Verified Construction of Fair Voting Rules

Michael Kirsten

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

April 8, 2024

Abstract

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

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

Contents

Chapter 1

Social-Choice Types

1.1 Preference Relation

theory *Preference-Relation*
  imports *Main*
begin

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

1.1.1 Definition

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

type-synonym $'a\ Preference\text{-}Relation = 'a\ rel$

type-synonym $'a\ Vote = 'a\ set \times 'a\ Preference\text{-}Relation$

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

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

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

lemma *lin-imp-antisym*:
  fixes
    $A :: 'a\ set$ **and**
    $r :: 'a\ Preference\text{-}Relation$
  assumes *linear-order-on A r*
  shows *antisym r*

⟨*proof*⟩

lemma *lin-imp-trans*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation*
assumes *linear-order-on A r*
shows *trans r*
⟨*proof*⟩

1.1.2 Ranking

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

lemma *rank-gt-zero*:
fixes
r :: *'a Preference-Relation* **and**
a :: *'a*
assumes
refl: $a \preceq_r a$ **and**
fin: *finite r*
shows *rank r a* ≥ *1*
⟨*proof*⟩

1.1.3 Limited Preference

definition *limited* :: *'a set* ⇒ *'a Preference-Relation* ⇒ *bool* **where**
limited A r ≡ *r* ⊆ *A* × *A*

lemma *limited-dest*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
a :: *'a* **and**
b :: *'a*
assumes
$a \preceq_r b$ **and**
limited A r
shows *a* ∈ *A* ∧ *b* ∈ *A*
⟨*proof*⟩

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

definition *connex* :: *'a set* ⇒ *'a Preference-Relation* ⇒ *bool* **where**
connex A r ≡ *limited A r* ∧ (∀ *a* ∈ *A*. ∀ *b* ∈ *A*. $a \preceq_r b \lor b \preceq_r a$)

lemma *connex-imp-refl*:
fixes

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

lemma *lin-ord-imp-connex*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *linear-order-on* A r
shows *connex* A r
⟨*proof*⟩

lemma *connex-antsym-and-trans-imp-lin-ord*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
connex-r: *connex* A r **and**
antisym-r: *antisym* r **and**
trans-r: *trans* r
shows *linear-order-on* A r
⟨*proof*⟩

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

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

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

⟨*proof*⟩

lemma *limit-presv-trans*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes *trans* r
shows *trans* (*limit* A r)
⟨*proof*⟩

lemma *limit-presv-lin-ord*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
linear-order-on B r **and**
$A \subseteq B$
shows *linear-order-on* A (*limit* A r)
⟨*proof*⟩

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

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

lemma *limit-trans*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*

assumes $A \subseteq B$
shows *limit* A r = *limit* A (*limit* B r)
⟨*proof*⟩

lemma *lin-ord-not-empty*:
fixes r :: ′*a Preference-Relation*
assumes $r \neq \{\}$
shows ¬ *linear-order-on* {} r
⟨*proof*⟩

lemma *lin-ord-singleton*:
fixes a :: ′*a*
shows ∀ r. *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
⟨*proof*⟩

1.1.4 Auxiliary Lemmas

lemma *above-trans*:
fixes
r :: ′*a Preference-Relation* **and**
a :: ′*a* **and**
b :: ′*a*
assumes
trans r **and**
$(a, b) \in r$
shows *above* r $b \subseteq$ *above* r a
⟨*proof*⟩

lemma *above-refl*:
fixes
A :: ′*a set* **and**
r :: ′*a Preference-Relation* **and**
a :: ′*a*
assumes
refl-on A r **and**
$a \in A$
shows $a \in$ *above* r a
⟨*proof*⟩

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

above r' $a = \{a\}$
shows *above* r $a = \{a\}$
⟨*proof*⟩

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

lemma *pref-imp-in-above*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
shows $(a \preceq_r b) = (b \in$ *above* r $a)$
⟨*proof*⟩

lemma *limit-presv-above*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes
$b \in$ *above* r a **and**
$a \in A$ **and**
$b \in A$
shows $b \in$ *above* (*limit* A r) a
⟨*proof*⟩

lemma *limit-rel-presv-above*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
b :: $'a$
assumes $b \in$ *above* (*limit* B r) a
shows $b \in$ *above* r a
⟨*proof*⟩

lemma *above-one*:
fixes

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

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

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

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

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

shows $(above\ r\ a = \{a\}) = (rank\ r\ a = 1)$
⟨*proof*⟩

lemma *rank-unique*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$ **and**
$b :: 'a$
assumes
lin-ord: *linear-order-on* $A\ r$ **and**
fin-A: *finite* A **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
a-neq-b: $a \neq b$
shows $rank\ r\ a \neq rank\ r\ b$
⟨*proof*⟩

lemma *above-presv-limit*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$
shows $above\ (limit\ A\ r)\ a \subseteq A$
⟨*proof*⟩

1.1.5 Lifting Property

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

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

lemma *trivial-equiv-rel*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes *linear-order-on* $A\ r$
shows $\forall\ a \in A.$ *equiv-rel-except-a* $A\ r\ r\ a$
⟨*proof*⟩

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

A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* A r r' a
shows *equiv-rel-except-a* A r r' a
⟨*proof*⟩

lemma *lifted-imp-switched*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* A r r' a
shows $\forall\ a' \in A - \{a\}.\ \neg\ (a' \preceq_r a \wedge a \preceq_{r'} a')$
⟨*proof*⟩

lemma *lifted-mono*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
a' :: $'a$
assumes
lifted: *lifted* A r r' a **and**
a'-pref-a: $a' \preceq_r a$
shows $a' \preceq_{r'} a$
⟨*proof*⟩

lemma *lifted-above-subset*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* A r r' a
shows *above* r' $a \subseteq$ *above* r a
⟨*proof*⟩

lemma *lifted-above-mono*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
a' :: $'a$
assumes

lifted-a: *lifted* $A\ r\ r'\ a$ **and**
a'-*in-A-sub-a*: $a' \in A - \{a\}$
shows *above* $r\ a' \subseteq$ *above* $r'\ a' \cup \{a\}$
⟨*proof*⟩

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

lemma *negl-diff-imp-eq-limit*:
fixes
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
change: *equiv-rel-except-a* $A'\ r\ r'\ a$ **and**
subset: $A \subseteq A'$ **and**
not-in-A: $a \notin A$
shows *limit* $A\ r$ = *limit* $A\ r'$
⟨*proof*⟩

theorem *lifted-above-winner-alts*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
a' :: $'a$
assumes
lifted-a: *lifted* $A\ r\ r'\ a$ **and**
a'-*above-a*$'$: *above* $r\ a' = \{a'\}$ **and**
fin-A: *finite* A
shows *above* $r'\ a' = \{a'\} \vee$ *above* $r'\ a = \{a\}$
⟨*proof*⟩

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

r :: $'a$ *Preference-Relation* **and**
r' :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
lifted $A\ r\ r'\ a$ **and**
above $r\ a = \{a\}$ **and**
finite A
shows *above* $r'\ a = \{a\}$
⟨*proof*⟩

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

end

1.2 Norm

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

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

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

1.2.1 Definition

type-synonym *Norm* = *ereal list* ⇒ *ereal*

definition *norm* :: *Norm* ⇒ *bool* **where**
norm n ≡ ∀ (*x*::*ereal list*). *n x* ≥ *0* ∧ (∀ *i* < *length x*. (*x*!*i* = *0*) ⟶ *n x* = *0*)

1.2.2 Auxiliary Lemmas

lemma *sum-over-image-of-bijection*:
fixes
A :: *'a set* **and**
A' :: *'b set* **and**
f :: *'a* ⇒ *'b* **and**
g :: *'a* ⇒ *ereal*
assumes *bij-betw f A A'*
shows (∑ *a* ∈ *A*. *g a*) = (∑ *a'* ∈ *A'*. *g* (*the-inv-into A f a'*))
⟨*proof*⟩

1.2.3 Common Norms

fun *l-one* :: *Norm* **where**
l-one x = (∑ *i* < *length x*. |*x*!*i*|)

1.2.4 Properties

definition *symmetry* :: *Norm* ⇒ *bool* **where**
symmetry n ≡ ∀ *x y*. *x* <~~> *y* ⟶ *n x* = *n y*

1.2.5 Theorems

theorem *l-one-is-sym*: *symmetry l-one*
⟨*proof*⟩

end

1.3 Electoral Result

theory *Result*
imports *Main*
begin

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

1.3.1 Auxiliary Functions

type-synonym $'r\ Result = {'r}\ set * {'r}\ set * {'r}\ set$

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

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

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

1.3.2 Definition

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

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

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

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

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

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

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

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

end

1.4 Preference Profile

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

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

1.4.1 Definition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

— Here, we count the occurrences of a ballot in an election, i.e., how many voters specifically chose that exact ballot.
fun *vote-count* :: $'a$ *Preference-Relation* $\Rightarrow$ $('a,\ 'v)$ *Election* $\Rightarrow$ *nat* **where**
vote-count $p\ E =$ *card* $\{v \in$ (*voters-*$\mathcal{E}$ E). (*profile-*$\mathcal{E}$ E) $v = p\}$

1.4.2 Vote Count

lemma *sum-comp*:
fixes
$f :: 'x \Rightarrow 'z$::*comm-monoid-add* **and**
$g :: 'y \Rightarrow 'x$ **and**
$X :: 'x$ *set* **and**
$Y :: 'y$ *set*
assumes *bij-betw* $g\ Y\ X$
shows *sum* $f\ X =$ *sum* $(f \circ g)\ Y$
⟨*proof*⟩

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

shows *sum* (λ *p. vote-count p E*) *UNIV* = *card* (*voters-$\mathcal{E}$ E*)
⟨*proof*⟩

1.4.3 Voter Permutations

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

fun *rename* :: ($'v \Rightarrow 'v$) $\Rightarrow$ ($'a$, $'v$) *Election* $\Rightarrow$ ($'a$, $'v$) *Election* **where**
rename π (A, V, p) = (A, π ‘ V, $p \circ$ (*the-inv* π))

lemma *rename-sound*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
prof: *profile V A p* **and**
renamed: (A, V', q) = *rename* π (A, V, p) **and**
bij: *bij* π
shows *profile* V' *A q*
⟨*proof*⟩

lemma *rename-finite*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
finite-profile V A p **and**
(A, V', q) = *rename* π (A, V, p) **and**
bij π
shows *finite-profile* V' *A q*
⟨*proof*⟩

lemma *rename-inv*:
fixes
π :: $'v \Rightarrow 'v$ **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile*
assumes *bij* π
shows *rename* π (*rename* (*the-inv* π) (A, V, p)) = (A, V, p)
⟨*proof*⟩

lemma *rename-inj*:
fixes π :: $'v \Rightarrow 'v$
assumes *bij* π

shows *inj* (*rename* π)
⟨*proof*⟩

lemma *rename-surj*:
fixes π :: $'v \Rightarrow 'v$
assumes *bij* π
shows
on-valid-elections: *rename* π ‘ *valid-elections* = *valid-elections* **and**
on-finite-elections: *rename* π ‘ *finite-elections* = *finite-elections*
⟨*proof*⟩

1.4.4 List Representation for Ordered Voters

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

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

lemma *map2-helper*:
fixes
f :: $'x \Rightarrow 'y \Rightarrow 'z$ **and**
g :: $'x \Rightarrow 'x$ **and**
h :: $'y \Rightarrow 'y$ **and**
l :: $'x$ *list* **and**
l' :: $'y$ *list*
shows *map2* f (*map* g l) (*map* h l') = *map2* (λ x y. f (g x) (h y)) l l'
⟨*proof*⟩

lemma *to-list-simp*:
fixes
i :: *nat* **and**
V :: $'v$::*linorder set* **and**
p :: ($'a$, $'v$) *Profile*
assumes i < *card* V
shows (*to-list* V p)!i = p ((*sorted-list-of-set* V)!i)
⟨*proof*⟩

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

lemma *set-card-upper-bound*:

fixes
  i :: *nat* **and**
  V :: *nat set*
assumes
  fin-V: *finite* V **and**
  bound-v: $\forall\ v \in V.\ v < i$
shows *card* $V \leq i$
⟨*proof*⟩

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

lemma *to-list-permutes-under-bij*:
fixes
  π :: $'v$::*linorder* $\Rightarrow$ $'v$ **and**
  V :: $'v$ *set* **and**
  p :: $('a, 'v)$ *Profile*
assumes *bij* π
shows
  let $\varphi = (\lambda\ i.\ card\ \{v \in \pi\ `\ V.\ v < \pi\ ((sorted\text{-}list\text{-}of\text{-}set\ V)!i)\})$
    in $(to\text{-}list\ V\ p) = permute\text{-}list\ \varphi\ (to\text{-}list\ (\pi\ `\ V)\ (\lambda\ x.\ p\ (the\text{-}inv\ \pi\ x)))$
⟨*proof*⟩

1.4.5 Preference Counts and Comparisons

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

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

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

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

assumes *finite* V
shows *prefer-count* $V\ p\ a\ b \leq$ *card* V
$\langle proof \rangle$

lemma *set-compr*:
fixes
$A :: 'a\ set$ **and**
$f :: 'a \Rightarrow 'a\ set$
shows $\{f\ x \mid x.\ x \in A\} = f\ `\ A$
$\langle proof \rangle$

lemma *pref-count-set-compr*:
fixes
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)$ *Profile* **and**
$a :: 'a$
shows $\{$*prefer-count* $V\ p\ a\ a' \mid a'.\ a' \in A - \{a\}\} =$
(*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**
$a :: 'a$ **and**
$b :: 'a$
assumes
prof: *profile* $V\ A\ p$ **and**
fin: *finite* V **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
neq: $a \neq b$
shows *prefer-count* $V\ p\ a\ b =$ *card* V $-$ (*prefer-count* $V\ p\ b\ a$)
$\langle proof \rangle$

lemma *pref-count-sym*:
fixes
$p :: ('a, 'v)$ *Profile* **and**
$V :: 'v\ set$ **and**
$a :: 'a$ **and**
$b :: 'a$ **and**
$c :: 'a$
assumes
pref-count-ineq: *prefer-count* $V\ p\ a\ c \geq$ *prefer-count* $V\ p\ c\ b$ **and**
prof: *profile* $V\ A\ p$ **and**
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**

c-in-A: $c \in A$ **and**
a-neq-c: $a \neq c$ **and**
c-neq-b: $c \neq b$
shows *prefer-count V p b c* $\geq$ *prefer-count V p c a*
⟨*proof*⟩

lemma *empty-prof-imp-zero-pref-count*:
fixes
p :: ($'a$, $'v$) *Profile* **and**
V :: $'v$ *set* **and**
a :: $'a$ **and**
b :: $'a$
assumes $V = \{\}$
shows *prefer-count V p a b* = *0*
⟨*proof*⟩

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

lemma *wins-inf-voters*:
fixes
p :: ($'a$, $'v$) *Profile* **and**
a :: $'a$ **and**
b :: $'a$ **and**
V :: $'v$ *set*
assumes *infinite V*
shows $\neg$ *wins V b p a*
⟨*proof*⟩

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

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

lemma *wins-irreflex*:
fixes
p :: ($'a$, $'v$) *Profile* **and**
a :: $'a$ **and**
V :: $'v$ *set*
shows $\neg$ *wins V a p a*
⟨*proof*⟩

1.4.6 Condorcet Winner

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

lemma *cond-winner-unique-eq*:
fixes
$V :: 'v$ *set* **and**
$A :: 'a$ *set* **and**
$p :: ('a, 'v)$ *Profile* **and**
$a :: 'a$ **and**
$b :: 'a$
assumes
condorcet-winner $V\ A\ p\ a$ **and**
condorcet-winner $V\ A\ p\ b$
shows $b = a$
⟨*proof*⟩

lemma *cond-winner-unique*:
fixes
$A :: 'a$ *set* **and**
$p :: ('a, 'v)$ *Profile* **and**
$a :: 'a$
assumes *condorcet-winner* $V\ A\ p\ a$
shows $\{a' \in A.\ \mathit{condorcet\text{-}winner}\ V\ A\ p\ a'\} = \{a\}$
⟨*proof*⟩

lemma *cond-winner-unique-2*:
fixes
$V :: 'v$ *set* **and**
$A :: 'a$ *set* **and**
$p :: ('a, 'v)$ *Profile* **and**
$a :: 'a$ **and**
$b :: 'a$
assumes
condorcet-winner $V\ A\ p\ a$ **and**
$b \neq a$
shows $\neg$ *condorcet-winner* $V\ A\ p\ b$
⟨*proof*⟩

1.4.7 Limited Profile

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

fun *limit-profile* :: $'a$ *set* $\Rightarrow$ $('a, 'v)$ *Profile* $\Rightarrow$ $('a, 'v)$ *Profile* **where**
limit-profile $A\ p = (\lambda\ v.\ \mathit{limit}\ A\ (p\ v))$

lemma *limit-prof-trans*:
  fixes
    $A :: 'a\ set$ **and**
    $B :: 'a\ set$ **and**
    $C :: 'a\ set$ **and**
    $p :: ('a, 'v)\ Profile$
  assumes
    $B \subseteq A$ **and**
    $C \subseteq B$
  shows $limit\text{-}profile\ C\ p = limit\text{-}profile\ C\ (limit\text{-}profile\ B\ p)$
  $\langle proof \rangle$

lemma *limit-profile-sound*:
  fixes
    $A :: 'a\ set$ **and**
    $B :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a, 'v)\ Profile$
  assumes
    $profile\ V\ B\ p$ **and**
    $A \subseteq B$
  shows $profile\ V\ A\ (limit\text{-}profile\ A\ p)$
$\langle proof \rangle$

1.4.8 Lifting Property

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

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

lemma *lifted-imp-equiv-prof-except-a*:
  fixes
    $A :: 'a\ set$ **and**
    $V :: 'v\ set$ **and**
    $p :: ('a, 'v)\ Profile$ **and**
    $p' :: ('a, 'v)\ Profile$ **and**
    $a :: 'a$

assumes *lifted* V A p p' a
shows *equiv-prof-except-a* V A p p' a
⟨*proof*⟩

lemma *negl-diff-imp-eq-limit-prof*:
fixes
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
p' :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
change: *equiv-prof-except-a* V A' p q a **and**
subset: $A \subseteq A'$ **and**
not-in-A: $a \notin A$
shows $\forall$ $v \in V$. (*limit-profile* A p) v = (*limit-profile* A q) v
— With the current definitions of *equiv-prof-except-a* and *limit-prof*, we can only conclude that the limited profiles coincide on the given voter set, since *limit-prof* may change the profiles everywhere, while *equiv-prof-except-a* only makes statements about the voter set.
⟨*proof*⟩

lemma *limit-prof-eq-or-lifted*:
fixes
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
p' :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
lifted-a: *lifted* V A' p p' a **and**
subset: $A \subseteq A'$
shows ($\forall$ $v \in V$. *limit-profile* A p v = *limit-profile* A p' v)
$\vee$ *lifted* V A (*limit-profile* A p) (*limit-profile* A p') a
⟨*proof*⟩

end

1.5 Social Choice Result

theory *Social-Choice-Result*
imports *Result*
begin

1.5.1 Social Choice Result

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

fun *well-formed-$\mathcal{SCF}$* :: *'a set* ⇒ *'a Result* ⇒ *bool* **where**
  well-formed-$\mathcal{SCF}$ A res = (*disjoint3 res* ∧ *set-equals-partition A res*)

fun *limit-set-$\mathcal{SCF}$* :: *'a set* ⇒ *'a set* ⇒ *'a set* **where**
  limit-set-$\mathcal{SCF}$ A r = $A \cap r$

1.5.2 Auxiliary Lemmas

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

lemma *result-count*:
  fixes
    A :: *'a set* **and**
    e :: *'a set* **and**
    r :: *'a set* **and**
    d :: *'a set*
  assumes
    wf-result: *well-formed-$\mathcal{SCF}$ 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-$\mathcal{SCF}$ A r*
  shows *defer-r r* ⊆ *A*
⟨*proof*⟩

lemma *elect-subset*:
  fixes
    A :: *'a set* **and**
    r :: *'a Result*
  assumes *well-formed-$\mathcal{SCF}$ A r*
  shows *elect-r r* ⊆ *A*
⟨*proof*⟩

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

end

1.6 Social Welfare Result

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

1.6.1 Social Welfare Result

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

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

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

end

1.7 Specific Electoral Result Types

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

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

⟨*ML*⟩

Results from social choice functions ($\mathcal{SCF}s$), for the purpose of composability and modularity given as three sets of (potentially tied) alternatives. See `Social_Choice_Result.thy` for details.

global-interpretation *$\mathcal{SCF}$-result*:
  result well-formed-$\mathcal{SCF}$ limit-set-$\mathcal{SCF}$
⟨*proof*⟩

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

global-interpretation *committee-result*:
  $result\ \lambda\ A\ r.\ set\text{-}equals\text{-}partition\ (Pow\ A)\ r \wedge disjoint3\ r$
  $\lambda\ A\ rs.\ \{r \cap A \mid r.\ r \in rs\}$
⟨*proof*⟩

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 *$\mathcal{SWF}$-result*:
  result well-formed-$\mathcal{SWF}$ limit-set-$\mathcal{SWF}$
⟨*proof*⟩

⟨*ML*⟩

end

1.8 Function Symmetry Properties

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

1.8.1 Functions

type-synonym $('x, 'y)\ binary\text{-}fun = 'x \Rightarrow 'y \Rightarrow 'y$

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

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

1.8.2 Relations for Symmetry Constructions

fun $\mathit{restricted\text{-}rel} :: {'x}\ \mathit{rel} \Rightarrow {'x}\ \mathit{set} \Rightarrow {'x}\ \mathit{set} \Rightarrow {'x}\ \mathit{rel}$ **where**
$\mathit{restricted\text{-}rel}\ r\ s\ s' = r \cap s \times s'$

