

Verified Construction of Fair Voting Rules

Michael Kirsten

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

April 8, 2024

Abstract

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

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

Contents

1	Social-Choice Types	9
1.1	Preference Relation	9
1.1.1	Definition	9
1.1.2	Ranking	10
1.1.3	Limited Preference	10
1.1.4	Auxiliary Lemmas	13
1.1.5	Lifting Property	16
1.2	Norm	19
1.2.1	Definition	20
1.2.2	Auxiliary Lemmas	20
1.2.3	Common Norms	20
1.2.4	Properties	20
1.2.5	Theorems	20
1.3	Electoral Result	20
1.3.1	Auxiliary Functions	21
1.3.2	Definition	21
1.4	Preference Profile	22
1.4.1	Definition	22
1.4.2	Vote Count	23
1.4.3	Voter Permutations	24
1.4.4	List Representation for Ordered Voters	25
1.4.5	Preference Counts and Comparisons	26
1.4.6	Condorcet Winner	29
1.4.7	Limited Profile	29
1.4.8	Lifting Property	30
1.5	Social Choice Result	31
1.5.1	Social Choice Result	32
1.5.2	Auxiliary Lemmas	32
1.6	Social Welfare Result	33
1.6.1	Social Welfare Result	33
1.7	Specific Electoral Result Types	33
1.8	Function Symmetry Properties	34
1.8.1	Functions	34

1.8.2	Relations for Symmetry Constructions	35
1.8.3	Invariance and Equivariance	35
1.8.4	Auxiliary Lemmas	35
1.8.5	Rewrite Rules	36
1.8.6	Group Actions	39
1.8.7	Invariance and Equivariance	39
1.9	Symmetry Properties of Voting Rules	43
1.9.1	Definitions	43
1.9.2	Auxiliary Lemmas	45
1.9.3	Anonymity Lemmas	47
1.9.4	Neutrality Lemmas	48
1.9.5	Homogeneity Lemmas	50
1.9.6	Reversal Symmetry Lemmas	50
1.10	Result-Dependent Voting Rule Properties	51
1.10.1	Properties Dependent on the Result Type	51
1.10.2	Interpretations	52
2	Refined Types	53
2.1	Preference List	53
2.1.1	Well-Formedness	53
2.1.2	Auxiliary Lemmas About Lists	53
2.1.3	Ranking	54
2.1.4	Definition	55
2.1.5	Limited Preference	56
2.1.6	Auxiliary Definitions	57
2.1.7	Auxiliary Lemmas	57
2.1.8	First Occurrence Indices	59
2.2	Preference (List) Profile	60
2.2.1	Definition	60
2.3	Ordered Relation Type	61
2.4	Alternative Election Type	61
3	Quotient Rules	63
3.1	Quotients of Equivalence Relations	63
3.1.1	Definitions	63
3.1.2	Well-Definedness	63
3.1.3	Equivalence Relations	64
3.2	Quotients of Equivalence Relations on Election Sets	65
3.2.1	Auxiliary Lemmas	65
3.2.2	Anonymity Quotient: Grid	65
3.2.3	Homogeneity Quotient: Simplex	66

4	Component Types	69
4.1	Distance	69
4.1.1	Definition	69
4.1.2	Conditions	70
4.1.3	Standard Distance Property	70
4.1.4	Auxiliary Lemmas	70
4.1.5	Swap Distance	71
4.1.6	Spearman Distance	72
4.1.7	Properties	72
4.2	Votewise Distance	74
4.2.1	Definition	74
4.2.2	Inference Rules	75
4.3	Consensus	76
4.3.1	Definition	76
4.3.2	Consensus Conditions	76
4.3.3	Properties	77
4.3.4	Auxiliary Lemmas	77
4.3.5	Theorems	78
4.4	Electoral Module	79
4.4.1	Definition	80
4.4.2	Auxiliary Definitions	80
4.4.3	Properties	81
4.4.4	Reversal Symmetry of Social Welfare Rules	82
4.4.5	Social Choice Modules	82
4.4.6	Equivalence Definitions	83
4.4.7	Auxiliary Lemmas	84
4.4.8	Non-Blocking	89
4.4.9	Electing	89
4.4.10	Properties	90
4.4.11	Inference Rules	93
4.4.12	Social Choice Properties	94
4.5	Electoral Module on Election Quotients	94
4.6	Evaluation Function	95
4.6.1	Definition	95
4.6.2	Property	95
4.6.3	Theorems	96
4.7	Elimination Module	97
4.7.1	General Definitions	97
4.7.2	Social Choice Definitions	97
4.7.3	Common Social Choice Eliminators	97
4.7.4	Soundness	98
4.7.5	Only participating voters impact the result	99
4.7.6	Non-Blocking	100
4.7.7	Non-Electing	101

4.7.8	Inference Rules	102
4.8	Aggregator	102
4.8.1	Definition	103
4.8.2	Properties	103
4.9	Maximum Aggregator	103
4.9.1	Definition	103
4.9.2	Auxiliary Lemma	104
4.9.3	Soundness	104
4.9.4	Properties	104
4.10	Termination Condition	104
4.10.1	Definition	105
4.11	Defer Equal Condition	105
4.11.1	Definition	105
5	Basic Modules	106
5.1	Defer Module	106
5.1.1	Definition	106
5.1.2	Soundness	106
5.1.3	Properties	106
5.2	Elect First Module	106
5.2.1	Definition	107
5.2.2	Soundness	107
5.3	Consensus Class	107
5.3.1	Definition	107
5.3.2	Consensus Choice	108
5.3.3	Auxiliary Lemmas	108
5.3.4	Consensus Rules	109
5.3.5	Properties	109
5.3.6	Inference Rules	110
5.3.7	Theorems	111
5.4	Distance Rationalization	111
5.4.1	Definitions	112
5.4.2	Standard Definitions	112
5.4.3	Auxiliary Lemmas	113
5.4.4	Soundness	114
5.4.5	Inference Rules	115
5.5	Votewise Distance Rationalization	115
5.5.1	Common Rationalizations	116
5.5.2	Theorems	116
5.5.3	Equivalence Lemmas	116
5.6	Symmetry in Distance-Rationalizable Rules	117
5.6.1	Minimizer Function	117
5.6.2	Distance Rationalization as Minimizer	119
5.6.3	Symmetry Property Inference Rules	121

5.6.4	Further Properties	122
5.7	Distance Rationalization on Election Quotients	122
5.7.1	Quotient Distances	123
5.7.2	Quotient Consensus and Results	126
5.7.3	Quotient Distance Rationalization	127
5.8	Result and Property Locale Code Generation	129
5.9	Drop Module	131
5.9.1	Definition	131
5.9.2	Soundness	131
5.9.3	Non-Electing	131
5.9.4	Properties	132
5.10	Pass Module	132
5.10.1	Definition	132
5.10.2	Soundness	132
5.10.3	Non-Blocking	133
5.10.4	Non-Electing	133
5.10.5	Properties	133
5.11	Elect Module	134
5.11.1	Definition	134
5.11.2	Soundness	134
5.11.3	Electing	135
5.12	Plurality Module	135
5.12.1	Definition	135
5.12.2	Soundness	136
5.12.3	Non-Blocking	136
5.12.4	Non-Electing	136
5.12.5	Property	136
5.13	Borda Module	137
5.13.1	Definition	137
5.13.2	Soundness	137
5.13.3	Non-Blocking	138
5.13.4	Non-Electing	138
5.14	Condorcet Module	138
5.14.1	Definition	138
5.14.2	Soundness	138
5.14.3	Property	138
5.15	Copeland Module	139
5.15.1	Definition	139
5.15.2	Soundness	139
5.15.3	Only Voters Determine Election Result	139
5.15.4	Lemmas	139
5.15.5	Property	141
5.16	Minimax Module	141
5.16.1	Definition	141

5.16.2	Soundness	141
5.16.3	Lemma	141
5.16.4	Property	142
6	Compositional Structures	143
6.1	Drop And Pass Compatibility	143
6.1.1	Properties	143
6.2	Revision Composition	144
6.2.1	Definition	144
6.2.2	Soundness	144
6.2.3	Composition Rules	144
6.3	Sequential Composition	145
6.3.1	Definition	145
6.3.2	Soundness	147
6.3.3	Lemmas	147
6.3.4	Composition Rules	149
6.4	Parallel Composition	152
6.4.1	Definition	153
6.4.2	Soundness	153
6.4.3	Composition Rule	153
6.5	Loop Composition	154
6.5.1	Definition	154
6.5.2	Soundness	156
6.5.3	Lemmas	156
6.5.4	Composition Rules	159
6.6	Maximum Parallel Composition	160
6.6.1	Definition	160
6.6.2	Soundness	161
6.6.3	Lemmas	161
6.6.4	Composition Rules	164
6.7	Elect Composition	165
6.7.1	Definition	165
6.7.2	Auxiliary Lemmas	165
6.7.3	Soundness	166
6.7.4	Electing	166
6.7.5	Composition Rule	166
6.8	Defer One Loop Composition	166
6.8.1	Definition	167
7	Voting Rules	168
7.1	Plurality Rule	168
7.1.1	Definition	168
7.1.2	Soundness	169
7.1.3	Electing	169

7.1.4	Property	169
7.2	Borda Rule	170
7.2.1	Definition	170
7.2.2	Soundness	170
7.2.3	Anonymity Property	170
7.3	Pairwise Majority Rule	171
7.3.1	Definition	171
7.3.2	Soundness	171
7.3.3	Condorcet Consistency Property	171
7.4	Copeland Rule	172
7.4.1	Definition	172
7.4.2	Soundness	172
7.4.3	Condorcet Consistency Property	172
7.5	Minimax Rule	172
7.5.1	Definition	172
7.5.2	Soundness	172
7.5.3	Condorcet Consistency Property	173
7.6	Black's Rule	173
7.6.1	Definition	173
7.6.2	Soundness	173
7.6.3	Condorcet Consistency Property	173
7.7	Nanson-Baldwin Rule	174
7.7.1	Definition	174
7.7.2	Soundness	174
7.8	Classic Nanson Rule	174
7.8.1	Definition	174
7.8.2	Soundness	175
7.9	Schwartz Rule	175
7.9.1	Definition	175
7.9.2	Soundness	175
7.10	Sequential Majority Comparison	175
7.10.1	Definition	176
7.10.2	Soundness	176
7.10.3	Electing	176
7.10.4	(Weak) Monotonicity Property	176
7.11	Kemeny Rule	177
7.11.1	Definition	177
7.11.2	Soundness	177
7.11.3	Anonymity Property	177
7.11.4	Neutrality Property	177

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
```

<proof>

lemma *lin-imp-trans*:

fixes

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

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

assumes *linear-order-on* A r

shows *trans* r

<proof>

1.1.2 Ranking

fun *rank* :: $'a \text{ Preference-Relation} \Rightarrow 'a \Rightarrow \text{nat}$ **where**

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

lemma *rank-gt-zero*:

fixes

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

$a :: 'a$

assumes

refl: $a \preceq_r a$ **and**

fin: *finite* r

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

<proof>

1.1.3 Limited Preference

definition *limited* :: $'a \text{ set} \Rightarrow 'a \text{ Preference-Relation} \Rightarrow \text{bool}$ **where**

$\text{limited } A \ r \equiv r \subseteq A \times A$

lemma *limited-dest*:

fixes

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

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

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

$b :: 'a$

assumes

$a \preceq_r b$ **and**

limited $A \ r$

shows $a \in A \wedge b \in A$

<proof>

fun *limit* :: $'a \text{ set} \Rightarrow 'a \text{ Preference-Relation} \Rightarrow 'a \text{ Preference-Relation}$ **where**

$\text{limit } A \ r = \{(a, b) \in r. a \in A \wedge b \in A\}$

definition *connex* :: $'a \text{ set} \Rightarrow 'a \text{ Preference-Relation} \Rightarrow \text{bool}$ **where**

$\text{connex } A \ r \equiv \text{limited } A \ r \wedge (\forall a \in A. \forall b \in A. a \preceq_r b \vee b \preceq_r a)$

lemma *connex-imp-refl*:

fixes

```

    A :: 'a set and
    r :: 'a Preference-Relation
assumes connex A r
shows refl-on A r
⟨proof⟩

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⟩

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⟩

lemma limit-to-limits:
fixes
    A :: 'a set and
    r :: 'a Preference-Relation
shows limited A (limit A r)
⟨proof⟩

lemma limit-presv-connex:
fixes
    B :: 'a set and
    A :: 'a set and
    r :: 'a Preference-Relation
assumes
    connex: connex B r and
    subset: A ⊆ B
shows connex A (limit A r)
⟨proof⟩

lemma limit-presv-antisym:
fixes
    A :: 'a set and
    r :: 'a Preference-Relation
assumes antisym r
shows antisym (limit A r)

```

$\langle proof \rangle$

lemma *limit-presv-trans:*

fixes

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

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

assumes *trans* r

shows *trans* (*limit* A r)

$\langle proof \rangle$

lemma *limit-presv-lin-ord:*

fixes

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

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

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

assumes

linear-order-on B r **and**

$A \subseteq B$

shows *linear-order-on* A (*limit* A r)

$\langle proof \rangle$

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* $s = \text{limit } A \text{ } r$ *in* $a \preceq_s b$

$\langle proof \rangle$

lemma *limit-rel-presv-prefs:*

fixes

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

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

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

$b :: 'a$

assumes $(a, b) \in \text{limit } A \text{ } r$

shows $a \preceq_r b$

$\langle proof \rangle$

lemma *limit-trans:*

fixes

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

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

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

assumes $A \subseteq B$
shows $\text{limit } A \ r = \text{limit } A \ (\text{limit } B \ r)$
 $\langle \text{proof} \rangle$

lemma *lin-ord-not-empty*:
fixes $r :: 'a \text{ Preference-Relation}$
assumes $r \neq \{\}$
shows $\neg \text{linear-order-on } \{\} \ r$
 $\langle \text{proof} \rangle$

lemma *lin-ord-singleton*:
fixes $a :: 'a$
shows $\forall \ r. \text{linear-order-on } \{a\} \ r \longrightarrow r = \{(a, a)\}$
 $\langle \text{proof} \rangle$

1.1.4 Auxiliary Lemmas

lemma *above-trans*:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
 $\text{trans } r$ **and**
 $(a, b) \in r$
shows $\text{above } r \ b \subseteq \text{above } r \ a$
 $\langle \text{proof} \rangle$

lemma *above-refl*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes
 $\text{refl-on } A \ r$ **and**
 $a \in A$
shows $a \in \text{above } r \ a$
 $\langle \text{proof} \rangle$

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
 $\text{linear-order-on } A \ r$ **and**
 $\text{linear-order-on } A \ r'$ **and**
 $\text{above } r \ a \subseteq \text{above } r' \ a$ **and**

$above\ r'\ a = \{a\}$
shows $above\ r\ a = \{a\}$
 $\langle proof \rangle$

lemma *above-connex:*

fixes
 $A :: 'a\ set$ **and**
 $r :: 'a\ Preference-Relation$ **and**
 $a :: 'a$
assumes
 $connex\ A\ r$ **and**
 $a \in A$
shows $a \in above\ r\ a$
 $\langle proof \rangle$

lemma *pref-imp-in-above:*

fixes
 $r :: 'a\ Preference-Relation$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
shows $(a \preceq_r b) = (b \in above\ r\ a)$
 $\langle proof \rangle$

lemma *limit-presv-above:*

fixes
 $A :: 'a\ set$ **and**
 $r :: 'a\ Preference-Relation$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
 $b \in above\ r\ a$ **and**
 $a \in A$ **and**
 $b \in A$
shows $b \in above\ (limit\ A\ r)\ a$
 $\langle proof \rangle$

lemma *limit-rel-presv-above:*

fixes
 $A :: 'a\ set$ **and**
 $B :: 'a\ set$ **and**
 $r :: 'a\ Preference-Relation$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes $b \in above\ (limit\ B\ r)\ a$
shows $b \in above\ r\ a$
 $\langle proof \rangle$

lemma *above-one:*

fixes

$A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$
assumes
 $\text{lin-ord-}r$: $\text{linear-order-on } A \ r$ **and**
 $\text{fin-}A$: $\text{finite } A$ **and**
 $\text{non-empty-}A$: $A \neq \{\}$
shows $\exists a \in A. \text{above } r \ a = \{a\} \wedge (\forall a' \in A. \text{above } r \ a' = \{a'\} \longrightarrow a' = a)$
 $\langle \text{proof} \rangle$

lemma above-one-eq :
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
 lin-ord : $\text{linear-order-on } A \ r$ **and**
 $\text{fin-}A$: $\text{finite } A$ **and**
 $\text{not-empty-}A$: $A \neq \{\}$ **and**
 $\text{above-}a$: $\text{above } r \ a = \{a\}$ **and**
 $\text{above-}b$: $\text{above } r \ b = \{b\}$
shows $a = b$
 $\langle \text{proof} \rangle$

lemma $\text{above-one-imp-rank-one}$:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes $\text{above } r \ a = \{a\}$
shows $\text{rank } r \ a = 1$
 $\langle \text{proof} \rangle$

lemma $\text{rank-one-imp-above-one}$:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes
 lin-ord : $\text{linear-order-on } A \ r$ **and**
 rank-one : $\text{rank } r \ a = 1$
shows $\text{above } r \ a = \{a\}$
 $\langle \text{proof} \rangle$

theorem above-rank :
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes $\text{linear-order-on } A \ r$

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

lemma *rank-unique*:

fixes

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

assumes

lin-ord: *linear-order-on* $A \ r$ **and**
fin-A: *finite* A **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
a-neg-b: $a \neq b$

shows $\text{rank } r \ a \neq \text{rank } r \ b$

$\langle \text{proof} \rangle$

lemma *above-presv-limit*:

fixes

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

shows $\text{above } (\text{limit } A \ r) \ a \subseteq A$

$\langle \text{proof} \rangle$

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**

equiv-rel-except-a $A \ r \ r' \ a \equiv$

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**

lifted $A \ r \ r' \ a \equiv$

equiv-rel-except-a $A \ r \ r' \ a \wedge (\exists a' \in A - \{a\}. a \preceq_r a' \wedge a' \preceq_{r'} a)$

lemma *trivial-equiv-rel*:

fixes

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

assumes *linear-order-on* $A \ r$

shows $\forall a \in A. \text{equiv-rel-except-a } A \ r \ r \ a$

$\langle \text{proof} \rangle$

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 *lifted* $A \ r \ r' \ a$
shows *equiv-rel-except-a* $A \ r \ r' \ a$
 $\langle \text{proof} \rangle$

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 $\forall \ a' \in A - \{a\}. \neg (a' \preceq_r a \wedge a \preceq_{r'} a')$
 $\langle \text{proof} \rangle$

lemma *lifted-mono*:
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: *lifted* $A \ r \ r' \ a$ **and**
 $a' \text{-pref-}a$: $a' \preceq_r a$
shows $a' \preceq_{r'} a$
 $\langle \text{proof} \rangle$

lemma *lifted-above-subset*:
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 $\text{above } r' \ a \subseteq \text{above } r \ a$
 $\langle \text{proof} \rangle$

lemma *lifted-above-mono*:
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: lifted A r r' a and
 a' -in- A -sub- a : $a' \in A - \{a\}$
shows *above r $a' \subseteq$ above r' $a' \cup \{a\}$*
 <proof>

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

fixes

$A :: 'a$ set and
 $A' :: 'a$ set and
 $r :: 'a$ Preference-Relation and
 $r' :: 'a$ Preference-Relation and
 $a :: 'a$

assumes

lifted: lifted A' r r' a and
subset: $A \subseteq A'$

shows *limit A $r =$ limit A $r' \vee$ lifted A (limit A r) (limit A $r')$ a*
 <proof>

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

fixes

$A :: 'a$ set and
 $A' :: 'a$ set and
 $r :: 'a$ Preference-Relation and
 $r' :: 'a$ Preference-Relation and
 $a :: 'a$

assumes

change: equiv-rel-except- a A' r r' a and
subset: $A \subseteq A'$ and
not-in- A : $a \notin A$

shows *limit A $r =$ limit A r'*
 <proof>

theorem *lifted-above-winner-alts:*

fixes

$A :: 'a$ set and
 $r :: 'a$ Preference-Relation and
 $r' :: 'a$ Preference-Relation and
 $a :: 'a$ and
 $a' :: 'a$

assumes

lifted-a: lifted A r r' a and
 a' -above- a' : above r $a' = \{a'\}$ and
fn- A : finite A

shows *above r' $a' = \{a'\} \vee$ above r' $a = \{a\}$*
 <proof>

theorem *lifted-above-winner-single:*

fixes

$A :: 'a$ set and

```

    r :: 'a Preference-Relation and
    r' :: 'a Preference-Relation and
    a :: 'a
  assumes
    lifted A r r' a and
    above r a = {a} and
    finite A
  shows above r' a = {a}
  <proof>

theorem lifted-above-winner-other:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation and
    r' :: 'a Preference-Relation and
    a :: 'a and
    a' :: 'a
  assumes
    lifted-a: lifted A r r' a and
    a'-above-a': above r' a' = {a'} and
    fin-A: finite A and
    a-not-a': a ≠ a'
  shows above r a' = {a'}
  <proof>

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**

$norm\ n \equiv \forall\ (x :: ereal\ list).\ n\ x \geq 0 \wedge (\forall\ i < length\ x.\ (x!i = 0) \longrightarrow n\ x = 0)$

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 $(\sum\ a \in A.\ g\ a) = (\sum\ a' \in A'.\ g\ (the-inv-into\ A\ f\ a'))$

<proof>

1.2.3 Common Norms

fun *l-one* $:: Norm$ **where**

$l-one\ x = (\sum\ i < length\ x.\ |x!i|)$

1.2.4 Properties

definition *symmetry* $:: Norm \Rightarrow bool$ **where**

$symmetry\ n \equiv \forall\ x\ y.\ x <^{~~} y \longrightarrow n\ x = n\ y$

1.2.5 Theorems

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

<proof>

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* \Rightarrow *bool* **where**

disjoint3 (*e*, *r*, *d*) =
 $((e \cap r = \{\}) \wedge$
 $(e \cap d = \{\}) \wedge$
 $(r \cap d = \{\}))$

fun *set-equals-partition* :: *'r set* \Rightarrow *'r Result* \Rightarrow *bool* **where**

set-equals-partition *X* (*e*, *r*, *d*) = (*e* \cup *r* \cup *d* = *X*)

1.3.2 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*) \Longrightarrow *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
           HOL–Combinatorics.Permutations
begin

```

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

1.4.1 Definition

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

many alternatives and voters.

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

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

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

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

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

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

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

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

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

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

1.4.2 Vote Count

lemma $sum-comp$:

fixes
 $f :: 'x \Rightarrow 'z :: comm-monoid-add$ **and**
 $g :: 'y \Rightarrow 'x$ **and**
 $X :: 'x \text{ set}$ **and**
 $Y :: 'y \text{ set}$
assumes $bij-betw \ g \ Y \ X$
shows $sum \ f \ X = sum \ (f \circ g) \ Y$
 $\langle proof \rangle$

lemma $vote-count-sum$:

fixes $E :: ('a, 'v) \text{ Election}$
assumes
 $finite \ (voters-\mathcal{E} \ E)$ **and**
 $finite \ (UNIV :: ('a \times 'a) \text{ set})$

shows $\text{sum } (\lambda p. \text{vote-count } p \ E) \ UNIV = \text{card } (\text{voters-}\mathcal{E} \ E)$
 $\langle \text{proof} \rangle$

1.4.3 Voter Permutations

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

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

lemma rename-sound :

fixes

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

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

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

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

assumes

$\text{prof}: \text{profile } V \ A \ p$ **and**

$\text{renamed}: (A, V', q) = \text{rename } \pi \ (A, V, p)$ **and**

$\text{bij}: \text{bij } \pi$

shows $\text{profile } V' \ A \ q$

$\langle \text{proof} \rangle$

lemma rename-finite :

fixes

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

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

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

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

assumes

$\text{finite-profile } V \ A \ p$ **and**

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

$\text{bij } \pi$

shows $\text{finite-profile } V' \ A \ q$

$\langle \text{proof} \rangle$

lemma rename-inv :

fixes

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

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

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

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

assumes $\text{bij } \pi$

shows $\text{rename } \pi \ (\text{rename } (\text{the-inv } \pi) \ (A, V, p)) = (A, V, p)$

$\langle \text{proof} \rangle$

lemma rename-inj :

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

assumes $\text{bij } \pi$

shows *inj* (*rename* π)
 $\langle proof \rangle$

lemma *rename-surj*:

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

assumes *bij* π

shows

on-valid-elections: *rename* π ' *valid-elections* = *valid-elections* **and**

on-finite-elections: *rename* π ' *finite-elections* = *finite-elections*

$\langle proof \rangle$

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* \Rightarrow (*'a, 'v*) *Profile*

\Rightarrow (*'a Preference-Relation*) *list* **where**

to-list *V* *p* = (*if* (*finite* *V*)

then (*map* *p* (*sorted-list-of-set* *V*))

else [])

lemma *map2-helper*:

fixes

f :: *'x* \Rightarrow *'y* \Rightarrow *'z* **and**

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

h :: *'y* \Rightarrow *'y* **and**

l :: *'x list* **and**

l' :: *'y list*

shows *map2* *f* (*map* *g* *l*) (*map* *h* *l'*) = *map2* (λ *x y. f* (*g* *x*) (*h* *y*)) *l* *l'*

$\langle proof \rangle$

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*)

$\langle proof \rangle$

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*)

$\langle proof \rangle$

lemma *set-card-upper-bound*:

