Verified Construction of Fair Voting Rules

Michael Kirsten

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

June 15, 2023

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-Relation* $=$ $'a$ *rel*

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

lemma *lin-imp-antisym*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation*
  assumes *linear-order-on A r*
  shows *antisym r*
  using *assms*
  unfolding *linear-order-on-def partial-order-on-def*
  by *simp*

lemma *lin-imp-trans*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation*

assumes *linear-order-on A r*
shows *trans r*
using *assms order-on-defs*
by *blast*

1.1.2 Ranking

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

lemma *rank-gt-zero*:
fixes
r :: *'a Preference-Relation* **and**
x :: *'a*
assumes
refl: $x \preceq_r x$ **and**
fin: *finite r*
shows *rank r x* $\geq$ *1*
proof –
have $x \in \{y \in \mathit{Field}\ r.\ (x, y) \in r\}$
using *FieldI2 refl*
by *fastforce*
hence $\{y \in \mathit{Field}\ r.\ (x, y) \in r\} \neq \{\}$
by *blast*
hence *card* $\{y \in \mathit{Field}\ r.\ (x, y) \in r\} \neq 0$
by (*simp add*: *fin finite-Field*)
moreover have *card* $\{y \in \mathit{Field}\ r.\ (x, y) \in r\} \geq 0$
using *fin*
by *auto*
ultimately show *?thesis*
using *Collect-cong FieldI2 above-def*
less-one not-le-imp-less rank.elims
by (*metis* (*no-types*, *lifting*))
qed

1.1.3 Limited Preference

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

lemma *limitedI*:
fixes
r :: *'a Preference-Relation* **and**
A :: *'a set*
assumes $\bigwedge x\ y.\ x \preceq_r y \Longrightarrow x \in A \land y \in A$
shows *limited A r*
using *assms*
unfolding *limited-def*
by *auto*

lemma *limited-dest*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation* **and**
    x :: $'a$ **and**
    y :: $'a$
  assumes
    $x \preceq_r y$ **and**
    limited A r
  shows $x \preceq_r y \Longrightarrow$ *limited* $A\ r \Longrightarrow x \in A \wedge y \in A$
  unfolding *limited-def*
  by *auto*

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

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

lemma *connex-imp-refl*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation*
  assumes *connex* A r
  shows *refl-on* A r
proof
  from *assms*
  show $r \subseteq A \times A$
    unfolding *connex-def limited-def*
    by *simp*
next
  fix x :: $'a$
  assume $x \in A$
  with *assms*
  have $x \preceq_r x$
    unfolding *connex-def*
    by *metis*
  thus $(x, x) \in r$
    by *simp*
qed

lemma *lin-ord-imp-connex*:
  fixes
    A :: $'a$ *set* **and**
    r :: $'a$ *Preference-Relation*
  assumes *linear-order-on* A r
  shows *connex* A r
proof (*unfold connex-def limited-def*, *safe*)
  fix

a :: $'a$ **and**
b :: $'a$
assume $(a,\ b) \in r$
with *assms*
show $a \in A$
using *partial-order-onD*(*1*) *order-on-defs*(*3*) *refl-on-domain*
by *metis*
next
fix
a :: $'a$ **and**
b :: $'a$
assume $(a,\ b) \in r$
with *assms*
show $b \in A$
using *partial-order-onD*(*1*) *order-on-defs*(*3*) *refl-on-domain*
by *metis*
next
fix
a :: $'a$ **and**
b :: $'a$
assume
a-in-A: $a \in A$ **and**
b-in-A: $b \in A$ **and**
not-y-pref-r-x: $\neg\ b \preceq_r a$
have $(b,\ a) \notin r$
using *not-y-pref-r-x*
by *simp*
with *a-in-A b-in-A*
have $(a,\ b) \in r$
using *assms partial-order-onD*(*1*) *refl-onD*
unfolding *linear-order-on-def total-on-def*
by *metis*
thus $a \preceq_r b$
by *simp*
qed

lemma *connex-antsym-and-trans-imp-lin-ord*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
connex-r: *connex* A r **and**
antisym-r: *antisym* r **and**
trans-r: *trans* r
shows *linear-order-on* A r
proof (*unfold connex-def linear-order-on-def partial-order-on-def*
preorder-on-def refl-on-def total-on-def, *safe*)
fix
a :: $'a$ **and**

b :: $'a$

assume $(a, b) \in r$

thus $a \in A$

using *connex-r refl-on-domain connex-imp-refl*

by *metis*

next

fix

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

b :: $'a$

assume $(a, b) \in r$

thus $b \in A$

using *connex-r refl-on-domain connex-imp-refl*

by *metis*

next

fix a :: $'a$

assume $a \in A$

thus $(a, a) \in r$

using *connex-r connex-imp-refl refl-onD*

by *metis*

next

from *trans-r*

show *trans r*

by *simp*

next

from *antisym-r*

show *antisym r*

by *simp*

next

fix

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

b :: $'a$

assume

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

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

b-not-pref-r-a: $(b, a) \notin r$

from *a-in-A b-in-A*

have $a \preceq_r b \vee b \preceq_r a$

using *connex-r*

unfolding *connex-def*

by *metis*

hence $(a, b) \in r \vee (b, a) \in r$

by *simp*

thus $(a, b) \in r$

using *b-not-pref-r-a*

by *metis*

qed

lemma *limit-to-limits*:

fixes

A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
shows *limited* A (*limit* A r)
unfolding *limited-def*
by *fastforce*

lemma *limit-presv-connex*:
fixes
S :: $'a$ *set* **and**
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation*
assumes
connex: *connex* S r **and**
subset: $A \subseteq S$
shows *connex* A (*limit* A r)
proof (*unfold connex-def limited-def*, *simp*, *safe*)
let $?s = \{(a, b).\ (a, b) \in r \wedge a \in A \wedge b \in A\}$
fix
x :: $'a$ **and**
y :: $'a$ **and**
a :: $'a$ **and**
b :: $'a$
assume
x-in-A: $x \in A$ **and**
y-in-A: $y \in A$ **and**
not-y-pref-r-x: $(y, x) \notin r$
have $y \preceq_r x \vee x \preceq_r y$
using *x-in-A y-in-A connex connex-def in-mono subset*
by *metis*
hence
$x \preceq_{?s} y \vee y \preceq_{?s} x$
using *x-in-A y-in-A*
by *auto*
hence $x \preceq_{?s} y$
using *not-y-pref-r-x*
by *simp*
thus $(x, y) \in r$
by *simp*
qed

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

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

lemma *limit-presv-lin-ord*:
  fixes
    A :: *'a set* **and**
    S :: *'a set* **and**
    r :: *'a Preference-Relation*
  assumes
    linear-order-on S r **and**
      $A \subseteq S$
    shows *linear-order-on A* (*limit A r*)
  using *assms connex-antsym-and-trans-imp-lin-ord limit-presv-antisym limit-presv-connex limit-presv-trans lin-ord-imp-connex order-on-defs*(*1*, *2*, *3*)
  by *metis*

lemma *limit-presv-prefs-1*:
  fixes
    A :: *'a set* **and**
    r :: *'a Preference-Relation* **and**
    x :: *'a* **and**
    y :: *'a*
  assumes
    x-less-y: $x \preceq_r y$ **and**
    x-in-A: $x \in A$ **and**
    y-in-A: $y \in A$
  shows *let* s = *limit A r in* $x \preceq_s y$
  using *x-in-A x-less-y y-in-A*
  by *simp*

lemma *limit-presv-prefs-2*:
  fixes
    A :: *'a set* **and**
    r :: *'a Preference-Relation* **and**
    x :: *'a* **and**
    y :: *'a*
  assumes $(x, y) \in$ *limit A r*
  shows $x \preceq_r y$
  using *mem-Collect-eq assms*
  by *simp*

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

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

lemma *lin-ord-singleton*:
  fixes a :: $'a$
  shows $\forall$ r. *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof (*clarify*)
  fix r :: $'a$ *Preference-Relation*
  assume *lin-ord-r-a*: *linear-order-on* $\{a\}$ r
  hence $a \preceq_r a$
    using *lin-ord-imp-connex singletonI*
    unfolding *connex-def*
    by *metis*
  moreover from *lin-ord-r-a*
  have $\forall$ $(x, y) \in r$. $x = a \land y = a$
    using *connex-imp-refl lin-ord-imp-connex*
        refl-on-domain split-beta
    by *fastforce*
  ultimately show $r = \{(a, a)\}$
    by *auto*
qed

1.1.4 Auxiliary Lemmas

lemma *above-trans*:
  fixes
    r :: $'a$ *Preference-Relation* **and**
    a :: $'a$ **and**
    b :: $'a$
  assumes
    trans r **and**
    $(a, b) \in r$
  shows *above* r $b \subseteq$ *above* r a
  using *Collect-mono assms transE*

unfolding *above-def*
by *metis*

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
using *assms refl-onD*
unfolding *above-def*
by *simp*

lemma *above-subset-geq-one*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
s :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
linear-order-on A r $\wedge$ *linear-order-on* A s **and**
above r $a \subseteq$ *above* s a **and**
above s $a = \{a\}$
shows *above* r $a = \{a\}$
using *assms connex-imp-refl above-refl insert-absorb lin-ord-imp-connex mem-Collect-eq refl-on-domain singletonI subset-singletonD*
unfolding *above-def*
by *metis*

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

lemma *pref-imp-in-above*:
fixes
r :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**

$b :: 'a$
shows $a \preceq_r b \equiv b \in above\ r\ a$
unfolding *above-def*
by *simp*

lemma *limit-presv-above*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$ **and**
$b :: 'a$
assumes
$b \in above\ r\ a$ **and**
$a \in A$ **and**
$b \in A$
shows $b \in above\ (limit\ A\ r)\ a$
using *assms pref-imp-in-above limit-presv-prefs-1*
by *metis*

lemma *limit-presv-above-2*:
fixes
$A :: 'a\ set$ **and**
$B :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$ **and**
$a :: 'a$ **and**
$b :: 'a$
assumes $b \in above\ (limit\ B\ r)\ a$
shows $b \in above\ r\ a$
using *assms limit-presv-prefs-2*
mem-Collect-eq pref-imp-in-above
unfolding *above-def*
by *metis*

lemma *above-one*:
fixes
$A :: 'a\ set$ **and**
$r :: 'a\ Preference\text{-}Relation$
assumes
lin-ord-r: $linear\text{-}order\text{-}on\ A\ r$ **and**
fin-ne-A: $finite\ A \wedge A \neq \{\}$
shows $\exists\ a \in A.\ above\ r\ a = \{a\} \wedge (\forall\ x \in A.\ above\ r\ x = \{x\} \longrightarrow x = a)$
proof −
obtain $n :: nat$ **where**
len-n-plus-one: $n + 1 = card\ A$
using *Suc-eq-plus1 antisym-conv2 fin-ne-A card-eq-0-iff*
gr0-implies-Suc le0
by *metis*
have
$(linear\text{-}order\text{-}on\ A\ r \wedge finite\ A \wedge A \neq \{\} \wedge n + 1 = card\ A)$

$\longrightarrow (\exists\ a.\ a \in A \wedge above\ r\ a = \{a\})$
proof (*induction n arbitrary*: *A r*)
  case *0*
  show *?case*
  proof (*clarify*)
    assume
      lin-ord-r: *linear-order-on A r* **and**
      len-A-is-one: *0 + 1 = card A*
    then obtain *a* **where**
      $\{a\} = A$
      using *card-1-singletonE add.left-neutral*
      by *metis*
    hence $a \in A \wedge above\ r\ a = \{a\}$
      using *above-def lin-ord-r connex-imp-refl above-refl*
        lin-ord-imp-connex refl-on-domain
      by *fastforce*
    thus $\exists\ a.\ a \in A \wedge above\ r\ a = \{a\}$
      by *metis*
  qed
next
  case (*Suc n*)
  show *?case*
  proof (*clarify*)
    assume
      lin-ord-r: *linear-order-on A r* **and**
      fin-A: *finite A* **and**
      A-not-empty: $A \neq \{\}$ **and**
      len-A-n-plus-one: *Suc n + 1 = card A*
    then obtain *B* **where**
      subset-B-card: $card\ B = n + 1 \wedge B \subseteq A$
      using *Suc-inject add-Suc card.insert-remove finite.cases*
        insert-Diff-single subset-insertI
      by (*metis* (*mono-tags*, *lifting*))
    then obtain *a* **where**
      a: $\{a\} = A - B$
      using *Suc-eq-plus1 add-diff-cancel-left′ fin-A len-A-n-plus-one*
        card-1-singletonE card-Diff-subset finite-subset
      by *metis*
    have $\exists\ b \in B.\ above\ (limit\ B\ r)\ b = \{b\}$
      using *subset-B-card Suc.IH add-diff-cancel-left′ lin-ord-r card-eq-0-iff*
        diff-le-self leD lessI limit-presv-lin-ord
      unfolding *One-nat-def*
      by *metis*
    then obtain *b* **where**
      alt-b: $above\ (limit\ B\ r)\ b = \{b\}$
      by *blast*
    hence *b-above*: $\{a.\ (b, a) \in limit\ B\ r\} = \{b\}$
      unfolding *above-def*
      by *metis*

hence *b-pref-b*: $b \preceq_r b$
  using *CollectD limit-presv-prefs-2 singletonI*
  by (*metis* (*lifting*))
show $\exists\ a.\ a \in A \wedge above\ r\ a = \{a\}$
proof (*cases*)
  assume *a-pref-r-b*: $a \preceq_r b$
  have *refl-A*:
    $\forall\ A\ r\ a\ a'.$
      $(refl\text{-}on\ A\ r \wedge (a{::}'a,\ a') \in r) \longrightarrow a \in A \wedge a' \in A$
    using *refl-on-domain*
    by *metis*
  have *connex-refl*:
    $\forall\ A\ r.\ connex\ (A{::}'a\ set)\ r \longrightarrow refl\text{-}on\ A\ r$
    using *connex-imp-refl*
    by *metis*
  have $\forall\ A\ r.\ linear\text{-}order\text{-}on\ (A{::}'a\ set)\ r \longrightarrow connex\ A\ r$
    by (*simp add*: *lin-ord-imp-connex*)
  hence $refl\text{-}on\ A\ r$
    using *connex-refl lin-ord-r*
    by *metis*
  hence $a \in A \wedge b \in A$
    using *refl-A a-pref-r-b*
    by *simp*
  hence *b-in-r*:
    $\forall\ a.\ a \in A \longrightarrow (b = a \vee (b,\ a) \in r \vee (a,\ b) \in r)$
    using *lin-ord-r order-on-defs*(*3*)
    unfolding *total-on-def*
    by *metis*
  have *b-in-lim-B-r*: $(b,\ b) \in limit\ B\ r$
    using *alt-b mem-Collect-eq singletonI*
    unfolding *above-def*
    by *metis*
  have *b-wins*:
    $\{a.\ (b,\ a) \in limit\ B\ r\} = \{b\}$
    using *alt-b*
    unfolding *above-def*
    by (*metis* (*no-types*))
  have *b-refl*: $(b,\ b) \in \{(a',\ a).\ (a',\ a) \in r \wedge a' \in B \wedge a \in B\}$
    using *b-in-lim-B-r*
    by *simp*
  moreover have *b-wins-B*:
    $\forall\ x \in B.\ b \in above\ r\ x$
    using *subset-B-card b-in-r b-wins b-refl CollectI*
        Product-Type.Collect-case-prodD
    unfolding *above-def*
    by *fastforce*
  moreover have $b \in above\ r\ a$
    using *a-pref-r-b pref-imp-in-above*
    by *metis*

ultimately have *b-wins*: $\forall\ x \in A.\ b \in above\ r\ x$
  using *Diff-iff a empty-iff insert-iff*
  by (*metis* (*no-types*))
hence $\forall\ x \in A.\ x \in above\ r\ b \longrightarrow x = b$
  using *CollectD lin-ord-r lin-imp-antisym*
  unfolding *above-def antisym-def*
  by *metis*
hence $\forall\ x \in A.\ (x \in above\ r\ b) = (x = b)$
  using *b-wins*
  by *blast*
moreover have *above-b-in-A*: $above\ r\ b \subseteq A$
  using *lin-ord-r connex-imp-refl lin-ord-imp-connex*
    mem-Collect-eq refl-on-domain subsetI
  unfolding *above-def*
  by *metis*
ultimately have $above\ r\ b = \{b\}$
  using *alt-b*
  unfolding *above-def*
  by *fastforce*
thus *?thesis*
  using *above-b-in-A*
  by *blast*
next
  assume $\neg\ a \preceq_r b$
  hence *b-smaller-a*: $b \preceq_r a$
    using *subset-B-card DiffE a lin-ord-r alt-b limit-to-limits*
      limited-dest singletonI subset-iff
      lin-ord-imp-connex pref-imp-in-above
    unfolding *connex-def*
    by *metis*
  hence *b-smaller-a-0*: $(b,\ a) \in r$
    by *simp*
  have *lin-ord-subset-A*:
    $\forall\ A\ r\ A'.$
      $(linear\text{-}order\text{-}on\ (A::'a\ set)\ r \wedge A' \subseteq A) \longrightarrow$
        $linear\text{-}order\text{-}on\ A'\ (limit\ A'\ r)$
    using *limit-presv-lin-ord*
    by *metis*
  have
    $\{a.\ (b,\ a) \in limit\ B\ r\} = \{b\}$
    using *alt-b*
    unfolding *above-def*
    by *metis*
  hence *b-in-B*: $b \in B$
    by *auto*
  have *limit-B*:
    $partial\text{-}order\text{-}on\ B\ (limit\ B\ r) \wedge total\text{-}on\ B\ (limit\ B\ r)$
    using *lin-ord-subset-A subset-B-card lin-ord-r*
    unfolding *order-on-defs*(*3*)

by *metis*
have
$\forall\ A\ r.$
$\textit{total-on}\ A\ r = (\forall\ a.\ (a{::}'a) \notin A \lor$
$(\forall\ a'.\ (a' \notin A \lor a = a') \lor (a,\ a') \in r \lor (a',\ a) \in r))$
unfolding *total-on-def*
by *metis*
hence
$\forall\ a.\ a \notin B \lor$
$(\forall\ a'.\ a' \in B \longrightarrow$
$(a = a' \lor (a,\ a') \in \textit{limit}\ B\ r \lor (a',\ a) \in \textit{limit}\ B\ r))$
using *limit-B*
by *simp*
hence $\forall\ x \in B.\ b \in \textit{above}\ r\ x$
using *limit-presv-prefs-2 pref-imp-in-above singletonD mem-Collect-eq lin-ord-r alt-b b-above b-pref-b subset-B-card b-in-B*
by (*metis* (*lifting*))
hence $\forall\ x \in B.\ x \preceq_r b$
unfolding *above-def*
by *simp*
hence *b-wins-2*: $\forall\ x \in B.\ (x,\ b) \in r$
by *simp*
have *trans r*
using *lin-ord-r lin-imp-trans*
by *metis*
hence $\forall\ x \in B.\ (x,\ a) \in r$
using *transE b-smaller-a-0 b-wins-2*
by *metis*
hence $\forall\ x \in B.\ x \preceq_r a$
by *simp*
hence *nothing-above-a*: $\forall\ x \in A.\ x \preceq_r a$
using *a lin-ord-r lin-ord-imp-connex above-connex Diff-iff empty-iff insert-iff pref-imp-in-above*
by *metis*
have $\forall\ x \in A.(\ x \in \textit{above}\ r\ a) = (x = a)$
using *lin-ord-r lin-imp-antisym nothing-above-a pref-imp-in-above CollectD*
unfolding *antisym-def above-def*
by *metis*
moreover have *above-a-in-A*: $\textit{above}\ r\ a \subseteq A$
using *lin-ord-r connex-imp-refl lin-ord-imp-connex mem-Collect-eq refl-on-domain*
unfolding *above-def*
by *fastforce*
ultimately have $\textit{above}\ r\ a = \{a\}$
using *a*
unfolding *above-def*
by *blast*
thus *?thesis*
using *above-a-in-A*

          by *blast*
        qed
      qed
    qed
    hence $\exists\ a.\ a \in A \wedge above\ r\ a = \{a\}$
      using *fin-ne-A lin-ord-r len-n-plus-one*
      by *blast*
    thus *?thesis*
      using *assms lin-ord-imp-connex pref-imp-in-above singletonD*
      unfolding *connex-def*
      by *metis*
qed

lemma *above-one-2*:
  fixes
    $A :: 'a\ set$ **and**
    $r :: 'a\ Preference\text{-}Relation$ **and**
    $a :: 'a$ **and**
    $b :: 'a$
  assumes
    lin-ord: $linear\text{-}order\text{-}on\ A\ r$ **and**
    fin-not-emp: $finite\ A \wedge A \neq \{\}$ **and**
    above-a: $above\ r\ a = \{a\}$ **and**
    above-b: $above\ r\ b = \{b\}$
  shows $a = b$
proof $-$
  have $a \preceq_r a$
    using *above-a singletonI pref-imp-in-above*
    by *metis*
  also have $b \preceq_r b$
    using *above-b singletonI pref-imp-in-above*
    by *metis*
  moreover have
    $\exists\ a \in A.\ above\ r\ a = \{a\} \wedge$
      $(\forall\ x \in A.\ above\ r\ x = \{x\} \longrightarrow x = a)$
    using *lin-ord fin-not-emp*
    by (*simp add*: *above-one*)
  moreover have $connex\ A\ r$
    using *lin-ord*
    by (*simp add*: *lin-ord-imp-connex*)
  ultimately show $a = b$
    using *above-a above-b limited-dest*
    unfolding *connex-def*
    by *metis*
qed

lemma *rank-one-1*:
  fixes
    $r :: 'a\ Preference\text{-}Relation$ **and**

```
    a :: 'a
  assumes above r a = {a}
  shows rank r a = 1
  using assms
  by simp

lemma rank-one-2:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation and
    a :: 'a
  assumes
    lin-ord: linear-order-on A r and
    rank-one: rank r a = 1
  shows above r a = {a}
proof −
  from lin-ord
  have refl: refl-on A r
    using linear-order-on-def partial-order-onD(1)
    by blast
  from lin-ord rank-one
  have a ∈ A
    unfolding rank.simps above-def linear-order-on-def
    partial-order-on-def preorder-on-def total-on-def
    using card-1-singletonE insertI1 mem-Collect-eq refl-onD1
    by metis
  with refl
  have a ∈ above r a
    using above-refl
    by fastforce
  with rank-one
  show above r a = {a}
    using card-1-singletonE rank.simps singletonD
    by metis
qed

theorem above-rank:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation and
    a :: 'a
  assumes lin-ord: linear-order-on A r
  shows (above r a = {a}) = (rank r a = 1)
  using lin-ord rank-one-1 rank-one-2
  by metis

lemma above-presv-limit:
  fixes
    A :: 'a set and
```

r :: $'a$ *Preference-Relation* **and**
x :: $'a$
shows *above* (*limit* A r) $x \subseteq A$
unfolding *above-def*
by *auto*

1.1.5 Lifting Property

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

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

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

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

lemma *lifted-mono*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
s :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes *lifted* A r s a
shows $\forall\ x \in A - \{a\}.\ \neg(x \preceq_r a \wedge a \preceq_s x)$

proof (*safe*)
fix $x :: 'a$
assume
x-in-A: $x \in A$ **and**
x-exist: $x \notin \{\}$ **and**
x-neq-a: $x \neq a$ **and**
x-pref-a: $x \preceq_r a$ **and**
a-pref-x: $a \preceq_s x$
from *x-pref-a*
have *x-pref-a-0*: $(x, a) \in r$
by *simp*
from *a-pref-x*
have *a-pref-x-0*: $(a, x) \in s$
by *simp*
from *assms*
have *antisym r*
using *lifted-imp-equiv-rel-except-a lin-imp-antisym*
unfolding *equiv-rel-except-a-def*
by *metis*
hence *antisym-r*:
$(\forall\ x\ y.\ (x, y) \in r \longrightarrow (y, x) \in r \longrightarrow x = y)$
unfolding *antisym-def*
by *metis*
hence *imp-x-eq-a*:
$[\![(x, a) \in r; (a, x) \in r]\!] \Longrightarrow x = a$
by *simp*
from *assms*
have *lift-ex*: $\exists\ x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
unfolding *lifted-def*
by *metis*
from *lift-ex*
obtain $y :: 'a$ **where**
$y \in A - \{a\} \wedge a \preceq_r y \wedge y \preceq_s a$
by *metis*
hence *y-eq-r-s-exc-a*:
$y \in A - \{a\} \wedge (a, y) \in r \wedge (y, a) \in s$
by *simp*
from *assms*
have *equiv-r-s-exc-a*: *equiv-rel-except-a* $A\ r\ s\ a$
unfolding *lifted-def*
by *metis*
hence $\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.\ (x \preceq_r y) = (x \preceq_s y)$
unfolding *equiv-rel-except-a-def*
by *metis*
hence *equiv-r-s-exc-a-0*:
$\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.\ ((x, y) \in r) = ((x, y) \in s)$
by *simp*
from *equiv-r-s-exc-a*
have *trans*: $\forall\ x\ y\ z.\ (x, y) \in r \longrightarrow (y, z) \in r \longrightarrow (x, z) \in r$

unfolding *equiv-rel-except-a-def linear-order-on-def*
partial-order-on-def preorder-on-def trans-def
by *metis*
from *x-in-A x-neq-a x-pref-a-0 y-eq-r-s-exc-a equiv-r-s-exc-a equiv-r-s-exc-a-0*
have *x-pref-y-0*: $(x, y) \in s$
using *insertE insert-Diff trans*
unfolding *equiv-rel-except-a-def*
by *metis*
from *a-pref-x-0 x-pref-y-0 x-pref-a-0 imp-x-eq-a x-neq-a equiv-r-s-exc-a*
have $(a, y) \in s$
using *lin-imp-trans transE*
unfolding *equiv-rel-except-a-def*
by *metis*
with *y-eq-r-s-exc-a equiv-r-s-exc-a*
show *False*
using *antisymD DiffD2 lin-imp-antisym singletonI*
unfolding *equiv-rel-except-a-def*
by *metis*
qed

lemma *lifted-mono2*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
s :: *'a Preference-Relation* **and**
a :: *'a*
assumes
lifted: *lifted* $A\ r\ s\ a$ **and**
x-pref-a: $x \preceq_r a$
shows $x \preceq_s a$
proof (*simp*)
from *x-pref-a*
have *x-pref-a-0*: $(x, a) \in r$
by *simp*
with *lifted*
have *x-in-A*: $x \in A$
using *connex-imp-refl lin-ord-imp-connex refl-on-domain*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
have $\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.\ (x \preceq_r y) = (x \preceq_s y)$
using *lifted*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.\ ((x, y) \in r) = ((x, y) \in s)$
by *simp*
have $\exists\ x \in A - \{a\}.\ a \preceq_r x \land x \preceq_s a$
using *lifted*
unfolding *lifted-def*

by *metis*
hence *ex-lifted*: $\exists\ x \in A - \{a\}.\ (a,\ x) \in r \wedge (x,\ a) \in s$
by *simp*
show $(x,\ a) \in s$
proof (*cases* $x = a$)
case *True*
thus *?thesis*
using *connex-imp-refl refl-onD lifted lin-ord-imp-connex*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
next
case *False*
with *x-pref-a-0 x-in-A rest-eq ex-lifted*
show *?thesis*
using *insertE insert-Diff lifted lin-imp-trans*
lifted-imp-equiv-rel-except-a
unfolding *equiv-rel-except-a-def trans-def*
by *metis*
qed
qed

