

Verified Construction of Fair Voting Rules

Michael Kirsten

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

November 9, 2024

Abstract

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

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

Contents

1	Social-Choice Types	9
1.1	Auxiliary Lemmas	9
1.2	Preference Relation	10
1.2.1	Definition	10
1.2.2	Ranking	11
1.2.3	Limited Preference	11
1.2.4	Auxiliary Lemmas	14
1.2.5	Lifting Property	17
1.3	Norm	19
1.3.1	Definition	20
1.3.2	Auxiliary Lemmas	20
1.3.3	Common Norms	20
1.3.4	Properties	20
1.3.5	Theorems	20
1.4	Electoral Result	21
1.4.1	Auxiliary Functions	21
1.4.2	Definition	21
1.5	Preference Profile	22
1.5.1	Definition	22
1.5.2	Vote Count	23
1.5.3	Voter Permutations	24
1.5.4	List Representation	25
1.5.5	Preference Counts	26
1.5.6	Condorcet Winner	29
1.5.7	Limited Profile	30
1.5.8	Lifting Property	30
1.6	Social Choice Result	31
1.6.1	Definition	32
1.6.2	Auxiliary Lemmas	32
1.7	Social Welfare Result	33
1.8	Electoral Result Types	33
1.9	Symmetry Properties of Functions	34
1.9.1	Functions	34

1.9.2	Relations for Symmetry Constructions	34
1.9.3	Invariance and Equivariance	35
1.9.4	Auxiliary Lemmas	35
1.9.5	Rewrite Rules	35
1.9.6	Group Actions	38
1.9.7	Invariance and Equivariance	38
1.9.8	Function Composition	40
1.10	Symmetry Properties of Voting Rules	41
1.10.1	Definitions	42
1.10.2	Auxiliary Lemmas	43
1.10.3	Anonymity Lemmas	45
1.10.4	Neutrality Lemmas	46
1.10.5	Homogeneity Lemmas	48
1.10.6	Reversal Symmetry Lemmas	49
1.11	Result-Dependent Voting Rule Properties	49
1.11.1	Property Definitions	50
1.11.2	Interpretations	50
2	Refined Types	51
2.1	Preference List	51
2.1.1	Well-Formedness	51
2.1.2	Auxiliary Lemmas About Lists	51
2.1.3	Ranking	52
2.1.4	Definition	53
2.1.5	Limited Preference	54
2.1.6	Auxiliary Definitions	55
2.1.7	Auxiliary Lemmas	55
2.1.8	First Occurrence Indices	57
2.2	Preference (List) Profile	58
2.2.1	Definition	58
2.2.2	Refinement Proof	58
2.3	Ordered Relation Type	58
2.4	Alternative Election Type	59
3	Quotient Rules	61
3.1	Quotients of Equivalence Relations	61
3.1.1	Definitions	61
3.1.2	Well-Definedness	61
3.1.3	Equivalence Relations	62
3.2	Quotients of Election Set Equivalences	63
3.2.1	Auxiliary Lemmas	63
3.2.2	Anonymity Quotient: Grid	64
3.2.3	Homogeneity Quotient: Simplex	64

4	Component Types	67
4.1	Distance	67
4.1.1	Definition	67
4.1.2	Conditions	68
4.1.3	Standard-Distance Property	68
4.1.4	Auxiliary Lemmas	68
4.1.5	Swap Distance	69
4.1.6	Spearman Distance	70
4.1.7	Properties	70
4.2	Votewise Distance	72
4.2.1	Definition	72
4.2.2	Inference Rules	72
4.3	Consensus	74
4.3.1	Definition	74
4.3.2	Consensus Conditions	74
4.3.3	Properties	75
4.3.4	Auxiliary Lemmas	75
4.3.5	Theorems	76
4.4	Electoral Module	77
4.4.1	Definition	77
4.4.2	Auxiliary Definitions	78
4.4.3	Properties	78
4.4.4	Social-Welfare Properties	80
4.4.5	Social-Choice Modules	80
4.4.6	Equivalence Definitions	81
4.4.7	Auxiliary Lemmas	82
4.4.8	Non-Blocking	87
4.4.9	Electing	87
4.4.10	Properties	87
4.4.11	Inference Rules	90
4.4.12	Social-Choice Properties	91
4.5	Electoral Module on Election Quotients	92
4.6	Evaluation Function	92
4.6.1	Definition	93
4.6.2	Property	93
4.6.3	Theorems	93
4.7	Elimination Module	94
4.7.1	General Definitions	94
4.7.2	Social-Choice Definitions	95
4.7.3	Social-Choice Eliminators	95
4.7.4	Soundness	95
4.7.5	Independence of Non-Voters	96
4.7.6	Non-Blocking	97
4.7.7	Non-Electing	98

4.7.8	Inference Rules	99
4.8	Aggregator	100
4.8.1	Definition	100
4.8.2	Properties	100
4.9	Maximum Aggregator	101
4.9.1	Definition	101
4.9.2	Auxiliary Lemma	101
4.9.3	Soundness	101
4.9.4	Properties	101
4.10	Termination Condition	102
4.11	Defer Equal Condition	102
5	Basic Modules	103
5.1	Defer Module	103
5.1.1	Definition	103
5.1.2	Soundness	103
5.1.3	Properties	103
5.2	Elect-First Module	103
5.2.1	Definition	104
5.2.2	Soundness	104
5.3	Consensus Class	104
5.3.1	Definition	104
5.3.2	Consensus Choice	105
5.3.3	Auxiliary Lemmas	105
5.3.4	Consensus Rules	106
5.3.5	Properties	106
5.3.6	Inference Rules	107
5.3.7	Theorems	108
5.4	Distance Rationalization	109
5.4.1	Definitions	109
5.4.2	Standard Definitions	109
5.4.3	Auxiliary Lemmas	110
5.4.4	Soundness	111
5.4.5	Inference Rules	112
5.5	Votewise Distance Rationalization	112
5.5.1	Common Rationalizations	112
5.5.2	Theorems	113
5.5.3	Equivalence Lemmas	113
5.6	Symmetry in Distance-Rationalizable Rules	113
5.6.1	Minimizer Function	113
5.6.2	Minimizer Translation	116
5.6.3	Inference Rules	118
5.6.4	Properties	119
5.7	Distance Rationalization on Election Quotients	119

5.7.1	Distances	119
5.7.2	Consensus and Results	122
5.7.3	Distance Rationalization	124
5.8	Code Generation Interpretations for Results and Properties	125
5.8.1	Code Lemmas	125
5.8.2	Interpretation Declarations and Constants	127
5.9	Drop Module	127
5.9.1	Definition	127
5.9.2	Soundness	128
5.9.3	Non-Electing	128
5.9.4	Properties	128
5.10	Pass Module	129
5.10.1	Definition	129
5.10.2	Soundness	129
5.10.3	Non-Blocking	129
5.10.4	Non-Electing	130
5.10.5	Properties	130
5.11	Elect Module	131
5.11.1	Definition	131
5.11.2	Soundness	131
5.11.3	Electing	131
5.12	Plurality Module	131
5.12.1	Definition	132
5.12.2	Soundness	132
5.12.3	Non-Blocking	133
5.12.4	Non-Electing	133
5.12.5	Property	133
5.13	Borda Module	133
5.13.1	Definition	134
5.13.2	Soundness	134
5.13.3	Non-Blocking	134
5.13.4	Non-Electing	134
5.14	Condorcet Module	134
5.14.1	Definition	135
5.14.2	Soundness	135
5.14.3	Property	135
5.15	Copeland Module	135
5.15.1	Definition	135
5.15.2	Soundness	136
5.15.3	Lemmas	136
5.15.4	Property	137
5.16	Minimax Module	137
5.16.1	Definition	137
5.16.2	Soundness	138

5.16.3	Lemma	138
5.16.4	Property	138
6	Compositional Structures	139
6.1	Drop- and Pass-Compatibility	139
6.2	Revision Composition	140
6.2.1	Definition	140
6.2.2	Soundness	140
6.2.3	Composition Rules	140
6.3	Sequential Composition	141
6.3.1	Definition	141
6.3.2	Soundness	142
6.3.3	Lemmas	143
6.3.4	Composition Rules	145
6.4	Parallel Composition	148
6.4.1	Definition	148
6.4.2	Soundness	148
6.4.3	Composition Rule	148
6.5	Loop Composition	149
6.5.1	Definition	149
6.5.2	Soundness	151
6.5.3	Lemmas	151
6.5.4	Composition Rules	154
6.6	Maximum Parallel Composition	155
6.6.1	Definition	155
6.6.2	Soundness	156
6.6.3	Lemmas	156
6.6.4	Composition Rules	159
6.7	Elect Composition	160
6.7.1	Definition	160
6.7.2	Auxiliary Lemmas	160
6.7.3	Soundness	160
6.7.4	Electing	161
6.7.5	Composition Rule	161
6.8	Defer-One Loop Composition	161
7	Voting Rules	163
7.1	Plurality Rule	163
7.1.1	Definition	163
7.1.2	Soundness	164
7.1.3	Electing	164
7.1.4	Properties	164
7.2	Borda Rule	165
7.2.1	Definition	165

7.2.2	Soundness	165
7.2.3	Anonymity	165
7.3	Pairwise Majority Rule	166
7.3.1	Definition	166
7.3.2	Soundness	166
7.3.3	Condorcet Consistency	166
7.4	Copeland Rule	166
7.4.1	Definition	167
7.4.2	Soundness	167
7.4.3	Condorcet Consistency	167
7.5	Minimax Rule	167
7.5.1	Definition	167
7.5.2	Soundness	167
7.5.3	Condorcet Consistency	168
7.6	Black's Rule	168
7.6.1	Definition	168
7.6.2	Soundness	168
7.6.3	Condorcet Consistency	168
7.7	Nanson-Baldwin Rule	169
7.7.1	Definition	169
7.7.2	Soundness	169
7.8	Classic Nanson Rule	169
7.8.1	Definition	169
7.8.2	Soundness	170
7.9	Schwartz Rule	170
7.9.1	Definition	170
7.9.2	Soundness	170
7.10	Sequential Majority Comparison	170
7.10.1	Definition	171
7.10.2	Soundness	171
7.10.3	Electing	171
7.10.4	(Weak) Monotonicity	171
7.11	Kemeny Rule	172
7.11.1	Definition	172
7.11.2	Soundness	172
7.11.3	Anonymity	172
7.11.4	Neutrality	172

Chapter 1

Social-Choice Types

1.1 Auxiliary Lemmas

```
theory Auxiliary-Lemmas
  imports Main
begin
```

```
lemma sum-comp:
```

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

```
lemma the-inv-comp:
```

```
  fixes
     $f :: 'y \Rightarrow 'z$  and
     $g :: 'x \Rightarrow 'y$  and
     $s :: 'x \text{ set}$  and
     $t :: 'y \text{ set}$  and
     $u :: 'z \text{ set}$  and
     $x :: 'z$ 
  assumes
    bij-betw  $f$   $t$   $u$  and
    bij-betw  $g$   $s$   $t$  and
     $x \in u$ 
  shows the-inv-into  $s$   $(f \circ g) \ x = ((\text{the-inv-into } s \ g) \circ (\text{the-inv-into } t \ f)) \ x$ 
  <proof>
```

```
end
```

1.2 Preference Relation

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

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

1.2.1 Definition

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

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

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

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

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

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

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

```
lemma lin-imp-trans:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation
  assumes linear-order-on A r
  shows trans r
   $\langle$ proof $\rangle$ 
```

1.2.2 Ranking

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

lemma *rank-gt-zero*:

fixes
 r :: 'a Preference-Relation **and**
 a :: 'a
assumes
 refl: $a \preceq_r a$ **and**
 fin: finite *r*
shows $\text{rank } r \ a \geq 1$
 <proof>

1.2.3 Limited Preference

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

lemma *limited-dest*:

fixes
 A :: 'a set **and**
 r :: 'a Preference-Relation **and**
 a *b* :: 'a
assumes
 $a \preceq_r b$ **and**
 limited *A* *r*
shows $a \in A \wedge b \in A$
 <proof>

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

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

lemma *connex-imp-refl*:

fixes
 A :: 'a set **and**
 r :: 'a Preference-Relation
assumes *connex* *A* *r*
shows *refl-on* *A* *r*
 <proof>

lemma *lin-ord-imp-connex*:

fixes
 A :: 'a set **and**
 r :: 'a Preference-Relation
assumes *linear-order-on* *A* *r*

shows *connex* A r
 $\langle proof \rangle$

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
 $\langle proof \rangle$

lemma *limit-to-limits*:
fixes
 $A :: 'a$ set **and**
 $r :: 'a$ Preference-Relation
shows *limited* A (*limit* A r)
 $\langle proof \rangle$

lemma *limit-presv-connex*:
fixes
 A $B :: 'a$ set **and**
 $r :: 'a$ Preference-Relation
assumes
connex: *connex* B r **and**
subset: $A \subseteq B$
shows *connex* A (*limit* A r)
 $\langle proof \rangle$

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

lemma *limit-presv-lin-ord*:
fixes

$A \ B :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$
assumes
 $\text{linear-order-on } B \ r$ **and**
 $A \subseteq B$
shows $\text{linear-order-on } A \ (\text{limit } A \ r)$
 $\langle \text{proof} \rangle$

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

lemma *limit-rel-presv-prefs*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$ **and**
 $a \ b :: 'a$
assumes $(a, b) \in \text{limit } A \ r$
shows $a \preceq_r b$
 $\langle \text{proof} \rangle$

lemma *limit-trans*:
fixes
 $A \ B :: 'a \text{ set}$ **and**
 $r :: 'a \text{ Preference-Relation}$
assumes $A \subseteq B$
shows $\text{limit } A \ r = \text{limit } A \ (\text{limit } B \ r)$
 $\langle \text{proof} \rangle$

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 $\forall \ r. \text{linear-order-on } \{a\} \ r \longrightarrow r = \{(a, a)\}$
 $\langle \text{proof} \rangle$

1.2.4 Auxiliary Lemmas

lemma *above-trans*:

fixes

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

assumes

$\text{trans } r$ **and**
 $(a, b) \in r$

shows $\text{above } r \ b \subseteq \text{above } r \ a$
<proof>

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 $a \in \text{above } r \ a$
<proof>

lemma *above-subset-geq-one*:

fixes

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

assumes

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

shows $\text{above } r \ a = \{a\}$
<proof>

lemma *above-connex*:

fixes

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

assumes

$\text{connex } A \ r$ **and**
 $a \in A$

shows $a \in \text{above } r \ a$
<proof>

lemma *pref-imp-in-above*:

fixes

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

$a \ b :: 'a$
shows $(a \preceq_r b) = (b \in \text{above } r \ a)$
 $\langle \text{proof} \rangle$

lemma *limit-presv-above*:

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

lemma *limit-rel-presv-above*:

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

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

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**
 $a \ b :: 'a$
assumes
 $\text{lin-ord: linear-order-on } A \ r$ **and**
 $\text{fin-A: finite } A$ **and**
 $\text{a-in-A: } a \in A$ **and**
 $\text{b-in-A: } b \in A$ **and**
 $\text{a-neq-b: } a \neq b$
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 $\text{above } (\text{limit } A \ r) \ a \subseteq A$

$\langle \text{proof} \rangle$

1.2.5 Lifting Property

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

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

lemma *trivial-equiv-rel*:

fixes

A :: 'a set **and**

r :: 'a Preference-Relation

assumes linear-order-on A r

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

$\langle \text{proof} \rangle$

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

fixes

A :: 'a set **and**

r r' :: 'a Preference-Relation **and**

a :: 'a

assumes *lifted* A r r' a

shows *equiv-rel-except-a* A r r' a

$\langle \text{proof} \rangle$

lemma *lifted-imp-switched*:

fixes

A :: 'a set **and**

r r' :: 'a Preference-Relation **and**

a :: 'a

assumes *lifted* A r r' a

shows $\forall a' \in A - \{a\}. \neg (a' \preceq_r a \wedge a \preceq_{r'} a')$

$\langle \text{proof} \rangle$

lemma *lifted-mono*:

fixes

A :: 'a set **and**

r r' :: 'a Preference-Relation **and**

a a' :: 'a

assumes

lifted: *lifted* A r r' a **and**

a'-pref-a: $a' \preceq_r a$

shows $a' \preceq_{r'} a$
 $\langle proof \rangle$

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

lemma *lifted-above-mono*:
fixes
 $A :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a \ a' :: 'a$
assumes
lifted-a: *lifted* $A \ r \ r' \ a$ **and**
a'-in-A-sub-a: $a' \in A - \{a\}$
shows $\text{above } r \ a' \subseteq \text{above } r' \ a' \cup \{a\}$
 $\langle proof \rangle$

lemma *limit-lifted-imp-eq-or-lifted*:
fixes
 $A \ A' :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes
lifted: *lifted* $A' \ r \ r' \ a$ **and**
subset: $A \subseteq A'$
shows $\text{limit } A \ r = \text{limit } A \ r' \vee \text{lifted } A \ (\text{limit } A \ r) \ (\text{limit } A \ r') \ a$
 $\langle proof \rangle$

lemma *negl-diff-imp-eq-limit*:
fixes
 $A \ A' :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **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 $\text{limit } A \ r = \text{limit } A \ r'$
 $\langle proof \rangle$

theorem *lifted-above-winner-alts*:
fixes
 $A :: 'a \text{ set}$ **and**

$r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a \ a' :: 'a$
assumes
 $\text{lifted-}a$: $\text{lifted } A \ r \ r' \ a$ **and**
 $a'\text{-above-}a'$: $\text{above } r \ a' = \{a'\}$ **and**
 $\text{fin-}A$: $\text{finite } A$
shows $\text{above } r' \ a' = \{a'\} \vee \text{above } r' \ a = \{a\}$
 $\langle \text{proof} \rangle$

theorem $\text{lifted-above-winner-single}$:
fixes
 $A :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a :: 'a$
assumes
 $\text{lifted } A \ r \ r' \ a$ **and**
 $\text{above } r \ a = \{a\}$ **and**
 $\text{finite } A$
shows $\text{above } r' \ a = \{a\}$
 $\langle \text{proof} \rangle$

theorem $\text{lifted-above-winner-other}$:
fixes
 $A :: 'a \text{ set}$ **and**
 $r \ r' :: 'a \text{ Preference-Relation}$ **and**
 $a \ a' :: 'a$
assumes
 $\text{lifted-}a$: $\text{lifted } A \ r \ r' \ a$ **and**
 $a'\text{-above-}a'$: $\text{above } r' \ a' = \{a'\}$ **and**
 $\text{fin-}A$: $\text{finite } A$ **and**
 $a\text{-not-}a'$: $a \neq a'$
shows $\text{above } r \ a' = \{a'\}$
 $\langle \text{proof} \rangle$

end

1.3 Norm

theory Norm
imports $\text{HOL-Library.Extended-Real}$
 $\text{HOL-Combinatorics.List-Permutation}$
 Auxiliary-Lemmas
begin

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

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

1.3.1 Definition

type-synonym $Norm = \text{ereal list} \Rightarrow \text{ereal}$

definition $norm :: Norm \Rightarrow \text{bool}$ **where**

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

1.3.2 Auxiliary Lemmas

lemma *sum-over-image-of-bijection*:

fixes

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

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

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

$g :: 'a \Rightarrow \text{ereal}$

assumes *bij-betw* $f\ A\ A'$

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

<proof>

1.3.3 Common Norms

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

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

1.3.4 Properties

definition *symmetry* :: $Norm \Rightarrow \text{bool}$ **where**

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

1.3.5 Theorems

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

<proof>

end

1.4 Electoral Result

```
theory Result
  imports Main
begin
```

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

1.4.1 Auxiliary Functions

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

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

```
fun disjoint3 :: 'r Result  $\Rightarrow$  bool where
  disjoint3 (e, r, d) =
    ((e  $\cap$  r = {})  $\wedge$ 
     (e  $\cap$  d = {})  $\wedge$ 
     (r  $\cap$  d = {}))
```

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

1.4.2 Definition

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

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

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

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

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

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

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

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

end

1.5 Preference Profile

theory *Profile*
imports *Preference-Relation*
Auxiliary-Lemmas
HOL-Library.Extended-Nat
HOL-Combinatorics.Permutations
begin

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

1.5.1 Definition

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

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

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

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

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

$\text{voters-}\mathcal{E} \ E = \text{fst} (\text{snd } E)$

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

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

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

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

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

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

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

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

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

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

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

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

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

1.5.2 Vote Count

lemma $\text{vote-count-sum}:$
fixes $E :: ('a, 'v) \text{ Election}$
assumes
 $\text{finite } (\text{voters-}\mathcal{E} \ E) \text{ and}$

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

1.5.3 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**

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

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

$\langle \text{proof} \rangle$

lemma rename-prof :

fixes

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

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

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

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

assumes

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

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

$\text{bij } \pi$

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

$\text{finite } V$ **and**

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

$\text{bij } \pi$

shows $\text{finite } V'$

$\langle \text{proof} \rangle$


```

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