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

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

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

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

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

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

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

1.8.3 Invariance and Equivariance

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

datatype $({'x}, {'y})\ \mathit{symmetry} =$
$\mathit{Invariance}\ {'x}\ \mathit{rel}\ \mid$
$\mathit{Equivariance}\ {'x}\ \mathit{set}\ (({'x} \Rightarrow {'x}) \times ({'y} \Rightarrow {'y}))\ \mathit{set}$

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

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

1.8.4 Auxiliary Lemmas

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

assumes *inj f*
shows *inj* $(\lambda\ s.\ \{f\ `\ x \mid x.\ x \in s\})$
⟨*proof*⟩

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

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

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

lemma *the-inv-comp*:
fixes
$f :: {}'y \Rightarrow {}'z$ **and**
$g :: {}'x \Rightarrow {}'y$ **and**
$s :: {}'x$ *set* **and**
$t :: {}'y$ *set* **and**
$u :: {}'z$ *set* **and**
$x :: {}'z$
assumes
bij-betw f t u **and**
bij-betw g s t **and**
$x \in u$
shows *the-inv-into s* $(f \circ g)\ x = (($*the-inv-into s g*$) \circ ($*the-inv-into t f*$))\ x$
⟨*proof*⟩

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*⟩

1.8.5 Rewrite Rules

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

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

lemma *rewrite-invar-ind-by-act*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$s :: 'z$ *set* **and**
$t :: 'x$ *set* **and**
$\varphi :: ('z, 'x)$ *binary-fun*
shows *is-symmetry* f (*Invariance* (*action-induced-rel* $s\ t\ \varphi$)) =
$(\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y \in t \longrightarrow f\ y = f\ (\varphi\ x\ y))$
⟨*proof*⟩

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

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

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

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

fixes f :: $'a \Rightarrow 'a$
assumes $f \in$ *carrier* (*BijGroup UNIV*)
shows *bij f*
⟨*proof*⟩

lemma *rewrite-sym-group*:
fixes
f :: $'a \Rightarrow 'a$ **and**
g :: $'a \Rightarrow 'a$ **and**
s :: $'a$ *set*
assumes
$f \in$ *carrier* (*BijGroup s*) **and**
$g \in$ *carrier* (*BijGroup s*)
shows
rewrite-mult: $f \otimes_{BijGroup\ s} g$ = *extensional-continuation* $(f \circ g)$ s **and**
rewrite-mult-univ: $s = UNIV \longrightarrow f \otimes_{BijGroup\ s} g = f \circ g$
⟨*proof*⟩

lemma *simp-extensional-univ*:
fixes f :: $'a \Rightarrow 'b$
shows *extensional-continuation f UNIV* = f
⟨*proof*⟩

lemma *extensional-continuation-subset*:
fixes
f :: $'a \Rightarrow 'b$ **and**
s :: $'a$ *set* **and**
t :: $'a$ *set* **and**
x :: $'a$
assumes
$t \subseteq s$ **and**
$x \in t$
shows *extensional-continuation f s x* = *extensional-continuation f t x*
⟨*proof*⟩

lemma *rel-ind-by-coinciding-action-on-subset-eq-restr*:
fixes
φ :: $('a, 'b)$ *binary-fun* **and**
ψ :: $('a, 'b)$ *binary-fun* **and**
s :: $'a$ *set* **and**
t :: $'b$ *set* **and**
u :: $'b$ *set*
assumes
$u \subseteq t$ **and**
$\forall\ x \in s.\ \forall\ y \in u.\ \psi\ x\ y = \varphi\ x\ y$
shows *action-induced-rel* $s\ u\ \psi$ = *Restr* (*action-induced-rel* $s\ t\ \varphi$) u
⟨*proof*⟩

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

fixes
s :: *'x set* **and**
t :: *'y set* **and**
φ :: (*'x*, *'y*) *binary-fun* **and**
ψ :: (*'x*, *'y*) *binary-fun*
assumes $\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y = \psi\ x\ y$
shows *action-induced-rel s t* φ = *action-induced-rel s t* ψ
⟨*proof*⟩

1.8.6 Group Actions

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

theorem *group-act-induces-set-group-act*:
fixes
m :: *'x monoid* **and**
s :: *'y set* **and**
φ :: (*'x*, *'y*) *binary-fun*
defines φ*-img* ≡ (λ *x. extensional-continuation* (*image* (φ *x*)) (*Pow s*))
assumes *group-action m s* φ
shows *group-action m* (*Pow s*) φ*-img*
⟨*proof*⟩

1.8.7 Invariance and Equivariance

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

theorem *equivar-generators-imp-equivar-group*:
fixes
f :: *'x* ⇒ *'y* **and**
m :: *'z monoid* **and**
s :: *'z set* **and**
t :: *'x set* **and**
φ :: (*'z*, *'x*) *binary-fun* **and**
ψ :: (*'z*, *'y*) *binary-fun*
assumes
equivar: *is-symmetry f* (*action-induced-equivariance s t* φ ψ) **and**
action-φ: *group-action m t* φ **and**
action-ψ: *group-action m* (*f* ‘ *t*) ψ **and**
gen: *carrier m* = *generate m s*
shows *is-symmetry f* (*action-induced-equivariance* (*carrier m*) *t* φ ψ)
⟨*proof*⟩

lemma *invar-parameterized-fun*:
  fixes
    f :: $'x \Rightarrow ('x \Rightarrow 'y)$ **and**
    r :: $'x$ *rel*
  assumes
    param-invar: $\forall\ x.$ *is-symmetry* $(f\ x)$ $(Invariance\ r)$ **and**
    invar: *is-symmetry* f $(Invariance\ r)$
  shows *is-symmetry* $(\lambda\ x.\ f\ x\ x)$ $(Invariance\ r)$
  ⟨*proof*⟩

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

lemma *equivar-ind-by-act-coincide*:
  fixes
    s :: $'x$ *set* **and**
    t :: $'y$ *set* **and**
    f :: $'y \Rightarrow 'z$ **and**
    φ :: $('x, 'y)$ *binary-fun* **and**
    φ' :: $('x, 'y)$ *binary-fun* **and**
    ψ :: $('x, 'z)$ *binary-fun*
  assumes $\forall\ x \in s.\ \forall\ y \in t.\ \varphi\ x\ y = \varphi'\ x\ y$
  shows *is-symmetry* f $(action\text{-}induced\text{-}equivariance\ s\ t\ \varphi\ \psi)$ =
          is-symmetry f $(action\text{-}induced\text{-}equivariance\ s\ t\ \varphi'\ \psi)$
  ⟨*proof*⟩

lemma *equivar-under-subset*:
  fixes
    f :: $'x \Rightarrow 'y$ **and**
    s :: $'x$ *set* **and**
    t :: $'x$ *set* **and**
    τ :: $(('x \Rightarrow 'x) \times ('y \Rightarrow 'y))$ *set*
  assumes
    is-symmetry f $(Equivariance\ s\ \tau)$ **and**
    $t \subseteq s$
  shows *is-symmetry* f $(Equivariance\ t\ \tau)$
  ⟨*proof*⟩

lemma *equivar-under-subset'*:
  fixes
    f :: $'x \Rightarrow 'y$ **and**

$s :: 'x\ set$ **and**
$\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ set$ **and**
$\upsilon :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ set$
assumes
is-symmetry f (*Equivariance* s τ) **and**
$\upsilon \subseteq \tau$
shows *is-symmetry* f (*Equivariance* s υ)
⟨*proof*⟩

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

Invariance and Equivariance Function Composition

lemma *invar-comp*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$g :: 'y \Rightarrow 'z$ **and**
$r :: 'x\ rel$
assumes *is-symmetry* f (*Invariance* r)
shows *is-symmetry* $(g \circ f)$ (*Invariance* r)
⟨*proof*⟩

lemma *equivar-comp*:
fixes
$f :: 'x \Rightarrow 'y$ **and**
$g :: 'y \Rightarrow 'z$ **and**
$s :: 'x\ set$ **and**
$t :: 'y\ set$ **and**
$\tau :: (('x \Rightarrow 'x) \times ('y \Rightarrow 'y))\ set$ **and**
$\upsilon :: (('y \Rightarrow 'y) \times ('z \Rightarrow 'z))\ set$

defines
transitive-acts $\equiv$
$\{(\varphi, \psi).\ \exists\ \chi :: 'y \Rightarrow 'y.\ (\varphi, \chi) \in \tau \wedge (\chi, \psi) \in \upsilon \wedge \chi\ `\ f\ `\ s \subseteq t\}$
assumes
$f\ `\ s \subseteq t$ **and**
is-symmetry f (*Equivariance* s τ) **and**
is-symmetry g (*Equivariance* t υ)
shows *is-symmetry* $(g \circ f)$ (*Equivariance* s *transitive-acts*)
⟨*proof*⟩

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

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

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

shows *is-symmetry* $\bigcup$ (*action-induced-equivariance s UNIV*
(*set-action* (*set-action* φ)) (*set-action* φ))
⟨*proof*⟩

end

1.9 Symmetry Properties of Voting Rules

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

1.9.1 Definitions

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

fun *result-action* :: ('*x*, '*r*) *binary-fun* ⇒ ('*x*, '*r Result*) *binary-fun* **where**
result-action ψ *x* = (λ *r*. (ψ *x* ' *elect-r r*, ψ *x* ' *reject-r r*, ψ *x* ' *defer-r r*))

Anonymity

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

fun φ*-anon* :: ('*a*, '*v*) *Election set* ⇒ ('*v* ⇒ '*v*) ⇒ (('*a*, '*v*) *Election*
⇒ ('*a*, '*v*) *Election*) **where**
φ*-anon* $\mathcal{E}$ π = *extensional-continuation* (*rename* π) $\mathcal{E}$

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

Neutrality

fun *rel-rename* :: ('*a* ⇒ '*a*, '*a Preference-Relation*) *binary-fun* **where**
rel-rename π *r* = {(π *a*, π *b*) | *a b*. (*a*, *b*) ∈ *r*}

fun *alternatives-rename* :: ('*a* ⇒ '*a*, ('*a*, '*v*) *Election*) *binary-fun* **where**
alternatives-rename π $\mathcal{E}$ =
(π ' (*alternatives-*$\mathcal{E}$ $\mathcal{E}$), *voters-*$\mathcal{E}$ $\mathcal{E}$, (*rel-rename* π) ∘ (*profile-*$\mathcal{E}$ $\mathcal{E}$))

definition *neutrality*$_{\mathcal{G}}$ :: ('*a* ⇒ '*a*) *monoid* **where**

$neutrality_{\mathcal{G}} = BijGroup\ (UNIV::'a\ set)$

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

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

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

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

Homogeneity

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

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

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

Reversal Symmetry

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

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

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

fun $\varphi\text{-}rev :: ('a, 'v)\ Election\ set$
$\Rightarrow ('a\ rel \Rightarrow 'a\ rel, ('a, 'v)\ Election)\ binary\text{-}fun$ **where**

φ-rev $\mathcal{E}$ φ = extensional-continuation (rel-app φ) $\mathcal{E}$

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

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

1.9.2 Auxiliary Lemmas

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

lemma n-app-rewrite:
fixes
f :: 'x $\Rightarrow$ 'x **and**
n :: nat **and**
x :: 'x
shows (f $\circ$ n-app n f) x = (n-app n f $\circ$ f) x
$\langle proof \rangle$

lemma n-app-leaves-set:
fixes
A :: 'x set **and**
B :: 'x set **and**
f :: 'x $\Rightarrow$ 'x **and**
x :: 'x
assumes
fin-A: finite A **and**
fin-B: finite B **and**
x-el: x $\in$ A $-$ B **and**
bij: bij-betw f A B
obtains n :: nat **where**
n > 0 **and**
n-app n f x $\in$ B $-$ A **and**
$\forall$ m > 0. m < n $\longrightarrow$ n-app m f x $\in$ A $\cap$ B
$\langle proof \rangle$

lemma n-app-rev:
fixes
A :: 'x set **and**
B :: 'x set **and**
f :: 'x $\Rightarrow$ 'x **and**
n :: nat **and**
m :: nat **and**
x :: 'x **and**
y :: 'x
assumes

x-in-A: $x \in A$ **and**
y-in-A: $y \in A$ **and**
n-geq-m: $n \geq m$ **and**
n-app-eq-m-n: *n-app* $n\ f\ x =$ *n-app* $m\ f\ y$ **and**
n-app-x-in-A: $\forall\ n' < n.$ *n-app* $n'\ f\ x \in A$ **and**
n-app-y-in-A: $\forall\ m' < m.$ *n-app* $m'\ f\ y \in A$ **and**
fin-A: *finite* A **and**
fin-B: *finite* B **and**
bij-f-A-B: *bij-betw* $f\ A\ B$
shows *n-app* $(n - m)\ f\ x = y$
⟨*proof*⟩

lemma *n-app-inv*:
fixes
$A :: 'x\ set$ **and**
$B :: 'x\ set$ **and**
$f :: 'x \Rightarrow 'x$ **and**
$n :: nat$ **and**
$x :: 'x$
assumes
$x \in B$ **and**
$\forall\ m \geq 0.\ m < n \longrightarrow$ *n-app* m (*the-inv-into* $A\ f$) $x \in B$ **and**
bij-betw $f\ A\ B$
shows *n-app* $n\ f$ (*n-app* n (*the-inv-into* $A\ f$) x) $= x$
⟨*proof*⟩

lemma *bij-betw-finite-ind-global-bij*:
fixes
$A :: 'x\ set$ **and**
$B :: 'x\ set$ **and**
$f :: 'x \Rightarrow 'x$
assumes
fin-A: *finite* A **and**
fin-B: *finite* B **and**
bij: *bij-betw* $f\ A\ B$
obtains $g :: 'x \Rightarrow 'x$ **where**
bij g **and**
$\forall\ a \in A.\ g\ a = f\ a$ **and**
$\forall\ b \in B - A.\ g\ b \in A - B \wedge (\exists\ n > 0.$ *n-app* $n\ f\ (g\ b) = b)$ **and**
$\forall\ x \in UNIV - A - B.\ g\ x = x$
⟨*proof*⟩

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

$\langle proof \rangle$

1.9.3 Anonymity Lemmas

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

lemma *vote-count-anon-rel*:
fixes
$\mathcal{E}$ :: ($'a$, $'v$) *Election set* **and**
E :: ($'a$, $'v$) *Election* **and**
E' :: ($'a$, $'v$) *Election*
assumes
fin-voters-E: *finite* (*voters-*$\mathcal{E}$ E) **and**
fin-voters-E': *finite* (*voters-*$\mathcal{E}$ E') **and**
default-non-v: $\forall$ v. $v \notin$ *voters-*$\mathcal{E}$ E $\longrightarrow$ *profile-*$\mathcal{E}$ E v = {} **and**
default-non-v': $\forall$ v. $v \notin$ *voters-*$\mathcal{E}$ E' $\longrightarrow$ *profile-*$\mathcal{E}$ E' v = {} **and**
eq: *alternatives-*$\mathcal{E}$ E = *alternatives-*$\mathcal{E}$ E' $\wedge$ (E, E') $\in \mathcal{E} \times \mathcal{E}$
$\wedge$ ($\forall$ p. *vote-count* p E = *vote-count* p E')
shows (E, E') $\in$ *anonymity*$_{\mathcal{R}}$ $\mathcal{E}$
$\langle proof \rangle$

lemma *rename-comp*:
fixes
π :: $'v \Rightarrow$ $'v$ **and**
π' :: $'v \Rightarrow$ $'v$
assumes
bij π **and**
bij π'
shows *rename* π $\circ$ *rename* π' = *rename* ($\pi \circ \pi'$)
$\langle proof \rangle$

interpretation *anonymous-group-action*:
group-action anonymity$_{\mathcal{G}}$ *valid-elections* φ*-anon valid-elections*
$\langle proof \rangle$

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

1.9.4 Neutrality Lemmas

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

lemma *rel-rename-comp*:
  fixes
    π :: $'a \Rightarrow 'a$ **and**
    π' :: $'a \Rightarrow 'a$
  shows *rel-rename* $(\pi \circ \pi')$ = *rel-rename* $\pi \circ$ *rel-rename* π'
⟨*proof*⟩

lemma *rel-rename-sound*:
  fixes
    π :: $'a \Rightarrow 'a$ **and**
    r :: $'a$ *rel* **and**
    A :: $'a$ *set*
  assumes *inj* π
  shows
    refl-on $A\ r \longrightarrow$ *refl-on* $(\pi$ ‘ $A)$ (*rel-rename* $\pi\ r$) **and**
    antisym $r \longrightarrow$ *antisym* (*rel-rename* $\pi\ r$) **and**
    total-on $A\ r \longrightarrow$ *total-on* $(\pi$ ‘ $A)$ (*rel-rename* $\pi\ r$) **and**
    Relation.trans $r \longrightarrow$ *Relation.trans* (*rel-rename* $\pi\ r$)
⟨*proof*⟩

lemma *rename-subset*:
  fixes
    r :: $'a$ *rel* **and**
    s :: $'a$ *rel* **and**
    a :: $'a$ **and**
    b :: $'a$ **and**
    π :: $'a \Rightarrow 'a$
  assumes
    bij-π: *bij* π**and**
    rel-rename $\pi\ r$ = *rel-rename* $\pi\ s$ **and**
    $(a,\ b) \in r$
  shows $(a,\ b) \in s$
⟨*proof*⟩

lemma *rel-rename-bij*:
  fixes π :: $'a \Rightarrow 'a$
  assumes *bij-π*: *bij* π

shows *bij* (*rel-rename* π)
⟨*proof*⟩

lemma *alternatives-rename-comp*:
fixes
π :: *'a* $\Rightarrow$ *'a* **and**
π' :: *'a* $\Rightarrow$ *'a*
shows
alternatives-rename π $\circ$ *alternatives-rename* π' = *alternatives-rename* ($\pi \circ \pi'$)
⟨*proof*⟩

lemma *valid-elects-closed*:
fixes
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile* **and**
A' :: *'a set* **and**
V' :: *'v set* **and**
p' :: (*'a*, *'v*) *Profile* **and**
π :: *'a* $\Rightarrow$ *'a*
assumes
bij-π: *bij* π **and**
valid-elects: (*A*, *V*, *p*) $\in$ *valid-elections* **and**
renamed: (*A'*, *V'*, *p'*) = *alternatives-rename* π (*A*, *V*, *p*)
shows (*A'*, *V'*, *p'*) $\in$ *valid-elections*
⟨*proof*⟩

lemma *alternatives-rename-bij*:
fixes π :: (*'a* $\Rightarrow$ *'a*)
assumes *bij-*π: *bij* π
shows *bij-betw* (*alternatives-rename* π) *valid-elections valid-elections*
⟨*proof*⟩

interpretation φ*-neutral-action*:
group-action neutrality$_{\mathcal{G}}$ *valid-elections* φ*-neutr valid-elections*
⟨*proof*⟩

interpretation ψ*-neutral*$_{\text{c}}$*-action*: *group-action neutrality*$_{\mathcal{G}}$ *UNIV* ψ*-neutr*$_{\text{c}}$
⟨*proof*⟩

interpretation ψ*-neutral*$_{\text{w}}$*-action*: *group-action neutrality*$_{\mathcal{G}}$ *UNIV* ψ*-neutr*$_{\text{w}}$
⟨*proof*⟩

lemma *wf-result-neutrality-*$\mathcal{SCF}$:
is-symmetry (λ $\mathcal{E}$. *limit-set-*$\mathcal{SCF}$ (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
(*action-induced-equivariance* (*carrier neutrality*$_{\mathcal{G}}$) *valid-elections*
(φ*-neutr valid-elections*) (*set-action* ψ*-neutr*$_{\text{c}}$))
⟨*proof*⟩

lemma *wf-result-neutrality-$\mathcal{SWF}$*:
is-symmetry (λ $\mathcal{E}$. *limit-set-$\mathcal{SWF}$* (*alternatives-$\mathcal{E}$* $\mathcal{E}$) *UNIV*)
(*action-induced-equivariance* (*carrier neutrality$_\mathcal{G}$*) *valid-elections*
(φ*-neutr valid-elections*) (*set-action* ψ*-neutr*$_\mathrm{w}$))
⟨*proof*⟩

1.9.5 Homogeneity Lemmas

lemma *refl-homogeneity$_\mathcal{R}$*:
fixes $\mathcal{E}$ :: ($'a$, $'v$) *Election set*
assumes $\mathcal{E} \subseteq$ *finite-elections-$\mathcal{V}$*
shows *refl-on* $\mathcal{E}$ (*homogeneity$_\mathcal{R}$* $\mathcal{E}$)
⟨*proof*⟩

lemma (**in** *result*) *well-formed-res-homogeneity*:
is-symmetry (λ $\mathcal{E}$. *limit-set* (*alternatives-$\mathcal{E}$* $\mathcal{E}$) *UNIV*)
(*Invariance* (*homogeneity$_\mathcal{R}$* *UNIV*))
⟨*proof*⟩

lemma *refl-homogeneity$_\mathcal{R}$'*:
fixes $\mathcal{E}$ :: ($'a$, $'v$::*linorder*) *Election set*
assumes $\mathcal{E} \subseteq$ *finite-elections-$\mathcal{V}$*
shows *refl-on* $\mathcal{E}$ (*homogeneity$_\mathcal{R}$'* $\mathcal{E}$)
⟨*proof*⟩

lemma (**in** *result*) *well-formed-res-homogeneity'*:
is-symmetry (λ $\mathcal{E}$. *limit-set* (*alternatives-$\mathcal{E}$* $\mathcal{E}$) *UNIV*)
(*Invariance* (*homogeneity$_\mathcal{R}$'* *UNIV*))
⟨*proof*⟩

1.9.6 Reversal Symmetry Lemmas

lemma *rev-rev-id*: *rev-rel* $\circ$ *rev-rel* = *id*
⟨*proof*⟩

lemma *rev-rel-limit*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *rel*
shows *rev-rel* (*limit* A r) = *limit* A (*rev-rel* r)
⟨*proof*⟩

lemma *rev-rel-lin-ord*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *rel*
assumes *linear-order-on* A r
shows *linear-order-on* A (*rev-rel* r)
⟨*proof*⟩

interpretation *reversal*$_\mathcal{G}$*-group*: *group reversal*$_\mathcal{G}$
⟨*proof*⟩

interpretation *φ-reverse-action*:
  group-action reversal$_\mathcal{G}$ *valid-elections φ-rev valid-elections*
⟨*proof*⟩

interpretation *ψ-reverse-action*: *group-action reversal*$_\mathcal{G}$ *UNIV ψ-rev*
⟨*proof*⟩

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

end

1.10 Result-Dependent Voting Rule Properties

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

1.10.1 Properties Dependent on the Result Type

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

New result-type-dependent definitions for properties can be added here.