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

by *metis*
from *assms*
have *trans-s*: $\forall\ x\ y\ z.\ (x,\ y) \in s \longrightarrow (y,\ z) \in s \longrightarrow (x,\ z) \in s$
using *lin-imp-trans*
unfolding *trans-def lifted-def equiv-rel-except-a-def*
by *metis*
from *assms*
have *refl-r*: $(a,\ a) \in r$
using *connex-imp-refl lin-ord-imp-connex refl-onD*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
from *a-pref-x assms*
have $x \in A$
using *connex-imp-refl lin-ord-imp-connex refl-onD2*
unfolding *equiv-rel-except-a-def lifted-def*
by *metis*
with *a-pref-x lifted-r rest-eq trans-r trans-s refl-r*
show $(a,\ x) \in r$
using *Diff-iff singletonD*
by (*metis* (*full-types*))
qed

lemma *lifted-above-2*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
s :: $'a$ *Preference-Relation* **and**
a :: $'a$ **and**
x :: $'a$
assumes
lifted-a: *lifted* $A\ r\ s\ a$ **and**
x-in-A-sub-a: $x \in A - \{a\}$
shows *above* $r\ x \subseteq$ *above* $s\ x \cup \{a\}$
proof (*safe*, *simp*)
fix y :: $'a$
assume
y-in-above-r: $y \in$ *above* $r\ x$ **and**
y-not-in-above-s: $y \notin$ *above* $s\ x$
have $\forall\ z \in A - \{a\}.\ (x \preceq_r z) = (x \preceq_s z)$
using *x-in-A-sub-a lifted-a*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence $\forall\ z \in A - \{a\}.\ ((x,\ z) \in r) = ((x,\ z) \in s)$
by *simp*
hence $\forall\ z \in A - \{a\}.\ (z \in$ *above* $r\ x) = (z \in$ *above* $s\ x)$
unfolding *above-def*
by *simp*
hence $(y \in$ *above* $r\ x) = (y \in$ *above* $s\ x)$
using *lifted-a y-not-in-above-s lifted-mono2 limited-dest lifted-def*

lin-ord-imp-connex member-remove pref-imp-in-above
unfolding *equiv-rel-except-a-def remove-def connex-def*
by *metis*
thus $y = a$
using *y-in-above-r y-not-in-above-s*
by *simp*
qed

lemma *limit-lifted-imp-eq-or-lifted*:
fixes
A :: *'a set* **and**
S :: *'a set* **and**
r :: *'a Preference-Relation* **and**
s :: *'a Preference-Relation* **and**
a :: *'a*
assumes
lifted: *lifted* S r s a **and**
subset: $A \subseteq S$
shows *limit* A r = *limit* A s $\vee$ *lifted* A (*limit* A r) (*limit* A s) a
proof −
from *lifted*
have $\forall\ x \in S - \{a\}.\ \forall\ y \in S - \{a\}.\ (x \preceq_r y) = (x \preceq_s y)$
unfolding *lifted-def equiv-rel-except-a-def*
by *simp*
with *subset*
have *temp*: $\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.\ (x \preceq_r y) = (x \preceq_s y)$
by *auto*
hence *eql-rs*:
$\forall\ x \in A - \{a\}.\ \forall\ y \in A - \{a\}.$
$((x, y) \in (\mathit{limit}\ A\ r)) = ((x, y) \in (\mathit{limit}\ A\ s))$
using *DiffD1 limit-presv-prefs-1 limit-presv-prefs-2*
by *simp*
from *lifted subset*
have *lin-ord-r-s*: *linear-order-on* A (*limit* A r) $\wedge$ *linear-order-on* A (*limit* A s)
using *lifted-def equiv-rel-except-a-def limit-presv-lin-ord*
by *metis*
show *?thesis*
proof (*cases*)
assume *a-in-A*: $a \in A$
thus *?thesis*
proof (*cases*)
assume $\exists\ x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
with *a-in-A*
have *keep-lift*:
$\exists\ x \in A - \{a\}.\ (\mathit{let}\ q = \mathit{limit}\ A\ r\ \mathit{in}\ a \preceq_q x) \wedge$
$(\mathit{let}\ u = \mathit{limit}\ A\ s\ \mathit{in}\ x \preceq_u a)$
using *DiffD1 limit-presv-prefs-1*
by *simp*
thus *?thesis*

using *a-in-A temp lin-ord-r-s*
unfolding *lifted-def equiv-rel-except-a-def*
by *simp*
next
assume $\neg(\exists\ x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a)$
hence *strict-pref-to-a*:
$\forall\ x \in A - \{a\}.\ \neg(a \preceq_r x \wedge x \preceq_s a)$
by *simp*
moreover have *not-worse*:
$\forall\ x \in A - \{a\}.\ \neg(x \preceq_r a \wedge a \preceq_s x)$
using *lifted subset lifted-mono*
by *fastforce*
moreover have *connex*:
connex A (*limit A r*) $\wedge$ *connex A* (*limit A s*)
using *lifted subset limit-presv-lin-ord lin-ord-imp-connex*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
moreover have *connex-1*:
$\forall\ A\ r.\ connex\ A\ r =$
$(limited\ A\ r \wedge (\forall\ a.\ (a{::}'a) \in A \longrightarrow$
$(\forall\ a'.\ a' \in A \longrightarrow a \preceq_r a' \vee a' \preceq_r a)))$
unfolding *connex-def*
by (*simp add*: *Ball-def-raw*)
hence *limit-1*:
limited A (*limit A r*) $\wedge$
$(\forall\ a.\ a \notin A \vee$
$(\forall\ a'.$
$a' \notin A \vee (a,\ a') \in limit\ A\ r \vee$
$(a',\ a) \in limit\ A\ r\ ))$
using *connex connex-1*
by *simp*
have *limit-2*: $\forall\ a\ a'\ A\ r.\ (a{::}'a,\ a') \notin limit\ A\ r \vee a \preceq_r a'$
using *limit-presv-prefs-2*
by *metis*
have
limited A (*limit A s*) $\wedge$
$(\forall\ a.\ a \notin A \vee$
$(\forall\ a'.\ a' \notin A \vee$
$(let\ q = limit\ A\ s\ in\ a \preceq_q a' \vee a' \preceq_q a)))$
using *connex*
unfolding *connex-def*
by *metis*
hence *connex-2*:
limited A (*limit A s*) $\wedge$
$(\forall\ a.\ a \notin A \vee$
$(\forall\ a'.\ a' \notin A \vee$
$((a,\ a') \in limit\ A\ s \vee (a',\ a) \in limit\ A\ s)))$
by *simp*
ultimately have $\forall\ x \in A - \{a\}.\ (a \preceq_r x \wedge a \preceq_s x) \vee (x \preceq_r a \wedge x \preceq_s a)$

using *DiffD1 limit-1 limit-presv-prefs-2 a-in-A*
by *metis*
hence *r-eq-s-on-A-0*:
$\forall\ x \in A - \{a\}.\ ((a,\ x) \in r \wedge (a,\ x) \in s) \vee ((x,\ a) \in r \wedge (x,\ a) \in s)$
by *simp*
have $\forall\ x \in A - \{a\}.\ ((a,\ x) \in (\mathit{limit}\ A\ r)) = ((a,\ x) \in (\mathit{limit}\ A\ s))$
using *DiffD1 limit-2 limit-1 connex-2 a-in-A strict-pref-to-a not-worse*
by *metis*
hence
$\forall\ x \in A - \{a\}.$
$(\mathit{let}\ q = \mathit{limit}\ A\ r\ \mathit{in}\ a \preceq_q x) = (\mathit{let}\ q = \mathit{limit}\ A\ s\ \mathit{in}\ a \preceq_q x)$
by *simp*
moreover have
$\forall\ x \in A - \{a\}.\ ((x,\ a) \in (\mathit{limit}\ A\ r)) = ((x,\ a) \in (\mathit{limit}\ A\ s))$
using *a-in-A strict-pref-to-a not-worse DiffD1 limit-presv-prefs-2 connex-2 limit-1*
by *metis*
moreover have
$(a,\ a) \in (\mathit{limit}\ A\ r) \wedge (a,\ a) \in (\mathit{limit}\ A\ s)$
using *a-in-A connex connex-imp-refl refl-onD*
by *metis*
moreover have
$\mathit{limited}\ A\ (\mathit{limit}\ A\ r) \wedge \mathit{limited}\ A\ (\mathit{limit}\ A\ s)$
using *limit-to-limits*
by *metis*
ultimately have
$\forall\ x\ y.\ ((x,\ y) \in \mathit{limit}\ A\ r) = ((x,\ y) \in \mathit{limit}\ A\ s)$
using *eql-rs*
by *auto*
thus *?thesis*
by *simp*
qed
next
assume $a \notin A$
with *eql-rs*
have $\forall\ x \in A.\ \forall\ y \in A.\ ((x,\ y) \in (\mathit{limit}\ A\ r)) = ((x,\ y) \in (\mathit{limit}\ A\ s))$
by *simp*
thus *?thesis*
using *limit-to-limits limited-dest subrelI subset-antisym*
by *auto*
qed
qed

lemma *negl-diff-imp-eq-limit*:
fixes
S :: *'a set* **and**
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
s :: *'a Preference-Relation* **and**

a :: $'a$
assumes
change: *equiv-rel-except-a* S r s a **and**
subset: $A \subseteq S$ **and**
notInA: $a \notin A$
shows *limit* A r = *limit* A s
proof −
have $A \subseteq S - \{a\}$
unfolding *subset-Diff-insert*
using *notInA subset*
by *simp*
hence $\forall\ x \in A.\ \forall\ y \in A.\ (x \preceq_r y) = (x \preceq_s y)$
using *change in-mono*
unfolding *equiv-rel-except-a-def*
by *metis*
thus *?thesis*
by *auto*
qed

theorem *lifted-above-winner*:
fixes
X :: $'a$ *set* **and**
A :: $'a$ *set* **and**
r :: $'a$ *Preference-Relation* **and**
s :: $'a$ *Preference-Relation* **and**
a :: $'a$
assumes
lifted-a: *lifted* A r s a **and**
above-x: *above* r $x = \{x\}$ **and**
fin-A: *finite* A
shows *above* s $x = \{x\} \vee$ *above* s $a = \{a\}$
proof (*cases*)
assume $x = a$
thus *?thesis*
using *above-subset-geq-one lifted-a above-x lifted-above*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
next
assume *x-neq-a*: $x \neq a$
thus *?thesis*
proof (*cases*)
assume *above* s $x = \{x\}$
thus *?thesis*
by *simp*
next
assume *x-not-above*: *above* s $x \neq \{x\}$
have $\forall\ y \in A.\ y \preceq_r x$
proof (*safe*)
fix y :: $'a$

```
      assume y-in-A: y ∈ A
      hence alts-not-empty: A ≠ {}
        by blast
      have linear-order-on A r
        using lifted-a
        unfolding equiv-rel-except-a-def lifted-def
        by simp
      with alts-not-empty y-in-A
      show y ≼r x
        using above-one above-one-2 above-x fin-A lin-ord-imp-connex
              pref-imp-in-above singletonD
        unfolding connex-def
        by (metis (no-types))
    qed
    moreover have equiv-rel-except-a A r s a
      using lifted-a
      unfolding lifted-def
      by metis
    moreover have x ∈ A − {a}
      using above-one above-one-2 x-neq-a assms calculation
            insert-not-empty member-remove insert-absorb
      unfolding equiv-rel-except-a-def remove-def
      by metis
    ultimately have ∀ y ∈ A − {a}. y ≼s x
      using DiffD1 lifted-a
      unfolding equiv-rel-except-a-def
      by metis
    hence not-others: ∀ y ∈ A − {a}. above s y ≠ {y}
      using x-not-above empty-iff insert-iff pref-imp-in-above
      by metis
    hence above s a = {a}
      using Diff-iff all-not-in-conv lifted-a fin-A above-one singleton-iff
      unfolding lifted-def equiv-rel-except-a-def
      by metis
    thus ?thesis
      by simp
  qed
qed

theorem lifted-above-winner-2:
  fixes
    A :: 'a set and
    r :: 'a Preference-Relation and
    s :: 'a Preference-Relation and
    a :: 'a
  assumes
    lifted A r s a and
    above r a = {a} and
    finite A
```

shows *above s a* = {*a*}
using *assms lifted-above-winner*
by *metis*

theorem *lifted-above-winner-3*:
fixes
A :: *'a set* **and**
r :: *'a Preference-Relation* **and**
s :: *'a Preference-Relation* **and**
a :: *'a* **and**
x :: *'a*
assumes
lifted-a: *lifted A r s a* **and**
above-x: *above s x* = {*x*} **and**
fin-A: *finite A* **and**
x-not-a: *x* ≠ *a*
shows *above r x* = {*x*}
proof (*rule ccontr*)
assume *not-above-x*: *above r x* ≠ {*x*}
then obtain *y* **where**
y: *above r y* = {*y*}
using *lifted-a fin-A insert-Diff insert-not-empty above-one*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
hence *above s y* = {*y*} ∨ *above s a* = {*a*}
using *lifted-a fin-A lifted-above-winner*
by *metis*
moreover have ∀ *b*. *above s b* = {*b*} ⟶ *b* = *x*
using *all-not-in-conv lifted-a above-x fin-A above-one-2*
unfolding *lifted-def equiv-rel-except-a-def*
by *metis*
ultimately have *y* = *x*
using *x-not-a*
by *presburger*
moreover have *y* ≠ *x*
using *not-above-x y*
by *blast*
ultimately show *False*
by *simp*
qed

end

1.2 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.

1.2.1 Definition

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

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

1.2.2 Auxiliary Functions

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

fun *set-equals-partition* :: $'a\ set \Rightarrow 'a\ Result \Rightarrow bool$ **where**
 $set\text{-}equals\text{-}partition\ A\ (e, r, d) = (e \cup r \cup d = A)$

fun *well-formed* :: $'a\ set \Rightarrow 'a\ Result \Rightarrow bool$ **where**
 $well\text{-}formed\ A\ result = (disjoint3\ result \wedge set\text{-}equals\text{-}partition\ A\ result)$

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

abbreviation *elect-r* :: $'a\ Result \Rightarrow 'a\ set$ **where**
 $elect\text{-}r\ r \equiv fst\ r$

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

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

1.2.3 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* $A\ (e,\ r,\ d)$
  shows $A - (e \cup d) = r$
proof (*safe*)
  fix $a :: 'a$
  assume
    a-in-A: $a \in A$ **and**
    a-not-rej: $a \notin r$ **and**
    a-not-def: $a \notin d$
  from *assms*
  have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
    by *simp*
  thus $a \in e$
    using *a-in-A a-not-rej a-not-def*
    by *auto*
next
  fix $a :: 'a$
  assume *a-rej*: $a \in r$
  from *assms*
  have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
    by *simp*
  thus $a \in A$
    using *a-rej*
    by *auto*
next
  fix $a :: 'a$
  assume
    a-rej: $a \in r$ **and**
    a-elec: $a \in e$
  from *assms*
  have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
    by *simp*
  thus *False*
    using *a-rej a-elec*
    by *auto*
next
  fix $a :: 'a$
  assume
    a-rej: $a \in r$ **and**
    a-def: $a \in d$
  have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\}) \wedge (e \cup r \cup d = A)$
    using *assms*
    by *simp*

thus *False*
using *a-rej a-def*
by *auto*
qed

lemma *result-count*:
fixes
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
r :: $'a$ *set* **and**
d :: $'a$ *set*
assumes
wf: *well-formed* A (e, r, d) **and**
fin-A: *finite* A
shows *card* A = *card* e + *card* r + *card* d
proof $-$
have *set-partit*: $e \cup r \cup d = A$
using *wf*
by *simp*
have $(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\})$
using *wf*
by *simp*
thus *?thesis*
using *fin-A set-partit Int-Un-distrib2 finite-Un*
card-Un-disjoint sup-bot.right-neutral
by *metis*
qed

lemma *defer-subset*:
fixes
A :: $'a$ *set* **and**
r :: $'a$ *Result*
assumes *well-formed* A r
shows *defer-r* $r \subseteq A$
proof (*safe*)
fix a :: $'a$
assume *def-a*: $a \in$ *defer-r* r
obtain
alts :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* **and**
res :: $'a$ *Result* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Result* **where**
wf: A = *alts* r A $\wedge$ r = *res* r A $\wedge$ *disjoint3* (*res* r A) $\wedge$
set-equals-partition (*alts* r A) (*res* r A)
using *assms*
by *simp*
hence $\forall$ p. $\exists$ E R D. *set-equals-partition* A p $\longrightarrow$ $(E, R, D) = p \wedge E \cup R \cup D$
$= A$
by *simp*
thus $a \in A$
using *UnCI def-a wf snd-conv*

by *metis*
qed

lemma *elect-subset*:
fixes
A :: *'a set* **and**
r :: *'a Result*
assumes *well-formed A r*
shows *elect-r r* $\subseteq$ *A*
proof (*safe*)
fix *x* :: *'a*
assume *elec-res*: *x* $\in$ *elect-r r*
obtain
alts :: *'a Result* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **and**
res :: *'a Result* $\Rightarrow$ *'a set* $\Rightarrow$ *'a Result* **where**
wf: *A* = *alts r A* $\wedge$ *r* = *res r A* $\wedge$ *disjoint3* (*res r A*) $\wedge$
set-equals-partition (*alts r A*) (*res r A*)
using *assms*
by *simp*
hence $\forall$ *p*. $\exists$ *E R D*. *set-equals-partition A p* $\longrightarrow$ (*E*, *R*, *D*) = *p* $\wedge$ *E* $\cup$ *R* $\cup$ *D* = *A*
by *simp*
thus *x* $\in$ *A*
using *UnCI elec-res wf assms fst-conv*
by *metis*
qed

lemma *reject-subset*:
fixes
A :: *'a set* **and**
r :: *'a Result*
assumes *well-formed A r*
shows *reject-r r* $\subseteq$ *A*
proof (*safe*)
fix *a* :: *'a*
assume *rej-a*: *a* $\in$ *reject-r r*
obtain
alts :: *'a Result* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **and**
res :: *'a Result* $\Rightarrow$ *'a set* $\Rightarrow$ *'a Result* **where**
wf: *A* = *alts r A* $\wedge$ *r* = *res r A* $\wedge$ *disjoint3* (*res r A*) $\wedge$
set-equals-partition (*alts r A*) (*res r A*)
using *assms*
by *simp*
hence $\forall$ *p*. $\exists$ *E R D*. *set-equals-partition A p* $\longrightarrow$ (*E*, *R*, *D*) = *p* $\wedge$ *E* $\cup$ *R* $\cup$ *D* = *A*
by *simp*
thus *a* $\in$ *A*
using *UnCI assms rej-a wf fst-conv snd-conv disjoint3.cases*
by *metis*

qed

end

1.3 Preference Profile

theory *Profile*
  imports *Preference-Relation*
begin

Preference profiles denote the decisions made by the individual voters on the eligible alternatives. They are represented in the form of one preference relation (e.g., selected on a ballot) per voter, collectively captured in a list of such preference relations. Unlike a the common preference profiles in the social-choice sense, the profiles described here considers only the (sub-)set of alternatives that are received.

1.3.1 Definition

A profile contains one ballot for each voter.

type-synonym *'a Profile* = (*'a Preference-Relation*) *list*

type-synonym *'a Election* = (*'a set* × *'a Profile*)

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

definition *profile* :: *'a set* ⇒ *'a Profile* ⇒ *bool* **where**
  profile A p ≡ ∀ *i*::*nat*. *i* < *length p* ⟶ *linear-order-on A* (*p*!*i*)

lemma *profile-set* :
  fixes
    A :: *'a set* **and**
    p :: *'a Profile*
  shows *profile A p* ≡ (∀ *b* ∈ (*set p*). *linear-order-on A b*)
  unfolding *profile-def all-set-conv-all-nth*
  by *simp*

abbreviation *finite-profile* :: *'a set* ⇒ *'a Profile* ⇒ *bool* **where**
  finite-profile A p ≡ *finite A* ∧ *profile A p*

1.3.2 Preference Counts and Comparisons

The win count for an alternative a in a profile p is the amount of ballots in p that rank alternative a in first position.

```
fun win-count :: 'a Profile ⇒ 'a ⇒ nat where
  win-count p a =
    card {i::nat. i < length p ∧ above (p!i) a = {a}}

fun win-count-code :: 'a Profile ⇒ 'a ⇒ nat where
  win-count-code Nil a = 0 |
  win-count-code (p#ps) a =
     (if (above p a = {a}) then 1 else 0) + win-count-code ps a

lemma win-count-equiv[code]:
  fixes
    p :: 'a Profile and
    x :: 'a
  shows win-count p x = win-count-code p x
proof (induction p rule: rev-induct, simp)
  case (snoc a p)
  fix
    a :: 'a Preference-Relation and
    p :: 'a Profile
  assume base-case: win-count p x = win-count-code p x
  have size-one: length [a] = 1
    by simp
  have p-pos-in-ps: ∀ i < length p. p!i = (p@[a])!i
    by (simp add: nth-append)
  have
    win-count [a] x =
      (let q = [a] in
        card {i::nat. i < length q ∧
              (let r = (q!i) in (above r x = {x}))})
    by simp
  hence one-ballot-equiv: win-count [a] x = win-count-code [a] x
    using size-one
    by (simp add: nth-Cons')
  have pref-count-induct:
    win-count (p@[a]) x =
      win-count p x + win-count [a] x
  proof (simp)
    have
      {i. i = 0 ∧ (above ([a]!i) x = {x})} =
        (if (above a x = {x}) then {0} else {})
      by (simp add: Collect-conv-if)
    hence shift-idx-a:
      card {i. i = length p ∧ (above ([a]!0) x = {x})} =
        card {i. i = 0 ∧ (above ([a]!i) x = {x})}
      by simp
```

have *set-prof-eq*:
  $\{i.\ i < Suc\ (length\ p) \wedge (above\ ((p@[a])!i)\ x = \{x\})\} =$
    $\{i.\ i < length\ p \wedge (above\ (p!i)\ x = \{x\})\} \cup$
      $\{i.\ i = length\ p \wedge (above\ ([a]!0)\ x = \{x\})\}$
proof (*safe*, *simp-all*)
  fix
    $n :: nat$ **and**
    $n' :: 'a$
  assume
    $n < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!n)\ x = \{x\}$ **and**
    $n \neq length\ p$ **and**
    $n' \in above\ (p!n)\ x$
  thus $n' = x$
    using *less-antisym p-pos-in-ps singletonD*
    by *metis*
next
  fix $n :: nat$
  assume
    $n < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!n)\ x = \{x\}$ **and**
    $n \neq length\ p$
  thus $x \in above\ (p!n)\ x$
    using *less-antisym insertI1 p-pos-in-ps*
    by *metis*
next
  fix
    $n :: nat$ **and**
    $b :: 'a$
  assume
    $n < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!n)\ x = \{x\}$ **and**
    $b \in above\ a\ x$ **and**
    $b \neq x$
  thus $n < length\ p$
    using *less-antisym nth-append-length*
        p-pos-in-ps singletonD
    by *metis*
next
  fix
    $n :: nat$ **and**
    $b :: 'a$ **and**
    $b' :: 'a$
  assume
    $n < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!n)\ x = \{x\}$ **and**
    $b \in above\ a\ x$ **and**
    $b \neq x$ **and**
    $b' \in above\ (p!n)\ x$

thus $b' = x$
using *less-antisym p-pos-in-ps*
nth-append-length singletonD
by *metis*
next
fix
n :: *nat* **and**
b :: *'a*
assume
$n < Suc\ (length\ p)$ **and**
$above\ ((p@[a])!n)\ x = \{x\}$ **and**
$b \in above\ a\ x$ **and**
$b \neq x$
thus $x \in above\ (p!n)\ x$
using *insertI1 less-antisym nth-append*
nth-append-length singletonD
by *metis*
next
fix n :: *nat*
assume
$n < Suc\ (length\ p)$ **and**
$above\ ((p@[a])!n)\ x = \{x\}$ **and**
$x \notin above\ a\ x$
thus $n < length\ p$
using *insertI1 less-antisym nth-append-length*
by *metis*
next
fix
n :: *nat* **and**
b :: *'a*
assume
$n < Suc\ (length\ p)$ **and**
$above\ ((p@[a])!n)\ x = \{x\}$ **and**
$x \notin above\ a\ x$ **and**
$b \in above\ (p!n)\ x$
thus $b = x$
using *insertI1 less-antisym nth-append-length*
p-pos-in-ps singletonD
by *metis*
next
fix n :: *nat*
assume
$n < Suc\ (length\ p)$ **and**
$above\ ((p@[a])!n)\ x = \{x\}$ **and**
$x \notin above\ a\ x$
thus $x \in above\ (p!n)\ x$
using *insertI1 less-antisym nth-append-length p-pos-in-ps*
by *metis*
next

fix
n :: *nat* **and**
b :: $'a$
assume
$n <$ *length* p **and**
above $(p!n)$ $x = \{x\}$ **and**
$b \in$ *above* $((p@[a])!n)$ x
thus $b = x$
by (*simp add*: *nth-append*)
next
fix n :: *nat*
assume
$n <$ *length* p **and**
above $(p!n)$ $x = \{x\}$
thus $x \in$ *above* $((p@[a])!n)$ x
by (*simp add*: *nth-append*)
qed
have *fin-len-p*:
finite $\{n.\ n <$ *length* $p \wedge ($*above* $(p!n)$ $x = \{x\})\}$
by *simp*
have *fin-len-a-0*:
finite $\{n.\ n =$ *length* $p \wedge ($*above* $([a]!0)$ $x = \{x\})\}$
by *simp*
have
card $(\{i.\ i <$ *length* $p \wedge ($*above* $(p!i)$ $x = \{x\})\} \cup$
$\{i.\ i =$ *length* $p \wedge ($*above* $([a]!0)$ $x = \{x\})\}) =$
card $\{i.\ i <$ *length* $p \wedge ($*above* $(p!i)$ $x = \{x\})\}$ $+$
card $\{i.\ i =$ *length* $p \wedge ($*above* $([a]!0)$ $x = \{x\})\}$
using *fin-len-p fin-len-a-0 card-Un-disjoint*
by *blast*
thus
card $\{i.\ i <$ *Suc* $($*length* $p) \wedge ($*above* $((p@[a])!i)$ $x = \{x\})\} =$
card $\{i.\ i <$ *length* $p \wedge ($*above* $(p!i)$ $x = \{x\})\}$ $+$
card $\{i.\ i = 0 \wedge ($*above* $([a]!i)$ $x = \{x\})\}$
using *set-prof-eq shift-idx-a*
by *auto*
qed
have *win-count-code* $(p@[a])$ $x =$ *win-count-code* p x $+$ *win-count-code* $[a]$ x
proof (*induction* p, *simp*)
fix
r :: $'a$ *Preference-Relation* **and**
q :: $'a$ *Profile*
assume
win-count-code $(q@[a])$ $x =$
win-count-code q x $+$ *win-count-code* $[a]$ x
thus
win-count-code $((r\#q)@[a])$ $x =$
win-count-code $(r\#q)$ x $+$ *win-count-code* $[a]$ x
by *simp*

qed
thus *win-count* $(p@[a])\ x = $ *win-count-code* $(p@[a])\ x$
using *pref-count-induct base-case one-ballot-equiv*
by *presburger*
qed

fun *prefer-count* :: *'a Profile* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *nat* **where**
prefer-count p x y $=$
card $\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\}$

fun *prefer-count-code* :: *'a Profile* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *nat* **where**
prefer-count-code Nil x y $= 0\ |$
prefer-count-code $(p\#ps)\ x\ y =$
$(if\ y \preceq_p x\ then\ 1\ else\ 0) +$ *prefer-count-code ps x y*

lemma *pref-count-equiv*[*code*]:
fixes
p :: *'a Profile* **and**
x :: *'a* **and**
y :: *'a*
shows *prefer-count p x y* $=$ *prefer-count-code p x y*
proof (*induction p rule*: *rev-induct*, *simp*)
case (*snoc a p*)
fix
a :: *'a Preference-Relation* **and**
p :: *'a Profile*
assume *base-case*: *prefer-count p x y* $=$ *prefer-count-code p x y*
have *size-one*: *length* $[a] = 1$
by *simp*
have *p-pos-in-ps*:
$\forall\ i < length\ p.\ p!i = (p@[a])!i$
by (*simp add*: *nth-append*)
have
prefer-count $[a]\ x\ y =$
$(let\ q = [a]\ in$
card $\{i{::}nat.\ i < length\ q \wedge$
$(let\ r = (q!i)\ in\ (y \preceq_r x))\})$
by *simp*
hence *one-ballot-equiv*:
prefer-count $[a]\ x\ y =$ *prefer-count-code* $[a]\ x\ y$
using *size-one*
by (*simp add*: *nth-Cons'*)
have *pref-count-induct*:
prefer-count $(p@[a])\ x\ y =$
prefer-count p x y $+$ *prefer-count* $[a]\ x\ y$
proof (*simp*)
have
$\{i.\ i = 0 \wedge (y, x) \in [a]!i\} =$
$(if\ ((y, x) \in a)\ then\ \{0\}\ else\ \{\})$

by (*simp add*: *Collect-conv-if*)
hence *shift-idx-a*:
$card\ \{i.\ i = length\ p \wedge (y, x) \in [a]!0\} =$
$card\ \{i.\ i = 0 \wedge (y, x) \in [a]!i\}$
by *simp*
have *set-prof-eq*:
$\{i.\ i < Suc\ (length\ p) \wedge (y, x) \in (p@[a])!i\} =$
$\{i.\ i < length\ p \wedge (y, x) \in p!i\} \cup$
$\{i.\ i = length\ p \wedge (y, x) \in [a]!0\}$
proof (*safe*, *simp-all*)
fix *xa* :: *nat*
assume
$xa < Suc\ (length\ p)$ **and**
$(y, x) \in (p@[a])!xa$ **and**
$xa \neq length\ p$
thus $(y, x) \in p!xa$
using *less-antisym p-pos-in-ps*
by *metis*
next
fix *xa* :: *nat*
assume
$xa < Suc\ (length\ p)$ **and**
$(y, x) \in (p@[a])!xa$ **and**
$(y, x) \notin a$
thus $xa < length\ p$
using *less-antisym nth-append-length*
by *metis*
next
fix *xa* :: *nat*
assume
$xa < Suc\ (length\ p)$ **and**
$(y, x) \in (p@[a])!xa$ **and**
$(y, x) \notin a$
thus $(y, x) \in p!xa$
using *less-antisym nth-append-length p-pos-in-ps*
by *metis*
next
fix *xa* :: *nat*
assume
$xa < length\ p$ **and**
$(y, x) \in p!xa$
thus $(y, x) \in (p@[a])!xa$
using *less-antisym p-pos-in-ps*
by *metis*
qed
have *fin-len-p*: $finite\ \{n.\ n < length\ p \wedge (y, x) \in p!n\}$
by *simp*
have *fin-len-a-0*: $finite\ \{n.\ n = length\ p \wedge (y, x) \in [a]!0\}$
by *simp*

have
card $(\{i.\ i < length\ p \wedge (y, x) \in p!i\} \cup$
$\{i.\ i = length\ p \wedge (y, x) \in [a]!0\}) =$
card $\{i.\ i < length\ p \wedge (y, x) \in p!i\} +$
card $\{i.\ i = length\ p \wedge (y, x) \in [a]!0\}$
using *fin-len-p fin-len-a-0 card-Un-disjoint*
by *blast*
thus
card $\{i.\ i < Suc\ (length\ p) \wedge (y, x) \in (p@[a])!i\} =$
card $\{i.\ i < length\ p \wedge (y, x) \in p!i\} +$
card $\{i.\ i = 0 \wedge (y, x) \in [a]!i\}$
using *set-prof-eq shift-idx-a*
by *simp*
qed
have *pref-count-code-induct*:
prefer-count-code $(p@[a])\ x\ y =$
prefer-count-code $p\ x\ y$ + *prefer-count-code* $[a]\ x\ y$
proof (*simp*, *safe*)
assume *y-pref-x*: $(y, x) \in a$
show *prefer-count-code* $(p@[a])\ x\ y = Suc$ (*prefer-count-code* $p\ x\ y$)
proof (*induction p*, *simp-all*)
show $(y, x) \in a$
using *y-pref-x*
by *simp*
qed
next
assume *not-y-pref-x*: $(y, x) \notin a$
show *prefer-count-code* $(p@[a])\ x\ y$ = *prefer-count-code* $p\ x\ y$
proof (*induction p*, *simp-all*, *safe*)
assume $(y, x) \in a$
thus *False*
using *not-y-pref-x*
by *simp*
qed
qed
show *prefer-count* $(p@[a])\ x\ y$ = *prefer-count-code* $(p@[a])\ x\ y$
using *pref-count-code-induct pref-count-induct*
base-case one-ballot-equiv
by *presburger*
qed

lemma *set-compr*:
fixes
$S :: 'a\ set$ **and**
$f :: 'a \Rightarrow 'a\ set$
shows $\{\ f\ x \mid x.\ x \in S\ \} = f\ `\ S$
by *auto*

lemma *pref-count-set-compr*:

fixes
  A :: $'a$ *set* **and**
  p :: $'a$ *Profile* **and**
  x :: $'a$
shows $\{prefer\text{-}count\ p\ x\ y \mid y.\ y \in A - \{x\}\} = (prefer\text{-}count\ p\ x)\ \text{`}\ (A - \{x\})$
by *auto*

lemma *pref-count*:
fixes
  A :: $'a$ *set* **and**
  p :: $'a$ *Profile* **and**
  x :: $'a$ **and**
  y :: $'a$
assumes
  prof: *profile* A p **and**
  x-in-A: $x \in A$ **and**
  y-in-A: $y \in A$ **and**
  neq: $x \neq y$
shows $prefer\text{-}count\ p\ x\ y = (length\ p) - (prefer\text{-}count\ p\ y\ x)$
proof $-$
have *00*: $card\ \{i{::}nat.\ i < length\ p\} = length\ p$
  by *simp*
have *10*:
  $\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\} =$
    $\{i{::}nat.\ i < length\ p\} -$
      $\{i{::}nat.\ i < length\ p \wedge \neg\ (let\ r = (p!i)\ in\ (y \preceq_r x))\}$
  by *auto*
have *0*: $\forall\ i{::}nat.\ i < length\ p \longrightarrow linear\text{-}order\text{-}on\ A\ (p!i)$
  using *prof*
  unfolding *profile-def*
  by *simp*
hence $\forall\ i{::}nat.\ i < length\ p \longrightarrow connex\ A\ (p!i)$
  by (*simp add*: *lin-ord-imp-connex*)
hence *1*: $\forall\ i{::}nat.\ i < length\ p \longrightarrow$
      $\neg\ (let\ r = (p!i)\ in\ (y \preceq_r x)) \longrightarrow (let\ r = (p!i)\ in\ (x \preceq_r y))$
  using *x-in-A y-in-A*
  unfolding *connex-def*
  by *metis*
from *0*
have $\forall\ i{::}nat.\ i < length\ p \longrightarrow antisym\ (p!i)$
  using *lin-imp-antisym*
  by *metis*
hence $\forall\ i{::}nat.\ i < length\ p \longrightarrow ((y, x) \in (p!i) \longrightarrow (x, y) \notin (p!i))$
  using *antisymD neq*
  by *metis*
hence $\forall\ i{::}nat.\ i < length\ p \longrightarrow$
    $((let\ r = (p!i)\ in\ (y \preceq_r x)) \longrightarrow \neg\ (let\ r = (p!i)\ in\ (x \preceq_r y)))$
  by *simp*
with *1*

have
  $\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$
    ¬ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *x*)) = (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))
  by *metis*
hence *2*:
  {*i*::*nat*. *i* < *length p* ∧ ¬ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *x*))} =
    {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))}
  by *metis*
hence *20*:
  {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *x*))} =
    {*i*::*nat*. *i* < *length p*} −
      {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))}
  using *10 2*
  by *simp*
have
  {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))} ⊆
    {*i*::*nat*. *i* < *length p*}
  by (*simp add*: *Collect-mono*)
hence *30*:
  card ({*i*::*nat*. *i* < *length p*} −
    {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))}) =
   (*card* {*i*::*nat*. *i* < *length p*}) −
    card ({*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))})
  by (*simp add*: *card-Diff-subset*)
have *prefer-count p x y* =
      card {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *x*))}
  by *simp*
also have
  ... = *card* ({*i*::*nat*. *i* < *length p*} −
      {*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))})
  using *20*
  by *simp*
also have
  ... = (*card* {*i*::*nat*. *i* < *length p*}) −
       card ({*i*::*nat*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*x* $\preceq_r$ *y*))})
  using *30*
  by *metis*
also have
  ... = *length p* − (*prefer-count p y x*)
  by *simp*
finally show *?thesis*
  by (*simp add*: *20 30*)
qed

lemma *pref-count-sym*:
  fixes
    p :: *'a Profile* **and**
    x :: *'a* **and**
    a :: *'a* **and**

b :: $'a$

assumes

pref-count-ineq: *prefer-count* p a x $\geq$ *prefer-count* p x b **and**

prof: *profile* A p **and**

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

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

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

neq-1: $a \neq x$ **and**

neq-2: $x \neq b$

shows *prefer-count* p b x $\geq$ *prefer-count* p x a

proof −

from *prof a-in-A x-in-A neq-1*

have *0*: *prefer-count* p a x = (*length* p) − (*prefer-count* p x a)

using *pref-count*

by *metis*

moreover from *prof x-in-A b-in-A neq-2*

have *1*: *prefer-count* p x b = (*length* p) − (*prefer-count* p b x)

using *pref-count*

by (*metis* (*mono-tags*, *lifting*))

hence *2*: (*length* p) − (*prefer-count* p x a) $\geq$
(*length* p) − (*prefer-count* p b x)

using *calculation pref-count-ineq*

by *auto*

hence *3*: (*prefer-count* p x a) − (*length* p) $\leq$
(*prefer-count* p b x) − (*length* p)

using *a-in-A diff-is-0-eq diff-le-self neq-1*
pref-count prof x-in-A

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

hence (*prefer-count* p x a) $\leq$ (*prefer-count* p b x)

using *1 3 calculation pref-count-ineq*

by *linarith*

thus *?thesis*

by *linarith*

qed

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

fixes p :: $'a$ *Profile*

assumes $p = []$

shows $\forall$ x y. *prefer-count* p x y = *0*

using *assms*

by *simp*

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

fixes p :: $'a$ *Profile*

assumes $p = []$

shows $\forall$ x y. *prefer-count-code* p x y = *0*

using *assms*

by *simp*

lemma *pref-count-code-incr*:
fixes
ps :: *'a Profile* **and**
p :: *'a Preference-Relation* **and**
x :: *'a* **and**
y :: *'a* **and**
n :: *nat*
assumes
prefer-count-code ps x y = *n* **and**
$y \preceq_p x$
shows *prefer-count-code* (*p#ps*) *x y* = *n+1*
using *assms*
by *simp*

lemma *pref-count-code-not-smaller-imp-constant*:
fixes
ps :: *'a Profile* **and**
p :: *'a Preference-Relation* **and**
x :: *'a* **and**
y :: *'a* **and**
n :: *nat*
assumes
prefer-count-code ps x y = *n* **and**
$\neg(y \preceq_p x)$
shows *prefer-count-code* (*p#ps*) *x y* = *n*
using *assms*
by *simp*

fun *wins* :: *'a* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool* **where**
wins x p y =
(*prefer-count p x y* > *prefer-count p y x*)