```

fixes
   $i :: \text{nat}$  and
   $V :: \text{nat set}$ 
assumes
   $\text{fin-}V: \text{finite } V$  and
   $\text{bound-}v: \forall v \in V. v < i$ 
shows  $\text{card } V \leq i$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma sorted-list-of-set-nth-equals-card:
fixes
   $V :: 'v :: \text{linorder set}$  and
   $x :: 'v$ 
assumes
   $\text{fin-}V: \text{finite } V$  and
   $x\text{-}V: x \in V$ 
shows  $\text{sorted-list-of-set } V!(\text{card } \{v \in V. v < x\}) = x$ 
 $\langle \text{proof} \rangle$ 

```

```

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  $\text{bij } \pi$ 
shows
   $\text{let } \varphi = (\lambda i. \text{card } \{v \in \pi^{-1} V. v < \pi((\text{sorted-list-of-set } V)!i)\})$ 
   $\text{in } (\text{to-list } V p) = \text{permute-list } \varphi (\text{to-list } (\pi^{-1} V) (\lambda x. p(\text{the-inv } \pi x)))$ 
 $\langle \text{proof} \rangle$ 

```

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

```

```

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

```

```

lemma pref-count-voter-set-card:
fixes
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$  and
   $b :: 'a$ 

```

assumes *finite V*
shows *prefer-count V p a b ≤ card V*
 ⟨*proof*⟩

lemma *set-compr*:

fixes
A :: 'a set **and**
f :: 'a ⇒ 'a set
shows $\{f\ x \mid x. x \in A\} = f\ ` A$
 ⟨*proof*⟩

lemma *pref-count-set-compr*:

fixes
A :: 'a set **and**
V :: 'v set **and**
p :: ('a, 'v) Profile **and**
a :: 'a
shows $\{prefer-count\ V\ p\ a\ a' \mid a'. a' \in A - \{a\}\} =$
 $(prefer-count\ V\ p\ a)\ ` (A - \{a\})$
 ⟨*proof*⟩

lemma *pref-count*:

fixes
A :: 'a set **and**
V :: 'v set **and**
p :: ('a, 'v) Profile **and**
a :: 'a **and**
b :: 'a
assumes
prof: profile *V A p* **and**
fin: finite *V* **and**
a-in-A: *a* ∈ *A* **and**
b-in-A: *b* ∈ *A* **and**
neg: *a* ≠ *b*
shows *prefer-count V p a b = card V - (prefer-count V p b a)*
 ⟨*proof*⟩

lemma *pref-count-sym*:

fixes
p :: ('a, 'v) Profile **and**
V :: 'v set **and**
a :: 'a **and**
b :: 'a **and**
c :: 'a
assumes
pref-count-ineq: *prefer-count V p a c ≥ prefer-count V p c b* **and**
prof: profile *V A p* **and**
a-in-A: *a* ∈ *A* **and**
b-in-A: *b* ∈ *A* **and**

$c\text{-in-}A: c \in A$ **and**
 $a\text{-neq-}c: a \neq c$ **and**
 $c\text{-neq-}b: c \neq b$
shows $\text{prefer-count } V p b c \geq \text{prefer-count } V p c a$
 $\langle \text{proof} \rangle$

lemma *empty-prof-imp-zero-pref-count:*

fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $V :: 'v \text{ set}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes $V = \{\}$
shows $\text{prefer-count } V p a b = 0$
 $\langle \text{proof} \rangle$

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

lemma *wins-inf-voters:*

fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$ **and**
 $V :: 'v \text{ set}$
assumes *infinite* V
shows $\neg \text{wins } V b p a$
 $\langle \text{proof} \rangle$

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

lemma *wins-antisym:*

fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$ **and**
 $V :: 'v \text{ set}$
assumes $\text{wins } V a p b$ — This already implies that V is finite.
shows $\neg \text{wins } V b p a$
 $\langle \text{proof} \rangle$

lemma *wins-irreflex:*

fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$ **and**
 $V :: 'v \text{ set}$
shows $\neg \text{wins } V a p a$
 $\langle \text{proof} \rangle$

1.4.6 Condorcet Winner

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

lemma *cond-winner-unique-eq*:

fixes

V :: 'v set **and**

A :: 'a set **and**

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

a :: 'a **and**

b :: 'a

assumes

condorcet-winner V A p a **and**

condorcet-winner V A p b

shows b = a

<proof>

lemma *cond-winner-unique*:

fixes

A :: 'a set **and**

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

a :: 'a

assumes *condorcet-winner* V A p a

shows {a' \in A. *condorcet-winner* V A p a'} = {a}

<proof>

lemma *cond-winner-unique-2*:

fixes

V :: 'v set **and**

A :: 'a set **and**

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

a :: 'a **and**

b :: 'a

assumes

condorcet-winner V A p a **and**

b \neq a

shows \neg *condorcet-winner* V A p b

<proof>

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 \Rightarrow ('a, 'v) Profile \Rightarrow ('a, 'v) Profile **where**
limit-profile A p = (λ v. *limit* A (p v))

lemma *limit-prof-trans*:

fixes

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

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

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

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

assumes

$B \subseteq A$ **and**

$C \subseteq B$

shows $\text{limit-profile } C \ p = \text{limit-profile } C \ (\text{limit-profile } B \ p)$

$\langle \text{proof} \rangle$

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

$\text{profile } V \ B \ p$ **and**

$A \subseteq B$

shows $\text{profile } V \ A \ (\text{limit-profile } A \ p)$

$\langle \text{proof} \rangle$

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

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

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

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

$a :: 'a$

assumes *lifted* $V A p p' a$
shows *equiv-prof-except-a* $V A p p' a$
 <proof>

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

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

change: *equiv-prof-except-a* $V A' p q a$ **and**

subset: $A \subseteq A'$ **and**

not-in-A: $a \notin A$

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

— 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>

lemma *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

lifted-a: *lifted* $V A' p p' a$ **and**

subset: $A \subseteq A'$

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

$\vee \text{lifted } V A (\text{limit-profile } A p) (\text{limit-profile } A p') a$

<proof>

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 set and
  e :: 'a set and
  r :: 'a set and
  d :: 'a set
assumes well-formed-SCF A (e, r, d)
shows A - (e  $\cup$  d) = r
<proof>
```

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>
```

lemma *defer-subset*:

```
fixes
  A :: 'a set and
  r :: 'a Result
assumes well-formed-SCF A r
shows defer-r r  $\subseteq$  A
<proof>
```

lemma *elect-subset*:

```
fixes
  A :: 'a set and
  r :: 'a Result
assumes well-formed-SCF A r
shows elect-r r  $\subseteq$  A
<proof>
```



```

lemma reject-subset:
  fixes
     $A :: 'a \text{ set}$  and
     $r :: 'a \text{ Result}$ 
  assumes well-formed-SCF  $A \ r$ 
  shows  $\text{reject-}r \subseteq A$ 
   $\langle \text{proof} \rangle$ 

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

```

```

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

end

```

1.7 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.

$\langle ML \rangle$

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

$\langle proof \rangle$

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

global-interpretation committee-result:

result $\lambda A r. \text{set-equals-partition } (Pow A) r \wedge \text{disjoint3 } r$

$\lambda A rs. \{r \cap A \mid r. r \in rs\}$

$\langle proof \rangle$

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

$\langle proof \rangle$

$\langle ML \rangle$

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* :: $('x \Rightarrow 'y) \Rightarrow 'x \text{ set} \Rightarrow ('x \Rightarrow 'y)$ **where**

extensional-continuation $f s = (\lambda x. \text{if } (x \in s) \text{ then } (f x) \text{ else undefined})$

fun *preimg* :: $('x \Rightarrow 'y) \Rightarrow 'x \text{ set} \Rightarrow 'y \Rightarrow 'x \text{ set}$ **where**

preimg $f s x = \{x' \in s. f x' = x\}$

1.8.2 Relations for Symmetry Constructions

fun *restricted-rel* :: 'x rel \Rightarrow 'x set \Rightarrow 'x set \Rightarrow 'x rel **where**
restricted-rel r s s' = r \cap s \times s'

fun *closed-restricted-rel* :: 'x rel \Rightarrow 'x set \Rightarrow 'x set \Rightarrow bool **where**
closed-restricted-rel r s t = ((*restricted-rel* r t s) “ t \subseteq t)

fun *action-induced-rel* :: 'x set \Rightarrow 'y set \Rightarrow ('x, 'y) binary-fun \Rightarrow 'y rel **where**
action-induced-rel s t φ = {(y, y') \in t \times t. \exists x \in s. φ 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 φ =
 {((u, v), (x, y)). (u, v) \in t \times t \wedge (\exists z \in s. x = φ z u \wedge y = φ 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 ψ x = image (ψ x)

1.8.3 Invariance and Equivariance

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

datatype ('x, 'y) *symmetry* =
 Invariance 'x rel |
 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 τ) =
 (\forall (φ , ψ) \in τ . \forall x \in s. φ x \in s \longrightarrow f (φ x) = ψ (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 φ ψ = Equivariance t {(φ x, ψ x) | x. x \in s}

1.8.4 Auxiliary Lemmas

lemma *inj-imp-inj-on-set-system*:
 fixes f :: 'x \Rightarrow 'y

assumes *inj* *f*
shows *inj* $(\lambda s. \{f \text{ ` } x \mid x. x \in s\})$
 $\langle \text{proof} \rangle$

lemma *inj-and-surj-imp-surj-on-set-system*:
fixes *f* :: 'x \Rightarrow 'y
assumes
 inj *f* **and**
 surj *f*
shows *surj* $(\lambda s. \{f \text{ ` } x \mid x. x \in s\})$
 $\langle \text{proof} \rangle$

lemma *bij-imp-bij-on-set-system*:
fixes *f* :: 'x \Rightarrow 'y
assumes *bij* *f*
shows *bij* $(\lambda s. \{f \text{ ` } x \mid x. x \in s\})$
 $\langle \text{proof} \rangle$

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

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

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

1.8.5 Rewrite Rules

theorem *rewrite-invar-as-equivar*:
fixes

$f :: 'x \Rightarrow 'y$ **and**
 $s :: 'x \text{ set}$ **and**
 $t :: 'z \text{ set}$ **and**
 $\varphi :: ('z, 'x) \text{ binary-fun}$
shows $\text{is-symmetry } f \ (\text{Invariance } (\text{action-induced-rel } t \ s \ \varphi)) =$
 $\text{is-symmetry } f \ (\text{action-induced-equivariance } t \ s \ \varphi \ (\lambda \ g. \text{id}))$
 $\langle \text{proof} \rangle$

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 $\text{is-symmetry } f \ (\text{Invariance } (\text{action-induced-rel } s \ t \ \varphi)) =$
 $(\forall \ x \in s. \forall \ y \in t. \varphi \ x \ y \in t \longrightarrow f \ y = f \ (\varphi \ x \ y))$
 $\langle \text{proof} \rangle$

lemma *rewrite-equivariance*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $s :: 'z \text{ set}$ **and**
 $t :: 'x \text{ set}$ **and**
 $\varphi :: ('z, 'x) \text{ binary-fun}$ **and**
 $\psi :: ('z, 'y) \text{ binary-fun}$
shows $\text{is-symmetry } f \ (\text{action-induced-equivariance } s \ t \ \varphi \ \psi) =$
 $(\forall \ x \in s. \forall \ y \in t. \varphi \ x \ y \in t \longrightarrow f \ (\varphi \ x \ y) = \psi \ x \ (f \ y))$
 $\langle \text{proof} \rangle$

lemma *rewrite-group-action-img*:
fixes
 $m :: 'x \text{ monoid}$ **and**
 $s :: 'y \text{ set}$ **and**
 $\varphi :: ('x, 'y) \text{ binary-fun}$ **and**
 $t :: 'y \text{ set}$ **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 $\varphi \ (x \otimes_m y) \ 't = \varphi \ x \ ' \varphi \ y \ 't$
 $\langle \text{proof} \rangle$

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

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

fixes $f :: 'a \Rightarrow 'a$
assumes $f \in \text{carrier } (\text{BijGroup } \text{UNIV})$
shows $\text{bij } f$
 $\langle \text{proof} \rangle$

lemma *rewrite-sym-group:*

fixes
 $f :: 'a \Rightarrow 'a$ **and**
 $g :: 'a \Rightarrow 'a$ **and**
 $s :: 'a \text{ set}$
assumes
 $f \in \text{carrier } (\text{BijGroup } s)$ **and**
 $g \in \text{carrier } (\text{BijGroup } s)$
shows
 $\text{rewrite-mult: } f \otimes \text{BijGroup } s \ g = \text{extensional-continuation } (f \circ g) \ s$ **and**
 $\text{rewrite-mult-univ: } s = \text{UNIV} \longrightarrow f \otimes \text{BijGroup } s \ g = f \circ g$
 $\langle \text{proof} \rangle$

lemma *simp-extensional-univ:*

fixes $f :: 'a \Rightarrow 'b$
shows $\text{extensional-continuation } f \ \text{UNIV} = f$
 $\langle \text{proof} \rangle$

lemma *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 $\text{extensional-continuation } f \ s \ x = \text{extensional-continuation } f \ t \ x$
 $\langle \text{proof} \rangle$

lemma *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 $\text{action-induced-rel } s \ u \ \psi = \text{Restr } (\text{action-induced-rel } s \ t \ \varphi) \ u$
 $\langle \text{proof} \rangle$

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  $\text{action-induced-rel } s \ t \ \varphi = \text{action-induced-rel } s \ t \ \psi$ 
<proof>

```

1.8.6 Group Actions

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

```

fixes  $m :: 'x \text{ monoid}$ 
assumes  $\text{group } m$ 
shows  $\text{group-action } m \ \text{UNIV} \ (\lambda x. \text{id})$ 
<proof>

```

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  $\varphi\text{-img} \equiv (\lambda x. \text{extensional-continuation } (\text{image } (\varphi x)) \ (\text{Pow } s))$ 
assumes  $\text{group-action } m \ s \ \varphi$ 
shows  $\text{group-action } m \ (\text{Pow } s) \ \varphi\text{-img}$ 
<proof>

```

1.8.7 Invariance and Equivariance

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

theorem *equivar-generators-imp-equivar-group:*

```

fixes
   $f :: 'x \Rightarrow 'y$  and
   $m :: 'z \text{ monoid}$  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
   $\text{equivar: is-symmetry } f \ (\text{action-induced-equivariance } s \ t \ \varphi \ \psi)$  and
   $\text{action-}\varphi: \text{group-action } m \ t \ \varphi$  and
   $\text{action-}\psi: \text{group-action } m \ (f \circ t) \ \psi$  and
   $\text{gen: carrier } m = \text{generate } m \ s$ 
shows  $\text{is-symmetry } f \ (\text{action-induced-equivariance } (\text{carrier } m) \ t \ \varphi \ \psi)$ 
<proof>

```

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 $\text{is-symmetry } (\lambda x. f\ x\ x) \text{ (Invariance } r)$
 $\langle \text{proof} \rangle$

lemma *invar-under-subset-rel*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $r :: 'x \text{ rel}$
assumes
 $\text{subset: } r \subseteq \text{rel}$ **and**
 $\text{invar: is-symmetry } f \text{ (Invariance rel)}$
shows $\text{is-symmetry } f \text{ (Invariance } r)$
 $\langle \text{proof} \rangle$

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 $\text{is-symmetry } f \text{ (action-induced-equivariance } s\ t\ \varphi\ \psi) =$
 $\text{is-symmetry } f \text{ (action-induced-equivariance } s\ t\ \varphi'\ \psi)$
 $\langle \text{proof} \rangle$

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
 $\text{is-symmetry } f \text{ (Equivariance } s\ \tau)$ **and**
 $t \subseteq s$
shows $\text{is-symmetry } f \text{ (Equivariance } t\ \tau)$
 $\langle \text{proof} \rangle$

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
 $\text{is-symmetry } f \text{ (Equivariance } s \tau)$ **and**
 $v \subseteq \tau$
shows $\text{is-symmetry } f \text{ (Equivariance } s v)$
 $\langle \text{proof} \rangle$

theorem *group-action-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 $\text{equivar-prop} \equiv \text{action-induced-equivariance (carrier } m) \mathcal{D}_f \varphi \psi$
assumes
 $\text{action-}\varphi$: $\text{group-action } m \ s \ \varphi$ **and**
 action-res : $\text{group-action } m \ \text{UNIV } \psi$ **and**
 $\text{dom-in-}s$: $\mathcal{D}_f \subseteq s$ **and**
 closed-domain :
 $\text{closed-restricted-rel (action-induced-rel (carrier } m) \ s \ \varphi) \ s \ \mathcal{D}_f$ **and**
 $\text{equivar-}f$: $\text{is-symmetry } f \ \text{equivar-prop}$ **and**
 $\text{group-elem-}x$: $x \in \text{carrier } m$
shows $\forall y. \text{preimg } f \ \mathcal{D}_f \ (\psi \ x \ y) = (\varphi \ x) \text{ ` (preimg } f \ \mathcal{D}_f \ y)$
 $\langle \text{proof} \rangle$

Invariance and Equivariance Function Composition

lemma *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 $\text{is-symmetry } (g \circ f) \text{ (Invariance } r)$
 $\langle \text{proof} \rangle$

lemma *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
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 ' *s* \subseteq *t* **and**
is-symmetry *f* (*Equivariance* *s* τ) **and**
is-symmetry *g* (*Equivariance* *t* v)
shows *is-symmetry* (*g* \circ *f*) (*Equivariance* *s* *transitive-acts*)
 <proof>

lemma *equivar-ind-by-action-comp*:
fixes
f :: *'x* \Rightarrow *'y* **and**
g :: *'y* \Rightarrow *'z* **and**
s :: *'w* *set* **and**
t :: *'x* *set* **and**
u :: *'y* *set* **and**
 $\varphi :: ('w, 'x)$ *binary-fun* **and**
 $\chi :: ('w, 'y)$ *binary-fun* **and**
 $\psi :: ('w, 'z)$ *binary-fun*
assumes
f ' *t* \subseteq *u* **and**
 $\forall x \in s. \chi \ x \text{ ' } f \text{ ' } t \subseteq u$ **and**
is-symmetry *f* (*action-induced-equivariance* *s* *t* $\varphi \ \chi$) **and**
is-symmetry *g* (*action-induced-equivariance* *s* *u* $\chi \ \psi$)
shows *is-symmetry* (*g* \circ *f*) (*action-induced-equivariance* *s* *t* $\varphi \ \psi$)
 <proof>

lemma *equivar-set-minus*:
fixes
f :: *'x* \Rightarrow *'y* *set* **and**
g :: *'x* \Rightarrow *'y* *set* **and**
s :: *'z* *set* **and**
t :: *'x* *set* **and**
 $\varphi :: ('z, 'x)$ *binary-fun* **and**
 $\psi :: ('z, 'y)$ *binary-fun*
assumes
f-equivar: *is-symmetry* *f* (*action-induced-equivariance* *s* *t* φ (*set-action* ψ)) **and**
g-equivar: *is-symmetry* *g* (*action-induced-equivariance* *s* *t* φ (*set-action* ψ)) **and**
bij-a: $\forall a \in s. \text{bij } (\psi \ a)$
shows
is-symmetry ($\lambda b. f \ b - g \ b$) (*action-induced-equivariance* *s* *t* φ (*set-action* ψ))
 <proof>

lemma *equivar-union-under-image-action*:
fixes
f :: *'x* \Rightarrow *'y* **and**
s :: *'z* *set* **and**
 $\varphi :: ('z, 'x)$ *binary-fun*

```

    shows is-symmetry  $\cup$  (action-induced-equivariance s UNIV
      (set-action (set-action  $\varphi$ )) (set-action  $\varphi$ ))
  <proof>

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 'elect-r r,  $\psi$  x 'reject-r r,  $\psi$  x 'defer-r r))

```

Anonymity

```

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

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

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

```

Neutrality

```

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

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

definition neutrality $\mathcal{G}$  :: ('a  $\Rightarrow$  'a) monoid where

```

$neutrality_{\mathcal{G}} = \text{BijGroup } (UNIV::'a \text{ set})$
fun $\varphi\text{-neutr} :: ('a, 'v) \text{ Election set} \Rightarrow ('a \Rightarrow 'a, ('a, 'v) \text{ Election}) \text{ binary-fun where}$
 $\varphi\text{-neutr } \mathcal{E} \ \pi = \text{extensional-continuation } (\text{alternatives-rename } \pi) \ \mathcal{E}$
fun $neutrality_{\mathcal{R}} :: ('a, 'v) \text{ Election set} \Rightarrow ('a, 'v) \text{ Election rel where}$
 $neutrality_{\mathcal{R}} \ \mathcal{E} = \text{action-induced-rel } (\text{carrier } neutrality_{\mathcal{G}}) \ \mathcal{E} \ (\varphi\text{-neutr } \mathcal{E})$
fun $\psi\text{-neutr}_c :: ('a \Rightarrow 'a, 'a) \text{ binary-fun where}$
 $\psi\text{-neutr}_c \ \pi \ r = \pi \ r$
fun $\psi\text{-neutr}_w :: ('a \Rightarrow 'a, 'a \text{ rel}) \text{ binary-fun where}$
 $\psi\text{-neutr}_w \ \pi \ r = \text{rel-rename } \pi \ r$

Homogeneity

fun $homogeneity_{\mathcal{R}} :: ('a, 'v) \text{ Election set} \Rightarrow ('a, 'v) \text{ Election rel where}$
 $homogeneity_{\mathcal{R}} \ \mathcal{E} =$
 $\{(E, E') \in \mathcal{E} \times \mathcal{E}.$
 $\quad \text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E'$
 $\quad \wedge \text{finite } (\text{voters-}\mathcal{E} \ E) \wedge \text{finite } (\text{voters-}\mathcal{E} \ E')$
 $\quad \wedge (\exists \ n > 0. \forall \ r::('a \text{ Preference-Relation}).$
 $\quad \quad \text{vote-count } r \ E = n * (\text{vote-count } r \ E'))\}$
fun $copy\text{-list} :: \text{nat} \Rightarrow 'x \text{ list} \Rightarrow 'x \text{ list where}$
 $copy\text{-list} \ 0 \ l = []$
 $copy\text{-list} \ (\text{Suc } n) \ l = copy\text{-list } n \ l @ l$
fun $homogeneity_{\mathcal{R}}' :: ('a, 'v::\text{linorder}) \text{ Election set} \Rightarrow ('a, 'v) \text{ Election rel where}$
 $homogeneity_{\mathcal{R}}' \ \mathcal{E} =$
 $\{(E, E') \in \mathcal{E} \times \mathcal{E}.$
 $\quad \text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E'$
 $\quad \wedge \text{finite } (\text{voters-}\mathcal{E} \ E) \wedge \text{finite } (\text{voters-}\mathcal{E} \ E')$
 $\quad \wedge (\exists \ n > 0.$
 $\quad \quad \text{to-list } (\text{voters-}\mathcal{E} \ E') \ (\text{profile-}\mathcal{E} \ E') =$
 $\quad \quad \text{copy-list } n \ (\text{to-list } (\text{voters-}\mathcal{E} \ E) \ (\text{profile-}\mathcal{E} \ E)))\}$