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

sublocale *result-properties* ⊆ *result*
  ⟨*proof*⟩

1.10.2 Interpretations

global-interpretation *$\mathcal{SCF}$-properties*:
result-properties well-formed-$\mathcal{SCF}$ limit-set-$\mathcal{SCF}$ ψ*-neutr*$_{\mathrm{c}}$
⟨*proof*⟩

global-interpretation *$\mathcal{SWF}$-properties*:
result-properties well-formed-$\mathcal{SWF}$ limit-set-$\mathcal{SWF}$ ψ*-neutr*$_{\mathrm{w}}$
⟨*proof*⟩

end

Chapter 2

Refined Types

2.1 Preference List

theory *Preference-List*
  imports *../Preference-Relation*
    HOL−Combinatorics.Multiset-Permutations
    List−Index.List-Index
begin

Preference lists derive from preference relations, ordered from most to least preferred alternative.

2.1.1 Well-Formedness

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

abbreviation *well-formed-l* :: $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
  well-formed-l $l \equiv$ *distinct* l

2.1.2 Auxiliary Lemmas About Lists

lemma *is-arg-min-equal*:
  fixes
    $f :: 'a \Rightarrow 'b::ord$ **and**
    $g :: 'a \Rightarrow 'b$ **and**
    $S :: 'a\ set$ **and**
    $x :: 'a$
  assumes $\forall\ x \in S.\ f\ x = g\ x$
  shows *is-arg-min* $f\ (\lambda\ s.\ s \in S)\ x =$ *is-arg-min* $g\ (\lambda\ s.\ s \in S)\ x$
$\langle proof \rangle$

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

assumes
fin-A: *finite A* **and**
fin-B: *finite S*
shows *finite* $\{a\#l \mid a\ l.\ a \in A \wedge l \in S\}$
⟨*proof*⟩

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

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

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

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

2.1.3 Ranking

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

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

lemma *rank-l-equiv*: *rank-l* = *rank-l-idx*
⟨*proof*⟩

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**
$a \lesssim_l b = (a \in set\ l \land b \in set\ l \land index\ l\ a \geq index\ l\ b)$

lemma *rank-gt-zero*:
fixes
l :: *'a Preference-List* **and**
a :: *'a*
assumes $a \lesssim_l a$
shows *rank-l l a* $\geq$ *1*
$\langle proof \rangle$

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

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

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

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

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

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

lemma *pl-α-eq-imp-list-eq*:
fixes
l :: *'x list* **and**
l' :: *'x list*
assumes
fin-set-l: *finite* (*set* l) **and**
set-eq: *set* l = *set* l' **and**
dist-l: *distinct* l **and**
dist-l': *distinct* l' **and**
pl-α-eq: *pl-α* l = *pl-α* l'
shows $l = l'$
⟨*proof*⟩

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

2.1.5 Limited Preference

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

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

lemma *limited-dest*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List* **and**
a :: *'a* **and**
b :: *'a*
assumes
$a \lesssim_l b$ **and**
limited A l
shows $a \in A \wedge b \in A$
⟨*proof*⟩

lemma *limit-equiv*:
fixes
A :: *'a set* **and**

l :: $'a$ *list*
assumes *well-formed-l l*
shows *pl-α (limit-l A l) = limit A (pl-α l)*
⟨*proof*⟩

2.1.6 Auxiliary Definitions

definition *total-on-l* :: $'a$ *set* ⇒ $'a$ *Preference-List* ⇒ *bool* **where**
total-on-l A l ≡ $\forall\ a \in A.\ a \in set\ l$

definition *refl-on-l* :: $'a$ *set* ⇒ $'a$ *Preference-List* ⇒ *bool* **where**
refl-on-l A l ≡ $(\forall\ a.\ a \in set\ l \longrightarrow a \in A) \wedge (\forall\ a \in A.\ a \lesssim_l a)$

definition *trans* :: $'a$ *Preference-List* ⇒ *bool* **where**
trans l ≡ $\forall\ (a, b, c) \in set\ l \times set\ l \times set\ l.\ a \lesssim_l b \wedge b \lesssim_l c \longrightarrow a \lesssim_l c$

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

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

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

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

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

abbreviation *ballot-on* :: $'a$ *set* ⇒ $'a$ *Preference-List* ⇒ *bool* **where**
ballot-on A l ≡ *well-formed-l l* ∧ *linear-order-on-l A l*

2.1.7 Auxiliary Lemmas

lemma *list-trans*[*simp*]:
fixes l :: $'a$ *Preference-List*
shows *trans l*
⟨*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 :: ′*a* **and**
b :: ′*a*
assumes
trans l **and**
$a \lesssim_l b$
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 :: ′*a* **and**
b :: ′*a*
shows $a \lesssim_l b$ =
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
$\forall$ (j::*nat*) $\leq n$. $l!j \neq a$ **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
$a \in$ *set l* **and**
$n <$ (*rank-l l a*) $- 1$
shows $l!n \neq a$
⟨*proof*⟩

lemma *pos-in-list-yields-pos*:
fixes
l :: $'a$ *Preference-List* **and**
a :: $'a$
assumes $a \in$ *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. $y \lesssim_l z$) (*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*⟩

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 :: $'x$ *set*
  assumes *finite* X
  obtains *ord* :: $'x$ *rel* **where**
    linear-order-on X *ord*
⟨*proof*⟩

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

instance *Ordered-Preference* :: (*finite*) *finite*
⟨*proof*⟩

lemma *range-ord2pref*: *range ord2pref* = $\{p.$ *linear-order* $p\}$
  ⟨*proof*⟩

lemma *card-ord-pref*: *card* (*UNIV*::$'a$::*finite Ordered-Preference set*) =
    fact (*card* (*UNIV*::$'a$ *set*))
⟨*proof*⟩

end

2.4 Alternative Election Type

theory *Quotient-Type-Election*
  imports *Profile*
begin

lemma *election-equality-equiv*:
  election-equality E E **and**
  election-equality E $E' \Longrightarrow$ *election-equality* E' E **and**
  election-equality E $E' \Longrightarrow$ *election-equality* E' F
    $\Longrightarrow$ *election-equality* E F
⟨*proof*⟩

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

$\langle proof \rangle$

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

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

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

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

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

end

Chapter 3

Quotient Rules

3.1 Quotients of Equivalence Relations

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

3.1.1 Definitions

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

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

fun $\pi_{\mathcal{Q}}$ :: $('x \Rightarrow 'y) \Rightarrow ('x\ set \Rightarrow 'y)$ **where**
  $\pi_{\mathcal{Q}}\ f\ s = singleton\text{-}set\ (f\ `\ s)$

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

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

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

3.1.2 Well-Definedness

lemma *singleton-set-undef-if-card-neq-one*:
  fixes $s :: 'x\ set$

assumes *card* $s \neq 1$
shows *singleton-set* s = *undefined*
⟨*proof*⟩

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

If the given function is invariant under an equivalence relation, the induced function on sets is well-defined for all equivalence classes of that relation.

theorem *pass-to-quotient*:
fixes
f :: $'x \Rightarrow 'y$ **and**
r :: $'x$ *rel* **and**
s :: $'x$ *set*
assumes
f *respects* r **and**
equiv s r
shows $\forall\ t \in s\ //\ r.\ \forall\ x \in t.\ \pi_{\mathcal{Q}}\ f\ t = f\ x$
⟨*proof*⟩

A function on sets induces a function on the element type that is invariant under a given equivalence relation.

theorem *pass-to-quotient-inv*:
fixes
f :: $'x$ *set* $\Rightarrow 'x$ **and**
r :: $'x$ *rel* **and**
s :: $'x$ *set*
assumes *equiv* s r
defines *induced-fun* $\equiv$ (*inv*-$\pi_{\mathcal{Q}}$ (*relation-class* r) f)
shows
induced-fun *respects* r **and**
$\forall\ A \in s\ //\ r.\ \pi_{\mathcal{Q}}$ *induced-fun* $A = f\ A$
⟨*proof*⟩

3.1.3 Equivalence Relations

lemma *equiv-rel-restr*:
fixes
s :: $'x$ *set* **and**
t :: $'x$ *set* **and**
r :: $'x$ *rel*
assumes
equiv s r **and**
$t \subseteq s$
shows *equiv* t (*Restr* r t)
⟨*proof*⟩

lemma *rel-ind-by-group-act-equiv*:
  fixes
    m :: $'x$ *monoid* **and**
    s :: $'y$ *set* **and**
    φ :: $('x, 'y)$ *binary-fun*
  assumes *group-action* $m\ s\ \varphi$
  shows *equiv* s (*action-induced-rel* (*carrier* m) $s\ \varphi$)
⟨*proof*⟩

end

3.2 Quotients of Equivalence Relations on Election Sets

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

3.2.1 Auxiliary Lemmas

lemma *obtain-partition*:
  fixes
    X :: $'x$ *set* **and**
    N :: $'y \Rightarrow nat$ **and**
    Y :: $'y$ *set*
  assumes
    finite X **and**
    finite Y **and**
    sum $N\ Y = card\ X$
  shows $\exists\ \mathcal{X}.\ X = \bigcup\ \{\mathcal{X}\ i \mid i.\ i \in Y\} \land (\forall\ i \in Y.\ card\ (\mathcal{X}\ i) = N\ i) \land$
    $(\forall\ i\ j.\ i \neq j \longrightarrow i \in Y \land j \in Y \longrightarrow \mathcal{X}\ i \cap \mathcal{X}\ j = \{\})$
  ⟨*proof*⟩

3.2.2 Anonymity Quotient: Grid

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

— Here, we count the occurrences of a ballot per election in a set of elections for which the occurrences of the ballot per election coincide for all elections in the set.
fun $vote\text{-}count_{\mathcal{Q}}$ :: $'a$ *Preference-Relation* $\Rightarrow$ $('a, 'v)$ *Election set* $\Rightarrow$ *nat* **where**
  $vote\text{-}count_{\mathcal{Q}}\ p = \pi_{\mathcal{Q}}$ (*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*$_{\mathcal{Q}}$ (*ord2pref p*) *X*)

lemma *anon-rel-equiv*:
equiv (*elections-$\mathcal{A}$ UNIV*) (*anonymity*$_{\mathcal{R}}$ (*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*$_{\mathcal{Q}}$*-isomorphism*:
assumes *infinite* (*UNIV*::('*v set*))
shows *bij-betw* (*anonymity-class*::('*a*::*finite*, '*v*) *Election set*
$\Rightarrow$ *nat*^('*a Ordered-Preference*)) (*anonymity*$_{\mathcal{Q}}$ (*UNIV*::'*a set*))
(*UNIV*::(*nat*^('*a Ordered-Preference*)) *set*)
⟨*proof*⟩

3.2.3 Homogeneity Quotient: Simplex

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

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

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

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

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

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

fun *rat-vector* :: *real*$^\frown$'*b* ⇒ *rat*$^\frown$'*b* **where**
  rat-vector v = (χ *p*. *the-inv of-rat* (*v*\$*p*))

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

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

Auxiliary Lemmas

lemma *convex-combination-in-convex-hull*:
  fixes
    X :: (*real*$^\frown$'*b*) *set* **and**
    x :: *real*$^\frown$'*b*
  assumes ∃ *f*::(*real*$^\frown$'*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*$^\frown$'*b*). (∀ *i*. *v*\$*i* ≥ *0*) ∧ *sum* ((\$) *v*) *UNIV* = *1*}
⟨*proof*⟩

lemma *fract-distr-helper*:
  fixes
    a :: *int* **and**
    b :: *int* **and**
    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*$_\mathcal{R}$ *X*)
⟨*proof*⟩

lemma *fract-distr*:
  fixes
    A :: '*x set* **and**

$f :: 'x \Rightarrow int$ **and**
$b :: int$
assumes
finite A **and**
$b \neq 0$
shows $sum\ (\lambda\ a.\ Fract\ (f\ a)\ b)\ A = Fract\ (sum\ f\ A)\ b$
⟨*proof*⟩

Simplex Bijection

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

theorem *anonymity-homogeneity*$_{\mathcal{Q}}$*-isomorphism*:
assumes *infinite* $(UNIV::('v\ set))$
shows
bij-betw $(anonymity\text{-}homogeneity\text{-}class::('a::finite,\ 'v)\ Election\ set \Rightarrow rat\widehat{\ }('a\ Ordered\text{-}Preference))\ (anonymity\text{-}homogeneity_{\mathcal{Q}}\ (UNIV::'a\ set))\ (vote\text{-}simplex :: (rat\widehat{\ }('a\ Ordered\text{-}Preference))\ set)$
⟨*proof*⟩

end

Chapter 4

Component Types

4.1 Distance

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

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

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

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

4.1.1 Definition

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

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

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

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

4.1.2 Conditions

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

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

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

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

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

4.1.3 Standard Distance Property

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

4.1.4 Auxiliary Lemmas

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

lemma *arg-min-subset*:
fixes
$B :: 'b\ set$ **and**
$f :: 'b \Rightarrow 'a :: ord$
shows $arg\text{-}min\text{-}set\ f\ B \subseteq B$
$\langle proof \rangle$

lemma *sum-monotone*:
fixes
$A :: 'a\ set$ **and**
$f :: 'a \Rightarrow int$ **and**
$g :: 'a \Rightarrow int$
assumes $\forall\ a \in A.\ f\ a \leq g\ a$
shows $(\sum\ a \in A.\ f\ a) \leq (\sum\ a \in A.\ g\ a)$
$\langle proof \rangle$

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

g :: $'a \Rightarrow int$
shows $(\sum\ a \in A.\ f\ a) + (\sum\ a \in A.\ g\ a) = (\sum\ a \in A.\ f\ a + g\ a)$
⟨*proof*⟩

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

lemma *uneq-ereal*:
fixes
x :: int **and**
y :: int
assumes $x \leq y$
shows $ereal\ (real\text{-}of\text{-}int\ x) \leq ereal\ (real\text{-}of\text{-}int\ y)$
⟨*proof*⟩

4.1.5 Swap Distance

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

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

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

lemma *set-eq-filter*:
fixes
X :: $'a\ set$ **and**
P :: $'a \Rightarrow bool$
shows $\{x \in X.\ P\ x\} = Set.filter\ P\ X$
⟨*proof*⟩

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

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

then card (*pairwise-disagreements A r r′*)
else ∞)

lemma *swap-case-infinity*:
fixes
x :: *′a Vote* **and**
y :: *′a Vote*
assumes *alts-𝒱 x* $\neq$ *alts-𝒱 y*
shows *swap x y* $= \infty$
⟨*proof*⟩

lemma *swap-case-fin*:
fixes
x :: *′a Vote* **and**
y :: *′a Vote*
assumes *alts-𝒱 x* = *alts-𝒱 y*
shows *swap x y* = *card* (*pairwise-disagreements* (*alts-𝒱 x*) (*pref-𝒱 x*) (*pref-𝒱 y*))
⟨*proof*⟩

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 ∞)

lemma *spearman-case-inf*:
fixes
x :: *′a Vote* **and**
y :: *′a Vote*
assumes *alts-𝒱 x* $\neq$ *alts-𝒱 y*
shows *spearman x y* $= \infty$
⟨*proof*⟩

lemma *spearman-case-fin*:
fixes
x :: *′a Vote* **and**
y :: *′a Vote*
assumes *alts-𝒱 x* = *alts-𝒱 y*
shows *spearman x y* =
($\sum$ *a* $\in$ *alts-𝒱 x*. *abs* (*int* (*rank* (*pref-𝒱 x*) *a*) − *int* (*rank* (*pref-𝒱 y*) *a*)))
⟨*proof*⟩

4.1.7 Properties

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

Definitions

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

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

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

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

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

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

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

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

Auxiliary Lemmas

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

lemma *invar-dist-image*:
fixes
d :: $'y$ *Distance* **and**
G :: $'x$ *monoid* **and**
Y :: $'y$ *set* **and**
Y' :: $'y$ *set* **and**
φ :: $('x, 'y)$ *binary-fun* **and**
y :: $'y$ **and**
g :: $'x$
assumes
invar-d: *invariance*$_{\mathcal{D}}$ d (*carrier* G) $Y\ \varphi$ **and**
Y'-in-Y: $Y' \subseteq Y$ **and**
action-φ: *group-action* $G\ Y\ \varphi$ **and**
g-carrier: $g \in$ *carrier* G **and**
y-in-Y: $y \in Y$
shows $d\ (\varphi\ g\ y)$ ‘ $(\varphi\ g)$ ‘ $Y' = d\ y$ ‘ Y'
⟨*proof*⟩

lemma *swap-neutral*: *invariance*$_{\mathcal{D}}$ *swap* (*carrier neutrality*$_{\mathcal{G}}$)
UNIV $(\lambda\ \pi\ (A, q).\ (\pi$ ‘ A, *rel-rename* $\pi\ q))$
⟨*proof*⟩

end

4.2 Votewise Distance

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

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

4.2.1 Definition

fun *votewise-distance* :: $'a$ *Vote Distance* $\Rightarrow$ *Norm*
$\Rightarrow ('a,'v$::*linorder*) *Election Distance* **where**
votewise-distance $d\ n\ (A, V, p)\ (A', V', p')$ =

(*if* (*finite* V) $\wedge$ $V = V'$ $\wedge$ ($V \neq \{\}$ $\vee$ $A = A'$)
then n (*map2* (λ q q'. d (A, q) (A', q')) (*to-list* V p) (*to-list* V' p'))
else ∞)

4.2.2 Inference Rules

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

lemma *permute-invariant-under-map*:
fixes
l :: *'a list* **and**
ls :: *'a list*
assumes l <~~> *ls*
shows *map f l* <~~> *map f ls*
$\langle$*proof*$\rangle$

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

lemma *permute-invariant-under-coinciding-funs*:
fixes
l :: *'v list* **and**
π-*1* :: *nat* $\Rightarrow$ *nat* **and**
π-*2* :: *nat* $\Rightarrow$ *nat*

assumes $\forall\ i < length\ l.\ \pi\text{-}1\ i = \pi\text{-}2\ i$
shows *permute-list* π*-1 l* = *permute-list* π*-2 l*
⟨*proof*⟩

lemma *symmetric-norm-imp-distance-anonymous*:
fixes
d :: *'a Vote Distance* **and**
n :: *Norm*
assumes *symmetry n*
shows *distance-anonymity* (*votewise-distance d n*)
⟨*proof*⟩

lemma *neutral-dist-imp-neutral-votewise-dist*:
fixes
d :: *'a Vote Distance* **and**
n :: *Norm*
defines *vote-action* $\equiv$ (λ π (*A*, *q*). (π ‘ *A*, *rel-rename* π *q*))
assumes *invar*: $invariance_{\mathcal{D}}$ *d* (*carrier* $neutrality_{\mathcal{G}}$) *UNIV vote-action*
shows *distance-neutrality valid-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 $nonempty\text{-}set_{\mathcal{C}}$ :: (*'a*, *'v*) *Consensus* **where**
$nonempty\text{-}set_{\mathcal{C}}$ (*A*, *V*, *p*) = (*A* $\neq$ {})

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

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

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

Equal top ranked alternatives.

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

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

Equal votes.

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

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

Unanimity condition.

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

Strong unanimity condition.

