V A p, {})
unfolding SCF-result.distance-R-std.simps
by safe

lemma anonymity-SCF-code-lemma:
shows SCF-result.anonymity =
  (λ m::('a, 'v, 'a Result) Electoral-Module).
    SCF-result.electoral-module m ∧
    (∀ A V p π::('v ⇒ 'v).
      bij π ⟶ (let (A', V', q) = (rename π (A, V, p)) in
        finite-profile V A p ∧ finite-profile V' A' q ⟶ m V A p = m V' A' q)))
unfolding SCF-result.anonymity-def
by simp

```

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

```

declare [[lc-add SCF-result.electoral-module electoral-module-SCF-code-lemma]]
declare [[lc-add SCF-result.RW RW-SCF-code-lemma]]
declare [[lc-add SCF-result.RW-std RW-std-SCF-code-lemma]]
declare [[lc-add SCF-result.distance-R distance-R-SCF-code-lemma]]
declare [[lc-add SCF-result.distance-R-std distance-R-std-SCF-code-lemma]]
declare [[lc-add SCF-result.anonymity anonymity-SCF-code-lemma]]

```

Constant aliases to use when exporting code instead of the interpreted functions

```

definition RW-SCF-code = SCF-result.RW
definition RW-std-SCF-code = SCF-result.RW-std
definition distance-R-SCF-code = SCF-result.distance-R
definition distance-R-std-SCF-code = SCF-result.distance-R-std
definition electoral-module-SCF-code = SCF-result.electoral-module
definition anonymity-SCF-code = SCF-result.anonymity

```

```

setup Locale-Code.close-block

```

```

end

```

## 5.9 Drop Module

```

theory Drop-Module
imports Component-Types/Electoral-Module
        Component-Types/Social-Choice-Types/Result

```



**begin**

This is a family of electoral modules. For a natural number  $n$  and a lexicon (linear order)  $r$  of all alternatives, the according drop module rejects the lexicographically first  $n$  alternatives (from  $A$ ) and defers the rest. It is primarily used as counterpart to the pass module in a parallel composition, in order to segment the alternatives into two groups.

### 5.9.1 Definition

**fun** *drop-module* :: *nat*  $\Rightarrow$  *'a Preference-Relation*  $\Rightarrow$  (*'a, 'v, 'a Result*) *Electoral-Module*  
**where**

*drop-module*  $n$   $r$   $V$   $A$   $p$  =  
 ( $\{\}$ ,  
 $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\}$ ,  
 $\{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\}$ )

### 5.9.2 Soundness

**theorem** *drop-mod-sound[simp]*:

**fixes**

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

$n :: \text{nat}$

**shows** *SCF-result.electoral-module* (*drop-module*  $n$   $r$ )

**proof** (*unfold SCF-result.electoral-module.simps, safe*)

**fix**

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

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

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

**assume** *profile*  $V$   $A$   $p$

**let**  $?mod = \text{drop-module } n \ r$

**have**  $\forall a \in A. a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x \leq n\} \vee$   
 $a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x > n\}$

**by** *auto*

**hence**  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\} \cup \{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\} = A$

**by** *blast*

**hence** *set-partition: set-equals-partition*  $A$  (*drop-module*  $n$   $r$   $V$   $A$   $p$ )

**by** *simp*

**have**  $\forall a \in A.$

$\neg (a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x \leq n\} \wedge$   
 $a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x > n\})$

**by** *simp*

**hence**  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\} \cap \{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\} = \{\}$

**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**  $\forall a \in A. a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x > n\} \vee$   
 $a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x \leq n\}$   
**using** *CollectI not-less*  
**by** *metis*  
**hence**  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\} \cup \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\} = A$   
**by** *blast*  
**hence** *set-equals-partition A* (*pass-module n r V A p*)  
**by** *simp*  
**moreover have**  
 $\forall a \in A.$   
 $\neg (a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x > n\} \wedge$   
 $a \in \{x \in A. \text{rank } (\text{limit } A \ r) \ x \leq n\})$   
**by** *simp*  
**hence**  $\{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\} \cap \{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\} = \{\}$   
**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
    g0-n:  $n > 0$ 
  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 g0-n leD mem-Collect-eq above-rank calculation
    unfolding One-nat-def
    by (metis (no-types, lifting))
  hence reject (pass-module  $n$   $r$ )  $V$   $A$   $p \neq A$ 
    by simp
  thus  $a \in \{\}$ 
    using rej-pass-A
    by simp
qed

```

### 5.10.4 Non-Electing

The pass module is non-electing.

**theorem** *pass-mod-non-electing[simp]*:

```

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

**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 ≤ 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 (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  $w$  where w-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-unique a-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 linear-order r
  shows defers 2 (pass-module 2 r)
proof (unfold defers-def, safe)
  show SCF-result.electoral-module (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
    min-card-two: 2 ≤ card A and
    fin-A: finite A and
    prof-A: profile V A p
  from min-card-two
  have not-empty-A: A ≠ {}
    by auto
  moreover have limit-A-order: linear-order-on A (limit A r)
    using limit-presv-lin-ord assms
    by auto
  ultimately obtain a where
    above (limit A r) a = {a}
    using above-one min-card-two fin-A prof-A
    by blast
  hence  $\forall b \in A. \text{let } q = \text{limit } A \ r \text{ in } (b \preceq_q a)$ 
    using limit-A-order pref-imp-in-above empty-iff lin-ord-imp-connex
    by metis
  hence a-best:  $\forall b \in A. (b, a) \in \text{limit } A \ r$ 
    using insert-iff insert-subset above-presv-limit assms
    by metis
  unfolding connex-def
  by metis
  hence a-best:  $\forall b \in A. (b, a) \in \text{limit } A \ r$ 

```



by *simp*  
 hence *a-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-in-defer*:  $a \in \text{defer} (\text{pass-module } 2 \ r) \ V \ A \ p$   
 by *simp*  
 have *finite*  $(A - \{a\})$   
 using *fin-A*  
 by *simp*  
 moreover have *A-not-only-a*:  $A - \{a\} \neq \{\}$   
 using *Diff-empty Diff-idemp Diff-insert0 not-empty-A insert-Diff finite.emptyI*  
     *card.insert-remove card.empty min-card-two Suc-n-not-le-n numeral-2-eq-2*  
 by *metis*  
 moreover have *limit-A-without-a-order*:  
     *linear-order-on*  $(A - \{a\})$   $(\text{limit } (A - \{a\}) \ r)$   
 using *limit-presv-lin-ord assms top-greatest*  
 by *blast*  
 ultimately obtain *b* where  
     *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-in-limit*:  $\forall c \in A - \{a\}. (c, b) \in \text{limit } (A - \{a\}) \ r$   
 by *simp*  
 hence *b-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 *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-above-b*:  $b \in \text{above} (\text{limit } A \ r) \ b$   
 using *b b-best above-presv-limit mem-Collect-eq assms insert-subset*  
 unfolding *above-def*  
 by *metis*  
 ultimately have *above-b-eq-ab*:  $\text{above} (\text{limit } A \ r) \ b = \{a, b\}$   
 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
  unfolding SCF-result.electoral-module.simps
  by simp
```

```
lemma elect-mod-only-voters: voters-determine-election elect-module
  unfolding voters-determine-election.simps
  by simp
```

### 5.11.3 Electing

```
theorem elect-mod-electing[simp]: electing elect-module
  unfolding electing-def
  by simp
```

```
end
```

## 5.12 Plurality Module

```
theory Plurality-Module
  imports Component-Types/Elimination-Module
begin
```

The plurality module implements the plurality voting rule. The plurality rule elects all modules with the maximum amount of top preferences among

all alternatives, and rejects all the other alternatives. It is electing and induces the classical plurality (voting) rule from social-choice theory.

### 5.12.1 Definition

**fun** *plurality-score* :: ('a, 'v) *Evaluation-Function* **where**  
*plurality-score* V x A p = *win-count* V p x

**fun** *plurality* :: ('a, 'v, 'a *Result*) *Electoral-Module* **where**  
*plurality* V A p = *max-eliminator* *plurality-score* V A p

**fun** *plurality'* :: ('a, 'v, 'a *Result*) *Electoral-Module* **where**  
*plurality'* V A p =  
 ({},  
 {a ∈ 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})

**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-elim-equiv*:

**fixes**  
 A :: 'a set **and**  
 V :: 'v set **and**  
 p :: ('a, 'v) *Profile*  
**assumes**  
 non-empty-A: A ≠ {} **and**  
 fin-A: finite A **and**  
 prof: *profile* V A p  
**shows** *plurality* V A p = *plurality'* V A p  
**proof** (*unfold plurality.simps plurality'.simps plurality-score.simps, standard*)  
**have** *fst* (*max-eliminator* (λ V x A p. *win-count* V p x) V A p) = {}  
**by** *simp*  
**also have** ... = *fst* ({},  
 {a ∈ A. ∃ b ∈ A. *win-count* V p a < *win-count* V p b},  
 {a ∈ A. ∀ b ∈ A. *win-count* V p b ≤ *win-count* V p a})  
**by** *simp*  
**finally show**  
*fst* (*max-eliminator* (λ V x A p. *win-count* V p x) V A p) =  
*fst* ({},  
 {a ∈ A. ∃ b ∈ A. *win-count* V p a < *win-count* V p b},

```

      {a ∈ A. ∀ b ∈ A. win-count V p b ≤ win-count V p a})
    by simp
  next
    let ?no-max = {a ∈ A. win-count V p a < Max {win-count V p x | x. x ∈ A}}
    = A
    have ?no-max ⇒ {win-count V p x | x. x ∈ A} ≠ {}
      using non-empty-A
    by blast
  moreover have finite-winners: finite {win-count V p x | x. x ∈ A}
    using fin-A
  by simp
  ultimately have exists-max: ?no-max ⇒ False
    using Max-in
  by fastforce
  have rej-eq:
    reject-r (max-eliminator (λ V b A p. win-count V p b) V A p) =
      {a ∈ A. ∃ x ∈ A. win-count V p a < win-count V p x}
  proof (unfold max-eliminator.simps less-eliminator.simps elimination-module.simps
    elimination-set.simps, safe)
    fix a :: 'a
    assume
      a ∈ reject-r
      (if {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈ A}} ≠ A
        then ({}, {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}),
        A - {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}))
      else ({}, {}, A))
    moreover have A ≠ {b ∈ A. win-count V p b < Max {win-count V p x | x. x
∈ A}}
      using exists-max
    by metis
    ultimately have a ∈ {b ∈ A. win-count V p b < Max {win-count V p x | x. x
∈ A}}
      by force
    thus a ∈ A
      by fastforce
  next
    fix a :: 'a
    assume
      reject-a:
      a ∈ reject-r
      (if {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈ A}} ≠ A
        then ({}, {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}),
        A - {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}))
      else ({}, {}, A))
    hence elect-nonempty:

```

$\{b \in A. \text{win-count } V p b < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\} \neq A$   
 by *fastforce*  
**obtain**  $f :: \text{enat} \Rightarrow \text{bool}$  **where**  
 $\text{all-winners-possible}: \forall x. f x = (\exists y. x = \text{win-count } V p y \wedge y \in A)$   
 by *fastforce*  
**hence** *finite* (*Collect*  $f$ )  
 using *finite-winners*  
 by *presburger*  
**hence** *max-winner-possible*:  $f (\text{Max } (\text{Collect } f))$   
 using *all-winners-possible* *Max-in elect-nonempty*  
 by *blast*  
**obtain**  $g :: 'a \Rightarrow \text{bool}$  **where**  
 $\text{all-losers-possible}: \forall x. g x = (x \in A \wedge \text{win-count } V p x < \text{Max } (\text{Collect } f))$   
 by *moura*  
**hence**  $a \in \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p a \mid a. a \in A\}\} \longrightarrow$   
 $a \in \text{Collect } g$   
 using *all-winners-possible*  
 by *presburger*  
**hence**  
 $a \in \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p a \mid a. a \in A\}\}$   
 $\longrightarrow (\exists x \in A. \text{win-count } V p a < \text{win-count } V p x)$   
 using *max-winner-possible* *all-losers-possible* *all-winners-possible* *mem-Collect-eq*  
 by (*metis* (*no-types*))  
**thus**  $\exists x \in A. \text{win-count } V p a < \text{win-count } V p x$   
 using *reject-a elect-nonempty*  
 by *simp*  
**next**  
**fix**  
 $a :: 'a$  **and**  
 $b :: 'a$   
**assume**  
 $b \in A$  **and**  
 $\text{win-count } V p a < \text{win-count } V p b$   
**moreover from this have**  $\exists a. \text{win-count } V p b = \text{win-count } V p a \wedge a \in A$   
 by *blast*  
**ultimately have**  $\text{win-count } V p a < \text{Max } \{\text{win-count } V p a \mid a. a \in A\}$   
 using *finite-winners* *Max-gr-iff*  
 by *fastforce*  
**moreover assume**  $a \in A$   
**ultimately have**  $\{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\} \neq A$   
 $\longrightarrow a \in \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\}$   
 by *force*  
**moreover have**  $\{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\} = A \longrightarrow a \in \{\}$   
 using *exists-max*  
 by *metis*  
**ultimately show**  
 $a \in \text{reject-}r$

```

      (if {a ∈ A. win-count V p a < Max {win-count V p x | x. x ∈ A}} ≠ A
        then ({}, {a ∈ A. win-count V p a < Max {win-count V p x | x. x ∈
A}}),
          A - {a ∈ A. win-count V p a < Max {win-count V p x | x. x ∈
A}}))
        else ({}, {}, A))
    by simp
  qed
  have
    defer-r (max-eliminator (λ V b A p. win-count V p b) V A p) =
      {a ∈ A. ∀ b ∈ A. win-count V p b ≤ win-count V p a}
  proof (unfold max-eliminator.simps less-eliminator.simps elimination-module.simps
    elimination-set.simps, safe)
    fix a :: 'a
    assume
      a ∈ defer-r
        (if {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈ A}} ≠ A
          then ({}, {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}),
            A - {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}))
        else ({}, {}, A))
    moreover have A ≠ {b ∈ A. win-count V p b < Max {win-count V p x | x. x
∈ A}}
    using exists-max
    by metis
    ultimately have a ∈ A - {b ∈ A. win-count V p b < Max {win-count V p x
| x. x ∈ A}}
    by force
    thus a ∈ A
    by fastforce
  next
  fix
    a :: 'a and
    b :: 'a
  assume b ∈ A
  hence win-count V p b ∈ {win-count V p x | x. x ∈ A}
  by blast
  hence win-count V p b ≤ Max {win-count V p x | x. x ∈ A}
  using fin-A
  by simp
  moreover assume
    a ∈ defer-r
      (if {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈ A}} ≠ A
        then ({}, {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}),
          A - {b ∈ A. win-count V p b < Max {win-count V p x | x. x ∈
A}}))
        else ({}, {}, A))

```

**moreover have**  $\{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\} \neq A$   
**using** *exists-max*  
**by** *metis*  
**ultimately have**  $\neg \text{win-count } V p a < \text{win-count } V p b$   
**using** *dual-order.strict-trans1*  
**by** *force*  
**thus**  $\text{win-count } V p b \leq \text{win-count } V p a$   
**using** *linorder-le-less-linear*  
**by** *metis*  
**next**  
**fix**  $a :: 'a$   
**assume**  
 $a\text{-in-}A: a \in A$  **and**  
 $\text{win-count-lt-}b: \forall b \in A. \text{win-count } V p b \leq \text{win-count } V p a$   
**then obtain**  $f :: \text{enat} \Rightarrow 'a$  **where**  
 $\forall x. a \in A \wedge f x \in A$   
 $\wedge (\neg (\forall b. x = \text{win-count } V p b \longrightarrow b \notin A) \longrightarrow \text{win-count } V p (f x) = x)$   
**by** *moura*  
**moreover from this have**  
 $f (\text{Max } \{\text{win-count } V p x \mid x. x \in A\}) \in A$   
 $\longrightarrow \text{Max } \{\text{win-count } V p x \mid x. x \in A\} \leq \text{win-count } V p a$   
**using** *Max-in finite-winners win-count-lt-b*  
**by** *fastforce*  
**ultimately show**  
 $a \in \text{defer-}r$   
 $(\text{if } \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\} \neq A$   
 $\text{then } (\{\}, \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\},$   
 $A\})$   
 $A - \{a \in A. \text{win-count } V p a < \text{Max } \{\text{win-count } V p x \mid x. x \in A\}\})$   
 $\text{else } (\{\}, \{\}, A))$   
**by** *force*  
**qed**  
**thus**  $\text{snd } (\text{max-eliminator } (\lambda V b A p. \text{win-count } V p b) V A p) =$   
 $\text{snd } (\{\},$   
 $\{a \in A. \exists b \in A. \text{win-count } V p a < \text{win-count } V p b\},$   
 $\{a \in A. \forall b \in A. \text{win-count } V p b \leq \text{win-count } V p a\})$   
**using** *snd-conv rej-eq prod.exhaust-sel*  
**by** *(metis (no-types, lifting))*  
**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**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**have** *disjoint3* (  
 $\{\},$   
 $\{a \in A. \exists a' \in A. \text{win-count } V \text{ } p \text{ } a < \text{win-count } V \text{ } p \text{ } a'\},$   
 $\{a \in A. \forall a' \in A. \text{win-count } V \text{ } p \text{ } a' \leq \text{win-count } V \text{ } p \text{ } a\})$   
**by** *auto*  
**moreover** **have**  
 $\{a \in A. \exists x \in A. \text{win-count } V \text{ } p \text{ } a < \text{win-count } V \text{ } p \text{ } x\} \cup$   
 $\{a \in A. \forall x \in A. \text{win-count } V \text{ } p \text{ } x \leq \text{win-count } V \text{ } p \text{ } a\} = A$   
**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**  
 $p :: ('b, 'a) \text{ Profile}$  **and**  
 $p' :: ('b, 'a) \text{ Profile}$  **and**  
 $a :: 'b$   
**assume**  
 $\forall v \in V. p \text{ } v = p' \text{ } v$  **and**  
 $a \in A$   
**hence** *finite V*  $\longrightarrow$   
 $\text{card } \{v \in V. \text{above } (p \text{ } v) \text{ } a = \{a\}\} = \text{card } \{v \in V. \text{above } (p' \text{ } v) \text{ } a = \{a\}\}$   
**using** *Collect-cong*  
**by** (*metis (no-types, lifting)*)  
**thus**  $\text{win-count } V \text{ } p \text{ } a = \text{win-count } V \text{ } p' \text{ } a$   
**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*

### 5.12.3 Non-Blocking

The plurality module is non-blocking.

**theorem** *plurality-mod-non-blocking[simp]*: *non-blocking plurality*

**unfolding** *plurality.simps*  
**using** *max-elim-non-blocking*  
**by** *metis*

#### 5.12.4 Non-Electing

The plurality module is non-electing.

**theorem** *plurality-non-electing[simp]: non-electing plurality*  
**using** *max-elim-non-electing*  
**unfolding** *plurality.simps non-electing-def*  
**by** *metis*

**theorem** *plurality'-non-electing[simp]: non-electing plurality'*  
**unfolding** *non-electing-def*  
**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**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $q :: ('a, 'v) \text{ Profile}$  **and**  
 $a :: 'a$

**assumes**

*defer-a:  $a \in \text{defer plurality } V A p$*  **and**  
*lift-a:  $\text{lifted } V A p q a$*

**shows**  $\text{defer plurality } V A q = \text{defer plurality } V A p \vee \text{defer plurality } V A q = \{a\}$

**proof** –

**have** *set-disj:  $\forall b c. (b::'a) \notin \{c\} \vee b = c$*   
**by** *blast*

**have** *lifted-winner:  $\forall b \in A. \forall i \in V. \text{above } (p i) b = \{b\} \longrightarrow (\text{above } (q i) b = \{b\} \vee \text{above } (q i) a = \{a\})$*   
**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**  $\forall \ b \in A - \{a\}.$   
 $\quad \forall \ i \in V. (\text{above } (q \ i) \ a = \{a\} \longrightarrow \text{above } (q \ i) \ b \neq \{b\})$   
 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*:  
 $\quad \forall \ b \in A - \{a\}.$   
 $\quad \forall \ i \in V. (\text{above } (q \ i) \ b = \{b\} \longrightarrow \text{above } (p \ i) \ b = \{b\})$   
 using *lifted-above-winner-other lift-a*  
 unfolding *Profile.lifted-def*  
 by *metis*  
**hence**  $\forall \ b \in A - \{a\}.$   
 $\quad \{i \in V. \text{above } (q \ i) \ b = \{b\}\} \subseteq \{i \in V. \text{above } (p \ i) \ b = \{b\}\}$   
 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  $\vee$  defer plurality V A q = {a}*  
**proof** (*cases*)  
 assume *win-count V p a = win-count V q a*  
**hence**  $\text{card } \{i \in V. \text{above } (p \ i) \ a = \{a\}\} = \text{card } \{i \in V. \text{above } (q \ i) \ a = \{a\}\}$   
 using *win-count.simps Profile.lifted-def enat.inject lift-a*  
 by (*metis (mono-tags, lifting)*)  
**moreover have** *finite {i ∈ V. above (q i) a = {a}}*  
 using *Collect-mem-eq Profile.lifted-def finite-Collect-conjI lift-a*  
 by (*metis (mono-tags)*)  
**ultimately have**  $\{i \in V. \text{above } (p \ i) \ a = \{a\}\} = \{i \in V. \text{above } (q \ i) \ a = \{a\}\}$   
 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**  
 $\quad \forall \ b \in A - \{a\}. \forall \ i \in V.$   
 $\quad (\text{above } (p \ i) \ b = \{b\} \longrightarrow (\text{above } (q \ i) \ b = \{b\} \vee \text{above } (q \ i) \ a = \{a\}))$   
 using *lifted-winner*  
 by *auto*  
**moreover have**  
 $\quad \forall \ b \in A - \{a\}. \forall \ i \in V. (\text{above } (p \ i) \ b = \{b\} \longrightarrow \text{above } (p \ i) \ a \neq \{a\})$   
**proof** (*intro ballI impI, safe*)  
**fix**  
 $\quad b :: 'a \text{ and}$

```

    i :: 'v
  assume
    b ∈ A and
    i ∈ V
  moreover from this have A-not-empty: A ≠ {}
    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 ≠ a and
    abv-b: above (p i) b = {b} and
    abv-a: above (p i) a = {a}
  ultimately show False
    using above-one-eq A-not-empty fin-A
    by (metis (no-types))
qed
ultimately have above-PtoQ:
  ∀ b ∈ A - {a}. ∀ i ∈ V. (above (p i) b = {b} ⟶ above (q i) b = {b})
  by simp
hence ∀ b ∈ A.
  card {i ∈ V. above (p i) b = {b}} =
  card {i ∈ V. above (q i) b = {b}}
proof (safe)
  fix b :: 'a
  assume
    above-c: ∀ c ∈ A - {a}. ∀ i ∈ V. above (p i) c = {c} ⟶ above (q i) c =
{c} and
    b-in-A: b ∈ A
  show card {i ∈ V. above (p i) b = {b}} =
    card {i ∈ V. above (q i) b = {b}}
    using DiffI b-in-A set-disj above-PtoQ above-QtoP above-pq
    by (metis (no-types, lifting))
qed
hence {b ∈ A. ∀ c ∈ A. win-count V p c ≤ win-count V p b} =
  {b ∈ A. ∀ c ∈ A. win-count V q c ≤ win-count V q b}
  by auto
hence defer plurality' V A q = defer plurality' V A p ∨ defer plurality' V A q
= {a}
  by simp
hence defer plurality V A q = defer plurality V A p ∨ defer plurality V A q =
{a}
  using plurality-mod-elim-equiv empty-not-insert insert-absorb lift-a
  unfolding Profile.lifted-def
  by (metis (no-types, opaque-lifting))
thus ?thesis
  by simp
next

```

**assume**  $\text{win-count } V \ p \ a \neq \text{win-count } V \ q \ a$   
**hence**  $\text{strict-less: win-count } V \ p \ a < \text{win-count } V \ q \ a$   
**using**  $\text{win-count-a}$   
**by**  $\text{simp}$   
**have**  $a \in \text{defer plurality } V \ A \ p$   
**using**  $\text{defer-a plurality.elims}$   
**by**  $(\text{metis (no-types)})$   
**moreover have**  $\text{non-empty-A: } A \neq \{\}$   
**using**  $\text{lift-a equals0D equiv-prof-except-a-def lifted-imp-equiv-prof-except-a}$   
**by**  $\text{metis}$   
**moreover have**  $\text{fin-A: finite-profile } V \ A \ p$   
**using**  $\text{lift-a}$   
**unfolding**  $\text{Profile.lifted-def}$   
**by**  $\text{simp}$   
**ultimately have**  $a \in \text{defer plurality}' V \ A \ p$   
**using**  $\text{plurality-mod-elim-equiv}$   
**by**  $\text{metis}$   
**hence**  $a \text{-in-win-p: } a \in \{b \in A. \forall c \in A. \text{win-count } V \ p \ c \leq \text{win-count } V \ p \ b\}$   
**by**  $\text{simp}$   
**hence**  $\forall b \in A. \text{win-count } V \ p \ b \leq \text{win-count } V \ p \ a$   
**by**  $\text{simp}$   
**hence**  $\text{less: } \forall b \in A - \{a\}. \text{win-count } V \ q \ b < \text{win-count } V \ q \ a$   
**using**  $\text{DiffD1 antisym dual-order.trans not-le-imp-less win-count-a strict-less win-count-other}$   
**by**  $\text{metis}$   
**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**  $\text{lift-a not-le}$   
**unfolding**  $\text{Profile.lifted-def}$   
**by**  $\text{metis}$   
**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**  $\text{blast}$   
**hence**  $\forall b \in A - \{a\}. b \notin \text{defer plurality}' V \ A \ q$   
**by**  $\text{simp}$   
**hence**  $\forall b \in A - \{a\}. b \notin \text{defer plurality } V \ A \ q$   
**using**  $\text{lift-a non-empty-A plurality-mod-elim-equiv}$   
**unfolding**  $\text{Profile.lifted-def}$   
**by**  $(\text{metis (no-types, lifting)})$   
**hence**  $\forall b \in A - \{a\}. b \notin \text{defer plurality } V \ A \ q$   
**by**  $\text{simp}$   
**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**  $\text{less less-imp-le}$   
**by**  $\text{metis}$   
**moreover have**  $\text{win-count } V \ q \ a \leq \text{win-count } V \ q \ a$   
**by**  $\text{simp}$   
**ultimately have**  $\forall b \in A. \text{win-count } V \ q \ b \leq \text{win-count } V \ q \ a$   
**by**  $\text{auto}$

```

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-elim-equiv non-empty-A fin-A lift-a non-empty-A
  unfolding Profile.lifted-def
  by (metis (no-types))
thus ?thesis
  by simp
qed
moreover have  $\text{defer plurality} \ V \ A \ q \subseteq A$ 
  by simp
ultimately show ?thesis
  by blast
qed
qed

```

The plurality rule is invariant-monotone.