Alternative a wins against b implies that b does not win against a.

lemma *wins-antisym*:
fixes
p :: *'a Profile* **and**
a :: *'a* **and**
b :: *'a*
assumes *wins a p b*
shows ¬ *wins b p a*
using *assms*
by *simp*

lemma *wins-irreflex*:
fixes
p :: *'a Profile* **and**
a :: *'a*
shows ¬ *wins a p a*
using *wins-antisym*

by *metis*

1.3.3 Condorcet Winner

fun *condorcet-winner* :: *'a set* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool* **where**
condorcet-winner A p w =
$(finite\text{-}profile\ A\ p \land\ w \in A \land (\forall\ x \in A - \{w\}.\ wins\ w\ p\ x))$

lemma *cond-winner-unique*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a* **and**
b :: *'a*
assumes
winner-a: *condorcet-winner A p a* **and**
winner-b: *condorcet-winner A p b*
shows $b = a$
proof (*rule ccontr*)
assume *b-neq-a*: $b \neq a$
from *winner-b*
have *wins b p a*
using *b-neq-a insert-Diff insert-iff winner-a*
by *simp*
hence ¬ *wins a p b*
by (*simp add*: *wins-antisym*)
moreover from *winner-a*
have
a-wins-against-b: *wins a p b*
using *Diff-iff b-neq-a*
singletonD winner-b
by *simp*
ultimately show *False*
by *simp*
qed

lemma *cond-winner-unique-2*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a* **and**
b :: *'a*
assumes
winner: *condorcet-winner A p a* **and**
not-a: $b \neq a$
shows ¬ *condorcet-winner A p b*
using *not-a cond-winner-unique winner*
by *metis*

```
lemma cond-winner-unique-3:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assumes condorcet-winner A p a
  shows {b ∈ A. condorcet-winner A p b} = {a}
proof (safe, simp-all, safe)
  fix b :: 'a
  assume
    fin-A: finite A and
    prof-A: profile A p and
    b-in-A: b ∈ A and
    b-wins:
      ∀ x ∈ A − {b}.
        card {i. i < length p ∧ (b, x) ∈ p!i} <
          card {i. i < length p ∧ (x, b) ∈ p!i}
  from assms
  have assm:
    finite-profile A p ∧  a ∈ A ∧
      (∀ x ∈ A − {a}.
        card {i::nat. i < length p ∧ (a, x) ∈ p!i} <
          card {i::nat. i < length p ∧ (x, a) ∈ p!i})
    by simp
  hence
    (∀ x ∈ A − {a}.
      card {i::nat. i < length p ∧ (a, x) ∈ p!i} <
        card {i::nat. i < length p ∧ (x, a) ∈ p!i})
    by simp
  hence a-beats-b:
    b ≠ a ⟹
      card {i::nat. i < length p ∧ (a, b) ∈ p!i} <
        card {i::nat. i < length p ∧ (b, a) ∈ p!i}
    using b-in-A
    by simp
  also from assm
  have finite-profile A p
    by simp
  moreover from assm
  have a ∈ A
    by simp
  hence b-beats-a:
    b ≠ a ⟹
      card {i. i < length p ∧ (b, a) ∈ p!i} <
        card {i. i < length p ∧ (a, b) ∈ p!i}
    using b-wins
    by simp
  from a-beats-b b-beats-a
  show b = a
```

```
    by linarith
next
  from assms
  show a ∈ A
    by simp
next
  from assms
  show finite A
    by simp
next
  from assms
  show profile A p
    by simp
next
  from assms
  show a ∈ A
    by simp
next
  fix b :: 'a
  assume
    b-in-A: b ∈ A and
    a-wins:
      ¬ card {i. i < length p ∧ (a, b) ∈ p!i} <
        card {i. i < length p ∧ (b, a) ∈ p!i}
  from assms
  have
    finite-profile A p ∧ a ∈ A ∧
      (∀ x ∈ A − {a}.
        card {i::nat. i < length p ∧ (a, x) ∈ p!i} <
          card {i::nat. i < length p ∧ (x, a) ∈ p!i})
    by simp
  thus b = a
    using b-in-A a-wins insert-Diff insert-iff
    by (metis (no-types, lifting))
qed
```

1.3.4 Limited Profile

This function restricts a profile p to a set A and keeps all of A's preferences.

```
fun limit-profile :: 'a set ⇒ 'a Profile ⇒ 'a Profile where
  limit-profile A p = map (limit A) p

lemma limit-prof-trans:
  fixes
    A :: 'a set and
    B :: 'a set and
    C :: 'a set and
    p :: 'a Profile
  assumes
```

$B \subseteq A$ **and**
$C \subseteq B$ **and**
finite-profile A p
shows *limit-profile C p* = *limit-profile C* (*limit-profile B p*)
using *assms*
by *auto*

lemma *limit-profile-sound*:
fixes
A :: $'a$ *set* **and**
B :: $'a$ *set* **and**
p :: $'a$ *Profile*
assumes
profile: *finite-profile B p* **and**
subset: $A \subseteq B$
shows *finite-profile A* (*limit-profile A p*)
proof (*safe*)
have *finite B* $\longrightarrow A \subseteq B \longrightarrow$ *finite A*
using *rev-finite-subset*
by *metis*
with *profile*
show *finite A*
using *subset*
by *metis*
next
have *prof-is-lin-ord*:
$\forall$ *C q*.
(*profile* (*C*::$'a$ *set*) *q* $\longrightarrow$ ($\forall$ *n* < *length q*. *linear-order-on C* (*q*!*n*))) $\wedge$
(($\forall$ *n* < *length q*. *linear-order-on C* (*q*!*n*)) $\longrightarrow$ *profile C q*)
unfolding *profile-def*
by *simp*
have *limit-prof-simp*: *limit-profile A p* = *map* (*limit A*) *p*
by *simp*
obtain *n* :: *nat* **where**
prof-limit-n: (*n* < *length* (*limit-profile A p*) $\longrightarrow$
linear-order-on A (*limit-profile A p*!*n*)) $\longrightarrow$
profile A (*limit-profile A p*)
using *prof-is-lin-ord*
by *metis*
have *prof-n-lin-ord*: $\forall$ *n* < *length p*. *linear-order-on B* (*p*!*n*)
using *prof-is-lin-ord profile*
by *simp*
have *prof-length*: *length p* = *length* (*map* (*limit A*) *p*)
by *simp*
have *n* < *length p* $\longrightarrow$ *linear-order-on B* (*p*!*n*)
using *prof-n-lin-ord*
by *simp*
thus *profile A* (*limit-profile A p*)
using *prof-length prof-limit-n limit-prof-simp limit-presv-lin-ord nth-map subset*

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

lemma *limit-prof-presv-size*:
fixes
A :: *'a set* **and**
p :: *'a Profile*
shows *length p* = *length* (*limit-profile A p*)
by *simp*

1.3.5 Lifting Property

definition *equiv-prof-except-a* :: *'a set* ⇒ *'a Profile* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool*
where
equiv-prof-except-a A p q a ≡
finite-profile A p ∧ *finite-profile A q* ∧
a ∈ *A* ∧ *length p* = *length q* ∧
(∀ *i*::*nat*.
i < *length p* ⟶
equiv-rel-except-a A (*p*!*i*) (*q*!*i*) *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* :: *'a set* ⇒ *'a Profile* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool* **where**
lifted A p q a ≡
finite-profile A p ∧ *finite-profile A q* ∧
a ∈ *A* ∧ *length p* = *length q* ∧
(∀ *i*::*nat*.
(*i* < *length p* ∧ ¬*Preference-Relation.lifted A* (*p*!*i*) (*q*!*i*) *a*) ⟶
(*p*!*i*) = (*q*!*i*)) ∧
(∃ *i*::*nat*. *i* < *length p* ∧ *Preference-Relation.lifted A* (*p*!*i*) (*q*!*i*) *a*)

lemma *lifted-imp-equiv-prof-except-a*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes *lifted A p q a*
shows *equiv-prof-except-a A p q a*
proof (*unfold equiv-prof-except-a-def*, *safe*)
from *assms*
show *finite A*
unfolding *lifted-def*
by *metis*
next
from *assms*
show *profile A p*
unfolding *lifted-def*

by *metis*
next
from *assms*
show *finite A*
unfolding *lifted-def*
by *metis*
next
from *assms*
show *profile A q*
unfolding *lifted-def*
by *metis*
next
from *assms*
show $a \in A$
unfolding *lifted-def*
by *metis*
next
from *assms*
show *length p* = *length q*
unfolding *lifted-def*
by *metis*
next
fix *i* :: *nat*
assume *i* < *length p*
with *assms*
show *equiv-rel-except-a A* (*p*!*i*) (*q*!*i*) *a*
using *lifted-imp-equiv-rel-except-a trivial-equiv-rel*
unfolding *lifted-def profile-def*
by (*metis* (*no-types*))
qed

lemma *negl-diff-imp-eq-limit-prof*:
fixes
A :: *'a set* **and**
B :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes
change: *equiv-prof-except-a B p q a* **and**
subset: $A \subseteq B$ **and**
not-in-A: $a \notin A$
shows *limit-profile A p* = *limit-profile A q*
proof (*simp*)
have
∀ *i*::*nat*. *i* < *length p* ⟶
equiv-rel-except-a B (*p*!*i*) (*q*!*i*) *a*
using *change equiv-prof-except-a-def*
by *metis*

hence $\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$ *limit A* (*p*!*i*) = *limit A* (*q*!*i*)
  using *not-in-A negl-diff-imp-eq-limit subset*
  by *metis*
thus *map* (*limit A*) *p* = *map* (*limit A*) *q*
  using *change equiv-prof-except-a-def*
    length-map nth-equalityI nth-map
  by (*metis* (*mono-tags*, *lifting*))
qed

lemma *limit-prof-eq-or-lifted*:
  fixes
    A :: *'a set* **and**
    B :: *'a set* **and**
    p :: *'a Profile* **and**
    q :: *'a Profile* **and**
    a :: *'a*
  assumes
    lifted-a: *lifted B p q a* **and**
    subset: *A* $\subseteq$ *B*
  shows
    limit-profile A p = *limit-profile A q* $\vee$
      lifted A (*limit-profile A p*) (*limit-profile A q*) *a*
proof (*cases*)
  assume *a-in-A*: *a* $\in$ *A*
  have
    $\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$
      (*Preference-Relation.lifted B* (*p*!*i*) (*q*!*i*) *a* $\vee$ (*p*!*i*) = (*q*!*i*))
    using *lifted-a*
    unfolding *lifted-def*
    by *metis*
  hence *one*:
    $\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$
      (*Preference-Relation.lifted A* (*limit A* (*p*!*i*)) (*limit A* (*q*!*i*)) *a* $\vee$
        (*limit A* (*p*!*i*)) = (*limit A* (*q*!*i*)))
    using *limit-lifted-imp-eq-or-lifted subset*
    by *metis*
  thus *?thesis*
  proof (*cases*)
    assume $\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$ (*limit A* (*p*!*i*)) = (*limit A* (*q*!*i*))
    thus *?thesis*
      using *length-map lifted-a nth-equalityI nth-map*
        limit-profile.simps
      unfolding *lifted-def*
      by (*metis* (*mono-tags*, *lifting*))
  next
    assume *forall-limit-p-q*: $\neg$($\forall$ *i*::*nat*. *i* < *length p* $\longrightarrow$ (*limit A* (*p*!*i*)) = (*limit A* (*q*!*i*)))
    let *?p* = *limit-profile A p*
    let *?q* = *limit-profile A q*

have *profile A ?p* ∧ *profile A ?q*
using *lifted-a limit-profile-sound subset*
unfolding *lifted-def*
by *metis*
moreover have *length ?p* = *length ?q*
using *lifted-a*
unfolding *lifted-def*
by *fastforce*
moreover have ∃ *i*::*nat*. *i* < *length ?p* ∧ *Preference-Relation.lifted A* (*?p*!*i*) (*?q*!*i*) *a*
using *forall-limit-p-q length-map lifted-a limit-profile.simps nth-map one*
unfolding *lifted-def*
by (*metis* (*no-types*, *lifting*))
moreover have
∀ *i*::*nat*.
(*i* < *length ?p* ∧ ¬*Preference-Relation.lifted A* (*?p*!*i*) (*?q*!*i*) *a*) ⟶
(*?p*!*i*) = (*?q*!*i*)
using *length-map lifted-a limit-profile.simps nth-map one*
unfolding *lifted-def*
by *metis*
ultimately have *lifted A ?p ?q a*
using *a-in-A lifted-a rev-finite-subset subset*
unfolding *lifted-def*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
by *simp*
qed
next
assume *a* ∉ *A*
thus *?thesis*
using *lifted-a negl-diff-imp-eq-limit-prof subset*
lifted-imp-equiv-prof-except-a
by *metis*
qed

end

1.4 Preference List

theory *Preference-List*
imports ../*Preference-Relation*
List−Index.List-Index
begin

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

1.4.1 Well-Formedness

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

abbreviation *well-formed-l* :: *'a Preference-List* ⇒ *bool* **where**
  well-formed-l p ≡ *distinct p*

1.4.2 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 x = (*if* (*List.member l x*) *then* (*index l x*) + *1 else 0*)

definition *above-l* :: *'a Preference-List* ⇒ *'a* ⇒ *'a Preference-List* **where**
  above-l r c ≡ *take* (*rank-l r c*) *r*

lemma *rank-zero-imp-not-present*:
  fixes
    p :: *'a Preference-List* **and**
    a :: *'a*
  assumes *rank-l p a* = *0*
  shows ¬ *List.member p a*
  using *assms*
  by *force*

1.4.3 Definition

fun *is-less-preferred-than-l* ::
  'a ⇒ *'a Preference-List* ⇒ *'a* ⇒ *bool* (- $\lesssim$- - [*50*, *1000*, *51*] *50*) **where**
    $x \lesssim_l y$ = ((*List.member l x*) ∧ (*List.member l y*) ∧ (*rank-l l x* ≥ *rank-l l y*))

lemma *rank-gt-zero*:
  fixes
    l :: *'a Preference-List* **and**
    x :: *'a*
  assumes
    wf : *well-formed-l l* **and**
    refl: $x \lesssim_l x$
  shows *rank-l l x* ≥ *1*
  using *refl*
  by *simp*

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

lemma *rel-trans*:
  fixes *l* :: *'a Preference-List*
  shows *Relation.trans* (*pl-α l*)

unfolding *Relation.trans-def pl-α-def*
by *simp*

1.4.4 Limited Preference

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

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

lemma *limitedI*:
fixes
$l :: 'a\ Preference\text{-}List$ **and**
$A :: 'a\ set$
assumes $\bigwedge x\ y.\ x \lesssim_l y \Longrightarrow x \in A \wedge y \in A$
shows *limited A l*
using *assms*
unfolding *limited-def*
by *auto*

lemma *limited-dest*:
fixes
$A :: 'a\ set$ **and**
$l :: 'a\ Preference\text{-}List$ **and**
$x :: 'a$ **and**
$y :: 'a$
assumes
is-less-preferred-than-l x l y **and**
limited A l
shows $x \in A \wedge y \in A$
using *assms*
unfolding *limited-def*
by *simp*

1.4.5 Auxiliary Definitions

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

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

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

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

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

linear-order-on-l A pl $\equiv$ *preorder-on-l A pl* $\wedge$ *total-on-l A pl*

definition *connex-l* :: *'a set* $\Rightarrow$ *'a Preference-List* $\Rightarrow$ *bool* **where**
connex-l A r $\equiv$ *limited A r* $\wedge$ ($\forall$ $x \in A$. $\forall$ $y \in A$. $x \lesssim_r y \vee y \lesssim_r x$)

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

1.4.6 Auxiliary Lemmas

lemma *connex-imp-refl*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List*
assumes *connex-l A l*
shows *refl-on-l A l*
unfolding *connex-l-def refl-on-l-def*
using *assms connex-l-def*
by *metis*

lemma *lin-ord-imp-connex-l*:
fixes
A :: *'a set* **and**
r :: *'a Preference-List*
assumes *linear-order-on-l A r*
shows *connex-l A r*
using *assms Preference-List.connex-l-def is-less-preferred-than-l.simps linear-order-on-l-def preorder-on-l-def total-on-l-def assms nle-le*
by *metis*

lemma *above-trans*:
fixes
l :: *'a Preference-List* **and**
a :: *'a* **and**
b :: *'a*
assumes
trans: *trans l* **and**
less: $a \lesssim_l b$
shows *set* (*above-l l b*) $\subseteq$ *set* (*above-l l a*)
using *assms above-l-def set-take-subset-set-take*
unfolding *Preference-List.is-less-preferred-than-l.simps*
by *metis*

lemma *less-preferred-l-rel-eq*:
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*

unfolding *pl-α-def*
by *simp*

theorem *above-eq*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List* **and**
a :: *'a*
assumes
wf: *well-formed-l l* **and**
lin-ord: *linear-order-on-l A l*
shows *set* (*above-l l a*) = *Order-Relation.above* (*pl-α l*) *a*
proof (*safe*)
fix *b* :: *'a*
assume *b-member*: *b* ∈ *set* (*Preference-List.above-l l a*)
have *length* (*above-l l a*) = *rank-l l a*
unfolding *above-l-def*
using *Suc-le-eq*
by (*simp add*: *in-set-member*)
with *b-member wf lin-ord*
have *index l b* ≤ *index l a*
unfolding *rank-l.simps*
using *above-l-def Preference-List.rank-l.simps Suc-eq-plus1 Suc-le-eq bot-nat-0.extremum-strict index-take linorder-not-less*
by *metis*
with *b-member*
have $a \lesssim_l b$
using *above-l-def is-less-preferred-than-l.elims*(*3*) *rank-l.simps One-nat-def Suc-le-mono add.commute add-0 add-Suc empty-iff find-index-le-size in-set-member index-def le-antisym list.set*(*1*) *size-index-conv take-0*
by *metis*
hence *Preference-Relation.is-less-preferred-than a* (*pl-α l*) *b*
using *less-preferred-l-rel-eq*
by *metis*
thus *b* ∈ *Order-Relation.above* (*pl-α l*) *a*
using *pref-imp-in-above*
by *metis*
next
fix *b* :: *'a*
assume *b* ∈ *Order-Relation.above* (*pl-α l*) *a*
hence *Preference-Relation.is-less-preferred-than a* (*pl-α l*) *b*
using *pref-imp-in-above*
by *metis*
hence *a-less-pref-than-b*: $a \lesssim_l b$
using *less-preferred-l-rel-eq*
by *metis*
hence *rank-l l b* ≤ *rank-l l a*
by *auto*
with *a-less-pref-than-b*

show *b* ∈ *set* (*Preference-List.above-l l a*)
unfolding *Preference-List.above-l-def Preference-List.is-less-preferred-than-l.simps Preference-List.rank-l.simps*
using *Suc-eq-plus1 Suc-le-eq in-set-member index-less-size-conv set-take-if-index*
by (*metis* (*full-types*))
qed

theorem *rank-eq*:
fixes
A :: *'a set* **and**
l :: *'a Preference-List* **and**
a :: *'a*
assumes
wf: *well-formed-l l* **and**
lin-ord: *linear-order-on-l A l*
shows *rank-l l a* = *Preference-Relation.rank* (*pl-α l*) *a*
proof (*simp*, *safe*)
assume *a-in-l*: *List.member l a*
have *above-presv-rel*: *Order-Relation.above* (*pl-α l*) *a* = *set* (*above-l l a*)
using *wf lin-ord*
by (*simp add*: *above-eq*)
have *dist-l*: *distinct* (*above-l l a*)
unfolding *above-l-def*
using *wf distinct-take*
by *blast*
have *above-presv-card*: *card* (*set* (*above-l l a*)) = *length* (*above-l l a*)
using *dist-l distinct-card*
by *blast*
have *length* (*above-l l a*) = *rank-l l a*
unfolding *above-l-def*
by (*simp add*: *Suc-le-eq in-set-member*)
with *a-in-l above-presv-rel dist-l above-presv-card*
show *Suc* (*index l a*) = *card* (*Order-Relation.above* (*pl-α l*) *a*)
by *simp*
next
assume *a-not-in-l*: ¬ *List.member l a*
hence *a* ∉ (*set l*)
unfolding *pl-α-def in-set-member*
by *simp*
hence *a* ∉ *Order-Relation.above* (*pl-α l*) *a*
unfolding *Order-Relation.above-def pl-α-def*
using *a-not-in-l*
by *simp*
hence *Order-Relation.above* (*pl-α l*) *a* = {}
unfolding *Order-Relation.above-def*
using *a-not-in-l less-preferred-l-rel-eq*
by *fastforce*
thus *card* (*Order-Relation.above* (*pl-α l*) *a*) = *0*
by *fastforce*

qed

theorem *lin-ord-l-imp-rel*:
  fixes
    A :: $'a$ *set* **and**
    l :: $'a$ *Preference-List*
  assumes
    wf: *well-formed-l* l **and**
    lin-ord: *linear-order-on-l* A l
  shows *Order-Relation.linear-order-on* A (*pl-*α l)
proof (*unfold Order-Relation.linear-order-on-def partial-order-on-def*
      Order-Relation.preorder-on-def, *clarsimp*, *safe*)
  have *refl-on-l* A l
    using *lin-ord*
    unfolding *linear-order-on-l-def preorder-on-l-def*
    by *simp*
  thus *refl-on* A (*pl-*α l)
    using *lin-ord*
    unfolding *refl-on-l-def pl-*α*-def refl-on-def linear-order-on-l-def*
        preorder-on-l-def Preference-List.limited-def
    by *fastforce*
next
  show *Relation.trans* (*pl-*α l)
    unfolding *Preference-List.trans-def pl-*α*-def Relation.trans-def*
    by *simp*
next
  show *antisym* (*pl-*α l)
  proof (*unfold antisym-def pl-*α*-def is-less-preferred-than.simps*, *clarsimp*)
    fix
      a :: $'a$ **and**
      b :: $'a$
    assume
      List.member l a **and**
      index l a = *index* l b
    thus $a = b$
      unfolding *member-def*
      by *simp*
  qed
next
  have *linear-order-on-l* A l $\longrightarrow$ *connex-l* A l
    by (*simp add*: *lin-ord-imp-connex-l*)
  hence *connex-l* A l
    using *lin-ord*
    by *metis*
  thus *total-on* A (*pl-*α l)
    unfolding *connex-l-def pl-*α*-def total-on-def*
    by *simp*
qed

lemma *lin-ord-rel-imp-l*:
  fixes
    A :: *'a set* **and**
    l :: *'a Preference-List*
  assumes *Order-Relation.linear-order-on A* (*pl-α l*)
  shows *linear-order-on-l A l*
proof (*unfold linear-order-on-l-def preorder-on-l-def*, *clarsimp*, *safe*)
  show *Preference-List.limited A l*
    unfolding *pl-α-def linear-order-on-def*
   using *assms limitedI linear-order-on-def less-preferred-l-rel-eq partial-order-onD*(*1*)
      Preference-Relation.is-less-preferred-than.elims(*2*) *refl-on-def'* *case-prodD*
    by *metis*
next
  show *refl-on-l A l*
    unfolding *pl-α-def refl-on-l-def*
    using *assms Preference-Relation.lin-ord-imp-connex less-preferred-l-rel-eq*
      Preference-Relation.connex-def
    by *metis*
next
  show *Preference-List.trans l*
    unfolding *pl-α-def Preference-List.trans-def*
    by *fastforce*
next
  show *total-on-l A l*
    unfolding *pl-α-def*
    using *connex-def lin-ord-imp-connex assms total-on-l-def less-preferred-l-rel-eq*
      is-less-preferred-than-l.elims(*2*)
    by *metis*
qed

end

1.5 Preference (List) Profile

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

1.5.1 Definition

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

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

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

Abstraction from profile list to profile.

fun *pl-to-pr-α* :: *'a Profile-List* ⇒ *'a Profile* **where**
pl-to-pr-α pl = *map* (*Preference-List.pl-α*) *pl*

lemma *prof-abstr-presv-size*:
fixes *p* :: *'a Profile-List*
shows *length p* = *length* (*pl-to-pr-α p*)
by *simp*

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

definition *profile-l* :: *'a 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 A* (*pl-to-pr-α p*)
proof (*unfold profile-def*, *intro allI impI*)
fix *i* :: *nat*
assume *ir*: *i* < *length* (*pl-to-pr-α p*)
from *ir assms*
have *wf*: *well-formed-l* (*p*!*i*)
unfolding *profile-l-def*
by *simp*
from *ir assms*
have *linear-order-on-l A* (*p*!*i*)
unfolding *profile-l-def*
by *simp*
with *wf assms*
show *linear-order-on A* ((*pl-to-pr-α p*)!*i*)
using *lin-ord-l-imp-rel ir length-map nth-map pl-to-pr-α.simps*
by *metis*
qed

end

Chapter 2

Component Types

2.1 Electoral Module

theory *Electoral-Module*
  imports *Social-Choice-Types/Profile*
    Social-Choice-Types/Result
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.