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

4.3.3 Properties

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

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

4.3.4 Auxiliary Lemmas

lemma *cons-anon-conj*:
fixes
$c1 :: ({}'a,\ {}'v)\ Consensus$ **and**
$c2 :: ({}'a,\ {}'v)\ Consensus$
assumes
anon1: $consensus\text{-}anonymity\ c1$ **and**
anon2: $consensus\text{-}anonymity\ c2$
shows $consensus\text{-}anonymity\ (\lambda\ e.\ c1\ e \wedge c2\ e)$
$\langle proof \rangle$

theorem *cons-conjunction-invariant*:
fixes
$\mathfrak{C}$ :: ($'a$, $'v$) *Consensus set* **and**
rel :: ($'a$, $'v$) *Election rel*
defines $C \equiv (\lambda\ E.\ (\forall\ C' \in \mathfrak{C}.\ C'\ E))$
assumes $\bigwedge C'.\ C' \in \mathfrak{C} \Longrightarrow$ *is-symmetry* C' (*Invariance rel*)
shows *is-symmetry* C (*Invariance rel*)
⟨*proof*⟩

lemma *cons-anon-invariant*:
fixes
c :: ($'a$, $'v$) *Consensus* **and**
A :: $'a$ *set* **and**
A' :: $'a$ *set* **and**
V :: $'v$ *set* **and**
V' :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
q :: ($'a$, $'v$) *Profile* **and**
π :: $'v \Rightarrow 'v$
assumes
anon: *consensus-anonymity* c **and**
bij: *bij* π **and**
prof-p: *profile* $V\ A\ p$ **and**
renamed: *rename* π $(A, V, p) = (A', V', q)$ **and**
cond-c: c (A, V, p)
shows c (A', V', q)
⟨*proof*⟩

lemma *ex-anon-cons-imp-cons-anonymous*:
fixes
b :: ($'a$, $'v$) *Consensus* **and**
b':: $'b \Rightarrow$ ($'a$, $'v$) *Consensus*
assumes
general-cond-b: $b = (\lambda\ E.\ \exists\ x.\ b'\ x\ E)$ **and**
all-cond-anon: $\forall\ x.$ *consensus-anonymity* $(b'\ x)$
shows *consensus-anonymity* b
⟨*proof*⟩

4.3.5 Theorems

Anonymity

lemma *nonempty-set-cons-anonymous*: *consensus-anonymity* *nonempty-set*$_{\mathcal{C}}$
⟨*proof*⟩

lemma *nonempty-profile-cons-anonymous*: *consensus-anonymity* *nonempty-profile*$_{\mathcal{C}}$
⟨*proof*⟩

lemma *equal-top-cons'-anonymous*:
fixes a :: $'a$

shows *consensus-anonymity* (*equal-top*$_\mathcal{C}$*′ a*)
⟨*proof*⟩

lemma *eq-top-cons-anon*: *consensus-anonymity equal-top*$_\mathcal{C}$
⟨*proof*⟩

lemma *eq-vote-cons′-anonymous*:
fixes *r* :: *′a Preference-Relation*
shows *consensus-anonymity* (*equal-vote*$_\mathcal{C}$*′ r*)
⟨*proof*⟩

lemma *eq-vote-cons-anonymous*: *consensus-anonymity equal-vote*$_\mathcal{C}$
⟨*proof*⟩

Neutrality

lemma *nonempty-set*$_\mathcal{C}$*-neutral*: *consensus-neutrality valid-elections nonempty-set*$_\mathcal{C}$
⟨*proof*⟩

lemma *nonempty-profile*$_\mathcal{C}$*-neutral*: *consensus-neutrality valid-elections nonempty-profile*$_\mathcal{C}$
⟨*proof*⟩

lemma *equal-vote*$_\mathcal{C}$*-neutral*: *consensus-neutrality valid-elections equal-vote*$_\mathcal{C}$
⟨*proof*⟩

lemma *strong-unanimity*$_\mathcal{C}$*-neutral*:
consensus-neutrality valid-elections strong-unanimity$_\mathcal{C}$
⟨*proof*⟩

end

4.4 Electoral Module

theory *Electoral-Module*
imports *Social-Choice-Types/Property-Interpretations*
begin

Electoral modules are the principal component type of the composable modules voting framework, as they are a generalization of voting rules in the sense of social choice functions. These are only the types used for electoral modules. Further restrictions are encompassed by the electoral-module predicate.

An electoral module does not need to make final decisions for all alternatives, but can instead defer the decision for some or all of them to other modules. Hence, electoral modules partition the received (possibly empty) set of alternatives into elected, rejected and deferred alternatives. In particular, any of those sets, e.g., the set of winning (elected) alternatives, may also be left empty, as long as they collectively still hold all the received alternatives. Just like a voting rule, an electoral module also receives a profile which holds the voters preferences, which, unlike a voting rule, consider only the (sub-)set of alternatives that the module receives.

4.4.1 Definition

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

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

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

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

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

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

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

4.4.2 Auxiliary Definitions

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

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

fun *voters-determine-election* :: $('a, 'v, ('r\ Result))\ Electoral\text{-}Module \Rightarrow bool$ **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-modI*:
fixes $m :: ('a,\ 'v,\ ('r\ Result))\ Electoral\text{-}Module$
assumes $\bigwedge A\ V\ p.\ profile\ V\ A\ p \Longrightarrow well\text{-}formed\ A\ (m\ V\ A\ p)$
shows *electoral-module m*
⟨*proof*⟩

4.4.3 Properties

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

Anonymity

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

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

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

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

Homogeneity

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

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

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

lemma (**in** *result*) *hom-imp-anon*:
fixes *X* :: (′*a*, ′*v*) *Election set*
assumes
homogeneity X m **and**
∀ *E* ∈ *X*. *finite* (*voters-*$\mathcal{E}$ *E*)
shows *anonymity′ X m*
⟨*proof*⟩

Neutrality

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

fun (**in** *result-properties*) *neutrality* :: (′*a*, ′*v*) *Election set*
⇒ (′*a*, ′*v*, ′*b Result*) *Electoral-Module* ⇒ *bool* **where**
neutrality X m =
is-symmetry ($fun_{\mathcal{E}}$ *m*) (*action-induced-equivariance* (*carrier* $neutrality_{\mathcal{G}}$) *X*
(φ*-neutr X*) (*result-action* ψ*-neutr*))

4.4.4 Reversal Symmetry of Social Welfare Rules

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

definition *reversal-symmetry* :: (′*a*, ′*v*) *Election set*
⇒ (′*a*, ′*v*, ′*a rel Result*) *Electoral-Module* ⇒ *bool* **where**
reversal-symmetry X m =
is-symmetry ($fun_{\mathcal{E}}$ *m*) (*action-induced-equivariance* (*carrier* $reversal_{\mathcal{G}}$) *X*
(φ*-rev X*) (*result-action* ψ*-rev*))

4.4.5 Social Choice Modules

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

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

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

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

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

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

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

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

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

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

4.4.6 Equivalence Definitions

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

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

$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
prof-leq-result m V A p q a $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
profile V A p $\wedge$ *profile V A q* $\wedge$ $a \in A$ $\wedge$
$(a \in$ *reject m V A p* $\longrightarrow a \in$ *reject m V A q*$)$ $\wedge$
$(a \in$ *defer m V A p* $\longrightarrow a \notin$ *elect m V A q*$)$

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

definition *mod-contains-result* :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
$\Rightarrow ('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
mod-contains-result m n V A p a $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
$\mathcal{SCF}$*-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*
$\Rightarrow ('a, 'v, 'a$ *Result*$)$ *Electoral-Module* $\Rightarrow 'v$ *set* $\Rightarrow 'a$ *set*
$\Rightarrow ('a, 'v)$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
mod-contains-result-sym m n V A p a $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$
$\mathcal{SCF}$*-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 :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
V :: $'v$ *set* **and**
A :: $'a$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
e :: $'a$ *set* **and**
r :: $'a$ *set* **and**
d :: $'a$ *set*
assumes

elect m V A p = e **and**
reject m V A p = r **and**
defer m V A p = d
shows m V A p = (e, r, d)
⟨*proof*⟩

lemma *par-comp-result-sound*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
SCF-result.electoral-module m **and**
profile V A p
shows *well-formed-SCF* A (m V A p)
⟨*proof*⟩

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

lemma *result-disj*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
p :: ('*a*, '*v*) *Profile* **and**
V :: '*v set*
assumes
SCF-result.electoral-module m **and**
profile V A p
shows
(*elect* m V A p) ∩ (*reject* m V A p) = {} ∧
(*elect* m V A p) ∩ (*defer* m V A p) = {} ∧
(*reject* m V A p) ∩ (*defer* m V A p) = {}
⟨*proof*⟩

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

assumes
  $\mathcal{SCF}$*-result.electoral-module m* **and**
  profile V A p
shows *elect m V A p* $\subseteq$ *A*
⟨*proof*⟩

lemma *reject-in-alts*:
fixes
  m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile*
assumes
  $\mathcal{SCF}$*-result.electoral-module m* **and**
  profile V A p
shows *reject m V A p* $\subseteq$ *A*
⟨*proof*⟩

lemma *defer-in-alts*:
fixes
  m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile*
assumes
  $\mathcal{SCF}$*-result.electoral-module m* **and**
  profile V A p
shows *defer m V A p* $\subseteq$ *A*
⟨*proof*⟩

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

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

lemma *upper-card-bounds-for-result*:
fixes
  m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
  A :: *'a set* **and**
  V :: *'v set* **and**
  p :: (*'a*, *'v*) *Profile*

assumes
  $\mathcal{SCF}$*-result.electoral-module m* **and**
  profile V A p **and**
  finite A
shows
  upper-card-bound-for-elect: *card* (*elect m V A p*) $\leq$ *card A* **and**
  upper-card-bound-for-reject: *card* (*reject m V A p*) $\leq$ *card A* **and**
  upper-card-bound-for-defer: *card* (*defer m V A p*) $\leq$ *card A*
⟨*proof*⟩

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

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

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

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

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

lemma *mod-contains-result-comm*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes *mod-contains-result* m n V A p a
shows *mod-contains-result* n m V A p a
⟨*proof*⟩

lemma *not-rej-imp-elec-or-defer*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
profile V A p **and**
$a \in A$ **and**
$a \notin$ *reject* m V A p
shows $a \in$ *elect* m V A p $\lor$ $a \in$ *defer* m V A p
⟨*proof*⟩

lemma *single-elim-imp-red-def-set*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
eliminates 1 m **and**
card $A > 1$ **and**
profile V A p

shows *defer* $m\ V\ A\ p \subset A$
⟨*proof*⟩

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

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

4.4.8 Non-Blocking

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

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

4.4.9 Electing

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

definition *electing* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ *bool* **where**
electing m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p*. (*A* ≠ {} ∧ *finite A* ∧ *profile V A p*) ⟶ *elect m V A p* ≠ {})

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* ⇒ *bool* **where**
non-electing m ≡
$\mathcal{SCF}$*-result.electoral-module m*
∧ (∀ *A V p*. *profile V A p* ⟶ *elect m V A p* = {})

lemma *single-rej-decr-def-card*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes
rejecting: *rejects 1 m* **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile V A p*
shows *card* (*defer m V A p*) = *card A* − *1*
⟨*proof*⟩

lemma *single-elim-decr-def-card-2*:
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* ⇒ *bool* **where**
defer-deciding m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧ *non-electing m* ∧ *defers 1 m*

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

definition *decrementing* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* ⇒ *bool* **where**
decrementing m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p*. *profile V A p* ∧ *card A* > *1* ⟶ *card* (*reject m V A p*) ≥ *1*)

definition *defer-condorcet-consistency* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
⇒ *bool* **where**
defer-condorcet-consistency m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p a*. *condorcet-winner V A p a* ⟶
(*m V A p* = ({}, *A* − (*defer m V A p*), {*d* ∈ *A*. *condorcet-winner V A p d*})))

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

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

definition *defer-monotonicity* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* ⇒ *bool* **where**
defer-monotonicity m ≡
$\mathcal{SCF}$-*result.electoral-module m* ∧
(∀ *A V p q a*.
(*a* ∈ *defer m V A p* ∧ *lifted V A p q a*) ⟶ *a* ∈ *defer m V A q*)

An electoral module is defer-lift-invariant iff lifting a deferred alternative does not affect the outcome.

definition *defer-lift-invariance* :: $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ *bool* **where**
  defer-lift-invariance $m \equiv$
    $\mathcal{SCF}$*-result.electoral-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**
  $dli\text{-}rel\ m = \{((A, V, p), (A, V, q))\ |A\ V\ p\ q.\ (\exists\ a \in defer\ m\ V\ A\ p.\ lifted\ V\ A\ p\ q\ a)\}$

lemma *rewrite-dli-as-invariance*:
  fixes
    m :: $('a, 'v, 'a\ Result)$ *Electoral-Module*
  shows
    defer-lift-invariance $m =$
      $(\mathcal{SCF}$*-result.electoral-module* m
          $\wedge$ $(is\text{-}symmetry\ (fun_{\mathcal{E}}\ m)\ (Invariance\ (dli\text{-}rel\ m))))$
$\langle proof \rangle$

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

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

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 $\equiv$
$\mathcal{SCF}$*-result.electoral-module m* $\wedge$ *non-electing m* $\wedge$
$(\forall\ A\ V\ p\ q\ a.\ (a \in \textit{defer}\ m\ V\ A\ p \wedge \textit{lifted}\ V\ A\ p\ q\ a) \longrightarrow$
$(\textit{defer}\ m\ V\ A\ q = \textit{defer}\ m\ V\ A\ p \vee \textit{defer}\ m\ V\ A\ q = \{a\}))$

4.4.11 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
a :: '*a*
assumes
ccomp: *condorcet-compatibility m* **and**
dd: *defer-deciding m* **and**
winner: *condorcet-winner V A p a*
shows *defer m V A p* = {*a*}
⟨*proof*⟩

theorem *ccomp-and-dd-imp-dcc*[*simp*]:
fixes *m* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
ccomp: *condorcet-compatibility m* **and**
dd: *defer-deciding m*
shows *defer-condorcet-consistency m*
⟨*proof*⟩

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

theorem *disj-compat-comm*[*simp*]:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes *disjoint-compatibility m n*
shows *disjoint-compatibility n m*
⟨*proof*⟩

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

theorem *dl-inv-imp-def-mono*[*simp*]:
fixes *m* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes *defer-lift-invariance m*
shows *defer-monotonicity m*
⟨*proof*⟩

4.4.12 Social Choice Properties

Condorcet Consistency

definition *condorcet-consistency* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ *bool* **where**
condorcet-consistency m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p a. condorcet-winner V A p a* ⟶
(*m V A p* = ({*e* ∈ *A. condorcet-winner V A p e*}, *A* − (*elect m V A p*), {})))

lemma *condorcet-consistency'*:
fixes *m* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
shows *condorcet-consistency m* =
($\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p a. condorcet-winner V A p a* ⟶
(*m V A p* = ({*a*}, *A* − (*elect m V A p*), {}))))
⟨*proof*⟩

lemma *condorcet-consistency''*:
fixes *m* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
shows *condorcet-consistency m* =
($\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p a.*
condorcet-winner V A p a ⟶ *m V A p* = ({*a*}, *A* − {*a*}, {})))
⟨*proof*⟩

(Weak) Monotonicity

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

definition *monotonicity* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ *bool* **where**
monotonicity m ≡
$\mathcal{SCF}$*-result.electoral-module m* ∧
(∀ *A V p q a. a* ∈ *elect m V A p* ∧ *lifted V A p q a* ⟶ *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* ($fun_{\mathcal{E}}$ m) (*Invariance* r)) = ($fun_{\mathcal{E}}$ m *respects* r)
⟨*proof*⟩

lemma *invariance-is-congruence'*:
fixes
f :: $'x \Rightarrow 'y$ **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 ($fun_{\mathcal{E}}$ m) (*Invariance* r)
shows $\forall\ A \in X\ //\ r.\ \forall\ E \in A.\ \pi_{\mathcal{Q}}\ (fun_{\mathcal{E}}\ m)\ A = fun_{\mathcal{E}}\ m\ E$
⟨*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**

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

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

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

4.6.3 Theorems

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

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

type-synonym ('*a*, '*v*) *Electoral-Set* = '*v set* ⇒ '*a set* ⇒ ('*a*, '*v*) *Profile* ⇒ '*a set*

fun *elimination-set* :: ('*a*, '*v*) *Evaluation-Function* ⇒ *Threshold-Value* ⇒
    Threshold-Relation ⇒ ('*a*, '*v*) *Electoral-Set* **where**
 elimination-set e t r V A p = {*a* ∈ *A* . *r* (*e V a A p*) *t*}

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

4.7.2 Social Choice Definitions

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

4.7.3 Common Social Choice Eliminators

fun *less-eliminator* :: ('*a*, '*v*) *Evaluation-Function*
    ⇒ *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* ∈ *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 (≤) *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* ∈ *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 $\mathcal{SCF}$*-result.electoral-module* (*elimination-module e t r*)
⟨*proof*⟩

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

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

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

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

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

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

4.7.5 Only participating voters impact the result

lemma *voters-determine-elim-mod*[*simp*]:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *Threshold-Value* **and**
r :: *Threshold-Relation*
assumes *voters-determine-evaluation e*
shows *voters-determine-election* (*elimination-module e t r*)
⟨*proof*⟩

lemma *voters-determine-less-elim*[*simp*]:
fixes
e :: (*'a*, *'v*) *Evaluation-Function* **and**
t :: *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 :: *Threshold-Value*
shows *non-blocking* (*less-eliminator e t*)
⟨*proof*⟩

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

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

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

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

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

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

4.7.7 Non-Electing

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

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

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

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

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

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

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

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

4.7.8 Inference Rules

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

theorem *cr-eval-imp-ccomp-max-elim*[*simp*]:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
assumes *condorcet-rating e*
shows *condorcet-compatibility* (*max-eliminator e*)
⟨*proof*⟩

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

theorem *cr-eval-imp-dcc-max-elim*[*simp*]:
fixes *e* :: (*'a*, *'v*) *Evaluation-Function*
assumes *condorcet-rating e*
shows *defer-condorcet-consistency* (*max-eliminator e*)
⟨*proof*⟩

end

4.8 Aggregator

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

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

4.8.1 Definition

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

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

4.8.2 Properties

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

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

end

4.9 Maximum Aggregator

theory *Maximum-Aggregator*
imports *Aggregator*
begin

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

4.9.1 Definition

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

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

4.9.2 Auxiliary Lemma

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

4.9.3 Soundness

theorem *max-agg-sound*[*simp*]: *aggregator max-aggregator*
⟨*proof*⟩

4.9.4 Properties

The max-aggregator is conservative.

theorem *max-agg-consv*[*simp*]: *agg-conservative max-aggregator*
⟨*proof*⟩

The max-aggregator is commutative.

theorem *max-agg-comm*[*simp*]: *agg-commutative max-aggregator*
⟨*proof*⟩

end

4.10 Termination Condition

theory *Termination-Condition*
imports *Social-Choice-Types/Result*

begin

The termination condition is used in loops. It decides whether or not to terminate the loop after each iteration, depending on the current state of the loop.

4.10.1 Definition

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

end

4.11 Defer Equal Condition

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

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

4.11.1 Definition

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

end

Chapter 5

Basic Modules

5.1 Defer Module

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

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

5.1.1 Definition

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

5.1.2 Soundness

theorem *def-mod-sound*[*simp*]: $\mathcal{SCF}$*-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* (λ v. $v \in V$))

fun *elect-first-module* :: ($'a$, $'v$::*wellorder*, $'a$ *Result*) *Electoral-Module* **where**
elect-first-module V A p =
({$a \in A$. *above* (p (*least* V)) $a = \{a\}$},
{$a \in A$. *above* (p (*least* V)) $a \neq \{a\}$},
{})

5.2.2 Soundness

theorem *elect-first-mod-sound*: $\mathcal{SCF}$*-result.electoral-module elect-first-module*
⟨*proof*⟩

end

5.3 Consensus Class

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

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

5.3.1 Definition

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

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

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

5.3.2 Consensus Choice

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

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

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

A consensus class is deemed well-formed if the result of its mapping is completely determined by its consensus, the elected set of the electoral module's result.

definition *well-formed* :: ('*a*, '*v*) *Consensus* ⇒ ('*a*, '*v*, '*r*) *Electoral-Module*
          ⇒ *bool* **where**
  well-formed c m ≡
    ∀ *A V V' p p'*.
      profile V A p ∧ *profile V' A p'* ∧ *c* (*A*, *V*, *p*) ∧ *c* (*A*, *V'*, *p'*)
        ⟶ *m V A p* = *m V' A p'*

A sensible social choice rule for a given arbitrary consensus and social choice rule r is the one that chooses the result of r for all consensus elections and defers all candidates otherwise.

fun *consensus-choice* :: ('*a*, '*v*) *Consensus* ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module*
      ⇒ ('*a*, '*v*, '*a Result*) *Consensus-Class* **where**
  consensus-choice c m =
    (*let*
      w = (λ *V A p*. *if c* (*A*, *V*, *p*) *then m V A p else defer-module V A p*)
      in (*c*, *w*))

5.3.3 Auxiliary Lemmas

lemma *unanimity'-consensus-imp-elect-fst-mod-well-formed*:
  fixes *a* :: '*a*
  shows *well-formed*
      (λ *c*. *nonempty-set*$_\mathcal{C}$ *c* ∧ *nonempty-profile*$_\mathcal{C}$ *c*
          ∧ *equal-top*$_\mathcal{C}$' *a c*) *elect-first-module*
⟨*proof*⟩

lemma *strong-unanimity'consensus-imp-elect-fst-mod-completely-determined*:
  fixes *r* :: '*a Preference-Relation*
  shows *well-formed*

(λ *c. nonempty-set*$_{\mathcal{C}}$ *c* $\wedge$ *nonempty-profile*$_{\mathcal{C}}$ *c* $\wedge$ *equal-vote*$_{\mathcal{C}}$*'* *r c*) *elect-first-module*
⟨*proof*⟩

lemma *strong-unanimity'consensus-imp-elect-fst-mod-well-formed*:
fixes *r* :: *'a Preference-Relation*
shows *well-formed*
(λ *c. nonempty-set*$_{\mathcal{C}}$ *c* $\wedge$ *nonempty-profile*$_{\mathcal{C}}$ *c*
$\wedge$ *equal-vote*$_{\mathcal{C}}$*'* *r c*) *elect-first-module*
⟨*proof*⟩

lemma *cons-domain-valid*:
fixes *C* :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
shows *elections-*$\mathcal{K}$ *C* $\subseteq$ *valid-elections*
⟨*proof*⟩

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

5.3.4 Consensus Rules

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

Unanimity condition.

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

Strong unanimity condition.