```

theorem plurality-mod-def-inv-mono[simp]: defer-invariant-monotonicity plurality
proof (unfold defer-invariant-monotonicity-def, intro conjI impI allI)
  show SCF-result.electoral-module plurality
    using plurality-sound
    by metis
next
  show non-electing plurality
    by simp
next
  fix
     $A :: 'b \text{ set}$  and
     $V :: 'a \text{ set}$  and
     $p :: ('b, 'a) \text{ Profile}$  and
     $q :: ('b, 'a) \text{ Profile}$  and
     $a :: 'b$ 
    assume  $a \in \text{defer plurality} \ V \ A \ p \wedge \text{Profile.lifted } V \ A \ p \ q \ a$ 
    hence  $\text{defer plurality} \ V \ A \ q = \text{defer plurality} \ V \ A \ p \vee \text{defer plurality} \ V \ A \ q = \{a\}$ 
    using plurality-def-inv-mono-alts
    by metis
    thus  $\text{defer plurality} \ V \ A \ q = \text{defer plurality} \ V \ A \ p \vee \text{defer plurality} \ V \ A \ q = \{a\}$ 
    by simp
qed
end

```

## 5.13 Borda Module

```
theory Borda-Module
imports Component-Types/Elimination-Module
begin
```

This is the Borda module used by the Borda rule. The Borda rule is a voting rule, where on each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

### 5.13.1 Definition

```
fun borda-score :: ('a, 'v) Evaluation-Function where
  borda-score V x A p = ( $\sum$  y  $\in$  A. (prefer-count V p x y))

fun borda :: ('a, 'v, 'a Result) Electoral-Module where
  borda V A p = max-eliminator borda-score V A p
```

### 5.13.2 Soundness

```
theorem borda-sound: SCF-result.electoral-module borda
unfolding borda.simps
using max-elim-sound
by metis
```

### 5.13.3 Non-Blocking

The Borda module is non-blocking.

```
theorem borda-mod-non-blocking[simp]: non-blocking borda
unfolding borda.simps
using max-elim-non-blocking
by metis
```

### 5.13.4 Non-Electing

The Borda module is non-electing.

```
theorem borda-mod-non-electing[simp]: non-electing borda
using max-elim-non-electing
unfolding borda.simps non-electing-def
by metis
```

end

## 5.14 Condorcet Module

```
theory Condorcet-Module
  imports Component-Types/Elimination-Module
begin
```

This is the Condorcet module used by the Condorcet (voting) rule. The Condorcet rule is a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

### 5.14.1 Definition

```
fun condorcet-score :: ('a, 'v) Evaluation-Function where
  condorcet-score V x A p =
    (if (condorcet-winner V A p x) then 1 else 0)

fun condorcet :: ('a, 'v, 'a Result) Electoral-Module where
  condorcet V A p = (max-eliminator condorcet-score) V A p
```

### 5.14.2 Soundness

```
theorem condorcet-sound: SCF-result.electoral-module condorcet
  unfolding condorcet.simps
  using max-elim-sound
  by metis
```

### 5.14.3 Property

```
theorem condorcet-score-is-condorcet-rating: condorcet-rating condorcet-score
proof (unfold condorcet-rating-def, safe)
  fix
    A :: 'b set and
    V :: 'a set and
    p :: ('b, 'a) Profile and
    w :: 'b and
    l :: 'b
  assume
    c-win: condorcet-winner V A p w and
    l-neq-w: l  $\neq$  w
  have  $\neg$  condorcet-winner V A p l
```



```

    using cond-winner-unique-eq c-win l-neq-w
    by metis
  thus condorcet-score V l A p < condorcet-score V w A p
    using c-win zero-less-one
    unfolding condorcet-score.simps
    by (metis (full-types))
qed

theorem condorcet-is-dcc: defer-condorcet-consistency condorcet
proof (unfold defer-condorcet-consistency-def 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 Only participating voters impact the result

```
lemma voters-determine-copeland-score: voters-determine-evaluation copeland-score
proof (unfold copeland-score.simps voters-determine-evaluation.simps, safe)
```

```
  fix
```

```
    A :: 'b set and
```

```
    V :: 'a set and
```

```
    p :: ('b, 'a) Profile and
```

```
    p' :: ('b, 'a) Profile and
```

```
    a :: 'b
```

```
  assume
```

```
    ∀ v ∈ V. p v = p' v and
```

```
    a ∈ A
```

```
  hence ∀ x y. {v ∈ V. (x, y) ∈ p v} = {v ∈ V. (x, y) ∈ p' v}
```

```
    by blast
```

```
  hence ∀ x y. card {y ∈ A. wins V x p y} = card {y ∈ A. wins V x p' y} ∧
```

$\text{card } \{x \in A. \text{ wins } V x p y\} = \text{card } \{x \in A. \text{ wins } V x p' y\}$   
 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*

#### 5.15.4 Lemmas

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

**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 ∈ A - {w}. wins V w p a} ∪ {x ∈ {w}. wins V w p x} = {a ∈ A. wins V
w p a}
    using win-for-winner-not-reflexive
    by blast
    have finite-defeated: finite {a ∈ A - {w}. wins V w p a}
    using assms
    by simp
    have finite {a ∈ {w}. wins V w p a}
    by simp
    hence card ({a ∈ A - {w}. wins V w p a} ∪ {a ∈ {w}. wins V w p a}) =
        card {a ∈ A - {w}. wins V w p a} + card {a ∈ {w}. wins V w p a}
    using finite-defeated card-Un-disjoint
    by blast
    hence card {a ∈ A. wins V w p a} =
        card {a ∈ A - {w}. wins V w p a} + card {a ∈ {w}. wins V w p a}
    using union
    by simp
    thus ?thesis
    using winner-amount-one winner-amount-zero
    by linarith
qed

```

For a Condorcet winner  $w$ , we have: " $\text{card } \{y \in A . \text{wins } y \text{ } p \text{ } x = 0\}$ ".

**lemma** *cond-winner-imp-loss-count:*

```

fixes
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  w :: 'a
assumes condorcet-winner V A p w
shows card {a ∈ A. wins V a p w} = 0
using Collect-empty-eq card-eq-0-iff insert-Diff insert-iff wins-antisym assms
unfolding condorcet-winner.simps
by (metis (no-types, lifting))

```

Copeland score of a Condorcet winner.

**lemma** *cond-winner-imp-copeland-score:*

```

fixes
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  w :: 'a
assumes condorcet-winner V A p w
shows copeland-score V w A p = card A - 1
proof (unfold copeland-score.simps)
  have card {a ∈ A. wins V w p a} = card A - 1
  using cond-winner-imp-win-count assms
  by metis
  moreover have card {a ∈ A. wins V a p w} = 0

```

```

    using cond-winner-imp-loss-count assms
    by (metis (no-types))
  ultimately show
    enat (card {a ∈ A. wins V w p a} - card {a ∈ A. wins V a p w}) = enat (card
A - 1)
    by simp
qed

```

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

**lemma** *non-cond-winner-imp-win-count*:

```

  fixes
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    w :: 'a and
    l :: 'a
  assumes
    winner: condorcet-winner V A p w and
    loser: l ≠ w and
    l-in-A: l ∈ A
  shows card {a ∈ A . wins V l p a} ≤ card A - 2
proof -
  have wins V w p l
    using assms
    by auto
  hence ¬ wins V l p w
    using wins-antisym
    by simp
  moreover have ¬ wins V l p l
    using wins-irreflex
    by simp
  ultimately have wins-of-loser-eq-without-winner:
    {y ∈ A . wins V l p y} = {y ∈ A - {l, w} . wins V l p y}
    by blast
  have ∀ M f. finite M ⟶ card {x ∈ M . f x} ≤ card M
    by (simp add: card-mono)
  moreover have finite (A - {l, w})
    using finite-Diff winner
    by simp
  ultimately have card {y ∈ A - {l, w} . wins V l p y} ≤ card (A - {l, w})
    using winner
    by (metis (full-types))
  thus ?thesis
    using assms wins-of-loser-eq-without-winner
    by simp
qed

```

### 5.15.5 Property

The Copeland score is Condorcet rating.

**theorem** *copeland-score-is-cr: condorcet-rating copeland-score*

**proof** (*unfold condorcet-rating-def, unfold copeland-score.simps, safe*)

**fix**

$A :: 'b$  set **and**

$V :: 'v$  set **and**

$p :: ('b, 'v)$  Profile **and**

$w :: 'b$  **and**

$l :: 'b$

**assume**

*winner: condorcet-winner*  $V$   $A$   $p$   $w$  **and**

*l-in-A:  $l \in A$*  **and**

*l-neq-w:  $l \neq w$*

**hence**  $\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} \leq \text{card } A - 2$

**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**

$\text{enat } (\text{card } \{y \in A. \text{wins } V \ l \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ l\}) <$

$\text{enat } (\text{card } \{y \in A. \text{wins } V \ w \ p \ y\} - \text{card } \{y \in A. \text{wins } V \ y \ p \ w\})$

**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
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 :: 'a and
  l :: 'a
assumes
  prof: profile V A p and
  winner: condorcet-winner V A p w and
  l-in-A: l ∈ A and
  l-neq-w: l ≠ w
shows minimax-score V l A p ≤ prefer-count V p l w
proof (unfold minimax-score.simps, intro Min-le)
have finite V
using winner
by simp
moreover have ∀ E n. infinite E ⟶ (∃ e. ¬ e ≤ enat n ∧ e ∈ E)
using finite-enat-bounded
by blast
ultimately show finite {prefer-count V p l y | y. y ∈ A - {l}}
using pref-count-voter-set-card
by fastforce
next
have w ∈ A
using winner
by simp
thus prefer-count V p l w ∈ {prefer-count V p l y | y. y ∈ A - {l}}
```



using *l-neq-w*  
 by *blast*  
 qed

#### 5.16.4 Property

**theorem** *minimax-score-cond-rating: condorcet-rating minimax-score*

**proof** (*unfold condorcet-rating-def minimax-score.simps prefer-count.simps,*  
*safe, rule ccontr*)

**fix**  
*A* :: 'b set **and**  
*V* :: 'a set **and**  
*p* :: ('b, 'a) Profile **and**  
*w* :: 'b **and**  
*l* :: 'b  
**assume**  
*winner*: *condorcet-winner V A p w* **and**  
*l-in-A*: *l ∈ A* **and**  
*l-neq-w*: *l ≠ w* **and**  
*min-leq*:  
 $\neg \text{Min } \{ \text{if finite } V \text{ then enat } (\text{card } \{v \in V. \text{let } r = p \text{ v in } y \preceq_r l\}) \text{ else } \infty \mid$   
 $y. y \in A - \{l\}\}$   
 $< \text{Min } \{ \text{if finite } V \text{ then}$   
 $\text{enat } (\text{card } \{v \in V. \text{let } r = p \text{ v in } y \preceq_r w\}) \text{ else}$   
 $\infty \mid y. y \in A - \{w\}\}$   
**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 ∈ 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 {prefer-count V p w y | y . y ∈ 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* **where** *pref-count-eq-min*:

```

  prefer-count V p w n =
    Min {prefer-count V p w y | y . y ∈ A - {w}} and
  n-not-w: n ∈ A - {w}
  by metis
  hence n-in-A: n ∈ A
  using DiffE
  by metis
  have n-neq-w: n ≠ w
  using n-not-w
  by simp
  have w-in-A: w ∈ A
  using winner
  by simp
  have pref-count-n-w-ineq: prefer-count V p w n > prefer-count V p n w
  using n-not-w winner
  by auto
  have pref-count-l-w-n-ineq: prefer-count V p l w ≥ 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 ≥ prefer-count V p w l
  using n-in-A w-in-A l-in-A n-neq-w l-neq-w pref-count-sym winner
  unfolding condorcet-winner.simps
  by metis
  hence prefer-count V p l w > prefer-count V p w l
  using n-in-A w-in-A l-in-A n-neq-w l-neq-w pref-count-sym winner
    pref-count-n-w-ineq pref-count-l-w-n-ineq
  unfolding condorcet-winner.simps
  by auto
  hence wins V l p w
  by simp
  thus False
  using l-in-A-without-w wins-antisym winner
  unfolding condorcet-winner.simps
  by metis
qed

```

**theorem** *minimax-is-dcc: defer-condorcet-consistency minimax*

**proof** (unfold defer-condorcet-consistency-def 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
     $A :: 'b \text{ set}$  and
     $V :: 'a \text{ set}$  and
     $p :: ('b, 'a) \text{ Profile}$  and
     $w :: 'b$ 
  assume  $cwin-w: \text{condorcet-winner } V \ A \ p \ w$ 
  have  $\text{max-mmaxscore-dcc:}$ 
     $\text{defer-condorcet-consistency } ((\text{max-eliminator minimax-score})$ 
       $:: ('b, 'a, 'b \text{ Result}) \text{ Electoral-Module})$ 
  using  $\text{cr-eval-imp-dcc-max-elim minimax-score-cond-rating}$ 
  by  $\text{metis}$ 
  hence
     $\text{max-eliminator minimax-score } V \ A \ p =$ 
     $(\{\},$ 
       $A - \text{defer } (\text{max-eliminator minimax-score}) \ V \ A \ p,$ 
       $\{a \in A. \text{condorcet-winner } V \ A \ p \ a\})$ 
  using  $cwin-w$ 
  unfolding  $\text{defer-condorcet-consistency-def}$ 
  by  $\text{blast}$ 
  thus
     $\text{minimax } V \ A \ p =$ 
     $(\{\},$ 
       $A - \text{defer minimax } V \ A \ p,$ 
       $\{d \in A. \text{condorcet-winner } V \ A \ p \ d\})$ 
  by  $\text{simp}$ 
qed

end

```

## Chapter 6

# Compositional Structures

### 6.1 Drop And Pass Compatibility

```
theory Drop-And-Pass-Compatibility
  imports Basic-Modules/Drop-Module
           Basic-Modules/Pass-Module
begin
```

This is a collection of properties about the interplay and compatibility of both the drop module and the pass module.

#### 6.1.1 Properties

```
theorem drop-zero-mod-rej-zero[simp]:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes linear-order  $r$ 
  shows rejects 0 (drop-module 0 r)
proof (unfold rejects-def, safe)
  show SCF-result.electoral-module (drop-module 0 r)
    using assms drop-mod-sound
    by metis
next
  fix
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  assume
    fin-A: finite A and
    prof-A: profile V A p
  have connex UNIV r
    using assms lin-ord-imp-connex
    by auto
  hence connex: connex A (limit A r)
    using limit-presv-connex subset-UNIV
    by metis
```

```

have  $\forall B a. B \neq \{\} \vee (a::'a) \notin B$ 
  by simp
hence  $\forall a B. a \in A \wedge a \in B \longrightarrow \text{connex } B \ (limit\ A\ r) \longrightarrow$ 
   $\neg \text{card } (\text{above } (limit\ A\ r)\ a) \leq 0$ 
  using above-connex above-presv-limit card-eq-0-iff
  fin-A finite-subset le-0-eq assms
  by (metis (no-types))
hence  $\{a \in A. \text{card } (\text{above } (limit\ A\ r)\ a) \leq 0\} = \{\}$ 
  using connex
  by auto
hence  $\text{card } \{a \in A. \text{card } (\text{above } (limit\ A\ r)\ a) \leq 0\} = 0$ 
  using card.empty
  by (metis (full-types))
thus  $\text{card } (\text{reject } (\text{drop-module } 0\ r)\ V\ A\ p) = 0$ 
  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:  $n \leq \text{card } A$  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  $A\ r$ )  $\subseteq A \times A$ 
  unfolding linear-order-on-def partial-order-on-def preorder-on-def refl-on-def
  by simp
hence  $\forall a \in A. (\text{above } (limit\ A\ r)\ a) \subseteq A$ 
  unfolding above-def
  by auto
hence leq:  $\forall a \in A. \text{rank } (limit\ A\ r)\ a \leq \text{card } A$ 
  using fin-A
  by (simp add: card-mono)
have  $\forall a \in A. \{a\} \subseteq (\text{above } (limit\ A\ r)\ a)$ 

```

```

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  $\text{geq-1}: \forall a \in A. 1 \leq \text{rank } (\text{limit } A \ r) \ a$ 
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  $\text{inj}: \text{inj-on } (\text{rank } (\text{limit } A \ r)) \ A$ 
using fin-A inj-onI rank-unique lin-ord-limit
by metis
ultimately have  $\text{bij}: \text{bij-betw } (\text{rank } (\text{limit } A \ r)) \ A \ \{1 \ .. \ \text{card } A\}$ 
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  $\text{bij-inv}: \text{bij-betw } ?\text{inv-rank } \{1 \ .. \ \text{card } A\} \ A$ 
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  $\text{subset}: \{1 \ .. \ n\} \subseteq \{1 \ .. \ \text{card } A\}$ 
using card-n
by simp
ultimately have  $\text{card } (? \text{inv-rank } ' \{1 \ .. \ n\}) = n$ 
using numeral-One numeral-eq-iff semiring-norm(85) card-atLeastAtMost
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 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
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 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 geq-1
  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 set and
  V :: 'b set
have linear-order-on A (limit A r)
  using assms limit-presv-lin-ord
  by blast
hence profile V A ( $\lambda v. (limit A r)$ )
  using profile-def
  by blast
then obtain p :: ('a, 'b) Profile where
  profile V A p
  by blast
show  $\exists B \subseteq A. (\forall a \in B. indep\text{-of}\text{-alt } (drop\text{-module } n \ r) \ V \ A \ a \wedge$ 
   $(\forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (drop\text{-module } n \ r) \ V \ A \ p)) \wedge$ 
   $(\forall a \in A - B. indep\text{-of}\text{-alt } (pass\text{-module } n \ r) \ V \ A \ a \wedge$ 
   $(\forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (pass\text{-module } n \ r) \ V \ A \ p))$ 
proof
  have same-A:
     $\forall p \ q. (profile \ V \ A \ p \wedge profile \ V \ A \ q) \longrightarrow$ 
     $reject \ (drop\text{-module } n \ r) \ V \ A \ p = reject \ (drop\text{-module } n \ r) \ V \ A \ q$ 
  by auto
  let ?A = reject (drop-module n r) V A p
  have ?A  $\subseteq A$ 
  by auto
  moreover have  $\forall a \in ?A. indep\text{-of}\text{-alt } (drop\text{-module } n \ r) \ V \ A \ a$ 
  using assms drop-mod-sound
  unfolding drop-module.simps indep-of-alt-def
  by (metis (mono-tags, lifting))
  moreover have  $\forall a \in ?A. \forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (drop\text{-module } n$ 
   $r) \ V \ A \ p$ 
  by auto
  moreover have  $\forall a \in A - ?A. indep\text{-of}\text{-alt } (pass\text{-module } n \ r) \ V \ A \ a$ 
  using assms pass-mod-sound
  unfolding pass-module.simps indep-of-alt-def
  by metis
  moreover have  $\forall a \in A - ?A. \forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (pass\text{-module}$ 
   $n \ r) \ V \ A \ p$ 
  by auto
  ultimately show ?A  $\subseteq A \wedge$ 
     $(\forall a \in ?A. indep\text{-of}\text{-alt } (drop\text{-module } n \ r) \ V \ A \ a \wedge$ 
     $(\forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (drop\text{-module } n \ r) \ V \ A \ p)) \wedge$ 
     $(\forall a \in A - ?A. indep\text{-of}\text{-alt } (pass\text{-module } n \ r) \ V \ A \ a \wedge$ 
     $(\forall p. profile \ V \ A \ p \longrightarrow a \in reject \ (pass\text{-module } n \ r) \ V \ A \ p))$ 
  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 ( $\neg\downarrow$  50) **where**  
*m* $\downarrow$  == *revision-composition* m

### 6.2.2 Soundness

**theorem** *rev-comp-sound[simp]*:  
**fixes** m :: ('a, 'v, 'a Result) Electoral-Module  
**assumes** *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** *unity*:  
 $\forall$  A V p. *profile* V A p  $\longrightarrow$   
*set-equals-partition* A (*revision-composition* m V A p)  
**by** *simp*  
**have**  $\forall$  A V p. *profile* V A p  $\longrightarrow$  (A - *elect* m V A p)  $\cap$  *elect* m V A p = {}  
**by** *blast*  
**hence** *disjoint*:  
 $\forall$  A V p. *profile* V A p  $\longrightarrow$  *disjoint3* (*revision-composition* m V A p)  
**by** *simp*  
**from** *unity disjoint*  
**show** ?thesis

```

    unfolding SCF-result.electoral-module.simps
  by simp
qed

```

```

lemma voters-determine-rev-comp:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes voters-determine-election m
  shows voters-determine-election (revision-composition m)
  using assms
  unfolding voters-determine-election.simps revision-composition.simps
  by presburger

```

### 6.2.3 Composition Rules

An electoral module received by revision is never electing.

```

theorem rev-comp-non-electing[simp]:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes SCF-result.electoral-module m
  shows non-electing (m↓)
  using assms fstI rev-comp-sound revision-composition.simps
  using non-electing-def
  by metis

```

Revising an electing electoral module results in a non-blocking electoral module.

```

theorem rev-comp-non-blocking[simp]:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes electing m
  shows non-blocking (m↓)
proof (unfold non-blocking-def, safe)
  show SCF-result.electoral-module (m↓)
    using assms rev-comp-sound
    unfolding electing-def
    by (metis (no-types, lifting))
next
  fix
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    x :: 'a
  assume
    fin-A: finite A and
    prof-A: profile V A p and
    reject-A: reject (m↓) V A p = A and
    x-in-A: x ∈ A
  hence non-electing m
    using assms empty-iff Diff-disjoint Int-absorb2
    elect-in-alts prod.collapse prod.inject

```

```

    unfolding electing-def revision-composition.simps
    by (metis (no-types, lifting))
  thus  $x \in \{\}$ 
    using assms fin-A prof-A x-in-A
    unfolding electing-def non-electing-def
    by (metis (no-types, lifting))
qed