Reversal Symmetry

fun $rev\text{-rel} :: 'a \text{ rel} \Rightarrow 'a \text{ rel where}$
 $rev\text{-rel } r = \{(a, b). (b, a) \in r\}$
fun $rel\text{-app} :: ('a \text{ rel} \Rightarrow 'a \text{ rel}) \Rightarrow ('a, 'v) \text{ Election} \Rightarrow ('a, 'v) \text{ Election where}$
 $rel\text{-app } f \ (A, V, p) = (A, V, f \circ p)$
definition $reversal_{\mathcal{G}} :: ('a \text{ rel} \Rightarrow 'a \text{ rel}) \text{ monoid where}$
 $reversal_{\mathcal{G}} = (\text{carrier} = \{rev\text{-rel}, id\}, \text{monoid.mult} = \text{comp}, \text{one} = id)$
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**
 $\text{reversal}_{\mathcal{R}} \mathcal{E} = \text{action-induced-rel } (\text{carrier reversal}_{\mathcal{G}}) \mathcal{E} (\varphi\text{-rev } \mathcal{E})$

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

$\langle \text{proof} \rangle$

lemma $n\text{-app-leaves-set}$:

fixes

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

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

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

$x :: 'x$

assumes

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

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

$x\text{-el}$: $x \in A - B$ **and**

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

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

$n > 0$ **and**

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

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

$\langle \text{proof} \rangle$

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

fixes

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

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

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

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

$m :: \text{nat}$ **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**
 $n\text{-app-eq-}m\text{-}n$: $n\text{-app } n \text{ } f \text{ } x = n\text{-app } m \text{ } f \text{ } y$ **and**
 $n\text{-app-}x\text{-in-}A$: $\forall \ n' < n. \ n\text{-app } n' \text{ } f \text{ } x \in A$ **and**
 $n\text{-app-}y\text{-in-}A$: $\forall \ m' < m. \ n\text{-app } m' \text{ } f \text{ } y \in A$ **and**
 $\text{fin-}A$: *finite* A **and**
 $\text{fin-}B$: *finite* B **and**
 $\text{bij-}f\text{-}A\text{-}B$: $\text{bij-betw } f \text{ } A \text{ } B$
shows $n\text{-app } (n - m) \text{ } f \text{ } x = y$
 $\langle \text{proof} \rangle$

lemma $n\text{-app-inv}$:

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**
 $\forall \ m \geq 0. \ m < n \longrightarrow n\text{-app } m \text{ } (\text{the-inv-into } A \text{ } f) \text{ } x \in B$ **and**
 $\text{bij-betw } f \text{ } A \text{ } B$
shows $n\text{-app } n \text{ } f \text{ } (n\text{-app } n \text{ } (\text{the-inv-into } A \text{ } f) \text{ } x) = x$
 $\langle \text{proof} \rangle$

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

fixes
 $A :: 'x \text{ set}$ **and**
 $B :: 'x \text{ set}$ **and**
 $f :: 'x \Rightarrow 'x$
assumes
 $\text{fin-}A$: *finite* A **and**
 $\text{fin-}B$: *finite* B **and**
 bij : $\text{bij-betw } f \text{ } A \text{ } B$
obtains $g :: 'x \Rightarrow 'x$ **where**
 $\text{bij } g$ **and**
 $\forall \ a \in A. \ g \ a = f \ a$ **and**
 $\forall \ b \in B - A. \ g \ b \in A - B \wedge (\exists \ n > 0. \ n\text{-app } n \text{ } f \text{ } (g \ b) = b)$ **and**
 $\forall \ x \in \text{UNIV} - A - B. \ g \ x = x$
 $\langle \text{proof} \rangle$

lemma bij-betw-ext :

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

$\langle \text{proof} \rangle$

1.9.3 Anonymity Lemmas

lemma *anon-rel-vote-count*:

fixes

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

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

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

assumes

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

$(E, E') \in \text{anonymity}_{\mathcal{R}} \mathcal{E}$

shows alternatives- \mathcal{E} $E = \text{alternatives-}\mathcal{E} \ E' \wedge (E, E') \in \mathcal{E} \times \mathcal{E}$
 $\wedge (\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E')$

$\langle \text{proof} \rangle$

lemma *vote-count-anon-rel*:

fixes

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

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

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

assumes

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

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

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

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

eq: 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 $(E, E') \in \text{anonymity}_{\mathcal{R}} \mathcal{E}$

$\langle \text{proof} \rangle$

lemma *rename-comp*:

fixes

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

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

assumes

bij π **and**

bij π'

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

$\langle \text{proof} \rangle$

interpretation *anonymous-group-action*:

group-action *anonymity_G* *valid-elections* φ -anon *valid-elections*

$\langle \text{proof} \rangle$

lemma (in result) *well-formed-res-anon*:

is-symmetry $(\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$

$(\text{Invariance } (\text{anonymity}_{\mathcal{R}} \ \text{valid-elections}))$

$\langle \text{proof} \rangle$

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* π
shows $(\pi \ a, \pi \ b) \in \{(\pi \ x, \pi \ y) \mid x \ y. (x, y) \in r\}$
 $\longleftrightarrow (a, b) \in \{(x, y) \mid x \ y. (x, y) \in r\}$
 $\langle \text{proof} \rangle$

lemma *rel-rename-comp*:

fixes
 $\pi :: 'a \Rightarrow 'a$ **and**
 $\pi' :: 'a \Rightarrow 'a$
shows $\text{rel-rename } (\pi \circ \pi') = \text{rel-rename } \pi \circ \text{rel-rename } \pi'$
 $\langle \text{proof} \rangle$

lemma *rel-rename-sound*:

fixes
 $\pi :: 'a \Rightarrow 'a$ **and**
 $r :: 'a \text{ rel}$ **and**
 $A :: 'a \text{ set}$
assumes *inj* π
shows
 $\text{refl-on } A \ r \longrightarrow \text{refl-on } (\pi \ ` \ A) \ (\text{rel-rename } \pi \ r)$ **and**
 $\text{antisym } r \longrightarrow \text{antisym } (\text{rel-rename } \pi \ r)$ **and**
 $\text{total-on } A \ r \longrightarrow \text{total-on } (\pi \ ` \ A) \ (\text{rel-rename } \pi \ r)$ **and**
 $\text{Relation.trans } r \longrightarrow \text{Relation.trans } (\text{rel-rename } \pi \ r)$
 $\langle \text{proof} \rangle$

lemma *rename-subset*:

fixes
 $r :: 'a \text{ rel}$ **and**
 $s :: 'a \text{ rel}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$ **and**
 $\pi :: 'a \Rightarrow 'a$
assumes
 $\text{bij-}\pi$: *bij* π **and**
 $\text{rel-rename } \pi \ r = \text{rel-rename } \pi \ s$ **and**
 $(a, b) \in r$
shows $(a, b) \in s$
 $\langle \text{proof} \rangle$

lemma *rel-rename-bij*:

fixes $\pi :: 'a \Rightarrow 'a$
assumes $\text{bij-}\pi$: *bij* π

shows $\text{bij } (\text{rel-rename } \pi)$
 $\langle \text{proof} \rangle$

lemma *alternatives-rename-comp*:

fixes

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

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

shows

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

$\langle \text{proof} \rangle$

lemma *valid-elects-closed*:

fixes

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

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

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

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

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

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

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

assumes

$\text{bij-}\pi$: $\text{bij } \pi$ **and**

valid-elects : $(A, V, p) \in \text{valid-elections}$ **and**

renamed : $(A', V', p') = \text{alternatives-rename } \pi (A, V, p)$

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

$\langle \text{proof} \rangle$

lemma *alternatives-rename-bij*:

fixes $\pi :: ('a \Rightarrow 'a)$

assumes $\text{bij-}\pi$: $\text{bij } \pi$

shows $\text{bij-betw } (\text{alternatives-rename } \pi) \text{ valid-elections valid-elections}$

$\langle \text{proof} \rangle$

interpretation $\varphi\text{-neutral-action}$:

$\text{group-action neutrality}_{\mathcal{G}} \text{ valid-elections } \varphi\text{-neutr valid-elections}$

$\langle \text{proof} \rangle$

interpretation $\psi\text{-neutral}_{\text{c}}\text{-action}$: $\text{group-action neutrality}_{\mathcal{G}} \text{ UNIV } \psi\text{-neutr}_{\text{c}}$

$\langle \text{proof} \rangle$

interpretation $\psi\text{-neutral}_{\text{w}}\text{-action}$: $\text{group-action neutrality}_{\mathcal{G}} \text{ UNIV } \psi\text{-neutr}_{\text{w}}$

$\langle \text{proof} \rangle$

lemma *wf-result-neutrality-SCF*:

$\text{is-symmetry } (\lambda \mathcal{E}. \text{limit-set-SCF } (\text{alternatives-}\mathcal{E} \mathcal{E}) \text{ UNIV})$

$(\text{action-induced-equivariance } (\text{carrier neutrality}_{\mathcal{G}}) \text{ valid-elections})$

$(\varphi\text{-neutr valid-elections}) (\text{set-action } \psi\text{-neutr}_{\text{c}}))$

$\langle \text{proof} \rangle$

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_G*) *valid-elections*
 (φ -*neutr valid-elections*) (*set-action* ψ -*neutr_w*))
<proof>

1.9.5 Homogeneity Lemmas

lemma *refl-homogeneity_R*:
fixes $\mathcal{E} :: ('a, 'v) \text{ Election set}$
assumes $\mathcal{E} \subseteq \text{finite-elections-}\mathcal{V}$
shows *refl-on* \mathcal{E} (*homogeneity_R* \mathcal{E})
<proof>

lemma (*in result*) *well-formed-res-homogeneity*:
is-symmetry ($\lambda \mathcal{E}. \text{limit-set } (\text{alternatives-}\mathcal{E} \ \mathcal{E}) \text{ UNIV}$)
 (*Invariance* (*homogeneity_R* UNIV))
<proof>

lemma *refl-homogeneity_R'*:
fixes $\mathcal{E} :: ('a, 'v::\text{linorder}) \text{ Election set}$
assumes $\mathcal{E} \subseteq \text{finite-elections-}\mathcal{V}$
shows *refl-on* \mathcal{E} (*homogeneity_R'* \mathcal{E})
<proof>

lemma (*in result*) *well-formed-res-homogeneity'*:
is-symmetry ($\lambda \mathcal{E}. \text{limit-set } (\text{alternatives-}\mathcal{E} \ \mathcal{E}) \text{ UNIV}$)
 (*Invariance* (*homogeneity_R'* UNIV))
<proof>

1.9.6 Reversal Symmetry Lemmas

lemma *rev-rev-id*: *rev-rel* \circ *rev-rel* = *id*
<proof>

lemma *rev-rel-limit*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ rel}$
shows *rev-rel* (*limit* $A \ r$) = *limit* A (*rev-rel* r)
<proof>

lemma *rev-rel-lin-ord*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ rel}$
assumes *linear-order-on* $A \ r$
shows *linear-order-on* A (*rev-rel* r)
<proof>

interpretation *reversal_G-group: group reversal_G*
 ⟨*proof*⟩

interpretation *φ-reverse-action:*
group-action reversal_G valid-elections φ-rev valid-elections
 ⟨*proof*⟩

interpretation *ψ-reverse-action: group-action reversal_G UNIV ψ-rev*
 ⟨*proof*⟩

lemma *φ-ψ-rev-well-formed:*
shows *is-symmetry (λ \mathcal{E} . limit-set-SWF (alternatives- \mathcal{E} \mathcal{E}) UNIV)*
(action-induced-equivariance (carrier reversal_G) valid-elections
(φ-rev valid-elections) (set-action ψ-rev))
 ⟨*proof*⟩

end

1.10 Result-Dependent Voting Rule Properties

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

1.10.1 Properties Dependent on the Result Type

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

locale *result-properties = result +*
fixes *ψ-neutr :: ('a ⇒ 'a, 'b) binary-fun and*
 \mathcal{E} :: ('a, 'v) Election
assumes
act-neutr: group-action neutrality_G UNIV ψ-neutr and
well-formed-res-neutr:
is-symmetry (λ \mathcal{E} :: ('a, 'v) Election. limit-set (alternatives- \mathcal{E} \mathcal{E}) UNIV)
(action-induced-equivariance (carrier neutrality_G)
valid-elections (φ-neutr valid-elections) (set-action ψ-neutr))
sublocale *result-properties ⊆ result*
 ⟨*proof*⟩

1.10.2 Interpretations

global-interpretation *SCF-properties:*

result-properties well-formed-SCF limit-set-SCF ψ -neutr_c
⟨proof⟩

global-interpretation *SWF-properties:*

result-properties well-formed-SWF limit-set-SWF ψ -neutr_w
⟨proof⟩

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
   $\langle$ proof $\rangle$ 
```

```
lemma list-cons-presv-finiteness:
  fixes
    A :: 'a set and
    S :: 'a list set
```

```

assumes
  fin-A: finite A and
  fin-B: finite S
shows finite {a#l | a l. a ∈ A ∧ l ∈ S}
⟨proof⟩

```

```

lemma listset-finiteness:
  fixes l :: 'a set list
  assumes  $\forall i::nat. i < length\ l \longrightarrow finite\ (!i)$ 
  shows finite (listset l)
  ⟨proof⟩

```

```

lemma all-ls-elems-same-len:
  fixes l :: 'a set list
  shows  $\forall l'::('a\ list). l' \in listset\ l \longrightarrow length\ l' = length\ l$ 
  ⟨proof⟩

```

```

lemma all-ls-elems-in-ls-set:
  fixes l :: 'a set list
  shows  $\forall l' \in listset\ l. \forall i::nat < length\ l'. l'!i \in !i$ 
  ⟨proof⟩

```

```

lemma all-ls-in-ls-set:
  fixes l :: 'a set list
  shows  $\forall l'. length\ l' = length\ l$ 
   $\wedge (\forall i < length\ l'. l'!i \in !i) \longrightarrow l' \in listset\ l$ 
  ⟨proof⟩

```

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 ∈ 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
  ⟨proof⟩

```

```

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

```

$\langle proof \rangle$

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
 $\langle proof \rangle$

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

lemma *rel-trans*:
fixes *l* :: 'a Preference-List
shows trans (*pl- α* *l*)
 $\langle proof \rangle$

lemma *pl- α -lin-order*:
fixes
A :: 'a set **and**
r :: 'a rel
assumes *r* \in *pl- α* 'permutations-of-set *A*
shows linear-order-on *A* *r*
 $\langle proof \rangle$

lemma *lin-order-pl- α* :
fixes
r :: 'a rel **and**
A :: 'a set
assumes
lin-order: linear-order-on *A* *r* **and**
fin: finite *A*
shows *r* \in *pl- α* 'permutations-of-set *A*
 $\langle proof \rangle$

lemma *index-helper*:
fixes
l :: 'x list **and**
x :: 'x

assumes
finite (*set l*) **and**
distinct l **and**
 $x \in \text{set } l$
shows $\text{index } l \ x = \text{card } \{y \in \text{set } l. \text{index } l \ y < \text{index } l \ x\}$
 <proof>

lemma *pl- α -eq-imp-list-eq*:
fixes
 $l :: 'x \text{ list}$ **and**
 $l' :: 'x \text{ list}$
assumes
fin-set-l: *finite* (*set l*) **and**
set-eq: $\text{set } l = \text{set } l'$ **and**
dist-l: *distinct l* **and**
dist-l': *distinct l'* **and**
pl- α -eq: $\text{pl-}\alpha \ l = \text{pl-}\alpha \ l'$
shows $l = l'$
 <proof>

lemma *pl- α -bij-betw*:
fixes $X :: 'x \text{ set}$
assumes *finite X*
shows *bij-betw pl- α (permutations-of-set X) {r. linear-order-on X r}*
 <proof>

2.1.5 Limited Preference

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

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

lemma *limited-dest*:
fixes
 $A :: 'a \text{ set}$ **and**
 $l :: 'a \text{ Preference-List}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
 $a \lesssim_l b$ **and**
limited A l
shows $a \in A \wedge b \in A$
 <proof>

lemma *limit-equiv*:
fixes
 $A :: 'a \text{ set}$ **and**

$l :: 'a \text{ list}$
assumes *well-formed-l l*
shows $pl\text{-}\alpha \text{ (limit-}l \ A \ l) = \text{limit } A \ (pl\text{-}\alpha \ l)$
 <proof>

2.1.6 Auxiliary Definitions

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

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

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

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

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

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

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

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

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

2.1.7 Auxiliary Lemmas

lemma *list-trans[simp]*:
fixes $l :: 'a \text{ Preference-List}$
shows *trans l*
 <proof>

lemma *list-antisym[simp]*:
fixes $l :: 'a \text{ Preference-List}$
shows *antisym-l l*
 <proof>

lemma *lin-order-equiv-list-of-alts*:
fixes
 $A :: 'a \text{ set}$ **and**
 $l :: 'a \text{ Preference-List}$

shows *linear-order-on-l* $A\ l = (A = \text{set } l)$
 $\langle \text{proof} \rangle$

lemma *connex-imp-refl*:
fixes
 $A :: 'a\ \text{set}$ **and**
 $l :: 'a\ \text{Preference-List}$
assumes *connex-l* $A\ l$
shows *refl-on-l* $A\ l$
 $\langle \text{proof} \rangle$

lemma *lin-ord-imp-connex-l*:
fixes
 $A :: 'a\ \text{set}$ **and**
 $l :: 'a\ \text{Preference-List}$
assumes *linear-order-on-l* $A\ l$
shows *connex-l* $A\ l$
 $\langle \text{proof} \rangle$

lemma *above-trans*:
fixes
 $l :: 'a\ \text{Preference-List}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
trans l **and**
 $a \lesssim_l b$
shows $\text{set } (\text{above-}l\ l\ b) \subseteq \text{set } (\text{above-}l\ l\ a)$
 $\langle \text{proof} \rangle$

lemma *less-preferred-l-rel-equiv*:
fixes
 $l :: 'a\ \text{Preference-List}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
shows $a \lesssim_l b =$
 $\text{Preference-Relation.is-less-preferred-than } a\ (\text{pl-}\alpha\ l)\ b$
 $\langle \text{proof} \rangle$

theorem *above-equiv*:
fixes
 $l :: 'a\ \text{Preference-List}$ **and**
 $a :: 'a$
shows $\text{set } (\text{above-}l\ l\ a) = \text{above } (\text{pl-}\alpha\ l)\ a$
 $\langle \text{proof} \rangle$

theorem *rank-equiv*:
fixes
 $l :: 'a\ \text{Preference-List}$ **and**

$a :: 'a$
assumes *well-formed-l l*
shows $\text{rank-l } l \ a = \text{rank } (pl\text{-}\alpha \ l) \ a$
 $\langle \text{proof} \rangle$

lemma *lin-ord-equiv*:
fixes
 $A :: 'a \text{ set}$ **and**
 $l :: 'a \text{ Preference-List}$
shows $\text{linear-order-on-l } A \ l = \text{linear-order-on } A \ (pl\text{-}\alpha \ l)$
 $\langle \text{proof} \rangle$

2.1.8 First Occurrence Indices

lemma *pos-in-list-yields-rank*:
fixes
 $l :: 'a \text{ Preference-List}$ **and**
 $a :: 'a$ **and**
 $n :: \text{nat}$
assumes
 $\forall (j :: \text{nat}). j \leq n. l!j \neq a$ **and**
 $l!(n - 1) = a$
shows $\text{rank-l } l \ a = n$
 $\langle \text{proof} \rangle$

lemma *ranked-alt-not-at-pos-before*:
fixes
 $l :: 'a \text{ Preference-List}$ **and**
 $a :: 'a$ **and**
 $n :: \text{nat}$
assumes
 $a \in \text{set } l$ **and**
 $n < (\text{rank-l } l \ a) - 1$
shows $l!n \neq a$
 $\langle \text{proof} \rangle$

lemma *pos-in-list-yields-pos*:
fixes
 $l :: 'a \text{ Preference-List}$ **and**
 $a :: 'a$
assumes $a \in \text{set } l$
shows $l!(\text{rank-l } l \ a - 1) = a$
 $\langle \text{proof} \rangle$

lemma *rel-of-pref-pred-for-set-eq-list-to-rel*:
fixes $l :: 'a \text{ Preference-List}$
shows $\text{relation-of } (\lambda y z. y \lesssim_l z) \ (\text{set } l) = pl\text{-}\alpha \ l$
 $\langle \text{proof} \rangle$

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 ( $n < \text{length } pl \wedge n \geq 0$ )
                        then (map (Preference-List.pl- $\alpha$ ) pl)!n
                        else { })
```

```
lemma prof-abstr-presv-size:
  fixes p ::  $'a$  Profile-List
  shows length p = length (to-list {0 ..< length p} (pl-to-pr- $\alpha$  p))
  <proof>
```

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

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

```
lemma refinement:
  fixes
    A ::  $'a$  set and
    p ::  $'a$  Profile-List
  assumes profile-l A p
  shows profile {0 ..< length p} A (pl-to-pr- $\alpha$  p)
  <proof>
```

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  $\text{ord} :: 'x \text{ rel}$  where
    linear-order-on X ord
   $\langle \text{proof} \rangle$ 

typedef  $'a \text{ Ordered-Preference} =$ 
   $\{p :: 'a :: \text{finite} \text{ Preference-Relation}. \text{linear-order-on } (\text{UNIV} :: 'a \text{ set}) \ p\}$ 
  morphisms ord2pref pref2ord
   $\langle \text{proof} \rangle$ 

instance Ordered-Preference ::  $(\text{finite}) \text{ finite}$ 
   $\langle \text{proof} \rangle$ 

lemma range-ord2pref:  $\text{range ord2pref} = \{p. \text{linear-order } p\}$ 
   $\langle \text{proof} \rangle$ 

lemma card-ord-pref:  $\text{card } (\text{UNIV} :: 'a :: \text{finite} \text{ Ordered-Preference set}) =$ 
   $\text{fact } (\text{card } (\text{UNIV} :: 'a \text{ set}))$ 
   $\langle \text{proof} \rangle$ 

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
   $\implies \text{election-equality } E \ F$ 
   $\langle \text{proof} \rangle$ 

quotient-type  $( 'a, 'v) \text{ Election}_{\mathcal{Q}} =$ 
   $'a \text{ set} \times 'v \text{ set} \times ('a, 'v) \text{ Profile} / \text{election-equality}$ 

```

<proof>

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

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

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

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

abbreviation $profile-\mathcal{E}_Q :: ('a, 'v) Election_Q \Rightarrow ('a, 'v) Profile$ **where**
 $profile-\mathcal{E}_Q\ E \equiv Product-Type.snd\ (snd_Q\ E)$

end

Chapter 3

Quotient Rules

3.1 Quotients of Equivalence Relations

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

3.1.1 Definitions

```
fun singleton-set :: 'x set  $\Rightarrow$  'x where
  singleton-set s = (if (card s = 1) then (the-inv ( $\lambda$  x. {x}) s) else undefined)
— This is undefined if 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 $\text{singleton-set } s = \text{undefined}$
 $\langle \text{proof} \rangle$

lemma *singleton-set-def-if-card-one*:
fixes $s :: 'x \text{ set}$
assumes $\text{card } s = 1$
shows $\exists! x. x = \text{singleton-set } s \wedge \{x\} = s$
 $\langle \text{proof} \rangle$

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**
 $\text{equiv } s \text{ } r$
shows $\forall t \in s // r. \forall x \in t. \pi_{\mathcal{Q}} f t = f x$
 $\langle \text{proof} \rangle$

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 $\text{equiv } s \text{ } r$
defines $\text{induced-fun} \equiv (\text{inv-}\pi_{\mathcal{Q}} (\text{relation-class } r) f)$
shows
 induced-fun *respects* r **and**
 $\forall A \in s // r. \pi_{\mathcal{Q}} \text{ induced-fun } A = f A$
 $\langle \text{proof} \rangle$

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 $\text{equiv } t \text{ } (\text{Restr } r \text{ } t)$
 $\langle \text{proof} \rangle$