```

```

lemma rename-inj:
  fixes  $\pi :: 'v \Rightarrow 'v$ 
  assumes bij  $\pi$ 
  shows inj  $(\text{rename } \pi)$ 
<proof>

```

```

lemma rename-surj:
  fixes  $\pi :: 'v \Rightarrow 'v$ 
  assumes bij  $\pi$ 
  shows
     $\text{rename } \pi \text{ ' well-formed-elections} = \text{well-formed-elections}$  and
     $\text{rename } \pi \text{ ' finite-elections} = \text{finite-elections}$ 
<proof>

```

1.5.4 List Representation

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

```

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

```

```

lemma map-helper:
  fixes
     $f :: 'x \Rightarrow 'y \Rightarrow 'z$  and
     $g :: 'x \Rightarrow 'x$  and
     $h :: 'y \Rightarrow 'y$  and
     $l :: 'x \text{ list}$  and
     $l' :: 'y \text{ list}$ 
  shows  $\text{map2 } f (\text{map } g \ l) (\text{map } h \ l') = \text{map2 } (\lambda \ x \ y. f \ (g \ x) \ (h \ y)) \ l \ l'$ 
<proof>

```

```

lemma to-list-simp:
  fixes
     $i :: \text{nat}$  and
     $V :: 'v::\text{linorder set}$  and

```

$p :: ('a, 'v) \text{ Profile}$
assumes $i < \text{card } V$
shows $(\text{to-list } V \ p)!i = p \ ((\text{sorted-list-of-set } V)!i)$
 $\langle \text{proof} \rangle$

lemma *to-list-comp*:
fixes
 $V :: 'v::\text{linorder set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $f :: 'a \text{ rel} \Rightarrow 'a \text{ rel}$
shows $\text{to-list } V \ (f \circ p) = \text{map } f \ (\text{to-list } V \ p)$
 $\langle \text{proof} \rangle$

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

1.5.5 Preference Counts

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

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

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

lemma pref-count-voter-set-card:

fixes

V :: 'v set **and**

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

a b :: 'a

assumes finite V

shows prefer-count V p a b \leq card V

\langle proof \rangle

lemma set-compr:

fixes

A :: 'a set **and**

f :: 'a \Rightarrow 'a set

shows {f x | x. x \in A} = f ' A

\langle proof \rangle

lemma pref-count-set-compr:

fixes

A :: 'a set **and**

V :: 'v set **and**

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

a :: 'a

shows {prefer-count V p a a' | a'. a' \in A - {a}} =
 (prefer-count V p a) ' (A - {a})

\langle proof \rangle

lemma pref-count:

fixes

A :: 'a set **and**

V :: 'v set **and**

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

a b :: 'a

assumes

prof: profile V A p **and**

fn: finite V **and**

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

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

neg: a \neq b

shows prefer-count V p a b = card V - (prefer-count V p b a)

\langle proof \rangle

lemma *pref-count-sym*:
fixes
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $V :: 'v \text{ set}$ **and**
 $a \ b \ c :: 'a$
assumes
 $\text{pref-count-ineq: } \text{prefer-count } V \ p \ a \ c \geq \text{prefer-count } V \ p \ c \ b$ **and**
 $\text{prof: profile } V \ A \ p$ **and**
 $a\text{-in-}A: a \in A$ **and**
 $b\text{-in-}A: b \in A$ **and**
 $c\text{-in-}A: c \in A$ **and**
 $a\text{-neg-}c: a \neq c$ **and**
 $c\text{-neg-}b: c \neq b$
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**
 $a \ b :: 'a$
assumes $V = \{\}$
shows $\text{prefer-count } V \ p \ a \ b = 0$
 $\langle \text{proof} \rangle$

fun *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**
 $a \ b :: 'a$ **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**
 $a \ b :: 'a$ **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  $\neg \text{wins } V \ a \ p \ a$ 
 $\langle \text{proof} \rangle$ 

```

1.5.6 Condorcet Winner

```

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

```

lemma *cond-winner-unique-eq*:

```

fixes
   $V :: 'v \text{ set}$  and
   $A :: 'a \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a \ b :: 'a$ 
assumes
  condorcet-winner  $V \ A \ p \ a$  and
  condorcet-winner  $V \ A \ p \ b$ 
shows  $b = a$ 
 $\langle \text{proof} \rangle$ 

```

lemma *cond-winner-unique*:

```

fixes
   $A :: 'a \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a :: 'a$ 
assumes condorcet-winner  $V \ A \ p \ a$ 
shows  $\{a' \in A. \text{condorcet-winner } V \ A \ p \ a'\} = \{a\}$ 
 $\langle \text{proof} \rangle$ 

```

lemma *cond-winner-unique'*:

```

fixes
   $V :: 'v \text{ set}$  and
   $A :: 'a \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $a \ b :: 'a$ 
assumes
  condorcet-winner  $V \ A \ p \ a$  and
   $b \neq a$ 
shows  $\neg \text{condorcet-winner } V \ A \ p \ b$ 
 $\langle \text{proof} \rangle$ 

```

1.5.7 Limited Profile

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

fun *limit-profile* :: 'a set \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 B C :: 'a set **and**

p :: ('a, 'v) Profile

assumes

B \subseteq A **and**

C \subseteq B

shows *limit-profile* C p = *limit-profile* C (*limit-profile* B p)

<proof>

lemma *limit-profile-sound*:

fixes

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

V :: 'v set **and**

p :: ('a, 'v) Profile

assumes

profile V B p **and**

A \subseteq B

shows *profile* V A (*limit-profile* A p)

<proof>

1.5.8 Lifting Property

definition *equiv-prof-except-a* :: 'v set \Rightarrow 'a set \Rightarrow ('a, 'v) Profile \Rightarrow
('a, 'v) Profile \Rightarrow 'a \Rightarrow bool **where**
equiv-prof-except-a V A p p' a \equiv
profile V A p \wedge *profile* V A p' \wedge a \in A \wedge
(\forall v \in V. *equiv-rel-except-a* A (p v) (p' v) a)

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

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

lifted V A p p' a \equiv

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

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

\wedge (\exists v \in V. *Preference-Relation.lifted* A (p v) (p' v) a)

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

fixes

A :: 'a set **and**

```

    V :: 'v set and
    p p' :: ('a, 'v) Profile and
    a :: 'a
    assumes lifted V A p p' a
    shows equiv-prof-except-a V A p p' a
  <proof>

```

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

```

  fixes
    A A' :: 'a set and
    V :: 'v set and
    p p' :: ('a, 'v) Profile and
    a :: 'a
  assumes
    change: equiv-prof-except-a V A' p q a and
    subset: A ⊆ A' and
    not-in-A: a ∉ A
  shows ∀ v ∈ V. (limit-profile A p) v = (limit-profile A q) v
  — With the current definitions of equiv-prof-except-a and limit-prof, we can only
  conclude that the limited profiles coincide on the given voter set, since limit-prof
  may change the profiles everywhere, while equiv-prof-except-a only makes state-
  ments about the voter set.
  <proof>

```

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

```

  fixes
    A A' :: 'a set and
    V :: 'v set and
    p p' :: ('a, 'v) Profile and
    a :: 'a
  assumes
    lifted-a: lifted V A' p p' a and
    subset: A ⊆ A'
  shows (∀ v ∈ V. limit-profile A p v = limit-profile A p' v)
    ∨ lifted V A (limit-profile A p) (limit-profile A p') a
  <proof>

```

end

1.6 Social Choice Result

theory *Social-Choice-Result*

imports *Result*

begin

1.6.1 Definition

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

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

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

1.6.2 Auxiliary Lemmas

lemma *result-imp-rej*:
 fixes A e r d :: 'a set
 assumes *well-formed-SCF* A (e, r, d)
 shows A - (e \cup d) = r
 ⟨proof⟩

lemma *result-count*:
 fixes A e r d :: 'a set
 assumes
 wf-result: *well-formed-SCF* A (e, r, d) **and**
 fin-A: finite A
 shows card A = card e + card r + card d
 ⟨proof⟩

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 set **and**
 r :: 'a Result
 assumes *well-formed-SCF* A r
 shows reject-r r \subseteq A
 ⟨proof⟩

end

1.7 Social Welfare Result

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

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

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

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

end

1.8 Electoral Result Types

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

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

$\langle ML \rangle$

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-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
   $\lambda A r. \text{set-equals-partition } (\text{Pow } A) r \wedge \text{disjoint3 } r$ 
   $\lambda A rs. \{r \cap A \mid r. r \in rs\}$ 
 $\langle \text{proof} \rangle$ 
```

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

```
global-interpretation SWF-result: result well-formed-SWF limit-SWF
 $\langle \text{proof} \rangle$ 
```

$\langle ML \rangle$

end

1.9 Symmetry Properties of Functions

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

1.9.1 Functions

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

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

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

1.9.2 Relations for Symmetry Constructions

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

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

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

fun product :: 'x rel  $\Rightarrow$  ('x * 'x) rel where
```

$product\ r = \{(p, p'). (fst\ p, fst\ p') \in r \wedge (snd\ p, snd\ p') \in r\}$

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

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

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

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

1.9.3 Invariance and Equivariance

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

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

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

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

1.9.4 Auxiliary Lemmas

lemma *un-left-inv-singleton-set-system*: $\bigcup \circ singleton-set-system = id$
<proof>

lemma *preimg-comp*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $g :: 'x \Rightarrow 'x$ **and**
 $s :: 'x\ set$ **and**
 $x :: 'y$
shows *preimg* f (g ' s) x = g ' *preimg* (f \circ g) s x
<proof>

1.9.5 Rewrite Rules

theorem *rewrite-invar-as-equivar*:

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

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

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

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

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
 $f\ g :: 'a \Rightarrow 'a$ **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**
 $s\ t :: 'a \text{ set}$ **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
 $\varphi\ \psi :: ('a, 'b) \text{ binary-fun}$ **and**
 $s :: 'a \text{ set}$ **and**
 $t\ u :: 'b \text{ set}$
assumes
 $u \subseteq t$ **and**
 $\forall x \in s. \forall y \in u. \psi\ x\ y = \varphi\ x\ y$
shows $\text{action-induced-rel } s\ u\ \psi = \text{restricted-rel } (\text{action-induced-rel } s\ t\ \varphi)\ u\ \text{UNIV}$
 $\langle \text{proof} \rangle$

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

fixes
 $s :: 'x \text{ set}$ **and**
 $t :: 'y \text{ set}$ **and**
 $\varphi\ \psi :: ('x, 'y) \text{ binary-fun}$
assumes $\forall x \in s. \forall y \in t. \varphi\ x\ y = \psi\ x\ y$

shows *action-induced-rel* $s\ t\ \varphi = \text{action-induced-rel}\ s\ t\ \psi$
 ⟨proof⟩

1.9.6 Group Actions

lemma *const-id-is-group-action*:
fixes $m :: 'x\ monoid$
assumes *group* m
shows *group-action* $m\ UNIV\ (\lambda\ x.\ id)$
 ⟨proof⟩

theorem *group-act-induces-set-group-act*:
fixes
 $m :: 'x\ monoid$ **and**
 $s :: 'y\ set$ **and**
 $\varphi :: ('x, 'y)\ binary_fun$
defines $\varphi\text{-img} \equiv (\lambda\ x.\ extensional\ continuation\ (image\ (\varphi\ x))\ (Pow\ s))$
assumes *group-action* $m\ s\ \varphi$
shows *group-action* $m\ (Pow\ s)\ \varphi\text{-img}$
 ⟨proof⟩

1.9.7 Invariance and Equivariance

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

theorem *equivar-generators-imp-equivar-group*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $m :: 'z\ monoid$ **and**
 $s :: 'z\ set$ **and**
 $t :: 'x\ set$ **and**
 $\varphi :: ('z, 'x)\ binary_fun$ **and**
 $\psi :: ('z, 'y)\ binary_fun$
assumes
equivar: *is-symmetry* $f\ (\text{action-induced-equivariance}\ s\ t\ \varphi\ \psi)$ **and**
action- φ : *group-action* $m\ t\ \varphi$ **and**
action- ψ : *group-action* $m\ (f\ ` t)\ \psi$ **and**
 gen : $carrier\ m = generate\ m\ s$
shows *is-symmetry* $f\ (\text{action-induced-equivariance}\ (carrier\ m)\ t\ \varphi\ \psi)$
 ⟨proof⟩

lemma *invar-parameterized-fun*:
fixes
 $f :: 'x \Rightarrow ('x \Rightarrow 'y)$ **and**
 $r :: 'x\ rel$
assumes

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

lemma *invar-under-subset-rel:*

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

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

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

lemma *equivar-under-subset:*

fixes
 $f :: 'x \Rightarrow 'y \text{ and}$
 $s\ t :: 'x\ \text{set and}$
 $\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ \text{set}$
assumes
 $\text{is-symmetry } f \text{ (Equivariance } s\ \tau) \text{ and}$
 $t \subseteq s$
shows $\text{is-symmetry } f \text{ (Equivariance } t\ \tau)$
 $\langle \text{proof} \rangle$

lemma *equivar-under-subset':*

fixes
 $f :: 'x \Rightarrow 'y \text{ and}$
 $s :: 'x\ \text{set and}$
 $\tau\ v :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ \text{set}$
assumes
 $\text{is-symmetry } f \text{ (Equivariance } s\ \tau) \text{ and}$
 $v \subseteq \tau$
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**
 $\mathcal{D}_f s :: 'x \text{ set}$ **and**
 $m :: 'z \text{ monoid}$ **and**
 $\varphi :: ('z, 'x) \text{ binary-fun}$ **and**
 $\psi :: ('z, 'y) \text{ binary-fun}$ **and**
 $x :: 'z$
defines *equivar-prop* \equiv *action-induced-equivariance* (*carrier* m) $\mathcal{D}_f \varphi \psi$
assumes
action- φ : *group-action* $m s \varphi$ **and**
action-res: *group-action* $m \text{ UNIV } \psi$ **and**
dom-in-s: $\mathcal{D}_f \subseteq s$ **and**
closed-domain:
closed-restricted-rel (*action-induced-rel* (*carrier* m) $s \varphi$) $s \mathcal{D}_f$ **and**
equivar-f: *is-symmetry* f *equivar-prop* **and**
group-elem-x: $x \in \text{carrier } m$
shows $\forall y. \text{preimg } f \mathcal{D}_f (\psi x y) = (\varphi x) \text{ ` } (\text{preimg } f \mathcal{D}_f y)$
 $\langle \text{proof} \rangle$

1.9.8 Function Composition

lemma *invar-comp*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $g :: 'y \Rightarrow 'z$ **and**
 $r :: 'x \text{ rel}$
assumes *is-symmetry* f (*Invariance* r)
shows *is-symmetry* $(g \circ f)$ (*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
 $f \text{ ` } s \subseteq t$ **and**
is-symmetry f (*Equivariance* $s \tau$) **and**
is-symmetry g (*Equivariance* $t v$)
shows *is-symmetry* $(g \circ f)$ (*Equivariance* s *transitive-acts*)
 $\langle \text{proof} \rangle$

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
 $f \text{ ' } t \subseteq u$ **and**
 $\forall x \in s. \chi \text{ ' } x \text{ ' } f \text{ ' } t \subseteq u$ **and**
 $\text{is-symmetry } f \text{ (action-induced-equivariance } s \text{ ' } t \text{ ' } \varphi \text{ ' } \chi)$ **and**
 $\text{is-symmetry } g \text{ (action-induced-equivariance } s \text{ ' } u \text{ ' } \chi \text{ ' } \psi)$
shows $\text{is-symmetry } (g \circ f) \text{ (action-induced-equivariance } s \text{ ' } t \text{ ' } \varphi \text{ ' } \psi)$
 $\langle \text{proof} \rangle$

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

lemma *equivar-union-under-image-action*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $s :: 'z$ *set* **and**
 $\varphi :: ('z, 'x)$ *binary-fun*
shows $\text{is-symmetry } \bigcup \text{ (action-induced-equivariance } s \text{ ' } UNIV \text{ (set-action (set-action } \varphi)) \text{ (set-action } \varphi))$
 $\langle \text{proof} \rangle$

end

1.10 Symmetry Properties of Voting Rules

theory *Voting-Symmetry*