```

Revising an invariant monotone electoral module results in a defer-invariant-monotone electoral module.

```

theorem rev-comp-def-inv-mono[simp]:
  fixes  $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
  assumes invariant-monotonicity  $m$ 
  shows defer-invariant-monotonicity ( $m \downarrow$ )
proof (unfold defer-invariant-monotonicity-def, safe)
  show  $SCF\text{-result.electoral-module } (m \downarrow)$ 
    using assms rev-comp-sound
    unfolding invariant-monotonicity-def
    by metis
next
  show non-electing ( $m \downarrow$ )
    using assms rev-comp-non-electing
    unfolding invariant-monotonicity-def
    by simp
next
  fix
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  and
     $q :: ('a, 'v) \text{ Profile}$  and
     $a :: 'a$  and
     $x :: 'a$  and
     $x' :: 'a$ 
  assume
    rev-p-defer-a:  $a \in \text{defer } (m \downarrow) \ V \ A \ p$  and
    a-lifted: lifted  $V \ A \ p \ q \ a$  and
    rev-q-defer-x:  $x \in \text{defer } (m \downarrow) \ V \ A \ q$  and
    x-non-eq-a:  $x \neq a$  and
    rev-q-defer-x':  $x' \in \text{defer } (m \downarrow) \ V \ A \ q$ 
  from rev-p-defer-a
  have elect-a-in-p:  $a \in \text{elect } m \ V \ A \ p$ 
    by simp
  from rev-q-defer-x x-non-eq-a
  have elect-no-unique-a-in-q:  $\text{elect } m \ V \ A \ q \neq \{a\}$ 
    by force
  from assms
  have  $\text{elect } m \ V \ A \ q = \text{elect } m \ V \ A \ p$ 
    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
   $p :: ('a, 'v) \text{ Profile}$  and
   $q :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$  and
   $x :: 'a$  and
   $x' :: 'a$ 
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-defer- $x$ :  $x \in \text{defer } (m \downarrow) V A q$  and
  x-non-eq- $a$ :  $x \neq a$  and
  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  $\text{elect } m V A q \neq \{a\}$ 
  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
   $p :: ('a, 'v) \text{ Profile}$  and
   $q :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$  and
   $x :: 'a$  and
   $x' :: 'a$ 
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
   $p :: ('a, 'v) \ \text{Profile}$  and
   $q :: ('a, 'v) \ \text{Profile}$  and
   $a :: 'a$  and
   $x :: 'a$  and
   $x' :: 'a$ 
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  $\text{defer } (m \downarrow) \ V \ A \ q = \text{elect } m \ V \ A \ q$ 
  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.

$$\begin{aligned} \text{fun } \textit{sequential-composition} :: (&'a, 'v, 'a \textit{Result}) \textit{Electoral-Module} \Rightarrow \\ &(&'a, 'v, 'a \textit{Result}) \textit{Electoral-Module} \Rightarrow \\ &(&'a, 'v, 'a \textit{Result}) \textit{Electoral-Module} \textbf{ where} \\ \textit{sequential-composition } m \ n \ V \ A \ p = \\ &(\textit{let } \textit{new-A} = \textit{defer } m \ V \ A \ p; \\ &\quad \textit{new-p} = \textit{limit-profile } \textit{new-A} \ p \textit{ in } ( \\ &\quad (\textit{elect } m \ V \ A \ p) \cup (\textit{elect } n \ V \ \textit{new-A} \ \textit{new-p}), \\ &\quad (\textit{reject } m \ V \ A \ p) \cup (\textit{reject } n \ V \ \textit{new-A} \ \textit{new-p}), \\ &\quad \textit{defer } n \ V \ \textit{new-A} \ \textit{new-p})) \end{aligned}$$

**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))

398

$$\begin{aligned} & \text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \\ &= \text{elect } m \ V \ A \ p' \cup \text{elect } n \ V \ (\text{defer } m \ V \ A \ p') \ (\text{limit-profile } (\text{defer } m \ V \ A \\ & p') \ p') \\ & \text{using } \text{assms eq coincide-limit} \\ & \text{unfolding voters-determine-election.simps} \\ & \text{by metis} \\ & \text{moreover have} \\ & \text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \\ &= \text{reject } m \ V \ A \ p' \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p') \ (\text{limit-profile } (\text{defer } m \ V \ A \\ & p') \ p') \\ & \text{using } \text{assms eq coincide-limit} \\ & \text{unfolding voters-determine-election.simps} \\ & \text{by metis} \\ & \text{moreover have} \\ & \text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \\ &= \text{defer } n \ V \ (\text{defer } m \ V \ A \ p') \ (\text{limit-profile } (\text{defer } m \ V \ A \ p') \ p') \\ & \text{using } \text{assms eq coincide-limit} \\ & \text{unfolding voters-determine-election.simps} \\ & \text{by metis} \\ & \text{ultimately show } (m \triangleright n) \ V \ A \ p = (m \triangleright n) \ V \ A \ p' \\ & \text{unfolding sequential-composition.simps} \\ & \text{by metis} \\ & \text{qed} \end{aligned}$$

**lemma** *seq-comp-presv-disj*:

**fixes**

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

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

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

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

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

**assumes** *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*:

$\forall A' V' p' m' a.$

$(\text{profile } V' A' p' \wedge$

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

$(a :: 'a) \in \text{defer } m' V' A' p') \longrightarrow$

$a \in A'$

**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  $\cap$  reject m V A p = {}  $\wedge$ 
  elect m V A p  $\cap$  defer m V A p = {}  $\wedge$ 
  reject m V A p  $\cap$  defer m V A p = {}
  using prof module-m
  by (simp add: result-disj)
have reject n V (defer m V A p) (limit-profile (defer m V A p) p)  $\subseteq$  defer m V
A p
  using def-presv-prof reject-in-alts prof module-m module-n
  by metis
with disjoint-m module-m module-n prof
have elect-reject-diff: elect m V A p  $\cap$  reject n V ?new-A ?new-p = {}
  using disj-n
  by blast
from prof module-m module-n
have elec-n-in-def-m:
  elect n V (defer m V A p) (limit-profile (defer m V A p) p)  $\subseteq$  defer m V A p
  using def-presv-prof elect-in-alts
  by metis
have elect-defer-diff: elect m V A p  $\cap$  defer n V ?new-A ?new-p = {}
proof -
  obtain f :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
     $\forall B B'. (\exists a b. a \in B' \wedge b \in B \wedge a = b) =$ 
     $(f B B' \in B' \wedge (\exists a. a \in B \wedge f B B' = a))$ 
  using disjoint-iff
  by metis
  then obtain g :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
     $\forall B B'. (B \cap B' = \{\} \longrightarrow (\forall a b. a \in B \wedge b \in B' \longrightarrow a \neq b)) \wedge$ 
     $(B \cap B' \neq \{\} \longrightarrow f B B' \in B \wedge g B B' \in B' \wedge f B B' = g B B')$ 
  by auto
  thus ?thesis
  using defer-in-A disj-n module-n prof-def-lim prof
  by (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 ( $\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ ?\text{new-}A \ ?\text{new-}p$ )  $\cap$ 
      ( $\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ ?\text{new-}A \ ?\text{new-}p$ ) = {}
proof (safe)
  fix  $x :: 'a$ 
  assume
     $x \in \text{elect } m \ V \ A \ p$  and
     $x \in \text{reject } m \ V \ A \ p$ 
  hence  $x \in \text{elect } m \ V \ A \ p \cap \text{reject } m \ V \ A \ p$ 
  by simp
  thus  $x \in \{\}$ 
  using disj-n
  by simp
next
  fix  $x :: 'a$ 
  assume
     $x \in \text{elect } m \ V \ A \ p$  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 elect-reject-diff
  by blast
next
  fix  $x :: 'a$ 
  assume
     $x \in \text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$  and
     $x \in \text{reject } m \ V \ A \ p$ 
  thus  $x \in \{\}$ 
  using rej-intersect-new-elect-empty
  by blast
next
  fix  $x :: 'a$ 
  assume
     $x \in \text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$  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 disjoint-iff-not-equal module-n prof-def-lim result-disj prof
  by metis
qed
moreover have
  ( $\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ ?\text{new-}A \ ?\text{new-}p$ )  $\cap$  ( $\text{defer } n \ V \ ?\text{new-}A \ ?\text{new-}p$ ) = {}
  using Int-Un-distrib2 Un-empty elect-defer-diff module-n
  prof-def-lim result-disj prof
  by (metis (no-types))
moreover have
  ( $\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ ?\text{new-}A \ ?\text{new-}p$ )  $\cap$  ( $\text{defer } n \ V \ ?\text{new-}A \ ?\text{new-}p$ ) =
{}
proof (safe)
  fix  $x :: 'a$ 
  assume

```

```

    x-in-def:  $x \in \text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$  and
    x-in-rej:  $x \in \text{reject } m \ V \ A \ p$ 
from x-in-def
have  $x \in \text{defer } m \ V \ A \ p$ 
    using defer-in-A module-n prof-def-lim prof
    by blast
with x-in-rej
have  $x \in \text{reject } m \ V \ A \ p \cap \text{defer } m \ V \ A \ p$ 
    by fastforce
thus  $x \in \{\}$ 
    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-elec-or-def
    by fastforce
qed
ultimately have
    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
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
assumes module-m: SCF-result.electoral-module m and
    module-n: SCF-result.electoral-module n and
    prof: profile V A p
shows set-equals-partition A ((m  $\triangleright$  n) V A p)
proof –
let  $?new-A = \text{defer } m \ V \ A \ p$ 
let  $?new-p = \text{limit-profile } ?new-A \ p$ 
have 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  $\text{elect } n \ V \ ?new-A \ ?new-p \cup$ 

```

```

      reject n V ?new-A ?new-p  $\cup$ 
      defer n V ?new-A ?new-p = ?new-A
    using module-m module-n prof def-presv-prof result-presv-alts
    by metis
  hence (elect m V A p  $\cup$  elect n V ?new-A ?new-p)  $\cup$ 
        (reject m V A p  $\cup$  reject n V ?new-A ?new-p)  $\cup$ 
        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 :: ('a, 'v, 'a Result) Electoral-Module **and**

```

    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
    prof-A: profile V A p
have  $\forall$  r. well-formed-SCF (A::'a set) r =
    (disjoint3 r  $\wedge$  set-equals-partition A r)
by simp
thus well-formed-SCF A ((m  $\triangleright$  n) V A p)
using assms seq-comp-presv-disj seq-comp-presv-alts prof-A
by metis
qed

```

### 6.3.3 Lemmas

**lemma** seq-comp-decrease-only-defer:

```

fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
assumes
    module-m: SCF-result.electoral-module m and
    module-n: SCF-result.electoral-module n and
    prof: profile V A p and
    empty-defer: defer m V A p = {}
shows (m  $\triangleright$  n) V A p = m V A p
proof -
have  $\forall$  m' A' V' p'.
    (SCF-result.electoral-module m'  $\wedge$  profile V' A' p')  $\longrightarrow$ 
    profile V' (defer m' V' A' p') (limit-profile (defer m' V' A' p') p')
using def-presv-prof prof
by metis
hence prof-no-alt: profile V {} (limit-profile (defer m V A p) p)
using empty-defer prof module-m
by metis
show ?thesis
proof
have
    (elect m V A p)  $\cup$  (elect n V (defer m V A p) (limit-profile (defer m V A p)
    p)) =

```

```

      elect m V A p
    using elect-in-alts[of n V defer m V A p (limit-profile (defer m V A p) p)]
      empty-defer module-n prof prof-no-alt
    by auto
  thus elect (m ▷ n) V A p = elect m V A p
    using fst-conv
    unfolding sequential-composition.simps
    by metis
next
have rej-empty:
  ∀ m' V' p'.
    (SCF-result.electoral-module m'
     ∧ profile V' ({::'a set} p') → reject m' V' {} p' = {})
  using bot.extremum-uniqueI reject-in-alts
  by metis
have (reject m V A p, defer n V {} (limit-profile {} p)) = snd (m V A p)
  using bot.extremum-uniqueI defer-in-alts empty-defer
    module-n prod.collapse prof-no-alt
  by (metis (no-types))
thus snd ((m ▷ n) V A p) = snd (m V A p)
  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 :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    n-electing-m: non-electing m and
    def-one-m: defers 1 m and
    electing-n: electing n and
    f-prof: finite-profile V A p
  shows elect (m ▷ n) V A p = defer m V A p
proof (cases)
  assume A = {}
  with electing-n n-electing-m f-prof
  show ?thesis
    using bot.extremum-uniqueI defer-in-alts elect-in-alts seq-comp-sound
    unfolding electing-def non-electing-def
    by metis
next
  assume non-empty-A: A ≠ {}
  from n-electing-m f-prof

```

```

have ele: elect m V A p = {}
  unfolding non-electing-def
  by simp
from non-empty-A def-one-m f-prof finite
have def-card: card (defer m V A p) = 1
  unfolding defers-def
  by (simp add: Suc-leI card-gt-0-iff)
with n-electing-m f-prof
have def:  $\exists a \in A. \text{defer } m \text{ V } A \text{ } p = \{a\}$ 
  using card-1-singletonE defer-in-alts singletonI subsetCE
  unfolding non-electing-def
  by metis
from ele def n-electing-m
have rej:  $\exists a \in A. \text{reject } m \text{ V } A \text{ } p = A - \{a\}$ 
  using Diff-empty def-one-m f-prof reject-not-elec-or-def
  unfolding defers-def
  by metis
from ele rej def n-electing-m f-prof
have res-m:  $\exists a \in A. m \text{ V } A \text{ } p = (\{\}, A - \{a\}, \{a\})$ 
  using Diff-empty elect-rej-def-combination reject-not-elec-or-def
  unfolding non-electing-def
  by metis
hence  $\exists a \in A. \text{elect } (m \triangleright n) \text{ V } A \text{ } p = \text{elect } n \text{ V } \{a\} \text{ (limit-profile } \{a\} \text{ } p)$ 
  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  $\exists a \in A. \text{elect } (m \triangleright n) \text{ V } A \text{ } p = \{a\}$ 
  using electing-for-only-alt fst-conv def-presv-prof sup-bot.left-neutral
  unfolding non-electing-def sequential-composition.simps
  by metis
with def def-card electing-n n-electing-m f-prof res-m
show ?thesis
  using def-presv-prof electing-for-only-alt fst-conv sup-bot.left-neutral
  unfolding non-electing-def sequential-composition.simps
  by metis
qed

```

**lemma** *seq-comp-def-card-bounded*:

**fixes**

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

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

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

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

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

**assumes**

*SCF-result.electoral-module*  $m$  **and**

*SCF-result.electoral-module*  $n$  **and**

*finite-profile*  $V \text{ } A \text{ } p$

**shows**  $\text{card } (\text{defer } (m \triangleright n) \ V \ A \ p) \leq \text{card } (\text{defer } m \ V \ A \ p)$   
**using** *card-mono defer-in-alts assms def-presv-prof snd-conv finite-subset*  
**unfolding** *sequential-composition.simps*  
**by** *metis*

**lemma** *seq-comp-def-set-bounded:*

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**assumes**  
 $\text{SCF-result.electoral-module } m$  **and**  
 $\text{SCF-result.electoral-module } n$  **and**  
 $\text{profile } V \ A \ p$   
**shows**  $\text{defer } (m \triangleright n) \ V \ A \ p \subseteq \text{defer } m \ V \ A \ p$   
**using** *defer-in-alts assms snd-conv def-presv-prof*  
**unfolding** *sequential-composition.simps*  
**by** *metis*

**lemma** *seq-comp-defers-def-set:*

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**shows**  $\text{defer } (m \triangleright n) \ V \ A \ p = \text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$   
**using** *snd-conv*  
**unfolding** *sequential-composition.simps*  
**by** *metis*

**lemma** *seq-comp-def-then-elect-elec-set:*

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**shows**  $\text{elect } (m \triangleright n) \ V \ A \ p =$   
 $\text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \cup (\text{elect } m \ V \ A \ p)$   
**using** *Un-commute fst-conv*  
**unfolding** *sequential-composition.simps*  
**by** *metis*

**lemma** *seq-comp-elim-one-red-def-set:*

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$   
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$   
 $A :: 'a \text{ set and}$   
 $V :: 'v \text{ set and}$   
 $p :: ('a, 'v) \text{ Profile}$   
**assumes**  
 $SCF\text{-result.electoral-module } m \text{ and}$   
 $eliminates\ 1\ n \text{ and}$   
 $profile\ V\ A\ p \text{ and}$   
 $card\ (defer\ m\ V\ A\ p) > 1$   
**shows**  $defer\ (m \triangleright n)\ V\ A\ p \subset defer\ m\ V\ A\ p$   
**using**  $assms\ snd\ conv\ def\ presv\ prof\ single\ elim\ imp\ red\ def\ set$   
**unfolding**  $sequential\ composition.simps$   
**by**  $metis$

**lemma**  $seq\ comp\ def\ set\ trans$ :

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$   
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$   
 $A :: 'a \text{ set and}$   
 $V :: 'v \text{ set and}$   
 $p :: ('a, 'v) \text{ Profile and}$   
 $a :: 'a$   
**assumes**  
 $a \in (defer\ (m \triangleright n)\ V\ A\ p) \text{ and}$   
 $SCF\text{-result.electoral-module } m \wedge SCF\text{-result.electoral-module } n \text{ and}$   
 $profile\ V\ A\ p$   
**shows**  $a \in defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\ profile\ (defer\ m\ V\ A\ p)\ p) \wedge$   
 $a \in defer\ m\ V\ A\ p$   
**using**  $seq\ comp\ def\ set\ bounded\ assms\ in\ mono\ seq\ comp\ defers\ def\ set$   
**by**  $(metis\ (no\ types,\ opaque\ lifting))$

### 6.3.4 Composition Rules

The sequential composition preserves the non-blocking property.

**theorem**  $seq\ comp\ presv\ non\ blocking[simp]$ :

**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module and}$   
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**assumes**  
 $non\ blocking\ m$   
 $non\ blocking\ n$   
**shows**  $non\ blocking\ (m \triangleright n)$   
**proof** —  
**fix**  
 $A :: 'a \text{ set and}$   
 $V :: 'v \text{ set and}$   
 $p :: ('a, 'v) \text{ Profile}$



```

let ?input-sound =  $A \neq \{\}$   $\wedge$  finite-profile  $V A p$ 
from non-blocking-m
have ?input-sound  $\longrightarrow$  reject  $m V A p \neq A$ 
  unfolding non-blocking-def
  by simp
with non-blocking-m
have A-reject-diff: ?input-sound  $\longrightarrow A - \text{reject } m V A p \neq \{\}$ 
  using Diff-eq-empty-iff reject-in-alts subset-antisym
  unfolding non-blocking-def
  by metis
from non-blocking-m
have ?input-sound  $\longrightarrow$  well-formed-SCF  $A (m V A p)$ 
  unfolding SCF-result.electoral-module.simps non-blocking-def
  by simp
hence ?input-sound  $\longrightarrow$  elect  $m V A p \cup$  defer  $m V A p = A - \text{reject } m V A p$ 
  using non-blocking-m elec-and-def-not-rej
  unfolding non-blocking-def
  by metis
with A-reject-diff
have ?input-sound  $\longrightarrow$  elect  $m V A p \cup$  defer  $m V A p \neq \{\}$ 
  by simp
hence ?input-sound  $\longrightarrow$  (elect  $m V A p \neq \{\}$   $\vee$  defer  $m V A p \neq \{\}$ )
  by simp
with non-blocking-m non-blocking-n
show ?thesis
proof (unfold non-blocking-def)
  assume
    emod-reject-m:
      SCF-result.electoral-module  $m \wedge$ 
      ( $\forall A V p. A \neq \{\} \wedge$  finite  $A \wedge$  profile  $V A p \longrightarrow$  reject  $m V A p \neq A$ ) and
    emod-reject-n:
      SCF-result.electoral-module  $n \wedge$ 
      ( $\forall A V p. A \neq \{\} \wedge$  finite  $A \wedge$  profile  $V A p \longrightarrow$  reject  $n V A p \neq A$ )
  show
    SCF-result.electoral-module ( $m \triangleright n$ )  $\wedge$ 
    ( $\forall A V p. A \neq \{\} \wedge$  finite  $A \wedge$  profile  $V A p \longrightarrow$  reject ( $m \triangleright n$ )  $V A p \neq A$ )
  proof (safe)
    show SCF-result.electoral-module ( $m \triangleright n$ )
      using emod-reject-m emod-reject-n seq-comp-sound
      by metis
  next
  fix
     $A :: 'a$  set and
     $V :: 'v$  set and
     $p :: ('a, 'v)$  Profile and
     $x :: 'a$ 
  assume
    fin-A: finite  $A$  and
    prof-A: profile  $V A p$  and

```

$rej-mn$ :  $reject (m \triangleright n) \ V \ A \ p = A$  **and**  
 $x-in-A$ :  $x \in A$   
**from**  $emod-reject-m \ fin-A \ prof-A$   
**have**  $fin-defer$ :  
 $finite (defer \ m \ V \ A \ p) \wedge profile \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p)$   
**using**  $def-presv-prof \ defer-in-alts \ finite-subset$   
**by**  $(metis \ (no-types))$   
**from**  $emod-reject-m \ emod-reject-n \ fin-A \ prof-A$   
**have**  $seq-elect$ :  
 $elect (m \triangleright n) \ V \ A \ p =$   
 $elect \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) \cup elect \ m \ V$   
 $A \ p$   
**using**  $seq-comp-def-then-elect-elec-set$   
**by**  $metis$   
**from**  $emod-reject-n \ emod-reject-m \ fin-A \ prof-A$   
**have**  $def-limit$ :  
 $defer (m \triangleright n) \ V \ A \ p = defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p)$   
 $A \ p) \ p)$   
**using**  $seq-comp-defers-def-set$   
**by**  $metis$   
**from**  $emod-reject-n \ emod-reject-m \ fin-A \ prof-A$   
**have**  $elect (m \triangleright n) \ V \ A \ p \cup defer (m \triangleright n) \ V \ A \ p = A - reject (m \triangleright n) \ V \ A$   
 $p$   
**using**  $elec-and-def-not-rej \ seq-comp-sound$   
**by**  $metis$   
**hence**  $elect-def-disj$ :  
 $elect \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) \cup$   
 $elect \ m \ V \ A \ p \cup$   
 $defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) = \{\}$   
**using**  $def-limit \ seq-elect \ Diff-cancel \ rej-mn$   
**by**  $auto$   
**have**  $rej-def-eq-set$ :  
 $defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) -$   
 $defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) = \{\} \longrightarrow$   
 $reject \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) =$   
 $defer \ m \ V \ A \ p$   
**using**  $elect-def-disj \ emod-reject-n \ fin-defer$   
**by**  $(simp \ add: \ reject-not-elec-or-def)$   
**have**  
 $defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) -$   
 $defer \ n \ V (defer \ m \ V \ A \ p) (limit-profile (defer \ m \ V \ A \ p) \ p) = \{\} \longrightarrow$   
 $elect \ m \ V \ A \ p = elect \ m \ V \ A \ p \cap defer \ m \ V \ A \ p$   
**using**  $elect-def-disj$   
**by**  $blast$   
**thus**  $x \in \{\}$   
**using**  $rej-def-eq-set \ result-disj \ fin-defer \ Diff-cancel \ Diff-empty \ fin-A \ prof-A$   
 $emod-reject-m \ emod-reject-n \ reject-not-elec-or-def \ x-in-A$   
**by**  $metis$

qed  
qed  
qed

Sequential composition preserves the non-electing property.

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

Composing an electoral module that defers exactly 1 alternative in sequence after an electoral module that is electing results (still) in an electing electoral module.

**theorem** *seq-comp-electing[simp]*:  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**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-m1-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**  $\forall m'.$   
 $(\neg \text{electing } m' \vee \text{SCF-result.electoral-module } m' \wedge$   
 $(\forall A' V' p'. (A' \neq \{\} \wedge \text{finite } A' \wedge \text{profile } V' A' p') \longrightarrow \text{elect } m' V'$   
 $A' p' \neq \{\}))$   
 $\wedge (\exists A V p. (\text{electing } m' \vee \neg \text{SCF-result.electoral-module } m' \vee A \neq \{\}$   
 $\wedge \text{finite } A \wedge \text{profile } V A p \wedge \text{elect } m' V A p = \{\}))$   
**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*  
*def-m1-not-empty bot-eq-sup-iff finite-subset*  
**unfolding** *electing-def*  
**by** *metis*  
**qed**  
**qed**

**lemma** *def-lift-inv-seq-comp-help*:  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $q :: ('a, 'v) \text{ Profile}$  **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*  $m \wedge \text{SCF-result.electoral-module } n$   
**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*  $V A p q a \longrightarrow \text{finite-profile } V A p$   
**unfolding** *lifted-def*  
**by** *simp*  
**ultimately** **have** *defer-subset: defer*  $(m \triangleright n) V A p \subseteq \text{defer } m V A p$   
**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:  $\neg lifted \ V \ ?new-Ap \ ?new-p \ ?new-q \ a$ 
from def-and-lifted
have finite-profile  $V \ A \ q$ 
unfolding lifted-def
by simp
with modules new-A-eq
have prof-p:  $profile \ V \ ?new-Ap \ ?new-q$ 
using def-presv-prof
by (metis (no-types))
moreover from modules profile-p def-and-lifted
have prof-q:  $profile \ V \ ?new-Ap \ ?new-p$ 
using def-presv-prof
by (metis (no-types))
moreover from defer-subset def-and-lifted
have  $a \in ?new-Ap$ 
by blast
ultimately have lifted-stmt:
 $(\exists \ v \in V. \ Preference-Relation.lifted \ ?new-Ap \ (?new-p \ v) \ (?new-q \ v) \ a \longrightarrow$ 
 $(\exists \ v \in V. \neg Preference-Relation.lifted \ ?new-Ap \ (?new-p \ v) \ (?new-q \ v) \ a \wedge$ 
 $(?new-p \ v) \neq (?new-q \ v)))$ 
using unlifted-a def-and-lifted defer-in-alts infinite-super modules profile-p
unfolding lifted-def
by metis
from def-and-lifted modules
have  $\forall \ v \in V. (Preference-Relation.lifted \ A \ (p \ v) \ (q \ v) \ a \vee (p \ v) = (q \ v))$ 
unfolding Profile.lifted-def
by metis