```

lemma rel-ind-by-group-act-equiv:
  fixes
     $m :: 'x \text{ monoid}$  and
     $s :: 'y \text{ set}$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$ 
  assumes group-action  $m \ s \ \varphi$ 
  shows equiv  $s \ (\text{action-induced-rel } (\text{carrier } m) \ s \ \varphi)$ 
   $\langle \text{proof} \rangle$ 

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

```

3.2.2 Anonymity Quotient: Grid

```

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

```

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

```

fun vote-countQ ::  $'a \text{ Preference-Relation} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow \text{nat}$  where
  vote-countQ  $p = \pi_Q \ (\text{vote-count } p)$ 

```

fun *anonymity-class* :: ('a::finite, 'v) Election set
 \Rightarrow (nat, 'a Ordered-Preference) vec **where**
anonymity-class X = (χ p. vote-count_Q (ord2pref p) X)

lemma *anon-rel-equiv*:
equiv (elections- \mathcal{A} UNIV) (anonymity_R (elections- \mathcal{A} UNIV))
⟨proof⟩

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

theorem *anonymity_Q-isomorphism*:
assumes infinite (UNIV::('v set))
shows bij-betw (anonymity-class::('a::finite, 'v) Election set
 \Rightarrow nat[^]('a Ordered-Preference)) (anonymity_Q (UNIV::'a set))
(UNIV::(nat[^]('a Ordered-Preference)) set)
⟨proof⟩

3.2.3 Homogeneity Quotient: Simplex

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

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

fun *anonymity-homogeneity_Q* :: 'a set \Rightarrow ('a, 'v) Election set set **where**
anonymity-homogeneity_Q A =
quotient (elections- \mathcal{A} A) (anonymity-homogeneity_R (elections- \mathcal{A} A))

fun *vote-fraction_Q* :: 'a Preference-Relation \Rightarrow ('a, 'v) Election set \Rightarrow rat **where**
vote-fraction_Q p = π_Q (vote-fraction p)

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

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

definition *standard-basis* :: $(real^b \text{ set})$ **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* :: $(rat^b \text{ set})$ **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

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

assumes $\exists f :: (real^b) \Rightarrow real.$
 $\text{sum } f \ X = 1 \wedge (\forall x \in X. f \ x \geq 0)$
 $\wedge x = \text{sum } (\lambda x. (f \ x) *_{\mathbb{R}} x) \ X$

shows $x \in \text{convex hull } X$
 $\langle \text{proof} \rangle$

lemma *standard-simplex-rewrite*: $\text{convex hull standard-basis} =$
 $\{v :: (real^b). (\forall i. v\$i \geq 0) \wedge \text{sum } ((\$) \ v) \ UNIV = 1\}$
 $\langle \text{proof} \rangle$

lemma *fract-distr-helper*:
fixes

$a :: int$ **and**
 $b :: int$ **and**
 $c :: int$

assumes $c \neq 0$
shows $\text{Fract } a \ c + \text{Fract } b \ c = \text{Fract } (a + b) \ c$
 $\langle \text{proof} \rangle$

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 $\text{equiv } X \ (\text{anonymity-homogeneity}_{\mathcal{R}} \ X)$
 $\langle \text{proof} \rangle$

lemma *fract-distr*:
fixes
 $A :: 'x \text{ set}$ **and**

```

  f :: 'x ⇒ int and
  b :: int
assumes
  finite A and
  b ≠ 0
shows sum (λ a. Fract (f a) b) A = Fract (sum f A) b
⟨proof⟩

```

Simplex Bijection

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

theorem *anonymity-homogeneity_Q-isomorphism:*

```

assumes infinite (UNIV::('v set))
shows
  bij-betw (anonymity-homogeneity-class::('a::finite, 'v) Election set ⇒
    rat^('a Ordered-Preference)) (anonymity-homogeneityQ (UNIV::'a set))
    (vote-simplex :: (rat^('a Ordered-Preference)) set)
⟨proof⟩

```

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* $f\ B \subseteq B$
 $\langle \text{proof} \rangle$

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 $(\sum\ a \in A. f\ a) \leq (\sum\ a \in A. g\ a)$
 $\langle \text{proof} \rangle$

lemma *distrib*:
fixes
 $A :: 'a\ \text{set}$ **and**
 $f :: 'a \Rightarrow \text{int}$ **and**

$g :: 'a \Rightarrow \text{int}$
shows $(\sum a \in A. f\ a) + (\sum a \in A. g\ a) = (\sum a \in A. f\ a + g\ a)$
 $\langle \text{proof} \rangle$

lemma *distrib-ereal*:

fixes
 $A :: 'a\ \text{set}$ **and**
 $f :: 'a \Rightarrow \text{int}$ **and**
 $g :: 'a \Rightarrow \text{int}$
shows $\text{ereal}(\text{real-of-int}((\sum a \in A. (f::'a \Rightarrow \text{int})\ a) + (\sum a \in A. g\ a))) =$
 $\text{ereal}(\text{real-of-int}((\sum a \in A. (f\ a) + (g\ a))))$
 $\langle \text{proof} \rangle$

lemma *uneq-ereal*:

fixes
 $x :: \text{int}$ **and**
 $y :: \text{int}$
assumes $x \leq y$
shows $\text{ereal}(\text{real-of-int}\ x) \leq \text{ereal}(\text{real-of-int}\ y)$
 $\langle \text{proof} \rangle$

4.1.5 Swap Distance

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

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

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

lemma *set-eq-filter*:

fixes
 $X :: 'a\ \text{set}$ **and**
 $P :: 'a \Rightarrow \text{bool}$
shows $\{x \in X. P\ x\} = \text{Set.filter}\ P\ X$
 $\langle \text{proof} \rangle$

lemma *pairwise-disagreements-eq*[code]: $\text{pairwise-disagreements} = \text{pairwise-disagreements}'$
 $\langle \text{proof} \rangle$

fun *swap* :: $'a\ \text{Vote Distance}$ **where**
 $\text{swap}\ (A, r)\ (A', r') =$
 $(\text{if}\ A = A')$

then card (pairwise-disagreements A r r')
 else ∞)

lemma *swap-case-infinity*:

fixes

$x :: 'a$ Vote **and**

$y :: 'a$ Vote

assumes $\text{alts-}\mathcal{V} \ x \neq \text{alts-}\mathcal{V} \ y$

shows $\text{swap} \ x \ y = \infty$

$\langle \text{proof} \rangle$

lemma *swap-case-fin*:

fixes

$x :: 'a$ Vote **and**

$y :: 'a$ Vote

assumes $\text{alts-}\mathcal{V} \ x = \text{alts-}\mathcal{V} \ y$

shows $\text{swap} \ x \ y = \text{card} \ (\text{pairwise-disagreements} \ (\text{alts-}\mathcal{V} \ x) \ (\text{pref-}\mathcal{V} \ x) \ (\text{pref-}\mathcal{V} \ y))$

$\langle \text{proof} \rangle$

4.1.6 Spearman Distance

fun *spearman* :: $'a$ Vote Distance **where**

spearman $(A, x) (A', y) =$

(if $A = A'$

then $\sum a \in A. \text{abs} \ (\text{int} \ (\text{rank} \ x \ a) - \text{int} \ (\text{rank} \ y \ a))$

else ∞)

lemma *spearman-case-inf*:

fixes

$x :: 'a$ Vote **and**

$y :: 'a$ Vote

assumes $\text{alts-}\mathcal{V} \ x \neq \text{alts-}\mathcal{V} \ y$

shows *spearman* $x \ y = \infty$

$\langle \text{proof} \rangle$

lemma *spearman-case-fin*:

fixes

$x :: 'a$ Vote **and**

$y :: 'a$ Vote

assumes $\text{alts-}\mathcal{V} \ x = \text{alts-}\mathcal{V} \ y$

shows *spearman* $x \ y =$

$(\sum a \in \text{alts-}\mathcal{V} \ x. \text{abs} \ (\text{int} \ (\text{rank} \ (\text{pref-}\mathcal{V} \ x) \ a) - \text{int} \ (\text{rank} \ (\text{pref-}\mathcal{V} \ y) \ a)))$

$\langle \text{proof} \rangle$

4.1.7 Properties

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

Definitions

fun *total-invariance* _{\mathcal{D}} :: 'x Distance \Rightarrow 'x rel \Rightarrow bool **where**
total-invariance _{\mathcal{D}} d rel = *is-symmetry* (tup d) (*Invariance* (product rel))

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

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

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

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

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

definition *distance-homogeneity'* :: ('a, 'v::linorder) Election set
 \Rightarrow ('a, 'v) Election Distance \Rightarrow bool **where**
distance-homogeneity' X d = *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 = ($\forall (x, y) \in r. \forall (a, b) \in r. d a x = d b y$)
 $\langle proof \rangle$

lemma *rewrite-invariance* _{\mathcal{D}} :
fixes
d :: 'y Distance **and**
X :: 'x set **and**
Y :: 'y set **and**

```

     $\varphi :: ('x, 'y) \text{ binary-fun}$ 
shows  $\text{invariance}_{\mathcal{D}} d X Y \varphi =$ 
     $(\forall x \in X. \forall y \in Y. \forall z \in Y. d y z = d (\varphi x y) (\varphi x z))$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma invar-dist-image:
  fixes
     $d :: 'y \text{ Distance}$  and
     $G :: 'x \text{ monoid}$  and
     $Y :: 'y \text{ set}$  and
     $Y' :: 'y \text{ set}$  and
     $\varphi :: ('x, 'y) \text{ binary-fun}$  and
     $y :: 'y$  and
     $g :: 'x$ 
  assumes
    invar-d:  $\text{invariance}_{\mathcal{D}} d (\text{carrier } G) Y \varphi$  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  $d (\varphi g y) ' (\varphi g) ' Y' = d y ' Y'$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma swap-neutral:  $\text{invariance}_{\mathcal{D}} \text{swap} (\text{carrier neutrality}_{\mathcal{G}})$ 
     $UNIV (\lambda \pi (A, q). (\pi ' A, \text{rel-rename } \pi q))$ 
 $\langle \text{proof} \rangle$ 

```

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 ::  $'a \text{ Vote Distance} \Rightarrow \text{Norm}$ 
     $\Rightarrow ('a, 'v::\text{linorder}) \text{ Election Distance}$  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 (λ q q'. d (A, q) (A', q')) (to-list V p) (to-list V' p'))
 else ∞)

4.2.2 Inference Rules

lemma *symmetric-norm-inv-under-map2-permute:*

fixes
 d :: 'a Vote Distance **and**
 n :: Norm **and**
 A :: 'a set **and**
 A' :: 'a set **and**
 φ :: nat \Rightarrow nat **and**
 p :: ('a Preference-Relation) list **and**
 p' :: ('a Preference-Relation) list
assumes
 perm: φ permutes {0.. $\text{length } p$ } **and**
 len-eq: $\text{length } p = \text{length } p'$ **and**
 sym-n: symmetry n
shows n (map2 (λ q q'. d (A, q) (A', q')) p p') =
 n (map2 (λ q q'. d (A, q) (A', q')) (permute-list φ p) (permute-list φ p'))
 <proof>

lemma *permute-invariant-under-map:*

fixes
 l :: 'a list **and**
 ls :: 'a list
assumes l <~~> ls
shows map f l <~~> map f ls
 <proof>

lemma *linorder-rank-injective:*

fixes
 V :: 'v::linorder set **and**
 v :: 'v **and**
 v' :: 'v
assumes
 v-in-V: v \in V **and**
 v'-in-V: v' \in V **and**
 v'-neg-v: v' \neq v **and**
 fin-V: finite V
shows card {x \in V. x < v} \neq card {x \in V. x < v'}
 <proof>

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

fixes
 l :: 'v list **and**
 π -1 :: nat \Rightarrow nat **and**
 π -2 :: nat \Rightarrow nat

```

assumes  $\forall i < \text{length } l. \pi-1\ i = \pi-2\ i$ 
shows  $\text{permute-list } \pi-1\ l = \text{permute-list } \pi-2\ l$ 
 $\langle \text{proof} \rangle$ 

lemma symmetric-norm-imp-distance-anonymous:
fixes
   $d :: 'a\ \text{Vote Distance}$  and
   $n :: \text{Norm}$ 
assumes symmetry  $n$ 
shows distance-anonymity (votewise-distance  $d\ n$ )
 $\langle \text{proof} \rangle$ 

lemma neutral-dist-imp-neutral-votewise-dist:
fixes
   $d :: 'a\ \text{Vote Distance}$  and
   $n :: \text{Norm}$ 
defines vote-action  $\equiv (\lambda \pi\ (A, q). (\pi\ 'A, \text{rel-rename } \pi\ q))$ 
assumes invar: invarianceD  $d$  (carrier neutralityG) UNIV vote-action
shows distance-neutrality valid-elections (votewise-distance  $d\ n$ )
 $\langle \text{proof} \rangle$ 

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  $\Rightarrow \text{bool}$ 

```

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

```

$nonempty-profile_C (A, V, p) = (V \neq \{\})$

Equal top ranked alternatives.

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

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

Equal votes.

fun $equal-vote_C' :: 'a Preference-Relation \Rightarrow ('a, 'v) Consensus$ **where**
 $equal-vote_C' r (A, V, p) = (\forall v \in V. (p v) = r)$

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

Unanimity condition.

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

Strong unanimity condition.

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

4.3.3 Properties

definition $consensus-anonymity :: ('a, 'v) Consensus \Rightarrow bool$ **where**

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

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

4.3.4 Auxiliary Lemmas

lemma $cons-anon-conj$:

fixes

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

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

assumes

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

$anon2: consensus-anonymity c2$

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

$\langle proof \rangle$

theorem *cons-conjunction-invariant*:

fixes
 $\mathfrak{C} :: ('a, 'v)$ *Consensus set* **and**
 $rel :: ('a, 'v)$ *Election rel*
defines $C \equiv (\lambda E. (\forall C' \in \mathfrak{C}. C' E))$
assumes $\bigwedge C'. C' \in \mathfrak{C} \implies is-symmetry\ C' (Invariance\ rel)$
shows $is-symmetry\ C (Invariance\ rel)$
 $\langle proof \rangle$

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**
 $\pi :: 'v \Rightarrow 'v$
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 $c\ (A', V', q)$
 $\langle proof \rangle$

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 $consensus-anonymity\ b$
 $\langle proof \rangle$

4.3.5 Theorems

Anonymity

lemma *nonempty-set-cons-anonymous*: $consensus-anonymity\ nonempty-set_c$
 $\langle proof \rangle$

lemma *nonempty-profile-cons-anonymous*: $consensus-anonymity\ nonempty-profile_c$
 $\langle proof \rangle$

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

fixes $a :: 'a$

shows *consensus-anonymity* (*equal-top_C* ' *a*)
 ⟨*proof*⟩

lemma *eq-top-cons-anon: consensus-anonymity equal-top_C*
 ⟨*proof*⟩

lemma *eq-vote-cons'-anonymous:*
fixes *r* :: 'a *Preference-Relation*
shows *consensus-anonymity* (*equal-vote_C* ' *r*)
 ⟨*proof*⟩

lemma *eq-vote-cons-anonymous: consensus-anonymity equal-vote_C*
 ⟨*proof*⟩

Neutrality

lemma *nonempty-set_C-neutral: consensus-neutrality valid-elections nonempty-set_C*
 ⟨*proof*⟩

lemma *nonempty-profile_C-neutral: consensus-neutrality valid-elections nonempty-profile_C*
 ⟨*proof*⟩

lemma *equal-vote_C-neutral: consensus-neutrality valid-elections equal-vote_C*
 ⟨*proof*⟩

lemma *strong-unanimity_C-neutral:*
consensus-neutrality valid-elections strong-unanimity_C
 ⟨*proof*⟩

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 alterna-

tives, 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 $(\text{'a}, \text{'v}, \text{'r}) \text{ Electoral-Module} = \text{'v set} \Rightarrow \text{'a set} \Rightarrow (\text{'a}, \text{'v}) \text{ Profile} \Rightarrow \text{'r}$

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

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

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

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

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

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

fun $\text{voters-determine-election} :: (\text{'a}, \text{'v}, (\text{'r Result})) \text{ Electoral-Module} \Rightarrow \text{bool} \text{ where}$

voters-determine-election $m =$
 $(\forall A \ V \ p \ p'. (\forall v \in V. p \ v = p' \ v) \longrightarrow m \ V \ A \ p = m \ V \ A \ p')$

lemma (*in result*) *electoral-modI*:
fixes $m :: ('a, 'v, ('r \text{ Result})) \text{ Electoral-Module}$
assumes $\bigwedge A \ V \ p. \text{profile } V \ A \ p \implies \text{well-formed } A \ (m \ V \ A \ p)$
shows *electoral-module* m
<proof>

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

Homogeneity

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

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

— This does not require any specific behaviour on infinite voter sets ... It might make sense to extend the definition to that case somehow.

```

fun homogeneity' :: ('a, 'v::linorder) Election set  $\Rightarrow$  ('a, 'v, 'b Result) Electoral-Module
 $\Rightarrow$  bool where
  homogeneity' X m = is-symmetry (funE m) (Invariance (homogeneityR' 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>

```

Neutrality

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

```

fun (in result-properties) neutrality :: ('a, 'v) Election set
 $\Rightarrow$  ('a, 'v, 'b Result) Electoral-Module  $\Rightarrow$  bool where
  neutrality X m =
    is-symmetry (funE m) (action-induced-equivariance (carrier neutralityG) 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) Electoral-Module  $\Rightarrow$  bool where
  reversal-symmetry X m =
    is-symmetry (funE m) (action-induced-equivariance (carrier reversalG) 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 defers :: nat  $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module  $\Rightarrow$  bool where
  defers n m  $\equiv$ 
    SCF-result.electoral-module m  $\wedge$ 
    ( $\forall A \ V \ p. (\text{card } A \geq n \wedge \text{finite } A \wedge \text{profile } V \ A \ p) \longrightarrow \text{card } (\text{defer } m \ V \ A \ p) = n$ )

```

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

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

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

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

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

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

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

definition $\text{indep-of-alt} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'v \text{ set}$
 $\Rightarrow 'a \text{ set} \Rightarrow 'a \Rightarrow \text{bool}$ **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

$elect\ m\ V\ A\ p = e$ **and**
 $reject\ m\ V\ A\ p = r$ **and**
 $defer\ m\ V\ A\ p = d$
shows $m\ V\ A\ p = (e, r, d)$
 $\langle proof \rangle$

lemma *par-comp-result-sound*:
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 $well\ formed\ SCF\ A\ (m\ V\ A\ p)$
 $\langle proof \rangle$

lemma *result-presv-alts*:
fixes
 $m :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $SCF\ result.electoral\ module\ m$ **and**
 $profile\ V\ A\ p$
shows $(elect\ m\ V\ A\ p) \cup (reject\ m\ V\ A\ p) \cup (defer\ m\ V\ A\ p) = A$
 $\langle proof \rangle$

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
 $(reject\ m\ V\ A\ p) \cap (defer\ m\ V\ A\ p) = \{\}$
 $\langle proof \rangle$

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* \subseteq *A*
<proof>

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*
<proof>

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*
<proof>

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*)
<proof>

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: $\text{card } (\text{elect } m V A p) \leq \text{card } A$ **and**
upper-card-bound-for-reject: $\text{card } (\text{reject } m V A p) \leq \text{card } A$ **and**
upper-card-bound-for-defer: $\text{card } (\text{defer } m V A p) \leq \text{card } A$
 $\langle \text{proof} \rangle$

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 $\text{reject } m V A p = A - (\text{elect } m V A p) - (\text{defer } m V A p)$
 $\langle \text{proof} \rangle$

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 $\text{elect } m V A p \cup \text{defer } m V A p = A - (\text{reject } m V A p)$
 $\langle \text{proof} \rangle$

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

lemma *electoral-mod-defer-elem*:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**

$V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
 $SCF\text{-result.electoral-module } m$ **and**
 $profile \ V \ A \ p$ **and**
 $a \in A$ **and**
 $a \notin elect \ m \ V \ A \ p$ **and**
 $a \notin reject \ m \ V \ A \ p$
shows $a \in defer \ m \ V \ A \ p$
 $\langle proof \rangle$

lemma *mod-contains-result-comm:*
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 $mod\text{-contains-result } m \ n \ V \ A \ p \ a$
shows $mod\text{-contains-result } n \ m \ V \ A \ p \ a$
 $\langle proof \rangle$

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
 $SCF\text{-result.electoral-module } m$ **and**
 $profile \ V \ A \ p$ **and**
 $a \in A$ **and**
 $a \notin reject \ m \ V \ A \ p$
shows $a \in elect \ m \ V \ A \ p \vee a \in defer \ m \ V \ A \ p$
 $\langle proof \rangle$

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**
 $card \ A > 1$ **and**
 $profile \ V \ A \ p$

shows *defer* $m \ V \ A \ p \subset A$
 $\langle proof \rangle$

lemma *eq-alt-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

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 $m \ V \ A \ p = m \ V \ A \ q$

$\langle proof \rangle$

lemma *eq-def-and-elect-imp-eq*:

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**

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

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: *elect* $m \ V \ A \ p = \text{elect } n \ V \ A \ q$ **and**

def-eq: *defer* $m \ V \ A \ p = \text{defer } n \ V \ A \ q$

shows $m \ V \ A \ p = n \ V \ A \ q$

$\langle proof \rangle$

4.4.8 Non-Blocking

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

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

$\langle \text{proof} \rangle$

theorem *electing-imp-non-blocking*:

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

assumes *electing* *m*

shows *non-blocking* *m*

$\langle \text{proof} \rangle$

4.4.10 Properties

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

definition *non-electing* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow bool **where**

non-electing *m* \equiv

SCF-result.electoral-module *m*

$\wedge (\forall A V p. \text{profile } V A p \longrightarrow \text{elect } m V A p = \{\})$

lemma *single-rej-decr-def-card*:

fixes

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

A :: 'a set **and**

V :: 'v set **and**

p :: ('a, 'v) Profile

assumes

rejecting: *rejects* 1 *m* **and**

non-electing: *non-electing* *m* **and**

f-prof: *finite-profile* *V* *A* *p*

shows *card* (*defer* *m* *V* *A* *p*) = *card* *A* - 1

$\langle \text{proof} \rangle$

lemma *single-elim-decr-def-card-2*:

fixes

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

A :: 'a 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: *card A > 1* **and**
prof-p: *profile V A p*
shows *card (defer m V A p) = card A - 1*
 $\langle \text{proof} \rangle$

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

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

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

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>

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

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

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

definition *invariant-monotonicity* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow bool **where**

invariant-monotonicity m \equiv
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}

\langle proof \rangle

theorem *ccomp-and-dd-imp-dcc[simp]*:

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

ccomp: *condorcet-compatibility* m **and**
dd: *defer-deciding* m

shows *defer-condorcet-consistency* m

\langle proof \rangle

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

theorem *disj-compat-comm[simp]*:

fixes
 m :: ('a, 'v, 'a Result) Electoral-Module **and**
 n :: ('a, 'v, 'a Result) Electoral-Module

assumes *disjoint-compatibility* m n

shows *disjoint-compatibility* n m

\langle proof \rangle

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

theorem *dl-inv-imp-def-mono[simp]*:

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

assumes *defer-lift-invariance* m

shows *defer-monotonicity* m

\langle proof \rangle

4.4.12 Social Choice Properties

Condorcet Consistency

definition *condorcet-consistency* :: ('a, 'v, 'a Result) Electoral-Module
 \Rightarrow 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 Result) Electoral-Module

shows *condorcet-consistency* m =

$(\text{SCF-result.electoral-module } m \wedge$
 $(\forall A V p a. \text{condorcet-winner } V A p a \longrightarrow$
 $(m V A p = (\{a\}, A - (\text{elect } m V A p), \{\})))$

<proof>

lemma *condorcet-consistency''*:

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

shows *condorcet-consistency* m =

$(\text{SCF-result.electoral-module } m \wedge$
 $(\forall A V p a. \text{condorcet-winner } V A p a \longrightarrow m V A p = (\{a\}, A - \{a\}, \{\})))$

<proof>

(Weak) Monotonicity

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

definition *monotonicity* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow bool **where**
monotonicity m \equiv