```

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

```

1.10.1 Definitions

```

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

```

```

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

```

Anonymity

```

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

```

```

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

```

```

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

```

```

fun  $\psi$ -neutralc :: ('a  $\Rightarrow$  'a, 'a) binary-fun where
   $\psi$ -neutralc  $\pi$  r =  $\pi$  r

```

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

Homogeneity

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

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

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

Reversal Symmetry

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

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

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

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

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

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

1.10.2 Auxiliary Lemmas

fun $n\text{-app} :: \text{nat} \Rightarrow ('x \Rightarrow 'x) \Rightarrow ('x \Rightarrow 'x) \text{ where}$

n-app-id: $n\text{-app } 0 \ f = \text{id} \mid$
n-app-suc: $n\text{-app } (\text{Suc } n) \ f = f \circ n\text{-app } n \ f$

lemma *n-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-app-leaves-set*:

fixes

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

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

$x :: 'x$

assumes

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

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

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

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

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-app-rev*:

fixes

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

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

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

$x \ y :: 'x$

assumes

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

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

n-geq-m: $n \geq m$ **and**

n-app-eq-m-n: $n\text{-app } n \ f \ x = n\text{-app } m \ f \ y$ **and**

n-app-x-in-A: $\forall \ n' < n. \ n\text{-app } n' \ f \ x \in A$ **and**

n-app-y-in-A: $\forall \ m' < m. \ n\text{-app } m' \ f \ y \in A$ **and**

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

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

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

shows $n\text{-app } (n - m) \ f \ x = y$

$\langle \text{proof} \rangle$

lemma *n-app-inv*:

fixes

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

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

lemma *bij-betw-finite-ind-global-bij*:

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

lemma *bij-betw-ext*:

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

1.10.3 Anonymity Lemmas

lemma *anon-rel-vote-count*:

fixes
 $\mathcal{E} :: ('a, 'v) \text{ Election set}$ **and**
 $E \ E' :: ('a, 'v) \text{ Election}$
assumes
 $\text{finite } (\text{voters-}\mathcal{E} \ E)$ **and**
 $(E, E') \in \text{anonymity}_{\mathcal{R}} \ \mathcal{E}$
shows $\text{alternatives-}\mathcal{E} \ E = \text{alternatives-}\mathcal{E} \ E' \wedge E \in \mathcal{E}$
 $\wedge (\forall p. \text{vote-count } p \ E = \text{vote-count } p \ E')$
 $\langle \text{proof} \rangle$

lemma *vote-count-anon-rel*:

fixes

$\mathcal{E} :: ('a, 'v)$ Election set **and**
 $E E' :: ('a, 'v)$ 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 $\pi \ \pi' :: 'v \Rightarrow 'v$
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 $_{\mathcal{G}}$* *well-formed-elections* φ -anon *well-formed-elections*
 $\langle \text{proof} \rangle$

lemma (*in result*) *anonymity*:
is-symmetry $(\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$
(Invariance (anonymity $_{\mathcal{R}}$ well-formed-elections))
 $\langle \text{proof} \rangle$

1.10.4 Neutrality Lemmas

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

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

lemma *rename-subset*:

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

lemma *rel-rename-bij*:

fixes $\pi :: 'a \Rightarrow 'a$
assumes $\text{bij-}\pi$: $\text{bij } \pi$
shows $\text{bij } (\text{rel-rename } \pi)$
 $\langle \text{proof} \rangle$

lemma *alternatives-rename-comp*:

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

lemma *well-formed-elects-closed*:

fixes
 $A \ A' :: 'a \text{ set}$ **and**
 $V \ V' :: 'v \text{ set}$ **and**
 $p \ p' :: ('a, 'v) \text{ Profile}$ **and**
 $\pi :: 'a \Rightarrow 'a$
assumes
 $\text{bij-}\pi$: $\text{bij } \pi$ **and**
 wf-elects : $(A, V, p) \in \text{well-formed-elections}$ **and**
 renamed : $(A', V', p') = \text{alternatives-rename } \pi \ (A, V, p)$
shows $(A', V', p') \in \text{well-formed-elections}$
 $\langle \text{proof} \rangle$

lemma *alternatives-rename-bij*:

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

assumes *bij- π : bij π*
shows *bij-betw (alternatives-rename π) well-formed-elections well-formed-elections*
<proof>

interpretation *φ -neutral-action: group-action neutrality_G well-formed-elections*
 φ -neutral well-formed-elections
<proof>

interpretation *ψ -neutral_c-action: group-action neutrality_G UNIV ψ -neutral_c*
<proof>

interpretation *ψ -neutral_w-action: group-action neutrality_G UNIV ψ -neutral_w*
<proof>

lemma *neutrality-SCF: is-symmetry ($\lambda \mathcal{E}$. limit-SCF (alternatives- \mathcal{E} \mathcal{E}) UNIV)*
(action-induced-equivariance (carrier neutrality_G) well-formed-elections
(φ -neutral well-formed-elections) (set-action ψ -neutral_c))
<proof>

lemma *neutrality-SWF: is-symmetry ($\lambda \mathcal{E}$. limit-SWF (alternatives- \mathcal{E} \mathcal{E}) UNIV)*
(action-induced-equivariance (carrier neutrality_G) well-formed-elections
(φ -neutral well-formed-elections) (set-action ψ -neutral_w))
<proof>

1.10.5 Homogeneity Lemmas

definition *reflp-on' :: 'a set \Rightarrow 'a rel \Rightarrow bool where*
reflp-on' A r \longleftrightarrow reflat-on A ($\lambda x y$. (x, y) \in r)

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

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

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

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

<proof>

1.10.6 Reversal Symmetry Lemmas

lemma *reverse-reverse-id*: $\text{reverse-rel} \circ \text{reverse-rel} = \text{id}$
<proof>

lemma *reverse-rel-limit*:

fixes

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

$r :: 'a \text{ rel}$

shows $\text{reverse-rel} (\text{limit } A \ r) = \text{limit } A (\text{reverse-rel } r)$

<proof>

lemma *reverse-rel-lin-ord*:

fixes

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

$r :: 'a \text{ rel}$

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

shows *linear-order-on* $A (\text{reverse-rel } r)$

<proof>

interpretation *reversal_G-group*: *group reversal_G*

<proof>

interpretation *φ-reverse-action*: *group-action reversal_G well-formed-elections*
φ-reverse well-formed-elections

<proof>

interpretation *ψ-reverse-action*: *group-action reversal_G UNIV ψ-reverse*

<proof>

lemma *reversal-symmetry*: *is-symmetry* $(\lambda \mathcal{E}. \text{limit-SWF} (\text{alternatives-}\mathcal{E} \ \mathcal{E}) \text{UNIV})$
(action-induced-equivariance (carrier reversal_G) well-formed-elections
(φ-reverse well-formed-elections) (set-action ψ-reverse))

<proof>

end

1.11 Result-Dependent Voting Rule Properties

theory *Property-Interpretations*

imports *Voting-Symmetry*

Result-Interpretations

begin

1.11.1 Property Definitions

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

locale *result-properties* = *result* +
fixes ψ -neutral :: ('a \Rightarrow 'a, 'b) *binary-fun* **and**
voter-type :: 'v *itself*
assumes
action-neutral: *group-action neutrality*_G *UNIV* ψ -neutral **and**
neutrality:
is-symmetry ($\lambda \mathcal{E} :: ('a, 'v)$ *Election*. *limit* (*alternatives- \mathcal{E}* \mathcal{E}) *UNIV*)
(*action-induced-equivariance* (*carrier neutrality*_G)
well-formed-elections
(φ -neutral *well-formed-elections*) (*set-action* ψ -neutral))

sublocale *result-properties* \subseteq *result*
<proof>

1.11.2 Interpretations

global-interpretation *SCF-properties*: *result-properties well-formed-SCF*
limit-SCF ψ -neutral_c
<proof>

global-interpretation *SWF-properties*: *result-properties well-formed-SWF*
limit-SWF ψ -neutral_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 g :: 'a  $\Rightarrow$  'b::ord and
    S :: 'a set and
    x :: 'a
  assumes  $\forall x \in S. f\ x = g\ x$ 
  shows is-arg-min f ( $\lambda s. s \in S$ ) x = is-arg-min g ( $\lambda s. s \in S$ ) x
   $\langle$ 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 $finite\ \{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
 $\langle proof \rangle$

lemma *listset-finiteness*:
fixes $l :: 'a\ set\ list$
assumes $\forall\ i::nat.\ i < length\ l \longrightarrow finite\ (l[i])$
shows $finite\ (listset\ l)$
 $\langle proof \rangle$

lemma *all-ls-elems-same-len*:
fixes $l :: 'a\ set\ list$
shows $\forall\ l' :: 'a\ list.\ l' \in listset\ l \longrightarrow length\ l' = length\ l$
 $\langle proof \rangle$

lemma *all-ls-elems-in-ls-set*:
fixes $l :: 'a\ set\ list$
shows $\forall\ l' \in listset\ l.\ \forall\ i::nat < length\ l'.\ l'[i] \in l[i]$
 $\langle proof \rangle$

lemma *all-ls-in-ls-set*:
fixes $l :: 'a\ set\ list$
shows $\forall\ l'.\ length\ l' = length\ l$
 $\wedge (\forall\ i < length\ l'.\ l'[i] \in l[i] \longrightarrow l' \in listset\ l)$
 $\langle proof \rangle$

2.1.3 Ranking

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

fun *rank-l* :: $'a\ Preference-List \Rightarrow 'a \Rightarrow nat$ **where**
 $rank-l\ l\ a = (if\ a \in set\ l\ then\ index\ l\ a + 1\ else\ 0)$

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

lemma *rank-l-equiv*: $rank-l = rank-l-idx$
 $\langle proof \rangle$

lemma *rank-zero-imp-not-present*:
fixes
 $p :: 'a\ Preference-List$ **and**
 $a :: 'a$
assumes $rank-l\ p\ a = 0$
shows $a \notin set\ p$
 $\langle proof \rangle$

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
 <proof>

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*)
 <proof>

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

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

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

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

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**
 $a \ b :: 'a$
assumes
 $a \lesssim_l b$ **and**
 $\text{limited } A \ l$
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 $\text{pl-}\alpha \ (\text{limit-l } A \ l) = \text{limit } A \ (\text{pl-}\alpha \ l)$
 <proof>

2.1.6 Auxiliary Definitions

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

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

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

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

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

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

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

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

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

2.1.7 Auxiliary Lemmas

lemma *list-trans[simp]*:
fixes l :: 'a Preference-List
shows *trans* l
 <proof>

lemma *list-antisym[simp]*:
fixes l :: 'a Preference-List
shows *antisym-l* l
 <proof>

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

lemma *connex-imp-refl*:
fixes

```

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

lemma lin-ord-imp-connex-l:
  fixes
    A :: 'a set and
    l :: 'a Preference-List
  assumes linear-order-on-l A l
  shows connex-l A l
  ⟨proof⟩

lemma above-trans:
  fixes
    l :: 'a Preference-List and
    a b :: 'a
  assumes
    trans l and
    a ≲l b
  shows set (above-l l b) ⊆ set (above-l l a)
  ⟨proof⟩

lemma less-preferred-l-rel-equiv:
  fixes
    l :: 'a Preference-List and
    a b :: 'a
  shows a ≲l b =
    Preference-Relation.is-less-preferred-than a (pl-α l) b
  ⟨proof⟩

theorem above-equiv:
  fixes
    l :: 'a Preference-List and
    a :: 'a
  shows set (above-l l a) = above (pl-α l) a
  ⟨proof⟩

theorem rank-equiv:
  fixes
    l :: 'a Preference-List and
    a :: 'a
  assumes well-formed-l l
  shows rank-l l a = rank (pl-α l) a
  ⟨proof⟩

lemma lin-ord-equiv:
  fixes

```



```

    A :: 'a set and
    l :: 'a Preference-List
  shows linear-order-on-l A l = linear-order-on A (pl-α l)
  ⟨proof⟩

```

2.1.8 First Occurrence Indices

lemma *pos-in-list-yields-rank*:

```

  fixes
    l :: 'a Preference-List and
    a :: 'a and
    n :: nat
  assumes
    ∀ (j::nat) ≤ n. l!j ≠ a and
    l!(n - 1) = a
  shows rank-l l a = n
  ⟨proof⟩

```

lemma *ranked-alt-not-at-pos-before*:

```

  fixes
    l :: 'a Preference-List and
    a :: 'a and
    n :: nat
  assumes
    a ∈ set l and
    n < (rank-l l a) - 1
  shows l!n ≠ a
  ⟨proof⟩

```

lemma *pos-in-list-yields-pos*:

```

  fixes
    l :: 'a Preference-List and
    a :: 'a
  assumes a ∈ set l
  shows l!(rank-l l a - 1) = a
  ⟨proof⟩

```

lemma *rel-of-pref-pred-for-set-eq-list-to-rel*:

```

  fixes l :: 'a Preference-List
  shows relation-of (λ y z. y ≲l z) (set l) = pl-α l
  ⟨proof⟩

```

end

2.2 Preference (List) Profile

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

2.2.1 Definition

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

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

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

Abstraction from profile list to profile.

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

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

2.2.2 Refinement Proof

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

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

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

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 :: (finite) 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'  $\longrightarrow$  election-equality E' E and
  election-equality E E'  $\longrightarrow$  election-equality E' F
     $\longrightarrow \text{election-equality } E F$ 
   $\langle \text{proof} \rangle$ 

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

fun fstQ ::  $( 'a, 'v) \text{ Election}_{\mathcal{Q}} \Rightarrow 'a \text{ set}$  where

```

```

    fstQ E = Product-Type.fst (rep-ElectionQ E)

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

abbreviation alternatives-ℰQ :: ('a, 'v) ElectionQ ⇒ 'a set where
    alternatives-ℰQ E ≡ fstQ E

abbreviation voters-ℰQ :: ('a, 'v) ElectionQ ⇒ 'v set where
    voters-ℰQ E ≡ Product-Type.fst (sndQ E)

abbreviation profile-ℰQ :: ('a, 'v) ElectionQ ⇒ ('a, 'v) Profile where
    profile-ℰQ E ≡ Product-Type.snd (sndQ E)

end

```

Chapter 3

Quotient Rules

3.1 Quotients of Equivalence Relations

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

3.1.1 Definitions

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

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

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

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

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

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

3.1.2 Well-Definedness

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

assumes $\text{card } s \neq 1$
shows $\text{singleton-set } s = \text{undefined}$
 $\langle \text{proof} \rangle$

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

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

3.1.3 Equivalence Relations

lemma *restr-equals-restricted-rel*:
fixes
 $s \ t :: 'a \text{ set}$ **and**
 $r :: 'a \text{ rel}$
assumes
 $\text{closed-restricted-rel } r \ s \ t$ **and**
 $t \subseteq s$
shows $\text{restricted-rel } r \ t \ s = \text{Restr } r \ t$
 $\langle \text{proof} \rangle$

```

lemma equiv-rel-restr:
  fixes
     $s\ t :: 'x\ \text{set}$  and
     $r :: 'x\ \text{rel}$ 
  assumes
     $\text{equiv } s\ r$  and
     $t \subseteq s$ 
  shows  $\text{equiv } t\ (\text{Restr } r\ t)$ 
   $\langle \text{proof} \rangle$ 

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  $\text{group-action } m\ s\ \varphi$ 
  shows  $\text{equiv } s\ (\text{action-induced-rel } (\text{carrier } m)\ s\ \varphi)$ 
   $\langle \text{proof} \rangle$ 

end

```

3.2 Quotients of Election Set Equivalences

```

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

begin

```

3.2.1 Auxiliary Lemmas

```

lemma obtain-partition:
  fixes
     $A :: 'a\ \text{set}$  and
     $N :: 'b \Rightarrow \text{nat}$  and
     $B :: 'b\ \text{set}$ 
  assumes
     $\text{finite } A$  and
     $\text{finite } B$  and
     $\text{sum } N\ B = \text{card } A$ 
  shows  $\exists\ \mathcal{X}. A = \bigcup \{\mathcal{X}\ i \mid i. i \in B\} \wedge (\forall\ i \in B. \text{card } (\mathcal{X}\ i) = N\ i) \wedge$ 
     $(\forall\ i\ j. i \neq j \longrightarrow i \in B \wedge j \in B \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$ 
   $\langle \text{proof} \rangle$ 

```

3.2.2 Anonymity Quotient: Grid

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

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

fun *vote-count*_Q :: 'a Preference-Relation \Rightarrow ('a, 'v) Election set \Rightarrow nat **where**
*vote-count*_Q p = π_Q (*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^{nat} ('a Ordered-Preference)) (*anonymity*_Q (UNIV :: 'a set))
 (UNIV :: (nat^{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* ⇒ ('a, 'v) *Election set* ⇒ rat **where**
*vote-fraction*_Q p = π_Q (*vote-fraction* p)

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

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

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

The rational points in the simplex.

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

Auxiliary Lemmas

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

fixes

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

x :: real^{^b}

assumes ∃ f :: (real^{^b}) ⇒ real.

sum f *X* = 1 ∧ (∀ x ∈ *X*. f x ≥ 0)

∧ x = *sum* (λ x. (f x) *_R x) *X*

shows x ∈ *convex hull* *X*

⟨*proof*⟩

lemma *standard-simplex-rewrite*: *convex hull standard-basis* =

{v :: (real^{^b}). (∀ i. v\$i ≥ 0) ∧ *sum* (($\$$) v) *UNIV* = 1}

⟨*proof*⟩

lemma *fract-distr-helper*:

fixes a b c :: int

assumes c ≠ 0

shows *Fract* a c + *Fract* b c = *Fract* (a + b) c

⟨*proof*⟩

lemma *anonymity-homogeneity-is-equivalence*:

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

assumes ∀ E ∈ *X*. *finite* (*voters- \mathcal{E}* E)

shows *equiv* *X* (*anonymity-homogeneity*_R *X*)

$\langle proof \rangle$

lemma *fract-distr*:

fixes

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

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

$b :: \text{int}$

assumes

finite A **and**

$b \neq 0$

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

$\langle proof \rangle$

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* ($UNIV :: 'v \text{ set}$)

shows

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

$\langle proof \rangle$

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**
 $f\ g :: 'a \Rightarrow \text{int}$
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**
 $f\ g :: 'a \Rightarrow \text{int}$

shows $(\sum a \in A. f\ a) + (\sum a \in A. g\ a) = (\sum a \in A. f\ a + g\ a)$
 $\langle proof \rangle$

lemma *distrib-ereal*:

fixes

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

$f\ g :: 'a \Rightarrow int$

shows $ereal\ (real-of-int\ ((\sum a \in A. (f :: 'a \Rightarrow int)\ a) + (\sum a \in A. g\ a))) =$
 $ereal\ (real-of-int\ ((\sum a \in A. (f\ a) + (g\ a))))$

$\langle proof \rangle$

lemma *uneq-ereal*:

fixes $x\ y :: int$

assumes $x \leq y$

shows $ereal\ (real-of-int\ x) \leq ereal\ (real-of-int\ y)$

$\langle proof \rangle$

4.1.5 Swap Distance

fun *neq-ord* :: $'a\ Preference-Relation \Rightarrow 'a\ Preference-Relation \Rightarrow$

$'a \Rightarrow 'a \Rightarrow bool$ **where**

$neq-ord\ r\ s\ a\ b = ((a \preceq_r\ b \wedge b \preceq_s\ a) \vee (b \preceq_r\ a \wedge a \preceq_s\ b))$

fun *pairwise-disagreements* :: $'a\ set \Rightarrow 'a\ Preference-Relation \Rightarrow$

$'a\ Preference-Relation \Rightarrow ('a \times 'a)\ set$ **where**

$pairwise-disagreements\ A\ r\ s = \{(a, b) \in A \times A. a \neq b \wedge neq-ord\ r\ s\ a\ b\}$

fun *pairwise-disagreements'* :: $'a\ set \Rightarrow 'a\ Preference-Relation \Rightarrow$

$'a\ Preference-Relation \Rightarrow ('a \times 'a)\ set$ **where**

$pairwise-disagreements'\ A\ r\ s =$

$Set.filter\ (\lambda\ (a, b). a \neq b \wedge neq-ord\ r\ s\ a\ b)\ (A \times A)$

lemma *set-eq-filter*:

fixes

$X :: 'a\ set$ **and**

$P :: 'a \Rightarrow bool$

shows $\{x \in X. P\ x\} = Set.filter\ P\ X$

$\langle proof \rangle$

lemma *pairwise-disagreements-eq*[code]: $pairwise-disagreements = pairwise-disagreements'$

$\langle proof \rangle$

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

$swap\ (A, r)\ (A', r') =$

$(if\ A = A'$

$then\ card\ (pairwise-disagreements\ A\ r\ r')$

$else\ \infty)$

lemma *swap-case-infinity*:

```

fixes  $x\ y :: 'a\ Vote$ 
assumes  $alts\mathcal{V}\ x \neq alts\mathcal{V}\ y$ 
shows  $swap\ x\ y = \infty$ 
 $\langle proof \rangle$ 

```

```

lemma swap-case-fin:
  fixes  $x\ y :: 'a\ Vote$ 
  assumes  $alts\mathcal{V}\ x = alts\mathcal{V}\ y$ 
  shows  $swap\ x\ y = card\ (pairwise-disagreements\ (alts\mathcal{V}\ x)\ (pref\mathcal{V}\ x)\ (pref\mathcal{V}\ y))$ 
   $\langle proof \rangle$ 

```

4.1.6 Spearman Distance

```

fun spearman ::  $'a\ Vote\ Distance$  where
  spearman  $(A, x)\ (A', y) =$ 
    (if  $A = A'$ 
     then  $\sum a \in A. abs\ (int\ (rank\ x\ a) - int\ (rank\ y\ a))$ 
     else  $\infty$ )

```

```

lemma spearman-case-inf:
  fixes  $x\ y :: 'a\ Vote$ 
  assumes  $alts\mathcal{V}\ x \neq alts\mathcal{V}\ y$ 
  shows  $spearman\ x\ y = \infty$ 
   $\langle proof \rangle$ 

```

```

lemma spearman-case-fin:
  fixes  $x\ y :: 'a\ Vote$ 
  assumes  $alts\mathcal{V}\ x = alts\mathcal{V}\ y$ 
  shows  $spearman\ x\ y =$ 
    ( $\sum a \in alts\mathcal{V}\ x. abs\ (int\ (rank\ (pref\mathcal{V}\ x)\ a) - int\ (rank\ (pref\mathcal{V}\ y)\ a))$ )
   $\langle proof \rangle$ 

```

4.1.7 Properties

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

Definitions

```

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

```

```

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

```

```

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

```

$(bij \pi \longrightarrow$
 $(d (A, V, p) (A', V', p')) =$
 $(d (rename \pi (A, V, p))) (rename \pi (A', V', p')))$

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

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

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

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

Auxiliary Lemmas

lemma *rewrite-total-invariance _{\mathcal{D}}* :
fixes
d :: 'x Distance **and**
r :: 'x rel
shows total-invariance _{\mathcal{D}} d r = ($\forall (x, y) \in r. \forall (a, b) \in r. d a x = d b y$)
<proof>

lemma *rewrite-invariance _{\mathcal{D}}* :
fixes
d :: 'y Distance **and**
X :: 'x set **and**
Y :: 'y set **and**
 φ :: ('x, 'y) binary-fun
shows invariance _{\mathcal{D}} d X Y φ =
($\forall x \in X. \forall y \in Y. \forall z \in Y. d y z = d (\varphi x y) (\varphi x z)$)
<proof>

lemma *invar-dist-image*:
fixes
d :: 'y Distance **and**
G :: 'x monoid **and**
Y Y' :: 'y set **and**

```

     $\varphi :: ('x, 'y)$  binary-fun and
     $y :: 'y$  and
     $g :: 'x$ 
assumes
    invar-d: invarianceD  $d$  (carrier  $G$ )  $Y$   $\varphi$  and
    Y'-in-Y:  $Y' \subseteq Y$  and
    action- $\varphi$ : group-action  $G$   $Y$   $\varphi$  and
    g-carrier:  $g \in \text{carrier } G$  and
    y-in-Y:  $y \in Y$ 
shows  $d (\varphi g y) \text{ ' } (\varphi g) \text{ ' } Y' = d y \text{ ' } Y'$ 
 $\langle \text{proof} \rangle$ 

lemma swap-neutral: invarianceD swap (carrier neutralityG)
    UNIV  $(\lambda \pi (A, q). (\pi \text{ ' } A, \text{rel-rename } \pi q))$ 
 $\langle \text{proof} \rangle$ 

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 ::  $'a$  Vote Distance  $\Rightarrow$  Norm  $\Rightarrow$ 
    ( $'a, 'v :: \text{linorder}$ ) Election Distance where
    votewise-distance  $d$   $n$   $(A, V, p)$   $(A', V', p')$  =
    (if (finite  $V$ )  $\wedge$   $V = V' \wedge (V \neq \{\}$   $\vee$   $A = A')$ 
    then  $n$  (map2  $(\lambda q q'. d (A, q) (A', q'))$  (to-list  $V$   $p$ ) (to-list  $V'$   $p')$ )
    else  $\infty$ )

```

4.2.2 Inference Rules

```

lemma symmetric-norm-inv-under-map-permute:
fixes
     $d :: 'a$  Vote Distance and
     $n :: \text{Norm}$  and
     $A A' :: 'a$  set and

```


$\varphi :: \text{nat} \Rightarrow \text{nat}$ **and**
 $p \ p' :: ('a \text{ Preference-Relation}) \text{ list}$
assumes
 $\text{perm}: \varphi \text{ permutes } \{0 \dots \text{length } p\}$ **and**
 $\text{len-eq}: \text{length } p = \text{length } p'$ **and**
 $\text{sym-n}: \text{symmetry } n$
shows $n \ (\text{map2 } (\lambda q \ q'. d \ (A, q) \ (A', q')) \ p \ p') =$
 $n \ (\text{map2 } (\lambda q \ q'. d \ (A, q) \ (A', q')) \ (\text{permute-list } \varphi \ p) \ (\text{permute-list } \varphi \ p'))$
 $\langle \text{proof} \rangle$

lemma *permute-invariant-under-map*:
fixes $l \ l' :: 'a \text{ list}$
assumes $l < \sim \sim > l'$
shows $\text{map } f \ l < \sim \sim > \text{map } f \ l'$
 $\langle \text{proof} \rangle$

lemma *linorder-rank-injective*:
fixes
 $V :: 'v :: \text{linorder set}$ **and**
 $v \ v' :: 'v$
assumes
 $v\text{-in-}V: v \in V$ **and**
 $v'\text{-in-}V: v' \in V$ **and**
 $v'\text{-neq-}v: v' \neq v$ **and**
 $\text{fin-}V: \text{finite } V$
shows $\text{card } \{x \in V. x < v\} \neq \text{card } \{x \in V. x < v'\}$
 $\langle \text{proof} \rangle$

lemma *permute-invariant-under-coinciding-funs*:
fixes
 $l :: 'v \text{ list}$ **and**
 $\pi_1 \ \pi_2 :: \text{nat} \Rightarrow \text{nat}$
assumes $\forall \ i < \text{length } l. \pi_1 \ i = \pi_2 \ i$
shows $\text{permute-list } \pi_1 \ l = \text{permute-list } \pi_2 \ l$
 $\langle \text{proof} \rangle$

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

lemma *neutral-dist-imp-neutral-votewise-dist*:
fixes
 $d :: 'a \text{ Vote Distance}$ **and**
 $n :: \text{Norm}$
defines $\text{vote-action} \equiv (\lambda \pi \ (A, q). (\pi \ 'A, \text{rel-rename } \pi \ q))$

```

    assumes invar: invarianceD d (carrier neutralityG) UNIV vote-action
    shows distance-neutrality well-formed-elections (votewise-distance d n)
    <proof>

end

```

4.3 Consensus

```

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

```

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

4.3.1 Definition

```

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

```

4.3.2 Consensus Conditions

Nonempty alternative set.

```

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

```

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

```

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

```

Equal top ranked alternatives.

```

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

```

```

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

```

Equal votes.

```

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

```

```

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

```

Unanimity condition.