```

```

with def-and-lifted modules mono-m
have  $\forall v \in V.$ 
  (Preference-Relation.lifted ?new-Ap (?new-p v) (?new-q v) a  $\vee$ 
   (?new-p v) = (?new-q v))
  using limit-lifted-imp-eq-or-lifted defer-in-alts
  unfolding Profile.lifted-def limit-profile.simps
  by (metis (no-types, lifting))
with lifted-stmt
have  $\forall v \in V. ( ?new-p\ v ) = ( ?new-q\ v )$ 
  by blast
with mono-m
show ?thesis
  using leI not-less-zero nth-equalityI voters-determine-n
  unfolding voters-determine-election.simps
  by presburger
qed
from mono-m mono-n
show ?thesis
  unfolding sequential-composition.simps
  by (metis (full-types))
qed

```

Sequential composition preserves the property defer-lift-invariance.

```

theorem seq-comp-presv-def-lift-inv[simp]:
fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  n :: ('a, 'v, 'a Result) Electoral-Module
assumes
  defer-lift-invariance m and
  defer-lift-invariance n and
  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 :: ('a, 'v) Profile and
  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 :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module
  assumes
    non-blocking-m: non-blocking m and
    non-electing-m: non-electing m and
    def-one-n: defers 1 n
  shows defers 1 (m  $\triangleright$  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  $\leq$  card A and
    fin-A: finite A and
    prof-A: profile V A p
  from pos-card
  have A  $\neq$  {}
    by auto
  with fin-A prof-A
  have reject m V A p  $\neq$  A
    using non-blocking-m
    unfolding non-blocking-def
    by simp
  hence  $\exists a. a \in A \wedge a \notin \text{reject } m \text{ } V \text{ } A \text{ } p$ 
    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 :: ('a, 'v, 'a Result) Electoral-Module and
    m' :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module
  assumes
    compatible: disjoint-compatibility m n and
    module-m': SCF-result.electoral-module m' and
    voters-determine-m': voters-determine-election m'
  shows disjoint-compatibility (m ▷ m') n
proof (unfold disjoint-compatibility-def, safe)
  show SCF-result.electoral-module (m ▷ m')
    using compatible module-m' seq-comp-sound
    unfolding disjoint-compatibility-def

```

```

    by metis
next
  show SCF-result.electoral-module n
    using compatible
    unfolding disjoint-compatibility-def
    by metis
next
  fix
    S :: 'a set and
    V :: 'v set
  have modules:
    SCF-result.electoral-module (m ▷ m') ∧ SCF-result.electoral-module n
    using compatible module-m' seq-comp-sound
    unfolding disjoint-compatibility-def
    by metis
  obtain A :: 'a set where rej-A:
     $A \subseteq S \wedge$ 
     $(\forall a \in A. \text{indep-of-alt } m \text{ } V \text{ } S \text{ } a \wedge (\forall p. \text{profile } V \text{ } S \text{ } p \longrightarrow a \in \text{reject } m \text{ } V \text{ } S \text{ } p)) \wedge$ 
     $(\forall a \in S - A. \text{indep-of-alt } n \text{ } V \text{ } S \text{ } a \wedge (\forall p. \text{profile } V \text{ } S \text{ } p \longrightarrow a \in \text{reject } n \text{ } V \text{ } S \text{ } p))$ 
    using compatible
    unfolding disjoint-compatibility-def
    by (metis (no-types, lifting))
  show
     $\exists A \subseteq S. (\forall a \in A. \text{indep-of-alt } (m \triangleright m') \text{ } V \text{ } S \text{ } a \wedge$ 
     $(\forall p. \text{profile } V \text{ } S \text{ } p \longrightarrow a \in \text{reject } (m \triangleright m') \text{ } V \text{ } S \text{ } p)) \wedge$ 
     $(\forall a \in S - A. \text{indep-of-alt } n \text{ } V \text{ } S \text{ } a \wedge (\forall p. \text{profile } V \text{ } S \text{ } p \longrightarrow a \in \text{reject } n \text{ } V \text{ } S \text{ } p))$ 
  proof
    have  $\forall a \text{ } p \text{ } q. a \in A \wedge \text{equiv-prof-except-}a \text{ } V \text{ } S \text{ } p \text{ } q \text{ } a \longrightarrow$ 
       $(m \triangleright m') \text{ } V \text{ } S \text{ } p = (m \triangleright m') \text{ } V \text{ } S \text{ } q$ 
    proof (safe)
      fix
        a :: 'a and
        p :: ('a, 'v) Profile and
        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-def: defer m V S p = defer m V S q
        using rej-A
        unfolding indep-of-alt-def
        by metis
      from lifting-equiv-p-q
      have profiles: profile V S p ∧ profile V S q
        unfolding equiv-prof-except-a-def
        by simp
    end
  end

```

hence  $(\text{defer } m \ V \ S \ p) \subseteq S$   
 using *compatible defer-in-alts*  
 unfolding *disjoint-compatibility-def*  
 by *metis*  
 moreover have  $a \notin \text{defer } m \ V \ S \ q$   
 using *a-in-A compatible defer-not-elec-or-rej[of m V A p]*  
     *profiles rej-A IntI emptyE result-disj*  
 unfolding *disjoint-compatibility-def*  
 by *metis*  
 ultimately have  
    $\forall v \in V. \text{limit-profile } (\text{defer } m \ V \ S \ p) \ p \ v = \text{limit-profile } (\text{defer } m \ V \ S \ q) \ q$   
 v  
   using *lifting-equiv-p-q negl-diff-imp-eq-limit-prof[of V S p q a defer m V S*  
 q]  
     unfolding *eq-def limit-profile.simps*  
     by *blast*  
 with *eq-def*  
 have  $m' \ V \ (\text{defer } m \ V \ S \ p) \ (\text{limit-profile } (\text{defer } m \ V \ S \ p) \ p) =$   
      $m' \ V \ (\text{defer } m \ V \ S \ q) \ (\text{limit-profile } (\text{defer } m \ V \ S \ q) \ q)$   
     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  $(m \triangleright m') \ V \ S \ p = (m \triangleright m') \ V \ S \ q$   
     unfolding *sequential-composition.simps*  
     by *(metis (full-types))*  
 qed  
 moreover have  $\forall a' \in A. \forall p'. \text{profile } V \ S \ p' \longrightarrow a' \in \text{reject } (m \triangleright m') \ V \ S \ p'$   
     using *rej-A UnI1 prod.sel*  
     unfolding *sequential-composition.simps*  
     by *metis*  
 ultimately show  $A \subseteq S \wedge$   
      $(\forall a' \in A. \text{indep-of-alt } (m \triangleright m') \ V \ S \ a' \wedge$   
      $(\forall p'. \text{profile } V \ S \ p' \longrightarrow a' \in \text{reject } (m \triangleright m') \ V \ S \ p')) \wedge$   
      $(\forall a' \in S - A. \text{indep-of-alt } n \ V \ S \ a' \wedge$   
      $(\forall p'. \text{profile } V \ S \ p' \longrightarrow a' \in \text{reject } n \ V \ S \ p'))$   
     using *rej-A indep-of-alt-def modules*  
     by *(metis (no-types, lifting))*  
 qed  
 qed  
 theorem *seq-comp-cond-compat[simp]*:  
   fixes  
      $m :: ('a, 'v, 'a \text{ Result}) \text{Electoral-Module}$  and  
      $n :: ('a, 'v, 'a \text{ Result}) \text{Electoral-Module}$   
   assumes  
     *dcc-m: defer-condorcet-consistency m* and

*nb-n: non-blocking n and*  
*ne-n: non-electing n*  
**shows** *condorcet-compatibility* ( $m \triangleright n$ )  
**proof** (*unfold condorcet-compatibility-def, safe*)  
**have** *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* ( $m \triangleright n$ )  
**using** *seq-comp-sound*  
**by** *metis*  
**thus** *SCF-result.electoral-module* ( $m \triangleright n$ )  
**by** *presburger*  
**next**  
**fix**  
*A :: 'a set and*  
*V :: 'v set and*  
*p :: ('a, 'v) Profile and*  
*a :: 'a*  
**assume**  
*cw-a: condorcet-winner V A p a and*  
*a-in-rej-seq-m-n: a ∈ reject (m ▷ 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**  $m \ V A p = (\{\}, A - (\text{defer } m \ V A p), \{a\})$   
**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* ( $m \triangleright n$ )  
**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(1) snd-conv*

**unfolding** *defer-condorcet-consistency-def*  
**by** (*metis* (*mono-tags*, *lifting*))  
**have** *elect m V A p = {}*  
**using** *cw-a def-m rej-m dcc-m prod.sel(1)*  
**unfolding** *defer-condorcet-consistency-def*  
**by** (*metis* (*mono-tags*, *lifting*))  
**hence** *diff-elect-m: A - elect m V A p = A*  
**using** *Diff-empty*  
**by** (*metis* (*full-types*))  
**have** *cond-win:*  
*finite A ∧ finite V ∧ profile V A p ∧ a ∈ A ∧ (∀ a'. a' ∈ A - {a'} → 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 ∧*  
*(∀ A' V' p'. A' ≠ {} ∧ finite A' ∧ finite V' ∧ profile V' A' p' → reject n*  
*V' A' p' ≠ A')*  
**using** *nb-n non-blocking-def*  
**by** *metis*  
**have** *def-seq-diff: defer (m ▷ n) V A p = A - elect (m ▷ n) V A p - reject (m*  
*▷ n) V A p*  
**using** *defer-not-elec-or-rej cond-win sound-seq-m-n*  
**by** *metis*  
**have** *set-ins: ∀ a' A'. (a'::'a) ∈ A' → 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** *snd (elect m V A p ∪ elect n V (defer m V A p) (limit-profile (defer m V*  
*A p) p),*  
*reject m V A p ∪ reject n V (defer m V A p) (limit-profile (defer m V A*  
*p) p),*  
*defer n V (defer m V A p) (limit-profile (defer m V A p) p)) =*  
*(reject m V A p ∪ reject n V (defer m V A p) (limit-profile (defer m V*  
*A p) p),*  
*defer n V (defer m V A p) (limit-profile (defer m V A p) p))*  
**by** *blast*  
**hence** *seq-snd-simplified:*  
*snd ((m ▷ n) V A p) =*  
*(reject m V A p ∪ reject n V (defer m V A p) (limit-profile (defer m V A p)*  
*p),*  
*defer n V (defer m V A p) (limit-profile (defer m V A p) p))*  
**using** *sequential-composition.simps*  
**by** *metis*  
**hence** *seq-rej-union-eq-rej:*  
*reject m V A p ∪ reject n V (defer m V A p) (limit-profile (defer m V A p) p)*  
 =

$\text{reject } (m \triangleright n) \ V \ A \ p$   
**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-elec-or-def seq-comp-def-then-elect-elec-set finite-subset*  
*seq-comp-defers-def-set sup-bot.left-neutral*  
**unfolding** *non-electing-def*  
**by** (*metis (no-types, lifting)*)  
**thus** *False*  
**using** *a-in-rej-seq-m-n*  
**by** *blast*  
**next**  
**fix**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $a :: 'a$  **and**  
 $a' :: 'a$   
**assume**  
 $\text{cw-}a$ : *condorcet-winner*  $V \ A \ p \ a$  **and**  
 $\text{not-cw-}a'$ :  $\neg \text{condorcet-winner } V \ A \ p \ a'$  **and**  
 $a'$ -*in-elect-seq-m-n*:  $a' \in \text{elect } (m \triangleright n) \ V \ A \ p$   
**hence**  $\exists a''.$  *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*  $(m \triangleright n)$   
**using** *seq-comp-sound*

by *metis*  
**have**  $\text{reject } m \ V \ A \ p = A - \{a\}$   
 using *cw-a dcc-m prod.sel(1) snd-conv result-m*  
**unfolding** *defer-condorcet-consistency-def*  
 by (*metis (mono-tags, lifting)*)  
**hence**  $a' \text{-in-rej}: a' \in \text{reject } m \ V \ A \ p$   
 using *Diff-iff cw-a not-cw-a' a'-in-elect-seq-m-n condorcet-winner.elims(1)*  
*elect-in-alts singleton-iff sound-seq-m-n subset-iff*  
 by (*metis (no-types, lifting)*)  
**have**  $\forall \ p' \ A' \ p''. \ p' = (A'::'a \text{ set}, p''::'a \text{ set} \times 'a \text{ set}) \longrightarrow \text{snd } p' = p''$   
 by *simp*  
**hence**  $m\text{-seq-n}$ :  
 $\text{snd } (\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p)$   
 $p),$   
 $\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p)$   
 $p),$   
 $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)) =$   
 $(\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A$   
 $p) \ p),$   
 $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p))$   
 by *blast*  
**have**  $a' \in \text{elect } m \ V \ A \ p$   
 using  $a' \text{-in-elect-seq-m-n condorcet-winner.simps cw-a def-presv-prof ne-n}$   
*seq-comp-def-then-elect-elec-set sound-m sup-bot.left-neutral*  
**unfolding** *non-electing-def*  
 by (*metis (no-types)*)  
**hence**  $a \text{-in-rej-union}$ :  
 $a \in \text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A$   
 $p) \ p)$   
 using *Diff-iff a'-in-rej condorcet-winner.simps cw-a*  
*reject-not-elec-or-def sound-m*  
 by (*metis (no-types)*)  
**have**  $m\text{-seq-n-full}$ :  
 $(m \triangleright n) \ V \ A \ p =$   
 $(\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p),$   
 $\text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p)$   
 $p),$   
 $\text{defer } n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p))$   
**unfolding** *sequential-composition.simps*  
 by *metis*  
**have**  $\forall \ A' \ A''. \ (A'::'a \text{ set}) = \text{fst } (A', A''::'a \text{ set})$   
 by *simp*  
**hence**  $a \in \text{reject } (m \triangleright n) \ V \ A \ p$   
 using  $a \text{-in-rej-union } m\text{-seq-n } m\text{-seq-n-full}$   
 by *presburger*  
**moreover have**  
 $\text{finite } A \wedge \text{finite } V \wedge \text{profile } V \ A \ p \wedge a \in A \wedge (\forall \ a''. \ a'' \in A - \{a\} \longrightarrow \text{wins}$   
 $V \ a \ p \ a'')$   
 using *cw-a m-seq-n-full a'-in-elect-seq-m-n a'-in-rej ne-n sound-m*

```

    unfolding condorcet-winner.simps
  by metis
ultimately show False
using a'-in-elect-seq-m-n IntI empty-iff result-disj sound-seq-m-n a'-in-rej def-presv-prof
fst-conv m-seq-n-full ne-n non-electing-def sound-m sup-bot.right-neutral
  by metis
next
fix
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  a :: 'a and
  a' :: 'a
assume
  cw-a: condorcet-winner V A p a and
  a'-in-A: a' ∈ 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(1) snd-conv
  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  $a' \in \text{reject } (m \triangleright n) \vee A \ p$ 
  using fst-eqD
  by (metis (no-types))
qed

```

Composing a defer-condorcet-consistent electoral module in sequence with a non-blocking and non-electing electoral module results in a defer-condorcet-consistent module.

```

theorem seq-comp-dcc[simp]:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
  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
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  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 - (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  $\triangleright$  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 condorcet-winner.elims(2) empty-iff
      seq-comp-def-set-bounded sound-m subset-singletonD nb-n
  unfolding non-blocking-def
  by metis
  thus a' = a
    using a'-in-def-seq-m-n
  by blast
next
  have ∃ a'. defer-condorcet-consistency m ∧ condorcet-winner V A p a'
    using cw-a dcc-m
  by blast
  hence m V A p = ({}, A - (defer m V A p), {a})
    using defer-condorcet-consistency-def cw-a cond-winner-unique
  by (metis (no-types, lifting))
  hence elect-m-empty: elect m V A p = {}
    using eq-fst-iff
  by metis
  have profile V (defer m V A p) (limit-profile (defer m V A p) p)
    using condorcet-winner.simps cw-a def-presv-prof sound-m
  by (metis (no-types))
  hence elect n V (defer m V A p) (limit-profile (defer m V A p) p) = {}
    using ne-n non-electing-def
  by metis
  hence elect (m ▷ n) V A p = {}
    using elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral
  by (metis (no-types))
  moreover have condorcet-compatibility (m ▷ n)
    using dcc-m nb-n ne-n
  by simp
  hence a ∉ reject (m ▷ n) V A p
    unfolding condorcet-compatibility-def
  using cw-a
  by metis

```

```

ultimately show  $a \in \text{defer } (m \triangleright n) \ V \ A \ p$ 
  using cw-a electoral-mod-defer-elem empty-iff
        sound-seq-m-n condorcet-winner.simps
  by metis
qed
have profile  $V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$ 
  using condorcet-winner.simps cw-a def-presv-prof sound-m
  by (metis (no-types))
hence elect  $n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) = \{\}$ 
  using ne-n
  unfolding non-electing-def
  by metis
hence elect  $(m \triangleright n) \ V \ A \ p = \{\}$ 
  using elect-m-empty seq-comp-def-then-elect-elec-set sup-bot.right-neutral
  by (metis (no-types))
moreover have def-seq-m-n-eq-a:  $\text{defer } (m \triangleright n) \ V \ A \ p = \{a\}$ 
  using cw-a defer-eq-a
  by (metis (no-types))
ultimately have  $(m \triangleright n) \ V \ A \ p = (\{\}, A - \{a\}, \{a\})$ 
  using Diff-empty cw-a elect-rej-def-combination
        reject-not-elec-or-def sound-seq-m-n condorcet-winner.simps
  by (metis (no-types))
moreover have  $\{a' \in A. \text{condorcet-winner } V \ A \ p \ a'\} = \{a\}$ 
  using cw-a cond-winner-unique
  by metis
ultimately show  $(m \triangleright n) \ V \ A \ p$ 
  =  $(\{\}, A - \text{defer } (m \triangleright n) \ V \ A \ p, \{a' \in A. \text{condorcet-winner } V \ A \ p \ a'\})$ 
  using def-seq-m-n-eq-a
  by metis
qed

```

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

```

theorem seq-comp-mono[simp]:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{Electoral-Module}$ 
  assumes
    def-monotone-m: defer-lift-invariance  $m$  and
    non-ele-m: non-electing  $m$  and
    def-one-m: defers 1  $m$  and
    electing-n: electing  $n$ 
  shows monotonicity  $(m \triangleright n)$ 
proof (unfold monotonicity-def, safe)
  have 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 ( $m \triangleright n$ )
  using seq-comp-sound
  by metis
next
fix
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  q :: ('a, 'v) Profile and
  w :: 'a
assume
  elect-w-in-p:  $w \in \text{elect } (m \triangleright n) \ V \ A \ p$  and
  lifted-w: Profile.lifted V A p q w
thus  $w \in \text{elect } (m \triangleright n) \ V \ A \ q$ 
  unfolding lifted-def
  using seq-comp-def-then-elect lifted-w assms
  unfolding defer-lift-invariance-def
  by metis
qed

```

Composing a defer-invariant-monotone electoral module in sequence before a non-electing, defer-monotone electoral module that defers exactly 1 alternative results in a defer-lift-invariant electoral module.

```

theorem def-inv-mono-imp-def-lift-inv[simp]:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module
  assumes
    strong-def-mon-m: defer-invariant-monotonicity m and
    non-electing-n: non-electing n and
    defers-one: defers 1 n and
    defer-monotone-n: defer-monotonicity n and
    voters-determine-n: voters-determine-election n
  shows defer-lift-invariance ( $m \triangleright n$ )
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 :: ('a, 'v) Profile and
  q :: ('a, 'v) Profile and
  a :: 'a
assume
  defer-a-p: a ∈ defer (m ▷ n) V A p and
  lifted-a: Profile.lifted V A p q a
have non-electing-m: non-electing m
  using strong-def-mon-m
  unfolding defer-invariant-monotonicity-def
  by simp
have electoral-mod-m: SCF-result.electoral-module m
  using strong-def-mon-m
  unfolding defer-invariant-monotonicity-def
  by metis
have electoral-mod-n: SCF-result.electoral-module n
  using defers-one
  unfolding defers-def
  by metis
have finite-profile-p: finite-profile V A p
  using lifted-a
  unfolding Profile.lifted-def
  by simp
have finite-profile-q: finite-profile V A q
  using lifted-a
  unfolding Profile.lifted-def
  by simp
have 1 ≤ card A
  using Profile.lifted-def card-eq-0-iff emptyE less-one lifted-a linorder-le-less-linear
  by metis
hence n-defers-exactly-one-p: card (defer n V A p) = 1
  using finite-profile-p defers-one
  unfolding defers-def
  by (metis (no-types))
have fin-prof-def-m-q: profile V (defer m V A q) (limit-profile (defer m V A q)
q)
  using def-presv-prof electoral-mod-m finite-profile-q
  by (metis (no-types))
have def-seq-m-n-q:
  defer (m ▷ n) V A q = defer n V (defer m V A q) (limit-profile (defer m V A
q) q)
  using seq-comp-defers-def-set
  by simp
have prof-def-m: profile V (defer m V A p) (limit-profile (defer m V A p) p)
  using def-presv-prof electoral-mod-m finite-profile-p

```

```

  by (metis (no-types))
hence prof-seq-comp-m-n:
  profile V (defer n V (defer m V A p) (limit-profile (defer m V A p) p))
    (limit-profile (defer n V (defer m V A p) (limit-profile (defer m V A p) p))
      (limit-profile (defer m V A p) p))
  using def-presv-prof electoral-mod-n
  by (metis (no-types))
have a-non-empty:  $a \notin \{\}$ 
  by simp
have def-seq-m-n:
  defer (m  $\triangleright$  n) V A p = defer n V (defer m V A p) (limit-profile (defer m V A
p) p)
  using seq-comp-defers-def-set
  by simp
have 1  $\leq$  card (defer n V (defer m V A p) (limit-profile (defer m V A p) p))
  using a-non-empty card-gt-0-iff defer-a-p electoral-mod-n prof-def-m
    seq-comp-defers-def-set One-nat-def Suc-leI defer-in-alts
    electoral-mod-m finite-profile-p finite-subset
  by (metis (mono-tags))
hence card (defer n V (defer n V (defer m V A p) (limit-profile (defer m V A
p) p))
  (limit-profile (defer n V (defer m V A p) (limit-profile (defer m V A p) p))
    (limit-profile (defer m V A p) p))) = 1
  using n-defers-exactly-one-p prof-seq-comp-m-n defers-one defer-in-alts
    electoral-mod-m finite-profile-p finite-subset prof-def-m
  unfolding defers-def
  by metis
hence defer-seq-m-n-eq-one: card (defer (m  $\triangleright$  n) V A p) = 1
  using One-nat-def Suc-leI a-non-empty card-gt-0-iff def-seq-m-n defer-a-p
    defers-one electoral-mod-m prof-def-m finite-profile-p
    seq-comp-def-set-trans defer-in-alts rev-finite-subset
  unfolding defers-def
  by metis
hence def-seq-m-n-eq-a: defer (m  $\triangleright$  n) V A p = {a}
  using defer-a-p is-singleton-altdef is-singleton-the-elem singletonD
  by (metis (no-types))
show (m  $\triangleright$  n) V A p = (m  $\triangleright$  n) V A q
proof (cases)
  assume defer m V A q  $\neq$  defer m V A p
  hence defer m V A q = {a}
    using defer-a-p electoral-mod-n finite-profile-p lifted-a seq-comp-def-set-trans
      strong-def-mon-m
    unfolding defer-invariant-monotonicity-def
    by (metis (no-types))
  moreover from this
  have (a  $\in$  defer m V A p)  $\longrightarrow$  card (defer (m  $\triangleright$  n) V A q) = 1
    using card-eq-0-iff card-insert-disjoint defers-one electoral-mod-m empty-iff
      order-refl finite.emptyI seq-comp-defers-def-set def-presv-prof
      finite-profile-q finite.insertI