2.1.1 Definition

An electoral module maps a set of alternatives and a profile to a result.

type-synonym $'a$ *Electoral-Module* $=$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Result*

2.1.2 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.

definition *electoral-module* :: *'a Electoral-Module* ⇒ *bool* **where**
electoral-module m ≡ ∀ *A p. finite-profile A p* ⟶ *well-formed A* (*m A p*)

lemma *electoral-modI*:
fixes *m* :: *'a Electoral-Module*
assumes ⋀ *A p. finite-profile A p* ⟹ *well-formed A* (*m A p*)
shows *electoral-module m*
unfolding *electoral-module-def*
using *assms*
by *simp*

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 Electoral-Module ⇒ *'a set* ⇒ *'a Profile* ⇒ *'a set* **where**
elect m A p ≡ *elect-r* (*m A p*)

abbreviation *reject* ::
'a Electoral-Module ⇒ *'a set* ⇒ *'a Profile* ⇒ *'a set* **where**
reject m A p ≡ *reject-r* (*m A p*)

abbreviation *defer* ::
'a Electoral-Module ⇒ *'a set* ⇒ *'a Profile* ⇒ *'a set* **where**
defer m A p ≡ *defer-r* (*m A p*)

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

definition *defers* :: *nat* ⇒ *'a Electoral-Module* ⇒ *bool* **where**
defers n m ≡
electoral-module m ∧
(∀ *A p.* (*card A* ≥ *n* ∧ *finite-profile A p*) ⟶
card (*defer m 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* ⇒ *'a Electoral-Module* ⇒ *bool* **where**
rejects n m ≡
electoral-module m ∧
(∀ *A p.* (*card A* ≥ *n* ∧ *finite-profile A p*) ⟶ *card* (*reject m A p*) = *n*)

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

definition *eliminates* :: *nat* ⇒ *'a Electoral-Module* ⇒ *bool* **where**
eliminates n m ≡

electoral-module m $\wedge$
$(\forall\ A\ p.\ (card\ A > n \wedge finite\text{-}profile\ A\ p) \longrightarrow card\ (reject\ m\ A\ p) = n)$

"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 Electoral-Module* $\Rightarrow$ *bool* **where**
elects n m $\equiv$
electoral-module m $\wedge$
$(\forall\ A\ p.\ (card\ A \geq n \wedge finite\text{-}profile\ A\ p) \longrightarrow card\ (elect\ m\ A\ p) = n)$

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

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

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

2.1.3 Equivalence Definitions

definition *prof-contains-result* :: *'a Electoral-Module* $\Rightarrow$ *'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$
'a Profile $\Rightarrow$ *'a* $\Rightarrow$ *bool* **where**
prof-contains-result m A p q a $\equiv$
electoral-module m $\wedge$ *finite-profile A p* $\wedge$ *finite-profile A q* $\wedge$ $a \in A$ $\wedge$
$(a \in elect\ m\ A\ p \longrightarrow a \in elect\ m\ A\ q) \wedge$
$(a \in reject\ m\ A\ p \longrightarrow a \in reject\ m\ A\ q) \wedge$
$(a \in defer\ m\ A\ p \longrightarrow a \in defer\ m\ A\ q)$

definition *prof-leq-result* :: *'a Electoral-Module* $\Rightarrow$ *'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$
'a Profile $\Rightarrow$ *'a* $\Rightarrow$ *bool* **where**
prof-leq-result m A p q a $\equiv$
electoral-module m $\wedge$ *finite-profile A p* $\wedge$ *finite-profile A q* $\wedge$ $a \in A$ $\wedge$
$(a \in reject\ m\ A\ p \longrightarrow a \in reject\ m\ A\ q) \wedge$
$(a \in defer\ m\ A\ p \longrightarrow a \notin elect\ m\ A\ q)$

definition *prof-geq-result* :: *'a Electoral-Module* $\Rightarrow$ *'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$
'a Profile $\Rightarrow$ *'a* $\Rightarrow$ *bool* **where**
prof-geq-result m A p q a $\equiv$
electoral-module m $\wedge$ *finite-profile A p* $\wedge$ *finite-profile A q* $\wedge$ $a \in A$ $\wedge$
$(a \in elect\ m\ A\ p \longrightarrow a \in elect\ m\ A\ q) \wedge$
$(a \in defer\ m\ A\ p \longrightarrow a \notin reject\ m\ A\ q)$

definition *mod-contains-result* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* $\Rightarrow$
'a set $\Rightarrow$ *'a Profile* $\Rightarrow$ *'a* $\Rightarrow$ *bool* **where**
mod-contains-result m n A p a $\equiv$

electoral-module m $\wedge$ *electoral-module* n $\wedge$ *finite-profile* A p $\wedge$ $a \in A$ $\wedge$
($a \in$ *elect* m A $p \longrightarrow a \in$ *elect* n A p) $\wedge$
($a \in$ *reject* m A $p \longrightarrow a \in$ *reject* n A p) $\wedge$
($a \in$ *defer* m A $p \longrightarrow a \in$ *defer* n A p)

2.1.4 Auxiliary Lemmas

lemma *combine-ele-rej-def*:
fixes
m :: $'a$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
e :: $'a$ *set* **and**
r :: $'a$ *set* **and**
d :: $'a$ *set*
assumes
elect m A p = e **and**
reject m A p = r **and**
defer m A p = d
shows m A p = (e, r, d)
using *assms*
by *auto*

lemma *par-comp-result-sound*:
fixes
m :: $'a$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
assumes
electoral-module m **and**
finite-profile A p
shows *well-formed* A (m A p)
using *assms*
unfolding *electoral-module-def*
by *simp*

lemma *result-presv-alts*:
fixes
m :: $'a$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
assumes
e-mod: *electoral-module* m **and**
f-prof: *finite-profile* A p
shows (*elect* m A p) $\cup$ (*reject* m A p) $\cup$ (*defer* m A p) = A
proof (*safe*)
fix a :: $'a$
assume *elec-a*: $a \in$ *elect* m A p
have *partit*: $\forall$ p. *set-equals-partition* A $p \longrightarrow$ ($\exists$ E R D. p = (E, R, D) $\wedge$ E $\cup$

```
R ∪ D = A)
    by simp
  have set-partit: set-equals-partition A (m A p)
    using e-mod f-prof
    unfolding electoral-module-def
    by simp
  thus a ∈ A
    using UnI1 elec-a fstI partit
    by (metis (no-types))
next
  fix a :: 'a
  assume rej-a: a ∈ reject m A p
  have partit: ∀ p. set-equals-partition A p ⟶ (∃ E R D. p = (E, R, D) ∧ E ∪
R ∪ D = A)
    by simp
  have set-equals-partition A (m A p)
    using e-mod f-prof
    unfolding electoral-module-def
    by simp
  thus a ∈ A
    using UnI1 rej-a fstI partit
          sndI subsetD sup-ge2
    by metis
next
  fix a :: 'a
  assume def-a: a ∈ defer m A p
  have partit: ∀ p. set-equals-partition A p ⟶ (∃ E R D. p = (E, R, D) ∧ E ∪
R ∪ D = A)
    by simp
  have set-equals-partition A (m A p)
    using e-mod f-prof
    unfolding electoral-module-def
    by simp
  thus a ∈ A
    using def-a partit sndI subsetD sup-ge2
    by metis
next
  fix a :: 'a
  assume
    a-in-A: a ∈ A and
    not-def-a: a ∉ defer m A p and
    not-rej-a: a ∉ reject m A p
  have partit: ∀ p. set-equals-partition A p ⟶ (∃ E R D. p = (E, R, D) ∧ E ∪
R ∪ D = A)
    by simp
  from e-mod f-prof
  have set-equals-partition A (m A p)
    unfolding electoral-module-def
    by simp
```

thus $a \in$ *elect m A p*
using *a-in-A not-def-a not-rej-a fst-conv partit snd-conv Un-iff*
by *metis*
qed

lemma *result-disj*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module: *electoral-module m* **and**
profile: *finite-profile A p*
shows
(*elect m A p*) $\cap$ (*reject m A p*) = {} $\wedge$
(*elect m A p*) $\cap$ (*defer m A p*) = {} $\wedge$
(*reject m A p*) $\cap$ (*defer m A p*) = {}
proof (*safe*, *simp-all*)
fix *a* :: *'a*
assume
elect-a: $a \in$ *elect m A p* **and**
reject-a: $a \in$ *reject m A p*
have *well-formed A* (*m A p*)
using *module profile*
unfolding *electoral-module-def*
by *metis*
thus *False*
using *prod.exhaust-sel DiffE UnCI elect-a reject-a result-imp-rej*
by (*metis* (*no-types*))
next
fix *a* :: *'a*
assume
elect-a: $a \in$ *elect m A p* **and**
defer-a: $a \in$ *defer m A p*
have *disj*:
$\forall$ *p*. *disjoint3 p* $\longrightarrow$ ($\exists$ *B C D*. $p = (B, C, D) \wedge B \cap C = \{\} \wedge B \cap D = \{\}$
$\wedge\ C \cap D = \{\}$)
by *simp*
have *well-formed A* (*m A p*)
using *module profile*
unfolding *electoral-module-def*
by *metis*
hence *disjoint3* (*m A p*)
by *simp*
then obtain
elec :: *'a Result* $\Rightarrow$ *'a set* **and**
rej :: *'a Result* $\Rightarrow$ *'a set* **and**
def :: *'a Result* $\Rightarrow$ *'a set*
where

$m\ A\ p =$
  $(elec\ (m\ A\ p),\ rej\ (m\ A\ p),\ def\ (m\ A\ p)) \wedge$
    $elec\ (m\ A\ p) \cap rej\ (m\ A\ p) = \{\} \wedge$
    $elec\ (m\ A\ p) \cap def\ (m\ A\ p) = \{\} \wedge$
    $rej\ (m\ A\ p) \cap def\ (m\ A\ p) = \{\}$
  using *elect-a defer-a disj*
  by *metis*
hence $((elect\ m\ A\ p) \cap (reject\ m\ A\ p) = \{\}) \wedge$
    $((elect\ m\ A\ p) \cap (defer\ m\ A\ p) = \{\}) \wedge$
    $((reject\ m\ A\ p) \cap (defer\ m\ A\ p) = \{\})$
  using *eq-snd-iff fstI*
  by *metis*
thus *False*
  using *elect-a defer-a disjoint-iff-not-equal*
  by (*metis* (*no-types*))
next
fix a :: $'a$
assume
  reject-a: $a \in reject\ m\ A\ p$ **and**
  defer-a: $a \in defer\ m\ A\ p$
have *well-formed* $A\ (m\ A\ p)$
  using *module profile*
  unfolding *electoral-module-def*
  by *simp*
thus *False*
  using *prod.exhaust-sel DiffE UnCI reject-a defer-a result-imp-rej*
  by (*metis* (*no-types*))
qed

lemma *elect-in-alts*:
fixes
  m :: $'a$ *Electoral-Module* **and**
  A :: $'a$ *set* **and**
  p :: $'a$ *Profile*
assumes
  electoral-module m **and**
  finite-profile $A\ p$
shows *elect* $m\ A\ p \subseteq A$
using *le-supI1 assms result-presv-alts sup-ge1*
by *metis*

lemma *reject-in-alts*:
fixes
  m :: $'a$ *Electoral-Module* **and**
  A :: $'a$ *set* **and**
  p :: $'a$ *Profile*
assumes
  electoral-module m **and**
  finite-profile $A\ p$

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

lemma *defer-in-alts*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
electoral-module m **and**
finite-profile A p
shows *defer m A p* $\subseteq$ *A*
using *assms result-presv-alts*
by *auto*

lemma *def-presv-fin-prof*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
electoral-module m **and**
finite-profile A p
shows *let new-A = defer m A p in finite-profile new-A (limit-profile new-A p)*
using *defer-in-alts infinite-super limit-profile-sound assms*
by *metis*

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

lemma *upper-card-bounds-for-result*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
electoral-module m **and**
finite-profile A p
shows
card (elect m A p) $\leq$ *card A* $\wedge$
card (reject m A p) $\leq$ *card A* $\wedge$
card (defer m A p) $\leq$ *card A*
using *assms*
by *(simp add: card-mono defer-in-alts elect-in-alts reject-in-alts)*

lemma *reject-not-elec-or-def*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**

p :: *'a Profile*
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *reject m A p* = *A* − (*elect m A p*) − (*defer m A p*)
proof −
have *well-formed A* (*m A p*)
using *e-mod f-prof*
unfolding *electoral-module-def*
by *simp*
hence (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
using *e-mod f-prof result-presv-alts*
by *simp*
moreover have
(*elect m A p*) ∩ (*reject m A p*) = {} ∧
(*reject m A p*) ∩ (*defer m A p*) = {}
using *e-mod f-prof result-disj*
by *blast*
ultimately show *?thesis*
by *blast*
qed

lemma *elec-and-def-not-rej*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *elect m A p* ∪ *defer m A p* = *A* − (*reject m A p*)
proof −
from *e-mod f-prof*
have *well-formed A* (*m A p*)
unfolding *electoral-module-def*
by *simp*
hence
disjoint3 (*m A p*) ∧ *set-equals-partition A* (*m A p*)
by *simp*
have (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
using *e-mod f-prof result-presv-alts*
by *blast*
moreover have
(*elect m A p*) ∩ (*reject m A p*) = {} ∧
(*reject m A p*) ∩ (*defer m A p*) = {}
using *e-mod f-prof result-disj*
by *blast*
ultimately show *?thesis*
by *blast*

qed

lemma *defer-not-elec-or-rej*:
  fixes
    m :: $'a$ *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile*
  assumes
    e-mod: *electoral-module* m **and**
    f-prof: *finite-profile* A p
  shows $\mathit{defer}\ m\ A\ p = A - (\mathit{elect}\ m\ A\ p) - (\mathit{reject}\ m\ A\ p)$
proof −
  from *e-mod f-prof*
  have *well-formed* A $(m\ A\ p)$
    unfolding *electoral-module-def*
    by *simp*
  hence $(\mathit{elect}\ m\ A\ p) \cup (\mathit{reject}\ m\ A\ p) \cup (\mathit{defer}\ m\ A\ p) = A$
    using *e-mod f-prof result-presv-alts*
    by *simp*
  moreover have
    $(\mathit{elect}\ m\ A\ p) \cap (\mathit{defer}\ m\ A\ p) = \{\} \wedge$
      $(\mathit{reject}\ m\ A\ p) \cap (\mathit{defer}\ m\ A\ p) = \{\}$
    using *e-mod f-prof result-disj*
    by *blast*
  ultimately show *?thesis*
    by *blast*
qed

lemma *electoral-mod-defer-elem*:
  fixes
    m :: $'a$ *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    a :: $'a$
  assumes
    electoral-module m **and**
    finite-profile A p **and**
    $a \in A$ **and**
    $a \notin \mathit{elect}\ m\ A\ p$ **and**
    $a \notin \mathit{reject}\ m\ A\ p$
  shows $a \in \mathit{defer}\ m\ A\ p$
  using *DiffI assms reject-not-elec-or-def*
  by *metis*

lemma *mod-contains-result-comm*:
  fixes
    m :: $'a$ *Electoral-Module* **and**
    n :: $'a$ *Electoral-Module* **and**
    A :: $'a$ *set* **and**

p :: *'a Profile* **and**
a :: *'a*
assumes *mod-contains-result m n A p a*
shows *mod-contains-result n m A p a*
proof (*unfold mod-contains-result-def*, *safe*)
from *assms*
show *electoral-module n*
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show *electoral-module m*
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show *finite A*
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show *profile A p*
unfolding *mod-contains-result-def*
by *safe*
next
from *assms*
show $a \in A$
unfolding *mod-contains-result-def*
by *safe*
next
assume $a \in$ *elect n A p*
thus $a \in$ *elect m A p*
using *IntI assms electoral-mod-defer-elem empty-iff mod-contains-result-def result-disj*
by (*metis* (*mono-tags*, *lifting*))
next
assume $a \in$ *reject n A p*
thus $a \in$ *reject m A p*
using *IntI assms electoral-mod-defer-elem empty-iff mod-contains-result-def result-disj*
by (*metis* (*mono-tags*, *lifting*))
next
assume $a \in$ *defer n A p*
thus $a \in$ *defer m A p*
using *IntI assms electoral-mod-defer-elem empty-iff mod-contains-result-def result-disj*
by (*metis* (*mono-tags*, *lifting*))
qed

lemma *not-rej-imp-elec-or-def*:
fixes
m :: *′a Electoral-Module* **and**
A :: *′a set* **and**
p :: *′a Profile* **and**
a :: *′a*
assumes
electoral-module m **and**
finite-profile A p **and**
$a \in A$ **and**
$a \notin$ *reject m A p*
shows $a \in$ *elect m A p* $\vee$ $a \in$ *defer m A p*
using *assms electoral-mod-defer-elem*
by *metis*

lemma *single-elim-imp-red-def-set*:
fixes
m :: *′a Electoral-Module* **and**
A :: *′a set* **and**
p :: *′a Profile*
assumes
eliminates 1 m **and**
card A > *1* **and**
finite-profile A p
shows *defer m A p* $\subset$ *A*
using *Diff-eq-empty-iff Diff-subset card-eq-0-iff defer-in-alts eliminates-def eq-iff not-one-le-zero psubsetI reject-not-elec-or-def assms*
by *metis*

lemma *eq-alts-in-profs-imp-eq-results*:
fixes
m :: *′a Electoral-Module* **and**
A :: *′a set* **and**
p :: *′a Profile* **and**
q :: *′a Profile*
assumes
eq: $\forall$ $a \in A$. *prof-contains-result m A p q a* **and**
mod-m: *electoral-module m* **and**
fin-prof-p: *finite-profile A p* **and**
fin-prof-q: *finite-profile A q*
shows *m A p* = *m A q*
proof −
have *elected-in-A*: *elect m A q* $\subseteq$ *A*
using *elect-in-alts mod-m fin-prof-q*
by *metis*
have *rejected-in-A*: *reject m A q* $\subseteq$ *A*
using *reject-in-alts mod-m fin-prof-q*
by *metis*
have *deferred-in-A*: *defer m A q* $\subseteq$ *A*

using *defer-in-alts mod-m fin-prof-q*
by *metis*
have $\forall\ a \in$ *elect m A p*. $a \in$ *elect m A q*
using *elect-in-alts eq prof-contains-result-def mod-m fin-prof-p in-mono*
by *metis*
moreover have $\forall\ a \in$ *elect m A q*. $a \in$ *elect m A p*
proof
fix $a :: 'a$
assume *q-elect-a*: $a \in$ *elect m A q*
hence *a-in-A*: $a \in A$
using *elected-in-A*
by *blast*
have *a-not-deferred-q*: $a \notin$ *defer m A q*
using *q-elect-a fin-prof-q mod-m result-disj*
by *blast*
have *a-not-rejected-q*: $a \notin$ *reject m A q*
using *disjoint-iff-not-equal fin-prof-q mod-m q-elect-a result-disj*
by *metis*
show $a \in$ *elect m A p*
using *a-in-A electoral-mod-defer-elem eq a-not-deferred-q a-not-rejected-q prof-contains-result-def*
by *metis*
qed
moreover have $\forall\ a \in$ *reject m A p*. $a \in$ *reject m A q*
using *reject-in-alts eq prof-contains-result-def mod-m fin-prof-p*
by *fastforce*
moreover have $\forall\ a \in$ *reject m A q*. $a \in$ *reject m A p*
proof
fix $a :: 'a$
assume *q-rejects-a*: $a \in$ *reject m A q*
hence *a-in-A*: $a \in A$
using *rejected-in-A*
by *blast*
have *a-not-deferred-q*: $a \notin$ *defer m A q*
using *q-rejects-a fin-prof-q mod-m result-disj*
by *blast*
have *a-not-elected-q*: $a \notin$ *elect m A q*
using *disjoint-iff-not-equal fin-prof-q mod-m q-rejects-a result-disj*
by *metis*
show $a \in$ *reject m A p*
using *a-in-A electoral-mod-defer-elem eq a-not-deferred-q a-not-elected-q prof-contains-result-def*
by *metis*
qed
moreover have $\forall\ a \in$ *defer m A p*. $a \in$ *defer m A q*
using *defer-in-alts eq prof-contains-result-def mod-m fin-prof-p*
by *fastforce*
moreover have $\forall\ a \in$ *defer m A q*. $a \in$ *defer m A p*
proof

```
    fix a :: 'a
    assume q-defers-a: a ∈ defer m A q
    hence a-in-A: a ∈ A
      using deferred-in-A
      by blast
    have a-not-elected-q: a ∉ elect m A q
      using q-defers-a fin-prof-q mod-m result-disj
      by blast
    have a-not-rejected-q: a ∉ reject m A q
      using disjoint-iff-not-equal fin-prof-q mod-m q-defers-a result-disj
      by metis
    show a ∈ defer m A p
      using a-in-A electoral-mod-defer-elem eq a-not-elected-q a-not-rejected-q
            prof-contains-result-def
      by metis
  qed
  ultimately show ?thesis
    using prod.collapse subsetI subset-antisym
    by (metis (no-types))
qed

lemma eq-def-and-elect-imp-eq:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module and
    A :: 'a set and
    p :: 'a Profile and
    q :: 'a Profile
  assumes
    mod-m: electoral-module m and
    mod-n: electoral-module n and
    fin-p: finite-profile A p and
    fin-q: finite-profile A q and
    elec-eq: elect m A p = elect n A q and
    def-eq: defer m A p = defer n A q
  shows m A p = n A q
proof -
  have reject m A p = A - ((elect m A p) ∪ (defer m A p))
    using mod-m fin-p combine-ele-rej-def result-imp-rej
    unfolding electoral-module-def
    by metis
  moreover have reject n A q = A - ((elect n A q) ∪ (defer n A q))
    using mod-n fin-q combine-ele-rej-def result-imp-rej
    unfolding electoral-module-def
    by metis
  ultimately show ?thesis
    using elec-eq def-eq prod-eqI
    by metis
qed
```

2.1.5 Non-Blocking

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

definition *non-blocking* :: *'a Electoral-Module* ⇒ *bool* **where**
  non-blocking m ≡
    electoral-module m ∧
      (∀ *A p*.
        ((*A* ≠ {} ∧ *finite-profile A p*) ⟶ *reject m A p* ≠ *A*))

2.1.6 Electing

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

definition *electing* :: *'a Electoral-Module* ⇒ *bool* **where**
  electing m ≡
    electoral-module m ∧
      (∀ *A p*. (*A* ≠ {} ∧ *finite-profile A p*) ⟶ *elect m A p* ≠ {})

lemma *electing-for-only-alt*:
  fixes
    m :: *'a Electoral-Module* **and**
    A :: *'a set* **and**
    p :: *'a Profile*
  assumes
    one-alt: *card A* = *1* **and**
    electing: *electing m* **and**
    f-prof: *finite-profile A p*
  shows *elect m A p* = *A*
proof (*safe*)
  fix *a* :: *'a*
  assume *elect-a*: *a* ∈ *elect m A p*
  have *electoral-module m* ⟶ *elect m A p* ⊆ *A*
    using *elect-in-alts f-prof*
    by (*simp add*: *elect-in-alts*)
  hence *elect m A p* ⊆ *A*
    using *electing*
    unfolding *electing-def*
    by *metis*
  thus *a* ∈ *A*
    using *elect-a*
    by *blast*
next
  fix *a* :: *'a*
  assume *a* ∈ *A*
  with *electing*
  show *a* ∈ *elect m A p*
    unfolding *electing-def*
    using *f-prof one-alt One-nat-def Suc-leI card-seteq*

card-gt-0-iff elect-in-alts infinite-super
by *metis*
qed

theorem *electing-imp-non-blocking*:
fixes *m* :: *'a Electoral-Module*
assumes *electing m*
shows *non-blocking m*
proof (*unfold non-blocking-def*, *safe*)
from *assms*
show *electoral-module m*
unfolding *electing-def*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a*
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
rej-A: *reject m A p* = *A* **and**
a-in-A: $a \in A$
have *electoral-module m* $\wedge$ ($\forall$ *A q*. $A \neq \{\}$ $\wedge$ *finite A* $\wedge$ *profile A q* $\longrightarrow$ *elect m A q* $\neq \{\}$)
using *assms*
unfolding *electing-def*
by *metis*
thus $a \in \{\}$
using *Diff-cancel Un-empty elec-and-def-not-rej fin-A prof-A rej-A a-in-A*
by (*metis* (*no-types*))
qed

2.1.7 Properties

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

definition *non-electing* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**
non-electing m $\equiv$
electoral-module m $\wedge$ ($\forall$ *A p*. *finite-profile A p* $\longrightarrow$ *elect m A p* $= \{\}$)

lemma *single-elim-decr-def-card*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
rejecting: *rejects 1 m* **and**
not-empty: $A \neq \{\}$ **and**
non-electing: *non-electing m* **and**

f-prof: *finite-profile A p*
shows *card* (*defer m A p*) = *card A* − *1*
proof −
have *no-elect*: *electoral-module m* ∧ (∀ *A q*. *finite A* ∧ *profile A q* ⟶ *elect m A q* = {})
using *non-electing-def f-prof not-empty non-electing*
by (*metis* (*no-types*))
have *rejected-in-A*: *reject m A p* ⊆ *A*
using *no-elect f-prof reject-in-alts*
by *metis*
have *A* = *A* − *elect m A p*
using *no-elect f-prof*
by *blast*
thus *?thesis*
using *f-prof rejected-in-A rejecting not-empty*
by (*simp add*: *Suc-leI card-Diff-subset card-gt-0-iff*
defer-not-elec-or-rej finite-subset
rejects-def)
qed

lemma *single-elim-decr-def-card-2*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
eliminating: *eliminates 1 m* **and**
not-empty: *card A* > *1* **and**
non-electing: *non-electing m* **and**
f-prof: *finite-profile A p*
shows *card* (*defer m A p*) = *card A* − *1*
proof −
have *no-elect*: *electoral-module m* ∧ (∀ *A q*. *finite A* ∧ *profile A q* ⟶ *elect m A q* = {})
using *non-electing-def f-prof not-empty non-electing*
by (*metis* (*no-types*))
have *rejected-in-A*: *reject m A p* ⊆ *A*
using *no-elect f-prof reject-in-alts*
by *metis*
have *A* = *A* − *elect m A p*
using *no-elect f-prof*
by *blast*
thus *?thesis*
using *f-prof rejected-in-A eliminating not-empty*
by (*simp add*: *card-Diff-subset defer-not-elec-or-rej eliminates-def finite-subset*)
qed

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

definition *defer-deciding* :: *'a Electoral-Module* ⇒ *bool* **where**
  defer-deciding m ≡
    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 Electoral-Module* ⇒ *bool* **where**
  decrementing m ≡
    electoral-module m ∧ (
      ∀ *A p*. *finite-profile A p* ⟶
        (*card A* > *1* ⟶ *card* (*reject m A p*) ≥ *1*))

definition *defer-condorcet-consistency* :: *'a Electoral-Module* ⇒ *bool* **where**
  defer-condorcet-consistency m ≡
    electoral-module m ∧
    (∀ *A p a*. *condorcet-winner A p a* ∧ *finite A* ⟶
      (*m A p* =
        ({},
        A − (*defer m A p*),
        {*d* ∈ *A*. *condorcet-winner A p d*})))

definition *condorcet-compatibility* :: *'a Electoral-Module* ⇒ *bool* **where**
  condorcet-compatibility m ≡
    electoral-module m ∧
    (∀ *A p a*. *condorcet-winner A p a* ∧ *finite A* ⟶
      (*a* ∉ *reject m A p* ∧
        (∀ *b*. ¬*condorcet-winner A p b* ⟶ *b* ∉ *elect m A p*) ∧
          (*a* ∈ *elect m A p* ⟶
            (∀ *b*. ¬*condorcet-winner A p b* ⟶ *b* ∈ *reject m A p*))))

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

definition *defer-monotonicity* :: *'a Electoral-Module* ⇒ *bool* **where**
  defer-monotonicity m ≡
    electoral-module m ∧
      (∀ *A p q a*.
        (*finite A* ∧ *a* ∈ *defer m A p* ∧ *lifted A p q a*) ⟶ *a* ∈ *defer m A q*)

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

definition *defer-lift-invariance* :: *'a Electoral-Module* ⇒ *bool* **where**
  defer-lift-invariance m ≡
    electoral-module m ∧
      (∀ *A p q a*.
        (*a* ∈ (*defer m A p*) ∧ *lifted A p q a*) ⟶ *m A p* = *m A q*)

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 Electoral-Module* ⇒ *'a Electoral-Module* ⇒ *bool* **where**
disjoint-compatibility m n ≡
electoral-module m ∧ *electoral-module n* ∧
(∀ *A*. *finite A* ⟶
(∃ *B* ⊆ *A*.
(∀ *a* ∈ *B*. *indep-of-alt m A a* ∧
(∀ *p*. *finite-profile A p* ⟶ *a* ∈ *reject m A p*)) ∧
(∀ *a* ∈ *A* − *B*. *indep-of-alt n A a* ∧
(∀ *p*. *finite-profile A p* ⟶ *a* ∈ *reject n 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 Electoral-Module* ⇒ *bool* **where**
invariant-monotonicity m ≡
electoral-module m ∧
(∀ *A p q a*. (*a* ∈ *elect m A p* ∧ *lifted A p q a*) ⟶
(*elect m A q* = *elect m A p* ∨ *elect m 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 Electoral-Module* ⇒ *bool* **where**
defer-invariant-monotonicity m ≡
electoral-module m ∧ *non-electing m* ∧
(∀ *A p q a*. (*a* ∈ *defer m A p* ∧ *lifted A p q a*) ⟶
(*defer m A q* = *defer m A p* ∨ *defer m A q* = {*a*}))

2.1.8 Inference Rules

lemma *ccomp-and-dd-imp-def-only-winner*:
fixes
m :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a*
assumes
ccomp: *condorcet-compatibility m* **and**
dd: *defer-deciding m* **and**
winner: *condorcet-winner A p a*
shows *defer m A p* = {*a*}
proof (*rule ccontr*)
assume *not-w*: *defer m A p* ≠ {*a*}
from *dd*
have *def-1*:
defers 1 m

unfolding *defer-deciding-def*
by *metis*
hence *c-win*:
finite-profile A p ∧ *a* ∈ *A* ∧ (∀ *b* ∈ *A* − {*a*}. *wins a p b*)
using *winner*
by *simp*
hence *card* (*defer m A p*) = *1*
using *Suc-leI card-gt-0-iff def-1 equals0D*
unfolding *One-nat-def defers-def*
by *metis*
hence *0*: ∃ *b* ∈ *A*. *defer m A p* = {*b*}
using *card-1-singletonE dd defer-in-alts insert-subset c-win*
unfolding *defer-deciding-def*
by *metis*
with *not-w*
have ∃ *b* ∈ *A*. *b* ≠ *a* ∧ *defer m A p* = {*b*}
by *metis*
hence *not-in-defer*: *a* ∉ *defer m A p*
by *auto*
have *non-electing m*
using *dd*
unfolding *defer-deciding-def*
by *simp*
hence *not-in-elect*: *a* ∉ *elect m A p*
using *c-win equals0D*
unfolding *non-electing-def*
by *simp*
from *not-in-defer not-in-elect*
have *one-side*:
a ∈ *reject m A p*
using *ccomp c-win electoral-mod-defer-elem*
unfolding *condorcet-compatibility-def*
by *metis*
from *ccomp*
have *other-side*: *a* ∉ *reject m A p*
using *c-win winner*
unfolding *condorcet-compatibility-def*
by *simp*
thus *False*
by (*simp add*: *one-side*)
qed

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

```
  from dd
  show electoral-module m
    unfolding defer-deciding-def
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assume
    prof-A: profile A p and
    a-in-A: a ∈ A and
    finiteness: finite A and
    c-winner: ∀ b ∈ A − {a}.
                card {i. i < length p ∧ (a, b) ∈ (p!i)} <
                  card {i. i < length p ∧ (b, a) ∈ (p!i)}
  hence winner: condorcet-winner A p a
    by simp
  hence
    m A p =
      ({},
        A − defer m A p,
        {c ∈ A. condorcet-winner A p c})
  proof −
    from dd
    have 0: elect m A p = {}
      using winner
      unfolding defer-deciding-def non-electing-def
      by simp
    from dd ccomp
    have 1: defer m A p = {a}
      using ccomp-and-dd-imp-def-only-winner winner
      by simp
    from 0 1
    have 2: reject m A p = A − defer m A p
      using Diff-empty dd reject-not-elec-or-def winner
      unfolding defer-deciding-def
      by fastforce
    from 0 1 2
    have 3: m A p = ({}, A − defer m A p, {a})
      using combine-ele-rej-def
      by metis
    have {a} = {c ∈ A. condorcet-winner A p c}
      using cond-winner-unique-3 winner
      by metis
    thus ?thesis
      using 3
      by simp
  qed
```

```
  hence
    m A p =
      ({},
        A − defer m A p,
        {c ∈ A. ∀ b ∈ A − {c}. wins c p b})
    using finiteness prof-A winner Collect-cong
    by simp
  hence
    m A p =
        ({},
          A − defer m A p,
          {c ∈ A. ∀ b ∈ A − {c}. prefer-count p b c < prefer-count p c b})
    by simp
  hence
    m A p =
        ({},
          A − defer m A p,
          {c ∈ A. ∀ b ∈ A − {c}.
            card {i. i < length p ∧ (let r = (p!i) in (c ≼r b))} <
                card {i. i < length p ∧ (let r = (p!i) in (b ≼r c))}})
    by simp
  thus
    m A p =
        ({},
          A − defer m A p,
          {c ∈ A. ∀ b ∈ A − {c}.
            card {i. i < length p ∧ (c, b) ∈ (p!i)} <
              card {i. i < length p ∧ (b, c) ∈ (p!i)}})
    by simp
qed
```