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

Strong unanimity condition.

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

4.3.3 Properties

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

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

4.3.4 Auxiliary Lemmas

lemma *cons-anon-conj*:
fixes c c' :: ('a, 'v) Consensus
assumes
 consensus-anonymity c **and**
 consensus-anonymity c'
shows *consensus-anonymity* (λ e. c e ∧ c' e)
 ⟨proof⟩

theorem *cons-conjunction-invariant*:
fixes
 ℄ :: ('a, 'v) Consensus set **and**
 rel :: ('a, 'v) Election rel
defines C ≡ (λ E. (∀ C' ∈ ℄. C' E))
assumes ∀ C'. C' ∈ ℄ ⟶ is-symmetry C' (Invariance rel)
shows is-symmetry C (Invariance rel)
 ⟨proof⟩

lemma *cons-anon-invariant*:
fixes
 c :: ('a, 'v) Consensus **and**
 A A' :: 'a set **and**
 V V' :: 'v set **and**
 p q :: ('a, 'v) Profile **and**
 π :: 'v ⇒ 'v
assumes
 anon: *consensus-anonymity* c **and**

bij- π : *bij* π **and**
prof- p : *profile* V A p **and**
renamed: *rename* π $(A, V, p) = (A', V', q)$ **and**
cond- c : c (A, V, p)
shows c (A', V', q)
 $\langle \text{proof} \rangle$

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: $\forall x. \text{consensus-anonymity } (b' x)$
shows *consensus-anonymity* b
 $\langle \text{proof} \rangle$

4.3.5 Theorems

Anonymity

lemma *nonempty-set-cons-anonymous:* *consensus-anonymity nonempty-set_C*
 $\langle \text{proof} \rangle$

lemma *nonempty-profile-cons-anonymous:* *consensus-anonymity nonempty-profile_C*
 $\langle \text{proof} \rangle$

lemma *equal-top-cons'-anonymous:*
fixes $a :: 'a$
shows *consensus-anonymity* $(\text{equal-top}_C 'a)$
 $\langle \text{proof} \rangle$

lemma *eq-top-cons-anon:* *consensus-anonymity equal-top_C*
 $\langle \text{proof} \rangle$

lemma *eq-vote-cons'-anonymous:*
fixes $r :: 'a$ *Preference-Relation*
shows *consensus-anonymity* $(\text{equal-vote}_C 'r)$
 $\langle \text{proof} \rangle$

lemma *eq-vote-cons-anonymous:* *consensus-anonymity equal-vote_C*
 $\langle \text{proof} \rangle$

Neutrality

lemma *nonempty-set_C-neutral:* *consensus-neutrality well-formed-elections nonempty-set_C*
 $\langle \text{proof} \rangle$

lemma *nonempty-profile_C-neutral:* *consensus-neutrality well-formed-elections nonempty-profile_C*
 $\langle \text{proof} \rangle$

```

lemma equal-voteC-neutral: consensus-neutrality well-formed-elections equal-voteC
  <proof>

lemma strong-unanimityC-neutral: consensus-neutrality
  well-formed-elections strong-unanimityC
  <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 alternatives, but can instead defer the decision for some or all of them to other modules. Hence, electoral modules partition the received (possibly empty) set of alternatives into elected, rejected and deferred alternatives. In particular, any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives. Just like a voting rule, an electoral module also receives a profile which holds the voters preferences, which, unlike a voting rule, consider only the (sub-)set of alternatives that the module receives.

4.4.1 Definition

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

```

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

```

```

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

```

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

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

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

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

4.4.2 Auxiliary Definitions

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

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

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

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

4.4.3 Properties

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

Anonymity

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

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

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

fun *anonymity'* :: ('a, 'v) *Election set* \Rightarrow ('a, 'v, 'r) *Electoral-Module* \Rightarrow *bool* **where**
anonymity' *X m* = *is-symmetry* (*fun_E* *m*) (*Invariance* (*anonymity_R* *X*))

Homogeneity

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

fun (*in result*) *homogeneity* :: ('a, 'v) *Election set* \Rightarrow
('a, 'v, ('r Result)) *Electoral-Module* \Rightarrow *bool* **where**
homogeneity *X m* = *is-symmetry* (*fun_E* *m*) (*Invariance* (*homogeneity_R* *X*))
— This does not require any specific behaviour on infinite voter sets ... It might make sense to extend the definition to that case somehow.

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

lemma (*in result*) *hom-imp-anon*:
fixes
X :: ('a, 'v) *Election set* **and**
m :: ('a, 'v, ('r Result)) *Electoral-Module*
assumes
homogeneity *X m* **and**
 $\forall E \in X. \text{finite } (\text{voters-}\mathcal{E} \ E)$
shows *anonymity'* *X m*
 $\langle \text{proof} \rangle$

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 (*fun_E* *m*) (*action-induced-equivariance* (*carrier neutrality_G*) *X*
(φ -neutral *X*) (*result-action* ψ -neutral))

4.4.4 Social-Welfare Properties

Reversal Symmetry

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

definition *reversal-symmetry* :: ('a, 'v) Election set \Rightarrow
('a, 'v, 'a rel Result) Electoral-Module \Rightarrow bool **where**
reversal-symmetry X m =
is-symmetry (fun_E m) (action-induced-equivariance (carrier reversal_G) X
(φ -reverse X) (result-action ψ -reverse))

4.4.5 Social-Choice Modules

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

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

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

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

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

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

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

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

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

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

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

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

4.4.6 Equivalence Definitions

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

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

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

definition *mod-contains-result* :: ($'a, 'v, 'a \text{ Result}$) *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**
mod-contains-result $m \ n \ V \ A \ p \ a \equiv$
 $SCF\text{-result.electoral-module } m \wedge$
 $SCF\text{-result.electoral-module } n \wedge$

$profile\ V\ A\ p \wedge a \in A \wedge$
 $(a \in elect\ m\ V\ A\ p \longrightarrow a \in elect\ n\ V\ A\ p) \wedge$
 $(a \in reject\ m\ V\ A\ p \longrightarrow a \in reject\ n\ V\ A\ p) \wedge$
 $(a \in defer\ m\ V\ A\ p \longrightarrow a \in defer\ n\ V\ A\ p)$

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

4.4.7 Auxiliary Lemmas

lemma *elect-rej-def-combination*:

fixes

$m :: (\text{'a}, \text{'v}, \text{'a Result})\ \text{Electoral-Module}$ **and**
 $V :: \text{'v set}$ **and**
 $A :: \text{'a set}$ **and**
 $p :: (\text{'a}, \text{'v})\ \text{Profile}$ **and**
 $e\ r\ d :: \text{'a set}$

assumes

$elect\ m\ V\ A\ p = e$ **and**
 $reject\ m\ V\ A\ p = r$ **and**
 $defer\ m\ V\ A\ p = d$

shows $m\ V\ A\ p = (e, r, d)$
 $\langle proof \rangle$

lemma *par-comp-result-sound*:

fixes

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

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

$m :: (\text{'a}, \text{'v}, \text{'a Result})\ \text{Electoral-Module}$ **and**
 $A :: \text{'a set}$ **and**
 $V :: \text{'v set}$ **and**
 $p :: (\text{'a}, \text{'v})\ \text{Profile}$

assumes
SCF-result.electoral-module *m* **and**
profile *V A p*
shows $(\text{elect } m \ V \ A \ p) \cup (\text{reject } m \ V \ A \ p) \cup (\text{defer } m \ V \ A \ p) = A$
 $\langle \text{proof} \rangle$

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

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 $\text{elect } m \ V \ A \ p \subseteq A$
 $\langle \text{proof} \rangle$

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 $\text{reject } m \ V \ A \ p \subseteq A$
 $\langle \text{proof} \rangle$

lemma *defer-in-alts*:
fixes
m :: ('a, 'v, 'a Result) *Electoral-Module* **and**
A :: 'a set **and**
V :: 'v set **and**

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

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

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

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

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

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
 $SCF\text{-result.electoral-module } m$ **and**
 $profile\ V\ A\ p$
shows $elect\ m\ V\ A\ p \cup defer\ m\ V\ A\ p = A - (reject\ m\ V\ A\ p)$
 $\langle proof \rangle$

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
 $SCF\text{-result.electoral-module } m$ **and**
 $profile\ V\ A\ p$
shows $defer\ m\ V\ A\ p = A - (elect\ m\ V\ A\ p) - (reject\ m\ V\ A\ p)$
 $\langle proof \rangle$

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
 $m\ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **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**
 $\text{profile } V A p$ **and**
 $a \in A$ **and**
 $a \notin \text{reject } m V A p$
shows $a \in \text{elect } m V A p \vee a \in \text{defer } m V A p$
 $\langle \text{proof} \rangle$

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
 $\text{eliminates } 1 m$ **and**
 $\text{card } A > 1$ **and**
 $\text{profile } V A p$
shows $\text{defer } m V A p \subset A$
 $\langle \text{proof} \rangle$

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**
 $p \ q :: ('a, 'v) \text{ Profile}$
assumes
 $\text{eq: } \forall a \in A. \text{prof-contains-result } m V A p \ q \ a$ **and**
 $\text{mod-}m: SCF\text{-result.electoral-module } m$ **and**
 $\text{prof-}p: \text{profile } V A p$ **and**
 $\text{prof-}q: \text{profile } V A q$
shows $m V A p = m V A q$
 $\langle \text{proof} \rangle$

lemma *eq-def-and-elect-imp-eq*:
fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p \ q :: ('a, 'v) \text{ Profile}$
assumes

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* = *elect* *n V A q* **and**
def-eq: *defer* *m V A p* = *defer* *n V A q*
shows *m V A p* = *n V A q*
 ⟨*proof*⟩

4.4.8 Non-Blocking

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

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

4.4.9 Electing

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

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

lemma *electing-for-only-alt*:

fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
one-alt: *card* *A* = 1 **and**
electing: *electing* *m* **and**
prof: *profile* *V A p*
shows *elect* *m V A p* = *A*
 ⟨*proof*⟩

theorem *electing-imp-non-blocking*:

fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes *electing* *m*
shows *non-blocking* *m*
 ⟨*proof*⟩

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
 <proof>

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

fixes
 m :: ('a, 'v, 'a Result) Electoral-Module **and**
 A :: 'a set **and**
 V :: 'v set **and**
 p :: ('a, 'v) Profile
assumes
 eliminating: eliminates 1 m **and**
 non-electing: non-electing m **and**
 not-empty: card A > 1 **and**
 prof-p: profile V A p
shows card (defer m V A p) = card A - 1
 <proof>

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

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

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

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

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

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

definition *condorcet-compatibility* :: ('a, 'v, 'a Result)

Electoral-Module \Rightarrow bool **where**

condorcet-compatibility $m \equiv$

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

$(\forall\ A\ V\ p\ a.\ condorcet\text{-}winner\ V\ A\ p\ a \longrightarrow$

$(a \notin reject\ m\ V\ A\ p \wedge$

$(\forall\ b.\ \neg\ condorcet\text{-}winner\ V\ A\ p\ b \longrightarrow b \notin elect\ m\ V\ A\ p) \wedge$

$(a \in elect\ m\ V\ A\ p \longrightarrow$

$(\forall\ b \in A.\ \neg\ condorcet\text{-}winner\ V\ A\ p\ b \longrightarrow b \in reject\ m\ V\ A\ p))))$

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

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

defer-monotonicity $m \equiv$

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

$(\forall\ A\ V\ p\ q\ a.$

$(a \in defer\ m\ V\ A\ p \wedge lifted\ V\ A\ p\ q\ a) \longrightarrow a \in defer\ m\ V\ A\ q)$

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 $m \equiv$

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

$(\forall\ A\ V\ p\ q\ a.\ (a \in (defer\ m\ V\ A\ p) \wedge lifted\ V\ A\ p\ q\ a)$

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

fun *dli-rel* :: ('a, 'v, 'a Result) *Electoral-Module* \Rightarrow ('a, 'v) *Election rel* **where**

dli-rel $m = \{((A, V, p), (A, V, q)) \mid A\ V\ p\ q.\ (\exists\ a \in defer\ m\ V\ A\ p.\ lifted\ V\ A\ p\ q\ a)\}$

lemma *rewrite-dli-as-invariance*:

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

shows

defer-lift-invariance $m =$

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

$\wedge (is\text{-}symmetry\ (fun_{\mathcal{E}}\ m)\ (Invariance\ (dli\text{-}rel\ m))))$

<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 $m\ n \equiv$

$$\begin{aligned}
& SCF\text{-}result.electoral\text{-}module\ m \wedge SCF\text{-}result.electoral\text{-}module\ n \wedge \\
& (\forall\ V. \\
& (\forall\ A. \\
& (\exists\ B \subseteq A. \\
& (\forall\ a \in B. indep\text{-}of\text{-}alt\ m\ V\ A\ a \wedge \\
& (\forall\ p. profile\ V\ A\ p \longrightarrow a \in reject\ m\ V\ A\ p)) \wedge \\
& (\forall\ a \in A - B. indep\text{-}of\text{-}alt\ n\ V\ A\ a \wedge \\
& (\forall\ p. profile\ V\ A\ p \longrightarrow a \in reject\ n\ V\ A\ p))))))
\end{aligned}$$

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 $m \equiv$

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

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 $m \equiv$

$$\begin{aligned}
& SCF\text{-}result.electoral\text{-}module\ m \wedge non\text{-}electing\ m \wedge \\
& (\forall\ A\ V\ p\ q\ a. (a \in defer\ m\ V\ A\ p \wedge lifted\ V\ A\ p\ q\ a) \longrightarrow \\
& (defer\ m\ V\ A\ q = defer\ m\ V\ A\ p \vee defer\ m\ V\ A\ q = \{a\}))
\end{aligned}$$

4.4.11 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:

fixes

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

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

$V :: 'v\ set$ **and**

$p :: ('a, 'v)\ \textit{Profile}$ **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 $m :: ('a, 'v, 'a\ Result)\ \textit{Electoral-Module}$

assumes

ccomp: *condorcet-compatibility* m **and**

dd: *defer-deciding* m

shows *defer-condorcet-consistency* m
 $\langle \text{proof} \rangle$

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

theorem *disj-compat-comm[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
assumes *disjoint-compatibility* $m\ n$
shows *disjoint-compatibility* $n\ m$
 $\langle \text{proof} \rangle$

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

theorem *dl-inv-imp-def-mono[simp]*:
fixes $m :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
assumes *defer-lift-invariance* m
shows *defer-monotonicity* m
 $\langle \text{proof} \rangle$

4.4.12 Social-Choice Properties

Condorcet Consistency

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

lemma *condorcet-consistency'*:
fixes $m :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
shows *condorcet-consistency* $m =$
 $(\text{SCF-result.electoral-module } m \wedge$
 $(\forall\ A\ V\ p\ a.\ \text{condorcet-winner } V\ A\ p\ a \longrightarrow$
 $(m\ V\ A\ p = (\{a\}, A - (\text{elect } m\ V\ A\ p), \{\}))))$
 $\langle \text{proof} \rangle$

lemma *condorcet-consistency''*:
fixes $m :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
shows *condorcet-consistency* $m =$
 $(\text{SCF-result.electoral-module } m \wedge$
 $(\forall\ A\ V\ p\ a.\$
 $\text{condorcet-winner } V\ A\ p\ a \longrightarrow m\ V\ A\ p = (\{a\}, A - \{a\}, \{\})))$
 $\langle \text{proof} \rangle$

(Weak) Monotonicity

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

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

```

monotonicity m ≡
  SCF-result.electoral-module m ∧
  (∀ A V p q a. a ∈ elect m V A p ∧ lifted V A p q a → a ∈ elect m V A q)

end

```

4.5 Electoral Module on Election Quotients

```

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

lemma invariance-is-congruence:
  fixes
    m :: ('a, 'v, 'r) Electoral-Module and
    r :: ('a, 'v) Election rel
  shows (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  $\Rightarrow$  'a  $\Rightarrow$  'a set  $\Rightarrow$  ('a, 'v) Profile  $\Rightarrow$  enat

```

4.6.2 Property

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

```

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

```

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

```

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

```

4.6.3 Theorems

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

```

theorem cond-winner-imp-max-eval-val:
  fixes
     $e :: ('a, 'v) \text{Evaluation-Function}$  and
     $A :: 'a \text{ set}$  and
     $V :: 'v \text{ set}$  and
     $p :: ('a, 'v) \text{Profile}$  and
     $a :: 'a$ 
  assumes
    rating: condorcet-rating  $e$  and
    f-prof: finite-profile  $V A p$  and
    winner: condorcet-winner  $V A p a$ 
  shows  $e V a A p = \text{Max } \{e V b A p \mid b. b \in A\}$ 
  <proof>

```

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

```

theorem non-cond-winner-not-max-eval:
  fixes
     $e :: ('a, 'v)$  Evaluation-Function and
     $A :: 'a$  set and
     $V :: 'v$  set and
     $p :: ('a, 'v)$  Profile and
     $a\ b :: 'a$ 
  assumes
    rating: condorcet-rating  $e$  and
    f-prof: finite-profile  $V\ A\ p$  and
    winner: condorcet-winner  $V\ A\ p\ a$  and
    lin-A:  $b \in A$  and
    loser:  $a \neq b$ 
  shows  $e\ V\ b\ A\ p < \text{Max } \{e\ V\ c\ A\ p \mid c. c \in A\}$ 
  <proof>