```

**unfolding** *One-nat-def defers-def*  
**by** *metis*  
**moreover have**  $a \in \text{defer } m \ V \ A \ p$   
**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: defer m V A q = 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: elect m V A p = 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**  
 $(\forall v \in V. (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p) \ v = (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ q) \ v)$   
 $\implies n \ V \ (\text{defer } m \ V \ A \ p) \ (\text{limit-profile } (\text{defer } m \ V \ A \ p) \ p)$   
 $= n \ V \ (\text{defer } m \ V \ A \ q) \ (\text{limit-profile } (\text{defer } m \ V \ A \ q) \ q)$   
**using** *voters-determine-n def-eq*  
**unfolding** *voters-determine-election.simps*

by *presburger*  
 moreover have  

$$\text{lifted } V \text{ (defer } m \text{ } V \text{ } A \text{ } q) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } p) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } q) \text{ } a$$

$$\implies \text{defer } n \text{ } V \text{ (defer } m \text{ } V \text{ } A \text{ } p) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } p)$$

$$= \text{defer } n \text{ } V \text{ (defer } m \text{ } V \text{ } A \text{ } q) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } q) \text{ } q)$$
 proof –  
 assume *lifted*:  

$$\text{Profile.lifted } V \text{ (defer } m \text{ } V \text{ } A \text{ } q) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } p) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } q) \text{ } a$$
 hence  $a \in \text{defer } n \text{ } V \text{ (defer } m \text{ } V \text{ } A \text{ } q) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } q) \text{ } q)$   
 using *lifted-a def-seq-m-n defer-a-p defer-monotone-n fin-prof-def-m-q def-eq*  
 unfolding *defer-monotonicity-def*  
 by *metis*  
 hence  $a \in \text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } q$   
 using *def-seq-m-n-q*  
 by *simp*  
 moreover have  $\text{card } (\text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } q) = 1$   
 using *def-seq-m-n-q defers-one def-eq defer-seq-m-n-eq-one defers-def lifted electoral-mod-m fin-prof-def-m-q finite-profile-p seq-comp-def-card-bounded Profile.lifted-def*  
 by (*metis (no-types, lifting)*)  
 ultimately have  $\text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } q = \{a\}$   
 using *a-non-empty card-1-singletonE insertE*  
 by *metis*  
 thus  $\text{defer } n \text{ } V \text{ (defer } m \text{ } V \text{ } A \text{ } p) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } p) \text{ } p)$   

$$= \text{defer } n \text{ } V \text{ (defer } m \text{ } V \text{ } A \text{ } q) \text{ (limit-profile (defer } m \text{ } V \text{ } A \text{ } q) \text{ } q)$$
 using *def-seq-m-n-eq-a def-seq-m-n-q def-seq-m-n*  
 by *presburger*  
 qed  
 ultimately have  $\text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } p = \text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } q$   
 using *def-seq-m-n def-seq-m-n-q*  
 by *presburger*  
 hence  $\text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } p = \text{defer } (m \triangleright n) \text{ } V \text{ } A \text{ } q$   
 using *a-non-empty def-eq def-seq-m-n def-seq-m-n-q defer-a-p defer-monotone-n finite-profile-p defer-seq-m-n-eq-one defers-one electoral-mod-m fin-prof-def-m-q*  
 unfolding *defers-def*  
 by (*metis (no-types, lifting)*)  
 moreover from *this*  
 have  $\text{reject } (m \triangleright n) \text{ } V \text{ } A \text{ } p = \text{reject } (m \triangleright n) \text{ } V \text{ } A \text{ } q$   
 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  $\text{snd } ((m \triangleright n) \text{ } V \text{ } A \text{ } p) = \text{snd } ((m \triangleright n) \text{ } V \text{ } A \text{ } q)$   
 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
   $m \parallel_a n == \text{parallel-composition } m n a$ 

```

### 6.4.2 Soundness

```

theorem par-comp-sound[simp]:
fixes
   $m ::$  ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module and
   $n ::$  ( $'a, 'v, 'a \text{ Result}$ ) Electoral-Module and
   $a ::$   $'a \text{ Aggregator}$ 
assumes
  SCF-result.electoral-module  $m$  and

```

```

    SCF-result.electoral-module  $n$  and
    aggregator  $a$ 
  shows SCF-result.electoral-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
     $\forall a'. \text{aggregator } a' =$ 
     $(\forall A' e r d e' r' d'.$ 
     $(\text{well-formed-SCF } (A'::'a \text{ set}) (e, r', d)$ 
     $\wedge \text{well-formed-SCF } A' (r, d', e'))$ 
     $\longrightarrow \text{well-formed-SCF } A' (a' A' (e, r', d) (r, d', e')))$ 
  unfolding aggregator-def
  by blast
  moreover have
     $\forall m' V' A' p'.$ 
     $(\text{SCF-result.electoral-module } m' \wedge \text{finite } (A'::'a \text{ set})$ 
     $\wedge \text{finite } (V'::'v \text{ set}) \wedge \text{profile } V' A' p') \longrightarrow \text{well-formed-SCF } A' (m' V' A'$ 
     $p')$ 
  using par-comp-result-sound
  by (metis (no-types))
  ultimately have well-formed-SCF  $A (a A (m V A p) (n V A p))$ 
  using elect-rej-def-combination assms
  by (metis par-comp-result-sound)
  thus well-formed-SCF  $A ((m \parallel_a n) V A p)$ 
  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
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $a :: 'a \text{ Aggregator}$ 
  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 set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  w :: 'a
assume
  prof-A: profile V A p and
  w-wins: w  $\in$  elect (m  $\parallel_a$  n) V A p
have emod-m: 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  $\forall r r' d d' e e' A' f.$ 
  ((well-formed-SCF (A'::'a set) (e', r', d')  $\wedge$ 
    well-formed-SCF A' (e, r, d))  $\longrightarrow$ 
    elect-r (f A' (e', r', d') (e, r, d))  $\subseteq$  e'  $\cup$  e  $\wedge$ 
    reject-r (f A' (e', r', d') (e, r, d))  $\subseteq$  r'  $\cup$  r  $\wedge$ 
    defer-r (f A' (e', r', d') (e, r, d))  $\subseteq$  d'  $\cup$  d) =
    ((well-formed-SCF A' (e', r', d')  $\wedge$ 
    well-formed-SCF A' (e, r, d))  $\longrightarrow$ 
    elect-r (f A' (e', r', d') (e, r, d))  $\subseteq$  e'  $\cup$  e  $\wedge$ 
    reject-r (f A' (e', r', d') (e, r, d))  $\subseteq$  r'  $\cup$  r  $\wedge$ 
    defer-r (f A' (e', r', d') (e, r, d))  $\subseteq$  d'  $\cup$  d)
  by linarith
hence  $\forall a'. \text{agg-conservative } a' =$ 
  (aggregator a'  $\wedge$ 
    ( $\forall A' e e' d d' r r'.$ 
      (well-formed-SCF (A'::'a 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 agg-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  $\subseteq$  ((elect m V A p)  $\cup$  (elect n V A p)))
using emod-m emod-n par-comp-result-sound
      prod.collapse prof-A
by metis
hence w  $\in$  ((elect m V A p)  $\cup$  (elect n V A p))
using w-wins
by auto
thus w  $\in$  {}
using sup-bot-right prof-A
      non-electing-m non-electing-n
unfolding non-electing-def
by (metis (no-types, lifting))
qed

end

```

## 6.5 Loop Composition

```

theory Loop-Composition
imports Basic-Modules/Component-Types/Termination-Condition
      Basic-Modules/Defer-Module
      Sequential-Composition
begin

```

The loop composition uses the same module in sequence, combined with a termination condition, until either

- the termination condition is met or
- no new decisions are made (i.e., a fixed point is reached).

### 6.5.1 Definition

**lemma** *loop-termination-helper*:

**fixes**

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$  **and**  
 $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$

**assumes**

$\neg t (acc \ V \ A \ p)$  **and**  
 $defer (acc \triangleright m) \ V \ A \ p \subset defer \ acc \ V \ A \ p$  **and**  
 $finite (defer \ acc \ V \ A \ p)$

**shows**  $((acc \triangleright m, m, t, V, A, p), (acc, m, t, V, A, p)) \in$   
 $measure (\lambda (acc, m, t, V, A, p). card (defer \ acc \ V \ A \ p))$

**using** *assms psubset-card-mono*

**by** *simp*

This function handles the accumulator for the following loop composition function.

**function** *loop-comp-helper* ::

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$   
 $'a \text{ Termination-Condition} \Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **where**  
 $finite (defer \ acc \ V \ A \ p) \wedge (defer (acc \triangleright m) \ V \ A \ p) \subset (defer \ acc \ V \ A \ p)$   
 $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow$   
 $loop\text{-}comp\text{-}helper \ acc \ m \ t \ V \ A \ p = acc \ V \ A \ p \mid$   
 $\neg (finite (defer \ acc \ V \ A \ p) \wedge (defer (acc \triangleright m) \ V \ A \ p) \subset (defer \ acc \ V \ A \ p))$   
 $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow$   
 $loop\text{-}comp\text{-}helper \ acc \ m \ t \ V \ A \ p = loop\text{-}comp\text{-}helper (acc \triangleright m) \ m \ t \ V \ A \ p$

**proof** –

**fix**

$P :: bool$  **and**

$accum ::$

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \times ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
 $\times 'a \text{ Termination-Condition} \times 'v \text{ set} \times 'a \text{ set} \times ('a, 'v) \text{ Profile}$

**have** *accum-exists*:  $\exists m \ n \ t \ V \ A \ p. (m, n, t, V, A, p) = accum$

**using** *prod-cases5*

**by** *metis*

**assume**

$\bigwedge acc \ V \ A \ p \ m \ t.$   
 $finite (defer \ acc \ V \ A \ p) \wedge defer (acc \triangleright m) \ V \ A \ p \subset defer \ acc \ V \ A \ p$   
 $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow accum = (acc, m, t, V, A, p) \Longrightarrow P$  **and**

$\bigwedge acc \ V \ A \ p \ m \ t.$   
 $\neg (finite (defer \ acc \ V \ A \ p) \wedge defer (acc \triangleright m) \ V \ A \ p \subset defer \ acc \ V \ A \ p)$   
 $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow accum = (acc, m, t, V, A, p) \Longrightarrow P$

**thus**  $P$

**using** *accum-exists*

**by** *metis*

**next**

```

fix
   $t :: 'a \text{ Termination-Condition}$  and
   $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $t' :: 'a \text{ Termination-Condition}$  and
   $acc' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A' :: 'a \text{ set}$  and
   $V' :: 'v \text{ set}$  and
   $p' :: ('a, 'v) \text{ Profile}$  and
   $m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
assume
   $finite (defer\ acc\ V\ A\ p) \wedge defer (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$ 
     $\longrightarrow t (acc\ V\ A\ p)$  and
   $finite (defer\ acc'\ V'\ A'\ p') \wedge defer (acc' \triangleright m')\ V'\ A'\ p' \subset defer\ acc'\ V'\ A'\ p'$ 
     $\longrightarrow t' (acc'\ V'\ A'\ p')$  and
   $(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$ 
thus  $acc\ V\ A\ p = acc'\ V'\ A'\ p'$ 
by fastforce
next
fix
   $t :: 'a \text{ Termination-Condition}$  and
   $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $t' :: 'a \text{ Termination-Condition}$  and
   $acc' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A' :: 'a \text{ set}$  and
   $V' :: 'v \text{ set}$  and
   $p' :: ('a, 'v) \text{ Profile}$  and
   $m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
assume
   $finite (defer\ acc\ V\ A\ p) \wedge defer (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$ 
     $\longrightarrow t (acc\ V\ A\ p)$  and
   $\neg (finite (defer\ acc'\ V'\ A'\ p') \wedge defer (acc' \triangleright m')\ V'\ A'\ p' \subset defer\ acc'\ V'\ A'$ 
 $p'$ 
     $\longrightarrow t' (acc'\ V'\ A'\ p'))$  and
   $(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$ 
thus  $acc\ V\ A\ p = loop\text{-}comp\text{-}helper\text{-}sumC (acc' \triangleright m', m', t', V', A', p')$ 
by force
next
fix
   $t :: 'a \text{ Termination-Condition}$  and
   $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and

```

$V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t' :: 'a \text{ Termination-Condition}$  **and**  
 $acc' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A' :: 'a \text{ set}$  **and**  
 $V' :: 'v \text{ set}$  **and**  
 $p' :: ('a, 'v) \text{ Profile}$  **and**  
 $m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**assume**  
 $\neg (\text{finite } (\text{defer } acc \ V \ A \ p) \wedge \text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p$   
 $\quad \longrightarrow t \ (acc \ V \ A \ p))$  **and**  
 $\neg (\text{finite } (\text{defer } acc' \ V' \ A' \ p') \wedge \text{defer } (acc' \triangleright m') \ V' \ A' \ p' \subset \text{defer } acc' \ V' \ A'$   
 $p'$   
 $\quad \longrightarrow t' \ (acc' \ V' \ A' \ p'))$  **and**  
 $(acc, m, t, V, A, p) = (acc', m', t', V', A', p')$   
**thus**  $\text{loop-comp-helper-sumC } (acc \triangleright m, m, t, V, A, p) =$   
 $\text{loop-comp-helper-sumC } (acc' \triangleright m', m', t', V', A', p')$   
**by force**  
**qed**  
**termination**  
**proof** (*safe*)  
**fix**  
 $m :: ('b, 'a, 'b \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('b, 'a, 'b \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'b \text{ Termination-Condition}$  **and**  
 $A :: 'b \text{ set}$  **and**  
 $V :: 'a \text{ set}$  **and**  
 $p :: ('b, 'a) \text{ Profile}$   
**have** *term-rel*:  
 $\exists R. \text{wf } R \wedge$   
 $(\text{finite } (\text{defer } m \ V \ A \ p) \wedge \text{defer } (m \triangleright n) \ V \ A \ p \subset \text{defer } m \ V \ A \ p \longrightarrow t \ (m$   
 $V \ A \ p) \vee$   
 $((m \triangleright n, n, t, V, A, p), (m, n, t, V, A, p)) \in R)$   
**using** *loop-termination-helper wf-measure termination*  
**by** (*metis (no-types)*)  
**obtain**  
 $R :: (((('b, 'a, 'b \text{ Result}) \text{ Electoral-Module} \times ('b, 'a, 'b \text{ Result}) \text{ Electoral-Module}$   
 $\times$   
 $(('b \text{ Termination-Condition}) \times 'a \text{ set} \times 'b \text{ set} \times ('b, 'a) \text{ Profile}) \times$   
 $(('b, 'a, 'b \text{ Result}) \text{ Electoral-Module} \times ('b, 'a, 'b \text{ Result}) \text{ Electoral-Module}$   
 $\times$   
 $(('b \text{ Termination-Condition}) \times 'a \text{ set} \times 'b \text{ set} \times ('b, 'a) \text{ Profile}) \text{ set}$  **where**  
 $\text{wf } R \wedge$   
 $(\text{finite } (\text{defer } m \ V \ A \ p) \wedge \text{defer } (m \triangleright n) \ V \ A \ p \subset \text{defer } m \ V \ A \ p \longrightarrow t \ (m \ V$   
 $A \ p) \vee$   
 $((m \triangleright n, n, t, V, A, p), m, n, t, V, A, p) \in R)$   
**using** *term-rel*  
**by** *presburger*

**have**  $\forall R'$ .  
*All* (*loop-comp-helper-dom* ::  
 ('b, 'a, 'b Result) Electoral-Module  $\times$  ('b, 'a, 'b Result) Electoral-Module  
 $\times$  'b Termination-Condition  $\times$  'a set  $\times$  'b set  $\times$  ('b, 'a) Profile  $\Rightarrow$  bool)  $\vee$   
 ( $\exists t' m' A' V' p' n'. \text{wf } R' \longrightarrow$   
 (( $m' \triangleright n', n', t', V' :: 'a \text{ set}, A' :: 'b \text{ set}, p'$ ),  $m', n', t', V', A', p'$ )  $\notin R' \wedge$   
 finite (defer  $m' V' A' p'$ )  $\wedge$  defer ( $m' \triangleright n'$ )  $V' A' p' \subset$  defer  $m' V' A' p'$ )  
 $\wedge$   
 $\neg t' (m' V' A' p')$ )  
**using** *termination*  
**by** *metis*  
**thus** *loop-comp-helper-dom* ( $m, n, t, V, A, p$ )  
**using** *loop-termination-helper wf-measure*  
**by** *metis*  
**qed**

**lemma** *loop-comp-code-helper*[code]:  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$  **and**  
 $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$   
**shows**  
*loop-comp-helper*  $acc\ m\ t\ V\ A\ p =$   
 (if ( $t\ (acc\ V\ A\ p) \vee \neg ((\text{defer } (acc \triangleright m)\ V\ A\ p) \subset (\text{defer } acc\ V\ A\ p))$ )  $\vee$   
 infinite (defer  $acc\ V\ A\ p$ ))  
 then ( $acc\ V\ A\ p$ ) else (*loop-comp-helper* ( $acc \triangleright m$ )  $m\ t\ V\ A\ p$ ))  
**using** *loop-comp-helper.simps*  
**by** (*metis* (*no-types*))

**function** *loop-composition* :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$  'a Termination-Condition  
 $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module **where**  
 $t\ (\{\}, \{\}, A) \Longrightarrow \text{loop-composition } m\ t\ V\ A\ p = \text{defer-module } V\ A\ p \mid$   
 $\neg(t\ (\{\}, \{\}, A)) \Longrightarrow \text{loop-composition } m\ t\ V\ A\ p = (\text{loop-comp-helper } m\ m\ t)\ V\ A\ p$   
**by** (*fastforce, simp-all*)  
**termination**  
**using** *termination wf-empty*  
**by** *blast*

**abbreviation** *loop* :: ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$  'a Termination-Condition  
 $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module ( $- \odot -$  50) **where**  
 $m \odot_t \equiv \text{loop-composition } m\ t$

**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 =
  (if (t ({}, {}, A))
    then (defer-module V A p) else (loop-comp-helper m m t) V A p)
by simp

lemma loop-comp-helper-imp-partit:
fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  t :: 'a Termination-Condition and
  acc :: ('a, 'v, 'a Result) Electoral-Module and
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  n :: nat
assumes
  module-m: SCF-result.electoral-module m and
  profile: profile V A p and
  module-acc: SCF-result.electoral-module acc and
  defer-card-n: n = card (defer acc V A p)
shows well-formed-SCF A (loop-comp-helper acc m t V A p)
using assms
proof (induct arbitrary: acc rule: less-induct)
case (less)
have  $\forall m' n'.$ 
  (SCF-result.electoral-module m'  $\wedge$  SCF-result.electoral-module n')
   $\longrightarrow$  SCF-result.electoral-module (m'  $\triangleright$  n')
using seq-comp-sound
by metis
hence SCF-result.electoral-module (acc  $\triangleright$  m)
using less.premis module-m
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)  $\longrightarrow$ 
  well-formed-SCF A (loop-comp-helper acc m t V A p)
using less.hyps less.premis loop-comp-helper.simps(2)
  psubset-card-mono
by metis
moreover have well-formed-SCF A (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**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$   
**assumes** *SCF-result.electoral-module*  $m$   
**shows** *SCF-result.electoral-module*  $(m \circ_t)$   
**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**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$  **and**  
 $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $n :: \text{nat}$   
**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*:  $n = \text{card} (\text{defer } acc V A p)$   
**shows** *defer*  $(\text{loop-comp-helper } acc m t) V A p \subseteq \text{defer } acc V A p$   
**using** *assms*  
**proof** (*induct arbitrary: acc rule: less-induct*)  
**case** (*less*)  
**have** *emod-acc-m*: *SCF-result.electoral-module*  $(acc \triangleright m)$   
**using** *less.premis 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**  $\neg t (acc V A p) \wedge \text{defer } (acc \triangleright m) V A p \subset \text{defer } acc V A p \wedge$   
 $\text{finite } (\text{defer } acc V A p) \longrightarrow$   
 $\text{defer } (\text{loop-comp-helper } (acc \triangleright m) m t) V A p \subseteq \text{defer } acc V A p$   
**using** *emod-acc-m less.hyps less.premis*  
**by** *blast*  
**hence**  $\neg t (acc V A p) \wedge \text{defer } (acc \triangleright m) V A p \subset \text{defer } acc V A p \wedge$   
 $\text{finite } (\text{defer } acc V A p) \longrightarrow$   
 $\text{defer } (\text{loop-comp-helper } acc m t) V A p \subseteq \text{defer } acc V A p$   
**using** *loop-comp-helper.simps(2)*  
**by** *metis*

```

thus ?case
  using eq-iff loop-comp-code-helper
  by (metis (no-types))
qed

```

### 6.5.3 Lemmas

**lemma** *loop-comp-helper-def-lift-inv-helper*:

```

fixes
   $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $t :: 'a \text{ Termination-Condition}$  and
   $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $n :: \text{nat}$ 
assumes
  monotone-m: defer-lift-invariance m and
  prof: profile V A p and
  dli-acc: defer-lift-invariance acc and
  card-n-defer:  $n = \text{card} (\text{defer } acc \ V \ A \ p)$  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  $\longrightarrow$ 
   $(\forall q \ a. a \in (\text{defer } (acc \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$ 
   $\text{card } (\text{defer } (acc \triangleright m) \ V \ A \ p) = \text{card } (\text{defer } (acc \triangleright m) \ V \ A \ q))$ 
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  $\longrightarrow$ 
   $(\forall q \ a. (a \in (\text{defer } (acc \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a) \longrightarrow$ 
   $\text{card } (\text{defer } acc \ V \ A \ p) = \text{card } (\text{defer } acc \ V \ A \ q))$ 
using assms seq-comp-def-set-trans
unfolding defer-lift-invariance-def
by metis
thus ?case
proof (cases)

```

**assume** *card-unchanged*:  $\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) = \text{card } (\text{defer } \text{acc} \ V \ A \ p)$   
**have** *defer-lift-invariance*  $\text{acc} \longrightarrow$   
 $(\forall \ q \ a. \ a \in (\text{defer } \text{acc} \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$   
 $(\text{loop-comp-helper } \text{acc} \ m \ t) \ V \ A \ q = \text{acc } V \ A \ q)$   
**proof** (*safe*)  
**fix**  
 $q :: ('a, 'v) \text{ Profile}$  **and**  
 $a :: 'a$   
**assume**  
 $\text{dli-acc}: \text{defer-lift-invariance } \text{acc}$  **and**  
 $\text{a-in-def-acc}: a \in \text{defer } \text{acc} \ V \ A \ p$  **and**  
 $\text{lifted-A}: \text{Profile.lifted } V \ A \ p \ q \ a$   
**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*  $\text{acc}$   
**using** *dli-acc*  
**unfolding** *defer-lift-invariance-def*  
**by** *simp*  
**moreover have** *acc-eq-pq*:  $\text{acc } V \ A \ q = \text{acc } V \ A \ p$   
**using** *a-in-def-acc dli-acc lifted-A*  
**unfolding** *defer-lift-invariance-def*  
**by** (*metis (full-types)*)  
**ultimately have** *finite* ( $\text{defer } \text{acc} \ V \ A \ p$ )  
 $\longrightarrow \text{loop-comp-helper } \text{acc} \ m \ t \ V \ A \ q = \text{acc } V \ A \ q$   
**using** *card-unchanged defer-card-comp prof loop-comp-code-helper*  
 $\text{psubset-card-mono dual-order.strict-iff-order}$   
 $\text{seq-comp-def-set-bounded less}$   
**by** (*metis (mono-tags, lifting)*)  
**thus**  $\text{loop-comp-helper } \text{acc} \ m \ t \ V \ A \ q = \text{acc } V \ A \ q$   
**using** *acc-eq-pq loop-comp-code-helper*  
**by** (*metis (full-types)*)  
**qed**  
**moreover from** *card-unchanged*  
**have**  $(\text{loop-comp-helper } \text{acc} \ m \ t) \ V \ A \ p = \text{acc } V \ A \ p$   
**using** *loop-comp-code-helper order.strict-iff-order psubset-card-mono*  
**by** *metis*  
**ultimately have**  
 $\text{defer-lift-invariance } (\text{acc} \triangleright m) \wedge \text{defer-lift-invariance } \text{acc} \longrightarrow$   
 $(\forall \ q \ a. \ a \in (\text{defer } (\text{loop-comp-helper } \text{acc} \ m \ t) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a$   
 $\longrightarrow$   
 $(\text{loop-comp-helper } \text{acc} \ m \ t) \ V \ A \ p = (\text{loop-comp-helper } \text{acc} \ m \ t) \ V$   
 $A \ q)$   
**unfolding** *defer-lift-invariance-def*  
**by** *metis*  
**moreover have** *defer-lift-invariance* ( $\text{acc} \triangleright m$ )  
**using** *less monotone-m seq-comp-presv-def-lift-inv*