SCF-result.electoral-module m \wedge
 $(\forall A V p q a. a \in \text{elect } m V A p \wedge \text{lifted } V A p q a \longrightarrow a \in \text{elect } m V A q)$

end

4.5 Electoral Module on Election Quotients

theory *Quotient-Module*

imports *Quotients/Relation-Quotients*

Electoral-Module

begin

lemma *invariance-is-congruence*:

fixes

```

    m :: ('a, 'v, 'r) Electoral-Module and
    r :: ('a, 'v) Election rel
shows (is-symmetry (funε m) (Invariance r)) = (funε m respects r)
⟨proof⟩

lemma invariance-is-congruence':
fixes
  f :: 'x ⇒ 'y and
  r :: 'x rel
shows (is-symmetry f (Invariance r)) = (f respects r)
⟨proof⟩

theorem pass-to-election-quotient:
fixes
  m :: ('a, 'v, 'r) Electoral-Module and
  r :: ('a, 'v) Election rel and
  X :: ('a, 'v) Election set
assumes
  equiv X r and
  is-symmetry (funε m) (Invariance r)
shows ∀ A ∈ X // r. ∀ E ∈ A. πQ (funε m) A = funε m E
⟨proof⟩

end

```

4.6 Evaluation Function

```

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

```

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

4.6.1 Definition

```

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

```

4.6.2 Property

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

```

definition condorcet-rating :: ('a, 'v) Evaluation-Function ⇒ bool where

```

condorcet-rating $f \equiv$
 $\forall A V p w . \text{condorcet-winner } V A p w \longrightarrow$
 $(\forall l \in A . l \neq w \longrightarrow f V l A p < f V w A p)$

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
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
 $\text{rating: condorcet-rating } e$ **and**
 $f\text{-prof: finite-profile } V A p$ **and**
 $\text{winner: condorcet-winner } V A p a$
shows $e V a A p = \text{Max } \{e V b A p \mid b. b \in A\}$
 $\langle \text{proof} \rangle$

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
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$ **and**
 $b :: 'a$
assumes
 $\text{rating: condorcet-rating } e$ **and**
 $f\text{-prof: finite-profile } V A p$ **and**
 $\text{winner: condorcet-winner } V A p a$ **and**
 $\text{lin-A: } b \in A$ **and**
 $\text{loser: } a \neq b$
shows $e V b A p < \text{Max } \{e V c A p \mid c. c \in A\}$
 $\langle \text{proof} \rangle$

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  $\Rightarrow$  enat  $\Rightarrow$  bool

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

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

```

4.7.2 Social Choice Definitions

```

fun elimination-module :: ('a, 'v) Evaluation-Function  $\Rightarrow$  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 - (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

```

$\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{less-eliminator } e \text{ } t \text{ } V \text{ } A \text{ } p = \text{elimination-module } e \text{ } t \text{ } (<) \text{ } V \text{ } A \text{ } p$

fun $\text{max-eliminator} :: ('a, 'v) \text{ Evaluation-Function}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{max-eliminator } e \text{ } V \text{ } A \text{ } p =$
 $\text{less-eliminator } e \text{ } (\text{Max } \{e \text{ } V \text{ } x \text{ } A \text{ } p \mid x. x \in A\}) \text{ } V \text{ } A \text{ } p$

fun $\text{leq-eliminator} :: ('a, 'v) \text{ Evaluation-Function}$
 $\Rightarrow \text{Threshold-Value}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{leq-eliminator } e \text{ } t \text{ } V \text{ } A \text{ } p = \text{elimination-module } e \text{ } t \text{ } (\leq) \text{ } V \text{ } A \text{ } p$

fun $\text{min-eliminator} :: ('a, 'v) \text{ Evaluation-Function}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{min-eliminator } e \text{ } V \text{ } A \text{ } p =$
 $\text{leq-eliminator } e \text{ } (\text{Min } \{e \text{ } V \text{ } x \text{ } A \text{ } p \mid x. x \in A\}) \text{ } V \text{ } A \text{ } p$

fun $\text{less-average-eliminator} :: ('a, 'v) \text{ Evaluation-Function}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{less-average-eliminator } e \text{ } V \text{ } A \text{ } p = \text{less-eliminator } e \text{ } (\text{average } e \text{ } V \text{ } A \text{ } p) \text{ } V \text{ } A \text{ } p$

fun $\text{leq-average-eliminator} :: ('a, 'v) \text{ Evaluation-Function}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $\text{leq-average-eliminator } e \text{ } V \text{ } A \text{ } p = \text{leq-eliminator } e \text{ } (\text{average } e \text{ } V \text{ } A \text{ } p) \text{ } V \text{ } A \text{ } p$

4.7.4 Soundness

lemma $\text{elim-mod-sound[simp]}:$
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function } \mathbf{and}$
 $t :: \text{Threshold-Value } \mathbf{and}$
 $r :: \text{Threshold-Relation}$
shows $\text{SCF-result.electoral-module } (\text{elimination-module } e \text{ } t \text{ } r)$
 $\langle \text{proof} \rangle$

lemma $\text{less-elim-sound[simp]}:$
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function } \mathbf{and}$
 $t :: \text{Threshold-Value}$
shows $\text{SCF-result.electoral-module } (\text{less-eliminator } e \text{ } t)$
 $\langle \text{proof} \rangle$

lemma $\text{leq-elim-sound[simp]}:$
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function } \mathbf{and}$
 $t :: \text{Threshold-Value}$
shows $\text{SCF-result.electoral-module } (\text{leq-eliminator } e \text{ } t)$
 $\langle \text{proof} \rangle$

lemma *max-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*max-eliminator* e)
 ⟨*proof*⟩

lemma *min-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*min-eliminator* e)
 ⟨*proof*⟩

lemma *less-avg-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*less-average-eliminator* e)
 ⟨*proof*⟩

lemma *leq-avg-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*leq-average-eliminator* e)
 ⟨*proof*⟩

4.7.5 Only participating voters impact the result

lemma *voters-determine-elim-mod*[simp]:
fixes
 $e :: ('a, 'v)$ *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*⟩

lemma *voters-determine-less-elim*[simp]:
fixes
 $e :: ('a, 'v)$ *Evaluation-Function* **and**
 $t :: \text{Threshold-Value}$
assumes *voters-determine-evaluation* e
shows *voters-determine-election* (*less-eliminator* e t)
 ⟨*proof*⟩

lemma *voters-determine-leq-elim*[simp]:
fixes
 $e :: ('a, 'v)$ *Evaluation-Function* **and**
 $t :: \text{Threshold-Value}$
assumes *voters-determine-evaluation* e
shows *voters-determine-election* (*leq-eliminator* e t)
 ⟨*proof*⟩

lemma *voters-determine-max-elim*[simp]:

fixes $e :: ('a, 'v) \text{ Evaluation-Function}$
assumes $\text{voters-determine-evaluation } e$
shows $\text{voters-determine-election } (\text{max-eliminator } e)$
 $\langle \text{proof} \rangle$

lemma $\text{voters-determine-min-elim}[\text{simp}]$:
fixes $e :: ('a, 'v) \text{ Evaluation-Function}$
assumes $\text{voters-determine-evaluation } e$
shows $\text{voters-determine-election } (\text{min-eliminator } e)$
 $\langle \text{proof} \rangle$

lemma $\text{voters-determine-less-avg-elim}[\text{simp}]$:
fixes $e :: ('a, 'v) \text{ Evaluation-Function}$
assumes $\text{voters-determine-evaluation } e$
shows $\text{voters-determine-election } (\text{less-average-eliminator } e)$
 $\langle \text{proof} \rangle$

lemma $\text{voters-determine-leq-avg-elim}[\text{simp}]$:
fixes $e :: ('a, 'v) \text{ Evaluation-Function}$
assumes $\text{voters-determine-evaluation } e$
shows $\text{voters-determine-election } (\text{leq-average-eliminator } e)$
 $\langle \text{proof} \rangle$

4.7.6 Non-Blocking

lemma $\text{elim-mod-non-blocking}$:
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $t :: \text{Threshold-Value}$ **and**
 $r :: \text{Threshold-Relation}$
shows $\text{non-blocking } (\text{elimination-module } e \ t \ r)$
 $\langle \text{proof} \rangle$

lemma $\text{less-elim-non-blocking}$:
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $t :: \text{Threshold-Value}$
shows $\text{non-blocking } (\text{less-eliminator } e \ t)$
 $\langle \text{proof} \rangle$

lemma $\text{leq-elim-non-blocking}$:
fixes
 $e :: ('a, 'v) \text{ Evaluation-Function}$ **and**
 $t :: \text{Threshold-Value}$
shows $\text{non-blocking } (\text{leq-eliminator } e \ t)$
 $\langle \text{proof} \rangle$

lemma $\text{max-elim-non-blocking}$:
fixes $e :: ('a, 'v) \text{ Evaluation-Function}$

shows *non-blocking* (*max-eliminator e*)
 ⟨*proof*⟩

lemma *min-elim-non-blocking*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-blocking* (*min-eliminator e*)
 ⟨*proof*⟩

lemma *less-avg-elim-non-blocking*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-blocking* (*less-average-eliminator e*)
 ⟨*proof*⟩

lemma *leq-avg-elim-non-blocking*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-blocking* (*leq-average-eliminator e*)
 ⟨*proof*⟩

4.7.7 Non-Electing

lemma *elim-mod-non-electing*:
fixes
 e :: ('a, 'v) *Evaluation-Function* **and**
 t :: *Threshold-Value* **and**
 r :: *Threshold-Relation*
shows *non-electing* (*elimination-module e t r*)
 ⟨*proof*⟩

lemma *less-elim-non-electing*:
fixes
 e :: ('a, 'v) *Evaluation-Function* **and**
 t :: *Threshold-Value*
shows *non-electing* (*less-eliminator e t*)
 ⟨*proof*⟩

lemma *leq-elim-non-electing*:
fixes
 e :: ('a, 'v) *Evaluation-Function* **and**
 t :: *Threshold-Value*
shows *non-electing* (*leq-eliminator e t*)
 ⟨*proof*⟩

lemma *max-elim-non-electing*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-electing* (*max-eliminator e*)
 ⟨*proof*⟩

lemma *min-elim-non-electing*:
fixes *e* :: ('a, 'v) *Evaluation-Function*

shows *non-electing* (*min-eliminator* *e*)
 ⟨*proof*⟩

lemma *less-avg-elim-non-electing*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-electing* (*less-average-eliminator* *e*)
 ⟨*proof*⟩

lemma *leq-avg-elim-non-electing*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
shows *non-electing* (*leq-average-eliminator* *e*)
 ⟨*proof*⟩

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*⟩

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*⟩

end

4.8 Aggregator

theory *Aggregator*
imports *Social-Choice-Types/Social-Choice-Result*
begin

An aggregator gets two partitions (results of electoral modules) as input and output another partition. They are used to aggregate results of parallel composed electoral modules. They are commutative, i.e., the order of the aggregated modules does not affect the resulting aggregation. Moreover,

they are conservative in the sense that the resulting decisions are subsets of the two given partitions' decisions.

4.8.1 Definition

type-synonym *'a* Aggregator = *'a* set \Rightarrow *'a* Result \Rightarrow *'a* Result \Rightarrow *'a* Result

definition *aggregator* :: *'a* Aggregator \Rightarrow bool **where**

aggregator agg \equiv
 $\forall A e e' d d' r r'.$
 $(\text{well-formed-SCF } A (e, r, d) \wedge \text{well-formed-SCF } A (e', r', d')) \longrightarrow$
 $\text{well-formed-SCF } A (\text{agg } A (e, r, d) (e', r', d'))$

4.8.2 Properties

definition *agg-commutative* :: *'a* Aggregator \Rightarrow bool **where**

agg-commutative agg \equiv
 $\text{aggregator agg} \wedge (\forall A e e' d d' r r'.$
 $\text{agg } A (e, r, d) (e', r', d') = \text{agg } A (e', r', d') (e, r, d))$

definition *agg-conservative* :: *'a* Aggregator \Rightarrow bool **where**

agg-conservative agg \equiv
 $\text{aggregator agg} \wedge$
 $(\forall A e e' d d' r r'.$
 $((\text{well-formed-SCF } A (e, r, d) \wedge \text{well-formed-SCF } A (e', r', d')) \longrightarrow$
 $\text{elect-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (e \cup e') \wedge$
 $\text{reject-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (r \cup r') \wedge$
 $\text{defer-r } (\text{agg } A (e, r, d) (e', r', d')) \subseteq (d \cup d'))$

end

4.9 Maximum Aggregator

theory *Maximum-Aggregator*

imports *Aggregator*

begin

The max(imum) aggregator takes two partitions of an alternative set A as input. It returns a partition where every alternative receives the maximum result of the two input partitions.

4.9.1 Definition

fun *max-aggregator* :: *'a* Aggregator **where**

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

4.9.2 Auxiliary Lemma

lemma *max-agg-rej-set*:

fixes

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

$a :: 'a$

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* $A (e, r, d) (e', r', d')$) = $r \cap r'$
 $\langle \text{proof} \rangle$

4.9.3 Soundness

theorem *max-agg-sound[simp]*: *aggregator max-aggregator*
 $\langle \text{proof} \rangle$

4.9.4 Properties

The max-aggregator is conservative.

theorem *max-agg-consv[simp]*: *agg-conservative max-aggregator*
 $\langle \text{proof} \rangle$

The max-aggregator is commutative.

theorem *max-agg-comm[simp]*: *agg-commutative max-aggregator*
 $\langle \text{proof} \rangle$

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  
  <proof>
```

5.1.3 Properties

```
theorem def-mod-non-electing: non-electing defer-module  
  <proof>
```

```
theorem def-mod-def-lift-inv: defer-lift-invariance defer-module  
  <proof>
```

```
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>

```

```

end

```

5.3 Consensus Class

```

theory Consensus-Class
  imports Consensus
           ../Defer-Module
           ../Elect-First-Module
begin

```

A consensus class is a pair of a set of elections and a mapping that assigns a unique alternative to each election in that set (of elections). This alternative is then called the consensus alternative (winner). Here, we model the mapping by an electoral module that defers alternatives which are not in the consensus.

5.3.1 Definition

```

type-synonym ('a, 'v, 'r) Consensus-Class = ('a, 'v) Consensus  $\times$  ('a, 'v, 'r)
  Electoral-Module

fun consensus-K :: ('a, 'v, 'r) Consensus-Class  $\Rightarrow$  ('a, 'v) Consensus
  where consensus-K K = fst K

```

fun $\text{rule-}\mathcal{K} :: ('a, 'v, 'r) \text{ Consensus-Class} \Rightarrow ('a, 'v, 'r) \text{ Electoral-Module}$
where $\text{rule-}\mathcal{K} K = \text{snd } K$

5.3.2 Consensus Choice

Returns those consensus elections on a given alternative and voter set from a given consensus that are mapped to the given unique winner by a given consensus rule.

fun $\mathcal{K}_{\mathcal{E}} :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set}$ **where**
 $\mathcal{K}_{\mathcal{E}} K w =$
 $\{(A, V, p) \mid A \ V \ p. (\text{consensus-}\mathcal{K} K) (A, V, p) \wedge \text{finite-profile } V \ A \ p$
 $\wedge \text{elect } (\text{rule-}\mathcal{K} K) \ V \ A \ p = \{w\}\}$

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

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 $\text{well-formed} :: ('a, 'v) \text{ Consensus} \Rightarrow ('a, 'v, 'r) \text{ Electoral-Module}$
 $\Rightarrow \text{bool}$ **where**

$\text{well-formed } c \ m \equiv$
 $\forall \ A \ V \ V' \ p \ p'.$
 $\text{profile } V \ A \ p \wedge \text{profile } V' \ A \ p' \wedge c \ (A, V, p) \wedge c \ (A, V', p')$
 $\longrightarrow m \ V \ A \ p = m \ V' \ A \ p'$

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 $\text{consensus-choice} :: ('a, 'v) \text{ Consensus} \Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **where**
 $\text{consensus-choice } c \ m =$
 $(\text{let}$
 $w = (\lambda \ V \ A \ p. \text{if } c \ (A, V, p) \text{ then } m \ V \ A \ p \text{ else defer-module } V \ A \ p)$
 $\text{in } (c, w))$

5.3.3 Auxiliary Lemmas

lemma $\text{unanimity'-consensus-imp-elect-fst-mod-well-formed}:$

fixes $a :: 'a$

shows well-formed

$(\lambda \ c. \text{nonempty-set}_C \ c \wedge \text{nonempty-profile}_C \ c$
 $\wedge \text{equal-top}_C \ 'a \ c) \text{ elect-first-module}$

$\langle \text{proof} \rangle$

lemma $\text{strong-unanimity'-consensus-imp-elect-fst-mod-completely-determined}:$

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

shows well-formed

$(\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c \wedge \text{equal-vote}_C 'r c) \text{elect-first-module}$
 $\langle \text{proof} \rangle$

lemma *strong-unanimity'consensus-imp-elect-fst-mod-well-formed:*

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

shows *well-formed*

$(\lambda c. \text{nonempty-set}_C c \wedge \text{nonempty-profile}_C c$
 $\wedge \text{equal-vote}_C 'r c) \text{elect-first-module}$

$\langle \text{proof} \rangle$

lemma *cons-domain-valid:*

fixes $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$

shows *elections- \mathcal{K} $C \subseteq \text{valid-elections}$*

$\langle \text{proof} \rangle$

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} $C \subseteq \text{finite-elections-}\mathcal{V}$

$\langle \text{proof} \rangle$

5.3.4 Consensus Rules

definition *non-empty-set* $:: ('a, 'v, 'r) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**

non-empty-set $c \equiv \exists K. \text{consensus-}\mathcal{K} c K$

Unanimity condition.

definition *unanimity* $:: ('a, 'v::\text{wellorder}, 'a \text{ Result}) \text{ Consensus-Class}$ **where**

unanimity $= \text{consensus-choice unanimity}_C \text{elect-first-module}$

Strong unanimity condition.

definition *strong-unanimity* $:: ('a, 'v::\text{wellorder}, 'a \text{ Result}) \text{ Consensus-Class}$ **where**

strong-unanimity $= \text{consensus-choice strong-unanimity}_C \text{elect-first-module}$

5.3.5 Properties

definition *consensus-rule-anonymity* $:: ('a, 'v, 'r) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**

consensus-rule-anonymity $c \equiv$

$(\forall A V p \pi::('v \Rightarrow 'v)).$

$\text{bij } \pi \longrightarrow$

$(\text{let } (A', V', q) = (\text{rename } \pi (A, V, p)) \text{ in}$

$\text{profile } V A p \longrightarrow \text{profile } V' A' q$

$\longrightarrow \text{consensus-}\mathcal{K} c (A, V, p)$

$\longrightarrow (\text{consensus-}\mathcal{K} c (A', V', q) \wedge (\text{rule-}\mathcal{K} c V A p = \text{rule-}\mathcal{K} c V' A' q))))$

fun *consensus-rule-anonymity'* $:: ('a, 'v) \text{ Election set}$

$\Rightarrow ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**

consensus-rule-anonymity' $X C =$

$is-symmetry (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C)) (Invariance (anonymity_{\mathcal{R}} X))$
fun (**in** *result-properties*) *consensus-rule-neutrality* :: ('a, 'v) *Election set*
 $\Rightarrow ('a, 'v, 'b \text{ Result}) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**
consensus-rule-neutrality X C =
 $is-symmetry (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C))$
 $(action-induced-equivariance$
 $(carrier\ neutrality_G) X (\varphi-neutr\ X) (set-action\ \psi-neutr))$
fun *consensus-rule-reversal-symmetry* :: ('a, 'v) *Election set*
 $\Rightarrow ('a, 'v, 'a \text{ rel Result}) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**
consensus-rule-reversal-symmetry X C = $is-symmetry (elect-r \circ fun_{\mathcal{E}} (rule-\mathcal{K} C))$
 $(action-induced-equivariance (carrier\ reversal_G) X (\varphi-rev\ X) (set-action\ \psi-rev))$

5.3.6 Inference Rules

lemma *consensus-choice-equivar*:

fixes

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

$c :: ('a, 'v) \text{ Consensus}$ **and**

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

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

$\varphi :: ('x, ('a, 'v) \text{ Election}) \text{ binary-fun}$ **and**

$\psi :: ('x, 'a) \text{ binary-fun}$ **and**

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

defines *equivar* $\equiv action-induced-equivariance\ G\ X\ \varphi\ (set-action\ \psi)$

assumes

equivar-m: $is-symmetry (f \circ fun_{\mathcal{E}} m)$ *equivar* **and**

equivar-defer: $is-symmetry (f \circ fun_{\mathcal{E}} defer-module)$ *equivar* **and**

— This could be generalized to arbitrary modules instead of *defer-module*.

invar-cons: $is-symmetry\ c\ (Invariance\ (action-induced-rel\ G\ X\ \varphi))$

shows $is-symmetry (f \circ fun_{\mathcal{E}} (rule-\mathcal{K} (consensus-choice\ c\ m)))$

$(action-induced-equivariance\ G\ X\ \varphi\ (set-action\ \psi))$

<proof>

lemma *consensus-choice-anonymous*:

fixes

$\alpha :: ('a, 'v) \text{ Consensus}$ **and**

$\beta :: ('a, 'v) \text{ Consensus}$ **and**

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

$\beta' :: 'b \Rightarrow ('a, 'v) \text{ Consensus}$

assumes

beta-sat: $\beta = (\lambda E. \exists a. \beta' a\ E)$ **and**

beta'-anon: $\forall x. consensus-anonymity (\beta' x)$ **and**

anon-cons-cond: $consensus-anonymity\ \alpha$ **and**

conditions-univ: $\forall x. well-formed (\lambda E. \alpha\ E \wedge \beta' x\ E)\ m$

shows *consensus-rule-anonymity* $(consensus-choice (\lambda E. \alpha\ E \wedge \beta\ E)\ m)$

<proof>

5.3.7 Theorems

Anonymity

lemma *unanimity-anonymous: consensus-rule-anonymity unanimity*
 ⟨proof⟩

lemma *strong-unanimity-anonymous: consensus-rule-anonymity strong-unanimity*
 ⟨proof⟩

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 *is-symmetry (elect-r \circ fun_E defer-module)*
(action-induced-equivariance G X φ (set-action ψ))

⟨proof⟩

lemma *elect-first-winners-neutral: is-symmetry (elect-r \circ fun_E elect-first-module)*
(action-induced-equivariance (carrier neutrality_G)
valid-elections (φ -neutr valid-elections) (set-action ψ -neutr_c))

⟨proof⟩

lemma *strong-unanimity-neutral:*

defines *domain \equiv valid-elections \cap Collect strong-unanimity_C*

— We want to show neutrality on a set as general as possible, as this implies subset neutrality.

shows *SCF-properties.consensus-rule-neutrality domain strong-unanimity*

⟨proof⟩

lemma *strong-unanimity-neutral': SCF-properties.consensus-rule-neutrality*
(elections- \mathcal{K} strong-unanimity) strong-unanimity

⟨proof⟩

lemma *strong-unanimity-closed-under-neutrality: closed-restricted-rel*

(neutrality_R valid-elections) valid-elections (elections- \mathcal{K} strong-unanimity)

⟨proof⟩

end

5.4 Distance Rationalization

theory *Distance-Rationalization*

imports *Social-Choice-Types/Refined-Types/Preference-List*

Consensus-Class
Distance

begin

A distance rationalization of a voting rule is its interpretation as a procedure that elects an uncontroversial winner if there is one, and otherwise elects the alternatives that are as close to becoming an uncontroversial winner as possible. Within general distance rationalization, a voting rule is characterized by a distance on profiles and a consensus class.