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  $e\ V\ A\ p = (\text{let } \text{sum} = (\sum\ x \in A. e\ V\ x\ A\ p)\ \text{in}$ 

```

$(\text{if } (\text{sum} = \text{infinity}) \text{ then } (\text{infinity})$
 $\text{else } ((\text{the-enat sum}) \text{ div } (\text{card } A)))$

4.7.2 Social-Choice Definitions

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

4.7.3 Social-Choice Eliminators

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

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

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

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

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

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

4.7.4 Soundness

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

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

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

lemma *max-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*max-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *min-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*min-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *less-avg-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*less-average-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *leq-avg-elim-sound*[simp]:
fixes $e :: ('a, 'v)$ *Evaluation-Function*
shows *SCF-result.electoral-module* (*leq-average-eliminator* e)
 $\langle \text{proof} \rangle$

4.7.5 Independence of Non-Voters

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

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 :: *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) *Evaluation-Function*
assumes *voters-determine-evaluation e*
shows *voters-determine-election (max-eliminator e)*
 ⟨*proof*⟩

lemma *voters-determine-min-elim[simp]*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
assumes *voters-determine-evaluation e*
shows *voters-determine-election (min-eliminator e)*
 ⟨*proof*⟩

lemma *voters-determine-less-avg-elim[simp]*:
fixes *e* :: ('a, 'v) *Evaluation-Function*
assumes *voters-determine-evaluation e*
shows *voters-determine-election (less-average-eliminator e)*
 ⟨*proof*⟩

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

4.7.6 Non-Blocking

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

lemma *less-elim-non-blocking*:
fixes
 e :: ('a, 'v) *Evaluation-Function* **and**

$t :: \text{Threshold-Value}$
shows *non-blocking* (*less-eliminator* e t)
 $\langle \text{proof} \rangle$

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

lemma *max-elim-non-blocking*:
fixes $e :: ('a, 'v) \text{Evaluation-Function}$
shows *non-blocking* (*max-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *min-elim-non-blocking*:
fixes $e :: ('a, 'v) \text{Evaluation-Function}$
shows *non-blocking* (*min-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *less-avg-elim-non-blocking*:
fixes $e :: ('a, 'v) \text{Evaluation-Function}$
shows *non-blocking* (*less-average-eliminator* e)
 $\langle \text{proof} \rangle$

lemma *leq-avg-elim-non-blocking*:
fixes $e :: ('a, 'v) \text{Evaluation-Function}$
shows *non-blocking* (*leq-average-eliminator* e)
 $\langle \text{proof} \rangle$

4.7.7 Non-Electing

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

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

lemma *leq-elim-non-electing*:

```

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

lemma max-elim-non-electing:
  fixes  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
shows non-electing (max-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

lemma min-elim-non-electing:
  fixes  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
shows non-electing (min-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

lemma less-avg-elim-non-electing:
  fixes  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
shows non-electing (less-average-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

lemma leq-avg-elim-non-electing:
  fixes  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
shows non-electing (leq-average-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

```

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  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
assumes condorcet-rating  $e$ 
shows condorcet-compatibility (max-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

```

If the used evaluation function is Condorcet rating, max-eliminator is defer-Condorcet-consistent.

```

theorem cr-eval-imp-dcc-max-elim[simp]:
  fixes  $e :: ('a, 'v) \text{ Evaluation-Function}$ 
assumes condorcet-rating  $e$ 
shows defer-condorcet-consistency (max-eliminator  $e$ )
   $\langle \text{proof} \rangle$ 

```

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

4.8.2 Properties

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

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

end

4.9 Maximum Aggregator

```
theory Maximum-Aggregator
  imports Aggregator
begin
```

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

4.9.1 Definition

```
fun max-aggregator :: 'a Aggregator where
  max-aggregator  $A$   $(e, r, d)$   $(e', r', d') =$ 
     $(e \cup e',$ 
       $A - (e \cup e' \cup d \cup d'),$ 
       $(d \cup d') - (e \cup e'))$ 
```

4.9.2 Auxiliary Lemma

```
lemma max-agg-rej-set:
  fixes
     $A$   $e$   $e'$   $d$   $d'$   $r$   $r' :: 'a$  set and
     $a :: 'a$ 
  assumes
    wf-first-mod: well-formed-SCF  $A$   $(e, r, d)$  and
    wf-second-mod: well-formed-SCF  $A$   $(e', r', d')$ 
  shows reject-r  $(\text{max-aggregator } A (e, r, d) (e', r', d')) = r \cap r'$ 
   $\langle \text{proof} \rangle$ 
```

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.

```
type-synonym 'r Termination-Condition = 'r Result  $\Rightarrow$  bool  
  
end
```

4.11 Defer Equal Condition

```
theory Defer-Equal-Condition  
  imports Termination-Condition  
begin
```

This is a family of termination conditions. For a natural number n , the according defer-equal condition is true if and only if the given result's defer-set contains exactly n elements.

```
fun defer-equal-condition :: nat  $\Rightarrow$  'a Termination-Condition where  
  defer-equal-condition  $n$  ( $e, r, d$ ) = ( $\text{card } d = n$ )  
  
end
```

Chapter 5

Basic Modules

5.1 Defer Module

```
theory Defer-Module
  imports Component-Types/Electoral-Module
begin
```

The defer module is not concerned about the voter's ballots, and simply defers all alternatives. It is primarily used for defining an empty loop.

5.1.1 Definition

```
fun defer-module :: ('a, 'v, 'a Result) Electoral-Module where
  defer-module V A p = ({}, {}, A)
```

5.1.2 Soundness

```
theorem def-mod-sound[simp]: SCF-result.electoral-module defer-module
  <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 $\mathcal{K}_\mathcal{E} :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow 'r \Rightarrow ('a, 'v) \text{ Election set}$ **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 $\text{well-formed} :: ('a, 'v) \text{ Consensus} \Rightarrow ('a, 'v, 'r) \text{ Electoral-Module} \Rightarrow$
 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 $\text{elections-}\mathcal{K} C \subseteq \text{well-formed-elections}$
 $\langle \text{proof} \rangle$

lemma *cons-domain-finite:*
fixes $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
shows
 $\text{finite: elections-}\mathcal{K} C \subseteq \text{finite-elections}$ **and**
 $\text{finite-voters: elections-}\mathcal{K} C \subseteq \text{finite-elections-}\mathcal{V}$
 $\langle \text{proof} \rangle$

5.3.4 Consensus Rules

definition $\text{non-empty-set} :: ('a, 'v, 'r) \text{ Consensus-Class} \Rightarrow \text{bool}$ **where**
 $\text{non-empty-set } c \equiv \exists K. \text{consensus-}\mathcal{K} c K$

Unanimity condition.

definition $\text{unanimity} :: ('a, 'v::\text{wellorder}, 'a \text{ Result}) \text{ Consensus-Class}$ **where**
 $\text{unanimity} = \text{consensus-choice unanimity}_C \text{elect-first-module}$

Strong unanimity condition.

definition $\text{strong-unanimity} :: ('a, 'v::\text{wellorder}, 'a \text{ Result}) \text{ Consensus-Class}$ **where**
 $\text{strong-unanimity} = \text{consensus-choice strong-unanimity}_C \text{elect-first-module}$

5.3.5 Properties

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

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

consensus-rule-anonymity' $X \ C =$
is-symmetry (*elect-r* \circ *fun_E* (*rule-K* C)) (*Invariance* (*anonymity_R* X))

fun (**in** *result-properties*) *consensus-rule-neutrality* :: (*'a*, *'v*) *Election set* \Rightarrow
 (*'a*, *'v*, *'b Result*) *Consensus-Class* \Rightarrow *bool* **where**
consensus-rule-neutrality $X \ C =$
is-symmetry (*elect-r* \circ *fun_E* (*rule-K* C))
 (*action-induced-equivariance*
 (*carrier neutrality_G*) X (φ -*neutral* X) (*set-action* ψ -*neutral*))

fun *consensus-rule-reversal-symmetry* :: (*'a*, *'v*) *Election set* \Rightarrow
 (*'a*, *'v*, *'a rel Result*) *Consensus-Class* \Rightarrow *bool* **where**
consensus-rule-reversal-symmetry $X \ C =$ *is-symmetry* (*elect-r* \circ *fun_E* (*rule-K* C))
 (*action-induced-equivariance* (*carrier reversal_G*) X (φ -*reverse* X) (*set-action*
 ψ -*reverse*))

5.3.6 Inference Rules

lemma *if-else-cons-equivar*:

fixes

$m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $c :: ('a, 'v) \text{ Consensus}$ **and**
 $G :: 'b \text{ set}$ **and**
 $X :: ('a, 'v) \text{ Election set}$ **and**
 $\varphi :: ('b, ('a, 'v) \text{ Election}) \text{ binary-fun}$ **and**
 $\psi :: ('b, 'a) \text{ binary-fun}$ **and**
 $f :: 'a \text{ Result} \Rightarrow 'a \text{ set}$

defines

$\text{equivar} \equiv \text{action-induced-equivariance } G \ X \ \varphi \ (\text{set-action } \psi)$ **and**
 $\text{if-else-cons} \equiv (c, (\lambda \ V \ A \ p. \text{if } c \ (A, V, p) \text{ then } m \ V \ A \ p \text{ else } n \ V \ A \ p))$

assumes

$\text{equivar-m}: \text{is-symmetry } (f \circ \text{fun}_E \ m) \ \text{equivar}$ **and**
 $\text{equivar-n}: \text{is-symmetry } (f \circ \text{fun}_E \ n) \ \text{equivar}$ **and**
 $\text{invar-cons}: \text{is-symmetry } c \ (\text{Invariance } (\text{action-induced-rel } G \ X \ \varphi))$

shows $\text{is-symmetry } (f \circ \text{fun}_E \ (\text{rule-K } \text{if-else-cons}))$

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

<proof>

lemma *consensus-choice-anonymous*:

fixes

$\alpha \ \beta :: ('a, 'v) \text{ Consensus}$ **and**
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $\beta' :: 'b \Rightarrow ('a, 'v) \text{ Consensus}$

assumes

$\text{beta-sat}: \beta = (\lambda \ E. \exists \ a. \beta' \ a \ E)$ **and**
 $\text{beta'-anon}: \forall \ x. \text{consensus-anonymity } (\beta' \ x)$ **and**
 $\text{anon-cons-cond}: \text{consensus-anonymity } \alpha$ **and**
 $\text{conditions-univ}: \forall \ x. \text{well-formed } (\lambda \ E. \alpha \ E \wedge \beta' \ x \ E) \ m$

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

<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

$G :: 'b$ set **and**

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

$\varphi :: ('b, ('a, 'v)$ Election) binary-fun **and**

$\psi :: ('b, 'a)$ binary-fun

shows *is-symmetry* ($\text{elect-r} \circ \text{fun}_E$ defer-module)

(action-induced-equivariance G E φ (set-action ψ))

<proof>

lemma *elect-first-winners-neutral: is-symmetry* ($\text{elect-r} \circ \text{fun}_E$ elect-first-module)

(action-induced-equivariance (carrier neutrality_G)

well-formed-elections (φ -neutral well-formed-elections)

(set-action ψ -neutral_c))

<proof>

lemma *strong-unanimity-neutral:*

defines $\text{domain} \equiv \text{well-formed-elections} \cap \text{Collect strong-unanimity}_C$

— 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 well-formed-elections) well-formed-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 *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 *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 profiles :: 'v set  $\Rightarrow$  'a set  $\Rightarrow$  (('a, 'v) Profile) set where
  profiles V A =
    (if (infinite A  $\vee$  infinite V)
     then {} else {p. p ' V  $\subseteq$  (pl- $\alpha$  ' permutations-of-set A)})

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

```

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

```

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

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

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

```

5.4.3 Auxiliary Lemmas

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

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

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

$fin-A: \text{finite } A$ **and**
 $fin-B: \text{finite } S$
shows $finite \{a\#l \mid a \in A \wedge l \in S\}$
 $\langle proof \rangle$

lemma *listset-finiteness*:
fixes $l :: 'a \text{ set list}$
assumes $\forall i :: nat. i < length\ l \longrightarrow finite\ (l[i])$
shows $finite\ (listset\ l)$
 $\langle proof \rangle$

lemma *ls-entries-empty-imp-ls-set-empty*:
fixes $l :: 'a \text{ set list}$
assumes
 $0 < length\ l$ **and**
 $\forall i :: nat. i < length\ l \longrightarrow l[i] = \{\}$
shows $listset\ l = \{\}$
 $\langle proof \rangle$

lemma *all-ls-elems-same-len*:
fixes $l :: 'a \text{ set list}$
shows $\forall l' :: 'a \text{ list}. l' \in listset\ l \longrightarrow length\ l' = length\ l$
 $\langle proof \rangle$

lemma *fin-all-profs*:
fixes
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $x :: 'a \text{ Preference-Relation}$
assumes
 $fin-A: \text{finite } A$ **and**
 $fin-V: \text{finite } V$
shows $finite\ (profiles\ V\ A \cap \{p. \forall v. v \notin V \longrightarrow p\ v = x\})$
 $\langle proof \rangle$

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

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* ($distance-\mathcal{R}\ d\ K$)

<proof>

5.4.5 Inference Rules

lemma (in *result*) *standard-distance-imp-equal-score*:
fixes
 $d :: ('a, 'v)$ *Election Distance* **and**
 $K :: ('a, 'v, 'r \text{ Result})$ *Consensus-Class* **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v)$ *Profile* **and**
 $w :: 'r$
assumes
 $\text{irr-non-}V$: *voters-determine-distance* d **and**
 std : *standard* d
shows $\text{score } d \ K \ (A, V, p) \ w = \text{score-std } d \ K \ (A, V, p) \ w$
<proof>

lemma (in *result*) *anonymous-distance-and-consensus-imp-rule-anonymity*:
fixes
 $d :: ('a, 'v)$ *Election Distance* **and**
 $K :: ('a, 'v, 'r \text{ Result})$ *Consensus-Class*
assumes
 $d\text{-anon}$: *distance-anonymity* d **and**
 $K\text{-anon}$: *consensus-rule-anonymity* K
shows *anonymity* ($\text{distance-}\mathcal{R} \ d \ K$)
<proof>

end

5.5 Votewise Distance Rationalization

theory *Votewise-Distance-Rationalization*
imports *Distance-Rationalization*
Votewise-Distance

begin

A votewise distance rationalization of a voting rule is its distance rationalization with a distance function that depends on the submitted votes in a simple and a transparent manner by using a distance on individual orders and combining the components with a norm on \mathbb{R} to \mathbb{N} .

5.5.1 Common Rationalizations

fun $\text{swap-}\mathcal{R} :: ('a, 'v::\text{linorder}, 'a \text{ Result})$ *Consensus-Class* \Rightarrow
 $('a, 'v, 'a \text{ Result})$ *Electoral-Module* **where**
 $\text{swap-}\mathcal{R} \ K = \text{SCF-result.distance-}\mathcal{R} \ (\text{votewise-distance swap l-one}) \ K$

5.5.2 Theorems

lemma *votewise-non-voters-irrelevant:*

fixes

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

$N :: \text{Norm}$

shows *voters-determine-distance* (*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

($'a, 'v, 'a \text{ Result}$) *Consensus-Class* \Rightarrow ($'a, 'v$) *Election* $\Rightarrow 'a \Rightarrow \text{ereal}$

type-synonym ($'a, 'v$) *dist-rat-type* = ($'a, 'v$) *Election Distance* \Rightarrow

($'a, 'v, 'a \text{ Result}$) *Consensus-Class* $\Rightarrow 'v \text{ set} \Rightarrow 'a \text{ set} \Rightarrow ('a, 'v) \text{ Profile} \Rightarrow 'a \text{ set}$

type-synonym ($'a, 'v$) *dist-rat-std-type* = ($'a, 'v$) *Election Distance* \Rightarrow

($'a, 'v, 'a \text{ Result}$) *Consensus-Class* $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

type-synonym ($'a, 'v$) *dist-type* = ($'a, 'v$) *Election Distance* \Rightarrow

($'a, 'v, 'a \text{ Result}$) *Consensus-Class* $\Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$

lemma *equal-score-swap:* ($\text{score} :: ('a, 'v::\text{linorder}) \text{ score-type}$)

(*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* :: ($'a, 'v :: \text{linorder}$) *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* :: $'a \text{ Distance} \Rightarrow 'a \text{ set} \Rightarrow 'a \Rightarrow \text{ereal}$ **where**

distance-infimum d A $a = \text{Inf } (d \text{ a } 'A)$

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

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

Auxiliary Lemmas

lemma *rewrite-arg-min-set*:
 fixes
 f :: 'a \Rightarrow 'b::linorder **and**
 A :: 'a set
 shows arg-min-set f A = \bigcup (preimg f A ' {y \in (f ' A). \forall z \in f ' A. y \leq z})
 ⟨proof⟩

Equivariance

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

lemma *restr-induced-rel*:
 fixes
 A :: 'a set **and**
 B B' :: 'b set **and**
 φ :: ('a, 'b) binary-fun
 assumes B' \subseteq B
 shows *Restr* (action-induced-rel A B φ) B' = action-induced-rel A B' φ
 ⟨proof⟩

theorem *group-action-invar-dist-and-equivar-f-imp-equivar-minimizer*:

fixes
 f :: 'a \Rightarrow 'b **and**
 domain_f X :: 'a set **and**
 d :: 'a Distance **and**
 well-formed-img :: 'a \Rightarrow 'b set **and**
 G :: 'c monoid **and**
 φ :: ('c, 'a) binary-fun **and**
 ψ :: ('c, 'b) binary-fun
 defines equivar-prop-set-valued \equiv
 action-induced-equivariance (carrier G) X φ (set-action ψ)
 assumes
 action- φ : group-action G X φ **and**
 group-action-res: group-action G UNIV ψ **and**
 dom-in-X: domain_f \subseteq X **and**
 closed-domain:
 closed-restricted-rel (action-induced-rel (carrier G) X φ) X domain_f **and**
 equivar-img: is-symmetry well-formed-img equivar-prop-set-valued **and**

invar-d: invariance_D d (carrier G) X φ and
equivar-f:
is-symmetry f (action-induced-equivariance (carrier G) domain_f φ ψ)
shows *is-symmetry ($\lambda x.$ minimizer f domain_f d (well-formed-img x) x) equivar-prop-set-valued*
<proof>

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist:*

fixes
f :: 'a \Rightarrow 'b and
domain_f :: 'a set and
d :: 'a Distance and
rel :: 'a rel
assumes
r-refl: reflp-on' domain_f (Restrpf rel domain_f) and
tot-invar-d: total-invariance_D d rel
shows *is-symmetry (closest-preimg-distance f domain_f d) (Invariance rel)*
<proof>

lemma *refl-rel-and-tot-invar-dist-imp-invar-minimizer:*

fixes
f :: 'a \Rightarrow 'b and
domain_f :: 'a set and
d :: 'a Distance and
rel :: 'a rel and
img :: 'b set
assumes
r-refl: reflp-on' domain_f (Restrpf 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 :: 'a \Rightarrow 'b and
domain_f A :: 'a set and
d :: 'a Distance and
img :: 'b set and
G :: 'c monoid and
 φ :: ('c, 'a) binary-fun
defines
rel \equiv action-induced-rel (carrier G) A φ and
rel' \equiv action-induced-rel (carrier G) domain_f φ
assumes
action- φ : group-action G A φ and
domain_f \subseteq A and
closed-domain: closed-restricted-rel rel A domain_f and
invar-d: invariance_D d (carrier G) A φ and

$\text{invar-f: is-symmetry } f \text{ (Invariance rel)}$
shows $\text{is-symmetry (minimizer } f \text{ domain}_f \text{ d img) (Invariance rel)}$
 $\langle \text{proof} \rangle$

5.6.2 Minimizer Translation

lemma $\mathcal{K}_{\mathcal{E}}$ -is-preimg:

fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class and}$
 $E :: ('a, 'v) \text{ Election and}$
 $w :: 'r$
shows $\text{preimg (elect-r } \circ \text{ fun}_{\mathcal{E}} \text{ (rule-}\mathcal{K} \text{ } C)) \text{ (elections-}\mathcal{K} \text{ } C) \{w\} = \mathcal{K}_{\mathcal{E}} \text{ } C \text{ } w$
 $\langle \text{proof} \rangle$

lemma score-is-closest-preimg-dist:

fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class and}$
 $E :: ('a, 'v) \text{ Election and}$
 $w :: 'r$
shows $\text{score } d \text{ } C \text{ } E \text{ } w =$
 $\text{closest-preimg-distance (elect-r } \circ \text{ fun}_{\mathcal{E}} \text{ (rule-}\mathcal{K} \text{ } C)) \text{ (elections-}\mathcal{K} \text{ } C) \text{ } d \text{ } E \text{ } \{w\}$
 $\langle \text{proof} \rangle$

lemma (in result) $\mathcal{R}_{\mathcal{W}}$ -is-minimizer:

fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
shows $\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \text{ } d \text{ } C) =$
 $(\lambda E. \bigcup (\text{minimizer (elect-r } \circ \text{ fun}_{\mathcal{E}} \text{ (rule-}\mathcal{K} \text{ } C)) \text{ (elections-}\mathcal{K} \text{ } C) \text{ } d$
 $\text{(singleton-set-system (limit (alternatives-}\mathcal{E} \text{ } E) \text{ UNIV)) } E))$
 $\langle \text{proof} \rangle$

Invariance

theorem (in result) tot-invar-dist-imp-invar-dr-rule:

fixes
 $d :: ('a, 'v) \text{ Election Distance and}$
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class and}$
 $\text{rel} :: ('a, 'v) \text{ Election rel}$
assumes
 $r\text{-refl: reflp-on' (elections-}\mathcal{K} \text{ } C) \text{ (Restrp rel (elections-}\mathcal{K} \text{ } C)) \text{ and}$
 $\text{tot-invar-d: total-invariance}_{\mathcal{D}} \text{ } d \text{ rel and}$
 invar-res:
 $\text{is-symmetry } (\lambda E. \text{limit (alternatives-}\mathcal{E} \text{ } E) \text{ UNIV}) \text{ (Invariance rel)}$
shows $\text{is-symmetry (fun}_{\mathcal{E}} \text{ (distance-}\mathcal{R} \text{ } d \text{ } C)) \text{ (Invariance rel)}$
 $\langle \text{proof} \rangle$