```

    by simp
  ultimately show ?thesis
    using less monotone-m
    by metis
next
  assume card-changed:  $\neg (\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) = \text{card } (\text{defer } \text{acc} \ V \ A \ p))$ 
  with prof
  have card-smaller-for-p:
    SCF-result.electoral-module  $\text{acc} \wedge \text{finite } A \longrightarrow$ 
     $\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) < \text{card } (\text{defer } \text{acc} \ V \ A \ p)$ 
  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  $\text{acc} \longrightarrow$ 
     $(\forall \ q \ a. a \in (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$ 
     $\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ q) < \text{card } (\text{defer } \text{acc} \ V \ A \ q))$ 
  using lifted-def less
  unfolding defer-lift-invariance-def
  by (metis (no-types, lifting))
  thus ?thesis
  proof (cases)
    assume t-not-satisfied-for-p:  $\neg t (\text{acc} \ V \ A \ p)$ 
    hence t-not-satisfied-for-q:
      defer-lift-invariance  $\text{acc} \longrightarrow$ 
       $(\forall \ q \ a. a \in (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow \neg t (\text{acc} \ V \ A \ q))$ 
    using monotone-m prof seq-comp-def-set-trans
    unfolding defer-lift-invariance-def
    by metis
    have dli-card-def:
      defer-lift-invariance  $(\text{acc} \triangleright m) \wedge \text{defer-lift-invariance } \text{acc} \longrightarrow$ 
       $(\forall \ q \ a. a \in (\text{defer } (\text{acc} \triangleright m) \ V \ A \ p) \wedge \text{Profile.lifted } V \ A \ p \ q \ a \longrightarrow$ 
       $\text{card } (\text{defer } (\text{acc} \triangleright m) \ V \ A \ q) \neq (\text{card } (\text{defer } \text{acc} \ V \ A \ q)))$ 
    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-lift-invariance (acc ▷ m) ∧ defer-lift-invariance acc ⟶
    (∀ p' a. a ∈ (defer (acc ▷ m) V A p) ∧ lifted V A p p' a ⟶
      defer (acc ▷ m) V A p' ⊆ defer acc V A p')
  using Profile.lifted-def dli-card-def defer-lift-invariance-def
    monotone-m psubsetI seq-comp-def-set-bounded
  by (metis (no-types, opaque-lifting))
with t-not-satisfied-for-p
have rec-step-q:
  defer-lift-invariance (acc ▷ m) ∧ defer-lift-invariance acc ⟶
    (∀ q a. a ∈ (defer (acc ▷ m) V A p) ∧ lifted V A p q a ⟶
      loop-comp-helper acc m t V A q = loop-comp-helper (acc ▷ m) m t V
A q)
proof (safe)
  fix
  q :: ('a, 'v) Profile and
  a :: 'a
  assume
  a-in-def-impl-def-subset:
    ∀ q' a'. a' ∈ defer (acc ▷ m) V A p ∧ lifted V A p q' a' ⟶
      defer (acc ▷ m) V A q' ⊆ defer acc V A q' and
  dli-acc: defer-lift-invariance acc and
  a-in-def-seq-acc-m: a ∈ defer (acc ▷ m) V A p and
  lifted-pq-a: lifted V A p q a
  hence defer (acc ▷ m) V A q ⊆ defer acc V A q
  by metis
  moreover have SCF-result.electoral-module acc
  using dli-acc
  unfolding defer-lift-invariance-def
  by simp
  moreover have ¬ t (acc V A q)
  using dli-acc a-in-def-seq-acc-m lifted-pq-a t-not-satisfied-for-q
  by metis
  ultimately show loop-comp-helper acc m t V A q
    = loop-comp-helper (acc ▷ m) m t V A q
  using loop-comp-code-helper defer-in-alts finite-subset lifted-pq-a
  unfolding lifted-def
  by (metis (mono-tags, lifting))
qed
have rec-step-p:
  SCF-result.electoral-module acc ⟶
    loop-comp-helper acc m t V A p = loop-comp-helper (acc ▷ m) m t V A p
proof (safe)

```

```

assume emod-acc: SCF-result.electoral-module acc
have sound-imp-defer-subset:
  SCF-result.electoral-module  $m \longrightarrow \text{defer } (acc \triangleright m) \ V \ A \ p \subseteq \text{defer } acc \ V$ 
A p
  using emod-acc prof seq-comp-def-set-bounded
  by blast
hence card-ineq:  $\text{card } (\text{defer } (acc \triangleright m) \ V \ A \ p) < \text{card } (\text{defer } acc \ V \ A \ p)$ 
  using card-changed card-mono less order-neq-le-trans
  unfolding defer-lift-invariance-def
  by metis
have def-limited-acc:
  profile  $V \ (\text{defer } acc \ V \ A \ p) \ (\text{limit-profile } (\text{defer } acc \ V \ A \ p) \ p)$ 
  using def-presv-prof emod-acc prof
  by metis
have  $\text{defer } (acc \triangleright m) \ V \ A \ p \subseteq \text{defer } acc \ V \ A \ p$ 
  using sound-imp-defer-subset defer-lift-invariance-def monotone-m
  by blast
hence  $\text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p$ 
  using def-limited-acc card-ineq card-psubset less
  by metis
with def-limited-acc
show  $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V$ 
A p
  using loop-comp-code-helper t-not-satisfied-for-p less
  by (metis (no-types))
qed
show ?thesis
proof (safe)
  fix
     $q :: ('a, 'v) \text{ Profile}$  and
     $a :: 'a$ 
  assume
    a-in-defer-lch:  $a \in \text{defer } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p$  and
    a-lifted:  $\text{Profile.lifted } V \ A \ p \ q \ a$ 
  have mod-acc: SCF-result.electoral-module acc
  using less.premis
  unfolding defer-lift-invariance-def
  by simp
hence loop-comp-equiv:
   $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ p$ 
  using rec-step-p
  by blast
hence  $a \in \text{defer } (\text{loop-comp-helper } (acc \triangleright m) \ m \ t) \ V \ A \ p$ 
  using a-in-defer-lch
  by presburger
moreover have l-inv:  $\text{defer-lift-invariance } (acc \triangleright m)$ 
using less.premis monotone-m voters-determine-m seq-comp-presv-def-lift-inv[of
acc m]
  by blast

```

```

ultimately have  $a \in \text{defer } (acc \triangleright m) \ V \ A \ p$ 
  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
   $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = \text{loop-comp-helper } acc \ m \ t \ V \ A \ q$ 
proof -
  assume
    dli-acc-seq-m: defer-lift-invariance (acc  $\triangleright$  m) and
    a-in-def-seq: a  $\in$  defer (acc  $\triangleright$  m) V A p
  moreover from this have SCF-result.electoral-module (acc  $\triangleright$  m)
    unfolding defer-lift-invariance-def
    by blast
  moreover have  $a \in \text{defer } (\text{loop-comp-helper } (acc \triangleright m) \ m \ t) \ V \ A \ p$ 
    using loop-comp-equiv a-in-defer-lch
    by presburger
  ultimately have
     $\text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ p$ 
     $= \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ q$ 
    using monotone-m mod-acc less a-lifted card-smaller-for-p
    defer-in-alts infinite-super less
    unfolding lifted-def
    by (metis (no-types))
  moreover have  $\text{loop-comp-helper } acc \ m \ t \ V \ A \ q$ 
     $= \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ q$ 
    using dli-acc-seq-m a-in-def-seq less a-lifted rec-step-q
    by blast
  ultimately show ?thesis
    using loop-comp-equiv
    by presburger
qed
qed
next
  assume  $\neg \neg t \ (acc \ V \ A \ p)$ 
  thus ?thesis
    using loop-comp-code-helper less
    unfolding defer-lift-invariance-def
    by metis
qed
qed
qed

lemma loop-comp-helper-def-lift-inv:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $t :: 'a \text{ Termination-Condition}$  and
     $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and

```



```

  p :: ('a, 'v) Profile and
  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 :: ('a, 'v) Profile and
  q :: ('a, 'v) Profile and
  a :: 'a
assumes lifted V A p q a
shows finite-profile V A p
using assms
unfolding lifted-def
by simp

lemma loop-comp-helper-presv-def-lift-inv:
fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  t :: 'a Termination-Condition and
  acc :: ('a, 'v, 'a Result) Electoral-Module
assumes
  defer-lift-invariance m and
  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 :: ('a, 'v) Profile and

```

```

  q :: ('a, 'v) Profile and
  a :: 'a
assume
  a ∈ defer (loop-comp-helper acc m t) V A p and
  lifted V A p q a
thus loop-comp-helper acc m t V A p = loop-comp-helper acc m t V A q
  using lifted-imp-fin-prof loop-comp-helper-def-lift-inv assms
  by metis
qed

lemma loop-comp-presv-non-electing-helper:
fixes
  m :: ('a, 'v, 'a Result) Electoral-Module and
  t :: 'a Termination-Condition and
  acc :: ('a, 'v, 'a Result) Electoral-Module and
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  n :: nat
assumes
  non-electing-m: non-electing m and
  non-electing-acc: non-electing acc and
  prof: profile V A p and
  acc-defer-card: n = card (defer acc V A p)
shows elect (loop-comp-helper acc m t) V A p = {}
using acc-defer-card non-electing-acc
proof (induct n arbitrary: acc rule: less-induct)
case (less n)
thus ?case
proof (safe)
  fix x :: 'a
  assume
    acc-no-elect:
      (∧ i acc'. i < card (defer acc V A p) ⇒
        i = card (defer acc' V A p) ⇒ non-electing acc' ⇒
        elect (loop-comp-helper acc' m t) V A p = {}) and
    acc-non-elect: non-electing acc and
    x-in-acc-elect: x ∈ elect (loop-comp-helper acc m t) V A p
  have ∀ m' n'. non-electing m' ∧ non-electing n' ⇒ non-electing (m' ▷ n')
    by simp
  hence seq-acc-m-non-elect: non-electing (acc ▷ m)
    using acc-non-elect non-electing-m
    by blast
  have ∀ i m'.
    i < card (defer acc V A p) ∧ i = card (defer m' V A p) ∧
    non-electing m' ⇒
    elect (loop-comp-helper m' m t) V A p = {}
  using acc-no-elect
  by blast

```

**hence**  $\forall m'$ .  
 $\text{finite } (\text{defer } \text{acc } V A p) \wedge \text{defer } m' V A p \subset \text{defer } \text{acc } V A p \wedge$   
 $\text{non-electing } m' \longrightarrow$   
 $\text{elect } (\text{loop-comp-helper } m' m t) V A p = \{\}$   
**using** *psubset-card-mono*  
**by** *metis*  
**hence**  $\neg t (\text{acc } V A p) \wedge \text{defer } (\text{acc } \triangleright m) V A p \subset \text{defer } \text{acc } V A p \wedge$   
 $\text{finite } (\text{defer } \text{acc } V A p) \longrightarrow$   
 $\text{elect } (\text{loop-comp-helper } \text{acc } m t) V A p = \{\}$   
**using** *loop-comp-code-helper seq-acc-m-non-elect*  
**by** (*metis (no-types)*)  
**moreover have**  $\text{elect } \text{acc } V A p = \{\}$   
**using** *acc-non-elect prof non-electing-def*  
**by** *blast*  
**ultimately show**  $x \in \{\}$   
**using** *loop-comp-code-helper x-in-acc-elect*  
**by** (*metis (no-types)*)  
**qed**  
**qed**

**lemma** *loop-comp-helper-iter-elim-def-n-helper:*  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$  **and**  
 $\text{acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $n :: \text{nat}$  **and**  
 $x :: \text{nat}$   
**assumes**  
 $\text{non-electing-}m$ :  $\text{non-electing } m$  **and**  
 $\text{single-elimination}$ :  $\text{eliminates } 1 m$  **and**  
 $\text{terminate-if-}n\text{-left}$ :  $\forall r. t r = (\text{card } (\text{defer-}r r) = x)$  **and**  
 $\text{x-greater-zero}$ :  $x > 0$  **and**  
 $\text{prof}$ :  $\text{profile } V A p$  **and**  
 $\text{n-acc-defer-card}$ :  $n = \text{card } (\text{defer } \text{acc } V A p)$  **and**  
 $\text{n-ge-x}$ :  $n \geq x$  **and**  
 $\text{def-card-gt-one}$ :  $\text{card } (\text{defer } \text{acc } V A p) > 1$  **and**  
 $\text{acc-nonelect}$ :  $\text{non-electing } \text{acc}$   
**shows**  $\text{card } (\text{defer } (\text{loop-comp-helper } \text{acc } m t) V A p) = x$   
**using**  $\text{n-ge-x def-card-gt-one acc-nonelect n-acc-defer-card}$   
**proof** (*induct n arbitrary: acc rule: less-induct*)  
**case** (*less n*)  
**have**  $\text{mod-acc}$ :  $\text{SCF-result.electoral-module acc}$   
**using** *less*  
**unfolding** *non-electing-def*  
**by** *metis*

**hence** *step-reduces-defer-set*:  $\text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p$   
**using** *seq-comp-elim-one-red-def-set single-elimination prof less*  
**by** *metis*  
**thus** *?case*  
**proof** (*cases*  $t \ (acc \ V \ A \ p)$ )  
**case** *True*  
**assume** *term-satisfied*:  $t \ (acc \ V \ A \ p)$   
**thus**  $\text{card } (\text{defer-r } (\text{loop-comp-helper } acc \ m \ t \ V \ A \ p)) = x$   
**using** *loop-comp-code-helper term-satisfied terminate-if-n-left*  
**by** *metis*  
**next**  
**case** *False*  
**hence** *card-not-eq-x*:  $\text{card } (\text{defer } acc \ V \ A \ p) \neq x$   
**using** *terminate-if-n-left*  
**by** *metis*  
**have** *fin-def-acc*:  $\text{finite } (\text{defer } acc \ V \ A \ p)$   
**using** *prof mod-acc less card.infinite not-one-less-zero*  
**by** *metis*  
**hence** *rec-step*:  
 $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ p$   
**using** *False step-reduces-defer-set*  
**by** *simp*  
**have** *card-too-big*:  $\text{card } (\text{defer } acc \ V \ A \ p) > x$   
**using** *card-not-eq-x dual-order.order-iff-strict less*  
**by** *simp*  
**hence** *enough-leftover*:  $\text{card } (\text{defer } acc \ V \ A \ p) > 1$   
**using** *x-greater-zero*  
**by** *simp*  
**obtain**  $k$  **where**  
 $\text{new-card-k}: k = \text{card } (\text{defer } (acc \triangleright m) \ V \ A \ p)$   
**by** *metis*  
**have**  $\text{defer } acc \ V \ A \ p \subseteq A$   
**using** *defer-in-alts prof mod-acc*  
**by** *metis*  
**hence** *step-profile*:  $\text{profile } V \ (\text{defer } acc \ V \ A \ p) \ (\text{limit-profile } (\text{defer } acc \ V \ A \ p) \ p)$   
**using** *prof limit-profile-sound*  
**by** *metis*  
**hence**  
 $\text{card } (\text{defer } m \ V \ (\text{defer } acc \ V \ A \ p) \ (\text{limit-profile } (\text{defer } acc \ V \ A \ p) \ p)) =$   
 $\text{card } (\text{defer } acc \ V \ A \ p) - 1$   
**using** *enough-leftover non-electing-m*  
*single-elimination single-elim-decr-def-card-2*  
**by** *blast*  
**hence** *k-card*:  $k = \text{card } (\text{defer } acc \ V \ A \ p) - 1$   
**using** *mod-acc prof new-card-k non-electing-m seq-comp-defers-def-set*  
**by** *metis*  
**hence** *new-card-still-big-enough*:  $x \leq k$   
**using** *card-too-big*

```

    by linarith
  show ?thesis
proof (cases  $x < k$ )
  case True
  hence  $1 < \text{card } (\text{defer } (acc \triangleright m) \ V \ A \ p)$ 
    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  $SCF\text{-result.electoral-module } (acc \triangleright m)$ 
    using mod-acc eliminates-def seq-comp-sound single-elimination
    by metis
  moreover have  $\text{non-electing } (acc \triangleright m)$ 
    using less non-electing-m
    by simp
  ultimately have  $\text{card } (\text{defer } (\text{loop-comp-helper } (acc \triangleright m) \ m \ t) \ V \ A \ p) = x$ 
    using new-card-k new-card-still-big-enough less
    by metis
  thus ?thesis
    using rec-step
    by presburger
next
  case False
  thus ?thesis
    using dual-order.strict-iff-order new-card-k
      new-card-still-big-enough rec-step
      terminate-if-n-left
    by simp
qed
qed
qed

```

**lemma** *loop-comp-helper-iter-elim-def-n:*  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $t :: 'a \text{ Termination-Condition}$  **and**  
 $acc :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $x :: \text{nat}$   
**assumes**  
 $\text{non-electing } m$  **and**  
 $\text{eliminates } 1 \ m$  **and**  
 $\forall r. (t \ r) = (\text{card } (\text{defer-r } r) = x)$  **and**  
 $x > 0$  **and**  
 $\text{profile } V \ A \ p$  **and**

```

    card (defer acc V A p) ≥ x and
    non-electing acc
  shows card (defer (loop-comp-helper acc m t) V A p) = x
  using assms gr-implies-not0 le-neq-implies-less less-one linorder-neqE-nat nat-neq-iff
    less-le loop-comp-helper-iter-elim-def-n-helper loop-comp-code-helper
  by (metis (no-types, lifting))

lemma iter-elim-def-n-helper:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    x :: nat
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left: ∀ r. (t r) = (card (defer-r r) = x) and
    x-greater-zero: x > 0 and
    prof: profile V A p and
    enough-alternatives: card A ≥ x
  shows card (defer (m ∘t) V A p) = x
  proof (cases)
    assume card A = x
    thus ?thesis
      using terminate-if-n-left
      by simp
  next
    assume card-not-x: ¬ card A = x
    thus ?thesis
    proof (cases)
      assume card A < x
      thus ?thesis
        using enough-alternatives not-le
        by blast
    next
      assume ¬ card A < x
      hence card A > x
        using card-not-x
        by linarith
      moreover from this
      have card (defer m V A p) = card A - 1
        using non-electing-m single-elimination single-elim-decr-def-card-2
        prof x-greater-zero
        by fastforce
      ultimately have card (defer m V A p) ≥ x
        by linarith
      moreover have (m ∘t) V A p = (loop-comp-helper m m t) V A p

```

```

    using card-not-x terminate-if-n-left
    by simp
  ultimately show ?thesis
    using non-electing-m prof single-elimination terminate-if-n-left x-greater-zero
      loop-comp-helper-iter-elim-def-n
    by metis
qed
qed

```

#### 6.5.4 Composition Rules

The loop composition preserves defer-lift-invariance.

```

theorem loop-comp-presv-def-lift-inv[simp]:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition
  assumes defer-lift-invariance m and 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 set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and
    a :: 'a
  assume
    a  $\in$  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 Result) Electoral-Module and
    t :: 'a 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 set and
  V :: 'v set and
  p :: ('a, 'v) Profile and
  a :: 'a
assume
  profile V A p and
  a  $\in$  elect (m  $\circ_t$ ) 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))
qed

theorem iter-elim-def-n[simp]:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    n :: nat
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left:  $\forall r. t\ r = (\text{card } (\text{defer-r } r) = n)$  and
    x-greater-zero:  $n > 0$ 
  shows defers n (m  $\circ_t$ )
proof (unfold defers-def, safe)
  show SCF-result.electoral-module (m  $\circ_t$ )
    using loop-comp-sound non-electing-m
    unfolding non-electing-def
    by metis
next
fix
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile
assume
  n  $\leq$  card A and
  finite A and

```



```

    profile V A p
  thus card (defer (m  $\circ_t$ ) 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 == maximum-parallel-composition m n

```

### 6.6.2 Soundness

```

theorem max-par-comp-sound:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module
  assumes

```

$SCF\text{-result.electoral-module } m$  **and**  
 $SCF\text{-result.electoral-module } n$   
**shows**  $SCF\text{-result.electoral-module } (m \parallel_{\uparrow} n)$   
**using**  $assms \text{ max-agg-sound par-comp-sound}$   
**unfolding**  $maximum\text{-parallel-composition.simps}$   
**by**  $metis$

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

### 6.6.3 Lemmas

**lemma**  $max\text{-agg-eq-result}$ :  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $a :: 'a$   
**assumes**  
 $module\text{-}m: SCF\text{-result.electoral-module } m$  **and**  
 $module\text{-}n: SCF\text{-result.electoral-module } n$  **and**  
 $prof\text{-}p: \text{profile } V \ A \ p$  **and**  
 $a\text{-in-}A: a \in A$   
**shows**  $mod\text{-contains-result } (m \parallel_{\uparrow} n) \ m \ V \ A \ p \ a \ \vee$   
 $mod\text{-contains-result } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ a$   
**proof** ( $cases$ )  
**assume**  $a\text{-elect}: a \in \text{elect } (m \parallel_{\uparrow} n) \ V \ A \ p$   
**hence**  $\text{let } (e, r, d) = m \ V \ A \ p;$   
 $(e', r', d') = n \ V \ A \ p \text{ in}$   
 $a \in e \cup e'$   
**by**  $auto$   
**hence**  $a \in (\text{elect } m \ V \ A \ p) \cup (\text{elect } n \ V \ A \ p)$   
**by**  $auto$   
**moreover have**  
 $\forall m' n' V' A' p' a'. \ mod\text{-contains-result } m' \ n' \ V' \ A' \ p' \ (a'::'a) =$