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

5.3.5 Properties

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

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

is-symmetry (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*Invariance* ($anonymity_{\mathcal{R}}$ *X*))

fun (**in** *result-properties*) *consensus-rule-neutrality* :: (′*a*, ′*v*) *Election set*
⇒ (′*a*, ′*v*, ′*b Result*) *Consensus-Class* ⇒ *bool* **where**
consensus-rule-neutrality X C =
is-symmetry (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*))
(*action-induced-equivariance*
(*carrier* $neutrality_{\mathcal{G}}$) *X* (*φ-neutr X*) (*set-action ψ-neutr*))

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

5.3.6 Inference Rules

lemma *consensus-choice-equivar*:
fixes
m :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **and**
c :: (′*a*, ′*v*) *Consensus* **and**
G :: ′*x set* **and**
X :: (′*a*, ′*v*) *Election set* **and**
φ :: (′*x*, (′*a*, ′*v*) *Election*) *binary-fun* **and**
ψ :: (′*x*, ′*a*) *binary-fun* **and**
f :: ′*a Result* ⇒ ′*a set*
defines *equivar* ≡ *action-induced-equivariance G X φ* (*set-action ψ*)
assumes
equivar-m: *is-symmetry* (*f* ∘ $fun_{\mathcal{E}}$ *m*) *equivar* **and**
equivar-defer: *is-symmetry* (*f* ∘ $fun_{\mathcal{E}}$ *defer-module*) *equivar* **and**
— This could be generalized to arbitrary modules instead of *defer-module*.
invar-cons: *is-symmetry c* (*Invariance* (*action-induced-rel G X φ*))
shows *is-symmetry* (*f* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ (*consensus-choice c m*)))
(*action-induced-equivariance G X φ* (*set-action ψ*))
⟨*proof*⟩

lemma *consensus-choice-anonymous*:
fixes
α :: (′*a*, ′*v*) *Consensus* **and**
β :: (′*a*, ′*v*) *Consensus* **and**
m :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **and**
β′ :: ′*b* ⇒ (′*a*, ′*v*) *Consensus*
assumes
beta-sat: *β* = (λ *E*. ∃ *a*. *β*′ *a E*) **and**
beta′*-anon*: ∀ *x*. *consensus-anonymity* (*β*′ *x*) **and**
anon-cons-cond: *consensus-anonymity α* **and**
conditions-univ: ∀ *x*. *well-formed* (λ *E*. *α E* ∧ *β*′ *x E*) *m*
shows *consensus-rule-anonymity* (*consensus-choice* (λ *E*. *α E* ∧ *β E*) *m*)
⟨*proof*⟩

5.3.7 Theorems

Anonymity

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

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

Neutrality

lemma *defer-winners-equivariant*:
  fixes
    G :: *'x set* **and**
    X :: (*'a*, *'v*) *Election set* **and**
    φ :: (*'x*, (*'a*, *'v*) *Election*) *binary-fun* **and**
    ψ :: (*'x*, *'a*) *binary-fun*
  shows *is-symmetry* (*elect-r* ∘ $fun_{\mathcal{E}}$ *defer-module*)
        (*action-induced-equivariance* G X φ (*set-action* ψ))
  ⟨*proof*⟩

lemma *elect-first-winners-neutral*: *is-symmetry* (*elect-r* ∘ $fun_{\mathcal{E}}$ *elect-first-module*)
    (*action-induced-equivariance* (*carrier* $neutrality_{\mathcal{G}}$)
      valid-elections (φ-*neutr valid-elections*) (*set-action* ψ-$neutr_{c}$))
⟨*proof*⟩

lemma *strong-unanimity-neutral*:
  defines *domain* ≡ *valid-elections* ∩ *Collect* $strong\text{-}unanimity_{\mathcal{C}}$
  — We want to show neutrality on a set as general as possible, as this implies subset neutrality.
  shows $\mathcal{SCF}$-*properties.consensus-rule-neutrality domain strong-unanimity*
⟨*proof*⟩

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

lemma *strong-unanimity-closed-under-neutrality*: *closed-restricted-rel*
    ($neutrality_{\mathcal{R}}$ *valid-elections*) *valid-elections* (*elections*-$\mathcal{K}$ *strong-unanimity*)
⟨*proof*⟩

end

5.4 Distance Rationalization

theory *Distance-Rationalization*
  imports *Social-Choice-Types/Refined-Types/Preference-List*

Consensus-Class
Distance
begin

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

5.4.1 Definitions

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

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

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

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

5.4.2 Standard Definitions

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

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

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

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

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

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

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

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

fun (**in** *result*) *distance-*$\mathcal{R}$*-std* :: ('*a*, '*v*) *Election Distance*
⇒ ('*a*, '*v*, '*r Result*) *Consensus-Class*
⇒ ('*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-set 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* ⊆ *finite-elections*
⟨*proof*⟩

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

lemma *list-cons-presv-finiteness*:
fixes
A :: '*a set* **and**
S :: '*a list set*
assumes
fin-A: *finite A* **and**
fin-B: *finite S*
shows *finite* {*a*#*l* | *a l*. *a* ∈ *A* ∧ *l* ∈ *S*}

$\langle proof \rangle$

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

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 $\forall\ l'\ i{::}nat.\ l' \in listset\ l \wedge i < length\ l' \longrightarrow l'!i \in l!i$
$\langle proof \rangle$

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

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

5.4.4 Soundness

lemma (**in** *result*) $\mathcal{R}$*-sound*:
  fixes

K :: ($'a$, $'v$, $'r$ *Result*) *Consensus-Class* **and**
d :: ($'a$, $'v$) *Election Distance*
shows *electoral-module* (*distance-$\mathcal{R}$* d K)
⟨*proof*⟩

5.4.5 Inference Rules

lemma *is-arg-min-equal*:
fixes
f :: $'a \Rightarrow 'b{::}ord$ **and**
g :: $'a \Rightarrow 'b$ **and**
S :: $'a$ *set* **and**
x :: $'a$
assumes $\forall\ x \in S.\ f\ x = g\ x$
shows *is-arg-min* f (λ s. $s \in S$) x = *is-arg-min* g (λ s. $s \in S$) x
⟨*proof*⟩

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

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

end

5.5 Votewise Distance Rationalization

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

Votewise-Distance
begin

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

5.5.1 Common Rationalizations

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

5.5.2 Theorems

lemma *votewise-non-voters-irrelevant*:
fixes
d :: *'a Vote Distance* **and**
N :: *Norm*
shows *voters-determine-distance* (*votewise-distance d N*)
⟨*proof*⟩

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*
⇒ (*'a*, *'v*, *'a Result*) *Consensus-Class*
⇒ (*'a*, *'v*) *Election* ⇒ *'a* ⇒ *ereal*

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

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

type-synonym (*'a*, *'v*) *dist-type* = (*'a*, *'v*) *Election Distance*
⇒ (*'a*, *'v*, *'a Result*) *Consensus-Class*
⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module*

lemma *equal-score-swap*: (*score*::((*'a*, *'v*::*linorder*) *score-type*))
(*votewise-distance swap l-one*) =
score-std (*votewise-distance swap l-one*)
⟨*proof*⟩

lemma *swap-$\mathcal{R}$-code*[*code*]: *swap-$\mathcal{R}$* =
  ($\mathcal{SCF}$*-result.distance-$\mathcal{R}$-std*::(('*a*, '*v*::*linorder*) *dist-rat-std-type*))
    (*votewise-distance swap l-one*)
⟨*proof*⟩

end

5.6 Symmetry in Distance-Rationalizable Rules

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

5.6.1 Minimizer Function

fun *distance-infimum* :: '*x Distance* ⇒ '*x set* ⇒ '*x* ⇒ *ereal* **where**
  distance-infimum d X a = *Inf* (*d a* ‘ *X*)

fun *closest-preimg-distance* :: ('*x* ⇒ '*y*) ⇒ '*x set* ⇒ '*x Distance*
    ⇒ '*x* ⇒ '*y* ⇒ *ereal* **where**
  closest-preimg-distance f $domain_f$ *d x y* =
    distance-infimum d (*preimg f* $domain_f$ *y*) *x*

fun *minimizer* :: ('*x* ⇒ '*y*) ⇒ '*x set* ⇒ '*x Distance* ⇒ '*y set* ⇒ '*x* ⇒ '*y set* **where**
  minimizer f $domain_f$ *d Y x* =
    arg-min-set (*closest-preimg-distance f* $domain_f$ *d x*) *Y*

Auxiliary Lemmas

lemma *rewrite-arg-min-set*:
  fixes
    f :: '*x* ⇒ '*y*::*linorder* **and**
    X :: '*x set*
  shows *arg-min-set f X* = ⋃ (*preimg f X* ‘ {*y* ∈ (*f* ‘ *X*). ∀ *z* ∈ *f* ‘ *X*. *y* ≤ *z*})
⟨*proof*⟩

Equivariance

lemma *restr-induced-rel*:
  fixes
    X :: '*x set* **and**
    Y :: '*y set* **and**
    Y' :: '*y set* **and**
    φ :: ('*x*, '*y*) *binary-fun*
  assumes *Y'* ⊆ *Y*
  shows *Restr* (*action-induced-rel X Y φ*) *Y'* = *action-induced-rel X Y' φ*
  ⟨*proof*⟩

theorem *group-action-invar-dist-and-equivar-f-imp-equivar-minimizer*:
fixes
f :: $'x \Rightarrow 'y$ **and**
$domain_f$ :: $'x$ *set* **and**
d :: $'x$ *Distance* **and**
valid-img :: $'x \Rightarrow 'y$ *set* **and**
X :: $'x$ *set* **and**
G :: $'z$ *monoid* **and**
φ :: $('z, 'x)$ *binary-fun* **and**
ψ :: $('z, 'y)$ *binary-fun*
defines *equivar-prop-set-valued* $\equiv$
action-induced-equivariance (*carrier* G) X φ (*set-action* ψ)
assumes
action-φ: *group-action* G X φ **and**
group-action-res: *group-action* G *UNIV* ψ **and**
dom-in-X: $domain_f \subseteq X$ **and**
closed-domain:
closed-restricted-rel (*action-induced-rel* (*carrier* G) X φ) X $domain_f$ **and**
equivar-img: *is-symmetry valid-img equivar-prop-set-valued* **and**
invar-d: $invariance_{\mathcal{D}}$ d (*carrier* G) X φ **and**
equivar-f:
is-symmetry f (*action-induced-equivariance* (*carrier* G) $domain_f$ φ ψ)
shows *is-symmetry* (λ x. *minimizer* f $domain_f$ d (*valid-img* x) x) *equivar-prop-set-valued*
⟨*proof*⟩

Invariance

lemma *closest-dist-invar-under-refl-rel-and-tot-invar-dist*:
fixes
f :: $'x \Rightarrow 'y$ **and**
$domain_f$ :: $'x$ *set* **and**
d :: $'x$ *Distance* **and**
rel :: $'x$ *rel*
assumes
r-refl: *refl-on* $domain_f$ (*Restr rel* $domain_f$) **and**
tot-invar-d: $total\text{-}invariance_{\mathcal{D}}$ 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 :: $'x \Rightarrow 'y$ **and**
$domain_f$ :: $'x$ *set* **and**
d :: $'x$ *Distance* **and**
rel :: $'x$ *rel* **and**
img :: $'y$ *set*
assumes
r-refl: *refl-on* $domain_f$ (*Restr rel* $domain_f$) **and**

tot-invar-d: *total-invariance*$_{\mathcal{D}}$ *d rel*
shows *is-symmetry* (*minimizer f domain*$_f$ *d img*) (*Invariance rel*)
⟨*proof*⟩

theorem *group-act-invar-dist-and-invar-f-imp-invar-minimizer*:
fixes
f :: $'x \Rightarrow {'y}$ **and**
domain$_f$ :: *'x set* **and**
d :: *'x Distance* **and**
img :: *'y set* **and**
X :: *'x set* **and**
G :: *'z monoid* **and**
φ :: ('*z*, '*x*) *binary-fun*
defines
rel ≡ *action-induced-rel* (*carrier G*) *X* φ **and**
rel' ≡ *action-induced-rel* (*carrier G*) *domain*$_f$ φ
assumes
action-φ: *group-action G X* φ **and**
domain$_f$ ⊆ *X* **and**
closed-domain: *closed-restricted-rel rel X domain*$_f$ **and**

invar-d: *invariance*$_{\mathcal{D}}$ *d* (*carrier G*) *X* φ **and**
invar-f: *is-symmetry f* (*Invariance rel'*)
shows *is-symmetry* (*minimizer f domain*$_f$ *d img*) (*Invariance rel*)
⟨*proof*⟩

5.6.2 Distance Rationalization as Minimizer

lemma $\mathcal{K}_{\mathcal{E}}$*-is-preimg*:
fixes
d :: ('*a*, '*v*) *Election Distance* **and**
C :: ('*a*, '*v*, '*r Result*) *Consensus-Class* **and**
E :: ('*a*, '*v*) *Election* **and**
w :: '*r*
shows *preimg* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) $\{w\}$ = $\mathcal{K}_{\mathcal{E}}$ *C w*
⟨*proof*⟩

lemma *score-is-closest-preimg-dist*:
fixes
d :: ('*a*, '*v*) *Election Distance* **and**
C :: ('*a*, '*v*, '*r Result*) *Consensus-Class* **and**
E :: ('*a*, '*v*) *Election* **and**
w :: '*r*
shows *score d C E w* =
closest-preimg-distance (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d E* $\{w\}$
⟨*proof*⟩

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

d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r Result) *Consensus-Class*
shows $fun_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ d C) =
(λ E. $\bigcup$ (*minimizer* (*elect-r* $\circ$ $fun_{\mathcal{E}}$ (*rule-$\mathcal{K}$* C)) (*elections-$\mathcal{K}$* C) d
(*singleton-set-system* (*limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*)) E))
⟨*proof*⟩

Invariance

theorem (**in** *result*) *tot-invar-dist-imp-invar-dr-rule*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r Result) *Consensus-Class* **and**
rel :: ('a, 'v) *Election rel*
assumes
r-refl: *refl-on* (*elections-$\mathcal{K}$* C) (*Restr rel* (*elections-$\mathcal{K}$* C)) **and**
tot-invar-d: *total-invariance*$_{\mathcal{D}}$ d *rel* **and**
invar-res:
is-symmetry (λ E. *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*)
(*Invariance rel*)
shows *is-symmetry* ($fun_{\mathcal{E}}$ (*distance-$\mathcal{R}$* d C)) (*Invariance rel*)
⟨*proof*⟩

theorem (**in** *result*) *invar-dist-cons-imp-invar-dr-rule*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r Result) *Consensus-Class* **and**
G :: 'x *monoid* **and**
φ :: ('x, ('a, 'v) *Election*) *binary-fun* **and**
B :: ('a, 'v) *Election set*
defines
rel ≡ *action-induced-rel* (*carrier* G) B φ **and**
rel' ≡ *action-induced-rel* (*carrier* G) (*elections-$\mathcal{K}$* C) φ
assumes
action-φ: *group-action* G B φ **and**
consensus-C-in-B: *elections-$\mathcal{K}$* $C \subseteq B$ **and**
closed-domain:
closed-restricted-rel rel B (*elections-$\mathcal{K}$* C) **and**
invar-res:
is-symmetry (λ E. *limit-set* (*alternatives-$\mathcal{E}$* E) *UNIV*) (*Invariance rel*) **and**
invar-d: *invariance*$_{\mathcal{D}}$ d (*carrier* G) B φ **and**
invar-C-winners: *is-symmetry* (*elect-r* $\circ$ $fun_{\mathcal{E}}$ (*rule-$\mathcal{K}$* C)) (*Invariance rel'*)
shows *is-symmetry* ($fun_{\mathcal{E}}$ (*distance-$\mathcal{R}$* d C)) (*Invariance rel*)
⟨*proof*⟩

Equivariance

theorem (**in** *result*) *invar-dist-equivar-cons-imp-equivar-dr-rule*:
fixes
d :: ('a, 'v) *Election Distance* **and**

C :: ('a, 'v, 'r Result) *Consensus-Class* **and**
G :: 'x *monoid* **and**
φ :: ('x, ('a, 'v) *Election*) *binary-fun* **and**
ψ :: ('x, 'r) *binary-fun* **and**
B :: ('a, 'v) *Election set*
defines
rel $\equiv$ *action-induced-rel* (*carrier* G) B φ **and**
rel' $\equiv$ *action-induced-rel* (*carrier* G) (*elections-*$\mathcal{K}$ C) φ **and**
equivar-prop $\equiv$
action-induced-equivariance (*carrier* G) (*elections-*$\mathcal{K}$ C)
φ (*set-action* ψ) **and**
equivar-prop-global-set-valued $\equiv$
action-induced-equivariance (*carrier* G) B φ (*set-action* ψ) **and**
equivar-prop-global-result-valued $\equiv$
action-induced-equivariance (*carrier* G) B φ (*result-action* ψ)
assumes
action-φ: *group-action* G B φ **and**
group-act-res: *group-action* G *UNIV* ψ **and**
cons-elect-set: *elections-*$\mathcal{K}$ $C \subseteq B$ **and**
closed-domain: *closed-restricted-rel rel* B (*elections-*$\mathcal{K}$ C) **and**
equivar-res:
is-symmetry (λ E. *limit-set* (*alternatives-*$\mathcal{E}$ E) *UNIV*)
equivar-prop-global-set-valued **and**
invar-d: *invariance*$_{\mathcal{D}}$ d (*carrier* G) B φ **and**
equivar-C-winners: *is-symmetry* (*elect-r* $\circ$ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ C)) *equivar-prop*
shows *is-symmetry* (*fun*$_{\mathcal{E}}$ (*distance-*$\mathcal{R}$ d C)) *equivar-prop-global-result-valued*
⟨*proof*⟩

5.6.3 Symmetry Property Inference Rules

theorem (**in** *result*) *anon-dist-and-cons-imp-anon-dr*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r Result) *Consensus-Class*
assumes
anon-d: *distance-anonymity' valid-elections* d **and**
anon-C: *consensus-rule-anonymity'* (*elections-*$\mathcal{K}$ C) C **and**
closed-C: *closed-restricted-rel* (*anonymity*$_{\mathcal{R}}$ *valid-elections*)
valid-elections (*elections-*$\mathcal{K}$ C)
shows *anonymity' valid-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
neutr-d: *distance-neutrality valid-elections* d **and**
neutr-C: *consensus-rule-neutrality* (*elections-*$\mathcal{K}$ C) C **and**

closed-C: *closed-restricted-rel* (*neutrality*$_\mathcal{R}$ *valid-elections*)
valid-elections (*elections-*$\mathcal{K}$ *C*)
shows *neutrality valid-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
rev-sym-d: *distance-reversal-symmetry valid-elections d* **and**
rev-sym-C: *consensus-rule-reversal-symmetry* (*elections-*$\mathcal{K}$ *C*) *C* **and**
closed-C: *closed-restricted-rel* (*reversal*$_\mathcal{R}$ *valid-elections*)
valid-elections (*elections-*$\mathcal{K}$ *C*)
shows *reversal-symmetry valid-elections* ($\mathcal{SWF}$*-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*) *Election Distance* **and**
C :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
assumes *distance-homogeneity' finite-elections-*$\mathcal{V}$ *d*
shows *homogeneity' finite-elections-*$\mathcal{V}$ (*distance-*$\mathcal{R}$ *d C*)
⟨*proof*⟩

5.6.4 Further Properties

fun *decisiveness* :: (*'a*, *'v*) *Election set* ⇒ (*'a*, *'v*) *Election Distance* ⇒
(*'a*, *'v*, *'r Result*) *Electoral-Module* ⇒ *bool* **where**
decisiveness X d m =
($\nexists$ *E*. *E* ∈ *X* ∧ (∃ δ > *0*. ∀ *E'* ∈ *X*. *d E E'* < δ ⟶ *card* (*elect-r* (*fun*$_\mathcal{E}$ *m* *E'*)) > *1*))

end

5.7 Distance Rationalization on Election Quotients

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

Distance-Rationalization-Symmetry
begin

5.7.1 Quotient Distances

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

fun *relation-paths* :: $'x$ *rel* $\Rightarrow$ $'x$ *list set* **where**
relation-paths $r =$
$\{p.\ \exists\ k.\ (length\ p = 2 * k \wedge (\forall\ i < k.\ (p!(2 * i),\ p!(2 * i + 1)) \in r))\}$

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

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

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

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

fun *simple* :: $'x$ *rel* $\Rightarrow$ $'x$ *set* $\Rightarrow$ $'x$ *Distance* $\Rightarrow$ *bool* **where**
simple $r\ X\ d =$
$(\forall\ A \in X\ //\ r.$
$(\exists\ a \in A.\ \forall\ B \in X\ //\ r.$
$distance\text{-}infimum_{\mathcal{Q}}\ d\ A\ B = Inf\ \{d\ a\ b\ |\ b.\ b \in B\}))$
— We call a distance simple with respect to a relation if for all relation classes, there is an a in A that minimizes the infimum distance between A and all B such that the infimum distance between these sets coincides with the infimum distance over all b in B for a fixed a.

fun *product'* :: $'x$ *rel* $\Rightarrow$ $('x * 'x)$ *rel* **where**
product' $r = \{(p_1,\ p_2).\ ((fst\ p_1,\ fst\ p_2) \in r \wedge snd\ p_1 = snd\ p_2)$
$\vee\ ((snd\ p_1,\ snd\ p_2) \in r \wedge fst\ p_1 = fst\ p_2)\}$

Auxiliary Lemmas

lemma *tot-dist-invariance-is-congruence*:
fixes
d :: $'x$ *Distance* **and**
r :: $'x$ *rel*

shows $(total\text{-}invariance_{\mathcal{D}}\ d\ r) = (tup\ d\ respects\ (product\ r))$
⟨*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: $refl\text{-}on\ X\ r \Longrightarrow refl\text{-}on\ (X \times X)\ (product\ r)$ **and**
sym: $sym\text{-}on\ X\ r \Longrightarrow sym\text{-}on\ (X \times X)\ (product\ r)$
⟨*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: $total\text{-}invariance_{\mathcal{D}}\ d\ r$
shows $\forall\ A\ B.\ A \in X\ //\ r \wedge B \in X\ //\ r$
$\longrightarrow (\forall\ a\ b.\ a \in A \wedge b \in B \longrightarrow distance_{\mathcal{Q}}\ d\ A\ B = d\ a\ b)$
⟨*proof*⟩