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**
 $G :: 'b$ monoid **and**
 $\varphi :: ('b, ('a, 'v)$ Election) binary-fun **and**
 $B :: ('a, 'v)$ Election set

defines
 $rel \equiv action-induced-rel (carrier\ G)\ B\ \varphi$ **and**
 $rel' \equiv action-induced-rel (carrier\ G)\ (elections\text{-}\mathcal{K}\ C)\ \varphi$

assumes
 $action\text{-}\varphi$: group-action $G\ B\ \varphi$ **and**
 $consensus\text{-}C\text{-in-}B$: elections- $\mathcal{K}\ C \subseteq B$ **and**
 $closed\text{-}domain$:
 $closed\text{-}restricted\text{-}rel\ rel\ B\ (elections\text{-}\mathcal{K}\ C)$ **and**
 $invar\text{-}res$:
 $is\text{-}symmetry\ (\lambda\ E.\ limit\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)\ (Invariance\ rel)$ **and**
 $invar\text{-}d$: invariance $\mathcal{D}\ d\ (carrier\ G)\ B\ \varphi$ **and**
 $invar\text{-}C\text{-winners}$: $is\text{-}symmetry\ (elect\text{-}r \circ fun_{\mathcal{E}}\ (rule\text{-}\mathcal{K}\ C))\ (Invariance\ rel')$

shows $is\text{-}symmetry\ (fun_{\mathcal{E}}\ (distance\text{-}\mathcal{R}\ d\ C))\ (Invariance\ rel)$
 $\langle proof \rangle$

Equivariance

theorem (in result) $invar\text{-}dist\text{-}equivar\text{-}cons\text{-}imp\text{-}equivar\text{-}dr\text{-}rule$:

fixes
 $d :: ('a, 'v)$ Election Distance **and**
 $C :: ('a, 'v, 'r)$ Result Consensus-Class **and**
 $G :: 'b$ monoid **and**
 $\varphi :: ('b, ('a, 'v)$ Election) binary-fun **and**
 $\psi :: ('b, 'r)$ binary-fun **and**
 $B :: ('a, 'v)$ Election set

defines
 $rel \equiv action-induced-rel (carrier\ G)\ B\ \varphi$ **and**
 $rel' \equiv action-induced-rel (carrier\ G)\ (elections\text{-}\mathcal{K}\ C)\ \varphi$ **and**
 $equivar\text{-}prop \equiv$
 $action-induced-equivariance\ (carrier\ G)\ (elections\text{-}\mathcal{K}\ C)$
 $\varphi\ (set\text{-}action\ \psi)$ **and**
 $equivar\text{-}prop\text{-}global\text{-}set\text{-}valued \equiv$
 $action-induced-equivariance\ (carrier\ G)\ B\ \varphi\ (set\text{-}action\ \psi)$ **and**
 $equivar\text{-}prop\text{-}global\text{-}result\text{-}valued \equiv$
 $action-induced-equivariance\ (carrier\ G)\ B\ \varphi\ (result\text{-}action\ \psi)$

assumes
 $action\text{-}\varphi$: group-action $G\ B\ \varphi$ **and**
 $group\text{-}act\text{-}res$: group-action $G\ UNIV\ \psi$ **and**
 $cons\text{-}elect\text{-}set$: elections- $\mathcal{K}\ C \subseteq B$ **and**
 $closed\text{-}domain$: $closed\text{-}restricted\text{-}rel\ rel\ B\ (elections\text{-}\mathcal{K}\ C)$ **and**
 $equivar\text{-}res$:
 $is\text{-}symmetry\ (\lambda\ E.\ limit\ (alternatives\text{-}\mathcal{E}\ E)\ UNIV)$
 $equivar\text{-}prop\text{-}global\text{-}set\text{-}valued$ **and**

invar-d: invariance_D d (carrier G) B φ and
equivar-C-winners: is-symmetry (elect-r \circ fun_E (rule-K C)) equivar-prop
shows *is-symmetry (fun_E (distance- \mathcal{R} d C)) equivar-prop-global-result-valued*
 <proof>

5.6.3 Inference Rules

theorem (in result) *anon-dist-and-cons-imp-anon-dr:*

fixes

d :: ('a, 'v) Election Distance and
C :: ('a, 'v, 'r Result) Consensus-Class

assumes

anon-d: distance-anonymity' well-formed-elections d and
anon-C: consensus-rule-anonymity' (elections-K C) C and
closed-C: closed-restricted-rel (anonymity_R well-formed-elections)
well-formed-elections (elections-K C)

shows *anonymity' well-formed-elections (distance- \mathcal{R} d C)*

<proof>

theorem (in result-properties) *neutr-dist-and-cons-imp-neutr-dr:*

fixes

d :: ('a, 'v) Election Distance and
C :: ('a, 'v, 'b Result) Consensus-Class

assumes

neutral-d: distance-neutrality well-formed-elections d and
neutral-C: consensus-rule-neutrality (elections-K C) C and
closed-C: closed-restricted-rel (neutrality_R well-formed-elections)
well-formed-elections (elections-K C)

shows *neutrality well-formed-elections (distance- \mathcal{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

reverse-sym-d: distance-reversal-symmetry well-formed-elections d and
reverse-sym-C: consensus-rule-reversal-symmetry (elections-K C) C and
closed-C: closed-restricted-rel (reversal_R well-formed-elections)
well-formed-elections (elections-K C)

shows *reversal-symmetry well-formed-elections (SWF-result.distance- \mathcal{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- \mathcal{V} d*

shows *homogeneity finite-elections- \mathcal{V} (distance- \mathcal{R} d C)*

<proof>

theorem (in result) tot-hom-dist-imp-hom-dr':
 fixes
 $d :: ('a, 'v::linorder) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$
assumes distance-homogeneity' finite-elections- \mathcal{V} d
shows homogeneity' finite-elections- \mathcal{V} (distance- \mathcal{R} d C)
<proof>

5.6.4 Properties

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

5.7 Distance Rationalization on Election Quotients

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

5.7.1 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
 $\pi_Q(\text{tup } d)(A \times B))$)
fun relation-paths :: 'x rel \Rightarrow 'x list set **where**
 relation-paths $r =$
 $\{p. \exists k. (\text{length } p = 2 * k \wedge (\forall i < k. (p!(2 * i), p!(2 * i + 1)) \in r))\}$
fun admissible-paths :: 'x rel \Rightarrow 'x set \Rightarrow 'x set \Rightarrow 'x list set **where**
 admissible-paths r X $Y =$
 $\{x \# p @ [y] \mid x\ y\ p. x \in X \wedge y \in Y \wedge p \in \text{relation-paths } r\}$
fun path-length :: 'x list \Rightarrow 'x Distance \Rightarrow ereal **where**
 path-length [] $d = 0$ |
 path-length [x] $d = 0$ |
 path-length (x # y # xs) $d = d\ x\ y + \text{path-length } xs\ d$

fun *quotient-dist* :: 'x rel \Rightarrow 'x Distance \Rightarrow 'x set Distance **where**
quotient-dist r d A B =
 Inf ($\bigcup \{ \{ \text{path-length } p \ d \mid p. p \in \text{admissible-paths } r \ A \ B \} \}$)

fun *distance-infimum*_Q :: 'x Distance \Rightarrow 'x set Distance **where**
*distance-infimum*_Q d A B = Inf { d a b | a b. a \in A \wedge b \in B }

fun *simple* :: 'x rel \Rightarrow 'x set \Rightarrow 'x Distance \Rightarrow bool **where**
simple r X d =
 ($\forall A \in X // r.$
 ($\exists a \in A. \forall B \in X // r.$
*distance-infimum*_Q d A B = Inf { d a b | b. 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 = { (p₁, p₂). ((fst p₁, fst p₂) \in r \wedge snd p₁ = snd p₂)
 \vee ((snd p₁, snd p₂) \in r \wedge fst p₁ = fst p₂) }

Auxiliary Lemmas

lemma *tot-dist-invariance-is-congruence*:

fixes
 d :: 'x Distance **and**
 r :: 'x rel
shows (total-invariance_D d r) = (tup d respects (product r))
 <proof>

lemma *product-helper*:

fixes
 r :: 'x rel **and**
 X :: 'x set
shows
trans-imp: Relation.trans r \longrightarrow Relation.trans (product r) **and**
refl-imp: refl-on X r \longrightarrow refl-on (X \times X) (product r) **and**
sym: sym-on X r \longrightarrow sym-on (X \times X) (product r)
 <proof>

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_D d r
shows $\forall A \ B. A \in X // r \wedge B \in X // r$

$\longrightarrow (\forall a b. a \in A \wedge b \in B \longrightarrow \text{distance}_{\mathcal{Q}} d A B = d a b)$
 $\langle \text{proof} \rangle$

lemma *relation-paths-subset*:

fixes
 $n :: \text{nat}$ **and**
 $p :: 'x \text{ list}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$
assumes $r \subseteq X \times X$
shows $\forall p. p \in \text{relation-paths } r \longrightarrow (\forall i < \text{length } p. p[i] \in X)$
 $\langle \text{proof} \rangle$

lemma *admissible-path-len*:

fixes
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$ **and**
 $a b :: 'x$ **and**
 $p :: 'x \text{ list}$
assumes *refl-on* $X r$
shows *triangle-ineq* $X d \wedge p \in \text{relation-paths } r \wedge \text{total-invariance}_{\mathcal{D}} d r$
 $\wedge a \in X \wedge b \in X \longrightarrow \text{path-length } (a \# p @ [b]) d \geq d a b$
 $\langle \text{proof} \rangle$

lemma *quotient-dist-coincides-with-dist_Q*:

fixes
 $d :: 'x \text{ Distance}$ **and**
 $r :: 'x \text{ rel}$ **and**
 $X :: 'x \text{ set}$
assumes
 $\text{equiv: equiv } X r$ **and**
 $\text{tri: triangle-ineq } X d$ **and**
 $\text{invar: total-invariance}_{\mathcal{D}} d r$
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
 $\text{equiv-X-r: equiv } X r$ **and**
 $\text{tot-inv-d-r: total-invariance}_{\mathcal{D}} d r$
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
 $A\ B :: 'x\ set$ **and**
 $d :: 'x\ Distance$
shows $Inf\ \{d\ a\ b\ |\ a\ b.\ a \in A \wedge b \in B\} =$
 $Inf\ \{Inf\ \{d\ a\ b\ |\ b.\ b \in B\} \mid a.\ a \in A\}$
 $\langle proof \rangle$

lemma *invar-dist-simple*:
fixes
 $d :: 'y\ Distance$ **and**
 $G :: 'x\ monoid$ **and**
 $Y :: 'y\ set$ **and**
 $\varphi :: ('x, 'y)\ binary\ fun$
assumes
 $action\text{-}\varphi$: $group\text{-}action\ G\ Y\ \varphi$ **and**
 $invar$: $invariance_{\mathcal{D}}\ d\ (carrier\ G)\ Y\ \varphi$
shows $simple\ (action\text{-}induced\text{-}rel\ (carrier\ G)\ Y\ \varphi)\ Y\ d$
 $\langle proof \rangle$

lemma *tot-invar-dist-simple*:
fixes
 $d :: 'x\ Distance$ **and**
 $r :: 'x\ rel$ **and**
 $X :: 'x\ set$
assumes
 $equiv\text{-}on\text{-}X$: $equiv\ X\ r$ **and**
 $invar$: $total\text{-}invariance_{\mathcal{D}}\ d\ r$
shows $simple\ r\ X\ d$
 $\langle proof \rangle$

5.7.2 Consensus and Results

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

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

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:
fixes
 $X\ Y\ Z :: 'x\ set$ **and**
 $r :: 'x\ rel$
assumes
 $equiv\ X\ r$ **and**
 $Y \subseteq X$ **and**
 $Z \subseteq X$ **and**

$Z \in Y // r$ **and**
closed-restricted-rel $r X Y$
shows $Z \subseteq Y$
 ⟨proof⟩

lemma (in result) *limit-invar*:
fixes
 $d :: ('a, 'v)$ *Election Distance* **and**
 $r :: ('a, 'v)$ *Election rel* **and**
 $C :: ('a, 'v, 'r)$ *Result* *Consensus-Class* **and**
 $X A :: ('a, 'v)$ *Election set*
assumes
quot-class: $A \in X // r$ **and**
equiv-rel: *equiv* $X r$ **and**
cons-subset: *elections*- \mathcal{K} $C \subseteq X$ **and**
invar-res: *is-symmetry* $(\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV})$ (*Invariance* r)
shows $\forall a \in A. \text{limit } (\text{alternatives-}\mathcal{E} \ a) \ \text{UNIV} = \text{limit}_{\mathcal{Q}} A \ \text{UNIV}$
 ⟨proof⟩

lemma (in result) *preimg-invar*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $\text{domain}_f X :: 'x$ *set* **and**
 $d :: 'x$ *Distance* **and**
 $r :: 'x$ *rel*
assumes
equiv-rel: *equiv* $X r$ **and**
cons-subset: $\text{domain}_f \subseteq X$ **and**
closed-domain: *closed-restricted-rel* $r X \text{domain}_f$ **and**
invar-f: *is-symmetry* f (*Invariance* (*Restr* $r \text{domain}_f$))
shows $\forall y. (\text{preimg } f \text{domain}_f y) // r = \text{preimg } (\pi_{\mathcal{Q}} f) (\text{domain}_f // r) y$
 ⟨proof⟩

lemma *minimizer-helper*:
fixes
 $f :: 'x \Rightarrow 'y$ **and**
 $\text{domain}_f :: 'x$ *set* **and**
 $d :: 'x$ *Distance* **and**
 $Y :: 'y$ *set* **and**
 $x :: 'x$ **and**
 $y :: 'y$
shows $y \in \text{minimizer } f \text{domain}_f d Y x =$
 $(y \in Y \wedge (\forall y' \in Y.$
 $\text{Inf } (d x \text{ ' } (\text{preimg } f \text{domain}_f y)) \leq \text{Inf } (d x \text{ ' } (\text{preimg } f \text{domain}_f y'))))$
 ⟨proof⟩

lemma *rewr-singleton-set-system-union*:
fixes
 $Y :: 'x$ *set set* **and**

$X :: 'x \text{ set}$
assumes $Y \subseteq \text{singleton-set-system } X$
shows
singleton-set-union: $x \in \bigcup Y \longleftrightarrow \{x\} \in Y$ **and**
obtain-singleton: $A \in \text{singleton-set-system } X \longleftrightarrow (\exists x \in X. A = \{x\})$
 $\langle \text{proof} \rangle$

lemma *union-inf*:
fixes $X :: \text{ereal set set}$
shows $\text{Inf } \{\text{Inf } A \mid A. A \in X\} = \text{Inf } (\bigcup X)$
 $\langle \text{proof} \rangle$

5.7.3 Distance Rationalization

fun (**in result**) $\mathcal{R}_{\mathcal{Q}} :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ set}$ **where**
 $\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A =$
 $\bigcup (\text{minimizer } (\pi_{\mathcal{Q}} (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))) (\text{elections-}\mathcal{K}_{\mathcal{Q}} \ r \ C)$
 $(\text{distance-infimum}_{\mathcal{Q}} \ d) (\text{singleton-set-system } (\text{limit}_{\mathcal{Q}} \ A \ \text{UNIV})) \ A)$

fun (**in result**) $\text{distance-}\mathcal{R}_{\mathcal{Q}} :: ('a, 'v) \text{ Election rel} \Rightarrow ('a, 'v) \text{ Election Distance} \Rightarrow$
 $('a, 'v, 'r \text{ Result}) \text{ Consensus-Class} \Rightarrow ('a, 'v) \text{ Election set} \Rightarrow 'r \text{ Result}$ **where**
 $\text{distance-}\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A =$
 $(\mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\pi_{\mathcal{Q}} (\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ A - \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A,$
 $\{\})$

Proposition 4.17 by Hadjibeyli and Wilson [3].

theorem (**in result**) *invar-dr-simple-dist-imp-quotient-dr-winners*:
fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $C :: ('a, 'v, 'r \text{ Result}) \text{ Consensus-Class}$ **and**
 $r :: ('a, 'v) \text{ Election rel}$ **and**
 $X \ A :: ('a, 'v) \text{ Election set}$
assumes
simple: $\text{simple } r \ X \ d$ **and**
closed-domain: $\text{closed-restricted-rel } r \ X \ (\text{elections-}\mathcal{K} \ C)$ **and**
invar-res:
is-symmetry $(\lambda E. \text{limit } (\text{alternatives-}\mathcal{E} \ E) \ \text{UNIV}) \ (\text{Invariance } r)$ **and**
invar-C: $\text{is-symmetry } (\text{elect-}r \circ \text{fun}_{\mathcal{E}} (\text{rule-}\mathcal{K} \ C))$
 $(\text{Invariance } (\text{Restr } r \ (\text{elections-}\mathcal{K} \ C)))$ **and**
invar-dr: $\text{is-symmetry } (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ (\text{Invariance } r)$ **and**
quot-class: $A \in X // r$ **and**
equiv-rel: $\text{equiv } X \ r$ **and**
cons-subset: $\text{elections-}\mathcal{K} \ C \subseteq X$
shows $\pi_{\mathcal{Q}} (\text{fun}_{\mathcal{E}} (\mathcal{R}_{\mathcal{W}} \ d \ C)) \ A = \mathcal{R}_{\mathcal{Q}} \ r \ d \ C \ A$
 $\langle \text{proof} \rangle$

theorem (**in result**) *invar-dr-simple-dist-imp-quotient-dr*:

```

fixes
   $d :: ('a, 'v)$  Election Distance and
   $C :: ('a, 'v, 'r)$  Result Consensus-Class and
   $r :: ('a, 'v)$  Election rel and
   $X A :: ('a, 'v)$  Election set
assumes
  simple: simple  $r X d$  and
  closed-domain: closed-restricted-rel  $r X$  (elections- $\mathcal{K}$   $C$ ) and
  invar-res:
    is-symmetry ( $\lambda E.$  limit (alternatives- $\mathcal{E}$   $E$ ) UNIV)
    (Invariance  $r$ ) and
  invar-C: is-symmetry (elect- $r \circ \text{fun}_{\mathcal{E}}$  (rule- $\mathcal{K}$   $C$ ))
    (Invariance (Restr  $r$  (elections- $\mathcal{K}$   $C$ ))) and
  invar-dr: is-symmetry ( $\text{fun}_{\mathcal{E}}$  ( $\mathcal{R}_{\mathcal{W}}$   $d C$ )) (Invariance  $r$ ) and
  quot-class:  $A \in X // r$  and
  equiv-rel: equiv  $X r$  and
  cons-subset: elections- $\mathcal{K}$   $C \subseteq X$ 
shows  $\pi_{\mathcal{Q}}$  ( $\text{fun}_{\mathcal{E}}$  (distance- $\mathcal{R}$   $d C$ ))  $A = \text{distance-}\mathcal{R}_{\mathcal{Q}}$   $r d C A$ 
 $\langle \text{proof} \rangle$ 

end

```

5.8 Code Generation Interpretations for Results and Properties

```

theory Interpretation-Code
imports Electoral-Module
  Distance-Rationalization
begin
 $\langle ML \rangle$ 

```

5.8.1 Code Lemmas

Lemmas stating the explicit instantiations of interpreted abstract functions from locales.

```

lemma electoral-module-SCF-code-lemma:
fixes  $m :: ('a, 'v, 'a)$  Result Electoral-Module
shows SCF-result.electoral-module  $m =$ 
  ( $\forall A V p.$  profile  $V A p \longrightarrow \text{well-formed-SCF } A (m V A p)$ )
 $\langle \text{proof} \rangle$ 

```

```

lemma  $\mathcal{R}_{\mathcal{W}}$ -SCF-code-lemma:
fixes
   $d :: ('a, 'v)$  Election Distance and
   $K :: ('a, 'v, 'a)$  Result Consensus-Class and
   $V :: 'v$  set and

```

$A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows $\text{SCF-result}.\mathcal{R}_{\mathcal{W}} \ d \ K \ V \ A \ p =$
 $\text{arg-min-set } (\text{score } d \ K \ (A, V, p)) \ (\text{limit-SCF } A \ \text{UNIV})$
 $\langle \text{proof} \rangle$