5.4.1 Definitions

Returns the distance of an election to the preimage of a unique winner under the given consensus elections and consensus rule.

fun *score* :: ('a, 'v) Election Distance \Rightarrow ('a, 'v, 'r Result) Consensus-Class
 \Rightarrow ('a, 'v) Election \Rightarrow 'r \Rightarrow ereal **where**
score *d* *K* *E* *w* = Inf (*d* *E* ' ($\mathcal{K}_{\mathcal{E}}$ *K* *w*))

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

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

5.4.2 Standard Definitions

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

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

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

fun *all-profiles* :: 'v set \Rightarrow 'a set \Rightarrow (('a, 'v) Profile) set **where**
all-profiles *V* *A* =
(if (infinite *A* \vee infinite *V*)
then {} else {*p*. *p* ' *V* \subseteq (pl- α ' permutations-of-set *A*)})


```

fun  $\mathcal{K}_{\mathcal{E}}\text{-std} :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow 'r \Rightarrow 'a \text{ set} \Rightarrow 'v \text{ set}$ 
   $\Rightarrow ('a, 'v) \text{ Election set}$  where
   $\mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ A \ V =$ 
     $(\lambda p. (A, V, p))$ 
     $\text{'(Set.filter$ 
       $(\lambda p. (\text{consensus-}\mathcal{K} \ K) (A, V, p) \wedge \text{elect} (\text{rule-}\mathcal{K} \ K) \ V \ A \ p = \{w\})$ 
       $(\text{all-profiles } V \ A))$ 

```

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  $\text{score-std} :: ('a, 'v) \text{ Election Distance} \Rightarrow ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
   $\Rightarrow ('a, 'v) \text{ Election} \Rightarrow 'r \Rightarrow \text{ereal}$  where
   $\text{score-std } d \ K \ E \ w =$ 
     $(\text{if } \mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{voters-}\mathcal{E} \ E) = \{\}$ 
       $\text{then } \infty \text{ else Min } (d \ E \text{' } (\mathcal{K}_{\mathcal{E}}\text{-std } K \ w \ (\text{alternatives-}\mathcal{E} \ E) \ (\text{voters-}\mathcal{E} \ E))))$ 

```

```

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

```

```

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

```

5.4.3 Auxiliary Lemmas

```

lemma  $\text{fin-}\mathcal{K}_{\mathcal{E}}:$ 
  fixes  $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
  shows  $\text{elections-}\mathcal{K} \ C \subseteq \text{finite-elections}$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma  $\text{univ-}\mathcal{K}_{\mathcal{E}}:$ 
  fixes  $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ 
  shows  $\text{elections-}\mathcal{K} \ C \subseteq \text{UNIV}$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma  $\text{list-cons-presv-finiteness}:$ 
  fixes
     $A :: 'a \text{ set}$  and
     $S :: 'a \text{ list set}$ 
  assumes
     $\text{fin-A: finite } A$  and
     $\text{fin-B: finite } S$ 
  shows  $\text{finite } \{a \# l \mid a \ l. a \in A \wedge l \in S\}$ 

```

$\langle proof \rangle$

lemma *listset-finiteness*:

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

assumes $\forall i :: \text{nat}. i < \text{length } l \longrightarrow \text{finite } (l!i)$

shows $\text{finite } (\text{listset } l)$

$\langle proof \rangle$

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 $\text{listset } l = \{\}$

$\langle proof \rangle$

lemma *all-ls-elems-same-len*:

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

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

$\langle proof \rangle$

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

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

shows $\forall l' i :: \text{nat}. l' \in \text{listset } l \wedge i < \text{length } l' \longrightarrow l!i \in l!i$

$\langle proof \rangle$

lemma *fin-all-profs*:

fixes

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

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

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

assumes

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

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

shows $\text{finite } (\text{all-profiles } V A \cap \{p. \forall v. v \notin V \longrightarrow p v = x\})$

$\langle proof \rangle$

lemma *profile-permutation-set*:

fixes

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

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

shows $\text{all-profiles } V A =$

$\{p' :: ('a, 'v) \text{ Profile}. \text{finite-profile } V A p'\}$

$\langle proof \rangle$

5.4.4 Soundness

lemma (in *result*) \mathcal{R} -*sound*:

fixes

$K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $d :: ('a, 'v) \text{ Election Distance}$
shows *electoral-module* (*distance- \mathcal{R}* d K)
 $\langle \text{proof} \rangle$

5.4.5 Inference Rules

lemma *is-arg-min-equal*:
fixes
 $f :: 'a \Rightarrow 'b::ord$ **and**
 $g :: 'a \Rightarrow 'b$ **and**
 $S :: 'a \text{ set}$ **and**
 $x :: 'a$
assumes $\forall x \in S. f\ x = g\ x$
shows *is-arg-min* $f\ (\lambda s. s \in S)\ x = \text{is-arg-min}\ g\ (\lambda s. s \in S)\ x$
 $\langle \text{proof} \rangle$

lemma (*in result*) *standard-distance-imp-equal-score*:
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $w :: 'r$
assumes
 $\text{irr-non-}V$: *voters-determine-distance* d **and**
 std : *standard* d
shows *score* $d\ K\ (A, V, p)\ w = \text{score-std}\ d\ K\ (A, V, p)\ w$
 $\langle \text{proof} \rangle$

lemma (*in result*) *anonymous-distance-and-consensus-imp-rule-anonymity*:
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
assumes
 $d\text{-anon}$: *distance-anonymity* d **and**
 $K\text{-anon}$: *consensus-rule-anonymity* K
shows *anonymity* (*distance- \mathcal{R}* d K)
 $\langle \text{proof} \rangle$

end

5.5 Votewise Distance Rationalization

theory *Votewise-Distance-Rationalization*
imports *Distance-Rationalization*

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>

lemma *swap-standard*: *standard* (votewise-distance swap l-one)
 <proof>

5.5.3 Equivalence Lemmas

type-synonym ('a, 'v) *score-type* = ('a, 'v) Election Distance
 \Rightarrow ('a, 'v, 'a Result) Consensus-Class
 \Rightarrow ('a, 'v) Election \Rightarrow 'a \Rightarrow ereal

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

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

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

lemma *equal-score-swap*: (score::('a, 'v::linorder) score-type))
 (votewise-distance swap l-one) =
 score-std (votewise-distance swap l-one)
 <proof>

```

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>

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 = Inf (d a ' 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 domainf d x y =
    distance-infimum d (preimg f domainf y) x

fun minimizer :: ('x  $\Rightarrow$  'y)  $\Rightarrow$  'x set  $\Rightarrow$  'x Distance  $\Rightarrow$  'y set  $\Rightarrow$  'x  $\Rightarrow$  'y set where
  minimizer f domainf d Y x =
    arg-min-set (closest-preimg-distance f domainf d x) Y

```

Auxiliary Lemmas

```

lemma rewrite-arg-min-set:
  fixes
    f :: 'x  $\Rightarrow$  'y::linorder and
    X :: 'x set
  shows arg-min-set f X =  $\bigcup$  (preimg f X ' {y  $\in$  (f ' X).  $\forall$  z  $\in$  f ' X. y  $\leq$  z})
<proof>

```

Equivariance

```

lemma restr-induced-rel:
  fixes
    X :: 'x set and
    Y :: 'y set and
    Y' :: 'y set and
     $\varphi$  :: ('x, 'y) binary-fun
  assumes Y'  $\subseteq$  Y
  shows Restr (action-induced-rel X Y  $\varphi$ ) Y' = action-induced-rel X Y'  $\varphi$ 
<proof>

```

theorem *group-action-invar-dist-and-equivar-f-imp-equivar-minimizer:*
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $domain_f :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$ **and**
 $valid_img :: 'x \Rightarrow 'y \text{ set}$ **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 *equivar-prop-set-valued* \equiv
action-induced-equivariance (*carrier* G) X φ (*set-action* ψ)
assumes
action- φ : *group-action* G X φ **and**
group-action-res: *group-action* G *UNIV* ψ **and**
dom-in- X : $domain_f \subseteq X$ **and**
closed-domain:
closed-restricted-rel (*action-induced-rel* (*carrier* G) X φ) X $domain_f$ **and**
equivar-img: *is-symmetry* $valid_img$ *equivar-prop-set-valued* **and**
invar-d: *invariance_D* d (*carrier* G) X φ **and**
equivar-f:
is-symmetry f (*action-induced-equivariance* (*carrier* G) $domain_f$ φ ψ)
shows *is-symmetry* $(\lambda x. \text{minimizer } f \text{ domain}_f d (valid_img \ x) \ x)$ *equivar-prop-set-valued*
 $\langle proof \rangle$

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist:*
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $domain_f :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$ **and**
 $rel :: 'x \text{ rel}$
assumes
r-refl: *refl-on* $domain_f$ (*Restr* rel $domain_f$) **and**
tot-invar-d: *total-invariance_D* d rel
shows *is-symmetry* (*closest-preimg-distance* f $domain_f$ d) (*Invariance* rel)
 $\langle proof \rangle$

lemma *refl-rel-and-tot-invar-dist-imp-invar-minimizer:*

fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $domain_f :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$ **and**
 $rel :: 'x \text{ rel}$ **and**
 $img :: 'y \text{ set}$
assumes
r-refl: *refl-on* $domain_f$ (*Restr* rel $domain_f$) **and**

tot-invar-d: total-invariance_D d rel
shows *is-symmetry (minimizer f domain_f d img) (Invariance rel)*
 <proof>

theorem *group-act-invar-dist-and-invar-f-imp-invar-minimizer:*

fixes

f :: 'x ⇒ 'y and
domain_f :: 'x set and
d :: 'x Distance and
img :: 'y set and
X :: 'x set and
G :: 'z monoid and
φ :: ('z, 'x) binary-fun

defines

rel ≡ action-induced-rel (carrier G) X φ and
rel' ≡ action-induced-rel (carrier G) domain_f φ

assumes

action-φ: group-action G X φ and
domain_f ⊆ X and
closed-domain: closed-restricted-rel rel X domain_f and

invar-d: invariance_D d (carrier G) X φ and
invar-f: is-symmetry f (Invariance rel')

shows *is-symmetry (minimizer f domain_f d img) (Invariance rel)*
 <proof>

5.6.2 Distance Rationalization as Minimizer

lemma *K_E-is-preimg:*

fixes

d :: ('a, 'v) Election Distance and
C :: ('a, 'v, 'r Result) Consensus-Class and
E :: ('a, 'v) Election and
w :: 'r

shows *preimg (elect-r ∘ fun_E (rule-K C)) (elections-K C) {w} = K_E C w*
 <proof>

lemma *score-is-closest-preimg-dist:*

fixes

d :: ('a, 'v) Election Distance and
C :: ('a, 'v, 'r Result) Consensus-Class and
E :: ('a, 'v) Election and
w :: 'r

shows *score d C E w =*
closest-preimg-distance (elect-r ∘ fun_E (rule-K C)) (elections-K C) d E {w}
 <proof>

lemma *(in result) R_N-is-minimizer:*

fixes

$d :: ('a, 'v)$ Election Distance **and**
 $C :: ('a, 'v, 'r)$ Result Consensus-Class
shows $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} d C) =$
 $(\lambda E. \bigcup (\text{minimizer } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{elections-}\mathcal{K} C) d$
 $\quad (\text{singleton-set-system } (\text{limit-set } (\text{alternatives-}\mathcal{E} E) \text{ UNIV})) E))$
 $\langle \text{proof} \rangle$

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**
 $\text{rel} :: ('a, 'v)$ Election rel
assumes
 $r\text{-reft}$: $\text{reft-on } (\text{elections-}\mathcal{K} C) (\text{Restr rel } (\text{elections-}\mathcal{K} C))$ **and**
 tot-invar-d : $\text{total-invariance}_{\mathcal{D}} d \text{ rel}$ **and**
 invar-res :
 $\text{is-symmetry } (\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} E) \text{ UNIV})$
 (Invariance rel)
shows $\text{is-symmetry } (\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} d C)) (\text{Invariance rel})$
 $\langle \text{proof} \rangle$

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

$\text{rel} \equiv \text{action-induced-rel } (\text{carrier } G) B \varphi$ **and**
 $\text{rel}' \equiv \text{action-induced-rel } (\text{carrier } G) (\text{elections-}\mathcal{K} C) \varphi$

assumes

$\text{action-}\varphi$: group-action $G B \varphi$ **and**
 $\text{consensus-}C\text{-in-}B$: $\text{elections-}\mathcal{K} C \subseteq B$ **and**
 closed-domain :
 $\text{closed-restricted-rel rel } B (\text{elections-}\mathcal{K} C)$ **and**
 invar-res :
 $\text{is-symmetry } (\lambda E. \text{limit-set } (\text{alternatives-}\mathcal{E} E) \text{ UNIV}) (\text{Invariance rel})$ **and**
 invar-d : $\text{invariance}_{\mathcal{D}} d (\text{carrier } G) B \varphi$ **and**
 $\text{invar-}C\text{-winners}$: $\text{is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} C)) (\text{Invariance rel})$

shows $\text{is-symmetry } (\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} d C)) (\text{Invariance rel})$

$\langle \text{proof} \rangle$

Equivariance

theorem (**in result**) *invar-dist-equivar-cons-imp-equivar-dr-rule*:
fixes
 $d :: ('a, 'v)$ Election Distance **and**

$C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class and}$
 $G :: 'x \text{ monoid and}$
 $\varphi :: ('x, ('a, 'v) \text{ Election}) \text{ binary-fun and}$
 $\psi :: ('x, 'r) \text{ binary-fun and}$
 $B :: ('a, 'v) \text{ Election set}$
defines
 $\text{rel} \equiv \text{action-induced-rel (carrier } G) B \varphi \text{ and}$
 $\text{rel}' \equiv \text{action-induced-rel (carrier } G) (\text{elections-}\mathcal{K} \ C) \varphi \text{ and}$
 $\text{equivar-prop} \equiv$
 $\quad \text{action-induced-equivariance (carrier } G) (\text{elections-}\mathcal{K} \ C)$
 $\quad \varphi (\text{set-action } \psi) \text{ and}$
 $\text{equivar-prop-global-set-valued} \equiv$
 $\quad \text{action-induced-equivariance (carrier } G) B \varphi (\text{set-action } \psi) \text{ and}$
 $\text{equivar-prop-global-result-valued} \equiv$
 $\quad \text{action-induced-equivariance (carrier } G) B \varphi (\text{result-action } \psi)$
assumes
 $\text{action-}\varphi$: $\text{group-action } G B \varphi \text{ and}$
 group-act-res : $\text{group-action } G \text{ UNIV } \psi \text{ and}$
 cons-elect-set : $\text{elections-}\mathcal{K} \ C \subseteq B \text{ and}$
 closed-domain : $\text{closed-restricted-rel rel } B (\text{elections-}\mathcal{K} \ C) \text{ and}$
 equivar-res :
 $\quad \text{is-symmetry } (\lambda E. \text{limit-set (alternatives-}\mathcal{E} \ E) \text{ UNIV})$
 $\quad \text{equivar-prop-global-set-valued and}$
 invar-d : $\text{invariance}_{\mathcal{D}} d (\text{carrier } G) B \varphi \text{ and}$
 equivar-C-winners : $\text{is-symmetry (elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C)) \text{equivar-prop}$
shows $\text{is-symmetry (fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C)) \text{equivar-prop-global-result-valued}$
 $\langle \text{proof} \rangle$

5.6.3 Symmetry Property Inference Rules

theorem (*in result*) *anon-dist-and-cons-imp-anon-dr*:
fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
assumes
 anon-d : $\text{distance-anonymity}' \text{ valid-elections } d \text{ and}$
 anon-C : $\text{consensus-rule-anonymity}' (\text{elections-}\mathcal{K} \ C) C \text{ and}$
 closed-C : $\text{closed-restricted-rel (anonymity}_{\mathcal{R}} \text{ valid-elections)}$
 $\quad \text{valid-elections (elections-}\mathcal{K} \ C)$
shows $\text{anonymity}' \text{ valid-elections (distance-}\mathcal{R} \ d \ C)$
 $\langle \text{proof} \rangle$

theorem (*in result-properties*) *neutr-dist-and-cons-imp-neutr-dr*:
fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'b \text{ Result}) \text{ Consensus-Class}$
assumes
 neutr-d : $\text{distance-neutrality valid-elections } d \text{ and}$
 neutr-C : $\text{consensus-rule-neutrality (elections-}\mathcal{K} \ C) C \text{ and}$

closed-C: closed-restricted-rel (neutrality_R valid-elections)
valid-elections (elections-K C)
shows *neutrality valid-elections (distance-R d C)*
 <proof>

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-K C) C and
closed-C: closed-restricted-rel (reversal_R valid-elections)
valid-elections (elections-K C)
shows *reversal-symmetry valid-elections (SWF-result.distance-R d C)*
 <proof>

theorem *(in result) tot-hom-dist-imp-hom-dr:*
fixes
d :: ('a, nat) Election Distance and
C :: ('a, nat, 'r Result) Consensus-Class
assumes *distance-homogeneity finite-elections-V d*
shows *homogeneity finite-elections-V (distance-R d C)*
 <proof>

theorem *(in result) tot-hom-dist-imp-hom-dr':*
fixes
d :: ('a, 'v::linorder) Election Distance and
C :: ('a, 'v, 'r Result) Consensus-Class
assumes *distance-homogeneity' finite-elections-V d*
shows *homogeneity' finite-elections-V (distance-R d C)*
 <proof>

5.6.4 Further Properties

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

5.7 Distance Rationalization on Election Quotients

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

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 ∞ else
 π_Q (tup d) (A \times B)))

fun *relation-paths* :: 'x rel \Rightarrow 'x list set **where**
relation-paths r =
 $\{p. \exists k. (\text{length } p = 2 * k \wedge (\forall i < k. (p!(2 * i), p!(2 * i + 1)) \in r))\}$

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

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

fun *quotient-dist* :: 'x rel \Rightarrow 'x Distance \Rightarrow 'x set Distance **where**
quotient-dist r d A B =
 $\text{Inf } (\bigcup \{\{\text{path-length } p \ d \mid p. p \in \text{admissible-paths } r \ A \ B\}\})$

fun *distance-infimum*_Q :: 'x Distance \Rightarrow 'x set Distance **where**
*distance-infimum*_Q d A B = $\text{Inf } \{d \ a \ b \mid a \ b. a \in A \wedge b \in B\}$

fun *simple* :: 'x rel \Rightarrow 'x set \Rightarrow 'x Distance \Rightarrow bool **where**
simple r X d =
 $(\forall A \in X // r. (\exists a \in A. \forall B \in X // r. \text{distance-infimum}_Q \ d \ A \ B = \text{Inf } \{d \ a \ b \mid b. b \in B\}))$

— We call a distance simple with respect to a relation if for all relation classes, there is an a in A that minimizes the infimum distance between A and all B such that the infimum distance between these sets coincides with the infimum distance over all b in B for a fixed a .

fun *product'* :: 'x rel \Rightarrow ('x * 'x) rel **where**
product' r = $\{(p_1, p_2). ((fst \ p_1, fst \ p_2) \in r \wedge snd \ p_1 = snd \ p_2) \vee ((snd \ p_1, snd \ p_2) \in r \wedge fst \ p_1 = fst \ p_2)\}$

Auxiliary Lemmas

lemma *tot-dist-invariance-is-congruence*:

fixes

$d :: 'x \text{ Distance}$ **and**

$r :: 'x \text{ rel}$

shows $(total_invariance_{\mathcal{D}}\ d\ r) = (tup\ d\ respects\ (product\ r))$
 $\langle proof \rangle$

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**

$sym: sym_on\ X\ r \implies sym_on\ (X \times X)\ (product\ r)$

$\langle proof \rangle$

theorem *dist-pass-to-quotient*:

fixes

$d :: 'x\ Distance$ **and**

$r :: 'x\ rel$ **and**

$X :: 'x\ set$

assumes

$equiv_X_r: equiv\ X\ r$ **and**

$tot_inv_dist_d_r: total_invariance_{\mathcal{D}}\ d\ r$

shows $\forall\ A\ B. A \in X \ //\ r \wedge B \in X \ //\ r$

$\longrightarrow (\forall\ a\ b. a \in A \wedge b \in B \longrightarrow distance_{\mathcal{Q}}\ d\ A\ B = d\ a\ b)$

$\langle proof \rangle$

lemma *relation-paths-subset*:

fixes

$n :: nat$ **and**

$p :: 'x\ list$ **and**

$r :: 'x\ rel$ **and**

$X :: 'x\ set$

assumes $r \subseteq X \times X$

shows $\forall\ p. p \in relation_paths\ r \longrightarrow (\forall\ i < length\ p. p[i] \in X)$

$\langle proof \rangle$

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 $triangle_ineq\ X\ d \wedge p \in relation_paths\ r \wedge total_invariance_{\mathcal{D}}\ d\ r$

$\wedge a \in X \wedge b \in X \longrightarrow path_length\ (a\#\!p@[b])\ d \geq d\ a\ b$

$\langle proof \rangle$

lemma *quotient-dist-coincides-with-dist_Q*:

fixes
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$
assumes
 $\text{equiv}: \text{equiv } X \ r$ **and**
 $\text{tri}: \text{triangle-ineq } X \ d$ **and**
 $\text{invar}: \text{total-invariance}_{\mathcal{D}} \ d \ r$
shows $\forall A \in X // r. \forall B \in X // r. \text{quotient-dist } r \ d \ A \ B = \text{distance}_{\mathcal{Q}} \ d \ A \ B$
 $\langle \text{proof} \rangle$

lemma *inf-dist-coincides-with-dist_Q*:
fixes
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$
assumes
 $\text{equiv-}X\text{-}r: \text{equiv } X \ r$ **and**
 $\text{tot-inv-}d\text{-}r: \text{total-invariance}_{\mathcal{D}} \ d \ r$
shows $\forall A \in X // r. \forall B \in X // r. \text{distance-infimum}_{\mathcal{Q}} \ d \ A \ B = \text{distance}_{\mathcal{Q}} \ d \ A \ B$
 $\langle \text{proof} \rangle$

lemma *inf-helper*:
fixes
 $A :: 'x \text{ set}$ **and**
 $B :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$
shows $\text{Inf } \{d \ a \ b \mid a \ b. \ a \in A \wedge b \in B\} =$
 $\text{Inf } \{\text{Inf } \{d \ a \ b \mid b. \ b \in B\} \mid a. \ a \in A\}$
 $\langle \text{proof} \rangle$

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 $\text{simple } (\text{action-induced-rel } (\text{carrier } G) \ Y \ \varphi) \ Y \ d$
 $\langle \text{proof} \rangle$

lemma *tot-invar-dist-simple*:
fixes
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$

assumes
equiv-on-X: *equiv X r* **and**
invar: *total-invariance_D d r*
shows *simple r X d*
 ⟨*proof*⟩

5.7.2 Quotient Consensus and Results

fun *elections-K_Q* :: (*'a, 'v*) *Election rel* \Rightarrow (*'a, 'v, 'r Result*) *Consensus-Class*
 \Rightarrow (*'a, 'v*) *Election set set* **where**
elections-K_Q r C = (*elections-K C*) // *r*

fun (**in result**) *limit-set_Q* :: (*'a, 'v*) *Election set* \Rightarrow *'r set* \Rightarrow *'r set* **where**
limit-set_Q X res = $\bigcap \{ \text{limit-set (alternatives- \mathcal{E} E) res} \mid E. E \in X \}$

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:

fixes
X :: *'x set* **and**
Y :: *'x set* **and**
Z :: *'x set* **and**
r :: *'x rel*
assumes
equiv X r **and**
Y \subseteq *X* **and**
Z \subseteq *X* **and**
Z \in *Y* // *r* **and**
closed-restricted-rel r X Y
shows *Z* \subseteq *Y*
 ⟨*proof*⟩

lemma (**in result**) *limit-set-invar*:

fixes
d :: (*'a, 'v*) *Election Distance* **and**
r :: (*'a, 'v*) *Election rel* **and**
C :: (*'a, 'v, 'r Result*) *Consensus-Class* **and**
X :: (*'a, 'v*) *Election set* **and**
A :: (*'a, 'v*) *Election set*
assumes
quot-class: *A* \in *X* // *r* **and**
equiv-rel: *equiv X r* **and**
cons-subset: *elections-K C* \subseteq *X* **and**
invar-res: *is-symmetry* ($\lambda E. \text{limit-set (alternatives- \mathcal{E} E) UNIV}$) (*Invariance r*)
shows $\forall a \in A. \text{limit-set (alternatives- \mathcal{E} a) UNIV} = \text{limit-set}_Q A UNIV$
 ⟨*proof*⟩

lemma (**in result**) *preimg-invar*:

fixes
f :: *'x* \Rightarrow *'y* **and**

$domain_f :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$
assumes
 $equiv_rel: equiv\ X\ r$ **and**
 $cons_subset: domain_f \subseteq X$ **and**
 $closed_domain: closed_restricted_rel\ r\ X\ domain_f$ **and**
 $invar_f: is_symmetry\ f\ (Invariance\ (Restr\ r\ domain_f))$
shows $\forall\ y. (preimg\ f\ domain_f\ y) // r = preimg\ (\pi_Q\ f)\ (domain_f // r)\ y$
 $\langle proof \rangle$

lemma *minimizer-helper*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $domain_f :: 'x \text{ set}$ **and**
 $d :: 'x \text{ Distance}$ **and**
 $Y :: 'y \text{ set}$ **and**
 $x :: 'x$ **and**
 $y :: 'y$
shows $y \in minimizer\ f\ domain_f\ d\ Y\ x =$
 $(y \in Y \wedge (\forall\ y' \in Y.$
 $Inf\ (d\ x\ ' (preimg\ f\ domain_f\ y)) \leq Inf\ (d\ x\ ' (preimg\ f\ domain_f\ y'))))$
 $\langle proof \rangle$

lemma *rewr-singleton-set-system-union*:
fixes
 $Y :: 'x \text{ set set}$ **and**
 $X :: 'x \text{ set}$
assumes $Y \subseteq singleton_set_system\ X$
shows
 $singleton_set_union: x \in \bigcup Y \longleftrightarrow \{x\} \in Y$ **and**
 $obtain_singleton: A \in singleton_set_system\ X \longleftrightarrow (\exists\ x \in X. A = \{x\})$
 $\langle proof \rangle$

lemma *union-inf*:
fixes $X :: ereal\ set\ set$
shows $Inf\ \{Inf\ A \mid A. A \in X\} = Inf\ (\bigcup X)$
 $\langle proof \rangle$

5.7.3 Quotient Distance Rationalization

fun (in result) $\mathcal{R}_Q :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance}$
 $\Rightarrow ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ set}$ **where**
 $\mathcal{R}_Q\ r\ d\ C\ A =$
 $\bigcup (minimizer\ (\pi_Q\ (elect_r \circ fun_E\ (rule_K\ C)))\ (elections_K_Q\ r\ C)$
 $(distance_infimum_Q\ d)\ (singleton_set_system\ (limit_set_Q\ A\ UNIV))\ A)$
fun (in result) $distance_R_Q :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance}$

$\Rightarrow ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
 $\Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ Result} \text{ where}$

$\text{distance-}\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A =$
 $(\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit-set} (\text{alternatives-}\mathcal{E} \ E) \text{ UNIV}) \ A - \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\{\})$

Hadjibeyli and Wilson 2016 4.17

theorem (in result) *invar-dr-simple-dist-imp-quotient-dr-winners:*

fixes

$d :: ('a, 'v) \text{ Election Distance} \text{ and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \text{ and}$
 $r :: ('a, 'v) \text{ Election rel} \text{ and}$
 $X :: ('a, 'v) \text{ Election set} \text{ and}$
 $A :: ('a, 'v) \text{ Election set}$

assumes

$\text{simple: simple } r \ X \ d \text{ and}$
 $\text{closed-domain: closed-restricted-rel } r \ X \ (\text{elections-}\mathcal{K} \ C) \text{ and}$
 invar-res:
 $\text{is-symmetry } (\lambda E. \text{limit-set} (\text{alternatives-}\mathcal{E} \ E) \text{ UNIV}) \ (\text{Invariance } r) \text{ and}$
 $\text{invar-C: is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$
 $\quad (\text{Invariance } (\text{Restr } r \ (\text{elections-}\mathcal{K} \ C))) \text{ and}$
 $\text{invar-dr: is-symmetry } (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ (\text{Invariance } r) \text{ and}$
 $\text{quot-class: } A \in X \ // \ r \text{ and}$
 $\text{equiv-rel: equiv } X \ r \text{ and}$
 $\text{cons-subset: elections-}\mathcal{K} \ C \subseteq X$

shows $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A = \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$

(proof)

theorem (in result) *invar-dr-simple-dist-imp-quotient-dr:*

fixes

$d :: ('a, 'v) \text{ Election Distance} \text{ and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \text{ and}$
 $r :: ('a, 'v) \text{ Election rel} \text{ and}$
 $X :: ('a, 'v) \text{ Election set} \text{ and}$
 $A :: ('a, 'v) \text{ Election set}$

assumes

$\text{simple: simple } r \ X \ d \text{ and}$
 $\text{closed-domain: closed-restricted-rel } r \ X \ (\text{elections-}\mathcal{K} \ C) \text{ and}$
 invar-res:
 $\text{is-symmetry } (\lambda E. \text{limit-set} (\text{alternatives-}\mathcal{E} \ E) \text{ UNIV})$
 $\quad (\text{Invariance } r) \text{ and}$
 $\text{invar-C: is-symmetry } (\text{elect-r} \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$
 $\quad (\text{Invariance } (\text{Restr } r \ (\text{elections-}\mathcal{K} \ C))) \text{ and}$
 $\text{invar-dr: is-symmetry } (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ (\text{Invariance } r) \text{ and}$
 $\text{quot-class: } A \in X \ // \ r \text{ and}$
 $\text{equiv-rel: equiv } X \ r \text{ and}$
 $\text{cons-subset: elections-}\mathcal{K} \ C \subseteq X$

shows $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\text{distance-}\mathcal{R} \ d \ C)) \ A = \text{distance-}\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$

$\langle proof \rangle$

end

5.8 Result and Property Locale Code Generation