```

    (SCF-result.electoral-module m'
     ∧ SCF-result.electoral-module n'
     ∧ profile V' A' p' ∧ a' ∈ A'
     ∧ (a' ∉ elect m' V' A' p' ∨ a' ∈ elect n' V' A' p')
     ∧ (a' ∉ reject m' V' A' p' ∨ a' ∈ reject n' V' A' p')
     ∧ (a' ∉ defer m' V' A' p' ∨ a' ∈ defer n' V' A' p'))
  unfolding mod-contains-result-def
  by simp
moreover have module-mn: SCF-result.electoral-module (m ||↑ n)
  using module-m module-n max-par-comp-sound
  by metis
moreover have a ∉ defer (m ||↑ n) V A p
  using module-mn IntI a-elect empty-iff prof-p result-disj
  by (metis (no-types))
moreover have a ∉ reject (m ||↑ 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 ∉ elect (m ||↑ n) V A p
thus ?thesis
proof (cases)
  assume a-in-def: a ∈ defer (m ||↑ n) V A p
  thus ?thesis
  proof (safe)
    assume not-mod-cont-mn: ¬ mod-contains-result (m ||↑ n) n V A p a
    have par-emod: ∀ m' n'.
      SCF-result.electoral-module m' ∧
      SCF-result.electoral-module n' ⟶
      SCF-result.electoral-module (m' ||↑ n')
    using max-par-comp-sound
    by blast
    have set-intersect: ∀ a' A' A''. (a' ∈ A' ∩ A'') = (a' ∈ A' ∧ a' ∈ A'')
    by blast
    have wf-n: well-formed-SCF A (n V A p)
    using prof-p module-n
    unfolding SCF-result.electoral-module.simps
    by blast
    have wf-m: well-formed-SCF A (m V A p)
    using prof-p module-m
    unfolding SCF-result.electoral-module.simps
    by blast
    have e-mod-par: SCF-result.electoral-module (m ||↑ n)
    using par-emod module-m module-n
    by blast
    hence SCF-result.electoral-module (m ||m ax-aggregator n)
    by simp
  end
end

```

**hence** *result-disj-max*:  
 $\text{elect } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p \cap$   
 $\text{reject } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p = \{\} \wedge$   
 $\text{elect } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p \cap$   
 $\text{defer } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p = \{\} \wedge$   
 $\text{reject } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p \cap$   
 $\text{defer } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p = \{\}$   
**using** *prof-p result-disj*  
**by** *metis*  
**have** *a-not-elect*:  $a \notin \text{elect } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p$   
**using** *result-disj-max a-in-def*  
**by** *force*  
**have** *result-m*:  $(\text{elect } m \ V \ A \ p, \text{reject } m \ V \ A \ p, \text{defer } m \ V \ A \ p) = m \ V \ A \ p$   
**by** *auto*  
**have** *result-n*:  $(\text{elect } n \ V \ A \ p, \text{reject } n \ V \ A \ p, \text{defer } n \ V \ A \ p) = n \ V \ A \ p$   
**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**  $a \notin \text{elect } (m \parallel_m \text{ax-aggregator } n) \ V \ A \ p$   
**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-def*  
**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*:  
 $\text{max-aggregator } A \ (\text{elect } m \ V \ A \ p, \text{reject } m \ V \ A \ p, \text{defer } m \ V \ A \ p)$   
 $(\text{elect } n \ V \ A \ p, \text{reject } n \ V \ A \ p, \text{defer } n \ V \ A \ p) = (m \parallel_m \text{ax-aggregator } n)$   
 $V \ A \ p$   
**by** *simp*  
**have** *well-f-max*:  
 $\forall \ r' \ r'' \ e' \ e'' \ d' \ d'' \ A'.$

$well\text{-}formed\text{-}SCF\ A' (e', r', d') \wedge$   
 $well\text{-}formed\text{-}SCF\ A' (e'', r'', d'') \longrightarrow$   
 $reject\text{-}r\ (max\text{-}aggregator\ A' (e', r', d') (e'', r'', d'')) = r' \cap r''$   
**using** *max-agg-rej-set*  
**by** *metis*  
**have** *e-mod-disj*:  
 $\forall\ m' (V'::'v\ set)\ (A'::'a\ set)\ p'.$   
 $SCF\text{-}result.electoral\text{-}module\ m' \wedge profile\ V'\ A'\ p'$   
 $\longrightarrow elect\ m'\ V'\ A'\ p' \cup reject\ m'\ V'\ A'\ p' \cup defer\ m'\ V'\ A'\ p' = A'$   
**using** *result-presv-alts*  
**by** *blast*  
**hence** *e-mod-disj-n*:  $elect\ n\ V\ A\ p \cup reject\ n\ V\ A\ p \cup defer\ n\ V\ A\ p = A$   
**using** *prof-p module-n*  
**by** *metis*  
**have**  $\forall\ m'\ n'\ A'\ V'\ p' (b::'a).$   
 $mod\text{-}contains\text{-}result\ m'\ n'\ V'\ A'\ p'\ b =$   
 $(SCF\text{-}result.electoral\text{-}module\ m'$   
 $\wedge SCF\text{-}result.electoral\text{-}module\ n'$   
 $\wedge profile\ V'\ A'\ p' \wedge b \in A'$   
 $\wedge (b \in elect\ m'\ V'\ A'\ p' \longrightarrow b \in elect\ n'\ V'\ A'\ p')$   
 $\wedge (b \in reject\ m'\ V'\ A'\ p' \longrightarrow b \in reject\ n'\ V'\ A'\ p')$   
 $\wedge (b \in defer\ m'\ V'\ A'\ p' \longrightarrow b \in defer\ n'\ V'\ A'\ p'))$   
**unfolding** *mod-contains-result-def*  
**by** *simp*  
**hence**  $a \notin defer\ n\ V\ A\ p$   
**using** *a-not-mpar-rej a-in-A e-mod-par module-n not-a-elect*  
 $not\text{-}mod\text{-}cont\text{-}mn\ prof\text{-}p$   
**by** *blast*  
**hence**  $a \in reject\ n\ V\ A\ p$   
**using** *a-in-A a-not-elect-mn module-n not-rej-imp-elec-or-defer prof-p*  
**by** *metis*  
**hence**  $a \notin reject\ m\ V\ A\ p$   
**using** *well-f-max max-agg-res result-m result-n set-intersect*  
 $wf\text{-}m\ wf\text{-}n\ a\text{-not-mpar-rej}$   
**unfolding** *maximum-parallel-composition.simps*  
**by** *(metis (no-types))*  
**hence**  $a \notin defer\ (m \parallel_{\uparrow} n)\ V\ A\ p \vee a \in defer\ m\ V\ A\ p$   
**using** *e-mod-disj prof-p a-in-A module-m a-not-elect-mn*  
**by** *blast*  
**thus**  $mod\text{-}contains\text{-}result\ (m \parallel_{\uparrow} n)\ m\ V\ A\ p\ a$   
**using** *a-not-mpar-rej mod-cont-res-fg e-mod-par prof-p a-in-A*  
 $module\text{-}m\ a\text{-not-elect}$   
**unfolding** *maximum-parallel-composition.simps*  
**by** *metis*  
**qed**  
**next**  
**assume** *not-a-defer*:  $a \notin defer\ (m \parallel_{\uparrow} n)\ V\ A\ p$   
**have** *el-rej-defer*:  $(elect\ m\ V\ A\ p, reject\ m\ V\ A\ p, defer\ m\ V\ A\ p) = m\ V\ A\ p$   
**by** *auto*

**from** *not-a-elect not-a-defer*  
**have** *a-reject*:  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$   
**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**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **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*  $n$   
**shows**  $(a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p)$   
 $= (a \in \text{reject } m \ V \ A \ p \wedge a \in \text{reject } n \ V \ A \ p)$   
**proof**  
**assume** *rej-a*:  $a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p$   
**hence** *case*  $n \ V \ A \ p$  *of*  $(e, r, d) \Rightarrow$   
 $a \in \text{reject-r } (\text{max-aggregator } A$   
 $(\text{elect } m \ V \ A \ p, \text{reject } m \ V \ A \ p, \text{defer } m \ V \ A \ p) \ (e, r, d))$   
**by** *auto*  
**hence** *case snd*  $(m \ V \ A \ p)$  *of*  $(r, d) \Rightarrow$

$\text{case } n \ V \ A \ p \text{ of } (e', r', d') \Rightarrow$   
 $a \in \text{reject-}r \ (\text{max-aggregator } A \ (\text{elect } m \ V \ A \ p, r, d) \ (e', r', d'))$   
**by force**  
**with rej-a**  
**have**  $\text{let } (e, r, d) = m \ V \ A \ p;$   
 $(e', r', d') = n \ V \ A \ p \text{ in}$   
 $a \in \text{reject-}r \ (\text{max-aggregator } A \ (e, r, d) \ (e', r', d'))$   
**unfolding prod.case-eq-if**  
**by simp**  
**hence**  $\text{let } (e, r, d) = m \ V \ A \ p;$   
 $(e', r', d') = n \ V \ A \ p \text{ in}$   
 $a \in A - (e \cup e' \cup d \cup d')$   
**by simp**  
**hence**  
 $a \in A - (\text{elect } m \ V \ A \ p \cup \text{elect } n \ V \ A \ p \cup \text{defer } m \ V \ A \ p \cup \text{defer } n \ V \ A \ p)$   
**by auto**  
**thus**  $a \in \text{reject } m \ V \ A \ p \wedge a \in \text{reject } n \ V \ A \ p$   
**using** *Diff-iff Un-iff electoral-mod-defer-elem assms*  
**by metis**  
**next**  
**assume**  $a \in \text{reject } m \ V \ A \ p \wedge a \in \text{reject } n \ V \ A \ p$   
**moreover from this**  
**have**  $a \notin \text{elect } m \ V \ A \ p \wedge a \notin \text{defer } m \ V \ A \ p$   
 $\wedge a \notin \text{elect } n \ V \ A \ p \wedge a \notin \text{defer } n \ V \ A \ p$   
**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*  
 $\text{reject-not-elec-or-def assms}$   
**by (metis (no-types))**  
**qed**

**lemma** *max-agg-rej-fst-imp-seq-contained:*

**fixes**

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

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

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

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

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

$a :: 'a$

**assumes**

*f-prof: finite-profile*  $V \ A \ p$  **and**

*module-m: SCF-result.electoral-module*  $m$  **and**

*module-n: SCF-result.electoral-module*  $n$  **and**

*rejected: a ∈ reject n V A p*

**shows** *mod-contains-result*  $m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$

**using** *assms*

**proof** (*unfold mod-contains-result-def, safe*)

**show** *SCF-result.electoral-module*  $(m \parallel_{\uparrow} n)$

```

    using module-m module-n max-par-comp-sound
    by metis
next
  show  $a \in A$ 
    using f-prof module-n rejected reject-in-alts
    by blast
next
  assume a-in-elect:  $a \in \text{elect } m \ V \ A \ p$ 
  hence a-not-reject:  $a \notin \text{reject } m \ V \ A \ p$ 
    using disjoint-iff-not-equal f-prof module-m result-disj
    by metis
  have reject n  $V \ A \ p \subseteq A$ 
    using f-prof module-n
    by (simp add: reject-in-alts)
  hence  $a \in A$ 
    using in-mono rejected
    by metis
  with a-in-elect a-not-reject
  show  $a \in \text{elect } (m \parallel_{\uparrow} n) \ V \ A \ p$ 
    using f-prof max-agg-eq-result module-m module-n rejected
      max-agg-rej-iff-both-reject mod-contains-result-comm
      mod-contains-result-def
    by metis
next
  assume  $a \in \text{reject } m \ V \ A \ p$ 
  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
     $\forall m' A' V' p'. \quad$ 
     $\text{SCF-result.electoral-module } m' \wedge \text{finite } A' \wedge \text{finite } V' \wedge \text{profile } V' \ A' \ p' \wedge$ 
     $\longrightarrow \text{elect } m' \ V' \ A' \ p' \cup \text{reject } m' \ V' \ A' \ p' \cup \text{defer } m' \ V' \ A' \ p' = A'$ 
    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
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  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  $m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$ 
    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
    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 :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    a :: 'a
  assumes
    f-prof: finite-profile V A p and
    module-m: SCF-result.electoral-module m and
    module-n: SCF-result.electoral-module n and
    rejected: a ∈ reject m V A p
  shows mod-contains-result n (m ||↑ n) V A p a
  using assms
proof (unfold mod-contains-result-def, safe)
  show SCF-result.electoral-module (m ||↑ n)
    using module-m module-n max-par-comp-sound
    by metis
next
  show a ∈ A
    using f-prof in-mono module-m reject-in-alts rejected
    by (metis (no-types))
next
  assume a ∈ elect n V A p
  thus a ∈ elect (m ||↑ n) V A p
    using max-aggregator.simps[of
      A elect m V A p reject m V A p defer m V A p
      elect n V A p reject n V A p defer n V A p]
    by simp
next
  assume a ∈ reject n V A p
  thus a ∈ reject (m ||↑ n) V A p
    using f-prof max-agg-rej-iff-both-reject module-m module-n rejected
    by metis
next
  assume a ∈ defer n V A p
  moreover have a ∈ A
    using f-prof max-agg-rej-fst-imp-seq-contained module-m rejected
    unfolding mod-contains-result-def
    by metis
  ultimately show a ∈ defer (m ||↑ n) V A p
    using disjoint-iff-not-equal max-agg-eq-result max-agg-rej-iff-both-reject
    f-prof mod-contains-result-comm mod-contains-result-def

```

```

      module-m module-n rejected result-disj
    by (metis (no-types, opaque-lifting))
qed

lemma max-agg-rej-snd-equiv-seq-contained:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    a :: 'a
  assumes
    finite-profile V A p and
    SCF-result.electoral-module m and
    SCF-result.electoral-module n and
    a ∈ reject m V A p
  shows mod-contains-result-sym (m ||↑ n) n V A p a
  using assms
proof (unfold mod-contains-result-sym-def, safe)
  assume a ∈ reject (m ||↑ n) V A p
  thus a ∈ reject n V A p
    using assms max-agg-rej-iff-both-reject
    by (metis (no-types))
next
  have mod-contains-result n (m ||↑ n) V A p a
    using assms max-agg-rej-snd-imp-seq-contained
    by (metis (full-types))
  thus
    a ∈ elect (m ||↑ n) V A p  $\implies$  a ∈ elect n V A p and
    a ∈ defer (m ||↑ n) V A p  $\implies$  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  $\implies$  a ∈ elect (m ||↑ n) V A p and
    a ∈ reject n V A p  $\implies$  a ∈ reject (m ||↑ n) V A p and
    a ∈ defer n V A p  $\implies$  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**

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

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

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

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

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

**assumes**

*SCF-result.electoral-module*  $m$  **and**

*SCF-result.electoral-module*  $n$  **and**

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

*finite*  $A$

**shows**  $\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p = (\text{reject } m \ V \ A \ p) \cap (\text{reject } n \ V \ A \ p)$

**proof** –

**have**  $A = (\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p)$

$\wedge A = (\text{elect } n \ V \ A \ p) \cup (\text{reject } n \ V \ A \ p) \cup (\text{defer } n \ V \ A \ p)$

**using** *assms result-presv-alts*

**by** *metis*

**hence**  $A - ((\text{elect } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p)) = (\text{reject } m \ V \ A \ p)$

$\wedge A - ((\text{elect } n \ V \ A \ p) \cup (\text{defer } n \ V \ A \ p)) = (\text{reject } n \ V \ A \ p)$

**using** *assms reject-not-elec-or-def*

**by** *fastforce*

**hence**

$A - ((\text{elect } m \ V \ A \ p) \cup (\text{elect } n \ V \ A \ p)$

$\cup (\text{defer } m \ V \ A \ p) \cup (\text{defer } n \ V \ A \ p))$

$= (\text{reject } m \ V \ A \ p) \cap (\text{reject } n \ V \ A \ p)$

**by** *blast*

**hence** *let*  $(e, r, d) = m \ V \ A \ p$ ;

$(e', r', d') = n \ V \ A \ p$  *in*

$A - (e \cup e' \cup d \cup d') = r \cap r'$

**by** *fastforce*

**thus** *?thesis*

**by** *auto*

qed

**lemma** *dcompat-dec-by-one-mod*:

**fixes**

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

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

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

$V :: 'v \text{ set}$  **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**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**assumes**  
 $\text{non-electing } m$  **and**  
 $\text{non-electing } n$   
**shows**  $\text{non-electing } (m \parallel_{\uparrow} n)$   
**using** *assms*  
**by** *simp*

Using the max aggregator, composing two compatible electoral modules in parallel preserves defer-lift-invariance.

**theorem** *par-comp-def-lift-inv[simp]*:  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**  
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   
**assumes**  
 $\text{compatible: disjoint-compatibility } m n$  **and**  
 $\text{monotone-m: defer-lift-invariance } m$  **and**  
 $\text{monotone-n: defer-lift-invariance } n$   
**shows**  $\text{defer-lift-invariance } (m \parallel_{\uparrow} n)$   
**proof** (*unfold defer-lift-invariance-def, safe*)  
**have**  $\text{mod-m: SCF-result.electoral-module } m$   
**using** *monotone-m*  
**unfolding** *defer-lift-invariance-def*  
**by** *simp*  
**moreover** **have**  $\text{mod-n: SCF-result.electoral-module } n$   
**using** *monotone-n*  
**unfolding** *defer-lift-invariance-def*  
**by** *simp*  
**ultimately show**  $\text{SCF-result.electoral-module } (m \parallel_{\uparrow} n)$   
**using** *max-par-comp-sound*  
**by** *metis*  
**fix**  
 $A :: 'a \text{ set}$  **and**  
 $V :: 'v \text{ set}$  **and**  
 $p :: ('a, 'v) \text{ Profile}$  **and**  
 $q :: ('a, 'v) \text{ Profile}$  **and**

$a :: 'a$   
**assume**  
 $\text{defer-}a: a \in \text{defer } (m \parallel_{\uparrow} n) \ V \ A \ p$  **and**  
 $\text{lifted-}a: \text{Profile.lifted } V \ A \ p \ q \ a$   
**hence**  $f\text{-profs}: \text{finite-profile } V \ A \ p \wedge \text{finite-profile } V \ A \ q$   
**unfolding**  $\text{lifted-def}$   
**by**  $\text{simp}$   
**from**  $\text{compatible}$   
**obtain**  $B :: 'a \text{ set}$  **where**  
 $\text{alts}: B \subseteq A$   
 $\wedge (\forall b \in B. \text{indep-of-alt } m \ V \ A \ b \wedge$   
 $\quad (\forall p'. \text{finite-profile } V \ A \ p' \longrightarrow b \in \text{reject } m \ V \ A \ p'))$   
 $\wedge (\forall b \in A - B. \text{indep-of-alt } n \ V \ A \ b \wedge$   
 $\quad (\forall p'. \text{finite-profile } V \ A \ p' \longrightarrow b \in \text{reject } n \ V \ A \ p'))$   
**using**  $f\text{-profs}$   
**unfolding**  $\text{disjoint-compatibility-def}$   
**by**  $(\text{metis } (\text{no-types}, \text{lifting}))$   
**have**  $\forall b \in A. \text{prof-contains-result } (m \parallel_{\uparrow} n) \ V \ A \ p \ q \ b$   
**proof**  $(\text{cases})$   
**assume**  $a\text{-in-}B: a \in B$   
**hence**  $a \in \text{reject } m \ V \ A \ p$   
**using**  $\text{alts } f\text{-profs}$   
**by**  $\text{blast}$   
**with**  $\text{defer-}a$   
**have**  $\text{defer-}n: a \in \text{defer } n \ V \ A \ p$   
**using**  $\text{compatible } f\text{-profs } \text{max-agg-rej-snd-equiv-seq-contained}$   
**unfolding**  $\text{disjoint-compatibility-def } \text{mod-contains-result-sym-def}$   
**by**  $\text{metis}$   
**have**  $\forall b \in B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ b$   
**using**  $\text{alts } \text{compatible } \text{max-agg-rej-snd-equiv-seq-contained } f\text{-profs}$   
**unfolding**  $\text{disjoint-compatibility-def}$   
**by**  $\text{metis}$   
**moreover** **have**  $\forall b \in A. \text{prof-contains-result } n \ V \ A \ p \ q \ b$   
**proof**  $(\text{unfold } \text{prof-contains-result-def}, \text{clarify})$   
**fix**  $b :: 'a$   
**assume**  $b\text{-in-}A: b \in A$   
**show**  $\text{SCF-result.electoral-module } n \wedge \text{profile } V \ A \ p$   
 $\wedge \text{profile } V \ A \ q \wedge b \in A \wedge$   
 $(b \in \text{elect } n \ V \ A \ p \longrightarrow b \in \text{elect } n \ V \ A \ q) \wedge$   
 $(b \in \text{reject } n \ V \ A \ p \longrightarrow b \in \text{reject } n \ V \ A \ q) \wedge$   
 $(b \in \text{defer } n \ V \ A \ p \longrightarrow b \in \text{defer } n \ V \ A \ q)$   
**proof**  $(\text{safe})$   
**show**  $\text{SCF-result.electoral-module } n$   
**using**  $\text{monotone-}n$   
**unfolding**  $\text{defer-lift-invariance-def}$   
**by**  $\text{metis}$   
**next**  
**show**  
 $\text{profile } V \ A \ p$  **and**

```

    profile  $V A q$  and
     $b \in A$ 
    using  $f\text{-profs } b\text{-in-}A$ 
    by ( $\text{simp}$ ,  $\text{simp}$ ,  $\text{simp}$ )
next
show
   $b \in \text{elect } n \ V A p \implies b \in \text{elect } n \ V A q$  and
   $b \in \text{reject } n \ V A p \implies b \in \text{reject } n \ V A q$  and
   $b \in \text{defer } n \ V A p \implies b \in \text{defer } n \ V A q$ 
  using  $\text{defer-}n \ \text{lifted-}a \ \text{monotone-}n \ f\text{-profs}$ 
  unfolding  $\text{defer-lift-invariance-def}$ 
  by ( $\text{metis}$ ,  $\text{metis}$ ,  $\text{metis}$ )
qed
qed
moreover have  $\forall b \in B. \text{mod-contains-result } n \ (m \parallel_{\uparrow} n) \ V A q b$ 
  using  $\text{alts compatible max-agg-rej-snd-imp-seq-contained } f\text{-profs}$ 
  unfolding  $\text{disjoint-compatibility-def}$ 
  by  $\text{metis}$ 
ultimately have  $\text{prof-contains-result-of-comps-for-elems-in-}B$ :
   $\forall b \in B. \text{prof-contains-result } (m \parallel_{\uparrow} n) \ V A p q b$ 
  unfolding  $\text{mod-contains-result-def mod-contains-result-sym-def}$ 
   $\text{prof-contains-result-def}$ 
  by  $\text{simp}$ 
have  $\forall b \in A - B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ m \ V A p b$ 
  using  $\text{alts max-agg-rej-fst-equiv-seq-contained monotone-}m \ \text{monotone-}n \ f\text{-profs}$ 
  unfolding  $\text{defer-lift-invariance-def}$ 
  by  $\text{metis}$ 
moreover have  $\forall b \in A. \text{prof-contains-result } m \ V A p q b$ 
proof (unfold  $\text{prof-contains-result-def}$ , 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  $\text{monotone-}m$ 
    unfolding  $\text{defer-lift-invariance-def}$ 
    by  $\text{metis}$ 
  next
  show
     $\text{profile } V A p$  and
     $\text{profile } V A q$  and
     $b \in A$ 
    using  $f\text{-profs } b\text{-in-}A$ 
    by ( $\text{simp}$ ,  $\text{simp}$ ,  $\text{simp}$ )
  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 alts a-in-B lifted-a lifted-imp-equiv-prof-except-a
unfolding indep-of-alt-def
by (metis, metis, metis)
qed
qed
moreover have  $\forall b \in A - B. \text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ q \ b$ 
using alts max-agg-rej-fst-imp-seq-contained monotone-m monotone-n f-profs
unfolding defer-lift-invariance-def
by metis
ultimately have  $\forall b \in A - B. \text{prof-contains-result } (m \parallel_{\uparrow} n) \ V \ A \ p \ q \ b$ 
unfolding mod-contains-result-def mod-contains-result-sym-def
  prof-contains-result-def
by simp
thus ?thesis
using prof-contains-result-of-comps-for-elems-in-B
by blast
next
assume  $a \notin B$ 
hence a-in-set-diff:  $a \in A - B$ 
using DiffI lifted-a compatible f-profs
unfolding Profile.lifted-def
by (metis (no-types, lifting))
hence reject-n:  $a \in \text{reject } n \ V \ A \ p$ 
using alts f-profs
by blast
hence defer-m:  $a \in \text{defer } m \ V \ A \ p$ 
using mod-m mod-n defer-a f-profs max-agg-rej-fst-equiv-seq-contained
unfolding mod-contains-result-sym-def
by (metis (no-types))
have  $\forall b \in B. \text{mod-contains-result } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ b$ 
using alts compatible f-profs max-agg-rej-snd-imp-seq-contained mod-contains-result-comm
unfolding disjoint-compatibility-def
by metis
have  $\forall b \in B. \text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ b$ 
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-in-A:  $b \in A$ 
  show SCF-result.electoral-module  $n \wedge \text{profile } V \ A \ p \wedge$ 
    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 ∈ defer n V A p ⟶ b ∈ defer n V A q)
proof (safe)
  show SCF-result.electoral-module n
    using monotone-n
    unfolding defer-lift-invariance-def
    by metis
next
  show
    profile V A p and
    profile V A q and
    b ∈ A
    using f-profs b-in-A
    by (simp, simp, simp)
next
  show
    b ∈ elect n V A p ⟹ b ∈ elect n V A q and
    b ∈ reject n V A p ⟹ b ∈ reject n V A q and
    b ∈ defer n V A p ⟹ b ∈ defer n V A q
    using alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a
    unfolding indep-of-alt-def
    by (metis, metis, metis)
qed
qed
moreover have ∀ b ∈ B. mod-contains-result n (m ||↑ n) V A q b
  using alts compatible max-agg-rej-snd-imp-seq-contained f-profs
  unfolding disjoint-compatibility-def
  by metis
ultimately have prof-contains-result-of-comps-for-elems-in-B:
  ∀ b ∈ B. prof-contains-result (m ||↑ n) V A p q b
  unfolding mod-contains-result-def mod-contains-result-sym-def
    prof-contains-result-def
  by simp
have ∀ b ∈ A - B. mod-contains-result-sym (m ||↑ n) m V A p b
  using alts max-agg-rej-fst-equiv-seq-contained monotone-m monotone-n f-profs
  unfolding defer-lift-invariance-def
  by metis
moreover have ∀ b ∈ A. prof-contains-result m V A p q b
proof (unfold prof-contains-result-def, clarify)
  fix b :: 'a
  assume b-in-A: b ∈ A
  show SCF-result.electoral-module m ∧ profile V A p
    ∧ profile V A q ∧ b ∈ A
    ∧ (b ∈ elect m V A p ⟶ b ∈ elect m V A q)
    ∧ (b ∈ reject m V A p ⟶ b ∈ reject m V A q)
    ∧ (b ∈ defer m V A p ⟶ b ∈ defer m V A q)
proof (safe)
  show SCF-result.electoral-module m
    using monotone-m
    unfolding defer-lift-invariance-def

```

```

    by simp
next
show
  profile V A p and
  profile V A q and
  b ∈ A
  using f-profs b-in-A
  by (simp, simp, simp)
next
show
  b ∈ elect m V A p ⇒ b ∈ elect m V A q and
  b ∈ reject m V A p ⇒ b ∈ reject m V A q and
  b ∈ defer m V A p ⇒ b ∈ defer m V A q
  using defer-m lifted-a monotone-m
  unfolding defer-lift-invariance-def
  by (metis, metis, metis)
qed
qed
moreover have ∀ x ∈ A - B. mod-contains-result m (m ||↑ n) V A q x
  using alts max-agg-rej-fst-imp-seq-contained monotone-m monotone-n f-profs
  unfolding defer-lift-invariance-def
  by metis
ultimately have ∀ x ∈ A - B. prof-contains-result (m ||↑ n) V A p q x
  unfolding mod-contains-result-def mod-contains-result-sym-def
  prof-contains-result-def
  by simp
thus ?thesis
  using prof-contains-result-of-comps-for-elems-in-B
  by blast
qed
thus (m ||↑ n) V A p = (m ||↑ n) V A q
  using compatible f-profs eq-alts-in-profs-imp-eq-results max-par-comp-sound
  unfolding disjoint-compatibility-def
  by metis
qed

lemma par-comp-rej-card:
  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    n :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    c :: nat
  assumes
    compatible: disjoint-compatibility m n and
    prof: profile V A p and
    fin-A: finite A and
    reject-sum: card (reject m V A p) + card (reject n V A p) = card A + c

```

**shows**  $\text{card } (\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p) = c$   
**proof** –  
**obtain**  $B :: 'a \text{ set}$  **where**  
 $\text{alt-set: } B \subseteq A$   
 $\wedge (\forall a \in B. \text{indep-of-alt } m \ V \ A \ a \wedge$   
 $\quad (\forall q. \text{profile } V \ A \ q \longrightarrow a \in \text{reject } m \ V \ A \ q))$   
 $\wedge (\forall a \in A - B. \text{indep-of-alt } n \ V \ A \ a \wedge$   
 $\quad (\forall q. \text{profile } V \ A \ q \longrightarrow a \in \text{reject } n \ V \ A \ q))$   
**using** *compatible prof*  
**unfolding** *disjoint-compatibility-def*  
**by** *metis*  
**have** *reject-representation:*  
 $\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p = (\text{reject } m \ V \ A \ p) \cap (\text{reject } n \ V \ A \ p)$   
**using** *prof fin-A compatible max-agg-rej-intersect*  
**unfolding** *disjoint-compatibility-def*  
**by** *metis*  
**have** *SCF-result.electoral-module*  $m \wedge \text{SCF-result.electoral-module } n$   
**using** *compatible*  
**unfolding** *disjoint-compatibility-def*  
**by** *simp*  
**hence** *subsets:*  $(\text{reject } m \ V \ A \ p) \subseteq A \wedge (\text{reject } n \ V \ A \ p) \subseteq A$   
**using** *prof*  
**by** *(simp add: reject-in-alts)*  
**hence** *finite*  $(\text{reject } m \ V \ A \ p) \wedge \text{finite } (\text{reject } n \ V \ A \ p)$   
**using** *rev-finite-subset prof fin-A*  
**by** *metis*  
**hence** *card-difference:*  
 $\text{card } (\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p)$   
 $= \text{card } A + c - \text{card } ((\text{reject } m \ V \ A \ p) \cup (\text{reject } n \ V \ A \ p))$   
**using** *card-Un-Int reject-representation reject-sum*  
**by** *fastforce*  
**have**  $\forall a \in A. a \in (\text{reject } m \ V \ A \ p) \vee a \in (\text{reject } n \ V \ A \ p)$   
**using** *alt-set prof fin-A*  
**by** *blast*  
**hence**  $A = \text{reject } m \ V \ A \ p \cup \text{reject } n \ V \ A \ p$   
**using** *subsets*  
**by** *force*  
**thus**  $\text{card } (\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p) = c$   
**using** *card-difference*  
**by** *simp*  
**qed**

Using the max-aggregator for composing two compatible modules in parallel, whereof the first one is non-electing and defers exactly one alternative, and the second one rejects exactly two alternatives, the composition results in an electoral module that eliminates exactly one alternative.