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

```
theorem disj-compat-comm[simp]:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module
  assumes disjoint-compatibility m n
  shows disjoint-compatibility n m
proof (unfold disjoint-compatibility-def, safe)
  show electoral-module m
    using assms
    unfolding disjoint-compatibility-def
    by simp
next
  show electoral-module n
    using assms
    unfolding disjoint-compatibility-def
    by simp
next
```

fix A :: *'a set*
assume *fin-S*: *finite A*
obtain B **where**
old-A:
$(B \subseteq A \wedge$
$(\forall\ a \in B.$ *indep-of-alt m A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject m A p*$)) \wedge$
$(\forall\ a \in A - B.$ *indep-of-alt n A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject n A p*$)))$
using *assms fin-S*
unfolding *disjoint-compatibility-def*
by *metis*
hence
$(\exists\ B \subseteq A.$
$(\forall\ a \in A - B.$ *indep-of-alt n A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject n A p*$)) \wedge$
$(\forall\ a \in B.$ *indep-of-alt m A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject m A p*$)))$
by *auto*
hence
$(\exists\ B \subseteq A.$
$(\forall\ a \in A - B.$ *indep-of-alt n A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject n A p*$)) \wedge$
$(\forall\ a \in A - (A - B).$ *indep-of-alt m A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject m A p*$)))$
using *double-diff order-refl*
by *metis*
thus
$(\exists\ B \subseteq A.$
$(\forall\ a \in B.$ *indep-of-alt n A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject n A p*$)) \wedge$
$(\forall\ a \in A - B.$ *indep-of-alt m A a* $\wedge$
$(\forall\ p.$ *finite-profile A p* $\longrightarrow a \in$ *reject m A p*$)))$
by *fastforce*
qed

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

theorem *dl-inv-imp-def-mono*[*simp*]:
fixes m :: *'a Electoral-Module*
assumes *defer-lift-invariance m*
shows *defer-monotonicity m*
using *assms*
unfolding *defer-monotonicity-def defer-lift-invariance-def*
by *metis*

2.1.9 Social Choice Properties

Condorcet Consistency

definition *condorcet-consistency* :: *'a Electoral-Module* $\Rightarrow$ *bool* **where**

condorcet-consistency m ≡
  electoral-module m ∧
  (∀ *A p a*. *condorcet-winner A p a* ⟶
    (*m A p* =
      ({*e* ∈ *A*. *condorcet-winner A p e*},
        A − (*elect m A p*),
        {})))

lemma *condorcet-consistency2*:
  fixes *m* :: *'a Electoral-Module*
  shows *condorcet-consistency m* =
      (*electoral-module m* ∧
        (∀ *A p a*. *condorcet-winner A p a* ⟶
          (*m A p* =
            ({*a*}, *A* − (*elect m A p*), {}))))
proof (*safe*)
  assume *condorcet-consistency m*
  thus *electoral-module m*
    unfolding *condorcet-consistency-def*
    by *metis*
next
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    a :: *'a*
  assume
    condorcet-consistency m **and**
    condorcet-winner A p a
  thus
    m A p = ({*a*}, *A* − *elect m A p*, {})
    using *cond-winner-unique-3*
    unfolding *condorcet-consistency-def*
    by (*metis* (*mono-tags*, *lifting*))
next
  assume
    e-mod: *electoral-module m* **and**
    cwin:
    ∀ *A p a*. *condorcet-winner A p a* ⟶
      m A p = ({*a*}, *A* − *elect m A p*, {})
  have
    ∀ *f*. *condorcet-consistency f* =
      (*electoral-module f* ∧
        (∀ *A p a*. *condorcet-winner A p a* ⟶
          f A p = ({*a* ∈ *A*. *condorcet-winner A p a*},
                A − *elect f A p*, {})))
    unfolding *condorcet-consistency-def*
    by *blast*
  moreover have
    ∀ *A p a*. *condorcet-winner A p* (*a*::*'a*) ⟶

$\{b \in A.\ \textit{condorcet-winner}\ A\ p\ b\} = \{a\}$
using *cond-winner-unique-3*
by (*metis* (*full-types*))
ultimately show *condorcet-consistency m*
unfolding *condorcet-consistency-def*
using *cond-winner-unique-3 e-mod cwin*
by *presburger*
qed

(Weak) Monotonicity

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

definition *monotonicity* :: *'a Electoral-Module* ⇒ *bool* **where**
monotonicity m ≡
electoral-module m ∧
(∀ *A p q a*.
(*finite A* ∧ *a* ∈ *elect m A p* ∧ *lifted A p q a*) ⟶ *a* ∈ *elect m A q*)

Homogeneity

fun *times* :: *nat* ⇒ *'a list* ⇒ *'a list* **where**
times n l = *concat* (*replicate n l*)

definition *homogeneity* :: *'a Electoral-Module* ⇒ *bool* **where**
homogeneity m ≡
electoral-module m ∧
(∀ *A p n*.
(*finite-profile A p* ∧ *n* > *0* ⟶
(*m A p* = *m A* (*times n p*))))

end

2.2 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.

2.2.1 Definition

type-synonym $'a$ *Evaluation-Function* $=$ $'a \Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ *nat*

2.2.2 Property

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

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

2.2.3 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 :: $'a$ *Evaluation-Function* **and**
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    a :: $'a$
  assumes
    rating: *condorcet-rating e* **and**
    f-prof: *finite-profile A p* **and**
    winner: *condorcet-winner A p a*
  shows $e\ a\ A\ p = Max\ \{e\ b\ A\ p \mid b.\ b \in A\}$
proof $-$
  let $?set = \{e\ b\ A\ p \mid b.\ b \in A\}$ **and**
      $?eMax = Max\ \{e\ b\ A\ p \mid b.\ b \in A\}$ **and**
      $?eW = e\ a\ A\ p$
  from *f-prof*
  have *0*: *finite ?set*
    by *simp*
  have *1*: $?set \neq \{\}$
    using *condorcet-winner.simps winner*
    by *fastforce*
  have *2*: $?eW \in ?set$
    using *CollectI condorcet-winner.simps winner*
    by (*metis* (*mono-tags*, *lifting*))
  have *3*: $\forall$ $e \in ?set$. $e \leq ?eW$
  proof (*safe*)
    fix *b* :: $'a$
    assume *b-in-A*: $b \in A$
    have $\forall$ *n na*. $(n::nat) \neq na \lor n \leq na$
      by *simp*
    with *b-in-A*

show $e\ b\ A\ p \leq e\ a\ A\ p$
  using *less-imp-le rating winner*
  unfolding *condorcet-rating-def*
  by (*metis* (*no-types*))
qed
from *2 3*
have *4*: $?eW \in ?set \wedge (\forall\ a \in ?set.\ a \leq ?eW)$
  by *blast*
from *0 1 4 Max-eq-iff*
show *?thesis*
  by (*metis* (*no-types*, *lifting*))
qed

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

theorem *non-cond-winner-not-max-eval*:
fixes
  e :: $'a$ *Evaluation-Function* **and**
  A :: $'a$ *set* **and**
  p :: $'a$ *Profile* **and**
  a :: $'a$ **and**
  b :: $'a$
assumes
  rating: *condorcet-rating* e **and**
  f-prof: *finite-profile* $A\ p$ **and**
  winner: *condorcet-winner* $A\ p\ a$ **and**
  lin-A: $b \in A$ **and**
  loser: $a \neq b$
shows $e\ b\ A\ p < Max\ \{e\ c\ A\ p \mid c.\ c \in A\}$
proof −
have $e\ b\ A\ p < e\ a\ A\ p$
  using *lin-A loser rating winner*
  unfolding *condorcet-rating-def*
  by *metis*
also have $e\ a\ A\ p = Max\ \{e\ c\ A\ p \mid c.\ c \in A\}$
  using *cond-winner-imp-max-eval-val f-prof rating winner*
  by *fastforce*
finally show *?thesis*
  by *simp*
qed

end

2.3 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.

2.3.1 Definition

type-synonym $Threshold\text{-}Value = nat$

type-synonym $Threshold\text{-}Relation = nat \Rightarrow nat \Rightarrow bool$

type-synonym $'a\ Electoral\text{-}Set = 'a\ set \Rightarrow 'a\ Profile \Rightarrow 'a\ set$

fun $elimination\text{-}set :: 'a\ Evaluation\text{-}Function \Rightarrow Threshold\text{-}Value \Rightarrow$
$Threshold\text{-}Relation \Rightarrow 'a\ Electoral\text{-}Set$ **where**
$elimination\text{-}set\ e\ t\ r\ A\ p = \{a \in A\ .\ r\ (e\ a\ A\ p)\ t\ \}$

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

2.3.2 Common Eliminators

fun $less\text{-}eliminator :: 'a\ Evaluation\text{-}Function \Rightarrow Threshold\text{-}Value \Rightarrow$
$'a\ Electoral\text{-}Module$ **where**
$less\text{-}eliminator\ e\ t\ A\ p = elimination\text{-}module\ e\ t\ (<)\ A\ p$

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

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

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

fun *average* :: *'a Evaluation-Function* ⇒ *'a set* ⇒ *'a Profile* ⇒
Threshold-Value **where**
average e A p = ($\sum$ *x* ∈ *A*. *e x A p*) *div* (*card A*)

fun *less-average-eliminator* :: *'a Evaluation-Function* ⇒
'a Electoral-Module **where**
less-average-eliminator e A p = *less-eliminator e* (*average e A p*) *A p*

fun *leq-average-eliminator* :: *'a Evaluation-Function* ⇒
'a Electoral-Module **where**
leq-average-eliminator e A p = *leq-eliminator e* (*average e A p*) *A p*

2.3.3 Soundness

lemma *elim-mod-sound*[*simp*]:
fixes
e :: *'a Evaluation-Function* **and**
t :: *Threshold-Value* **and**
r :: *Threshold-Relation*
shows *electoral-module* (*elimination-module e t r*)
proof (*unfold electoral-module-def*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile*
have *set-equals-partition A* (*elimination-module e t r A p*)
by *auto*
thus *well-formed A* (*elimination-module e t r A p*)
by *simp*
qed

lemma *less-elim-sound*[*simp*]:
fixes
e :: *'a Evaluation-Function* **and**
t :: *Threshold-Value*
shows *electoral-module* (*less-eliminator e t*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p < t\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < t\} \cup A = A$
by *safe*
qed

lemma *leq-elim-sound*[*simp*]:
fixes
e :: *'a Evaluation-Function* **and**
t :: *Threshold-Value*

shows *electoral-module* (*leq-eliminator e t*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
show
$\{a \in A.\ e\ a\ A\ p \leq t\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p \leq t\} \cup A = A$
by *safe*
qed

lemma *max-elim-sound*[*simp*]:
fixes e :: $'a$ *Evaluation-Function*
shows *electoral-module* (*max-eliminator e*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
show
$\{a \in A.\ e\ a\ A\ p < \mathit{Max}\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < \mathit{Max}\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \cup A = A$
by *safe*
qed

lemma *min-elim-sound*[*simp*]:
fixes e :: $'a$ *Evaluation-Function*
shows *electoral-module* (*min-eliminator e*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
show
$\{a \in A.\ e\ a\ A\ p \leq \mathit{Min}\ \{e\ x\ A\ p\ |\ x.\ x \in A\}\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p \leq \mathit{Min}\ \{e\ x\ A\ p\ |\ x.\ x \in A\}\} \cup A = A$
by *safe*
qed

lemma *less-avg-elim-sound*[*simp*]:
fixes e :: $'a$ *Evaluation-Function*
shows *electoral-module* (*less-average-eliminator e*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
show
$\{a \in A.\ e\ a\ A\ p < (\sum\ x \in A.\ e\ x\ A\ p)\ \mathit{div}\ \mathit{card}\ A\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < (\sum\ x \in A.\ e\ x\ A\ p)\ \mathit{div}\ \mathit{card}\ A\} \cup A = A$
by *safe*
qed

lemma *leq-avg-elim-sound*[*simp*]:
  fixes e :: *'a Evaluation-Function*
  shows *electoral-module* (*leq-average-eliminator e*)
proof (*unfold electoral-module-def*, *safe*, *simp*)
  fix
    A :: *'a set* **and**
    p :: *'a Profile*
  show
    $\{a \in A.\ e\ a\ A\ p \leq (\sum\ x \in A.\ e\ x\ A\ p)\ div\ card\ A\} \neq A \longrightarrow$
      $\{a \in A.\ e\ a\ A\ p \leq (\sum\ x \in A.\ e\ x\ A\ p)\ div\ card\ A\} \cup A = A$
    by *safe*
qed

2.3.4 Non-Electing

lemma *elim-mod-non-electing*:
  fixes
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    e :: *'a Evaluation-Function* **and**
    t :: *Threshold-Value* **and**
    r :: *Threshold-Relation*
  assumes *profile*: *finite-profile A p*
  shows *non-electing* (*elimination-module e t r*)
  unfolding *non-electing-def*
  by *simp*

lemma *less-elim-non-electing*:
  fixes
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    e :: *'a Evaluation-Function* **and**
    t :: *Threshold-Value*
  assumes *profile*: *finite-profile A p*
  shows *non-electing* (*less-eliminator e t*)
  using *elim-mod-non-electing profile less-elim-sound*
  unfolding *non-electing-def*
  by *simp*

lemma *leq-elim-non-electing*:
  fixes
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    e :: *'a Evaluation-Function* **and**
    t :: *Threshold-Value*
  assumes *profile*: *finite-profile A p*
  shows *non-electing* (*leq-eliminator e t*)
  unfolding *non-electing-def*

```
  by simp

lemma max-elim-non-electing:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    e :: 'a Evaluation-Function
  assumes profile: finite-profile A p
  shows non-electing (max-eliminator e)
  unfolding non-electing-def
  by simp

lemma min-elim-non-electing:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    e :: 'a Evaluation-Function
  assumes profile: finite-profile A p
  shows non-electing (min-eliminator e)
  unfolding non-electing-def
  by simp

lemma less-avg-elim-non-electing:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    e :: 'a Evaluation-Function
  assumes profile: finite-profile A p
  shows non-electing (less-average-eliminator e)
proof (unfold non-electing-def, safe)
  show electoral-module (less-average-eliminator e)
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assume
    fin-A: finite A and
    prof-p: profile A p and
    elect-a: a ∈ elect (less-average-eliminator e) A p
  hence fin-prof: finite-profile A p
    by metis
  have non-electing (less-average-eliminator e)
    unfolding non-electing-def
    by simp
  hence {} = elect (less-average-eliminator e) A p
    using fin-prof
    unfolding non-electing-def
```

```
    by metis
  thus a ∈ {}
    using elect-a
    by metis
qed

lemma leq-avg-elim-non-electing:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    e :: 'a Evaluation-Function
  assumes profile: finite-profile A p
  shows non-electing (leq-average-eliminator e)
proof (unfold non-electing-def, safe)
  show electoral-module (leq-average-eliminator e)
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assume
    fin-A: finite A and
    prof-p: profile A p and
    elect-a: a ∈ elect (leq-average-eliminator e) A p
  have non-electing (leq-average-eliminator e)
    unfolding non-electing-def
    by simp
  hence {} = elect (leq-average-eliminator e) A p
    using fin-A prof-p
    unfolding non-electing-def
    by metis
  thus a ∈ {}
    using elect-a
    by metis
qed
```

2.3.5 Inference Rules

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

```
theorem cr-eval-imp-ccomp-max-elim[simp]:
  fixes
    A :: 'a set and
    p :: 'a Profile and
    e :: 'a Evaluation-Function
  assumes
    profile: finite-profile A p and
    rating: condorcet-rating e
```

```
  shows condorcet-compatibility (max-eliminator e)
proof (unfold condorcet-compatibility-def, safe)
  show electoral-module (max-eliminator e)
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assume
    c-win: condorcet-winner A p a and
    rej-a: a ∈ reject (max-eliminator e) A p
  have e a A p = Max {e b A p | b. b ∈ A}
    using c-win cond-winner-imp-max-eval-val rating
    by fastforce
  hence a ∉ reject (max-eliminator e) A p
    by simp
  thus False
    using rej-a
    by linarith
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a
  assume elect-a: a ∈ elect (max-eliminator e) A p
  have a ∉ elect (max-eliminator e) A p
    by simp
  thus False
    using elect-a
    by linarith
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    a :: 'a and
    a' :: 'a
  assume
    condorcet-winner A p a and
    a ∈ elect (max-eliminator e) A p
  thus a' ∈ reject (max-eliminator e) A p
    using profile rating condorcet-winner.elims(2)
          empty-iff max-elim-non-electing
    unfolding non-electing-def
    by metis
qed

lemma cr-eval-imp-dcc-max-elim-helper:
  fixes
```

A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
e :: $'a$ *Evaluation-Function* **and**
a :: $'a$
assumes
f-prof: *finite-profile* A p **and**
rating: *condorcet-rating* e **and**
winner: *condorcet-winner* A p a
shows *elimination-set* e (*Max* $\{e\ b\ A\ p \mid b.\ b \in A\}$) ($<$) A p $= A - \{a\}$
proof (*safe*, *simp-all*, *safe*)
assume $e\ a\ A\ p <$ *Max* $\{e\ b\ A\ p \mid b.\ b \in A\}$
thus *False*
using *cond-winner-imp-max-eval-val*
rating winner f-prof
by *fastforce*
next
fix a' :: $'a$
assume
$a' \in A$ **and**
$\neg\ e\ a'\ A\ p <$ *Max* $\{e\ b\ A\ p \mid b.\ b \in A\}$
thus $a' = a$
using *non-cond-winner-not-max-eval rating winner f-prof*
by (*metis* (*mono-tags*, *lifting*))
qed

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

theorem *cr-eval-imp-dcc-max-elim*[*simp*]:
fixes e :: $'a$ *Evaluation-Function*
assumes *rating*: *condorcet-rating* e
shows *defer-condorcet-consistency* (*max-eliminator* e)
proof (*unfold defer-condorcet-consistency-def*, *safe*, *simp*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
a :: $'a$
assume
winner: *condorcet-winner* A p a **and**
finite: *finite* A
hence *profile*: *finite-profile* A p
by *simp*
let *?trsh* = (*Max* $\{e\ b\ A\ p \mid b.\ b \in A\}$)
show
max-eliminator e A p =
($\{\}$,
$A -$ *defer* (*max-eliminator* e) A p,
$\{b \in A.$ *condorcet-winner* A p $b\}$)
proof (*cases elimination-set* e (*?trsh*) ($<$) A $p \neq A$)
case *True*

```
    from profile rating winner
    have 0: (elimination-set e ?trsh (<) A p) = A − {a}
      using cr-eval-imp-dcc-max-elim-helper
      by (metis (mono-tags, lifting))
    have
      max-eliminator e A p =
        ({},
          (elimination-set e ?trsh (<) A p),
          A − (elimination-set e ?trsh (<) A p))
      using True
      by simp
    also have ... = ({}, A − {a}, {a})
      using 0 winner
      by auto
    also have ... = ({},A − defer (max-eliminator e) A p, {a})
      using calculation
      by simp
    also have
      ... =
        ({},
          A − defer (max-eliminator e) A p,
          {b ∈ A. condorcet-winner A p b})
      using cond-winner-unique-3 winner Collect-cong
      by (metis (no-types, lifting))
    finally show ?thesis
      using finite winner
      by metis
  next
    case False
    have ?trsh = e a A p
      using rating winner
      by (simp add: cond-winner-imp-max-eval-val)
    thus ?thesis
      using winner False
      by auto
  qed
qed

end
```

2.4 Aggregator

```
theory Aggregator
  imports Social-Choice-Types/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.

2.4.1 Definition

type-synonym *'a Aggregator* = *'a set* ⇒ *'a Result* ⇒ *'a Result* ⇒ *'a Result*

definition *aggregator* :: *'a Aggregator* ⇒ *bool* **where**
  aggregator agg ≡
    ∀ *A e1 e2 d1 d2 r1 r2*.
      (*well-formed A* (*e1*, *r1*, *d1*) ∧ *well-formed A* (*e2*, *r2*, *d2*)) ⟶
      well-formed A (*agg A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*))

2.4.2 Properties

definition *agg-commutative* :: *'a Aggregator* ⇒ *bool* **where**
  agg-commutative agg ≡
    aggregator agg ∧ (∀ *A e1 e2 d1 d2 r1 r2*.
      agg A (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*) = *agg A* (*e2*, *r2*, *d2*) (*e1*, *r1*, *d1*))

definition *agg-conservative* :: *'a Aggregator* ⇒ *bool* **where**
  agg-conservative agg ≡
    aggregator agg ∧
    (∀ *A e1 e2 d1 d2 r1 r2*.
      ((*well-formed A* (*e1*, *r1*, *d1*) ∧ *well-formed A* (*e2*, *r2*, *d2*)) ⟶
        elect-r (*agg A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) ⊆ (*e1* ∪ *e2*) ∧
        reject-r (*agg A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) ⊆ (*r1* ∪ *r2*) ∧
        defer-r (*agg A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) ⊆ (*d1* ∪ *d2*)))

end

2.5 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.

2.5.1 Definition

fun *max-aggregator* :: *'a Aggregator* **where**
  max-aggregator A $(e1, r1, d1)$ $(e2, r2, d2)$ =
    $(e1 \cup e2,$
     $A - (e1 \cup e2 \cup d1 \cup d2),$
     $(d1 \cup d2) - (e1 \cup e2))$

2.5.2 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-1: *well-formed* A (e, r, d) **and**
    wf-2: *well-formed* A (e', r', d')
  shows *reject-r* (*max-aggregator* A (e, r, d) (e', r', d')) $= r \cap r'$
proof −
  have $A - (e \cup d) = r$
    using *wf-1*
    by (*simp add*: *result-imp-rej*)
  moreover have $A - (e' \cup d') = r'$
    using *wf-2*
    by (*simp add*: *result-imp-rej*)
  ultimately have $A - (e \cup e' \cup d \cup d') = r \cap r'$
    by *blast*
  moreover have $\{l \in A.\ l \notin e \cup e' \cup d \cup d'\} = A - (e \cup e' \cup d \cup d')$
    by (*simp add*: *set-diff-eq*)
  ultimately show *reject-r* (*max-aggregator* A (e, r, d) (e', r', d')) $= r \cap r'$
    by *simp*
qed

2.5.3 Soundness

theorem *max-agg-sound*[*simp*]: *aggregator max-aggregator*
proof (*unfold aggregator-def*, *simp*, *safe*)
  fix
    A :: *'a set* **and**
    e :: *'a set* **and**

e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**
r' :: $'a$ *set* **and**
a :: $'a$
assume
$e' \cup r' \cup d' = e \cup r \cup d$ **and**
$a \notin d$ **and**
$a \notin r$ **and**
$a \in e'$
thus $a \in e$
by *auto*
next
fix
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**
r' :: $'a$ *set* **and**
a :: $'a$
assume
$e' \cup r' \cup d' = e \cup r \cup d$ **and**
$a \notin d$ **and**
$a \notin r$ **and**
$a \in d'$
thus $a \in e$
by *auto*
qed

2.5.4 Properties

The max-aggregator is conservative.

theorem *max-agg-consv*[*simp*]: *agg-conservative max-aggregator*
proof (*unfold agg-conservative-def*, *safe*)
show *aggregator max-aggregator*
using *max-agg-sound*
by *metis*
next
fix
A :: $'a$ *set* **and**
e :: $'a$ *set* **and**
e' :: $'a$ *set* **and**
d :: $'a$ *set* **and**
d' :: $'a$ *set* **and**
r :: $'a$ *set* **and**
r' :: $'a$ *set* **and**

a :: $'a$

assume

elect-a: $a \in$ *elect-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**

a-not-in-e': $a \notin e'$

have $a \in e \cup e'$

using *elect-a*

by *simp*

hence $a \in e \cup e'$

by *metis*

thus $a \in e$

using *a-not-in-e'*

by *simp*

next

fix

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

e :: $'a$ *set* **and**

e' :: $'a$ *set* **and**

d :: $'a$ *set* **and**

d' :: $'a$ *set* **and**

r :: $'a$ *set* **and**

r' :: $'a$ *set* **and**

a :: $'a$

assume

wf-2: *well-formed* A (e', r', d') **and**

reject-a: $a \in$ *reject-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**

a-not-in-r': $a \notin r'$

have $a \in r \cup r'$

using *wf-2 reject-a*

by *force*

hence $a \in r \cup r'$

by *metis*

thus $a \in r$

using *a-not-in-r'*

by *simp*

next

fix

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

e :: $'a$ *set* **and**

e' :: $'a$ *set* **and**

d :: $'a$ *set* **and**

d' :: $'a$ *set* **and**

r :: $'a$ *set* **and**

r' :: $'a$ *set* **and**

a :: $'a$

assume

wf-2: *well-formed* A (e', r', d') **and**

defer-a: $a \in$ *defer-r* (*max-aggregator* A (e, r, d) (e', r', d')) **and**

a-not-in-d': $a \notin d'$

have $a \in d \cup d'$

using *wf-2 defer-a*
by *force*
hence $a \in d \cup d'$
by *metis*
thus $a \in d$
using *a-not-in-d'*
by *simp*
qed

The max-aggregator is commutative.

theorem *max-agg-comm*[*simp*]: *agg-commutative max-aggregator*
unfolding *agg-commutative-def*
by *auto*

end

2.6 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.

2.6.1 Definition

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

end

2.7 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.

2.7.1 Definition

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

end

Chapter 3

Basic Modules

3.1 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.

3.1.1 Definition

fun *defer-module* :: *'a Electoral-Module* **where**
  defer-module A p = ({}, {}, A)

3.1.2 Soundness

theorem *def-mod-sound*[*simp*]: *electoral-module defer-module*
  unfolding *electoral-module-def*
  by *simp*

3.1.3 Properties

theorem *def-mod-non-electing*: *non-electing defer-module*
  unfolding *non-electing-def*
  by *simp*

theorem *def-mod-def-lift-inv*: *defer-lift-invariance defer-module*
  unfolding *defer-lift-invariance-def*
  by *simp*

end

3.2 Drop Module

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

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

3.2.1 Definition

```
fun drop-module :: nat ⇒ 'a Preference-Relation ⇒ 'a Electoral-Module where
  drop-module n r A p =
    ({},
    {a ∈ A. card(above (limit A r) a) ≤ n},
    {a ∈ A. card(above (limit A r) a) > n})
```

3.2.2 Soundness