```
theory Interpretation-Code
imports Electoral-Module
        Distance-Rationalization
begin
 $\langle ML \rangle$ 
```

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  $SCF\text{-result.electoral-module } m =$ 
       $(\forall A V p. \text{profile } V A p \longrightarrow \text{well-formed-SCF } A (m V A p))$ 
 $\langle proof \rangle$ 
```

```
lemma  $\mathcal{R}_W$ -SCF-code-lemma:
fixes
   $d :: ('a, 'v) \text{ Election Distance}$  and
   $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$  and
   $V :: 'v \text{ set}$  and
   $A :: 'a \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
shows  $SCF\text{-result.}\mathcal{R}_W d K V A p =$ 
       $\text{arg-min-set (score } d K (A, V, p)) (\text{limit-set-SCF } A UNIV)$ 
 $\langle proof \rangle$ 
```

```
lemma distance- $\mathcal{R}$ -SCF-code-lemma:
fixes
   $d :: ('a, 'v) \text{ Election Distance}$  and
   $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$  and
   $V :: 'v \text{ set}$  and
   $A :: 'a \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$ 
shows  $SCF\text{-result.distance-}\mathcal{R} d K V A p =$ 
       $(SCF\text{-result.}\mathcal{R}_W d K V A p,$ 
         $(\text{limit-set-SCF } A UNIV) - SCF\text{-result.}\mathcal{R}_W d K V A p,$ 
         $\{\})$ 
 $\langle proof \rangle$ 
```

```
lemma  $\mathcal{R}_W$ -std-SCF-code-lemma:
fixes
   $d :: ('a, 'v) \text{ Election Distance}$  and
```

```

    K :: ('a, 'v, 'a Result) Consensus-Class and
    V :: 'v set and
    A :: 'a set and
    p :: ('a, 'v) Profile
  shows SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std d K V A p =
    arg-min-set (score-std d K (A, V, p)) (limit-set-SCF A UNIV)
  <proof>

```

lemma *distance- \mathcal{R} -std-SCF-code-lemma:*

```

  fixes
    d :: ('a, 'v) Election Distance and
    K :: ('a, 'v, 'a Result) Consensus-Class and
    V :: 'v set and
    A :: 'a set and
    p :: ('a, 'v) Profile
  shows SCF-result.distance- $\mathcal{R}$ -std d K V A p =
    (SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std d K V A p,
     (limit-set-SCF A UNIV) - SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std d K V A p,
     {})
  <proof>

```

lemma *anonymity-SCF-code-lemma:*

```

  shows SCF-result.anonymity =
    ( $\lambda$  m::('a, 'v, 'a Result) Electoral-Module).
    SCF-result.electoral-module m  $\wedge$ 
    ( $\forall$  A V p  $\pi$ ::('v  $\Rightarrow$  'v).
     bij  $\pi \longrightarrow$  (let (A', V', q) = (rename  $\pi$  (A, V, p)) in
      finite-profile V A p  $\wedge$  finite-profile V' A' q  $\longrightarrow$  m V A p = m V' A' q)))
  <proof>

```

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. $\mathcal{R}_{\mathcal{W}}$   $\mathcal{R}_{\mathcal{W}}$ -SCF-code-lemma]]
declare [[lc-add SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std  $\mathcal{R}_{\mathcal{W}}$ -std-SCF-code-lemma]]
declare [[lc-add SCF-result.distance- $\mathcal{R}$  distance- $\mathcal{R}$ -SCF-code-lemma]]
declare [[lc-add SCF-result.distance- $\mathcal{R}$ -std distance- $\mathcal{R}$ -std-SCF-code-lemma]]
declare [[lc-add SCF-result.anonymity anonymity-SCF-code-lemma]]

```

Constant aliases to use when exporting code instead of the interpreted functions

```

definition  $\mathcal{R}_{\mathcal{W}}$ -SCF-code = SCF-result. $\mathcal{R}_{\mathcal{W}}$ 
definition  $\mathcal{R}_{\mathcal{W}}$ -std-SCF-code = SCF-result. $\mathcal{R}_{\mathcal{W}}$ -std
definition distance- $\mathcal{R}$ -SCF-code = SCF-result.distance- $\mathcal{R}$ 
definition distance- $\mathcal{R}$ -std-SCF-code = SCF-result.distance- $\mathcal{R}$ -std
definition electoral-module-SCF-code = SCF-result.electoral-module
definition anonymity-SCF-code = SCF-result.anonymity

```

<ML>

end

5.9 Drop Module

```

theory Drop-Module
  imports Component-Types/Electoral-Module
           Component-Types/Social-Choice-Types/Result
begin

```

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according drop module rejects the lexicographically first n alternatives (from A) and defers the rest. It is primarily used as counterpart to the pass module in a parallel composition, in order to segment the alternatives into two groups.

5.9.1 Definition

```

fun drop-module :: nat  $\Rightarrow$  'a Preference-Relation
       $\Rightarrow$  ('a, 'v, 'a Result) Electoral-Module where
  drop-module n r V A p =
    ({},
     {a  $\in$  A. rank (limit A r) a  $\leq$  n},
     {a  $\in$  A. rank (limit A r) a  $>$  n})

```

5.9.2 Soundness

```

theorem drop-mod-sound[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows SCF-result.electoral-module (drop-module n r)
  <proof>

```

```

lemma voters-determine-drop-mod:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows voters-determine-election (drop-module n r)
  <proof>

```

5.9.3 Non-Electing

The drop module is non-electing.

```

theorem drop-mod-non-electing[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows non-electing (drop-module n r)
    ⟨proof⟩

```

5.9.4 Properties

The drop module is strictly defer-monotone.

```

theorem drop-mod-def-lift-inv[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows defer-lift-invariance (drop-module n r)
    ⟨proof⟩

```

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>

```

```

lemma voters-determine-pass-mod:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows voters-determine-election (pass-module n r)
  <proof>

```

5.10.3 Non-Blocking

The pass module is non-blocking.

```

theorem pass-mod-non-blocking[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  assumes
    order: linear-order r and
    g0-n: n > 0
  shows non-blocking (pass-module n r)
  <proof>

```

5.10.4 Non-Electing

The pass module is non-electing.

```

theorem pass-mod-non-electing[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  assumes linear-order r
  shows non-electing (pass-module n r)
  <proof>

```

5.10.5 Properties

The pass module is strictly defer-monotone.

```

theorem pass-mod-dl-inv[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  assumes linear-order r
  shows defer-lift-invariance (pass-module n r)
  <proof>

```

```

theorem pass-zero-mod-def-zero[simp]:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes linear-order  $r$ 
  shows defers 0 (pass-module 0 r)
  <proof>

```

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>

```

```

theorem pass-two-mod-def-two:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes linear-order  $r$ 
  shows defers 2 (pass-module 2 r)
  <proof>

```

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 ::  $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  where
  elect-module  $V \ A \ p = (A, \{\}, \{\})$ 

```

5.11.2 Soundness

```

theorem elect-mod-sound[simp]: SCF-result.electoral-module elect-module
  <proof>

```

lemma *elect-mod-only-voters: voters-determine-election elect-module*
 ⟨*proof*⟩

5.11.3 Electing

theorem *elect-mod-electing[simp]: electing elect-module*
 ⟨*proof*⟩

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
 ⟨*proof*⟩

lemma *plurality-mod-elim-equiv*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assumes
 $\text{non-empty-}A: A \neq \{\}$ **and**
 $\text{fin-}A: \text{finite } A$ **and**
 $\text{prof: profile } V \ A \ p$
shows $\text{plurality } V \ A \ p = \text{plurality}' \ V \ A \ p$
 $\langle \text{proof} \rangle$

5.12.2 Soundness

theorem *plurality-sound[simp]*: $\text{SCF-result.electoral-module plurality}$
 $\langle \text{proof} \rangle$

theorem *plurality'-sound[simp]*: $\text{SCF-result.electoral-module plurality}'$
 $\langle \text{proof} \rangle$

lemma *voters-determine-plurality-score*: $\text{voters-determine-evaluation plurality-score}$
 $\langle \text{proof} \rangle$

lemma *voters-determine-plurality*: $\text{voters-determine-election plurality}$
 $\langle \text{proof} \rangle$

5.12.3 Non-Blocking

The plurality module is non-blocking.

theorem *plurality-mod-non-blocking[simp]*: $\text{non-blocking plurality}$
 $\langle \text{proof} \rangle$

5.12.4 Non-Electing

The plurality module is non-electing.

theorem *plurality-non-electing[simp]*: $\text{non-electing plurality}$
 $\langle \text{proof} \rangle$

theorem *plurality'-non-electing[simp]*: $\text{non-electing plurality}'$
 $\langle \text{proof} \rangle$

5.12.5 Property

lemma *plurality-def-inv-mono-alts*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**


```

  p :: ('a, 'v) Profile and
  q :: ('a, 'v) Profile and
  a :: 'a
assumes
  defer-a: a ∈ defer plurality V A p and
  lift-a: lifted V A p q a
shows defer plurality V A q = defer plurality V A p
        ∨ defer plurality V A q = {a}
⟨proof⟩

```

The plurality rule is invariant-monotone.

theorem *plurality-mod-def-inv-mono[simp]: defer-invariant-monotonicity plurality*
 ⟨proof⟩

end

5.13 Borda Module

```

theory Borda-Module
imports Component-Types/Elimination-Module
begin

```

This is the Borda module used by the Borda rule. The Borda rule is a voting rule, where on each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.13.1 Definition

```

fun borda-score :: ('a, 'v) Evaluation-Function where
  borda-score V x A p = (∑ y ∈ A. (prefer-count V p x y))

```

```

fun borda :: ('a, 'v, 'a Result) Electoral-Module where
  borda V A p = max-eliminator borda-score V A p

```

5.13.2 Soundness

theorem *borda-sound: SCF-result.electoral-module borda*
 ⟨proof⟩

5.13.3 Non-Blocking

The Borda module is non-blocking.

theorem *borda-mod-non-blocking[simp]: non-blocking borda*
<proof>

5.13.4 Non-Electing

The Borda module is non-electing.

theorem *borda-mod-non-electing[simp]: non-electing borda*
<proof>

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*
<proof>

5.14.3 Property

theorem *condorcet-score-is-condorcet-rating: condorcet-rating condorcet-score*

<proof>

theorem *condorcet-is-dcc: defer-condorcet-consistency condorcet*
<proof>

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 $V\ x\ A\ p =$
 $\text{card } \{y \in A . \text{wins } V\ x\ p\ y\} - \text{card } \{y \in A . \text{wins } V\ y\ p\ x\}$

fun *copeland* :: ('a, 'v, 'a *Result*) *Electoral-Module* **where**
copeland $V\ A\ p = \text{max-eliminator } \text{copeland-score } V\ A\ p$

5.15.2 Soundness

theorem *copeland-sound: SCF-result.electoral-module copeland*
<proof>

5.15.3 Only Voters Determine Election Result

lemma *voters-determine-copeland-score: voters-determine-evaluation copeland-score*
<proof>

theorem *voters-determine-copeland: voters-determine-election copeland*
<proof>

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 $\text{card } \{a \in A. \text{wins } V w p a\} = \text{card } A - 1$
 $\langle \text{proof} \rangle$

For a Condorcet winner w , we have: " $\text{card } \{y \in A. \text{wins } y p w = 0\}$ ".

lemma *cond-winner-imp-loss-count*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $w :: 'a$
assumes *condorcet-winner* $V A p w$
shows $\text{card } \{a \in A. \text{wins } V a p w\} = 0$
 $\langle \text{proof} \rangle$

Copeland score of a Condorcet winner.

lemma *cond-winner-imp-copeland-score*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $w :: 'a$
assumes *condorcet-winner* $V A p w$
shows *copeland-score* $V w A p = \text{card } A - 1$
 $\langle \text{proof} \rangle$

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

lemma *non-cond-winner-imp-win-count*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $w :: 'a$ **and**
 $l :: 'a$
assumes
winner: *condorcet-winner* $V A p w$ **and**
loser: $l \neq w$ **and**
l-in-A: $l \in A$
shows $\text{card } \{a \in A. \text{wins } V l p a\} \leq \text{card } A - 2$
 $\langle \text{proof} \rangle$

5.15.5 Property

The Copeland score is Condorcet rating.

theorem *copeland-score-is-cr: condorcet-rating copeland-score*
<proof>

theorem *copeland-is-dcc: defer-condorcet-consistency copeland*
<proof>

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* =
 $\text{Min } \{\text{prefer-count } V p x y \mid y . y \in A - \{x\}\}$

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*
<proof>

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**
 $l-in-A: l \in A$ **and**
 $l-neq-w: l \neq w$
shows $minimax-score\ V\ l\ A\ p \leq prefer-count\ V\ p\ l\ w$
 $\langle proof \rangle$

5.16.4 Property

theorem $minimax-score-cond-rating: condorcet-rating\ minimax-score$
 $\langle proof \rangle$

theorem $minimax-is-dcc: defer-condorcet-consistency\ minimax$
 $\langle proof \rangle$

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

The drop module rejects n alternatives (if there are at least n alternatives).

```
theorem drop-two-mod-rej-n[simp]:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes linear-order  $r$ 
  shows rejects n (drop-module n r)
   $\langle \text{proof} \rangle$ 
```

The pass and drop module are (disjoint-)compatible.

```
theorem drop-pass-disj-compat[simp]:
  fixes
     $r :: 'a \text{ Preference-Relation}$  and
     $n :: \text{nat}$ 
  assumes linear-order  $r$ 
  shows disjoint-compatibility (drop-module n r) (pass-module n r)
   $\langle \text{proof} \rangle$ 
```

end

6.2 Revision Composition

theory *Revision-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

A revised electoral module rejects all originally rejected or deferred alternatives, and defers the originally elected alternatives. It does not elect any alternatives.

6.2.1 Definition

fun *revision-composition* :: ('a, 'v, 'a Result) *Electoral-Module*
 \Rightarrow ('a, 'v, 'a Result) *Electoral-Module* **where**
revision-composition m V A p = ({}, A - elect m V A p, elect m V A p)

abbreviation *rev* :: ('a, 'v, 'a Result) *Electoral-Module*
 \Rightarrow ('a, 'v, 'a Result) *Electoral-Module* (\downarrow 50) **where**
m \downarrow == *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>

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)
 <proof>

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 \downarrow)
 <proof>

Revising an electing electoral module results in a non-blocking electoral module.

theorem *rev-comp-non-blocking[simp]*:
fixes $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes *electing* m
shows *non-blocking* $(m \downarrow)$
 $\langle \text{proof} \rangle$

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* $(m \downarrow)$
 $\langle \text{proof} \rangle$

end

6.3 Sequential Composition

theory *Sequential-Composition*
imports *Basic-Modules/Component-Types/Electoral-Module*
begin

The sequential composition creates a new electoral module from two electoral modules. In a sequential composition, the second electoral module makes decisions over alternatives deferred by the first electoral module.

6.3.1 Definition

fun *sequential-composition* $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **where**
sequential-composition $m \ n \ V \ A \ p =$
 $(\text{let } \text{new-}A = \text{defer } m \ V \ A \ p;$
 $\text{new-}p = \text{limit-profile } \text{new-}A \ p \text{ in } ($
 $(\text{elect } m \ V \ A \ p) \cup (\text{elect } n \ V \ \text{new-}A \ \text{new-}p),$
 $(\text{reject } m \ V \ A \ p) \cup (\text{reject } n \ V \ \text{new-}A \ \text{new-}p),$
 $\text{defer } n \ V \ \text{new-}A \ \text{new-}p))$

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

```

    (infix  $\triangleright$  50) where
     $m \triangleright n == \text{sequential-composition } m \ n$ 

fun sequential-composition' :: ('a, 'v, 'a Result) Electoral-Module
    ⇒ ('a, 'v, 'a Result) Electoral-Module
    ⇒ ('a, 'v, 'a Result) Electoral-Module where
    sequential-composition'  $m \ n \ V \ A \ p =$ 
    (let (m-e, m-r, m-d) =  $m \ V \ A \ p$ ; new-A = m-d;
        new-p = limit-profile new-A  $p$ ;
        (n-e, n-r, n-d) =  $n \ V \ \text{new-A} \ \text{new-p}$  in
        ( $m\text{-e} \cup n\text{-e}, m\text{-r} \cup n\text{-r}, n\text{-d}$ ))

lemma voters-determine-seq-comp:
fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
assumes
    voters-determine-election  $m \wedge$  voters-determine-election  $n$ 
shows voters-determine-election ( $m \triangleright n$ )
<proof>

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 (( $m \triangleright n$ )  $V \ A \ p$ )
<proof>

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>

lemma seq-comp-alt-eq[fundef-cong, code]: sequential-composition = sequential-composition'
<proof>

```

6.3.2 Soundness

theorem *seq-comp-sound*[simp]:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $SCF\text{-result.electoral-module } m$ **and**
 $SCF\text{-result.electoral-module } n$
shows $SCF\text{-result.electoral-module } (m \triangleright n)$
 $\langle \text{proof} \rangle$

6.3.3 Lemmas

lemma *seq-comp-decrease-only-defer*:
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
 $\text{module-}m: SCF\text{-result.electoral-module } m$ **and**
 $\text{module-}n: SCF\text{-result.electoral-module } n$ **and**
 $\text{prof: profile } V \ A \ p$ **and**
 $\text{empty-defer: defer } m \ V \ A \ p = \{\}$
shows $(m \triangleright n) \ V \ A \ p = m \ V \ A \ p$
 $\langle \text{proof} \rangle$

lemma *seq-comp-def-then-elect*:
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
 $n\text{-electing-}m: \text{non-electing } m$ **and**
 $\text{def-one-}m: \text{defers } 1 \ m$ **and**
 $\text{electing-}n: \text{electing } n$ **and**
 $f\text{-prof: finite-profile } V \ A \ p$
shows $\text{elect } (m \triangleright n) \ V \ A \ p = \text{defer } m \ V \ A \ p$
 $\langle \text{proof} \rangle$

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
 $SCF\text{-result.electoral-module } m \text{ and}$
 $SCF\text{-result.electoral-module } n \text{ and}$
 $finite\text{-profile } V \ A \ p$
shows $card \ (defer \ (m \triangleright n) \ V \ A \ p) \leq card \ (defer \ m \ V \ A \ p)$
 $\langle proof \rangle$

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
 $SCF\text{-result.electoral-module } m \text{ and}$
 $SCF\text{-result.electoral-module } n \text{ and}$
 $profile \ V \ A \ p$
shows $defer \ (m \triangleright n) \ V \ A \ p \subseteq defer \ m \ V \ A \ p$
 $\langle proof \rangle$

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 $defer \ (m \triangleright n) \ V \ A \ p =$
 $defer \ n \ V \ (defer \ m \ V \ A \ p) \ (limit\text{-profile} \ (defer \ m \ V \ A \ p) \ p)$
 $\langle proof \rangle$

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 $elect \ (m \triangleright n) \ V \ A \ p =$
 $elect \ n \ V \ (defer \ m \ V \ A \ p)$
 $(limit\text{-profile} \ (defer \ m \ V \ A \ p) \ p) \cup (elect \ m \ V \ A \ p)$
 $\langle proof \rangle$

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
 $SCF\text{-result.electoral-module } m$ **and**
 $eliminates\ 1\ n$ **and**
 $profile\ V\ A\ p$ **and**
 $card\ (defer\ m\ V\ A\ p) > 1$
shows $defer\ (m \triangleright n)\ V\ A\ p \subset defer\ m\ V\ A\ p$
 $\langle proof \rangle$

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
 $a \in (defer\ (m \triangleright n)\ V\ A\ p)$ **and**
 $SCF\text{-result.electoral-module } m \wedge SCF\text{-result.electoral-module } n$ **and**
 $profile\ V\ A\ p$
shows $a \in defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-profile}\ (defer\ m\ V\ A\ p)\ p) \wedge$
 $a \in defer\ m\ V\ A\ p$
 $\langle proof \rangle$

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
 $non\text{-blocking-}m: non\text{-blocking } m$ **and**
 $non\text{-blocking-}n: non\text{-blocking } n$
shows $non\text{-blocking } (m \triangleright n)$
 $\langle proof \rangle$

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
 $non\text{-electing } m$ **and**
 $non\text{-electing } n$
shows $non\text{-electing } (m \triangleright n)$

$\langle proof \rangle$

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

def-one-m: *defers 1 m* **and**
electing-n: *electing n*

shows *electing* ($m \triangleright n$)

$\langle proof \rangle$

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 ($m \triangleright n$) $V \ A \ p = (m \triangleright n) \ V \ A \ q$

$\langle proof \rangle$

Sequential composition preserves the property defer-lift-invariance.

theorem *seq-comp-presv-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

defer-lift-invariance m **and**
defer-lift-invariance n **and**
voters-determine-election n

shows *defer-lift-invariance* ($m \triangleright n$)

$\langle proof \rangle$

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
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{non-blocking-}m$: $\text{non-blocking } m$ **and**
 $\text{non-electing-}m$: $\text{non-electing } m$ **and**
 $\text{def-one-}n$: $\text{defers } 1 \ n$
shows $\text{defers } 1 \ (m \triangleright n)$
 $\langle \text{proof} \rangle$

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 $\text{disj-compat-seq}[simp]$:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $m' :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 compatible : $\text{disjoint-compatibility } m \ n$ **and**
 $\text{module-}m'$: $\text{SCF-result.electoral-module } m'$ **and**
 $\text{voters-determine-}m'$: $\text{voters-determine-election } m'$
shows $\text{disjoint-compatibility } (m \triangleright m') \ n$
 $\langle \text{proof} \rangle$

theorem $\text{seq-comp-cond-compat}[simp]$:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{dcc-}m$: $\text{defer-condorcet-consistency } m$ **and**
 $\text{nb-}n$: $\text{non-blocking } n$ **and**
 $\text{ne-}n$: $\text{non-electing } n$
shows $\text{condorcet-compatibility } (m \triangleright n)$
 $\langle \text{proof} \rangle$

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 $\text{seq-comp-dcc}[simp]$:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{dcc-}m$: $\text{defer-condorcet-consistency } m$ **and**
 $\text{nb-}n$: $\text{non-blocking } n$ **and**
 $\text{ne-}n$: $\text{non-electing } n$
shows $\text{defer-condorcet-consistency } (m \triangleright n)$
 $\langle \text{proof} \rangle$

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⟩

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

6.4 Parallel Composition

theory *Parallel-Composition*
imports *Basic-Modules/Component-Types/Aggregator*
Basic-Modules/Component-Types/Electoral-Module
begin

The parallel composition composes a new electoral module from two electoral modules combined with an aggregator. Therein, the two modules each make a decision and the aggregator combines them to a single (aggregated) result.