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 $\text{SCF-result}.\text{distance-}\mathcal{R} \ d \ K \ V \ A \ p =$
 $(\text{SCF-result}.\mathcal{R}_{\mathcal{W}} \ d \ K \ V \ A \ p,$
 $(\text{limit-SCF } A \ \text{UNIV}) - \text{SCF-result}.\mathcal{R}_{\mathcal{W}} \ d \ K \ V \ A \ p,$
 $\{\})$
 $\langle \text{proof} \rangle$

lemma *$\mathcal{R}_{\mathcal{W}}$ -std-SCF-code-lemma:*

fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows $\text{SCF-result}.\mathcal{R}_{\mathcal{W}}\text{-std} \ d \ K \ V \ A \ p =$
 $\text{arg-min-set } (\text{score-std } d \ K \ (A, V, p)) \ (\text{limit-SCF } A \ \text{UNIV})$
 $\langle \text{proof} \rangle$

lemma *distance- \mathcal{R} -std-SCF-code-lemma:*

fixes
 $d :: ('a, 'v) \text{ Election Distance}$ **and**
 $K :: ('a, 'v, 'a \text{ Result}) \text{ Consensus-Class}$ **and**
 $V :: 'v \text{ set}$ **and**
 $A :: 'a \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$
shows $\text{SCF-result}.\text{distance-}\mathcal{R}\text{-std} \ d \ K \ V \ A \ p =$
 $(\text{SCF-result}.\mathcal{R}_{\mathcal{W}}\text{-std} \ d \ K \ V \ A \ p,$
 $(\text{limit-SCF } A \ \text{UNIV}) - \text{SCF-result}.\mathcal{R}_{\mathcal{W}}\text{-std} \ d \ K \ V \ A \ p,$
 $\{\})$
 $\langle \text{proof} \rangle$

lemma *anonymity-SCF-code-lemma:* $\text{SCF-result.anonymity} =$

$(\lambda \ m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}.$
 $\text{SCF-result.electoral-module } m \wedge$
 $(\forall \ A \ V \ p \ \pi :: ('v \Rightarrow 'v).$
 $\text{bij } \pi \longrightarrow (\text{let } (A', V', q) = (\text{rename } \pi \ (A, V, p)) \text{ in}$
 $\text{profile } V \ A \ p \wedge \text{profile } V' \ A' \ q \longrightarrow m \ V \ A \ p = m \ V' \ A' \ q)))$

<proof>

5.8.2 Interpretation Declarations and Constants

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

```
declare [[lc-add SCF-result.electoral-module electoral-module-SCF-code-lemma]]
declare [[lc-add SCF-result.RW RW-SCF-code-lemma]]
declare [[lc-add SCF-result.RW-std RW-std-SCF-code-lemma]]
declare [[lc-add SCF-result.distance-R distance-R-SCF-code-lemma]]
declare [[lc-add SCF-result.distance-R-std distance-R-std-SCF-code-lemma]]
declare [[lc-add SCF-result.anonymity anonymity-SCF-code-lemma]]
```

Constant aliases to use instead of the interpreted functions.

```
definition RW-SCF-code = SCF-result.RW
definition RW-std-SCF-code = SCF-result.RW-std
definition distance-R-SCF-code = SCF-result.distance-R
definition distance-R-std-SCF-code = SCF-result.distance-R-std
definition electoral-module-SCF-code = SCF-result.electoral-module
definition anonymity-SCF-code = SCF-result.anonymity
```

<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 =
```

$$\begin{aligned}
&(\{\}, \\
&\{a \in A. \text{rank } (\text{limit } A \ r) \ a \leq n\}, \\
&\{a \in A. \text{rank } (\text{limit } A \ r) \ a > n\})
\end{aligned}$$

5.9.2 Soundness

theorem *drop-mod-sound*[simp]:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $n :: \text{nat}$
shows *SCF-result.electoral-module* (*drop-module* $n \ r$)
 $\langle \text{proof} \rangle$

lemma *voters-determine-drop-mod*:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $n :: \text{nat}$
shows *voters-determine-election* (*drop-module* $n \ r$)
 $\langle \text{proof} \rangle$

5.9.3 Non-Electing

The drop module is non-electing.

theorem *drop-mod-non-electing*[simp]:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $n :: \text{nat}$
shows *non-electing* (*drop-module* $n \ r$)
 $\langle \text{proof} \rangle$

5.9.4 Properties

The drop module is strictly defer-monotone.

theorem *drop-mod-def-lift-inv*[simp]:
fixes
 $r :: 'a \text{ Preference-Relation}$ **and**
 $n :: \text{nat}$
shows *defer-lift-invariance* (*drop-module* $n \ r$)
 $\langle \text{proof} \rangle$

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

```
lemma voters-determine-pass-mod:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows voters-determine-election (pass-module n r)
   $\langle$ proof $\rangle$ 
```

5.10.3 Non-Blocking

The pass module is non-blocking.

```
theorem pass-mod-non-blocking[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  assumes
    order: linear-order r and
    greater-zero: n  $>$  0
  shows non-blocking (pass-module n r)
   $\langle$ proof $\rangle$ 
```

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 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 Preference-Relation  
  assumes linear-order r  
  shows defers 1 (pass-module 1 r)  
  ⟨proof⟩
```

```
theorem pass-two-mod-def-two:  
  fixes r :: 'a 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 Result) Electoral-Module where
  elect-module V A p = (A, {}, {})
```

5.11.2 Soundness

```
theorem elect-mod-sound[simp]: SCF-result.electoral-module elect-module
  <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 *set* **and**

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

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

assumes

non-empty-A: *A* ≠ {} **and**

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

prof: *profile V A p*

shows *plurality V A p* = *plurality' V A p*

⟨*proof*⟩

5.12.2 Soundness

theorem *plurality-sound[simp]*: *SCF-result.electoral-module plurality*
 ⟨*proof*⟩

theorem *plurality'-sound[simp]*: *SCF-result.electoral-module plurality'*
 ⟨*proof*⟩

lemma *voters-determine-plurality-score*: *voters-determine-evaluation plurality-score*
 ⟨*proof*⟩

lemma *voters-determine-plurality*: *voters-determine-election plurality*
 ⟨*proof*⟩

5.12.3 Non-Blocking

The plurality module is non-blocking.

theorem *plurality-mod-non-blocking[simp]: non-blocking plurality*
⟨*proof*⟩

5.12.4 Non-Electing

The plurality module is non-electing.

theorem *plurality-non-electing[simp]: non-electing plurality*
⟨*proof*⟩

theorem *plurality'-non-electing[simp]: non-electing plurality'*
⟨*proof*⟩

5.12.5 Property

lemma *plurality-def-inv-mono-alts:*

fixes

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

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

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

$a :: 'a$

assumes

defer-a: $a \in \text{defer plurality } V \ A \ p$ **and**

lift-a: $\text{lifted } V \ A \ p \ q \ a$

shows $\text{defer plurality } V \ A \ q = \text{defer plurality } V \ A \ p$
 $\vee \text{defer plurality } V \ A \ q = \{a\}$

⟨*proof*⟩

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

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 =
 card {y ∈ A . wins V x p y} − card {y ∈ A . wins V y p x}

fun *copeland* :: ('a, 'v, 'a *Result*) *Electoral-Module* **where**
copeland V A p = max-eliminator copeland-score V A p

5.15.2 Soundness

theorem *copeland-sound: SCF-result.electoral-module copeland*
<proof>

5.15.3 Lemmas

lemma *voters-determine-copeland-score: voters-determine-evaluation copeland-score*
<proof>

theorem *voters-determine-copeland: voters-determine-election copeland*
<proof>

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

lemma *cond-winner-imp-win-count:*

fixes

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

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

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

$w :: 'a$

assumes *condorcet-winner* $V A p w$

shows $\text{card } \{a \in A . \text{wins } V w p a\} = \text{card } A - 1$

<proof>

For a Condorcet winner w , we have: " $|\{y \in A . \text{wins } V y p w\}| = 0$ ".

lemma *cond-winner-imp-loss-count:*

fixes

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

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

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

$w :: 'a$

assumes *condorcet-winner* $V A p w$

shows $\text{card } \{a \in A . \text{wins } V a p w\} = 0$

<proof>

Copeland score of a Condorcet winner.

lemma *cond-winner-imp-copeland-score:*

fixes

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

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

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

$w :: 'a$

assumes *condorcet-winner* $V A p w$

shows *copeland-score* $V w A p = \text{card } A - 1$

<proof>

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

lemma *non-cond-winner-imp-win-count:*


```

fixes
   $A :: 'a \text{ set}$  and
   $V :: 'v \text{ set}$  and
   $p :: ('a, 'v) \text{ Profile}$  and
   $w \ l :: 'a$ 
assumes
  winner: condorcet-winner  $V \ A \ p \ w$  and
  loser:  $l \neq w$  and
  l-in-A:  $l \in A$ 
shows  $\text{card } \{a \in A . \text{wins } V \ l \ p \ a\} \leq \text{card } A - 2$ 
 $\langle \text{proof} \rangle$ 

```

5.15.4 Property

The Copeland score is Condorcet rating.

theorem *copeland-score-is-cr: condorcet-rating copeland-score*
 $\langle \text{proof} \rangle$

theorem *copeland-is-dcc: defer-condorcet-consistency copeland*
 $\langle \text{proof} \rangle$

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* :: $('a, 'v) \text{ Evaluation-Function}$ **where**
 $\text{minimax-score } V \ x \ A \ p =$
 $\text{Min } \{\text{prefer-count } V \ p \ x \ y \mid y . y \in A - \{x\}\}$

fun *minimax* :: $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **where**
 $\text{minimax } A \ p = \text{max-eliminator minimax-score } A \ p$

5.16.2 Soundness

theorem *minimax-sound: SCF-result.electoral-module minimax*
<proof>

5.16.3 Lemma

lemma *non-cond-winner-minimax-score:*

fixes

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

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

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

$w \ l :: 'a$

assumes

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

winner: *condorcet-winner* $V \ A \ p \ w$ **and**

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

l-neg-w: $l \neq w$

shows *minimax-score* $V \ l \ A \ p \leq \text{prefer-count } V \ p \ l \ w$

<proof>

5.16.4 Property

theorem *minimax-score-cond-rating: condorcet-rating minimax-score*

<proof>

theorem *minimax-is-dcc: defer-condorcet-consistency minimax*

<proof>

end

Chapter 6

Compositional Structures

6.1 Drop- and Pass-Compatibility

```
theory Drop-And-Pass-Compatibility
  imports Basic-Modules/Drop-Module
           Basic-Modules/Pass-Module
begin
```

This is a collection of properties about the interplay and compatibility of both the drop module and the pass module.

```
theorem drop-zero-mod-rej-zero[simp]:
  fixes  $r :: 'a \text{ Preference-Relation}$ 
  assumes 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
   $m\downarrow \equiv \text{revision-composition } m$ 
```

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)
   $\langle \text{proof} \rangle$ 
```

```
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)
   $\langle \text{proof} \rangle$ 
```

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 ( $m\downarrow$ )
   $\langle \text{proof} \rangle$ 
```

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* $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \text{ where}$
 $\text{sequential-composition } m \ n \ V \ A \ p =$
 $(\text{let } \text{new-A} = \text{defer } m \ V \ A \ p;$
 $\text{new-p} = \text{limit-profile } \text{new-A} \ p \text{ in } ($
 $\text{elect } m \ V \ A \ p) \cup (\text{elect } n \ V \ \text{new-A} \ \text{new-p}),$
 $(\text{reject } m \ V \ A \ p) \cup (\text{reject } n \ V \ \text{new-A} \ \text{new-p}),$
 $\text{defer } n \ V \ \text{new-A} \ \text{new-p}))$

abbreviation *sequence* $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
 $(\text{infix } \triangleright \ 50) \text{ where}$
 $m \triangleright n \equiv \text{sequential-composition } m \ n$

fun *sequential-composition'* $:: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$

$(\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module} \Rightarrow$
 $(\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module} \text{ where}$
 $\text{sequential-composition}' m n V A p =$
 $(\text{let } (m\text{-e}, m\text{-r}, m\text{-d}) = m \text{ } V \text{ } A \text{ } p; \text{ new-A} = m\text{-d};$
 $\text{ new-p} = \text{limit-profile new-A } p;$
 $(n\text{-e}, n\text{-r}, n\text{-d}) = n \text{ } V \text{ new-A new-p in}$
 $(m\text{-e} \cup n\text{-e}, m\text{-r} \cup n\text{-r}, n\text{-d}))$

lemma *voters-determine-seq-comp*:

fixes $m n :: (\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module}$
assumes *voters-determine-election* $m \wedge \text{voters-determine-election } n$
shows *voters-determine-election* $(m \triangleright n)$
 $\langle \text{proof} \rangle$

lemma *seq-comp-presv-disj*:

fixes
 $m n :: (\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module}$ **and**
 $A :: \text{'a set}$ **and**
 $V :: \text{'v set}$ **and**
 $p :: (\text{'a}, \text{'v}) \text{ Profile}$
assumes
 $\text{module-m: SCF-result.electoral-module } m$ **and**
 $\text{module-n: SCF-result.electoral-module } n$ **and**
 $\text{prof: profile } V \text{ } A \text{ } p$
shows *disjoint3* $((m \triangleright n) \text{ } V \text{ } A \text{ } p)$
 $\langle \text{proof} \rangle$

lemma *seq-comp-presv-alt*:

fixes
 $m n :: (\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module}$ **and**
 $A :: \text{'a set}$ **and**
 $V :: \text{'v set}$ **and**
 $p :: (\text{'a}, \text{'v}) \text{ Profile}$
assumes
 $\text{module-m: SCF-result.electoral-module } m$ **and**
 $\text{module-n: SCF-result.electoral-module } n$ **and**
 $\text{prof: profile } V \text{ } A \text{ } p$
shows *set-equals-partition* $A ((m \triangleright n) \text{ } V \text{ } A \text{ } p)$
 $\langle \text{proof} \rangle$

lemma *seq-comp-alt-eq*[*fundef-cong*, *code*]: *sequential-composition* = *sequential-composition'*
 $\langle \text{proof} \rangle$

6.3.2 Soundness

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

fixes $m n :: (\text{'a}, \text{'v}, \text{'a Result}) \text{ Electoral-Module}$
assumes
 $\text{SCF-result.electoral-module } m$ **and**

$SCF\text{-}result.electoral\text{-}module\ n$
shows $SCF\text{-}result.electoral\text{-}module\ (m \triangleright n)$
 $\langle proof \rangle$

6.3.3 Lemmas

lemma *seq-comp-decrease-only-defer*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $module\text{-}m: SCF\text{-}result.electoral\text{-}module\ m$ **and**
 $module\text{-}n: SCF\text{-}result.electoral\text{-}module\ n$ **and**
 $prof: profile\ V\ A\ p$ **and**
 $empty\text{-}defer: defer\ m\ V\ A\ p = \{\}$
shows $(m \triangleright n)\ V\ A\ p = m\ V\ A\ p$
 $\langle proof \rangle$

lemma *seq-comp-def-then-elect*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $n\text{-electing}\text{-}m: non\text{-}electing\ m$ **and**
 $def\text{-}one\text{-}m: defers\ 1\ m$ **and**
 $electing\text{-}n: electing\ n$ **and**
 $f\text{-}prof: finite\text{-}profile\ V\ A\ p$
shows $elect\ (m \triangleright n)\ V\ A\ p = defer\ m\ V\ A\ p$
 $\langle proof \rangle$

lemma *seq-comp-def-card-bounded*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $SCF\text{-}result.electoral\text{-}module\ m$ **and**
 $SCF\text{-}result.electoral\text{-}module\ n$ **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

$m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $SCF\text{-}result.electoral\text{-}module\ m$ **and**
 $SCF\text{-}result.electoral\text{-}module\ n$ **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
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
shows $defer\ (m \triangleright n)\ V\ A\ p =$
 $defer\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p)$
 $\langle proof \rangle$

lemma *seq-comp-def-then-elect-elec-set*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
shows $elect\ (m \triangleright n)\ V\ A\ p =$
 $elect\ n\ V\ (defer\ m\ V\ A\ p)\ (limit\text{-}profile\ (defer\ m\ V\ A\ p)\ p) \cup (elect\ m\ V\ A\ p)$
 $\langle proof \rangle$

lemma *seq-comp-elim-one-red-def-set*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$
assumes
 $SCF\text{-}result.electoral\text{-}module\ m$ **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
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**

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

6.3.4 Composition Rules

The sequential composition preserves the non-blocking property.

theorem *seq-comp-presv-non-blocking[simp]*:
fixes $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{non-blocking-}m$: $\text{non-blocking } m$ **and**
 $\text{non-blocking-}n$: $\text{non-blocking } n$
shows $\text{non-blocking } (m \triangleright n)$
 $\langle \text{proof} \rangle$

Sequential composition preserves the non-electing property.

theorem *seq-comp-presv-non-electing[simp]*:
fixes $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{non-electing } m$ **and**
 $\text{non-electing } n$
shows $\text{non-electing } (m \triangleright n)$
 $\langle \text{proof} \rangle$

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 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{def-one-}m$: $\text{defers } 1 \ m$ **and**
 $\text{electing-}n$: $\text{electing } n$
shows $\text{electing } (m \triangleright n)$
 $\langle \text{proof} \rangle$

lemma *def-lift-inv-seq-comp-help*:
fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**

$p \ q :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
monotone-m: defer-lift-invariance m and
monotone-n: defer-lift-invariance n and
voters-determine-n: voters-determine-election n and
def-and-lifted: $a \in (\text{defer } (m \triangleright n) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a$
shows $(m \triangleright n) \ V \ A \ p = (m \triangleright n) \ V \ A \ q$
 <proof>

Sequential composition preserves the property defer-lift-invariance.

theorem *seq-comp-presv-def-lift-inv[simp]:*
fixes $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
defer-lift-invariance m and
defer-lift-invariance n and
voters-determine-election n
shows *defer-lift-invariance $(m \triangleright n)$*
 <proof>

Composing a non-blocking, non-electing electoral module in sequence with an electoral module that defers exactly one alternative results in an electoral module that defers exactly one alternative.

theorem *seq-comp-def-one[simp]:*
fixes $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
non-blocking-m: non-blocking m and
non-electing-m: non-electing m and
def-one-n: defers 1 n
shows *defers 1 $(m \triangleright n)$*
 <proof>

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *disj-compat-seq[simp]:*
fixes $m \ m' \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
compatible: disjoint-compatibility m n and
module-m': SCF-result.electoral-module m' and
voters-determine-m': voters-determine-election m'
shows *disjoint-compatibility $(m \triangleright m') \ n$*
 <proof>

theorem *seq-comp-cond-compat[simp]:*
fixes $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
dcc-m: defer-condorcet-consistency m and

nb-n: non-blocking n and
ne-n: non-electing n
shows *condorcet-compatibility* ($m \triangleright n$)
 <proof>

Composing a defer-condorcet-consistent electoral module in sequence with a non-blocking and non-electing electoral module results in a defer-condorcet-consistent module.

theorem *seq-comp-dcc[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
dcc-m: defer-condorcet-consistency m and
nb-n: non-blocking n and
ne-n: non-electing n
shows *defer-condorcet-consistency* ($m \triangleright n$)
 <proof>

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *seq-comp-mono[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
def-monotone-m: defer-lift-invariance m and
non-ele-m: non-electing m and
def-one-m: defers 1 m and
electing-n: electing n
shows *monotonicity* ($m \triangleright n$)
 <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 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
strong-def-mon-m: defer-invariant-monotonicity m and
non-electing-n: non-electing n and
defers-one: defers 1 n and
defer-monotone-n: defer-monotonicity n and
voters-determine-n: voters-determine-election n
shows *defer-lift-invariance* ($m \triangleright n$)
 <proof>

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  $\equiv$  parallel-composition m n a

```

6.4.2 Soundness

```

theorem par-comp-sound[simp]:
  fixes
    m n :: ('a, 'v, 'a Result) Electoral-Module and
    a :: 'a Aggregator
  assumes
    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 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 acc :: ('a, 'v, 'a Result) Electoral-Module and
    t :: 'a Termination-Condition and
    A :: 'a set and
    V :: 'v set and
    p :: ('a, 'v) Profile
  assumes
    ¬ t (acc V A p) and
    defer (acc ▷ m) V A p ⊂ defer acc V A p and
    finite (defer acc V A p)
  shows ((acc ▷ m, m, t, V, A, p), (acc, m, t, V, A, p)) ∈
    measure (λ (acc, m, t, V, A, p). card (defer acc V A p))
  <proof>

```

This function handles the accumulator for the following loop composition function.

```

function loop-comp-helper :: ('a, 'v, 'a Result) Electoral-Module ⇒
  ('a, 'v, 'a Result) Electoral-Module ⇒ 'a Termination-Condition ⇒