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

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

lemma *quotient-dist-coincides-with-dist*$_{\mathcal{Q}}$:

fixes
  d :: $'x$ *Distance* **and**
  r :: $'x$ *rel* **and**
  X :: $'x$ *set*
assumes
  equiv: *equiv* X r **and**
  tri: *triangle-ineq* X d **and**
  invar: *total-invariance*$_{\mathcal{D}}$ d r
shows $\forall\ A \in X\ //\ r.\ \forall\ B \in X\ //\ r.$ *quotient-dist* r d A B = *distance*$_{\mathcal{Q}}$ d A B
⟨*proof*⟩

lemma *inf-dist-coincides-with-dist*$_{\mathcal{Q}}$:
fixes
  d :: $'x$ *Distance* **and**
  r :: $'x$ *rel* **and**
  X :: $'x$ *set*
assumes
  equiv-X-r: *equiv* X r **and**
  tot-inv-d-r: *total-invariance*$_{\mathcal{D}}$ d r
shows $\forall\ A \in X\ //\ r.\ \forall\ B \in X\ //\ r.$
    distance-infimum$_{\mathcal{Q}}$ d A B = *distance*$_{\mathcal{Q}}$ d A B
⟨*proof*⟩

lemma *inf-helper*:
fixes
  A :: $'x$ *set* **and**
  B :: $'x$ *set* **and**
  d :: $'x$ *Distance*
shows *Inf* $\{d\ a\ b \mid a\ b.\ a \in A \wedge b \in B\}$ =
    Inf $\{$*Inf* $\{d\ a\ b \mid b.\ b \in B\} \mid a.\ a \in A\}$
⟨*proof*⟩

lemma *invar-dist-simple*:
fixes
  d :: $'y$ *Distance* **and**
  G :: $'x$ *monoid* **and**
  Y :: $'y$ *set* **and**
  φ :: $('x, 'y)$ *binary-fun*
assumes
  action-φ: *group-action* G Y φ **and**
  invar: *invariance*$_{\mathcal{D}}$ d (*carrier* G) Y φ
shows *simple* (*action-induced-rel* (*carrier* G) Y φ) Y d
⟨*proof*⟩

lemma *tot-invar-dist-simple*:
fixes
  d :: $'x$ *Distance* **and**
  r :: $'x$ *rel* **and**
  X :: $'x$ *set*

assumes
  equiv-on-X: *equiv X r* **and**
  invar: *total-invariance*$_{\mathcal{D}}$ *d r*
shows *simple r X d*
⟨*proof*⟩

5.7.2 Quotient Consensus and Results

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

fun (**in** *result*) *limit-set*$_{\mathcal{Q}}$ :: ('*a*, '*v*) *Election set* ⇒ '*r set* ⇒ '*r set* **where**
limit-set$_{\mathcal{Q}}$ *X res* = ⋂ {*limit-set* (*alternatives-*$\mathcal{E}$ *E*) *res* | *E*. *E* ∈ *X*}

Auxiliary Lemmas

lemma *closed-under-equiv-rel-subset*:
fixes
  X :: '*x set* **and**
  Y :: '*x set* **and**
  Z :: '*x set* **and**
  r :: '*x rel*
assumes
  equiv X r **and**
  Y ⊆ *X* **and**
  Z ⊆ *X* **and**
  Z ∈ *Y* // *r* **and**
  closed-restricted-rel r X Y
shows *Z* ⊆ *Y*
⟨*proof*⟩

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

lemma (**in** *result*) *preimg-invar*:
fixes
  f :: '*x* ⇒ '*y* **and**

$domain_f$ :: *'x set* **and**
d :: *'x Distance* **and**
r :: *'x rel* **and**
X :: *'x set*
assumes
equiv-rel: *equiv X r* **and**
cons-subset: $domain_f \subseteq X$ **and**
closed-domain: *closed-restricted-rel r X* $domain_f$ **and**
invar-f: *is-symmetry f* (*Invariance* (*Restr r* $domain_f$))
shows $\forall\ y.$ (*preimg f* $domain_f$ y) // r = *preimg* ($\pi_{\mathcal{Q}}$ f) ($domain_f$ // r) y
⟨*proof*⟩

lemma *minimizer-helper*:
fixes
f :: *'x* ⇒ *'y* **and**
$domain_f$ :: *'x set* **and**
d :: *'x Distance* **and**
Y :: *'y set* **and**
x :: *'x* **and**
y :: *'y*
shows $y \in$ *minimizer f* $domain_f$ *d Y x* =
($y \in Y \wedge (\forall\ y' \in Y.$
Inf (*d x* ‘ (*preimg f* $domain_f$ y)) ≤ *Inf* (*d x* ‘ (*preimg f* $domain_f$ y'))))
⟨*proof*⟩

lemma *rewr-singleton-set-system-union*:
fixes
Y :: *'x set set* **and**
X :: *'x set*
assumes $Y \subseteq$ *singleton-set-system X*
shows
singleton-set-union: $x \in \bigcup\ Y \longleftrightarrow \{x\} \in Y$ **and**
obtain-singleton: $A \in$ *singleton-set-system X* $\longleftrightarrow (\exists\ x \in X.\ A = \{x\})$
⟨*proof*⟩

lemma *union-inf*:
fixes X :: *ereal set set*
shows *Inf* {*Inf A* | *A*. $A \in X$} = *Inf* ($\bigcup\ X$)
⟨*proof*⟩

5.7.3 Quotient Distance Rationalization

fun (**in** *result*) $\mathcal{R}_{\mathcal{Q}}$ :: (*'a*, *'v*) *Election rel* ⇒ (*'a*, *'v*) *Election Distance*
⇒ (*'a*, *'v*, *'r Result*) *Consensus-Class* ⇒ (*'a*, *'v*) *Election set* ⇒ *'r set* **where**
$\mathcal{R}_{\mathcal{Q}}$ *r d C A* =
$\bigcup$ (*minimizer* ($\pi_{\mathcal{Q}}$ (*elect-r* ∘ $fun_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*))) (*elections-*$\mathcal{K}_{\mathcal{Q}}$ *r C*)
(*distance-infimum*$_{\mathcal{Q}}$ *d*) (*singleton-set-system* (*limit-set*$_{\mathcal{Q}}$ *A UNIV*)) *A*)

fun (**in** *result*) *distance-*$\mathcal{R}_{\mathcal{Q}}$ :: (*'a*, *'v*) *Election rel* ⇒ (*'a*, *'v*) *Election Distance*

$\Rightarrow$ ('a, 'v, 'r Result) Consensus-Class
$\Rightarrow$ ('a, 'v) Election set $\Rightarrow$ 'r Result **where**
distance-$\mathcal{R}_\mathcal{Q}$ *r d C A* =
($\mathcal{R}_\mathcal{Q}$ *r d C A*,
$\pi_\mathcal{Q}$ (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) *A* − $\mathcal{R}_\mathcal{Q}$ *r d C A*,
{})

Hadjibeyli and Wilson 2016 4.17

theorem (**in** *result*) *invar-dr-simple-dist-imp-quotient-dr-winners*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r *Result*) *Consensus-Class* **and**
r :: ('a, 'v) *Election rel* **and**
X :: ('a, 'v) *Election set* **and**
A :: ('a, 'v) *Election set*
assumes
simple: *simple r X d* **and**
closed-domain: *closed-restricted-rel r X* (*elections-*$\mathcal{K}$ *C*) **and**
invar-res:
is-symmetry (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*) (*Invariance r*) **and**
invar-C: *is-symmetry* (*elect-r* $\circ$ $fun_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))
(*Invariance* (*Restr r* (*elections-*$\mathcal{K}$ *C*))) **and**
invar-dr: *is-symmetry* ($fun_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*)) (*Invariance r*) **and**
quot-class: $A \in X$ // *r* **and**
equiv-rel: *equiv X r* **and**
cons-subset: *elections-*$\mathcal{K}$ $C \subseteq X$
shows $\pi_\mathcal{Q}$ ($fun_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*)) *A* = $\mathcal{R}_\mathcal{Q}$ *r d C A*
⟨*proof*⟩

theorem (**in** *result*) *invar-dr-simple-dist-imp-quotient-dr*:
fixes
d :: ('a, 'v) *Election Distance* **and**
C :: ('a, 'v, 'r *Result*) *Consensus-Class* **and**
r :: ('a, 'v) *Election rel* **and**
X :: ('a, 'v) *Election set* **and**
A :: ('a, 'v) *Election set*
assumes
simple: *simple r X d* **and**
closed-domain: *closed-restricted-rel r X* (*elections-*$\mathcal{K}$ *C*) **and**
invar-res:
is-symmetry (λ *E. limit-set* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)
(*Invariance r*) **and**
invar-C: *is-symmetry* (*elect-r* $\circ$ $fun_\mathcal{E}$ (*rule-*$\mathcal{K}$ *C*))
(*Invariance* (*Restr r* (*elections-*$\mathcal{K}$ *C*))) **and**
invar-dr: *is-symmetry* ($fun_\mathcal{E}$ ($\mathcal{R}_\mathcal{W}$ *d C*)) (*Invariance r*) **and**
quot-class: $A \in X$ // *r* **and**
equiv-rel: *equiv X r* **and**
cons-subset: *elections-*$\mathcal{K}$ $C \subseteq X$
shows $\pi_\mathcal{Q}$ ($fun_\mathcal{E}$ (*distance-*$\mathcal{R}$ *d C*)) *A* = *distance-*$\mathcal{R}_\mathcal{Q}$ *r d C A*

⟨*proof*⟩

end

5.8 Result and Property Locale Code Generation

theory *Interpretation-Code*
 imports *Electoral-Module*
  Distance-Rationalization
begin
⟨*ML*⟩

Lemmas stating the explicit instantiations of interpreted abstract functions from locales.

lemma *electoral-module-$\mathcal{SCF}$-code-lemma*:
 fixes m :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module*
 shows *$\mathcal{SCF}$-result.electoral-module* m =
  (∀ *A V p*. *profile V A p* ⟶ *well-formed-$\mathcal{SCF}$ A* (*m V A p*))
 ⟨*proof*⟩

lemma *$\mathcal{R}_\mathcal{W}$-$\mathcal{SCF}$-code-lemma*:
 fixes
  d :: (′*a*, ′*v*) *Election Distance* **and**
  K :: (′*a*, ′*v*, ′*a Result*) *Consensus-Class* **and**
  V :: ′*v set* **and**
  A :: ′*a set* **and**
  p :: (′*a*, ′*v*) *Profile*
 shows *$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$ d K V A p* =
   arg-min-set (*score d K* (*A*, *V*, *p*)) (*limit-set-$\mathcal{SCF}$ A UNIV*)
 ⟨*proof*⟩

lemma *distance-$\mathcal{R}$-$\mathcal{SCF}$-code-lemma*:
 fixes
  d :: (′*a*, ′*v*) *Election Distance* **and**
  K :: (′*a*, ′*v*, ′*a Result*) *Consensus-Class* **and**
  V :: ′*v set* **and**
  A :: ′*a set* **and**
  p :: (′*a*, ′*v*) *Profile*
 shows *$\mathcal{SCF}$-result.distance-$\mathcal{R}$ d K V A p* =
  (*$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$ d K V A p*,
   (*limit-set-$\mathcal{SCF}$ A UNIV*) − *$\mathcal{SCF}$-result.$\mathcal{R}_\mathcal{W}$ d K V A p*,
   {})
 ⟨*proof*⟩

lemma *$\mathcal{R}_\mathcal{W}$-std-$\mathcal{SCF}$-code-lemma*:
 fixes
  d :: (′*a*, ′*v*) *Election Distance* **and**

K :: ('a, 'v, 'a *Result*) *Consensus-Class* **and**
V :: 'v *set* **and**
A :: 'a *set* **and**
p :: ('a, 'v) *Profile*
shows $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$-*std* d K V A p =
arg-min-set (*score-std* d K (A, V, p)) (*limit-set*-$\mathcal{SCF}$ A *UNIV*)
⟨*proof*⟩

lemma *distance*-$\mathcal{R}$-*std*-$\mathcal{SCF}$-*code-lemma*:
fixes
d :: ('a, 'v) *Election Distance* **and**
K :: ('a, 'v, 'a *Result*) *Consensus-Class* **and**
V :: 'v *set* **and**
A :: 'a *set* **and**
p :: ('a, 'v) *Profile*
shows $\mathcal{SCF}$-*result*.*distance*-$\mathcal{R}$-*std* d K V A p =
($\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$-*std* d K V A p,
(*limit-set*-$\mathcal{SCF}$ A *UNIV*) − $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$-*std* d K V A p,
{})
⟨*proof*⟩

lemma *anonymity*-$\mathcal{SCF}$-*code-lemma*:
shows $\mathcal{SCF}$-*result*.*anonymity* =
(λ m::(('a, 'v, 'a *Result*) *Electoral-Module*).
$\mathcal{SCF}$-*result*.*electoral-module* m $\wedge$
($\forall$ A V p π::('v $\Rightarrow$ 'v).
bij π $\longrightarrow$ (*let* (A', V', q) = (*rename* π (A, V, p)) *in*
finite-profile V A p $\wedge$ *finite-profile* V' A' q $\longrightarrow$ m V A p = m V' A' q)))
⟨*proof*⟩

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

declare [[*lc-add* $\mathcal{SCF}$-*result*.*electoral-module* *electoral-module*-$\mathcal{SCF}$-*code-lemma*]]
declare [[*lc-add* $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$ $\mathcal{R}_\mathcal{W}$-$\mathcal{SCF}$-*code-lemma*]]
declare [[*lc-add* $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$-*std* $\mathcal{R}_\mathcal{W}$-*std*-$\mathcal{SCF}$-*code-lemma*]]
declare [[*lc-add* $\mathcal{SCF}$-*result*.*distance*-$\mathcal{R}$ *distance*-$\mathcal{R}$-$\mathcal{SCF}$-*code-lemma*]]
declare [[*lc-add* $\mathcal{SCF}$-*result*.*distance*-$\mathcal{R}$-*std* *distance*-$\mathcal{R}$-*std*-$\mathcal{SCF}$-*code-lemma*]]
declare [[*lc-add* $\mathcal{SCF}$-*result*.*anonymity* *anonymity*-$\mathcal{SCF}$-*code-lemma*]]

Constant aliases to use when exporting code instead of the interpreted functions

definition $\mathcal{R}_\mathcal{W}$-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$
definition $\mathcal{R}_\mathcal{W}$-*std*-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.$\mathcal{R}_\mathcal{W}$-*std*
definition *distance*-$\mathcal{R}$-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.*distance*-$\mathcal{R}$
definition *distance*-$\mathcal{R}$-*std*-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.*distance*-$\mathcal{R}$-*std*
definition *electoral-module*-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.*electoral-module*
definition *anonymity*-$\mathcal{SCF}$-*code* = $\mathcal{SCF}$-*result*.*anonymity*

⟨*ML*⟩

end

5.9 Drop Module

theory *Drop-Module*
  imports *Component-Types/Electoral-Module*
    Component-Types/Social-Choice-Types/Result
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according drop module rejects the lexicographically first n alternatives (from A) and defers the rest. It is primarily used as counterpart to the pass module in a parallel composition, in order to segment the alternatives into two groups.

5.9.1 Definition

fun *drop-module* :: $nat \Rightarrow {'a}$ *Preference-Relation*
  $\Rightarrow ({'a}, {'v}, {'a}$ *Result*$)$ *Electoral-Module* **where**
  drop-module $n\ r\ V\ A\ p =$
    $(\{\},$
    $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\},$
    $\{a \in A.\ rank\ (limit\ A\ r)\ a > n\})$

5.9.2 Soundness

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

lemma *voters-determine-drop-mod*:
  fixes
    r :: ${'a}$ *Preference-Relation* **and**
    n :: *nat*
  shows *voters-determine-election* (*drop-module* n r)
  ⟨*proof*⟩

5.9.3 Non-Electing

The drop module is non-electing.

theorem *drop-mod-non-electing*[*simp*]:
  fixes
    $r :: 'a$ *Preference-Relation* **and**
    $n :: nat$
  shows *non-electing* (*drop-module* n r)
  ⟨*proof*⟩

5.9.4 Properties

The drop module is strictly defer-monotone.

theorem *drop-mod-def-lift-inv*[*simp*]:
  fixes
    $r :: 'a$ *Preference-Relation* **and**
    $n :: nat$
  shows *defer-lift-invariance* (*drop-module* n r)
  ⟨*proof*⟩

end

5.10 Pass Module

theory *Pass-Module*
  imports *Component-Types/Electoral-Module*
begin

This is a family of electoral modules. For a natural number n and a lexicon (linear order) r of all alternatives, the according pass module defers the lexicographically first n alternatives (from A) and rejects the rest. It is primarily used as counterpart to the drop module in a parallel composition in order to segment the alternatives into two groups.

5.10.1 Definition

fun *pass-module* :: $nat \Rightarrow 'a$ *Preference-Relation*
    $\Rightarrow ('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
  pass-module $n\ r\ V\ A\ p =$
    $(\{\},$
    $\{a \in A.\ rank\ (limit\ A\ r)\ a > n\},$
    $\{a \in A.\ rank\ (limit\ A\ r)\ a \leq n\})$

5.10.2 Soundness

theorem *pass-mod-sound*[*simp*]:
  fixes

r :: *'a Preference-Relation* **and**
n :: *nat*
shows *$\mathcal{SCF}$-result.electoral-module* (*pass-module n r*)
⟨*proof*⟩

lemma *voters-determine-pass-mod*:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
shows *voters-determine-election* (*pass-module n r*)
⟨*proof*⟩

5.10.3 Non-Blocking

The pass module is non-blocking.

theorem *pass-mod-non-blocking*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes
order: *linear-order r* **and**
g0-n: *n* > *0*
shows *non-blocking* (*pass-module n r*)
⟨*proof*⟩

5.10.4 Non-Electing

The pass module is non-electing.

theorem *pass-mod-non-electing*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *non-electing* (*pass-module n r*)
⟨*proof*⟩

5.10.5 Properties

The pass module is strictly defer-monotone.

theorem *pass-mod-dl-inv*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *defer-lift-invariance* (*pass-module n r*)
⟨*proof*⟩

theorem *pass-zero-mod-def-zero*[*simp*]:
  fixes *r* :: *'a 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*]: $\mathcal{SCF}$*-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' $V\ A\ p$ =
$(\{\},$
$\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\},$
$\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\})$

lemma *enat-leq-enat-set-max*:
fixes
x :: *enat* **and**
X :: *enat set*
assumes
$x \in X$ **and**
finite X
shows $x \leq Max\ X$
⟨*proof*⟩

lemma *plurality-mod-elim-equiv*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
non-empty-A: $A \neq \{\}$ **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*]: $\mathcal{SCF}$*-result.electoral-module plurality*
⟨*proof*⟩

theorem *plurality'-sound*[*simp*]: $\mathcal{SCF}$*-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 :: $'a$ *set* **and**
V :: $'v$ *set* **and**

p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
defer-a: $a \in$ *defer plurality* V A p **and**
lift-a: *lifted* V A p q a
shows *defer plurality* V A q = *defer plurality* V A p
$\vee$ *defer plurality* V A $q = \{a\}$
⟨*proof*⟩

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 = $(\sum\ y \in A.$ (*prefer-count* V p x y))

fun *borda* :: $('a, 'v, 'a$ *Result*) *Electoral-Module* **where**
borda V A p = *max-eliminator borda-score* V A p

5.13.2 Soundness

theorem *borda-sound*: $\mathcal{SCF}$*-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 \in A\ .\ wins\ V\ x\ p\ y\} - $ *card* $\{y \in A\ .\ wins\ V\ y\ p\ x\}$

fun *copeland* :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
copeland $V\ A\ p =$ *max-eliminator copeland-score* $V\ A\ p$

5.15.2 Soundness

theorem *copeland-sound*: $\mathcal{SCF}$*-result.electoral-module copeland*
⟨*proof*⟩

5.15.3 Only Voters Determine Election Result

lemma *voters-determine-copeland-score*: *voters-determine-evaluation copeland-score*
⟨*proof*⟩

theorem *voters-determine-copeland*: *voters-determine-election copeland*
⟨*proof*⟩

5.15.4 Lemmas

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

lemma *cond-winner-imp-win-count*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$
assumes *condorcet-winner* V A p w
shows *card* $\{a \in A.$ *wins* V w p $a\} =$ *card* $A - 1$
⟨*proof*⟩

For a Condorcet winner w, we have: "*card* $\{y \in A$. *wins* y p $x = 0$".

lemma *cond-winner-imp-loss-count*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$
assumes *condorcet-winner* V A p w
shows *card* $\{a \in A.$ *wins* V a p $w\} = 0$
⟨*proof*⟩

Copeland score of a Condorcet winner.

lemma *cond-winner-imp-copeland-score*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$
assumes *condorcet-winner* V A p w
shows *copeland-score* V w A $p =$ *card* $A - 1$
⟨*proof*⟩

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

lemma *non-cond-winner-imp-win-count*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
w :: $'a$ **and**
l :: $'a$
assumes
winner: *condorcet-winner* V A p w **and**
loser: $l \neq w$ **and**
l-in-A: $l \in A$
shows *card* $\{a \in A$. *wins* V l p $a\} \leq$ *card* $A - 2$
⟨*proof*⟩

5.15.5 Property

The Copeland score is Condorcet rating.

theorem *copeland-score-is-cr*: *condorcet-rating copeland-score*
⟨*proof*⟩

theorem *copeland-is-dcc*: *defer-condorcet-consistency copeland*
⟨*proof*⟩

end

5.16 Minimax Module

theory *Minimax-Module*
  imports *Component-Types/Elimination-Module*
begin

This is the Minimax module used by the Minimax voting rule. The Minimax rule elects the alternatives with the highest Minimax score. The module implemented herein only rejects the alternatives not elected by the voting rule, and defers the alternatives that would be elected by the full voting rule.