```
theorem drop-mod-sound[simp]:
  fixes
    r :: 'a Preference-Relation and
    n :: nat
  shows electoral-module (drop-module n r)
proof (intro electoral-modI)
  fix
    A :: 'a set and
    p :: 'a Profile
  let ?mod = drop-module n r
  have
    (∀ a ∈ A. a ∈ {x ∈ A. card(above (limit A r) x) ≤ n} ∨
      a ∈ {x ∈ A. card(above (limit A r) x) > n})
    by auto
  hence
    {a ∈ A. card(above (limit A r) a) ≤ n} ∪
      {a ∈ A. card(above (limit A r) a) > n} = A
    by blast
  hence 0: set-equals-partition A (drop-module n r A p)
    by simp
  have
    (∀ a ∈ A. ¬(a ∈ {x ∈ A. card(above (limit A r) x) ≤ n} ∧
      a ∈ {x ∈ A. card(above (limit A r) x) > n}))
    by auto
  hence
    {a ∈ A. card(above (limit A r) a) ≤ n} ∩
      {a ∈ A. card(above (limit A r) a) > n} = {}
```

by *blast*
hence *1*: *disjoint3* (*?mod A p*)
by *simp*
from *0 1* **show** *well-formed A* (*?mod A p*)
by *simp*
qed

3.2.3 Non-Electing

The drop module is non-electing.

theorem *drop-mod-non-electing*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *non-electing* (*drop-module n r*)
unfolding *non-electing-def*
using *assms*
by *simp*

3.2.4 Properties

The drop module is strictly defer-monotone.

theorem *drop-mod-def-lift-inv*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *defer-lift-invariance* (*drop-module n r*)
unfolding *defer-lift-invariance-def*
using *assms*
by *simp*

end

3.3 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.

3.3.1 Definition

fun *pass-module* :: *nat* ⇒ *'a Preference-Relation* ⇒ *'a Electoral-Module* **where**
pass-module n r A p =
$(\{\},$
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\},$
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\})$

3.3.2 Soundness

theorem *pass-mod-sound*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *electoral-module* (*pass-module n r*)
proof (*intro electoral-modI*)
fix
A :: *'a set* **and**
p :: *'a Profile*
let *?mod* = *pass-module n r*
have
$(\forall\ a \in A.\ a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\} \lor$
$a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\})$
using *CollectI not-less*
by *metis*
hence
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} \cup$
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} = A$
by *blast*
hence *0*: *set-equals-partition A* (*pass-module n r A p*)
by *simp*
have
$(\forall\ a \in A.\ \neg(a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\} \land$
$a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\}))$
by *auto*
hence
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} \cap$
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} = \{\}$
by *blast*
hence *1*: *disjoint3* (*?mod A p*)
by *simp*
from *0 1*
show *well-formed A* (*?mod A p*)
by *simp*

qed

3.3.3 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 (*unfold non-blocking-def*, *safe*, *simp-all*)
  show *electoral-module* (*pass-module n r*)
    using *pass-mod-sound order*
    by *simp*
next
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    a :: *'a*
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p* **and**
    card-A:
    $\{b \in A.\ n <$
      $card\ (above$
        $\{(b,\ c).\ (b,\ c) \in r \wedge$
          $b \in A \wedge c \in A\}\ b)\} = A$ **and**
    a-in-A: $a \in A$
  have *lin-ord-A*:
    linear-order-on A (*limit A r*)
    using *limit-presv-lin-ord order top-greatest*
    by *metis*
  have
    $\exists\ b \in A.\ above\ (limit\ A\ r)\ b = \{b\} \wedge$
      $(\forall\ c \in A.\ above\ (limit\ A\ r)\ c = \{c\} \longrightarrow c = b)$
    using *above-one fin-A lin-ord-A a-in-A*
    by *blast*
  hence *not-all*:
    $\{b \in A.\ card(above\ (limit\ A\ r)\ b) > n\} \neq A$
    using *Suc-leI assms*(*2*) *is-singletonI*
        is-singleton-altdef leD mem-Collect-eq
    unfolding *One-nat-def*
    by (*metis* (*no-types*, *lifting*))
  hence *reject* (*pass-module n r*) $A\ p \neq A$
    by *simp*
  thus *False*

using *order card-A*
by *simp*
qed

3.3.4 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*)
unfolding *non-electing-def*
using *assms*
by *simp*

3.3.5 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*)
unfolding *defer-lift-invariance-def*
using *assms*
by *simp*

theorem *pass-zero-mod-def-zero*[*simp*]:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 0* (*pass-module 0 r*)
proof (*unfold defers-def*, *safe*)
show *electoral-module* (*pass-module 0 r*)
using *pass-mod-sound assms*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
card-pos: $0 \leq$ *card A* **and**
finite-A: *finite A* **and**
prof-A: *profile A p*
have *lin-ord-on-A*:
linear-order-on A (*limit A r*)

using *assms limit-presv-lin-ord*
by *blast*
have *limit-is-connex*: *connex A* (*limit A r*)
using *lin-ord-imp-connex lin-ord-on-A*
by *simp*
obtain *select-alt* :: ($'a \Rightarrow bool$) $\Rightarrow$ $'a$ **where**
$\forall$ *p*. (*Collect p* = {} $\longrightarrow$ ($\forall$ *a*. $\neg$ *p a*)) $\wedge$
(*Collect p* $\neq$ {} $\longrightarrow$ *p* (*select-alt p*))
by *moura*
have $\forall$ *n*. $\neg$ (*n*::*nat*) $\leq$ *0* $\vee$ *n* = *0*
by *blast*
hence
$\forall$ *a A'*. $\neg$ *connex A'* (*limit A r*) $\vee$ *a* $\notin$ *A'* $\vee$ *a* $\notin$ *A* $\vee$
$\neg$ *card* (*above* (*limit A r*) *a*) $\leq$ *0*
using *above-connex above-presv-limit card-eq-0-iff*
equals0D finite-A assms rev-finite-subset
by (*metis* (*no-types*))
hence {*a* $\in$ *A*. *card* (*above* (*limit A r*) *a*) $\leq$ *0*} = {}
using *limit-is-connex*
by *auto*
hence *card* {*a* $\in$ *A*. *card* (*above* (*limit A r*) *a*) $\leq$ *0*} = *0*
using *card.empty*
by *metis*
thus *card* (*defer* (*pass-module 0 r*) *A p*) = *0*
by *simp*
qed

For any natural number n and any linear order, the according pass module defers n alternatives (if there are n alternatives). NOTE: The induction proof is still missing. The following are the proofs for n=1 and n=2.

theorem *pass-one-mod-def-one*[*simp*]:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 1* (*pass-module 1 r*)
proof (*unfold defers-def*, *safe*)
show *electoral-module* (*pass-module 1 r*)
using *pass-mod-sound assms*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
card-pos: *1* $\leq$ *card A* **and**
finite-A: *finite A* **and**
prof-A: *profile A p*
show *card* (*defer* (*pass-module 1 r*) *A p*) = *1*
proof −
have *A* $\neq$ {}

using *card-pos*
by *auto*
moreover have *lin-ord-on-A*:
linear-order-on A (*limit A r*)
using *assms limit-presv-lin-ord*
by *blast*
ultimately have *winner-exists*:
∃ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ∧
(∀ *b* ∈ *A*. *above* (*limit A r*) *b* = {*b*} ⟶ *b* = *a*)
using *finite-A*
by (*simp add*: *above-one*)
then obtain *w* **where** *w-unique-top*:
above (*limit A r*) *w* = {*w*} ∧
(∀ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ⟶ *a* = *w*)
using *above-one*
by *auto*
hence {*a* ∈ *A*. *card*(*above* (*limit A r*) *a*) ≤ *1*} = {*w*}
proof
assume
w-top: *above* (*limit A r*) *w* = {*w*} **and**
w-unique: ∀ *a* ∈ *A*. *above* (*limit A r*) *a* = {*a*} ⟶ *a* = *w*
have *card* (*above* (*limit A r*) *w*) ≤ *1*
using *w-top*
by *auto*
hence {*w*} ⊆ {*a* ∈ *A*. *card* (*above* (*limit A r*) *a*) ≤ *1*}
using *winner-exists w-unique-top*
by *blast*
moreover have {*a* ∈ *A*. *card*(*above* (*limit A r*) *a*) ≤ *1*} ⊆ {*w*}
proof
fix *a* :: *'a*
assume *a-in-winner-set*: *a* ∈ {*b* ∈ *A*. *card* (*above* (*limit A r*) *b*) ≤ *1*}
hence *a-in-A*: *a* ∈ *A*
by *auto*
hence *connex-limit*: *connex A* (*limit A r*)
using *lin-ord-imp-connex lin-ord-on-A*
by *simp*
hence *let q* = *limit A r in a* $\preceq_q$ *a*
using *connex-limit above-connex*
pref-imp-in-above a-in-A
by *metis*
hence (*a*, *a*) ∈ *limit A r*
by *simp*
hence *a-above-a*: *a* ∈ *above* (*limit A r*) *a*
unfolding *above-def*
by *simp*
have *above* (*limit A r*) *a* ⊆ *A*
using *above-presv-limit assms*
by *fastforce*
hence *above-finite*: *finite* (*above* (*limit A r*) *a*)

using *finite-A finite-subset*
by *simp*
have *card* (*above* (*limit A r*) *a*) $\leq$ *1*
using *a-in-winner-set*
by *simp*
moreover have *card* (*above* (*limit A r*) *a*) $\geq$ *1*
using *One-nat-def Suc-leI above-finite card-eq-0-iff equals0D neq0-conv a-above-a*
by *metis*
ultimately have *card* (*above* (*limit A r*) *a*) = *1*
by *simp*
hence {*a*} = *above* (*limit A r*) *a*
using *is-singletonE is-singleton-altdef singletonD a-above-a*
by *metis*
hence *a* = *w*
using *w-unique*
by (*simp add*: *a-in-A*)
thus *a* $\in$ {*w*}
by *simp*
qed
ultimately have
{*w*} = {*a* $\in$ *A*. *card* (*above* (*limit A r*) *a*) $\leq$ *1*}
by *auto*
thus *?thesis*
by *simp*
qed
hence *defer* (*pass-module 1 r*) *A p* = {*w*}
by *simp*
thus *card* (*defer* (*pass-module 1 r*) *A p*) = *1*
by *simp*
qed
qed

theorem *pass-two-mod-def-two*:
fixes *r* :: *'a Preference-Relation*
assumes *linear-order r*
shows *defers 2* (*pass-module 2 r*)
proof (*unfold defers-def*, *safe*)
show *electoral-module* (*pass-module 2 r*)
using *assms*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
min-2-card: *2* $\leq$ *card A* **and**
finA: *finite A* **and**
profA: *profile A p*

from *min-2-card*
have *not-empty-A*: $A \neq \{\}$
  by *auto*
moreover have *limitA-order*:
  linear-order-on A (*limit* A r)
  using *limit-presv-lin-ord assms*
  by *auto*
ultimately obtain a **where**
  a: *above* (*limit* A r) $a = \{a\}$
  using *above-one min-2-card finA profA*
  by *blast*
hence $\forall\ b \in A.$ *let* q = *limit* A r *in* $(b \preceq_q a)$
  using *limitA-order pref-imp-in-above empty-iff*
    insert-iff insert-subset above-presv-limit
    assms connex-def lin-ord-imp-connex
  by *metis*
hence *a-best*: $\forall\ b \in A.$ $(b, a) \in$ *limit* A r
  by *simp*
hence *a-above*: $\forall\ b \in A.$ $a \in$ *above* (*limit* A r) b
  unfolding *above-def*
  by *simp*
from a **have** $a \in \{a \in A.$ *card*(*above* (*limit* A r) a) $\leq 2\}$
  using *CollectI Suc-leI not-empty-A a-above card-UNIV-bool*
    card-eq-0-iff card-insert-disjoint empty-iff finA
    finite.emptyI insert-iff limitA-order above-one
    UNIV-bool nat.simps(*3*) *zero-less-Suc*
  by (*metis* (*no-types*, *lifting*))
hence *a-in-defer*: $a \in$ *defer* (*pass-module* *2* r) A p
  by *simp*
have *finite* $(A - \{a\})$
  by (*simp add*: *finA*)
moreover have *A-not-only-a*: $A - \{a\} \neq \{\}$
  using *min-2-card Diff-empty Diff-idemp Diff-insert0*
    One-nat-def not-empty-A card.insert-remove
    card-eq-0-iff finite.emptyI insert-Diff
    numeral-le-one-iff semiring-norm(*69*) *card.empty*
  by *metis*
moreover have *limitAa-order*:
  linear-order-on $(A - \{a\})$ (*limit* $(A - \{a\})$ r)
  using *limit-presv-lin-ord assms top-greatest*
  by *blast*
ultimately obtain b **where**
  b: *above* (*limit* $(A - \{a\})$ r) $b = \{b\}$
  using *above-one*
  by *metis*
hence $\forall\ c \in A - \{a\}.$ *let* q = *limit* $(A - \{a\})$ r *in* $(c \preceq_q b)$
  using *limitAa-order pref-imp-in-above empty-iff insert-iff*
    insert-subset above-presv-limit assms connex-def
    lin-ord-imp-connex

by *metis*
hence *b-in-limit*: $\forall\ c \in A - \{a\}.\ (c,\ b) \in$ *limit* $(A - \{a\})\ r$
by *simp*
hence *b-best*: $\forall\ c \in A - \{a\}.\ (c,\ b) \in$ *limit A r*
by *auto*
hence *c-not-above-b*: $\forall\ c \in A - \{a,\ b\}.\ c \notin$ *above* (*limit A r*) *b*
using *b Diff-iff Diff-insert2 above-presv-limit insert-subset*
assms limit-presv-above limit-presv-above-2
by *metis*
moreover have *above-subset*: *above* (*limit A r*) *b* $\subseteq A$
using *above-presv-limit assms*
by *metis*
moreover have *b-above-b*: $b \in$ *above* (*limit A r*) *b*
using *b b-best above-presv-limit mem-Collect-eq assms insert-subset*
unfolding *above-def*
by *metis*
ultimately have *above-b-eq-ab*: *above* (*limit A r*) $b = \{a,\ b\}$
using *a-above*
by *auto*
hence *card-above-b-eq-2*: *card* (*above* (*limit A r*) *b*) = *2*
using *A-not-only-a b-in-limit*
by *auto*
hence *b-in-defer*: $b \in$ *defer* (*pass-module 2 r*) *A p*
using *b-above-b above-subset*
by *auto*
from *b-best*
have *b-above*: $\forall\ c \in A - \{a\}.\ b \in$ *above* (*limit A r*) *c*
using *mem-Collect-eq*
unfolding *above-def*
by *metis*
have *connex A* (*limit A r*)
using *limitA-order lin-ord-imp-connex*
by *auto*
hence $\forall\ c \in A.\ c \in$ *above* (*limit A r*) *c*
by (*simp add*: *above-connex*)
hence $\forall\ c \in A - \{a,\ b\}.\ \{a,\ b,\ c\} \subseteq$ *above* (*limit A r*) *c*
using *a-above b-above*
by *auto*
moreover have $\forall\ c \in A - \{a,\ b\}.$ *card* $\{a,\ b,\ c\} = 3$
using *DiffE Suc-1 above-b-eq-ab card-above-b-eq-2*
above-subset card-insert-disjoint finA finite-subset
insert-commute numeral-3-eq-3
unfolding *One-nat-def*
by *metis*
ultimately have $\forall\ c \in A - \{a,\ b\}.$ *card* (*above* (*limit A r*) *c*) ≥ 3
using *card-mono finA finite-subset above-presv-limit assms*
by *metis*
hence $\forall\ c \in A - \{a,\ b\}.$ *card* (*above* (*limit A r*) *c*) > 2
using *less-le-trans numeral-less-iff order-refl semiring-norm*(*79*)

by *metis*
hence $\forall\ c \in A - \{a,\ b\}.\ c \notin$ *defer* (*pass-module* *2* *r*) *A* *p*
by (*simp add*: *not-le*)
moreover have *defer* (*pass-module* *2* *r*) *A* *p* $\subseteq$ *A*
by *auto*
ultimately have *defer* (*pass-module* *2* *r*) *A* *p* $\subseteq \{a,\ b\}$
by *blast*
with *a-in-defer* *b-in-defer*
have *defer* (*pass-module* *2* *r*) *A* *p* $= \{a,\ b\}$
by *fastforce*
thus *card* (*defer* (*pass-module* *2* *r*) *A* *p*) = *2*
using *above-b-eq-ab* *card-above-b-eq-2*
by *presburger*
qed

end

3.4 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.

3.4.1 Definition

fun *elect-module* :: *'a Electoral-Module* **where**
elect-module *A* *p* = (*A*, {}, {})

3.4.2 Soundness

theorem *elect-mod-sound*[*simp*]: *electoral-module* *elect-module*
unfolding *electoral-module-def*
by *simp*

3.4.3 Electing

theorem *elect-mod-electing*[*simp*]: *electing* *elect-module*
unfolding *electing-def*
by *simp*

end

3.5 Plurality Module

theory *Plurality-Module*
  imports *Component-Types/Electoral-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.

3.5.1 Definition

fun *plurality* :: *'a Electoral-Module* **where**
  plurality A p =
    $(\{a \in A.\ \forall\ x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\},$
     $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ p\ x > \textit{win-count}\ p\ a\},$
     $\{\})$

3.5.2 Soundness

theorem *plurality-sound*[*simp*]: *electoral-module plurality*
proof (*unfold electoral-module-def*, *safe*)
  fix
    A :: *'a set* **and**
    p :: *'a Profile*
  have *disjoint*:
    let elect = $\{a \in (A{::}'a\ set).\ \forall\ x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\}$;
      reject = $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ p\ a < \textit{win-count}\ p\ x\}$ *in*
    disjoint3 (*elect*, *reject*, {})
    by *auto*
  have
    let elect = $\{a \in (A{::}'a\ set).\ \forall\ x \in A.\ \textit{win-count}\ p\ x \leq \textit{win-count}\ p\ a\}$;
      reject = $\{a \in A.\ \exists\ x \in A.\ \textit{win-count}\ p\ a < \textit{win-count}\ p\ x\}$ *in*
    elect ∪ *reject* = *A*
    using *not-le-imp-less*
    by *auto*
  with *disjoint*
  show *well-formed A* (*plurality A p*)
    by *simp*
qed

3.5.3 Electing

lemma *plurality-electing-2*:
  fixes
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile*
  assumes
    A-non-empty: $A \neq \{\}$ **and**
    fin-prof-A: *finite-profile* A p
  shows *elect plurality* A $p \neq \{\}$
proof
  assume *plurality-elect-none*: *elect plurality* A $p = \{\}$
  obtain *max* **where**
    max: *max* = *Max* (*win-count* p ‘ A)
    by *simp*
  then obtain a **where**
    max-a: *win-count* p a = *max* $\wedge$ $a \in A$
    using *Max-in A-non-empty fin-prof-A empty-is-image*
        finite-imageI imageE
    by (*metis* (*no-types*, *lifting*))
  hence $\forall$ $b \in A$. *win-count* p $b \leq$ *win-count* p a
    using *A-non-empty fin-prof-A max*
    by *simp*
  moreover have $a \in A$
    using *max-a*
    by *simp*
  ultimately have
    $a \in \{b \in A.\ \forall\ c \in A.$ *win-count* p $c \leq$ *win-count* p $b\}$
    by *blast*
  hence $a \in$ *elect plurality* A p
    by *simp*
  thus *False*
    using *plurality-elect-none all-not-in-conv*
    by *metis*
qed

The plurality module is electing.

theorem *plurality-electing*[*simp*]: *electing plurality*
proof (*unfold electing-def*, *safe*)
  show *electoral-module plurality*
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    a :: $'a$
  assume
    fin-A: *finite* A **and**
    prof-p: *profile* A p **and**
    elect-none: *elect plurality* A $p = \{\}$ **and**

a-in-A: $a \in A$
have $\forall$ *A p*. $(A \neq \{\} \wedge$ *finite-profile A p*$) \longrightarrow$ *elect plurality A p* $\neq \{\}$
using *plurality-electing-2*
by (*metis* (*no-types*))
hence *empty-A*: $A = \{\}$
using *fin-A prof-p elect-none*
by (*metis* (*no-types*))
thus $a \in \{\}$
using *a-in-A*
by *simp*
qed

3.5.4 Property

lemma *plurality-inv-mono-2*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes
elect-a: $a \in$ *elect plurality A p* **and**
lift-a: *lifted A p q a*
shows *elect plurality A q* = *elect plurality A p* $\vee$ *elect plurality A q* $= \{a\}$
proof –
have *set-disj*: $\forall$ *b c*. $(b{::}'a) \notin \{c\} \vee b = c$
by *force*
have *lifted-winner*:
$\forall\ b \in A.$
$\forall$ *i::nat*. $i <$ *length p* $\longrightarrow$
(*above* $(p!i)$ $b = \{b\} \longrightarrow$
(*above* $(q!i)$ $b = \{b\} \vee$ *above* $(q!i)$ $a = \{a\}$))
using *lift-a lifted-above-winner*
unfolding *Profile.lifted-def*
by (*metis* (*no-types, lifting*))
hence
$\forall$ *i::nat*. $i <$ *length p* $\longrightarrow$
(*above* $(p!i)$ $a = \{a\} \longrightarrow$ *above* $(q!i)$ $a = \{a\}$)
using *elect-a*
by *auto*
hence *a-win-subset*:
$\{$*i::nat*. $i <$ *length p* $\wedge$ *above* $(p!i)$ $a = \{a\}\} \subseteq$
$\{$*i::nat*. $i <$ *length p* $\wedge$ *above* $(q!i)$ $a = \{a\}\}$
by *blast*
moreover have *sizes*: *length p* = *length q*
using *lift-a*
unfolding *Profile.lifted-def*
by *metis*
ultimately have *win-count-a*:

win-count p a $\leq$ *win-count q a*
by (*simp add*: *card-mono*)
have *fin-A*: *finite A*
using *lift-a*
unfolding *Profile.lifted-def*
by *metis*
hence
$\forall$ $b \in A - \{a\}$.
$\forall$ *i*::*nat*. $i <$ *length p* $\longrightarrow$
(*above* (*q*!*i*) $a = \{a\}$ $\longrightarrow$ *above* (*q*!*i*) $b \neq \{b\}$)
using *DiffE above-one-2 lift-a insertCI insert-absorb insert-not-empty sizes*
unfolding *Profile.lifted-def profile-def*
by *metis*
with *lifted-winner*
have *above-QtoP*:
$\forall$ $b \in A - \{a\}$.
$\forall$ *i*::*nat*. $i <$ *length p* $\longrightarrow$
(*above* (*q*!*i*) $b = \{b\}$ $\longrightarrow$ *above* (*p*!*i*) $b = \{b\}$)
using *lifted-above-winner-3 lift-a*
unfolding *Profile.lifted-def*
by *metis*
hence
$\forall$ $b \in A - \{a\}$.
$\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*q*!*i*) $b = \{b\}\} \subseteq$
$\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*p*!*i*) $b = \{b\}\}$
by (*simp add*: *Collect-mono*)
hence *win-count-other*:
$\forall$ $b \in A - \{a\}$. *win-count p b* $\geq$ *win-count q b*
by (*simp add*: *card-mono sizes*)
show
elect plurality A q = *elect plurality A p* $\vee$
elect plurality A q $= \{a\}$
proof (*cases*)
assume *win-count p a* = *win-count q a*
hence
card $\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*p*!*i*) $a = \{a\}\}$ =
card $\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*q*!*i*) $a = \{a\}\}$
using *sizes*
by *simp*
moreover have
finite $\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*q*!*i*) $a = \{a\}\}$
by *simp*
ultimately have
$\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*p*!*i*) $a = \{a\}\}$ =
$\{$*i*::*nat*. $i <$ *length p* $\wedge$ *above* (*q*!*i*) $a = \{a\}\}$
using *a-win-subset*
by (*simp add*: *card-subset-eq*)
hence *above-pq*:
$\forall$ *i*::*nat*. $i <$ *length p* $\longrightarrow$

$(above\ (p!i)\ a = \{a\}) = (above\ (q!i)\ a = \{a\})$
by *blast*
moreover have
$\forall\ b \in A - \{a\}.$
$\forall\ i::nat.\ i < length\ p \longrightarrow$
$(above\ (p!i)\ b = \{b\} \longrightarrow$
$(above\ (q!i)\ b = \{b\} \vee above\ (q!i)\ a = \{a\}))$
using *lifted-winner*
by *auto*
moreover have
$\forall\ b \in A - \{a\}.$
$\forall\ i::nat.\ i < length\ p \longrightarrow$
$(above\ (p!i)\ b = \{b\} \longrightarrow above\ (p!i)\ a \neq \{a\})$
proof (*rule ccontr*, *simp*, *safe*, *simp*)
fix
$b :: 'a$ **and**
$i :: nat$
assume
b-in-A: $b \in A$ **and**
i-in-range: $i < length\ p$ **and**
abv-b: $above\ (p!i)\ b = \{b\}$ **and**
abv-a: $above\ (p!i)\ a = \{a\}$
have *not-empty*: $A \neq \{\}$
using *b-in-A*
by *auto*
have *linear-order-on* A $(p!i)$
using *lift-a i-in-range*
unfolding *Profile.lifted-def profile-def*
by *simp*
thus $b = a$
using *not-empty abv-a abv-b fin-A above-one-2*
by *metis*
qed
ultimately have *above-PtoQ*:
$\forall\ b \in A - \{a\}.$
$\forall\ i::nat.\ i < length\ p \longrightarrow$
$(above\ (p!i)\ b = \{b\} \longrightarrow above\ (q!i)\ b = \{b\})$
by *simp*
hence
$\forall\ b \in A.$
$card\ \{i::nat.\ i < length\ p \wedge above\ (p!i)\ b = \{b\}\} =$
$card\ \{i::nat.\ i < length\ q \wedge above\ (q!i)\ b = \{b\}\}$
proof (*safe*)
fix $b :: 'a$
assume
$\forall\ c \in A - \{a\}.\ \forall\ i < length\ p.$
$above\ (p!i)\ c = \{c\} \longrightarrow above\ (q!i)\ c = \{c\}$ **and**
b-in-A: $b \in A$
show

```
      card {i. i < length p ∧ above (p!i) b = {b}} =
        card {i. i < length q ∧ above (q!i) b = {b}}
      using DiffI b-in-A set-disj above-PtoQ above-QtoP above-pq sizes
      by (metis (no-types, lifting))
  qed
  hence ∀ b ∈ A. win-count p b = win-count q b
    by simp
  hence
    {b ∈ A. ∀ c ∈ A. win-count p c ≤ win-count p b} =
      {b ∈ A. ∀ c ∈ A. win-count q c ≤ win-count q b}
    by auto
  thus ?thesis
    by simp
next
  assume win-count p a ≠ win-count q a
  hence strict-less:
    win-count p a < win-count q a
    using win-count-a
    by simp
  have a-in-win-p:
    a ∈ {b ∈ A. ∀ c ∈ A. win-count p c ≤ win-count p b}
    using elect-a
    by simp
  hence ∀ b ∈ A. win-count p b ≤ win-count p a
    by simp
  with strict-less win-count-other
  have less: ∀ b ∈ A − {a}. win-count q b < win-count q a
    using DiffD1 antisym dual-order.trans
          not-le-imp-less win-count-a
    by metis
  hence ∀ b ∈ A − {a}. ¬(∀ c ∈ A. win-count q c ≤ win-count q b)
    using lift-a not-le
    unfolding Profile.lifted-def
    by metis
  hence
    ∀ b ∈ A − {a}.
      b ∉ {c ∈ A. ∀ b ∈ A. win-count q b ≤ win-count q c}
    by blast
  hence ∀ b ∈ A − {a}. b ∉ elect plurality A q
    by simp
  moreover have a ∈ elect plurality A q
  proof −
    from less
    have ∀ b ∈ A − {a}. win-count q b ≤ win-count q a
      using less-imp-le
      by metis
    moreover have win-count q a ≤ win-count q a
      by simp
    ultimately have ∀ b ∈ A. win-count q b ≤ win-count q a
```

```
      by auto
    moreover have a ∈ A
      using a-in-win-p
      by simp
    ultimately have
      a ∈ {b ∈ A.
          ∀ c ∈ A. win-count q c ≤ win-count q b}
      by simp
    thus ?thesis
      by simp
  qed
  moreover have
    elect plurality A q ⊆ A
    by simp
  ultimately show ?thesis
    by auto
 qed
qed
```

The plurality rule is invariant-monotone.

```
theorem plurality-inv-mono[simp]: invariant-monotonicity plurality
proof (unfold invariant-monotonicity-def, intro conjI impI allI)
  show electoral-module plurality
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    q :: 'a Profile and
    a :: 'a
  assume a ∈ elect plurality A p ∧ Profile.lifted A p q a
  thus elect plurality A q = elect plurality A p ∨ elect plurality A q = {a}
    using plurality-inv-mono-2
    by metis
qed