```

('a, 'v, 'a Result) Electoral-Module **where**
 loop-comp-helper-finite:
 finite (defer acc V A p) \wedge (defer (acc \triangleright m) V A p) \subset (defer acc V A p)
 $\longrightarrow t$ (acc V A p) \implies
 loop-comp-helper acc m t V A p = acc V A p |
 loop-comp-helper-infinite:
 \neg (finite (defer acc V A p) \wedge (defer (acc \triangleright m) V A p) \subset (defer acc V A p))
 $\longrightarrow t$ (acc V A p) \implies
 loop-comp-helper acc m t V A p = loop-comp-helper (acc \triangleright m) m t V A p
 <proof>
termination
 <proof>

lemma loop-comp-code-helper[code]:
fixes
 m acc :: ('a, 'v, 'a Result) Electoral-Module **and**
 t :: 'a Termination-Condition **and**
 A :: 'a set **and**
 V :: 'v set **and**
 p :: ('a, 'v) Profile
shows
 loop-comp-helper acc m t V A p =
 (if (t (acc V A p) \vee \neg ((defer (acc \triangleright m) V A p) \subset (defer acc V A p))
 \vee infinite (defer acc V A p))
 then (acc V A p) else (loop-comp-helper (acc \triangleright m) m t V A p))
 <proof>

function loop-composition :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow
 'a Termination-Condition \Rightarrow ('a, 'v, 'a Result) Electoral-Module **where**
 t ({}, {}, A)
 \implies loop-composition m t V A p = defer-module V A p |
 \neg (t ({}, {}, A))
 \implies loop-composition m t V A p = (loop-comp-helper m m t) V A p
 <proof>
termination
 <proof>

abbreviation loop :: ('a, 'v, 'a Result) Electoral-Module \Rightarrow
 'a Termination-Condition \Rightarrow ('a, 'v, 'a Result) Electoral-Module
 (- \odot 50) **where**
 m $\odot_t \equiv$ loop-composition m t

lemma loop-comp-code[code]:
fixes
 m :: ('a, 'v, 'a Result) Electoral-Module **and**
 t :: 'a Termination-Condition **and**
 A :: 'a set **and**
 V :: 'v set **and**
 p :: ('a, 'v) Profile

shows *loop-composition* $m\ t\ V\ A\ p =$
 (*if* ($t\ (\{\},\{\},A)$)
 then (*defer-module* $V\ A\ p$) *else* (*loop-comp-helper* $m\ m\ t$) $V\ A\ p$)
<proof>

lemma *loop-comp-helper-imp-partit:*

fixes
 $m\ acc :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $t :: 'a\ Termination\ Condition$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$ **and**
 $n :: nat$
assumes
 module-m: $SCF\ result.electoral\ module\ m$ **and**
 profile: $profile\ V\ A\ p$ **and**
 module-acc: $SCF\ result.electoral\ module\ acc$ **and**
 defer-card-n: $n = card\ (defer\ acc\ V\ A\ p)$
shows *well-formed-SCF* $A\ (loop-comp-helper\ acc\ m\ t\ V\ A\ p)$
<proof>

6.5.2 Soundness

theorem *loop-comp-sound:*

fixes
 $m :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $t :: 'a\ Termination\ Condition$
assumes $SCF\ result.electoral\ module\ m$
shows $SCF\ result.electoral\ module\ (m\ \odot_t)$
<proof>

lemma *loop-comp-helper-imp-no-def-incr:*

fixes
 $m\ acc :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $t :: 'a\ Termination\ Condition$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$ **and**
 $n :: nat$
assumes
 module-m: $SCF\ result.electoral\ module\ m$ **and**
 profile: $profile\ V\ A\ p$ **and**
 mod-acc: $SCF\ result.electoral\ module\ acc$ **and**
 card-n-defer-acc: $n = card\ (defer\ acc\ V\ A\ p)$
shows $defer\ (loop-comp-helper\ acc\ m\ t)\ V\ A\ p \subseteq defer\ acc\ V\ A\ p$
<proof>

6.5.3 Lemmas

lemma *loop-comp-helper-def-lift-inv-helper:*

fixes
m acc :: ('a, 'v, 'a Result) Electoral-Module **and**
t :: 'a Termination-Condition **and**
A :: 'a set **and**
V :: 'v set **and**
p :: ('a, 'v) Profile **and**
n :: nat
assumes
monotone-m: defer-lift-invariance *m* **and**
prof: profile *V A p* **and**
dli-acc: defer-lift-invariance *acc* **and**
card-n-defer: *n* = card (defer *acc V A p*) **and**
defer-finite: finite (defer *acc V A p*) **and**
voters-determine-m: voters-determine-election *m*
shows
 $\forall q a. a \in (\text{defer } (\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ p) \wedge \text{lifted } V \ A \ p \ q \ a \longrightarrow$
 $(\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ p = (\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ q$
 <proof>

lemma *loop-comp-helper-def-lift-inv*:

fixes
m acc :: ('a, 'v, 'a Result) Electoral-Module **and**
t :: 'a Termination-Condition **and**
A :: 'a set **and**
V :: 'v set **and**
p q :: ('a, 'v) Profile **and**
a :: 'a
assumes
defer-lift-invariance m **and**
voters-determine-election m **and**
defer-lift-invariance acc **and**
profile V A p **and**
lifted V A p q a **and**
 $a \in \text{defer } (\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ p$
shows $(\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ p = (\text{loop-comp-helper } \text{acc } m \ t) \ V \ A \ q$
 <proof>

lemma *lifted-imp-fin-prof*:

fixes
A :: 'a set **and**
V :: 'v set **and**
p q :: ('a, 'v) Profile **and**
a :: 'a
assumes *lifted V A p q a*
shows *finite-profile V A p*
 <proof>

lemma *loop-comp-helper-presv-def-lift-inv*:

fixes

$m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$
assumes
 $\text{defer-lift-invariance } m$ **and**
 $\text{voters-determine-election } m$ **and**
 $\text{defer-lift-invariance } acc$
shows $\text{defer-lift-invariance } (\text{loop-comp-helper } acc \ m \ t)$
 $\langle \text{proof} \rangle$

lemma $\text{loop-comp-presv-non-electing-helper}$:
fixes
 $m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $n :: \text{nat}$
assumes
 $\text{non-electing-}m$: $\text{non-electing } m$ **and**
 $\text{non-electing-}acc$: $\text{non-electing } acc$ **and**
 prof : $\text{profile } V \ A \ p$ **and**
 acc-defer-card : $n = \text{card } (\text{defer } acc \ V \ A \ p)$
shows $\text{elect } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p = \{\}$
 $\langle \text{proof} \rangle$

lemma $\text{loop-comp-helper-iter-elim-def-n-helper}$:
fixes
 $m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $n \ x :: \text{nat}$
assumes
 $\text{non-electing-}m$: $\text{non-electing } m$ **and**
 $\text{single-elimination}$: $\text{eliminates } 1 \ m$ **and**
 $\text{terminate-if-n-left}$: $\forall \ r. \ t \ r = (\text{card } (\text{defer-r } r) = x)$ **and**
 x-greater-zero : $x > 0$ **and**
 prof : $\text{profile } V \ A \ p$ **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 : $\text{non-electing } acc$
shows $\text{card } (\text{defer } (\text{loop-comp-helper } acc \ m \ t) \ V \ A \ p) = x$
 $\langle \text{proof} \rangle$

lemma $\text{loop-comp-helper-iter-elim-def-n}$:
fixes

$m \text{ acc} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $x :: \text{nat}$
assumes
 $\text{non-electing } m$ **and**
 $\text{eliminates } 1 \ m$ **and**
 $\forall r. (t \ r) = (\text{card } (\text{defer-r } r) = x)$ **and**
 $x > 0$ **and**
 $\text{profile } V \ A \ p$ **and**
 $\text{card } (\text{defer acc } V \ A \ p) \geq x$ **and**
 non-electing acc
shows $\text{card } (\text{defer } (\text{loop-comp-helper acc } m \ t) \ V \ A \ p) = x$
 $\langle \text{proof} \rangle$

lemma *iter-elim-def-n-helper*:

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**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $x :: \text{nat}$
assumes
 $\text{non-electing-m: non-electing } m$ **and**
 $\text{single-elimination: eliminates } 1 \ m$ **and**
 $\text{terminate-if-n-left: } \forall r. (t \ r) = (\text{card } (\text{defer-r } r) = x)$ **and**
 $\text{x-greater-zero: } x > 0$ **and**
 $\text{prof: profile } V \ A \ p$ **and**
 $\text{enough-alternatives: card } A \geq x$
shows $\text{card } (\text{defer } (m \ \odot_t) \ V \ A \ p) = x$
 $\langle \text{proof} \rangle$

6.5.4 Composition Rules

The loop composition preserves defer-lift-invariance.

theorem *loop-comp-presv-def-lift-inv[simp]*:

fixes
 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $t :: 'a \text{ Termination-Condition}$
assumes
 $\text{defer-lift-invariance } m$ **and**
 $\text{voters-determine-election } m$
shows $\text{defer-lift-invariance } (m \ \odot_t)$
 $\langle \text{proof} \rangle$

The loop composition preserves the property non-electing.

```

theorem loop-comp-presv-non-electing[simp]:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $t :: 'a \text{ Termination-Condition}$ 
  assumes non-electing m
  shows non-electing (m  $\odot_t$ )
   $\langle \text{proof} \rangle$ 

theorem iter-elim-def-n[simp]:
  fixes
     $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$  and
     $t :: 'a \text{ Termination-Condition}$  and
     $n :: \text{nat}$ 
  assumes
    non-electing-m: non-electing m and
    single-elimination: eliminates 1 m and
    terminate-if-n-left:  $\forall r. t\ r = (\text{card} (\text{defer-r } r) = n)$  and
    x-greater-zero:  $n > 0$ 
  shows defers n (m  $\odot_t$ )
   $\langle \text{proof} \rangle$ 

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

```

maximum-parallel-composition $m\ n =$
 (let $a = \text{max-aggregator}$ in $(m \parallel_a n)$)

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

6.6.2 Soundness

theorem *max-par-comp-sound*:
fixes $m\ n :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
assumes
 SCF-result.electoral-module m **and**
 SCF-result.electoral-module n
shows *SCF-result.electoral-module* $(m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

lemma *voters-determine-max-par-comp*:
fixes $m\ n :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$
assumes
 voters-determine-election m **and**
 voters-determine-election n
shows *voters-determine-election* $(m \parallel_{\uparrow} n)$
 $\langle \text{proof} \rangle$

6.6.3 Lemmas

lemma *max-agg-eq-result*:
fixes
 $m\ n :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$ **and**
 $A :: 'a\ \text{set}$ **and**
 $V :: 'v\ \text{set}$ **and**
 $p :: ('a, 'v)\ \text{Profile}$ **and**
 $a :: 'a$
assumes
 module-m: *SCF-result.electoral-module* m **and**
 module-n: *SCF-result.electoral-module* n **and**
 prof-p: *profile* $V\ A\ p$ **and**
 a-in-A: $a \in A$
shows *mod-contains-result* $(m \parallel_{\uparrow} n)\ m\ V\ A\ p\ a \vee$
 mod-contains-result $(m \parallel_{\uparrow} n)\ n\ V\ A\ p\ a$
 $\langle \text{proof} \rangle$

lemma *max-agg-rej-iff-both-reject*:
fixes
 $m\ n :: ('a, 'v, 'a\ \text{Result})\ \text{Electoral-Module}$ **and**
 $A :: 'a\ \text{set}$ **and**
 $V :: 'v\ \text{set}$ **and**
 $p :: ('a, 'v)\ \text{Profile}$ **and**

$a :: 'a$
assumes
finite-profile $V A p$ **and**
SCF-result.electoral-module m **and**
SCF-result.electoral-module n
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
 $m n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
f-prof: *finite-profile* $V A p$ **and**
module-m: *SCF-result.electoral-module* m **and**
module-n: *SCF-result.electoral-module* n **and**
rejected: $a \in \text{reject } n V A p$
shows *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
 $m n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
finite-profile $V A p$ **and**
SCF-result.electoral-module m **and**
SCF-result.electoral-module n **and**
 $a \in \text{reject } n V A p$
shows *mod-contains-result-sym* $(m \parallel_{\uparrow} n) m V A p a$
 $\langle \text{proof} \rangle$

lemma *max-agg-rej-snd-imp-seq-contained:*
fixes
 $m n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **and**
 $A :: 'a \text{ set}$ **and**
 $V :: 'v \text{ set}$ **and**
 $p :: ('a, 'v) \text{ Profile}$ **and**
 $a :: 'a$
assumes
f-prof: *finite-profile* $V A p$ **and**
module-m: *SCF-result.electoral-module* m **and**

module-n: SCF-result.electoral-module n and
rejected: $a \in \text{reject } m \ V \ A \ p$
shows *mod-contains-result $n \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a$*
<proof>

lemma *max-agg-rej-snd-equiv-seq-contained:*

fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ and
 $A :: 'a \text{ set}$ and
 $V :: 'v \text{ set}$ and
 $p :: ('a, 'v) \text{ Profile}$ and
 $a :: 'a$
assumes
finite-profile $V \ A \ p$ and
SCF-result.electoral-module m and
SCF-result.electoral-module n and
 $a \in \text{reject } m \ V \ A \ p$
shows *mod-contains-result-sym $(m \parallel_{\uparrow} n) \ n \ V \ A \ p \ a$*
<proof>

lemma *max-agg-rej-intersect:*

fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ and
 $A :: 'a \text{ set}$ and
 $V :: 'v \text{ set}$ and
 $p :: ('a, 'v) \text{ Profile}$
assumes
SCF-result.electoral-module m and
SCF-result.electoral-module n and
profile $V \ A \ p$ and
finite A
shows *$\text{reject } (m \parallel_{\uparrow} n) \ V \ A \ p = (\text{reject } m \ V \ A \ p) \cap (\text{reject } n \ V \ A \ p)$*
<proof>

lemma *dcompat-dec-by-one-mod:*

fixes
 $m \ n :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ and
 $A :: 'a \text{ set}$ and
 $V :: 'v \text{ set}$ and
 $a :: 'a$
assumes
disjoint-compatibility $m \ n$ and
 $a \in A$
shows
 $(\forall p. \text{finite-profile } V \ A \ p \longrightarrow \text{mod-contains-result } m \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a)$
 $\vee (\forall p. \text{finite-profile } V \ A \ p \longrightarrow \text{mod-contains-result } n \ (m \parallel_{\uparrow} n) \ V \ A \ p \ a)$
<proof>

6.6.4 Composition Rules

Using a conservative aggregator, the parallel composition preserves the property non-electing.

theorem *conserv-max-agg-presv-non-electing[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
 non-electing m **and**
 non-electing n
shows *non-electing* $(m \parallel_{\uparrow} n)$
<proof>

Using the max aggregator, composing two compatible electoral modules in parallel preserves defer-lift-invariance.

theorem *par-comp-def-lift-inv[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
 compatible: disjoint-compatibility $m\ n$ **and**
 monotone-m: defer-lift-invariance m **and**
 monotone-n: defer-lift-invariance n
shows *defer-lift-invariance* $(m \parallel_{\uparrow} n)$
<proof>

lemma *par-comp-rej-card*:
fixes
 $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$ **and**
 $A :: 'a\ set$ **and**
 $V :: 'v\ set$ **and**
 $p :: ('a, 'v)\ Profile$ **and**
 $c :: nat$
assumes
 compatible: disjoint-compatibility $m\ n$ **and**
 prof: profile $V\ A\ p$ **and**
 fin-A: finite A **and**
 reject-sum: card (reject $m\ V\ A\ p) + card (reject\ n\ V\ A\ p) = card\ A + c$
shows *card (reject* $(m \parallel_{\uparrow} n)\ V\ A\ p) = c$
<proof>

Using the max-aggregator for composing two compatible modules in parallel, whereof the first one is non-electing and defers exactly one alternative, and the second one rejects exactly two alternatives, the composition results in an electoral module that eliminates exactly one alternative.

theorem *par-comp-elim-one[simp]*:
fixes $m\ n :: ('a, 'v, 'a\ Result)\ Electoral\ Module$
assumes
 defers-m-one: defers $1\ m$ **and**
 non-elec-m: non-electing m **and**
 rejec-n-two: rejects $2\ n$ **and**

```

    disj-comp: disjoint-compatibility m n
    shows eliminates 1 (m ||↑ n)
    ⟨proof⟩

end

```

6.7 Elect Composition

```

theory Elect-Composition
  imports Basic-Modules/Elect-Module
           Sequential-Composition
begin

```

The elect composition sequences an electoral module and the elect module. It finalizes the module's decision as it simply elects all their non-rejected alternatives. Thereby, any such elect-composed module induces a proper voting rule in the social choice sense, as all alternatives are either rejected or elected.

6.7.1 Definition

```

fun elector :: ('a, 'v, 'a Result) Electoral-Module ⇒
      ('a, 'v, 'a Result) Electoral-Module where
  elector m = (m ▷ elect-module)

```

6.7.2 Auxiliary Lemmas

```

lemma elector-seqcomp-assoc:
  fixes a b :: ('a, 'v, 'a Result) Electoral-Module
  shows (a ▷ (elector b)) = (elector (a ▷ b))
  ⟨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-elect:
  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 $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes
 $\text{module-}m: \text{SCF-result.electoral-module } m$ **and**
 $\text{non-block-}m: \text{non-blocking } m$
shows $\text{electing } (\text{elector } m)$
 $\langle \text{proof} \rangle$

6.7.5 Composition Rule

If m is defer-Condorcet-consistent, then $\text{elector}(m)$ is Condorcet consistent.

lemma *dcc-imp-cc-elector*:
fixes $m :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$
assumes $\text{defer-condorcet-consistency } m$
shows $\text{condorcet-consistency } (\text{elector } m)$
 $\langle \text{proof} \rangle$

end

6.8 Defer-One Loop Composition

theory *Defer-One-Loop-Composition*
imports *Basic-Modules/Component-Types/Defer-Equal-Condition*
 Loop-Composition
 Elect-Composition
begin

This is a family of loop compositions. It uses the same module in sequence until either no new decisions are made or only one alternative is remaining in the defer-set. The second family herein uses the above family and subsequently elects the remaining alternative.

fun $\text{iter} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ **where**
 $\text{iter } m =$
 $(\text{let } t = \text{defer-equal-condition } 1 \text{ in}$
 $(m \circlearrowleft_t))$

abbreviation $\text{defer-one-loop} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$
 $('a, 'v, 'a \text{ Result}) \text{ Electoral-Module}$ $(-\circlearrowleft_{\exists!d} \ 50)$ **where**
 $m \circlearrowleft_{\exists!d} \equiv \text{iter } m$

fun $\text{iter-elect} :: ('a, 'v, 'a \text{ Result}) \text{ Electoral-Module} \Rightarrow$

$('a, 'v, 'a \text{ Result}) \text{ Electoral-Module } \mathbf{where}$
 $iter-elect\ m = elector\ (m \circ_{\exists} !d)$
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 Properties

lemma $\text{plurality-rule-inv-mono-eq}$:

fixes

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

assumes

$\text{elect-}a$: $a \in \text{elect plurality-rule } V \ A \ p$ **and**
 $\text{lift-}a$: $\text{lifted } V \ A \ p \ q \ a$

shows $\text{elect plurality-rule } V \ A \ q = \text{elect plurality-rule } V \ A \ p$

$\vee \text{ elect plurality-rule } V A q = \{a\}$
 $\langle \text{proof} \rangle$

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* = *elector borda V A p*

fun *borda-rule_R* :: (*'a*, *'v::wellorder*, *'a Result*) *Electoral-Module* **where**
borda-rule_R *V A p* = *swap-R unanimity V A p*

7.2.2 Soundness

theorem *borda-rule-sound: SCF-result.electoral-module borda-rule*
 $\langle \text{proof} \rangle$

theorem *borda-rule_R-sound: SCF-result.electoral-module borda-rule_R*
 $\langle \text{proof} \rangle$

7.2.3 Anonymity

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

theorem *condorcet'-rule-sound*: *SCF-result.electoral-module condorcet'*
 $\langle \text{proof} \rangle$

theorem *pairwise-majority-rule'-sound*: *SCF-result.electoral-module pairwise-majority-rule'*
 $\langle \text{proof} \rangle$

7.3.3 Condorcet Consistency

theorem *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*
 $\langle \text{proof} \rangle$

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

```

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

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

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

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

theorem *kemeny-rule-anonymous: SCF-result.anonymity kemeny-rule*
<proof>

7.11.4 Neutrality

lemma *swap-dist-neutral: distance-neutrality well-formed-elections*
(votewise-distance swap l-one)
<proof>

theorem *kemeny-rule-neutral: SCF-properties.neutrality*
well-formed-elections kemeny-rule
<proof>

end

Bibliography

- [1] Karsten Diekhoff, Michael Kirsten, and Jonas Krämer. Formal property-oriented design of voting rules using composable modules. In Saša Pekeč and Kristen Brent Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019. [doi:10.1007/978-3-030-31489-7](https://doi.org/10.1007/978-3-030-31489-7).
- [2] Karsten Diekhoff, Michael Kirsten, and Jonas Krämer. Verified construction of fair voting rules. In Maurizio Gabbrielli, editor, *29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019), Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2020. [doi:10.1007/978-3-030-45260-5_6](https://doi.org/10.1007/978-3-030-45260-5_6).
- [3] Benjamin Hadjibeyli and Mark C. Wilson. Distance rationalization of social rules. *Computing Research Repository (CoRR)*, abs/1610.01902, 2016. [arXiv:1610.01902](https://arxiv.org/abs/1610.01902).