6.4.1 Definition

fun *parallel-composition* :: ('a, 'v, 'a Result) Electoral-Module
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module
 \Rightarrow 'a Aggregator
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module **where**
parallel-composition m n agg V A p = agg A (m V A p) (n V A p)

abbreviation *parallel* :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow 'a Aggregator
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module
 (- || - [50, 1000, 51] 50) **where**
 m ||_a n == *parallel-composition* m n a

6.4.2 Soundness

theorem *par-comp-sound[simp]*:
fixes
 m :: ('a, 'v, 'a Result) Electoral-Module **and**
 n :: ('a, 'v, 'a Result) Electoral-Module **and**
 a :: 'a Aggregator
assumes
SCF-result.electoral-module m **and**
SCF-result.electoral-module n **and**
aggregator a
shows *SCF-result.electoral-module* (m ||_a n)
 <proof>

6.4.3 Composition Rule

Using a conservative aggregator, the parallel composition preserves the property non-electing.

theorem *conserv-agg-presv-non-electing[simp]*:
fixes
 m :: ('a, 'v, 'a Result) Electoral-Module **and**
 n :: ('a, 'v, 'a Result) Electoral-Module **and**
 a :: 'a Aggregator
assumes
non-electing-m: non-electing m **and**
non-electing-n: non-electing n **and**
conservative: agg-conservative a
shows *non-electing* (m ||_a n)
 <proof>

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
     $\neg t (acc \triangleright V \ A \ p)$  and
     $\text{defer } (acc \triangleright m) \ V \ A \ p \subset \text{defer } acc \ V \ A \ p$  and
     $\text{finite } (\text{defer } acc \ V \ A \ p)$ 
  shows  $((acc \triangleright m, m, t, V, A, p), (acc, m, t, V, A, p)) \in$ 
     $\text{measure } (\lambda (acc, m, t, V, A, p). \text{card } (\text{defer } acc \ V \ A \ p))$ 
   $\langle \text{proof} \rangle$ 

```

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
   $\text{finite } (\text{defer } acc \ V \ A \ p) \wedge (\text{defer } (acc \triangleright m) \ V \ A \ p) \subset (\text{defer } acc \ V \ A \ p)$ 
   $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow$ 
   $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = acc \ V \ A \ p \mid$ 
   $\neg (\text{finite } (\text{defer } acc \ V \ A \ p) \wedge (\text{defer } (acc \triangleright m) \ V \ A \ p) \subset (\text{defer } acc \ V \ A \ p))$ 
   $\longrightarrow t (acc \ V \ A \ p) \Longrightarrow$ 
   $\text{loop-comp-helper } acc \ m \ t \ V \ A \ p = \text{loop-comp-helper } (acc \triangleright m) \ m \ t \ V \ A \ p$ 
   $\langle \text{proof} \rangle$ 
termination
   $\langle \text{proof} \rangle$ 

```

```

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
     $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p =$ 
       $(if\ (t\ (acc\ V\ A\ p) \vee \neg ((defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p))$ 
         $\vee\ infinite\ (defer\ acc\ V\ A\ p))$ 
         $then\ (acc\ V\ A\ p)\ else\ (loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ p))$ 
     $\langle proof \rangle$ 

function loop-composition ::  $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ 
   $\Rightarrow 'a \text{ Termination-Condition}$ 
   $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  where
     $t\ (\{\}, \{\}, A)$ 
       $\Rightarrow loop\text{-}composition\ m\ t\ V\ A\ p = defer\text{-}module\ V\ A\ p \mid$ 
     $\neg(t\ (\{\}, \{\}, A))$ 
       $\Rightarrow loop\text{-}composition\ m\ t\ V\ A\ p = (loop\text{-}comp\text{-}helper\ m\ m\ t)\ V\ A\ p$ 
     $\langle proof \rangle$ 
termination
     $\langle proof \rangle$ 

abbreviation loop ::  $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow 'a \text{ Termination-Condition}$ 
   $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$   $(- \circlearrowleft -)$  where
     $m \circlearrowleft_t \equiv loop\text{-}composition\ m\ t$ 

lemma loop-comp-code[code]:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $t :: 'a \text{ Termination-Condition}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{ Profile}$ 
  shows loop-composition  $m\ t\ V\ A\ p =$ 
     $(if\ (t\ (\{\}, \{\}, A))$ 
       $then\ (defer\text{-}module\ V\ A\ p)\ else\ (loop\text{-}comp\text{-}helper\ m\ m\ t)\ V\ A\ p)$ 
     $\langle proof \rangle$ 

lemma loop-comp-helper-imp-partit:
  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
 $\text{module-}m: \text{SCF-result.electoral-module } m$ **and**
 $\text{profile: profile } V A p$ **and**
 $\text{module-acc: SCF-result.electoral-module acc}$ **and**
 $\text{defer-card-}n: n = \text{card } (\text{defer acc } V A p)$
shows $\text{well-formed-SCF } A (\text{loop-comp-helper acc } m t V A p)$
 $\langle \text{proof} \rangle$

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 $\text{SCF-result.electoral-module } m$
shows $\text{SCF-result.electoral-module } (m \odot_t)$
 $\langle \text{proof} \rangle$

lemma $\text{loop-comp-helper-imp-no-def-incr:}$
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**
 $n :: \text{nat}$
assumes
 $\text{module-}m: \text{SCF-result.electoral-module } m$ **and**
 $\text{profile: profile } V A p$ **and**
 $\text{mod-acc: SCF-result.electoral-module acc}$ **and**
 $\text{card-}n\text{-defer-acc: } n = \text{card } (\text{defer acc } V A p)$
shows $\text{defer } (\text{loop-comp-helper acc } m t) V A p \subseteq \text{defer acc } V A p$
 $\langle \text{proof} \rangle$

6.5.3 Lemmas

lemma $\text{loop-comp-helper-def-lift-inv-helper:}$
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**
 $n :: \text{nat}$
assumes
 $\text{monotone-}m: \text{defer-lift-invariance } m$ **and**

prof: *profile* $V A p$ **and**
dli-acc: *defer-lift-invariance* *acc* **and**
card-n-defer: $n = \text{card } (\text{defer } \text{acc } V A p)$ **and**
defer-finite: *finite* $(\text{defer } \text{acc } V A p)$ **and**
voters-determine-m: *voters-determine-election* *m*
shows
 $\forall q a. a \in (\text{defer } (\text{loop-comp-helper } \text{acc } m t) V A p) \wedge \text{lifted } V A p q a \longrightarrow$
 $(\text{loop-comp-helper } \text{acc } m t) V A p = (\text{loop-comp-helper } \text{acc } m t) V A q$
 $\langle \text{proof} \rangle$

lemma *loop-comp-helper-def-lift-inv*:

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

$a \in \text{defer } (\text{loop-comp-helper } \text{acc } m t) V A p$

shows $(\text{loop-comp-helper } \text{acc } m t) V A p = (\text{loop-comp-helper } \text{acc } m t) V A q$
 $\langle \text{proof} \rangle$

lemma *lifted-imp-fin-prof*:

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 *lifted* $V A p q a$

shows *finite-profile* $V A p$
 $\langle \text{proof} \rangle$

lemma *loop-comp-helper-presv-def-lift-inv*:

fixes

$m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$ **and**
 $\text{acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

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>

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 = {}*
 <proof>

lemma *loop-comp-helper-iter-elim-def-n-helper:*
fixes
m :: ('a, 'v, 'a Result) Electoral-Module and
t :: 'a Termination-Condition and
acc :: ('a, 'v, 'a Result) Electoral-Module and
A :: 'a set and
V :: 'v set and
p :: ('a, 'v) Profile and
n :: nat and
x :: nat
assumes
non-electing-m: non-electing m and
single-elimination: eliminates 1 m and
terminate-if-n-left: $\forall r. t r = (\text{card } (\text{defer-r } r) = x)$ and
x-greater-zero: $x > 0$ and
prof: profile V A p and
n-acc-defer-card: $n = \text{card } (\text{defer acc V A p})$ and
n-ge-x: $n \geq x$ and
def-card-gt-one: $\text{card } (\text{defer acc V A p}) > 1$ and
acc-nonelect: non-electing acc
shows *card (defer (loop-comp-helper acc m t) V A p) = x*
 <proof>

lemma *loop-comp-helper-iter-elim-def-n:*
fixes
m :: ('a, 'v, 'a Result) Electoral-Module and
t :: 'a Termination-Condition and

```

    acc :: ('a, 'v, 'a Result) Electoral-Module and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    x :: nat
  assumes
    non-electing m and
    eliminates 1 m and
     $\forall r. (t\ r) = (\text{card } (\text{defer-r } r) = x)$  and
     $x > 0$  and
    profile V A p and
     $\text{card } (\text{defer acc V A p}) \geq x$  and
    non-electing acc
  shows  $\text{card } (\text{defer } (\text{loop-comp-helper acc m t}) V A p) = x$ 
  <proof>

```

lemma *iter-elim-def-n-helper*:

```

  fixes
    m :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile and
    x :: nat
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left:  $\forall r. (t\ r) = (\text{card } (\text{defer-r } r) = x)$  and
    x-greater-zero:  $x > 0$  and
    prof: profile V A p and
    enough-alternatives:  $\text{card } A \geq x$ 
  shows  $\text{card } (\text{defer } (m \circ_t) V A p) = x$ 
  <proof>

```

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>

```

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  $\odot_t$ )
<proof>

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  $\odot_t$ )
<proof>

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 $\text{max-parallel} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module (infix } \parallel_{\uparrow} 50) \text{ where}$
 $m \parallel_{\uparrow} n == \text{maximum-parallel-composition } m \ n$

6.6.2 Soundness

theorem $\text{max-par-comp-sound}$:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{SCF-result.electoral-module } m$ **and**
 $\text{SCF-result.electoral-module } n$
shows $\text{SCF-result.electoral-module } (m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

lemma $\text{voters-determine-max-par-comp}$:
fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{voters-determine-election } m$ **and**
 $\text{voters-determine-election } n$
shows $\text{voters-determine-election } (m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

6.6.3 Lemmas

lemma max-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
 $\text{module-m: SCF-result.electoral-module } m$ **and**
 $\text{module-n: SCF-result.electoral-module } n$ **and**
 $\text{prof-p: profile } V \ A \ p$ **and**
 $\text{a-in-A: } a \in A$
shows $\text{mod-contains-result } (m \parallel_{\uparrow} n) \ m \ V \ A \ p \ a \ \vee$
 $\text{mod-contains-result } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ a$
 $\langle \text{proof} \rangle$

lemma $\text{max-agg-rej-iff-both-reject}$:
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
 $\text{finite-profile } V \ A \ p$ **and**
 $\text{SCF-result.electoral-module } m$ **and**
 $\text{SCF-result.electoral-module } n$
shows $(a \in \text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p) =$
 $(a \in \text{reject } m \ V \ A \ p \wedge a \in \text{reject } n \ V \ A \ p)$
 $\langle \text{proof} \rangle$

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
 $f\text{-prof: finite-profile } V \ A \ p$ **and**
 $\text{module-m: SCF-result.electoral-module } m$ **and**
 $\text{module-n: SCF-result.electoral-module } n$ **and**
 $\text{rejected: } a \in \text{reject } n \ V \ A \ p$
shows $\text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$
 $\langle \text{proof} \rangle$

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
 $\text{finite-profile } V \ A \ p$ **and**
 $\text{SCF-result.electoral-module } m$ **and**
 $\text{SCF-result.electoral-module } n$ **and**
 $a \in \text{reject } n \ V \ A \ p$
shows $\text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ m \ V \ A \ p \ a$
 $\langle \text{proof} \rangle$

lemma *max-agg-rej-snd-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
 $f\text{-prof}: \text{finite-profile } V \ A \ p$ **and**
 $\text{module-}m: \text{SCF-result.electoral-module } m$ **and**
 $\text{module-}n: \text{SCF-result.electoral-module } n$ **and**
 $\text{rejected}: a \in \text{reject } m \ V \ A \ p$
shows $\text{mod-contains-result } n \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$
 $\langle \text{proof} \rangle$

lemma *max-agg-rej-snd-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
 $\text{finite-profile } V \ A \ p$ **and**
 $\text{SCF-result.electoral-module } m$ **and**
 $\text{SCF-result.electoral-module } n$ **and**
 $a \in \text{reject } m \ V \ A \ p$
shows $\text{mod-contains-result-sym } (m \parallel_{\uparrow} n) \ n \ V \ A \ p \ a$
 $\langle \text{proof} \rangle$

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
 $\text{SCF-result.electoral-module } m$ **and**
 $\text{SCF-result.electoral-module } n$ **and**
 $\text{profile } V \ A \ p$ **and**
 $\text{finite } A$
shows $\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p = (\text{reject } m \ V \ A \ p) \cap (\text{reject } n \ V \ A \ p)$
 $\langle \text{proof} \rangle$

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
 $(\forall p. \text{finite-profile } V\ A\ p \longrightarrow \text{mod-contains-result } m\ (m \parallel_{\uparrow} n)\ V\ A\ p\ a)$
 $\vee (\forall p. \text{finite-profile } V\ A\ p \longrightarrow \text{mod-contains-result } n\ (m \parallel_{\uparrow} n)\ V\ A\ p\ a)$
 $\langle \text{proof} \rangle$

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
non-electing m **and**
non-electing n
shows *non-electing* $(m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

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
compatible: *disjoint-compatibility* $m\ n$ **and**
monotone-m: *defer-lift-invariance* m **and**
monotone-n: *defer-lift-invariance* n
shows *defer-lift-invariance* $(m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

lemma *par-comp-rej-card*:
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**
 $c :: \text{nat}$
assumes
compatible: *disjoint-compatibility* $m\ n$ **and**
prof: *profile* $V\ A\ p$ **and**
fin-A: *finite* A **and**
reject-sum: $\text{card } (\text{reject } m\ V\ A\ p) + \text{card } (\text{reject } n\ V\ A\ p) = \text{card } A + c$

shows $\text{card } (\text{reject } (m \parallel_{\uparrow} n) \vee A \ p) = c$
 $\langle \text{proof} \rangle$

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**
 $n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
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)*
 $\langle \text{proof} \rangle$
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* :: $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **where**
 $\text{elector } m = (m \triangleright \text{elect-module})$

6.7.2 Auxiliary Lemmas

lemma *elector-seqcomp-assoc*:
fixes

```

    a :: ('a, 'v, 'a Result) Electoral-Module and
    b :: ('a, 'v, 'a Result) Electoral-Module
  shows (a ▷ (elector b)) = (elector (a ▷ b))
  ⟨proof⟩

```

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)
  ⟨proof⟩

```

```

lemma voters-determine-elector:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes voters-determine-election m
  shows voters-determine-election (elector m)
  ⟨proof⟩

```

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⟩

```

6.7.5 Composition Rule

If m is defer-Condorcet-consistent, then $\text{elector}(m)$ is Condorcet consistent.

```

lemma dcc-imp-cc-elector:
  fixes m :: ('a, 'v, 'a Result) Electoral-Module
  assumes defer-condorcet-consistency m
  shows condorcet-consistency (elector m)
  ⟨proof⟩

```

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
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module **where**

iter m =
 (let *t* = defer-equal-condition 1 in
 (*m* \circlearrowleft_t))

abbreviation *defer-one-loop* :: ('a, 'v, 'a Result) Electoral-Module
 \Rightarrow ('a, 'v, 'a Result) Electoral-Module ($\circlearrowleft_{\exists!d}$ 50) **where**
m $\circlearrowleft_{\exists!d} \equiv$ *iter m*

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

end

Chapter 7

Voting Rules

7.1 Plurality Rule

```
theory Plurality-Rule
  imports Compositional-Structures/Basic-Modules/Plurality-Module
           Compositional-Structures/Revision-Composition
           Compositional-Structures/Elect-Composition
begin
```

This is a definition of the plurality voting rule as elimination module as well as directly. In the former one, the max operator of the set of the scores of all alternatives is evaluated and is used as the threshold value.

7.1.1 Definition

```
fun plurality-rule :: ('a, 'v, 'a Result) Electoral-Module where
  plurality-rule V A p = elector plurality V A p
```

```
fun plurality-rule' :: ('a, 'v, 'a Result) Electoral-Module where
  plurality-rule' V A p =
    ({a ∈ 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⟩
```

```
lemma plurality-elim-equiv:
  fixes
    A :: 'a set and
```


$V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
assumes
 $A \neq \{\}$ **and**
 $\text{finite } A$ **and**
 $\text{profile } V \ A \ p$
shows $\text{plurality } V \ A \ p = (\text{plurality-rule}'\downarrow) \ V \ A \ p$
 $\langle \text{proof} \rangle$

7.1.2 Soundness

theorem $\text{plurality-rule-sound}[\text{simp}]$: $\text{SCF-result.electoral-module plurality-rule}$
 $\langle \text{proof} \rangle$

theorem $\text{plurality-rule}'\text{-sound}[\text{simp}]$: $\text{SCF-result.electoral-module plurality-rule}'$
 $\langle \text{proof} \rangle$

lemma $\text{voters-determine-plurality-rule}$: $\text{voters-determine-election plurality-rule}$
 $\langle \text{proof} \rangle$

7.1.3 Electing

lemma $\text{plurality-rule-elect-non-empty}$:

fixes

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

assumes

$A\text{-non-empty}$: $A \neq \{\}$ **and**
 $\text{prof-}A$: $\text{profile } V \ A \ p$ **and**
 $\text{fin-}A$: $\text{finite } A$

shows $\text{elect plurality-rule } V \ A \ p \neq \{\}$
 $\langle \text{proof} \rangle$

The plurality module is electing.

theorem $\text{plurality-rule-electing}[\text{simp}]$: $\text{electing plurality-rule}$
 $\langle \text{proof} \rangle$

7.1.4 Property

lemma $\text{plurality-rule-inv-mono-eq}$:

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

$\text{elect-}a$: $a \in \text{elect plurality-rule } V \ A \ p$ **and**
 $\text{lift-}a$: $\text{lifted } V \ A \ p \ q \ a$

shows *elect plurality-rule* $V A q = \text{elect plurality-rule } V A p$
 $\vee \text{elect plurality-rule } V A q = \{a\}$
 $\langle \text{proof} \rangle$

The plurality rule is invariant-monotone.

theorem *plurality-rule-inv-mono[simp]: invariant-monotonicity plurality-rule*
 $\langle \text{proof} \rangle$

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 $V A p = \text{elector borda } V A p$

fun *borda-rule_R* :: (*'a*, *'v::wellorder*, *'a Result*) *Electoral-Module* **where**
borda-rule_R $V A p = \text{swap-}\mathcal{R} \text{ unanimity } V A p$

7.2.2 Soundness

theorem *borda-rule-sound: SCF-result.electoral-module borda-rule*
 $\langle \text{proof} \rangle$

theorem *borda-rule_R-sound: SCF-result.electoral-module borda-rule_R*
 $\langle \text{proof} \rangle$

7.2.3 Anonymity Property

theorem *borda-rule_R-anonymous: SCF-result.anonymity borda-rule_R*
 $\langle \text{proof} \rangle$

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
   $\langle$ proof $\rangle$ 

```

```

theorem condorcet'-rule-sound: SCF-result.electoral-module condorcet'
   $\langle$ proof $\rangle$ 

```

```

theorem pairwise-majority-rule'-sound: SCF-result.electoral-module pairwise-majority-rule'
   $\langle$ proof $\rangle$ 

```

7.3.3 Condorcet Consistency Property

```

theorem condorcet-condorcet: condorcet-consistency pairwise-majority-rule
   $\langle$ proof $\rangle$ 

```

```

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*
<proof>

7.4.3 Condorcet Consistency Property

theorem *copeland-condorcet: condorcet-consistency copeland-rule*
<proof>

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*
<proof>

7.5.3 Condorcet Consistency Property

theorem *minimax-condorcet: condorcet-consistency minimax-rule*
<proof>
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*
<proof>

theorem *blacks-rule-sound: SCF-result.electoral-module blacks-rule*
<proof>

7.6.3 Condorcet Consistency Property

theorem *black-is-dcc: defer-condorcet-consistency black*
<proof>

theorem *black-condorcet: condorcet-consistency blacks-rule*
<proof>
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
  <proof>
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)  $\odot_{\exists!d}$ ) V A p

```

7.8.2 Soundness

theorem *classic-nanson-rule-sound: SCF-result.electoral-module classic-nanson-rule*
 <proof>
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*
 <proof>
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*) $\cup_{\exists !d}$)) *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>

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>

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>

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*
<proof>

7.11.3 Anonymity Property

theorem *kemeny-rule-anonymous: SCF-result.anonymity kemeny-rule*
<proof>

7.11.4 Neutrality Property

lemma *swap-dist-neutral: distance-neutrality valid-elections*
(votewise-distance swap l-one)
<proof>

theorem *kemeny-rule-neutral: SCF-properties.neutrality valid-elections kemeny-rule*
<proof>

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.