end
```

3.6 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.

3.6.1 Definition

fun *borda-score* :: *'a Evaluation-Function* **where**
  borda-score x A p = ($\sum$ *y* ∈ *A*. (*prefer-count p x y*))

fun *borda* :: *'a Electoral-Module* **where**
  borda A p = *max-eliminator borda-score A p*

3.6.2 Soundness

theorem *borda-sound*: *electoral-module borda*
  unfolding *borda.simps*
  using *max-elim-sound*
  by *metis*

end

3.7 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.

3.7.1 Definition

fun *condorcet-score* :: *'a Evaluation-Function* **where**
  condorcet-score x A p =
    (*if* (*condorcet-winner A p x*) *then 1 else 0*)

fun *condorcet* :: *'a Electoral-Module* **where**
  condorcet A p = (*max-eliminator condorcet-score*) *A p*

3.7.2 Soundness

theorem *condorcet-sound*: *electoral-module condorcet*
  unfolding *condorcet.simps*
  using *max-elim-sound*
  by *metis*

3.7.3 Property

theorem *condorcet-score-is-condorcet-rating*: *condorcet-rating condorcet-score*
proof (*unfold condorcet-rating-def*, *safe*)
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    w :: $'a$ **and**
    l :: $'a$
  assume
    c-win: *condorcet-winner* A p w **and**
    l-in-A: $l \in A$ **and**
    l-neq-w: $l \neq w$
  have $\neg$ *condorcet-winner* A p l
    using *c-win l-neq-w cond-winner-unique*
    by (*metis* (*no-types*))
  thus *condorcet-score* l A p $<$ *condorcet-score* w A p
    using *c-win*
    by *simp*
qed

theorem *condorcet-is-dcc*: *defer-condorcet-consistency condorcet*
proof (*unfold defer-condorcet-consistency-def electoral-module-def*, *safe*)
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile*
  assume
    finA: *finite* A **and**
    profA: *profile* A p
  have *well-formed* A (*max-eliminator condorcet-score* A p)
    using *finA profA max-elim-sound*
    unfolding *electoral-module-def*
    by *metis*
  thus *well-formed* A (*condorcet* A p)
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    a :: $'a$
  assume
    cwin-w: *condorcet-winner* A p a **and**
    finA: *finite* A

have *max-cscore-dcc*:
  defer-condorcet-consistency (*max-eliminator condorcet-score*)
  using *cr-eval-imp-dcc-max-elim*
  by (*simp add*: *condorcet-score-is-condorcet-rating*)
have
  max-eliminator condorcet-score A p =
    ({},
    A − *defer* (*max-eliminator condorcet-score*) *A p*,
    {*b* ∈ *A*. *condorcet-winner A p b*})
  using *cwin-w finA max-cscore-dcc*
  unfolding *defer-condorcet-consistency-def*
  by (*metis* (*no-types*))
thus
  condorcet A p =
    ({},
     A − *defer condorcet A p*,
     {*d* ∈ *A*. *condorcet-winner A p d*})
  by *simp*
qed

end

3.8 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.

3.8.1 Definition

fun *copeland-score* :: *'a Evaluation-Function* **where**
  copeland-score x A p =
    card {*y* ∈ *A* . *wins x p y*} − *card* {*y* ∈ *A* . *wins y p x*}

fun *copeland* :: *'a Electoral-Module* **where**
  copeland A p = *max-eliminator copeland-score A p*

3.8.2 Soundness

theorem *copeland-sound*: *electoral-module copeland*
  unfolding *copeland.simps*
  using *max-elim-sound*
  by *metis*

3.8.3 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**
    p :: $'a$ *Profile* **and**
    w :: $'a$
  assumes *condorcet-winner A p w*
  shows $card\ \{a \in A.\ wins\ w\ p\ a\} = card\ A - 1$
proof –
  from *assms*
  have 0: $\forall\ a \in A - \{w\}.\ wins\ w\ p\ a$
    by *simp*
  have 1: $\forall\ M.\ \{x \in M.\ True\} = M$
    by *blast*
  from *0 1*
  have $\{a \in A - \{w\}.\ wins\ w\ p\ a\} = A - \{w\}$
    by *blast*
  hence 10: $card\ \{a \in A - \{w\}.\ wins\ w\ p\ a\} = card\ (A - \{w\})$
    by *simp*
  from *assms*
  have 11: $w \in A$
    by *simp*
  hence $card\ (A - \{w\}) = card\ A - 1$
    using *card-Diff-singleton assms*
    by *metis*
  hence *winner-amount-one*:
    $card\ \{a \in A - \{w\}.\ wins\ w\ p\ a\} = card\ (A) - 1$
    using *10*
    by *linarith*
  have 2: $\forall\ a \in \{w\}.\ \neg\ wins\ a\ p\ a$
    by (*simp add*: *wins-irreflex*)
  have 3: $\forall\ M.\ \{x \in M\ .\ False\} = \{\}$
    by *blast*
  from *2 3*
  have $\{a \in \{w\}.\ wins\ w\ p\ a\} = \{\}$
    by *blast*
  hence *winner-amount-zero*: $card\ \{a \in \{w\}.\ wins\ w\ p\ a\} = 0$
    by *simp*
  have *disjunct*:
    $\{a \in A - \{w\}.\ wins\ w\ p\ a\} \cap \{a \in \{w\}.\ wins\ w\ p\ a\} = \{\}$

by *blast*
have *union*:
$\{a \in A - \{w\}.\ wins\ w\ p\ a\} \cup \{x \in \{w\}.\ wins\ w\ p\ x\} =$
$\{a \in A.\ wins\ w\ p\ a\}$
using *2*
by *blast*
have *finite-defeated*: *finite* $\{a \in A - \{w\}.\ wins\ w\ p\ a\}$
using *assms*
by *simp*
have *finitene-winners*: *finite* $\{a \in \{w\}.\ wins\ w\ p\ a\}$
by *simp*
from *finite-defeated finitene-winners disjunct card-Un-disjoint*
have
card $(\{a \in A - \{w\}.\ wins\ w\ p\ a\} \cup \{a \in \{w\}.\ wins\ w\ p\ a\}) =$
card $\{a \in A - \{w\}.\ wins\ w\ p\ a\}$ + *card* $\{a \in \{w\}.\ wins\ w\ p\ a\}$
by *blast*
with *union*
have *card* $\{a \in A.\ wins\ w\ p\ a\} =$
card $\{a \in A - \{w\}.\ wins\ w\ p\ a\}$ + *card* $\{a \in \{w\}.\ wins\ w\ p\ a\}$
by *simp*
with *winner-amount-one winner-amount-zero*
show *?thesis*
by *linarith*
qed

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**
p :: *'a Profile* **and**
w :: *'a*
assumes *winner*: *condorcet-winner A p w*
shows *card* $\{a \in A.\ wins\ a\ p\ w\} = 0$
using *Collect-empty-eq card-eq-0-iff insert-Diff insert-iff*
wins-antisym winner
unfolding *condorcet-winner.simps*
by (*metis* (*no-types*, *lifting*))

Copeland score of a Condorcet winner.

lemma *cond-winner-imp-copeland-score*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a*
assumes *winner*: *condorcet-winner A p w*
shows *copeland-score w A p* = *card A* − *1*
proof (*unfold copeland-score.simps*)
have *card-A-sub-one*: *card* $\{a \in A.\ wins\ w\ p\ a\}$ = *card A* − *1*
using *cond-winner-imp-win-count winner*

    by *simp*
  have *card-zero*: $card\ \{a \in A.\ wins\ a\ p\ w\} = 0$
    using *cond-winner-imp-loss-count winner*
    by (*metis* (*no-types*))
  have $card\ A - 1 - 0 = card\ A - 1$
    by *simp*
  thus
    $card\ \{a \in A.\ wins\ w\ p\ a\} - card\ \{a \in A.\ wins\ a\ p\ w\} =$
      $card\ A - 1$
    using *card-zero card-A-sub-one*
    by *simp*
qed

For a non-Condorcet winner l, we have: "card y in A . wins x p y <= |A| - 1 - 1".

lemma *non-cond-winner-imp-win-count*:
  fixes
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    w :: *'a* **and**
    l :: *'a*
  assumes
    winner: *condorcet-winner* $A\ p\ w$ **and**
    loser: $l \neq w$ **and**
    l-in-A: $l \in A$
  shows $card\ \{a \in A\ .\ wins\ l\ p\ a\} \leq card\ A - 2$
proof −
  from *winner loser l-in-A*
  have $wins\ w\ p\ l$
    by *simp*
  hence *0*: $\neg\ wins\ l\ p\ w$
    using *wins-antisym*
    by *simp*
  have *1*: $\neg\ wins\ l\ p\ l$
    using *wins-irreflex*
    by *simp*
  from *0 1* **have** *2*:
    $\{y \in A\ .\ wins\ l\ p\ y\} =$
        $\{y \in A - \{l, w\}\ .\ wins\ l\ p\ y\}$
    by *blast*
  have *3*: $\forall\ M\ f\ .\ finite\ M \longrightarrow card\ \{x \in M\ .\ f\ x\} \leq card\ M$
    by (*simp add*: *card-mono*)
  have *4*: $finite\ (A - \{l, w\})$
    using *finite-Diff winner*
    by *simp*
  from *3 4*
  have *5*:
    $card\ \{y \in A - \{l, w\}\ .\ wins\ l\ p\ y\} \leq$
      $card\ (A - \{l, w\})$

by (*metis* (*full-types*))
have $w \in A$
using *winner*
by *simp*
with *l-in-A*
have *card* $(A - \{l, w\}) = $ *card* $A -$ *card* $\{l, w\}$
by (*simp add*: *card-Diff-subset*)
hence *card* $(A - \{l, w\}) = $ *card* $A - 2$
using *loser*
by *simp*
with *5 2*
show *?thesis*
by *simp*
qed

3.8.4 Property

The Copeland score is Condorcet rating.

theorem *copeland-score-is-cr*: *condorcet-rating copeland-score*
proof (*unfold condorcet-rating-def*, *unfold copeland-score.simps*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**
l :: *'a*
assume
winner: *condorcet-winner* $A\ p\ w$ **and**
l-in-A: $l \in A$ **and**
l-neq-w: $l \neq w$
from *winner*
have *0*:
card $\{y \in A.$ *wins* $w\ p\ y\} -$ *card* $\{y \in A.$ *wins* $y\ p\ w\} = $ *card* $A - 1$
using *cond-winner-imp-copeland-score*
by *fastforce*
from *winner l-neq-w l-in-A*
have *1*:
card $\{y \in A.$ *wins* $l\ p\ y\} -$ *card* $\{y \in A.$ *wins* $y\ p\ l\} \leq$ *card* $A - 2$
using *non-cond-winner-imp-win-count*
by *fastforce*
have *2*: *card* $A - 2 <$ *card* $A - 1$
using *card-0-eq card-Diff-singleton diff-less-mono2*
empty-iff finite-Diff insertE insert-Diff
l-in-A l-neq-w neq0-conv one-less-numeral-iff
semiring-norm(*76*) *winner zero-less-diff*
unfolding *condorcet-winner.simps*
by *metis*
hence
card $\{y \in A.$ *wins* $l\ p\ y\} -$ *card* $\{y \in A.$ *wins* $y\ p\ l\} <$ *card* $A - 1$
using *1 le-less-trans*

by *blast*
thus
card $\{y \in A.\ wins\ l\ p\ y\} - card\ \{y \in A.\ wins\ y\ p\ l\} <$
card $\{y \in A.\ wins\ w\ p\ y\} - card\ \{y \in A.\ wins\ y\ p\ w\}$
using *0*
by *linarith*
qed

theorem *copeland-is-dcc*: *defer-condorcet-consistency copeland*
proof (*unfold defer-condorcet-consistency-def electoral-module-def*, *safe*)
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile*
assume
finA: *finite A* **and**
profA: *profile A p*
have *well-formed A* (*max-eliminator copeland-score A p*)
using *finA max-elim-sound profA*
unfolding *electoral-module-def*
by *metis*
thus *well-formed A* (*copeland A p*)
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$
assume
cwin-w: *condorcet-winner A p w* **and**
finA: *finite A*
have *max-cplscore-dcc*:
defer-condorcet-consistency (*max-eliminator copeland-score*)
using *cr-eval-imp-dcc-max-elim*
by (*simp add*: *copeland-score-is-cr*)
have
$\forall\ A\ p.$ (*copeland A p* = *max-eliminator copeland-score A p*)
by *simp*
thus
copeland A p =
({},
A − *defer copeland A p*,
$\{d \in A.$ *condorcet-winner A p d*$\}$)
using *Collect-cong cwin-w finA max-cplscore-dcc*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*no-types*, *lifting*))
qed

end

3.9 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.

3.9.1 Definition

fun *minimax-score* :: *'a Evaluation-Function* **where**
  minimax-score x A p =
    Min {*prefer-count p x y* | *y* . *y* ∈ *A* − {*x*}}

fun *minimax* :: *'a Electoral-Module* **where**
  minimax A p = *max-eliminator minimax-score A p*

3.9.2 Soundness

theorem *minimax-sound*: *electoral-module minimax*
  unfolding *minimax.simps*
  using *max-elim-sound*
  by *metis*

3.9.3 Lemma

lemma *non-cond-winner-minimax-score*:
  fixes
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    w :: *'a* **and**
    l :: *'a*
  assumes
    prof: *profile A p* **and**
    winner: *condorcet-winner A p w* **and**
    l-in-A: *l* ∈ *A* **and**
    l-neq-w: *l* ≠ *w*
  shows *minimax-score l A p* ≤ *prefer-count p l w*
proof (*simp*)
  let
    ?set = {*prefer-count p l y* | *y* . *y* ∈ *A* − {*l*}} **and**

?lscore = *minimax-score l A p*
have *finite A*
using *prof winner*
by *simp*
hence *finite* (*A* − {*l*})
using *finite-Diff*
by *simp*
hence *finite*: *finite ?set*
by *simp*
have *w* ∈ *A*
using *winner*
by *simp*
hence *0*: *w* ∈ *A* − {*l*}
using *l-neq-w*
by *force*
hence *not-empty*: *?set* ≠ {}
by *blast*
have *?lscore* = *Min ?set*
by *simp*
hence *1*: *?lscore* ∈ *?set* ∧ (∀ *p* ∈ *?set*. *?lscore* ≤ *p*)
using *local.finite not-empty Min-le Min-eq-iff*
by (*metis* (*no-types*, *lifting*))
thus *Min* {*card* {*i*. *i* < *length p* ∧ (*y*, *l*) ∈ *p*!*i*} | *y*. *y* ∈ *A* ∧ *y* ≠ *l*} ≤
card {*i*. *i* < *length p* ∧ (*w*, *l*) ∈ *p*!*i*}
using *0*
by *auto*
qed

3.9.4 Property

theorem *minimax-score-cond-rating*: *condorcet-rating minimax-score*
proof (*unfold condorcet-rating-def minimax-score.simps prefer-count.simps*, *safe*, *rule ccontr*)
fix
A :: ′*a set* **and**
p :: ′*a Profile* **and**
w :: ′*a* **and**
l :: ′*a*
assume
winner: *condorcet-winner A p w* **and**
l-in-A: *l* ∈ *A* **and**
l-neq-w:*l* ≠ *w* **and**
min-leq:
¬ *Min* {*card* {*i*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *l*))} |
y. *y* ∈ *A* − {*l*}} <
Min {*card* {*i*. *i* < *length p* ∧ (*let r* = (*p*!*i*) *in* (*y* $\preceq_r$ *w*))} |
y. *y* ∈ *A* − {*w*}}
hence *000*:
Min {*prefer-count p l y* | *y*. *y* ∈ *A* − {*l*}} ≥

Min {*prefer-count p w y* | *y*. $y \in A - \{w\}$}
by *auto*
have *prof*: *profile A p*
using *condorcet-winner.simps winner*
by *metis*
from *prof winner l-in-A l-neq-w*
have *100*:
prefer-count p l w $\geq$ *Min* {*prefer-count p l y* | *y* . $y \in A - \{l\}$}
using *non-cond-winner-minimax-score minimax-score.simps*
by *metis*
from *l-in-A*
have *l-in-A-without-w*: $l \in A - \{w\}$
by (*simp add*: *l-neq-w*)
hence *2*: {*prefer-count p w y* | *y* . $y \in A - \{w\}$} $\neq$ {}
by *blast*
have *finite* $(A - \{w\})$
using *prof condorcet-winner.simps winner finite-Diff*
by *metis*
hence *3*: *finite* {*prefer-count p w y* | *y* . $y \in A - \{w\}$}
by *simp*
from *2 3*
have *4*:
$\exists$ $n \in A - \{w\}$. *prefer-count p w n* =
Min {*prefer-count p w y* | *y* . $y \in A - \{w\}$}
using *Min-in*
by *fastforce*
then obtain *n* **where** *200*:
prefer-count p w n =
Min {*prefer-count p w y* | *y* . $y \in A - \{w\}$} **and**
6: $n \in A - \{w\}$
by *metis*
hence *n-in-A*: $n \in A$
using *DiffE*
by *metis*
from *6*
have *n-neq-w*: $n \neq w$
by *simp*
from *winner*
have *w-in-A*: $w \in A$
by *simp*
from *6 prof winner*
have *300*: *prefer-count p w n* > *prefer-count p n w*
by *simp*
from *100 000 200*
have *400*: *prefer-count p l w* $\geq$ *prefer-count p w n*
by *linarith*
with *prof n-in-A w-in-A l-in-A n-neq-w*
l-neq-w pref-count-sym
have *700*: *prefer-count p n w* $\geq$ *prefer-count p w l*

by *metis*
have *prefer-count p l w* > *prefer-count p w l*
using *300 400 700*
by *linarith*
hence *wins l p w*
by *simp*
thus *False*
using *l-in-A-without-w wins-antisym winner*
unfolding *condorcet-winner.simps*
by *metis*
qed

theorem *minimax-is-dcc*: *defer-condorcet-consistency minimax*
proof (*unfold defer-condorcet-consistency-def electoral-module-def*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
finA: *finite A* **and**
profA: *profile A p*
have *well-formed A* (*max-eliminator minimax-score A p*)
using *finA max-elim-sound par-comp-result-sound profA*
by *metis*
thus *well-formed A* (*minimax A p*)
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a*
assume
cwin-w: *condorcet-winner A p w* **and**
finA: *finite A*
have *max-mmaxscore-dcc*:
defer-condorcet-consistency (*max-eliminator minimax-score*)
using *cr-eval-imp-dcc-max-elim*
by (*simp add*: *minimax-score-cond-rating*)
hence
max-eliminator minimax-score A p =
({},
A − *defer* (*max-eliminator minimax-score*) *A p*,
{*a* ∈ *A*. *condorcet-winner A p a*})
using *cwin-w finA*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*no-types*))
thus
minimax A p =
({},
A − *defer minimax A p*,

$\{d \in A.\ condorcet\text{-}winner\ A\ p\ d\})$

by *simp*

qed

end

Chapter 4

Compositional Structures

4.1 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.

4.1.1 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 (*unfold rejects-def*, *safe*)
  show *electoral-module* (*drop-module 0 r*)
    using *assms*
    by *simp*
next
  fix
    A :: *'a set* **and**
    p :: *'a Profile*
  assume
    finite-A: *finite A* **and**
    prof-A: *profile A p*
  have *f1*: *connex UNIV r*
    using *assms lin-ord-imp-connex*
    by *auto*
  have *connex*:
    connex A (*limit A r*)
    using *f1 limit-presv-connex subset-UNIV*
    by *metis*

have
$\forall\ B\ a.\ B \neq \{\} \lor (a{::}'a) \notin B$
by *simp*
hence
$\forall\ a\ B.$
$\neg\ connex\ B\ (limit\ A\ r) \lor a \notin B \lor a \notin A \lor$
$\neg\ card\ (above\ (limit\ A\ r)\ a) \leq 0$
using *above-connex above-presv-limit card-eq-0-iff finite-A finite-subset le-0-eq assms*
by (*metis* (*no-types*))
hence $\{a \in A.\ card\ (above\ (limit\ A\ r)\ a) \leq 0\} = \{\}$
using *connex*
by *auto*
hence $card\ \{a \in A.\ card\ (above\ (limit\ A\ r)\ a) \leq 0\} = 0$
using *card.empty*
by (*metis* (*full-types*))
thus $card\ (reject\ (drop\text{-}module\ 0\ r)\ A\ p) = 0$
by *simp*
qed

The drop module rejects n alternatives (if there are n alternatives). NOTE: The induction proof is still missing. Following is the proof for n=2.

theorem *drop-two-mod-rej-two*[*simp*]:
fixes r :: *'a Preference-Relation*
assumes *linear-order r*
shows *rejects 2* (*drop-module 2 r*)
proof −
have *rej-drop-eq-def-pass*:
$reject\ (drop\text{-}module\ 2\ r) = defer\ (pass\text{-}module\ 2\ r)$
by *simp*
obtain
m :: (*'a Electoral-Module*) $\Rightarrow$ *nat* $\Rightarrow$ *'a set* **and**
m' :: (*'a Electoral-Module*) $\Rightarrow$ *nat* $\Rightarrow$ *'a Profile* **where**
$\forall\ f\ n.\ (\exists\ A\ p.\ n \leq card\ A \land finite\text{-}profile\ A\ p \land card\ (reject\ f\ A\ p) \neq n) =$
$(n \leq card\ (m\ f\ n) \land finite\text{-}profile\ (m\ f\ n)\ (m'\ f\ n) \land$
$card\ (reject\ f\ (m\ f\ n)\ (m'\ f\ n)) \neq n)$
by *moura*
hence *rejected-card*:
$\forall\ f\ n.$
$(\neg\ rejects\ n\ f \land electoral\text{-}module\ f \longrightarrow$
$n \leq card\ (m\ f\ n) \land finite\text{-}profile\ (m\ f\ n)\ (m'\ f\ n) \land$
$card\ (reject\ f\ (m\ f\ n)\ (m'\ f\ n)) \neq n)$
unfolding *rejects-def*
by *blast*
have
$2 \leq card\ (m\ (drop\text{-}module\ 2\ r)\ 2) \land finite\ (m\ (drop\text{-}module\ 2\ r)\ 2) \land$
$profile\ (m\ (drop\text{-}module\ 2\ r)\ 2)\ (m'\ (drop\text{-}module\ 2\ r)\ 2) \longrightarrow$
$card\ (reject\ (drop\text{-}module\ 2\ r)\ (m\ (drop\text{-}module\ 2\ r)\ 2)\ (m'\ (drop\text{-}module\ 2\ r)\ 2)) = 2$

using *rej-drop-eq-def-pass assms pass-two-mod-def-two*
unfolding *defers-def*
by (*metis* (*no-types*))
thus *?thesis*
using *rejected-card drop-mod-sound assms*
by *blast*
qed

The pass and drop module are (disjoint-)compatible.

theorem *drop-pass-disj-compat*[*simp*]:
fixes
r :: *'a Preference-Relation* **and**
n :: *nat*
assumes *linear-order r*
shows *disjoint-compatibility* (*drop-module n r*) (*pass-module n r*)
proof (*unfold disjoint-compatibility-def*, *safe*)
show *electoral-module* (*drop-module n r*)
using *assms*
by *simp*
next
show *electoral-module* (*pass-module n r*)
using *assms*
by *simp*
next
fix A :: *'a set*
assume *fin*: *finite A*
obtain p :: *'a Profile* **where**
finite-profile A p
using *empty-iff empty-set fin profile-set*
by *metis*
show
$\exists\ B \subseteq A.$
($\forall\ a \in B.$ *indep-of-alt* (*drop-module n r*) $A\ a \wedge$
($\forall\ p.$ *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*drop-module n r*) $A\ p$)) $\wedge$
($\forall\ a \in A - B.$ *indep-of-alt* (*pass-module n r*) $A\ a \wedge$
($\forall\ p.$ *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*pass-module n r*) $A\ p$))
proof
have *same-A*:
$\forall\ p\ q.$ (*finite-profile A p* $\wedge$ *finite-profile A q*) $\longrightarrow$
reject (*drop-module n r*) $A\ p =$
reject (*drop-module n r*) $A\ q$
by *auto*
let $?A =$ *reject* (*drop-module n r*) $A\ p$
have $?A \subseteq A$
by *auto*
moreover have $\forall\ a \in\ ?A.$ *indep-of-alt* (*drop-module n r*) $A\ a$
using *assms*

unfolding *indep-of-alt-def*
by *simp*
moreover have
$\forall\ a \in$ *?A*. $\forall\ p$. *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*drop-module n r*) *A p*
by *auto*
moreover have
($\forall\ a \in A -$ *?A*. *indep-of-alt* (*pass-module n r*) *A a*)
using *assms*
unfolding *indep-of-alt-def*
by *simp*
moreover have
$\forall\ a \in A -$ *?A*. $\forall\ p$. *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*pass-module n r*) *A p*
by *auto*
ultimately show
?A $\subseteq A \wedge$
($\forall\ a \in$ *?A*. *indep-of-alt* (*drop-module n r*) *A a* $\wedge$
($\forall\ p$. *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*drop-module n r*) *A p*)) $\wedge$
($\forall\ a \in A -$ *?A*. *indep-of-alt* (*pass-module n r*) *A a* $\wedge$
($\forall\ p$. *finite-profile A p* $\longrightarrow$
$a \in$ *reject* (*pass-module n r*) *A p*))
by *simp*
qed
qed

end

4.2 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.

4.2.1 Definition

fun *revision-composition* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* **where**
revision-composition m A p = ({}, *A* $-$ *elect m A p*, *elect m A p*)

abbreviation *rev* ::
'a Electoral-Module ⇒ *'a Electoral-Module* (-↓ *50*) **where**
  m↓ == *revision-composition m*

4.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
  fixes *m* :: *'a Electoral-Module*
  assumes *electoral-module m*
  shows *electoral-module* (*revision-composition m*)
proof −
  from *assms*
  have ∀ *A p*. *finite-profile A p* ⟶ *elect m A p* ⊆ *A*
    using *elect-in-alts*
    by *metis*
  hence ∀ *A p*. *finite-profile A p* ⟶ (*A* − *elect m A p*) ∪ *elect m A p* = *A*
    by *blast*
  hence *unity*:
    ∀ *A p*. *finite-profile A p* ⟶
      set-equals-partition A (*revision-composition m A p*)
    by *simp*
  have ∀ *A p*. *finite-profile A p* ⟶ (*A* − *elect m A p*) ∩ *elect m A p* = {}
    by *blast*
  hence *disjoint*:
    ∀ *A p*. *finite-profile A p* ⟶ *disjoint3* (*revision-composition m A p*)
    by *simp*
  from *unity disjoint*
  show *?thesis*
    by (*simp add*: *electoral-modI*)
qed

4.2.3 Composition Rules

An electoral module received by revision is never electing.

theorem *rev-comp-non-electing*[*simp*]:
  fixes *m* :: *'a Electoral-Module*
  assumes *electoral-module m*
  shows *non-electing* (*m*↓)
  using *assms*
  unfolding *non-electing-def*
  by *simp*

Revising an electing electoral module results in a non-blocking electoral module.

theorem *rev-comp-non-blocking*[*simp*]:
  fixes *m* :: *'a Electoral-Module*
  assumes *electing m*
  shows *non-blocking* (*m*↓)
proof (*unfold non-blocking-def*, *safe*, *simp-all*)

show *electoral-module* $(m\downarrow)$
  using *assms rev-comp-sound*
  unfolding *electing-def*
  by (*metis* (*no-types*, *lifting*))
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    x :: $'a$
  assume
    fin-A: *finite* A **and**
    prof-A: *profile* A p **and**
    no-elect: $A -$ *elect* m A $p = A$ **and**
    x-in-A: $x \in A$
  from *no-elect* **have** *non-elect*:
    non-electing m
    using *assms prof-A x-in-A fin-A empty-iff*
        Diff-disjoint Int-absorb2 elect-in-alts
    unfolding *electing-def*
    by (*metis* (*no-types*, *lifting*))
  show *False*
    using *non-elect assms empty-iff fin-A prof-A x-in-A*
    unfolding *electing-def non-electing-def*
    by (*metis* (*no-types*, *lifting*))
qed

Revising an invariant monotone electoral module results in a defer-invariant-monotone electoral module.

theorem *rev-comp-def-inv-mono*[*simp*]:
  fixes m :: $'a$ *Electoral-Module*
  assumes *invariant-monotonicity* m
  shows *defer-invariant-monotonicity* $(m\downarrow)$
proof (*unfold defer-invariant-monotonicity-def*, *safe*)
  show *electoral-module* $(m\downarrow)$
    using *assms rev-comp-sound*
    unfolding *invariant-monotonicity-def*
    by *simp*
next
  show *non-electing* $(m\downarrow)$
    using *assms rev-comp-non-electing*
    unfolding *invariant-monotonicity-def*
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    q :: $'a$ *Profile* **and**
    a :: $'a$ **and**
    x :: $'a$ **and**

xa :: $'a$
assume
rev-p-defer-a: $a \in$ *defer* $(m\downarrow)$ A p **and**
a-lifted: *lifted* A p q a **and**
rev-q-defer-x: $x \in$ *defer* $(m\downarrow)$ A q **and**
x-non-eq-a: $x \neq a$ **and**
rev-q-defer-xa: $xa \in$ *defer* $(m\downarrow)$ A q
from *rev-p-defer-a*
have *elect-a-in-p*: $a \in$ *elect* m A p
by *simp*
from *rev-q-defer-x x-non-eq-a*
have *elect-no-unique-a-in-q*: *elect* m A $q \neq \{a\}$
by *force*
from *assms*
have *elect* m A q = *elect* m A p
using *a-lifted elect-a-in-p elect-no-unique-a-in-q*
unfolding *invariant-monotonicity-def*
by (*metis* (*no-types*))
thus $xa \in$ *defer* $(m\downarrow)$ A p
using *rev-q-defer-xa*
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$ **and**
x :: $'a$ **and**
xa :: $'a$
assume
rev-p-defer-a: $a \in$ *defer* $(m\downarrow)$ A p **and**
a-lifted: *lifted* A p q a **and**
rev-q-defer-x: $x \in$ *defer* $(m\downarrow)$ A q **and**
x-non-eq-a: $x \neq a$ **and**
rev-p-defer-xa: $xa \in$ *defer* $(m\downarrow)$ A p
have *reject-and-defer*:
$(A -$ *elect* m A q, *elect* m A $q)$ = *snd* $((m\downarrow)$ A $q)$
by *force*
have *elect-p-eq-defer-rev-p*: *elect* m A p = *defer* $(m\downarrow)$ A p
by *simp*
hence *elect-a-in-p*: $a \in$ *elect* m A p
using *rev-p-defer-a*
by *presburger*
have *elect* m A $q \neq \{a\}$
using *rev-q-defer-x x-non-eq-a*
by *force*
with *assms*
show $xa \in$ *defer* $(m\downarrow)$ A q
using *a-lifted rev-p-defer-xa snd-conv elect-a-in-p*

elect-p-eq-defer-rev-p reject-and-defer
unfolding *invariant-monotonicity-def*
by (*metis* (*no-types*))
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$ **and**
x :: $'a$ **and**
xa :: $'a$
assume
rev-p-defer-a: $a \in$ *defer* $(m\downarrow)$ A p **and**
a-lifted: *lifted* A p q a **and**
rev-q-defer-xa: $xa \in$ *defer* $(m\downarrow)$ A q
from *assms*
show $xa \in$ *defer* $(m\downarrow)$ A p
using *a-lifted empty-iff insertE rev-p-defer-a rev-q-defer-xa snd-conv revision-composition.elims*
unfolding *invariant-monotonicity-def*
by *metis*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$ **and**
x :: $'a$ **and**
xa :: $'a$
assume
rev-p-defer-a: $a \in$ *defer* $(m\downarrow)$ A p **and**
a-lifted: *lifted* A p q a **and**
rev-q-not-defer-a: $a \notin$ *defer* $(m\downarrow)$ A q
from *assms*
have *lifted-inv*:
$\forall\ A\ p\ q\ a.\ a \in \mathit{elect}\ m\ A\ p \wedge \mathit{lifted}\ A\ p\ q\ a \longrightarrow$
$\mathit{elect}\ m\ A\ q = \mathit{elect}\ m\ A\ p \vee \mathit{elect}\ m\ A\ q = \{a\}$
unfolding *invariant-monotonicity-def*
by (*metis* (*no-types*))
have *p-defer-rev-eq-elect*: *defer* $(m\downarrow)$ A p = *elect* m A p
by *simp*
have *q-defer-rev-eq-elect*: *defer* $(m\downarrow)$ A q = *elect* m A q
by *simp*
thus $xa \in$ *defer* $(m\downarrow)$ A q
using *p-defer-rev-eq-elect lifted-inv a-lifted rev-p-defer-a rev-q-not-defer-a*
by *blast*
qed

end

4.3 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.

4.3.1 Definition

```
fun sequential-composition :: 'a Electoral-Module ⇒ 'a Electoral-Module ⇒
        'a Electoral-Module where
  sequential-composition m n A p =
    (let new-A = defer m A p;
        new-p = limit-profile new-A p in (
                (elect m A p) ∪ (elect n new-A new-p),
                (reject m A p) ∪ (reject n new-A new-p),
                defer n new-A new-p))

abbreviation sequence ::
  'a Electoral-Module ⇒ 'a Electoral-Module ⇒ 'a Electoral-Module
    (infix ▷ 50) where
  m ▷ n == sequential-composition m n

lemma seq-comp-presv-disj:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module and
    A :: 'a set and
    p :: 'a Profile
  assumes module-m: electoral-module m and
          module-n: electoral-module n and
          f-prof:  finite-profile A p
  shows disjoint3 ((m ▷ n) A p)
proof −
  let ?new-A = defer m A p
  let ?new-p = limit-profile ?new-A p
  have fin-def: finite (defer m A p)
    using def-presv-fin-prof f-prof module-m
    by metis
  have prof-def-lim:
```

profile (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
using *def-presv-fin-prof f-prof module-m*
by *metis*
have *defer-in-A*:
$\forall$ *prof f a A*.
(*profile A prof* $\wedge$ *finite A* $\wedge$ *electoral-module f* $\wedge$
(*a*::*'a*) $\in$ *defer f A prof*) $\longrightarrow$
a $\in$ *A*
using *UnCI result-presv-alts*
by (*metis* (*mono-tags*))
from *module-m f-prof*
have *disjoint-m*: *disjoint3* (*m A p*)
unfolding *electoral-module-def well-formed.simps*
by *blast*
from *module-m module-n def-presv-fin-prof f-prof*
have *disjoint-n*:
(*disjoint3* (*n ?new-A ?new-p*))
unfolding *electoral-module-def well-formed.simps*
by *metis*
have *disj-n*:
elect m A p $\cap$ *reject m A p* = {} $\wedge$
elect m A p $\cap$ *defer m A p* = {} $\wedge$
reject m A p $\cap$ *defer m A p* = {}
using *f-prof module-m*
by (*simp add*: *result-disj*)
from *f-prof module-m module-n*
have *rej-n-in-def-m*:
reject n (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*) $\subseteq$ *defer m A p*
using *def-presv-fin-prof reject-in-alts*
by *metis*
with *disjoint-m module-m module-n f-prof*
have *0*:
(*elect m A p* $\cap$ *reject n ?new-A ?new-p*) = {}
using *disj-n*
by (*simp add*: *disjoint-iff-not-equal subset-eq*)
from *f-prof module-m module-n*
have *elec-n-in-def-m*:
elect n (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*) $\subseteq$ *defer m A p*
using *def-presv-fin-prof elect-in-alts*
by *metis*
from *disjoint-m disjoint-n def-presv-fin-prof f-prof*
module-m module-n
have *1*:
(*elect m A p* $\cap$ *defer n ?new-A ?new-p*) = {}
proof –
obtain *sf* :: *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a* **where**
$\forall$ *a b*.

$(\exists\ c.\ c \in b \wedge (\exists\ d.\ d \in a \wedge c = d)) =$
$(sf\ a\ b \in b \wedge$
$(\exists\ e.\ e \in a \wedge sf\ a\ b = e))$
by *moura*
then obtain $sf2 :: 'a\ set \Rightarrow 'a\ set \Rightarrow 'a$ **where**
$\forall\ A\ B.$
$(A \cap B \neq \{\} \vee (\forall\ a.\ a \notin A \vee (\forall\ b.\ b \notin B \vee a \neq b))) \wedge$
$(A \cap B = \{\} \vee sf\ B\ A \in A \wedge sf2\ B\ A \in B \wedge$
$sf\ B\ A = sf2\ B\ A)$
by *auto*
thus *?thesis*
using *defer-in-A disj-n fin-def module-n prof-def-lim*
by (*metis* (*no-types*))
qed
from *disjoint-m disjoint-n def-presv-fin-prof f-prof*
module-m module-n
have *2*:
(*reject m A p* $\cap$ *reject n ?new-A ?new-p*) $= \{\}$
using *disjoint-iff-not-equal reject-in-alts*
set-rev-mp result-disj Int-Un-distrib2
Un-Diff-Int boolean-algebra-cancel.inf2
inf.order-iff inf-sup-aci(*1*) *subsetD*
rej-n-in-def-m disj-n
by *auto*
have $\forall\ A\ A'.\ \neg\ (A{::}'a\ set) \subseteq A' \vee A = A \cap A'$
by *blast*
with *disjoint-m disjoint-n def-presv-fin-prof f-prof*
module-m module-n elec-n-in-def-m
have *3*:
(*reject m A p* $\cap$ *elect n ?new-A ?new-p*) $= \{\}$
using *disj-n*
by *blast*
have
(*elect m A p* $\cup$ *elect n ?new-A ?new-p*) $\cap$
(*reject m A p* $\cup$ *reject n ?new-A ?new-p*) $= \{\}$
proof (*safe*)
fix $x :: 'a$
assume
elec-x: $x \in$ *elect m A p* **and**
rej-x: $x \in$ *reject m A p*
from *elec-x rej-x*
have $x \in$ *elect m A p* $\cap$ *reject m A p*
by *simp*
thus $x \in \{\}$
using *disj-n*
by *simp*
next
fix $x :: 'a$
assume

elec-x: $x \in$ *elect m A p* **and**
rej-lim-x:
$x \in$ *reject n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*)
from *elec-x rej-lim-x*
show $x \in \{\}$
using *0*
by *blast*
next
fix $x :: 'a$
assume
elec-lim-x:
$x \in$ *elect n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) **and**
rej-x: $x \in$ *reject m A p*
from *elec-lim-x rej-x*
show $x \in \{\}$
using *3*
by *blast*
next
fix $x :: 'a$
assume
elec-lim-x:
$x \in$ *elect n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) **and**
rej-lim-x:
$x \in$ *reject n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
from *elec-lim-x rej-lim-x*
show $x \in \{\}$
using *disjoint-iff-not-equal elec-lim-x fin-def*
module-n prof-def-lim rej-lim-x result-disj
by *metis*
qed
moreover from *0 1 2 3 disjoint-n module-m module-n f-prof*
have
(*elect m A p* $\cup$ *elect n ?new-A ?new-p*) $\cap$
(*defer n ?new-A ?new-p*) $= \{\}$
using *Int-Un-distrib2 Un-empty def-presv-fin-prof result-disj*
by *metis*
moreover from *0 1 2 3 f-prof disjoint-m disjoint-n module-m module-n*
have
(*reject m A p* $\cup$ *reject n ?new-A ?new-p*) $\cap$
(*defer n ?new-A ?new-p*) $= \{\}$
proof (*safe*)
fix $x :: 'a$
assume
elec-rej-disj:
elect m A p $\cap$
reject n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) $= \{\}$ **and**
elec-def-disj:
elect m A p $\cap$

defer n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
rej-rej-disj:
reject m A p ∩
reject n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
rej-elec-disj:
reject m A p ∩
elect n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
disj-p: *disjoint3* (*m A p*) **and**
disj-limit:
disjoint3 (*n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)) **and**
mod-m: *electoral-module m* **and**
mod-n: *electoral-module n* **and**
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
x-in-def:
x ∈ *defer n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) **and**
x-in-rej: *x* ∈ *reject m A p*
from *x-in-def*
have *x* ∈ *defer m A p*
using *defer-in-A fin-def module-n prof-def-lim*
by *blast*
with *x-in-rej*
have *x* ∈ *reject m A p* ∩ *defer m A p*
by *fastforce*
thus *x* ∈ {}
using *disj-n*
by *blast*
next
fix *x* :: *'a*
assume
elec-rej-disj:
elect m A p ∩
reject n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
elec-def-disj:
elect m A p ∩
defer n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
rej-rej-disj:
reject m A p ∩
reject n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
rej-elec-disj:
reject m A p ∩
elect n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) = {} **and**
disj-p: *disjoint3* (*m A p*) **and**
disj-limit:
disjoint3 (*n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)) **and**
mod-m: *electoral-module m* **and**
mod-n: *electoral-module n* **and**
fin-A: *finite A* **and**
prof-A: *profile A p* **and**

x-in-def:
$x \in$ *defer n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) **and**
x-in-rej:
$x \in$ *reject n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
from *x-in-def x-in-rej*
show $x \in \{\}$
using *fin-def module-n prof-def-lim reject-not-elec-or-def*
by *fastforce*
qed
ultimately have
disjoint3 (*elect m A p* $\cup$ *elect n ?new-A ?new-p*,
reject m A p $\cup$ *reject n ?new-A ?new-p*,
defer n ?new-A ?new-p)
by *simp*
thus *?thesis*
unfolding *sequential-composition.simps*
by *metis*
qed

lemma *seq-comp-presv-alts*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes *module-m*: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows *set-equals-partition A* ((*m* $\triangleright$ *n*) *A p*)
proof –
let *?new-A* = *defer m A p*
let *?new-p* = *limit-profile ?new-A p*
from *module-m f-prof*
have *set-equals-partition A* (*m A p*)
unfolding *electoral-module-def*
by *simp*
with *module-m f-prof*
have *0*:
elect m A p $\cup$ *reject m A p* $\cup$ *?new-A* = *A*
by (*simp add*: *result-presv-alts*)
from *module-n def-presv-fin-prof f-prof module-m*
have
set-equals-partition ?new-A (*n ?new-A ?new-p*)
unfolding *electoral-module-def well-formed.simps*
by *metis*
with *module-m module-n f-prof*
have *1*:
elect n ?new-A ?new-p $\cup$
reject n ?new-A ?new-p $\cup$

defer n ?new-A ?new-p = ?new-A
using *def-presv-fin-prof result-presv-alts*
by *metis*
from *0 1*
have
(*elect m A p* ∪ *elect n ?new-A ?new-p*) ∪
(*reject m A p* ∪ *reject n ?new-A ?new-p*) ∪
defer n ?new-A ?new-p = *A*
by *blast*
hence
set-equals-partition A
(*elect m A p* ∪ *elect n ?new-A ?new-p*,
reject m A p ∪ *reject n ?new-A ?new-p*,
defer n ?new-A ?new-p)
by *simp*
thus *?thesis*
unfolding *sequential-composition.simps*
by *metis*
qed

4.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n*
shows *electoral-module* (*m* ▷ *n*)
proof (*unfold electoral-module-def*, *safe*)
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
fin-A: *finite A* **and**
prof-A: *profile A p*
have ∀ *r*. *well-formed* (*A*::*'a set*) *r* =
(*disjoint3 r* ∧ *set-equals-partition A r*)
by *simp*
thus *well-formed A* ((*m* ▷ *n*) *A p*)
using *module-m module-n seq-comp-presv-disj*
seq-comp-presv-alts fin-A prof-A
by *metis*
qed

4.3.3 Lemmas