5.16.1 Definition

fun *minimax-score* :: (′*a*, ′*v*) *Evaluation-Function* **where**
  minimax-score V x A p =
    Min {*prefer-count V p x y* | *y* . *y* ∈ *A* − {*x*}}

fun *minimax* :: (′*a*, ′*v*, ′*a Result*) *Electoral-Module* **where**
  minimax A p = *max-eliminator minimax-score A p*

5.16.2 Soundness

theorem *minimax-sound*: *SCF-result.electoral-module minimax*
  ⟨*proof*⟩

5.16.3 Lemma

lemma *non-cond-winner-minimax-score*:
  fixes
    A :: ′*a set* **and**
    V :: ′*v set* **and**
    p :: (′*a*, ′*v*) *Profile* **and**
    w :: ′*a* **and**

l :: *'a*
assumes
prof: *profile V A p* **and**
winner: *condorcet-winner V A p w* **and**
l-in-A: $l \in A$ **and**
l-neq-w: $l \neq w$
shows *minimax-score V l A p* $\leq$ *prefer-count V p l w*
⟨*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.

6.1.1 Properties

theorem *drop-zero-mod-rej-zero*[*simp*]:
  fixes *r* :: *'a Preference-Relation*
  assumes *linear-order r*
  shows *rejects 0* (*drop-module 0 r*)
⟨*proof*⟩

The drop module rejects n alternatives (if there are at least n alternatives).

theorem *drop-two-mod-rej-n*[*simp*]:
  fixes *r* :: *'a Preference-Relation*
  assumes *linear-order r*
  shows *rejects n* (*drop-module n r*)
⟨*proof*⟩

The pass and drop module are (disjoint-)compatible.

theorem *drop-pass-disj-compat*[*simp*]:
  fixes
    r :: *'a Preference-Relation* **and**
    n :: *nat*
  assumes *linear-order r*
  shows *disjoint-compatibility* (*drop-module n r*) (*pass-module n r*)
⟨*proof*⟩

end

6.2 Revision Composition

theory *Revision-Composition*
  imports *Basic-Modules/Component-Types/Electoral-Module*
begin

A revised electoral module rejects all originally rejected or deferred alternatives, and defers the originally elected alternatives. It does not elect any alternatives.

6.2.1 Definition

fun *revision-composition* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
    ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
  revision-composition m V A p = ({}, *A* − *elect m V A p*, *elect m V A p*)

abbreviation *rev* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
    ⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* (-↓ *50*) **where**
  m↓ == *revision-composition m*

6.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
  fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes *SCF-result.electoral-module m*
  shows *SCF-result.electoral-module* (*revision-composition m*)
⟨*proof*⟩

lemma *voters-determine-rev-comp*:
  fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes *voters-determine-election m*
  shows *voters-determine-election* (*revision-composition m*)
  ⟨*proof*⟩

6.2.3 Composition Rules

An electoral module received by revision is never electing.

theorem *rev-comp-non-electing*[*simp*]:
  fixes *m* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes *SCF-result.electoral-module m*
  shows *non-electing* (*m*↓)
  ⟨*proof*⟩

Revising an electing electoral module results in a non-blocking electoral module.

theorem *rev-comp-non-blocking*[*simp*]:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  assumes *electing m*
  shows *non-blocking* ($m\downarrow$)
⟨*proof*⟩

Revising an invariant monotone electoral module results in a defer-invariant-monotone electoral module.

theorem *rev-comp-def-inv-mono*[*simp*]:
  fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  assumes *invariant-monotonicity m*
  shows *defer-invariant-monotonicity* ($m\downarrow$)
⟨*proof*⟩

end

6.3 Sequential Composition

theory *Sequential-Composition*
  imports *Basic-Modules/Component-Types/Electoral-Module*
begin

The sequential composition creates a new electoral module from two electoral modules. In a sequential composition, the second electoral module makes decisions over alternatives deferred by the first electoral module.

6.3.1 Definition

fun *sequential-composition* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
    $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
    $\Rightarrow$ ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
  sequential-composition m n V A p =
   (*let new-A* = *defer m V A p*;
     new-p = *limit-profile new-A p in* (
        (*elect m V A p*) ∪ (*elect n V new-A new-p*),
        (*reject m V A p*) ∪ (*reject n V new-A new-p*),
        defer n V new-A new-p))

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

(**infix** $\triangleright$ *50*) **where**
$m \triangleright n == \textit{sequential-composition } m\ n$

fun *sequential-composition'* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module*
⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
sequential-composition' *m n V A p* =
(*let* (*m-e*, *m-r*, *m-d*) = *m V A p*; *new-A* = *m-d*;
new-p = *limit-profile new-A p*;
(*n-e*, *n-r*, *n-d*) = *n V new-A new-p in*
(*m-e* ∪ *n-e*, *m-r* ∪ *n-r*, *n-d*))

lemma *voters-determine-seq-comp*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
voters-determine-election m ∧ *voters-determine-election n*
shows *voters-determine-election* ($m \triangleright n$)
⟨*proof*⟩

lemma *seq-comp-presv-disj*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes *module-m*: $\mathcal{SCF}$*-result.electoral-module m* **and**
module-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
prof: *profile V A p*
shows *disjoint3* (($m \triangleright n$) *V A p*)
⟨*proof*⟩

lemma *seq-comp-presv-alts*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile*
assumes *module-m*: $\mathcal{SCF}$*-result.electoral-module m* **and**
module-n: $\mathcal{SCF}$*-result.electoral-module n* **and**
prof: *profile V A p*
shows *set-equals-partition A* (($m \triangleright n$) *V A p*)
⟨*proof*⟩

lemma *seq-comp-alt-eq*[*fundef-cong*, *code*]: *sequential-composition* = *sequential-composition'*
⟨*proof*⟩

6.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
  assumes
    $\mathcal{SCF}$*-result.electoral-module* m **and**
    $\mathcal{SCF}$*-result.electoral-module* n
  shows $\mathcal{SCF}$*-result.electoral-module* $(m \triangleright n)$
⟨*proof*⟩

6.3.3 Lemmas

lemma *seq-comp-decrease-only-defer*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$,$'v$) *Profile*
  assumes
    module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
    module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
    prof: *profile* V A p **and**
    empty-defer: *defer* m V A $p = \{\}$
  shows $(m \triangleright n)$ V A $p = m$ V A p
⟨*proof*⟩

lemma *seq-comp-def-then-elect*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile*
  assumes
    n-electing-m: *non-electing* m **and**
    def-one-m: *defers 1* m **and**
    electing-n: *electing* n **and**
    f-prof: *finite-profile* V A p
  shows *elect* $(m \triangleright n)$ V A p = *defer* m V A p
⟨*proof*⟩

lemma *seq-comp-def-card-bounded*:
  fixes
    m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**

p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
$\mathcal{SCF}$*-result.electoral-module n* **and**
finite-profile V A p
shows *card* (*defer* (*m* ▷ *n*) *V A p*) ≤ *card* (*defer m V A p*)
⟨*proof*⟩

lemma *seq-comp-def-set-bounded*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module m* **and**
$\mathcal{SCF}$*-result.electoral-module n* **and**
profile V A p
shows *defer* (*m* ▷ *n*) *V A p* ⊆ *defer m V A p*
⟨*proof*⟩

lemma *seq-comp-defers-def-set*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
shows *defer* (*m* ▷ *n*) *V A p* =
defer n V (*defer m V A p*) (*limit-profile* (*defer m V A p*) *p*)
⟨*proof*⟩

lemma *seq-comp-def-then-elect-elec-set*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
A :: *'a set* **and**
V :: *'v set* **and**
p :: (*'a*, *'v*) *Profile*
shows *elect* (*m* ▷ *n*) *V A p* =
elect n V (*defer m V A p*)
(*limit-profile* (*defer m V A p*) *p*) ∪ (*elect m V A p*)
⟨*proof*⟩

lemma *seq-comp-elim-one-red-def-set*:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**

A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
eliminates 1 n **and**
profile V A p **and**
card (*defer* m V A p) > 1
shows *defer* $(m \triangleright n)$ V A $p \subset$ *defer* m V A p
⟨*proof*⟩

lemma *seq-comp-def-set-trans*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
$a \in$ (*defer* $(m \triangleright n)$ V A p) **and**
$\mathcal{SCF}$*-result.electoral-module* m $\wedge$ $\mathcal{SCF}$*-result.electoral-module* n **and**
profile V A p
shows $a \in$ *defer* n V (*defer* m V A p) (*limit-profile* (*defer* m V A p) p) $\wedge$
$a \in$ *defer* m V A p
⟨*proof*⟩

6.3.4 Composition Rules

The sequential composition preserves the non-blocking property.

theorem *seq-comp-presv-non-blocking*[*simp*]:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
assumes
non-blocking-m: *non-blocking* m **and**
non-blocking-n: *non-blocking* n
shows *non-blocking* $(m \triangleright n)$
⟨*proof*⟩

Sequential composition preserves the non-electing property.

theorem *seq-comp-presv-non-electing*[*simp*]:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module*
assumes
non-electing m **and**
non-electing n
shows *non-electing* $(m \triangleright n)$

$\langle proof \rangle$

Composing an electoral module that defers exactly 1 alternative in sequence after an electoral module that is electing results (still) in an electing electoral module.

theorem *seq-comp-electing*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    def-one-m: *defers 1 m* **and**
    electing-n: *electing n*
  shows *electing* ($m \triangleright n$)
$\langle proof \rangle$

lemma *def-lift-inv-seq-comp-help*:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    A :: *'a set* **and**
    V :: *'v set* **and**
    p :: (*'a*, *'v*) *Profile* **and**
    q :: (*'a*, *'v*) *Profile* **and**
    a :: *'a*
  assumes
    monotone-m: *defer-lift-invariance m* **and**
    monotone-n: *defer-lift-invariance n* **and**
    voters-determine-n: *voters-determine-election n* **and**
    def-and-lifted: $a \in$ (*defer* ($m \triangleright n$) *V A p*) $\wedge$ *lifted V A p q a*
  shows ($m \triangleright n$) *V A p* = ($m \triangleright n$) *V A q*
$\langle proof \rangle$

Sequential composition preserves the property defer-lift-invariance.

theorem *seq-comp-presv-def-lift-inv*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    defer-lift-invariance m **and**
    defer-lift-invariance n **and**
    voters-determine-election n
  shows *defer-lift-invariance* ($m \triangleright n$)
$\langle proof \rangle$

Composing a non-blocking, non-electing electoral module in sequence with an electoral module that defers exactly one alternative results in an electoral module that defers exactly one alternative.

theorem *seq-comp-def-one*[*simp*]:

fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
non-blocking-m: *non-blocking m* **and**
non-electing-m: *non-electing m* **and**
def-one-n: *defers 1 n*
shows *defers 1* ($m \triangleright n$)
⟨*proof*⟩

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *disj-compat-seq*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
m' :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
compatible: *disjoint-compatibility m n* **and**
module-m': $\mathcal{SCF}$*-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 :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
assumes
dcc-m: *defer-condorcet-consistency m* **and**
nb-n: *non-blocking n* **and**
ne-n: *non-electing n*
shows *condorcet-compatibility* ($m \triangleright n$)
⟨*proof*⟩

Composing a defer-condorcet-consistent electoral module in sequence with a non-blocking and non-electing electoral module results in a defer-condorcet-consistent module.

theorem *seq-comp-dcc*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
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 :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    def-monotone-m: *defer-lift-invariance m* **and**
    non-ele-m: *non-electing m* **and**
    def-one-m: *defers 1 m* **and**
    electing-n: *electing n*
  shows *monotonicity* (*m* ▷ *n*)
⟨*proof*⟩

Composing a defer-invariant-monotone electoral module in sequence before a non-electing, defer-monotone electoral module that defers exactly 1 alternative results in a defer-lift-invariant electoral module.

theorem *def-inv-mono-imp-def-lift-inv*[*simp*]:
  fixes
    m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
    n :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
  assumes
    strong-def-mon-m: *defer-invariant-monotonicity m* **and**
    non-electing-n: *non-electing n* **and**
    defers-one: *defers 1 n* **and**
    defer-monotone-n: *defer-monotonicity n* **and**
    voters-determine-n: *voters-determine-election n*
  shows *defer-lift-invariance* (*m* ▷ *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
⇒ ('a, 'v, 'a Result) Electoral-Module
⇒ 'a Aggregator
⇒ ('a, 'v, 'a Result) Electoral-Module **where**
parallel-composition m n agg V A p = agg A (m V A p) (n V A p)

abbreviation *parallel* :: ('a, 'v, 'a Result) Electoral-Module ⇒ 'a Aggregator
⇒ ('a, 'v, 'a Result) Electoral-Module
⇒ ('a, 'v, 'a Result) Electoral-Module
(- ‖- - [50, 1000, 51] 50) **where**
$m \parallel_a n$ == *parallel-composition m n a*

6.4.2 Soundness

theorem *par-comp-sound*[*simp*]:
fixes
m :: ('a, 'v, 'a Result) Electoral-Module **and**
n :: ('a, 'v, 'a Result) Electoral-Module **and**
a :: 'a Aggregator
assumes
SCF-result.electoral-module m **and**
SCF-result.electoral-module n **and**
aggregator a
shows *SCF-result.electoral-module* $(m \parallel_a n)$
⟨*proof*⟩

6.4.3 Composition Rule

Using a conservative aggregator, the parallel composition preserves the property non-electing.

theorem *conserv-agg-presv-non-electing*[*simp*]:
fixes
m :: ('a, 'v, 'a Result) Electoral-Module **and**
n :: ('a, 'v, 'a Result) Electoral-Module **and**
a :: 'a Aggregator
assumes
non-electing-m: *non-electing m* **and**
non-electing-n: *non-electing n* **and**
conservative: *agg-conservative a*
shows *non-electing* $(m \parallel_a n)$
⟨*proof*⟩

end

6.5 Loop Composition

theory *Loop-Composition*
  imports *Basic-Modules/Component-Types/Termination-Condition*
    Basic-Modules/Defer-Module
    Sequential-Composition
begin

The loop composition uses the same module in sequence, combined with a termination condition, until either

- the termination condition is met or
- no new decisions are made (i.e., a fixed point is reached).

6.5.1 Definition

lemma *loop-termination-helper*:
  fixes
    m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
    t :: $'a$ *Termination-Condition* **and**
    acc :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
    A :: $'a\ set$ **and**
    V :: $'v\ set$ **and**
    p :: $('a, 'v)$ *Profile*
  assumes
    $\neg\ t\ (acc\ V\ A\ p)$ **and**
    $defer\ (acc \triangleright m)\ V\ A\ p \subset defer\ acc\ V\ A\ p$ **and**
    $finite\ (defer\ acc\ V\ A\ p)$
  shows $((acc \triangleright m, m, t, V, A, p), (acc, m, t, V, A, p)) \in$
      $measure\ (\lambda\ (acc, m, t, V, A, p).\ card\ (defer\ acc\ V\ A\ p))$
  ⟨*proof*⟩

This function handles the accumulator for the following loop composition function.

function *loop-comp-helper* ::
  $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module* $\Rightarrow$
    $'a$ *Termination-Condition* $\Rightarrow$ $('a, 'v, 'a\ Result)$ *Electoral-Module* **where**
  $finite\ (defer\ acc\ V\ A\ p) \land (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
    $\longrightarrow t\ (acc\ V\ A\ p) \Longrightarrow$
  $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = acc\ V\ A\ p\ |$
  $\neg\ (finite\ (defer\ acc\ V\ A\ p) \land (defer\ (acc \triangleright m)\ V\ A\ p) \subset (defer\ acc\ V\ A\ p)$
    $\longrightarrow t\ (acc\ V\ A\ p)) \Longrightarrow$
  $loop\text{-}comp\text{-}helper\ acc\ m\ t\ V\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ V\ A\ p$
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *loop-comp-code-helper*[*code*]:
  fixes
    m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
    t :: '*a Termination-Condition* **and**
    acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
    A :: '*a set* **and**
    V :: '*v set* **and**
    p :: ('*a*, '*v*) *Profile*
  shows
    loop-comp-helper acc m t V A p =
      (*if* (*t* (*acc V A p*) ∨ ¬ ((*defer* (*acc* ▷ *m*) *V A p*) ⊂ (*defer acc V A p*))
      ∨ *infinite* (*defer acc V A p*))
      then (*acc V A p*) *else* (*loop-comp-helper* (*acc* ▷ *m*) *m t V A p*))
  ⟨*proof*⟩

function *loop-composition* :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
    ⇒ '*a Termination-Condition*
    ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
  t ({}, {}, *A*)
    ⟹ *loop-composition m t V A p* = *defer-module V A p* |
  ¬(*t* ({}, {}, *A*))
    ⟹ *loop-composition m t V A p* = (*loop-comp-helper m m t*) *V A p*
  ⟨*proof*⟩
termination
  ⟨*proof*⟩

abbreviation *loop* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* ⇒ '*a Termination-Condition*
    ⇒ ('*a*, '*v*, '*a Result*) *Electoral-Module* (- ↺- *50*) **where**
  $m \circlearrowleft_t$ ≡ *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 :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
    t :: '*a Termination-Condition* **and**
    acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
    A :: '*a set* **and**
    V :: '*v set* **and**

p :: ('a, 'v) *Profile* **and**
n :: *nat*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
profile: *profile* V A p **and**
module-acc: $\mathcal{SCF}$*-result.electoral-module* *acc* **and**
defer-card-n: n = *card* (*defer acc* V A p)
shows *well-formed-*$\mathcal{SCF}$ A (*loop-comp-helper acc* m t V A p)
⟨*proof*⟩

6.5.2 Soundness

theorem *loop-comp-sound*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
t :: 'a *Termination-Condition*
assumes $\mathcal{SCF}$*-result.electoral-module* m
shows $\mathcal{SCF}$*-result.electoral-module* (m $\circlearrowleft_t$)
⟨*proof*⟩

lemma *loop-comp-helper-imp-no-def-incr*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
t :: 'a *Termination-Condition* **and**
acc :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile* **and**
n :: *nat*
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
profile: *profile* V A p **and**
mod-acc: $\mathcal{SCF}$*-result.electoral-module* *acc* **and**
card-n-defer-acc: n = *card* (*defer acc* V A p)
shows *defer* (*loop-comp-helper acc* m t) V A p $\subseteq$ *defer acc* V A p
⟨*proof*⟩

6.5.3 Lemmas

lemma *loop-comp-helper-def-lift-inv-helper*:
fixes
m :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
t :: 'a *Termination-Condition* **and**
acc :: ('a, 'v, 'a *Result*) *Electoral-Module* **and**
A :: 'a *set* **and**
V :: 'v *set* **and**
p :: ('a, 'v) *Profile* **and**
n :: *nat*
assumes
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*. $a \in$ (*defer* (*loop-comp-helper acc m t*) *V A p*) $\wedge$ *lifted V A p q a* $\longrightarrow$
(*loop-comp-helper acc m t*) *V A p* = (*loop-comp-helper acc m t*) *V A q*
⟨*proof*⟩

lemma *loop-comp-helper-def-lift-inv*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
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**
$a \in$ *defer* (*loop-comp-helper acc m t*) *V A p*
shows (*loop-comp-helper acc m t*) *V A p* = (*loop-comp-helper acc m t*) *V A q*
⟨*proof*⟩

lemma *lifted-imp-fin-prof*:
fixes
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
q :: ('*a*, '*v*) *Profile* **and**
a :: '*a*
assumes *lifted V A p q a*
shows *finite-profile V A p*
⟨*proof*⟩

lemma *loop-comp-helper-presv-def-lift-inv*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
defer-lift-invariance m **and**
voters-determine-election m **and**

defer-lift-invariance acc
shows *defer-lift-invariance* (*loop-comp-helper acc m t*)
⟨*proof*⟩

lemma *loop-comp-presv-non-electing-helper*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
n :: *nat*
assumes
non-electing-m: *non-electing m* **and**
non-electing-acc: *non-electing acc* **and**
prof: *profile V A p* **and**
acc-defer-card: $n = $ *card* (*defer acc V A p*)
shows *elect* (*loop-comp-helper acc m t*) *V A p* = {}
⟨*proof*⟩

lemma *loop-comp-helper-iter-elim-def-n-helper*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**
acc :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
n :: *nat* **and**
x :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: ∀ *r*. *t r* = (*card* (*defer-r r*) = *x*) **and**
x-greater-zero: $x > 0$ **and**
prof: *profile V A p* **and**
n-acc-defer-card: $n = $ *card* (*defer acc V A p*) **and**
n-ge-x: $n \geq x$ **and**
def-card-gt-one: *card* (*defer acc V A p*) > *1* **and**
acc-nonelect: *non-electing acc*
shows *card* (*defer* (*loop-comp-helper acc m t*) *V A p*) = *x*
⟨*proof*⟩

lemma *loop-comp-helper-iter-elim-def-n*:
fixes
m :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
t :: '*a Termination-Condition* **and**

acc :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a, 'v)$ *Profile* **and**
x :: *nat*
assumes
non-electing m **and**
eliminates 1 m **and**
$\forall\ r.\ (t\ r) = (card\ (defer\text{-}r\ r) = x)$ **and**
$x > 0$ **and**
profile $V\ A\ p$ **and**
$card\ (defer\ acc\ V\ A\ p) \geq x$ **and**
non-electing acc
shows $card\ (defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ V\ A\ p) = x$
$\langle proof \rangle$

lemma *iter-elim-def-n-helper*:
fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
A :: $'a\ set$ **and**
V :: $'v\ set$ **and**
p :: $('a, 'v)$ *Profile* **and**
x :: *nat*
assumes
non-electing-m: *non-electing* m **and**
single-elimination: *eliminates* 1 m **and**
terminate-if-n-left: $\forall\ r.\ (t\ r) = (card\ (defer\text{-}r\ r) = x)$ **and**
x-greater-zero: $x > 0$ **and**
prof: *profile* $V\ A\ p$ **and**
enough-alternatives: $card\ A \geq x$
shows $card\ (defer\ (m\ \circlearrowleft_t)\ V\ A\ p) = x$
$\langle proof \rangle$

6.5.4 Composition Rules

The loop composition preserves defer-lift-invariance.

theorem *loop-comp-presv-def-lift-inv*[*simp*]:
fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition*
assumes *defer-lift-invariance* m **and** *voters-determine-election* m
shows *defer-lift-invariance* $(m\ \circlearrowleft_t)$
$\langle proof \rangle$

The loop composition preserves the property non-electing.

theorem *loop-comp-presv-non-electing*[*simp*]:
fixes
m :: $('a, 'v, 'a\ Result)$ *Electoral-Module* **and**

t :: *'a Termination-Condition*
assumes *non-electing m*
shows *non-electing* (m $\circlearrowleft_t$)
⟨*proof*⟩

theorem *iter-elim-def-n*[*simp*]:
fixes
m :: (*'a*, *'v*, *'a Result*) *Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
n :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall$ *r*. *t r* = (*card* (*defer-r r*) = *n*) **and**
x-greater-zero: $n > 0$
shows *defers n* (m $\circlearrowleft_t$)
⟨*proof*⟩

end

6.6 Maximum Parallel Composition

theory *Maximum-Parallel-Composition*
imports *Basic-Modules/Component-Types/Maximum-Aggregator*
Parallel-Composition
begin

This is a family of parallel compositions. It composes a new electoral module from two electoral modules combined with the maximum aggregator. Therein, the two modules each make a decision and then a partition is returned where every alternative receives the maximum result of the two input partitions. This means that, if any alternative is elected by at least one of the modules, then it gets elected, if any non-elected alternative is deferred by at least one of the modules, then it gets deferred, only alternatives rejected by both modules get rejected.