**theorem** *par-comp-elim-one[simp]:*  
**fixes**  
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  **and**

```

  n :: ('a, 'v, 'a Result) Electoral-Module
assumes
  defers-m-one: defers 1 m and
  non-elec-m: non-electing m and
  rejec-n-two: rejects 2 n and
  disj-comp: disjoint-compatibility m n
shows eliminates 1 (m  $\parallel_{\uparrow}$  n)
proof (unfold eliminates-def, safe)
  have 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  $\parallel_{\uparrow}$  n)
    using max-par-comp-sound
    by metis
next
fix
  A :: 'a set and
  V :: 'v set and
  p :: ('a, 'v) Profile
assume
  min-card-two: 1 < card A and
  prof: profile V A p
hence card-geq-one: card A  $\geq$  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

```

$\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$   
*elector m = (m  $\triangleright$  elect-module)*

### 6.7.2 Auxiliary Lemmas

**lemma** *elector-seqcomp-assoc*:

**fixes**

*a* :: ('a, 'v, 'a Result) Electoral-Module **and**

*b* :: ('a, 'v, 'a Result) Electoral-Module

**shows** (*a*  $\triangleright$  (*elector b*)) = (*elector (a*  $\triangleright$  *b*))

**unfolding** *elector.simps elect-module.simps sequential-composition.simps*

**using** *boolean-algebra-cancel.sup2 fst-eqD snd-eqD sup-commute*

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

### 6.7.3 Soundness

**theorem** *elector-sound[simp]*:

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

**assumes** *SCF-result.electoral-module m*

**shows** *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**  
*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**  $\text{finite } (A (\text{elector } m)) \wedge$   
 $\text{profile } (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m)) \wedge$   
 $\text{elect } (\text{elector } m) (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m)) = \{\} \wedge$   
 $d (\text{elector } m (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m))) = \{\} \wedge$   
 $\text{reject } (\text{elector } m) (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m)) =$   
 $r (\text{elector } m (V (\text{elector } m)) (A (\text{elector } m)) (p (\text{elector } m))) \longrightarrow$   
 $\text{electing } (\text{elector } m)$   
**using Diff-empty elector.simps non-block-m snd-conv non-blocking-def reject-not-elec-or-def**

```

      non-block seq-comp-presv-non-blocking
    by (metis (mono-tags, opaque-lifting))
  ultimately show ?thesis
    using non-block-m
    unfolding elector.simps
    by auto
qed

```

### 6.7.5 Composition Rule

If  $m$  is defer-Condorcet-consistent, then  $\text{elector}(m)$  is Condorcet consistent.

```

lemma dcc-imp-cc-elector:
  fixes  $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
  assumes defer-condorcet-consistency  $m$ 
  shows condorcet-consistency ( $\text{elector } m$ )
proof (unfold defer-condorcet-consistency-def condorcet-consistency-def, safe)
  show  $\text{SCF-result.electoral-module } (\text{elector } m)$ 
    using assms elector-sound
    unfolding defer-condorcet-consistency-def
    by metis
next
  fix
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$  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:  $\forall y \in A - \{w\}.$ 
     $\text{card } \{i \in V. (w, y) \in (p \ i)\}$ 
     $< \text{card } \{i \in V. (y, w) \in (p \ i)\}$ 
    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
    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**  $x\text{-in-elect-or-defer}: x \in \text{elect } m \ V \ A \ p \cup \text{defer } m \ V \ A \ p$   
**hence**  $x\text{-eq-}w: x = w$   
**using** *Diff-empty Diff-iff assms cond-winner-unique c-win fin-A fin-V insert-iff*  
 $\text{snd-conv prod.sel}(1) \text{ sup-bot.left-neutral}$   
**unfolding** *defer-condorcet-consistency-def*  
**by** (*metis (mono-tags, lifting)*)  
**have**  $\bigwedge x. x \in \text{elect } m \ V \ A \ p \implies x \in A$   
**using** *fin-A prof-A fin-V assms elect-in-alts in-mono*  
**unfolding** *defer-condorcet-consistency-def*  
**by** *metis*  
**moreover have**  $\bigwedge x. x \in \text{defer } m \ V \ A \ p \implies x \in A$   
**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**  
 $x\text{-not-in-defer}: x \notin \text{defer } m \ V \ A \ p$  **and**  
 $x \in A$  **and**  
 $\forall x' \in A - \{x\}.$   
 $\text{card } \{i \in V. (x, x') \in p \ i\}$   
 $< \text{card } \{i \in V. (x', x) \in p \ i\}$   
**hence** *c-win-x: condorcet-winner V A p x*  
**using** *fin-A prof-A fin-V*  
**by** *simp*  
**have** (*SCF-result.electoral-module*  $m \wedge \neg \text{defer-condorcet-consistency } m \longrightarrow$   
 $(\exists A \ V \ rs \ a. \text{condorcet-winner } V \ A \ rs \ a \wedge$   
 $m \ V \ A \ rs \neq (\{\}, A - \text{defer } m \ V \ A \ rs,$   
 $\{a \in A. \text{condorcet-winner } V \ A \ rs \ a\}))$   
 $\wedge (\text{defer-condorcet-consistency } m \longrightarrow$   
 $(\forall A \ V \ rs \ a. \text{finite } A \longrightarrow \text{finite } V \longrightarrow \text{condorcet-winner } V \ A \ rs \ a \longrightarrow$

```

      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.

### 6.8.1 Definition

```

fun iter :: ('a, 'v, 'a Result) Electoral-Module

```

```

⇒ ('a, 'v, 'a Result) Electoral-Module where
iter m =
  (let t = defer-equal-condition 1 in
   (m ∘t))

abbreviation defer-one-loop :: ('a, 'v, 'a Result) Electoral-Module
⇒ ('a, 'v, 'a Result) Electoral-Module (-∘∃!d 50) where
m ∘∃!d ≡ iter m

fun iter-elect :: ('a, 'v, 'a Result) Electoral-Module
⇒ ('a, 'v, 'a Result) Electoral-Module where
iter-elect m = elector (m ∘∃!d)

end

```

## Chapter 7

# Voting Rules

### 7.1 Plurality Rule

```
theory Plurality-Rule
  imports Compositional-Structures/Basic-Modules/Plurality-Module
           Compositional-Structures/Revision-Composition
           Compositional-Structures/Elect-Composition
begin
```

This is a definition of the plurality voting rule as elimination module as well as directly. In the former one, the max operator of the set of the scores of all alternatives is evaluated and is used as the threshold value.

#### 7.1.1 Definition

```
fun plurality-rule :: ('a, 'v, 'a Result) Electoral-Module where
  plurality-rule V A p = elector plurality V A p
```

```
fun plurality-rule' :: ('a, 'v, 'a Result) Electoral-Module where
  plurality-rule' V A p =
    ({a ∈ 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},
     {})
```

```
lemma plurality-revision-equiv:
  fixes
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  shows plurality' V A p = (plurality-rule' ↓) V A p
proof (unfold plurality'.simps revision-composition.simps, safe)
fix
  a :: 'a and
  b :: 'a
assume
```

```

     $b \in A$  and
     $\text{win-count } V p a < \text{win-count } V p b$  and
     $a \in \text{elect plurality-rule}' V A p$ 
thus False
    by fastforce
next
    fix  $a :: 'a$ 
    assume  $a \notin \text{elect plurality-rule}' V A p$ 
    moreover from this
    have  $a \notin A \vee (\exists x. x \in A \wedge \neg \text{win-count } V p x \leq \text{win-count } V p a)$ 
    by force
    moreover assume  $a \in A$ 
    ultimately show  $\exists x \in A. \text{win-count } V p a < \text{win-count } V p x$ 
    using linorder-le-less-linear
    by metis
next
    fix
     $a :: 'a$  and
     $b :: 'a$ 
    assume
     $a \in A$  and
     $\forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a$ 
    thus  $a \in \text{elect plurality-rule}' V A p$ 
    by simp
next
    fix  $a :: 'a$ 
    assume  $a \in \text{elect plurality-rule}' V A p$ 
    thus  $a \in A$ 
    by simp
next
    fix
     $a :: 'a$  and
     $b :: 'a$ 
    assume
     $a \in \text{elect plurality-rule}' V A p$  and
     $b \in A$ 
    thus  $\text{win-count } V p b \leq \text{win-count } V p a$ 
    by simp
qed

lemma plurality-elim-equiv:
fixes
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
assumes
     $A \neq \{\}$  and
    finite  $A$  and
    profile  $V A p$ 

```

shows  $\text{plurality } V A p = (\text{plurality-rule}' \downarrow) V A p$   
 using *assms plurality-mod-elim-equiv plurality-revision-equiv*  
 by (*metis (full-types)*)

### 7.1.2 Soundness

**theorem** *plurality-rule-sound[simp]*: *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* (  
 $\{a \in A. \forall a' \in A. \text{win-count } V p a' \leq \text{win-count } V p a\},$   
 $\{a \in A. \exists a' \in A. \text{win-count } V p a < \text{win-count } V p a'\},$   
 $\{\}$ )  
 by *auto*  
 moreover have  
 $\{a \in A. \forall x \in A. \text{win-count } V p x \leq \text{win-count } V p a\} \cup$   
 $\{a \in A. \exists x \in A. \text{win-count } V p a < \text{win-count } V p x\} = A$   
 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*

### 7.1.3 Electing

**lemma** *plurality-rule-elect-non-empty:*

fixes  
 $A :: 'a \text{ set}$  and  
 $V :: 'v \text{ set}$  and  
 $p :: ('a, 'v) \text{ Profile}$   
 assumes  
 $A\text{-non-empty: } A \neq \{\}$  and  
 $\text{prof-A: profile } V A p$  and  
 $\text{fin-A: finite } A$   
 shows *elect plurality-rule V A p*  $\neq \{\}$   
**proof**  
 assume *plurality-elect-none: elect plurality-rule V A p*  $= \{\}$

**obtain**  $max$  **where**  
 $max$ :  $max = Max (win-count \ V \ p \ ' \ A)$   
**by** *simp*  
**then obtain**  $a$  **where**  
 $max$ - $a$ :  $win-count \ V \ p \ a = max \wedge a \in A$   
**using** *Max-in A-non-empty fin-A prof-A empty-is-image finite-imageI imageE*  
**by** (*metis (no-types, lifting)*)  
**hence**  $\forall \ a' \in A. win-count \ V \ p \ a' \leq win-count \ V \ p \ a$   
**using** *fin-A prof-A max*  
**by** *simp*  
**moreover have**  $a \in A$   
**using** *max-a*  
**by** *simp*  
**ultimately have**  $a \in \{a' \in A. \forall \ c \in A. win-count \ V \ p \ c \leq win-count \ V \ p \ a'\}$   
**by** *blast*  
**hence**  $a \in elect \ plurality-rule' \ V \ A \ p$   
**by** *simp*  
**moreover have**  $elect \ plurality-rule' \ V \ A \ p = defer \ plurality \ V \ A \ p$   
**using** *plurality-elim-equiv fin-A prof-A A-non-empty snd-conv*  
**unfolding** *revision-composition.simps*  
**by** *metis*  
**ultimately have**  $a \in defer \ plurality \ V \ A \ p$   
**by** *blast*  
**hence**  $a \in elect \ plurality-rule \ V \ A \ p$   
**by** *simp*  
**thus** *False*  
**using** *plurality-elect-none all-not-in-conv*  
**by** *metis*  
**qed**

The plurality module is electing.

**theorem** *plurality-rule-electing[simp]: electing plurality-rule*

**proof** (*unfold electing-def, safe*)

**show** *SCF-result.electoral-module plurality-rule*

**using** *plurality-rule-sound*

**by** *simp*

**next**

**fix**

$A :: 'b \ set$  **and**

$V :: 'a \ set$  **and**

$p :: ('b, 'a) \ Profile$  **and**

$a :: 'b$

**assume**

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

*prof-p: profile V A p* **and**

*elect-none: elect plurality-rule V A p = {}* **and**

*a-in-A: a ∈ A*

**have**  $\forall \ A \ V \ p. A \neq \{\} \wedge finite \ A \wedge profile \ V \ A \ p$   
 $\longrightarrow elect \ plurality-rule \ V \ A \ p \neq \{\}$

```

    using plurality-rule-elect-non-empty
    by (metis (no-types))
  hence empty-A:  $A = \{\}$ 
    using fin-A prof-p elect-none
    by (metis (no-types))
  thus  $a \in \{\}$ 
    using a-in-A
    by simp
qed

```

#### 7.1.4 Property

**lemma** *plurality-rule-inv-mono-eq*:

```

  fixes
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    q :: ('a, 'v) Profile and
    a :: 'a
  assumes
    elect-a:  $a \in \text{elect } \text{plurality-rule } V A p$  and
    lift-a:  $\text{lifted } V A p q a$ 
  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  $\text{eq-p: } \text{elect } (\text{elector plurality}) V A p = \text{defer plurality } V A p$ 
      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

```

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 :: ('b, 'a) Profile and
  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
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<sub>R</sub>* :: ('a, 'v::wellorder, 'a Result) Electoral-Module **where**

*borda-rule<sub>R</sub>* 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 $\mathcal{R}$ -sound: SCF-result.electoral-module borda-rule $\mathcal{R}$ 
  unfolding borda-rule $\mathcal{R}$ .simps swap- $\mathcal{R}$ .simps
  using SCF-result. $\mathcal{R}$ -sound
  by metis

```

### 7.2.3 Anonymity Property

```

theorem borda-rule $\mathcal{R}$ -anonymous: SCF-result.anonymity borda-rule $\mathcal{R}$ 
proof (unfold borda-rule $\mathcal{R}$ .simps swap- $\mathcal{R}$ .simps)
  let ?swap-dist = votewise-distance swap l-one
  from l-one-is-sym
  have distance-anonymity ?swap-dist
    using symmetric-norm-imp-distance-anonymous[of l-one]
    by simp
  with unanimity-anonymous
  show SCF-result.anonymity (SCF-result.distance- $\mathcal{R}$  ?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)  $\circ_{\exists!d}$ ) 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'-rule-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'-rule-sound elector-sound iter.simps iter-elect.simps loop-comp-sound*  
  **by** *metis*

### 7.3.3 Condorcet Consistency Property

**theorem** *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*  
**proof** (*unfold pairwise-majority-rule.simps*)  
  **show** *condorcet-consistency (elector condorcet)*  
    **using** *condorcet-is-dcc dcc-imp-cc-elector*  
    **by** *metis*  
**qed**  
  
**end**

## 7.4 Copeland Rule

**theory** *Copeland-Rule*  
  **imports** *Compositional-Structures/Basic-Modules/Copeland-Module*  
          *Compositional-Structures/Elect-Composition*  
**begin**

This is the Copeland voting rule. The idea is to elect the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses.

### 7.4.1 Definition

**fun** *copeland-rule* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**  
  *copeland-rule V A p = elector copeland V A p*

### 7.4.2 Soundness

**theorem** *copeland-rule-sound*: *SCF-result.electoral-module copeland-rule*

```

unfolding copeland-rule.simps
using elector-sound copeland-sound
by metis

```

### 7.4.3 Condorcet Consistency Property

```

theorem copeland-condorcet: condorcet-consistency copeland-rule
proof (unfold copeland-rule.simps)
  show condorcet-consistency (elector copeland)
    using copeland-is-dcc dcc-imp-cc-elect
    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 Property

```

theorem minimax-condorcet: condorcet-consistency minimax-rule
proof (unfold minimax-rule.simps)
  show condorcet-consistency (elector minimax)
    using minimax-is-dcc dcc-imp-cc-elect
    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 Property

```
theorem black-is-dcc: defer-condorcet-consistency black
  unfolding black.simps
  using condorcet-is-dcc borda-mod-non-blocking borda-mod-non-electing seq-comp-dcc
  by metis

theorem black-condorcet: condorcet-consistency blacks-rule
  unfolding blacks-rule.simps
  using black-is-dcc dcc-imp-cc-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)  $\odot_{\exists!d}$ ) 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*  $t = \text{defer-equal-condition } 1 \text{ in } (?eliminator \circlearrowleft_t)$

**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 Property

The following proof is a fully modular proof for weak monotonicity of sequential majority comparison. It is composed of many small steps.

**theorem** *smc-monotone*:

**fixes**  $x :: 'a$  *Preference-Relation*

**assumes** *linear-order*  $x$

**shows** *monotonicity* (*smc*  $x$ )

**proof** –

**let**  $?pass2 = pass\text{-}module\ 2\ x$

**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-two ||↑ ?drop2
let ?loop =
  let t = defer-equal-condition 1 in (?eliminator  $\odot_t$ )

have 00010: defer-invariant-monotonicity (plurality-rule↓)
  by simp
have 00011: non-electing (plurality-rule↓)
  using rev-comp-non-electing plurality-rule-sound
  by blast
have 00012: non-electing ?tie-breaker
  using assms
  by simp
have 00013: defers 1 ?tie-breaker
  using assms pass-one-mod-def-one
  by simp
have 00014: defer-monotonicity ?tie-breaker
  using assms
  by simp
have 20000: non-blocking (plurality-rule↓)
  by simp
have 0000: defer-lift-invariance ?pass2
  using assms
  by simp
have 0001: defer-lift-invariance ?plurality-defer
  using 00010 00012 00013 00014 def-inv-mono-imp-def-lift-inv
  unfolding pass-module.simps voters-determine-election.simps
  by blast
have 0020: disjoint-compatibility ?pass2 ?drop2
  using assms
  by simp
have 1000: non-electing ?pass2
  using assms
  by simp
have 1001: non-electing ?plurality-defer
  using 00011 00012 seq-comp-presv-non-electing
  by blast
have 2000: non-blocking ?pass2
  using assms
  by simp
have 2001: defers 1 ?plurality-defer
  using 20000 00011 00013 seq-comp-def-one
  by blast
have 000: defer-lift-invariance ?compare-two
  using 0000 0001 seq-comp-presv-def-lift-inv
    voters-determine-plurality-rule voters-determine-pass-mod
    voters-determine-rev-comp voters-determine-seq-comp
  by blast
have 001: defer-lift-invariance ?drop2
  using assms

```

```

    by simp
  have 002: disjoint-compatibility ?compare-two ?drop2
    using assms 0020 disj-compat-seq pass-mod-sound plurality-rule-sound
      voters-determine-pass-mod rev-comp-sound seq-comp-sound voters-determine-seq-comp
      voters-determine-plurality-rule voters-determine-pass-mod voters-determine-rev-comp
    by metis
  have 100: non-electing ?compare-two
    using 1000 1001 seq-comp-presv-non-electing
    by simp
  have 101: non-electing ?drop2
    using assms
    by simp
  have 102: agg-conservative max-aggregator
    by simp
  have 200: defers 1 ?compare-two
    using 2000 1000 2001 seq-comp-def-one
    by simp
  have 201: rejects 2 ?drop2
    using assms
    by simp
  have 00: defer-lift-invariance ?eliminator
    using 000 001 002 par-comp-def-lift-inv
    by blast
  have 10: non-electing ?eliminator
    using 100 101 conserv-max-agg-presv-non-electing
    by blast
  have 20: eliminates 1 ?eliminator
    using 200 100 201 002 par-comp-elim-one
    by simp
  have 0: defer-lift-invariance ?loop
    using 00 loop-comp-presv-def-lift-inv
      voters-determine-plurality-rule voters-determine-pass-mod voters-determine-drop-mod
      voters-determine-rev-comp voters-determine-seq-comp voters-determine-max-par-comp
    by metis
  have 1: non-electing ?loop
    using 10 loop-comp-presv-non-electing
    by simp
  have 2: defers 1 ?loop
    using 10 20 iter-elim-def-n prod.exhaust-sel zero-less-one defer-equal-condition.simps
    by metis
  have 3: electing elect-module
    by simp
  show ?thesis
    using 0 1 2 3 assms seq-comp-mono
    unfolding Electoral-Module.monotonicity-def elector.simps
      Defer-One-Loop-Composition.iter.simps
      smc-sound smc.simps
    by (metis (full-types))
qed

```

end

## 7.11 Kemeny Rule

**theory** *Kemeny-Rule*

**imports**

*Compositional-Structures/Basic-Modules/Component-Types/Votewise-Distance-Rationalization*

*Compositional-Structures/Basic-Modules/Component-Types/Distance-Rationalization-Symmetry*

**begin**

This is the Kemeny rule. It creates a complete ordering of alternatives and evaluates each ordering of the alternatives in terms of the sum of preference reversals on each ballot that would have to be performed in order to produce that transitive ordering. The complete ordering which requires the fewest preference reversals is the final result of the method.

### 7.11.1 Definition

**fun** *kemeny-rule* :: ('a, 'v::wellorder, 'a Result) *Electoral-Module* **where**  
*kemeny-rule* *V A p* = *swap- $\mathcal{R}$  strong-unanimity V A p*

### 7.11.2 Soundness

**theorem** *kemeny-rule-sound*: *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 Property

**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 Property

**lemma** *swap-dist-neutral*: *distance-neutrality valid-elections*

```

                                (votewise-distance swap l-one)
using neutral-dist-imp-neutral-votewise-dist swap-neutral
by blast

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

end

```

# Bibliography

- [1] K. Diekhoff, M. Kirsten, and J. Krämer. Formal property-oriented design of voting rules using composable modules. In S. Pekeč and K. Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019.
- [2] K. Diekhoff, M. Kirsten, and J. Krämer. Verified construction of fair voting rules. In M. Gabbrielli, editor, *29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019), Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2020.