6.6.1 Definition

fun *maximum-parallel-composition* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module*
⇒ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
maximum-parallel-composition m n =
(*let a* = *max-aggregator in* ($m \parallel_a n$))

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

6.6.2 Soundness

theorem *max-par-comp-sound*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n
shows $\mathcal{SCF}$*-result.electoral-module* ($m \|_\uparrow n$)
⟨*proof*⟩

lemma *voters-determine-max-par-comp*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes
voters-determine-election m **and**
voters-determine-election n
shows *voters-determine-election* ($m \|_\uparrow n$)
⟨*proof*⟩

6.6.3 Lemmas

lemma *max-agg-eq-result*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: ($'a$, $'v$) *Profile* **and**
a :: $'a$
assumes
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
prof-p: *profile* V A p **and**
a-in-A: $a \in A$
shows *mod-contains-result* ($m \|_\uparrow n$) m V A p a $\vee$
mod-contains-result ($m \|_\uparrow n$) n V A p a
⟨*proof*⟩

lemma *max-agg-rej-iff-both-reject*:
fixes
m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
n :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**

A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a,'v)$ *Profile* **and**
a :: $'a$
assumes
finite-profile V A p **and**
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n
shows $(a \in$ *reject* $(m \parallel_\uparrow n)$ V A $p) =$
$(a \in$ *reject* m V A $p \wedge a \in$ *reject* n V A $p)$
⟨*proof*⟩

lemma *max-agg-rej-fst-imp-seq-contained*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
f-prof: *finite-profile* V A p **and**
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
rejected: $a \in$ *reject* n V A p
shows *mod-contains-result* m $(m \parallel_\uparrow n)$ V A p a
⟨*proof*⟩

lemma *max-agg-rej-fst-equiv-seq-contained*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
finite-profile V A p **and**
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n **and**
$a \in$ *reject* n V A p
shows *mod-contains-result-sym* $(m \parallel_\uparrow n)$ m V A p a
⟨*proof*⟩

lemma *max-agg-rej-snd-imp-seq-contained*:
fixes
m :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
n :: $('a, 'v, 'a$ *Result*$)$ *Electoral-Module* **and**
A :: $'a$ *set* **and**

$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ Profile$ **and**
$a :: 'a$
assumes
f-prof: *finite-profile* $V\ A\ p$ **and**
module-m: $\mathcal{SCF}$*-result.electoral-module* m **and**
module-n: $\mathcal{SCF}$*-result.electoral-module* n **and**
rejected: $a \in reject\ m\ V\ A\ p$
shows *mod-contains-result* $n\ (m \parallel_{\uparrow} n)\ V\ A\ p\ a$
$\langle proof \rangle$

lemma *max-agg-rej-snd-equiv-seq-contained*:
fixes
$m :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ Profile$ **and**
$a :: 'a$
assumes
finite-profile $V\ A\ p$ **and**
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n **and**
$a \in reject\ m\ V\ A\ p$
shows *mod-contains-result-sym* $(m \parallel_{\uparrow} n)\ n\ V\ A\ p\ a$
$\langle proof \rangle$

lemma *max-agg-rej-intersect*:
fixes
$m :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$p :: ('a, 'v)\ Profile$
assumes
$\mathcal{SCF}$*-result.electoral-module* m **and**
$\mathcal{SCF}$*-result.electoral-module* n **and**
profile $V\ A\ p$ **and**
finite A
shows $reject\ (m \parallel_{\uparrow} n)\ V\ A\ p = (reject\ m\ V\ A\ p) \cap (reject\ n\ V\ A\ p)$
$\langle proof \rangle$

lemma *dcompat-dec-by-one-mod*:
fixes
$m :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$n :: ('a, 'v, 'a\ Result)\ Electoral\text{-}Module$ **and**
$A :: 'a\ set$ **and**
$V :: 'v\ set$ **and**
$a :: 'a$

assumes
disjoint-compatibility m n **and**
$a \in A$
shows
$(\forall\ p.$ *finite-profile* $V\ A\ p \longrightarrow$ *mod-contains-result* $m\ (m \parallel_\uparrow n)\ V\ A\ p\ a)$
$\vee\ (\forall\ p.$ *finite-profile* $V\ A\ p \longrightarrow$ *mod-contains-result* $n\ (m \parallel_\uparrow n)\ V\ A\ p\ a)$
⟨*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 :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
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 :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
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 :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
A :: '*a set* **and**
V :: '*v set* **and**
p :: ('*a*, '*v*) *Profile* **and**
c :: *nat*
assumes
compatible: *disjoint-compatibility* m n **and**
prof: *profile* $V\ A\ p$ **and**
fin-A: *finite* A **and**
reject-sum: *card* (*reject* $m\ V\ A\ p$) + *card* (*reject* $n\ V\ A\ p$) = *card* $A + c$

shows *card* (*reject* (m $\|_\uparrow$ 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 :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **and**
n :: ('*a*, '*v*, '*a Result*) *Electoral-Module*
assumes
defers-m-one: *defers 1 m* **and**
non-elec-m: *non-electing m* **and**
rejec-n-two: *rejects 2 n* **and**
disj-comp: *disjoint-compatibility m n*
shows *eliminates 1* (m $\|_\uparrow$ 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 :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **and**
b :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
shows $(a \triangleright (elector\ b)) = (elector\ (a \triangleright b))$
⟨*proof*⟩

6.7.3 Soundness

theorem *elector-sound*[*simp*]:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes *SCF-result.electoral-module* m
shows *SCF-result.electoral-module* (*elector* m)
⟨*proof*⟩

lemma *voters-determine-elector*:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes *voters-determine-election* m
shows *voters-determine-election* (*elector* m)
⟨*proof*⟩

6.7.4 Electing

theorem *elector-electing*[*simp*]:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes
module-m: *SCF-result.electoral-module* m **and**
non-block-m: *non-blocking* m
shows *electing* (*elector* m)
⟨*proof*⟩

6.7.5 Composition Rule

If m is defer-Condorcet-consistent, then elector(m) is Condorcet consistent.

lemma *dcc-imp-cc-elector*:
fixes m :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module*
assumes *defer-condorcet-consistency* m
shows *condorcet-consistency* (*elector* m)
⟨*proof*⟩

end

6.8 Defer One Loop Composition

theory *Defer-One-Loop-Composition*
imports *Basic-Modules/Component-Types/Defer-Equal-Condition*

Loop-Composition
Elect-Composition

begin

This is a family of loop compositions. It uses the same module in sequence until either no new decisions are made or only one alternative is remaining in the defer-set. The second family herein uses the above family and subsequently elects the remaining alternative.

6.8.1 Definition

fun *iter* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
$\Rightarrow$ (*'a*, *'v*, *'a Result*) *Electoral-Module* **where**
iter m =
(*let t* = *defer-equal-condition 1 in*
(*m* $\circlearrowleft_t$))

abbreviation *defer-one-loop* :: (*'a*, *'v*, *'a Result*) *Electoral-Module*
$\Rightarrow$ (*'a*, *'v*, *'a Result*) *Electoral-Module* (-$\circlearrowleft_{\exists !d}$ *50*) **where**
m $\circlearrowleft_{\exists !d}$ $\equiv$ *iter m*

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

end

Chapter 7

Voting Rules

7.1 Plurality Rule

theory *Plurality-Rule*
  imports *Compositional-Structures/Basic-Modules/Plurality-Module*
    Compositional-Structures/Revision-Composition
    Compositional-Structures/Elect-Composition
begin

This is a definition of the plurality voting rule as elimination module as well as directly. In the former one, the max operator of the set of the scores of all alternatives is evaluated and is used as the threshold value.

7.1.1 Definition

fun *plurality-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
  plurality-rule V A p = *elector plurality* V A p

fun *plurality-rule'* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
  plurality-rule' V A p =
    $(\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ V\ p\ x \leq \textit{win-count}\ V\ p\ a\},$
    $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ V\ p\ x > \textit{win-count}\ V\ p\ a\},$
    $\{\})$

lemma *plurality-revision-equiv*:
  fixes
    A :: $'a$ *set* **and**
    V :: $'v$ *set* **and**
    p :: ($'a$, $'v$) *Profile*
  shows *plurality'* V A p = (*plurality-rule'*$\downarrow$) V A p
⟨*proof*⟩

lemma *plurality-elim-equiv*:
  fixes
    A :: $'a$ *set* **and**

V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
$A \neq \{\}$ **and**
finite A **and**
profile V A p
shows *plurality* V A p = (*plurality-rule*′↓) V A p
⟨*proof*⟩

7.1.2 Soundness

theorem *plurality-rule-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module plurality-rule*
⟨*proof*⟩

theorem *plurality-rule*′*-sound*[*simp*]: $\mathcal{SCF}$*-result.electoral-module plurality-rule*′
⟨*proof*⟩

lemma *voters-determine-plurality-rule*: *voters-determine-election plurality-rule*
⟨*proof*⟩

7.1.3 Electing

lemma *plurality-rule-elect-non-empty*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile*
assumes
A-non-empty: $A \neq \{\}$ **and**
prof-A: *profile* V A p **and**
fin-A: *finite* A
shows *elect plurality-rule* V A $p \neq \{\}$
⟨*proof*⟩

The plurality module is electing.

theorem *plurality-rule-electing*[*simp*]: *electing plurality-rule*
⟨*proof*⟩

7.1.4 Property

lemma *plurality-rule-inv-mono-eq*:
fixes
A :: $'a$ *set* **and**
V :: $'v$ *set* **and**
p :: $('a, 'v)$ *Profile* **and**
q :: $('a, 'v)$ *Profile* **and**
a :: $'a$
assumes
elect-a: $a \in$ *elect plurality-rule* V A p **and**
lift-a: *lifted* V A p q a

shows *elect plurality-rule* V A q = *elect plurality-rule* V A p
$\vee$ *elect plurality-rule* V A $q = \{a\}$
⟨*proof*⟩

The plurality rule is invariant-monotone.

theorem *plurality-rule-inv-mono*[*simp*]: *invariant-monotonicity plurality-rule*
⟨*proof*⟩

end

7.2 Borda Rule

theory *Borda-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
Compositional-Structures/Basic-Modules/Component-Types/Votewise-Distance-Rationalization
Compositional-Structures/Elect-Composition
begin

This is the Borda rule. On each ballot, each alternative is assigned a score that depends on how many alternatives are ranked below. The sum of all such scores for an alternative is hence called their Borda score. The alternative with the highest Borda score is elected.

7.2.1 Definition

fun *borda-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
borda-rule V A p = *elector borda* V A p

fun *borda-rule*$_{\mathcal{R}}$ :: ($'a$, $'v$::*wellorder*, $'a$ *Result*) *Electoral-Module* **where**
borda-rule$_{\mathcal{R}}$ V A p = *swap-*$\mathcal{R}$ *unanimity* V A p

7.2.2 Soundness

theorem *borda-rule-sound*: $\mathcal{SCF}$*-result.electoral-module borda-rule*
⟨*proof*⟩

theorem *borda-rule*$_{\mathcal{R}}$*-sound*: $\mathcal{SCF}$*-result.electoral-module borda-rule*$_{\mathcal{R}}$
⟨*proof*⟩

7.2.3 Anonymity Property

theorem *borda-rule*$_{\mathcal{R}}$*-anonymous*: $\mathcal{SCF}$*-result.anonymity borda-rule*$_{\mathcal{R}}$
⟨*proof*⟩

end

7.3 Pairwise Majority Rule

theory *Pairwise-Majority-Rule*
imports *Compositional-Structures/Basic-Modules/Condorcet-Module*
Compositional-Structures/Defer-One-Loop-Composition
begin

This is the pairwise majority rule, a voting rule that implements the Condorcet criterion, i.e., it elects the Condorcet winner if it exists, otherwise a tie remains between all alternatives.

7.3.1 Definition

fun *pairwise-majority-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
pairwise-majority-rule V A p = *elector condorcet V A p*

fun *condorcet'* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
condorcet' V A p = ((*min-eliminator condorcet-score*) $\circlearrowleft_{\exists!d}$) *V A p*

fun *pairwise-majority-rule'* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
pairwise-majority-rule' V A p = *iter-elect condorcet' V A p*

7.3.2 Soundness

theorem *pairwise-majority-rule-sound*: $\mathcal{SCF}$*-result.electoral-module pairwise-majority-rule*
⟨*proof*⟩

theorem *condorcet'-rule-sound*: $\mathcal{SCF}$*-result.electoral-module condorcet'*
⟨*proof*⟩

theorem *pairwise-majority-rule'-sound*: $\mathcal{SCF}$*-result.electoral-module pairwise-majority-rule'*
⟨*proof*⟩

7.3.3 Condorcet Consistency Property

theorem *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*
⟨*proof*⟩

end

7.4 Copeland Rule

theory *Copeland-Rule*
imports *Compositional-Structures/Basic-Modules/Copeland-Module*
Compositional-Structures/Elect-Composition
begin

This is the Copeland voting rule. The idea is to elect the alternatives with the highest difference between the amount of simple-majority wins and the amount of simple-majority losses.

7.4.1 Definition

fun *copeland-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
copeland-rule V A p = *elector copeland V A p*

7.4.2 Soundness

theorem *copeland-rule-sound*: $\mathcal{SCF}$*-result.electoral-module copeland-rule*
⟨*proof*⟩

7.4.3 Condorcet Consistency Property

theorem *copeland-condorcet*: *condorcet-consistency copeland-rule*
⟨*proof*⟩

end

7.5 Minimax Rule

theory *Minimax-Rule*
imports *Compositional-Structures/Basic-Modules/Minimax-Module*
Compositional-Structures/Elect-Composition
begin

This is the Minimax voting rule. It elects the alternatives with the highest Minimax score.

7.5.1 Definition

fun *minimax-rule* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
minimax-rule V A p = *elector minimax V A p*

7.5.2 Soundness

theorem *minimax-rule-sound*: $\mathcal{SCF}$*-result.electoral-module minimax-rule*
⟨*proof*⟩

7.5.3 Condorcet Consistency Property

theorem *minimax-condorcet*: *condorcet-consistency minimax-rule*
⟨*proof*⟩

end

7.6 Black's Rule

theory *Blacks-Rule*
imports *Pairwise-Majority-Rule*
Borda-Rule
begin

This is Black's voting rule. It is composed of a function that determines the Condorcet winner, i.e., the Pairwise Majority rule, and the Borda rule. Whenever there exists no Condorcet winner, it elects the choice made by the Borda rule, otherwise the Condorcet winner is elected.

7.6.1 Definition

fun *black* :: ('*a*, '*v*, '*a Result*) *Electoral-Module* **where**
black A p = (*condorcet* ▷ *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*: $\mathcal{SCF}$*-result.electoral-module black*
⟨*proof*⟩

theorem *blacks-rule-sound*: $\mathcal{SCF}$*-result.electoral-module blacks-rule*
⟨*proof*⟩

7.6.3 Condorcet Consistency Property

theorem *black-is-dcc*: *defer-condorcet-consistency black*
⟨*proof*⟩

theorem *black-condorcet*: *condorcet-consistency blacks-rule*
⟨*proof*⟩

end

7.7 Nanson-Baldwin Rule

theory *Nanson-Baldwin-Rule*
  imports *Compositional-Structures/Basic-Modules/Borda-Module*
    Compositional-Structures/Defer-One-Loop-Composition
begin

This is the Nanson-Baldwin voting rule. It excludes alternatives with the lowest Borda score from the set of possible winners and then adjusts the Borda score to the new (remaining) set of still eligible alternatives.

7.7.1 Definition

fun *nanson-baldwin-rule* :: ($'a$, $'v$, $'a$ *Result*) *Electoral-Module* **where**
  nanson-baldwin-rule A p =
    ((*min-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) A p

7.7.2 Soundness

theorem *nanson-baldwin-rule-sound*: $\mathcal{SCF}$*-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*) $\circlearrowleft_{\exists!d}$) V A p

7.8.2 Soundness

theorem *classic-nanson-rule-sound*: $\mathcal{SCF}$*-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*) $\circlearrowleft_{\exists!d}$) *V A p*

7.9.2 Soundness

theorem *schwartz-rule-sound*: $\mathcal{SCF}$*-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*))) $\|_{\uparrow}$
(*drop-module 2 x*)) $\circlearrowleft_{\exists\,!d}$)) *V A p*)

7.10.2 Soundness

As all base components are electoral modules (, aggregators, or termination conditions), and all used compositional structures create electoral modules, sequential majority comparison unsurprisingly is an electoral module.

theorem *smc-sound*:
fixes *x* :: *'a Preference-Relation*
shows $\mathcal{SCF}$*-result.electoral-module* (*smc x*)
⟨*proof*⟩

7.10.3 Electing

The sequential majority comparison electoral module is electing. This property is needed to convert electoral modules to a social choice function. Apart from the very last proof step, it is a part of the monotonicity proof below.

theorem *smc-electing*:
fixes *x* :: *'a Preference-Relation*
assumes *linear-order x*
shows *electing* (*smc x*)
⟨*proof*⟩

7.10.4 (Weak) Monotonicity Property

The following proof is a fully modular proof for weak monotonicity of sequential majority comparison. It is composed of many small steps.

theorem *smc-monotone*:
fixes *x* :: *'a Preference-Relation*
assumes *linear-order x*
shows *monotonicity* (*smc x*)
⟨*proof*⟩

end

7.11 Kemeny Rule

theory *Kemeny-Rule*
imports
Compositional-Structures/Basic-Modules/Component-Types/Votewise-Distance-Rationalization
Compositional-Structures/Basic-Modules/Component-Types/Distance-Rationalization-Symmetry
begin

This is the Kemeny rule. It creates a complete ordering of alternatives and evaluates each ordering of the alternatives in terms of the sum of preference reversals on each ballot that would have to be performed in order to produce that transitive ordering. The complete ordering which requires the fewest preference reversals is the final result of the method.

7.11.1 Definition

fun *kemeny-rule* :: (*'a*, *'v*::*wellorder*, *'a Result*) *Electoral-Module* **where**
kemeny-rule V A p = *swap-*$\mathcal{R}$ *strong-unanimity V A p*

7.11.2 Soundness

theorem *kemeny-rule-sound*: $\mathcal{SCF}$*-result.electoral-module kemeny-rule*
⟨*proof*⟩

7.11.3 Anonymity Property

theorem *kemeny-rule-anonymous*: $\mathcal{SCF}$*-result.anonymity kemeny-rule*
⟨*proof*⟩

7.11.4 Neutrality Property

lemma *swap-dist-neutral*: *distance-neutrality valid-elections*
(*votewise-distance swap l-one*)
⟨*proof*⟩

theorem *kemeny-rule-neutral*: $\mathcal{SCF}$*-properties.neutrality valid-elections kemeny-rule*
⟨*proof*⟩

end

Bibliography

[1] K. Diekhoff, M. Kirsten, and J. Krämer. Formal property-oriented design of voting rules using composable modules. In S. Pekeč and K. Venable, editors, *6th International Conference on Algorithmic Decision Theory (ADT 2019)*, volume 11834 of *Lecture Notes in Artificial Intelligence*, pages 164–166. Springer, 2019.

[2] K. Diekhoff, M. Kirsten, and J. Krämer. Verified construction of fair voting rules. In M. Gabbrielli, editor, *29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019), Revised Selected Papers*, volume 12042 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2020.