```
lemma seq-comp-dec-only-def:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module and
    A :: 'a set and
    p :: 'a Profile
  assumes
    module-m: electoral-module m and
    module-n: electoral-module n and
    f-prof: finite-profile A p and
    empty-defer: defer m A p = {}
  shows (m ▷ n) A p = m A p
proof
  have
    ∀ f A prof.
      (electoral-module f ∧ finite-profile A prof) ⟶
        finite-profile (defer f A prof)
          (limit-profile (defer f A prof) prof)
    using def-presv-fin-prof
    by metis
  hence prof-no-alt:
    profile {} (limit-profile (defer m A p) p)
    using empty-defer f-prof module-m
    by metis
  hence
    (elect m A p) ∪
      (elect n (defer m A p)
        (limit-profile (defer m A p) p))
    = elect m A p
    using elect-in-alts empty-defer module-n
    by auto
  thus elect (m ▷ n) A p = elect m A p
    using fst-conv
    unfolding sequential-composition.simps
    by metis
next
  have rej-empty:
    ∀ f prof.
      (electoral-module f ∧ profile ({}::'a set) prof) ⟶
        reject f {} prof = {}
    using bot.extremum-uniqueI infinite-imp-nonempty reject-in-alts
    by metis
  have prof-no-alt: profile {} (limit-profile (defer m A p) p)
    using empty-defer f-prof module-m limit-profile-sound
    by auto
  hence (reject m A p, defer n {} (limit-profile {} p)) = snd (m A p)
    using bot.extremum-uniqueI defer-in-alts empty-defer
        infinite-imp-nonempty module-n prod.collapse
```

by (*metis* (*no-types*))
thus *snd* $((m \triangleright n)\ A\ p) = $ *snd* $(m\ A\ p)$
using *rej-empty empty-defer module-n prof-no-alt*
by *simp*
qed

lemma *seq-comp-def-then-elect*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a 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* $A\ p$
shows *elect* $(m \triangleright n)\ A\ p = $ *defer* $m\ A\ p$
proof (*cases*)
assume $A = \{\}$
with *electing-n n-electing-m f-prof*
show *?thesis*
using *bot.extremum-uniqueI defer-in-alts elect-in-alts seq-comp-sound*
unfolding *electing-def non-electing-def*
by *metis*
next
assume *assm*: $A \neq \{\}$
from *n-electing-m f-prof*
have *ele*: *elect* $m\ A\ p = \{\}$
unfolding *non-electing-def*
by *simp*
from *assm def-one-m f-prof finite*
have *def-card*:
card (*defer* $m\ A\ p) = 1$
unfolding *defers-def*
by (*simp add*: *Suc-leI card-gt-0-iff*)
with *n-electing-m f-prof*
have *def*:
$\exists\ a \in A.$ *defer* $m\ A\ p = \{a\}$
using *card-1-singletonE defer-in-alts singletonI subsetCE*
unfolding *non-electing-def*
by *metis*
from *ele def n-electing-m*
have *rej*:
$\exists\ a \in A.$ *reject* $m\ A\ p = A - \{a\}$
using *Diff-empty def-one-m f-prof reject-not-elec-or-def*
unfolding *defers-def*
by *metis*
from *ele rej def n-electing-m f-prof*

have *res-m*:
$\exists\ a \in A.\ m\ A\ p = (\{\},\ A - \{a\},\ \{a\})$
using *Diff-empty combine-ele-rej-def reject-not-elec-or-def*
unfolding *non-electing-def*
by *metis*
hence
$\exists\ a \in A.\ elect\ (m \triangleright n)\ A\ p =$
$elect\ n\ \{a\}\ (limit\text{-}profile\ \{a\}\ p)$
using *prod.sel*(*1*, *2*) *sup-bot.left-neutral*
unfolding *sequential-composition.simps*
by *metis*
with *def-card def electing-n n-electing-m f-prof*
have
$\exists\ a \in A.\ elect\ (m \triangleright n)\ A\ p = \{a\}$
using *electing-for-only-alt prod.sel*(*1*) *def-presv-fin-prof sup-bot.left-neutral*
unfolding *non-electing-def sequential-composition.simps*
by *metis*
with *def def-card electing-n n-electing-m f-prof res-m*
show *?thesis*
using *def-presv-fin-prof electing-for-only-alt fst-conv sup-bot.left-neutral*
unfolding *non-electing-def sequential-composition.simps*
by *metis*
qed

lemma *seq-comp-def-card-bounded*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows $card\ (defer\ (m \triangleright n)\ A\ p) \leq card\ (defer\ m\ A\ p)$
using *card-mono defer-in-alts module-m module-n f-prof def-presv-fin-prof snd-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-set-bounded*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*

shows *defer* $(m \triangleright n)$ *A p* $\subseteq$ *defer m A p*
using *defer-in-alts module-m module-n prod.sel*(*2*) *f-prof def-presv-fin-prof*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-defers-def-set*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows
defer $(m \triangleright n)$ *A p* =
defer n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
using *snd-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-then-elect-elec-set*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows
elect $(m \triangleright n)$ *A p* =
elect n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) $\cup$
(*elect m A p*)
using *Un-commute fst-conv*
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-elim-one-red-def-set*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module m* **and**
module-n: *eliminates 1 n* **and**

f-prof: *finite-profile A p* **and**
enough-leftover: *card* (*defer m A p*) > *1*
shows *defer* ($m \triangleright n$) *A p* ⊂ *defer m A p*
using *enough-leftover module-m module-n f-prof snd-conv*
def-presv-fin-prof single-elim-imp-red-def-set
unfolding *sequential-composition.simps*
by *metis*

lemma *seq-comp-def-set-sound*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
e-mod-m: *electoral-module m* **and**
e-mod-n: *electoral-module n* **and**
fin-prof-p: *finite-profile A p*
shows *defer* ($m \triangleright n$) *A p* ⊆ *defer m A p*
proof (*safe*)
fix *x* :: *'a*
assume *x* ∈ *defer* ($m \triangleright n$) *A p*
thus *x* ∈ *defer m A p*
using *e-mod-m e-mod-n fin-prof-p in-mono seq-comp-def-set-bounded*
by (*metis* (*no-types*, *lifting*))
qed

lemma *seq-comp-def-set-trans*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes
a ∈ (*defer* ($m \triangleright n$) *A p*) **and**
electoral-module m ∧ *electoral-module n* **and**
finite-profile A p
shows *a* ∈ *defer n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) ∧ *a* ∈ *defer m*
A p
using *seq-comp-def-set-bounded assms*
in-mono seq-comp-defers-def-set
by (*metis* (*no-types*, *opaque-lifting*))

4.3.4 Composition Rules

The sequential composition preserves the non-blocking property.

theorem *seq-comp-presv-non-blocking*[*simp*]:

```
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module
  assumes
    non-blocking-m: non-blocking m and
    non-blocking-n: non-blocking n
  shows non-blocking (m ▷ n)
proof –
  fix
    A :: 'a set and
    p :: 'a Profile
  let ?input-sound = ((A::'a set) ≠ {} ∧ finite-profile A p)
  from non-blocking-m have
    ?input-sound ⟶ reject m A p ≠ A
    unfolding non-blocking-def
    by simp
  with non-blocking-m have 0:
    ?input-sound ⟶ A – reject m A p ≠ {}
    using Diff-eq-empty-iff reject-in-alts subset-antisym
    unfolding non-blocking-def
    by metis
  from non-blocking-m have
    ?input-sound ⟶ well-formed A (m A p)
    unfolding electoral-module-def non-blocking-def
    by simp
  hence
    ?input-sound ⟶
        elect m A p ∪ defer m A p = A – reject m A p
    using non-blocking-m elec-and-def-not-rej
    unfolding non-blocking-def
    by metis
  with 0 have
    ?input-sound ⟶ elect m A p ∪ defer m A p ≠ {}
    by simp
  hence ?input-sound ⟶ (elect m A p ≠ {} ∨ defer m A p ≠ {})
    by simp
  with non-blocking-m non-blocking-n
  show ?thesis
  proof (unfold non-blocking-def)
    assume
      emod-reject-m:
      electoral-module m ∧
        (∀ A p. A ≠ {} ∧ finite-profile A p ⟶
          reject m A p ≠ A) and
      emod-reject-n:
      electoral-module n ∧
        (∀ A p. A ≠ {} ∧ finite-profile A p ⟶
          reject n A p ≠ A)
    show
```

electoral-module $(m \triangleright n) \wedge$
  $(\forall\ A\ p.$
    $A \neq \{\} \wedge$ *finite-profile* $A\ p \longrightarrow$
      reject $(m \triangleright n)\ A\ p \neq A)$
proof (*safe*)
  show *electoral-module* $(m \triangleright n)$
    using *emod-reject-m emod-reject-n*
    by *simp*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    x :: $'a$
  assume
    fin-A: *finite* A **and**
    prof-A: *profile* $A\ p$ **and**
    rej-mn: *reject* $(m \triangleright n)\ A\ p = A$ **and**
    x-in-A: $x \in A$
  from *emod-reject-m fin-A prof-A*
  have *fin-defer*:
    finite-profile (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p)
    using *def-presv-fin-prof*
    by (*metis* (*no-types*))
  from *emod-reject-m emod-reject-n fin-A prof-A*
  have *seq-elect*:
    elect $(m \triangleright n)\ A\ p =$
      elect n (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p) $\cup$
        elect $m\ A\ p$
    using *seq-comp-def-then-elect-elec-set*
    by *metis*
  from *emod-reject-n emod-reject-m fin-A prof-A*
  have *def-limit*:
    defer $(m \triangleright n)\ A\ p =$
      defer n (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p)
    using *seq-comp-defers-def-set*
    by *metis*
  from *emod-reject-n emod-reject-m fin-A prof-A*
  have
    elect $(m \triangleright n)\ A\ p \cup$ *defer* $(m \triangleright n)\ A\ p = A -$ *reject* $(m \triangleright n)\ A\ p$
    using *elec-and-def-not-rej seq-comp-sound*
    by *metis*
  hence *elect-def-disj*:
    elect n (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p) $\cup$
      elect $m\ A\ p\ \cup$
      defer n (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p) $= \{\}$
    using *def-limit seq-elect Diff-cancel rej-mn*
    by *auto*
  have *rej-def-eq-set*:
    defer n (*defer* $m\ A\ p$) (*limit-profile* (*defer* $m\ A\ p$) p) $-$

```
        defer n (defer m A p) (limit-profile (defer m A p) p) = {} ⟶
          reject n (defer m A p) (limit-profile (defer m A p) p) =
            defer m A p
      using elect-def-disj emod-reject-n fin-defer
      by (simp add: reject-not-elec-or-def)
    have
      defer n (defer m A p) (limit-profile (defer m A p) p) −
        defer n (defer m A p) (limit-profile (defer m A p) p) = {} ⟶
          elect m A p = elect m A p ∩ defer m A p
      using elect-def-disj
      by blast
    thus x ∈ {}
      using rej-def-eq-set result-disj fin-defer Diff-cancel Diff-empty
            emod-reject-m emod-reject-n fin-A prof-A reject-not-elec-or-def x-in-A
      by metis
  qed
 qed
qed
```

Sequential composition preserves the non-electing property.

```
theorem seq-comp-presv-non-electing[simp]:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module
  assumes
    m-elect: non-electing m and
    n-elect: non-electing n
  shows non-electing (m ▷ n)
proof (unfold non-electing-def, safe)
  from m-elect n-elect
  have electoral-module m ∧ electoral-module n
    unfolding non-electing-def
    by blast
  thus electoral-module (m ▷ n)
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    x :: 'a
  assume
    finite A and
    profile A p and
    x ∈ elect (m ▷ n) A p
  with m-elect n-elect
  show x ∈ {}
    unfolding non-electing-def
    using seq-comp-def-then-elect-elec-set def-presv-fin-prof
          Diff-empty Diff-partition empty-subsetI
```

by *metis*
qed

Composing an electoral module that defers exactly 1 alternative in sequence after an electoral module that is electing results (still) in an electing electoral module.

theorem *seq-comp-electing*[*simp*]:
  fixes
    m :: *'a Electoral-Module* **and**
    n :: *'a Electoral-Module*
  assumes
    def-one-m: *defers 1 m* **and**
    electing-n: *electing n*
  shows *electing* ($m \triangleright n$)
proof −
  have $\forall$ *A p*. (*card A* $\geq$ *1* $\wedge$ *finite-profile A p*) $\longrightarrow$ *card* (*defer m A p*) = *1*
    using *def-one-m*
    unfolding *defers-def*
    by *blast*
  hence *def-m1-not-empty*:
    $\forall$ *A p*. (*A* $\neq$ {} $\wedge$ *finite-profile A p*) $\longrightarrow$ *defer m A p* $\neq$ {}
    using *One-nat-def Suc-leI card-eq-0-iff*
        card-gt-0-iff zero-neq-one
    by *metis*
  thus *?thesis*
  proof −
    obtain
      f-set ::
      (*'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$ *'a Result*) $\Rightarrow$ *'a set* **and**
      f-prof ::
      (*'a set* $\Rightarrow$ *'a Profile* $\Rightarrow$ *'a Result*) $\Rightarrow$ *'a Profile* **where**
      f-mod:
      $\forall$ *f*.
        ($\neg$ *electing f* $\vee$ *electoral-module f* $\wedge$
          ($\forall$ *A prof*. (*A* $\neq$ {} $\wedge$ *finite A* $\wedge$ *profile A prof*) $\longrightarrow$ *elect f A prof* $\neq$ {}))
$\wedge$
        (*electing f* $\vee$ $\neg$ *electoral-module f* $\vee$ *f-set f* $\neq$ {} $\wedge$ *finite* (*f-set f*) $\wedge$
          profile (*f-set f*) (*f-prof f*) $\wedge$ *elect f* (*f-set f*) (*f-prof f*) = {})
      unfolding *electing-def*
      by *moura*
    hence *f-elect*:
      electoral-module n $\wedge$
        ($\forall$ *A prof*. (*A* $\neq$ {} $\wedge$ *finite A* $\wedge$ *profile A prof*) $\longrightarrow$ *elect n A prof* $\neq$ {})
      using *electing-n*
      by *metis*
    have *def-card-one*:
      electoral-module m $\wedge$
        ($\forall$ *A prof*.
          (*1* $\leq$ *card A* $\wedge$ *finite A* $\wedge$ *profile A prof*) $\longrightarrow$

card (*defer m A prof*) = *1*)
using *def-one-m*
unfolding *defers-def*
by *blast*
hence *electoral-module* ($m \triangleright n$)
using *f-elect seq-comp-sound*
by *metis*
with *f-mod f-elect def-card-one*
show *?thesis*
using *seq-comp-def-then-elect-elec-set def-presv-fin-prof def-m1-not-empty bot-eq-sup-iff*
by *metis*
qed
qed

lemma *def-lift-inv-seq-comp-help*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-n: *defer-lift-invariance n* **and**
def-and-lifted: $a \in$ (*defer* ($m \triangleright n$) *A p*) $\wedge$ *lifted A p q a*
shows ($m \triangleright n$) *A p* = ($m \triangleright n$) *A q*
proof −
let *?new-Ap* = *defer m A p*
let *?new-Aq* = *defer m A q*
let *?new-p* = *limit-profile ?new-Ap p*
let *?new-q* = *limit-profile ?new-Aq q*
from *monotone-m monotone-n*
have *modules*:
electoral-module m $\wedge$ *electoral-module n*
unfolding *defer-lift-invariance-def*
by *simp*
hence *finite-profile A p* $\longrightarrow$ *defer* ($m \triangleright n$) *A p* $\subseteq$ *defer m A p*
using *seq-comp-def-set-bounded*
by *metis*
moreover have *profile-p*: *lifted A p q a* $\longrightarrow$ *finite-profile A p*
unfolding *lifted-def*
by *simp*
ultimately have *defer-subset*: *defer* ($m \triangleright n$) *A p* $\subseteq$ *defer m A p*
using *def-and-lifted*
by *blast*
hence *mono-m*: *m A p* = *m A q*
using *monotone-m def-and-lifted modules profile-p*

```
      seq-comp-def-set-trans
  unfolding defer-lift-invariance-def
  by metis
hence new-A-eq: ?new-Ap = ?new-Aq
  by presburger
have defer-eq:
  defer (m ▷ n) A p = defer n ?new-Ap ?new-p
  using snd-conv
  unfolding sequential-composition.simps
  by metis
hence mono-n:
  n ?new-Ap ?new-p = n ?new-Aq ?new-q
proof (cases)
  assume lifted ?new-Ap ?new-p ?new-q a
  thus ?thesis
    using defer-eq mono-m monotone-n def-and-lifted
    unfolding defer-lift-invariance-def
    by (metis (no-types, lifting))
next
  assume a2: ¬lifted ?new-Ap ?new-p ?new-q a
  from def-and-lifted
  have finite-profile A q
    unfolding lifted-def
    by simp
  with modules new-A-eq
  have 1:
    finite-profile ?new-Ap ?new-q
    using def-presv-fin-prof
    by (metis (no-types))
  moreover from modules profile-p def-and-lifted
  have 0:
    finite-profile ?new-Ap ?new-p
    using def-presv-fin-prof
    by (metis (no-types))
  moreover from defer-subset def-and-lifted
  have 2: a ∈ ?new-Ap
    by blast
  moreover from def-and-lifted
  have eql-lengths:
    length ?new-p = length ?new-q
    unfolding lifted-def
    by simp
  ultimately have 0:
    (∀ i::nat. i < length ?new-p ⟶
        ¬Preference-Relation.lifted ?new-Ap (?new-p!i) (?new-q!i) a) ∨
     (∃ i::nat. i < length ?new-p ∧
        ¬Preference-Relation.lifted ?new-Ap (?new-p!i) (?new-q!i) a ∧
          (?new-p!i) ≠ (?new-q!i))
    using a2
```

unfolding *lifted-def*
by (*metis* (*no-types*, *lifting*))
from *def-and-lifted modules*
have
$\forall$ *i*. (*0* $\leq$ *i* $\wedge$ *i* < *length ?new-p*) $\longrightarrow$
(*Preference-Relation.lifted A* (*p*!*i*) (*q*!*i*) *a* $\vee$ (*p*!*i*) = (*q*!*i*))
using *limit-prof-presv-size*
unfolding *Profile.lifted-def*
by *metis*
with *def-and-lifted modules mono-m*
have
$\forall$ *i*. (*0* $\leq$ *i* $\wedge$ *i* < *length ?new-p*) $\longrightarrow$
(*Preference-Relation.lifted ?new-Ap* (*?new-p*!*i*) (*?new-q*!*i*) *a* $\vee$
(*?new-p*!*i*) = (*?new-q*!*i*))
using *limit-lifted-imp-eq-or-lifted defer-in-alts*
limit-prof-presv-size nth-map
unfolding *Profile.lifted-def limit-profile.simps*
by (*metis* (*no-types*, *lifting*))
with *0 eql-lengths mono-m*
show *?thesis*
using *leI not-less-zero nth-equalityI*
by *metis*
qed
from *mono-m mono-n*
show *?thesis*
unfolding *sequential-composition.simps*
by (*metis* (*full-types*))
qed

Sequential composition preserves the property defer-lift-invariance.

theorem *seq-comp-presv-def-lift-inv*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module*
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-n: *defer-lift-invariance n*
shows *defer-lift-invariance* (*m* $\triangleright$ *n*)
using *monotone-m monotone-n def-lift-inv-seq-comp-help*
seq-comp-sound defer-lift-invariance-def
by (*metis* (*full-types*))

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 Electoral-Module* **and**
n :: *'a Electoral-Module*

assumes
non-blocking-m: *non-blocking m* **and**
non-electing-m: *non-electing m* **and**
def-1-n: *defers 1 n*
shows *defers 1* $(m \triangleright n)$
proof (*unfold defers-def*, *safe*)
have *electoral-mod-m*: *electoral-module m*
using *non-electing-m*
unfolding *non-electing-def*
by *simp*
have *electoral-mod-n*: *electoral-module n*
using *def-1-n*
unfolding *defers-def*
by *simp*
show *electoral-module* $(m \triangleright n)$
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
pos-card: $1 \leq$ *card A* **and**
fin-A: *finite A* **and**
prof-A: *profile A p*
from *pos-card* **have**
$A \neq \{\}$
by *auto*
with *fin-A prof-A* **have** *m-non-blocking*:
reject m A p $\neq A$
using *non-blocking-m*
unfolding *non-blocking-def*
by *simp*
hence
$\exists\ a.\ a \in A \land a \notin$ *reject m A p*
using *pos-card non-electing-m*
reject-in-alts subset-antisym subset-iff
fin-A prof-A subsetI
unfolding *non-electing-def*
by *slow*
hence *defer m A p* $\neq \{\}$
using *electoral-mod-defer-elem empty-iff pos-card*
non-electing-m fin-A prof-A
unfolding *non-electing-def*
by (*metis* (*no-types*))
hence *defer-non-empty*:
card (*defer m A p*) ≥ 1
using *Suc-leI card-gt-0-iff pos-card fin-A prof-A*
non-blocking-m def-presv-fin-prof

```
    unfolding One-nat-def non-blocking-def
    by metis
  have defer-fun:
    defer (m ▷ n) A p =
      defer n (defer m A p) (limit-profile (defer m A p) p)
    using def-1-n fin-A non-blocking-m prof-A seq-comp-defers-def-set
    unfolding defers-def non-blocking-def
    by (metis (no-types, opaque-lifting))
  have
    ∀ n f. defers n f =
      (electoral-module f ∧
        (∀ A prof.
          (¬ n ≤ card (A::'a set) ∨ infinite A ∨
            ¬ profile A prof) ∨
          card (defer f A prof) = n))
    unfolding defers-def
    by blast
  hence
    card (defer n (defer m A p)
      (limit-profile (defer m A p) p)) = 1
    using defer-non-empty def-1-n fin-A prof-A
          non-blocking-m def-presv-fin-prof
    unfolding non-blocking-def
    by metis
  thus card (defer (m ▷ n) A p) = 1
    using defer-fun
    by simp
qed
```

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

```
theorem disj-compat-seq[simp]:
  fixes
    m :: 'a Electoral-Module and
    m' :: 'a Electoral-Module and
    n :: 'a Electoral-Module
  assumes
    compatible: disjoint-compatibility m n and
    module-m': electoral-module m'
  shows disjoint-compatibility (m ▷ m') n
proof (unfold disjoint-compatibility-def, safe)
  show electoral-module (m ▷ m')
    using compatible module-m' seq-comp-sound
    unfolding disjoint-compatibility-def
    by metis
next
  show electoral-module n
    using compatible
```

unfolding *disjoint-compatibility-def*
by *metis*
next
fix S :: $'a$ *set*
assume *fin-S*: *finite* S
have *modules*:
electoral-module $(m \triangleright m')$ $\wedge$ *electoral-module* n
using *compatible module-m′ seq-comp-sound*
unfolding *disjoint-compatibility-def*
by *metis*
obtain A **where** A:
$A \subseteq S \wedge$
$(\forall\ a \in A.$ *indep-of-alt* $m\ S\ a \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $m\ S\ p)) \wedge$
$(\forall\ a \in S - A.$ *indep-of-alt* $n\ S\ a \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $n\ S\ p))$
using *compatible fin-S*
unfolding *disjoint-compatibility-def*
by (*metis* (*no-types*, *lifting*))
show
$\exists\ A \subseteq S.$
$(\forall\ a \in A.$ *indep-of-alt* $(m \triangleright m')\ S\ a \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $(m \triangleright m')\ S\ p)) \wedge$
$(\forall\ a \in S - A.$ *indep-of-alt* $n\ S\ a \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $n\ S\ p))$
proof
have
$\forall\ a\ p\ q.$
$a \in A \wedge$ *equiv-prof-except-a* $S\ p\ q\ a \longrightarrow$
$(m \triangleright m')\ S\ p = (m \triangleright m')\ S\ q$
proof (*safe*)
fix
a :: $'a$ **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile*
assume
a: $a \in A$ **and**
b: *equiv-prof-except-a* $S\ p\ q\ a$
have *eq-def*:
defer $m\ S\ p =$ *defer* $m\ S\ q$
using $A\ a\ b$
unfolding *indep-of-alt-def*
by *metis*
from $a\ b$
have *profiles*:
finite-profile $S\ p\ \wedge$ *finite-profile* $S\ q$
unfolding *equiv-prof-except-a-def*
by *simp*
hence (*defer* $m\ S\ p) \subseteq S$

using *compatible defer-in-alts*
unfolding *disjoint-compatibility-def*
by *metis*
hence
limit-profile (*defer m S p*) *p* =
limit-profile (*defer m S q*) *q*
using *A DiffD2 a b compatible defer-not-elec-or-rej profiles negl-diff-imp-eq-limit-prof*
unfolding *disjoint-compatibility-def eq-def*
by (*metis* (*no-types*, *lifting*))
with *eq-def*
have m' (*defer m S p*) (*limit-profile* (*defer m S p*) *p*) =
m' (*defer m S q*) (*limit-profile* (*defer m S q*) *q*)
by *simp*
moreover have $m\ S\ p = m\ S\ q$
using *A a b*
unfolding *indep-of-alt-def*
by *metis*
ultimately show $(m \triangleright m')\ S\ p = (m \triangleright m')\ S\ q$
unfolding *sequential-composition.simps*
by (*metis* (*full-types*))
qed
moreover have
$\forall\ a \in A.\ \forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $(m \triangleright m')\ S\ p$
using *A UnI1 prod.sel*
unfolding *sequential-composition.simps*
by *metis*
ultimately show
$A \subseteq S\ \wedge$
$(\forall\ a \in A.$ *indep-of-alt* $(m \triangleright m')\ S\ a\ \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $(m \triangleright m')\ S\ p))\ \wedge$
$(\forall\ a \in S - A.$ *indep-of-alt* $n\ S\ a\ \wedge$
$(\forall\ p.$ *finite-profile* $S\ p \longrightarrow a \in$ *reject* $n\ S\ p))$
using *A indep-of-alt-def modules*
by (*metis* (*mono-tags*, *lifting*))
qed
qed

Composing a defer-lift invariant and a non-electing electoral module that defers exactly one alternative in sequence with an electing electoral module results in a monotone electoral module.

theorem *seq-comp-mono*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module*
assumes
def-monotone-m: *defer-lift-invariance m* **and**
non-ele-m: *non-electing m* **and**
def-one-m: *defers 1 m* **and**

electing-n: *electing n*
shows *monotonicity* $(m \triangleright n)$
proof (*unfold monotonicity-def*, *safe*)
have *electoral-mod-m*: *electoral-module m*
using *non-ele-m*
unfolding *non-electing-def*
by *simp*
have *electoral-mod-n*: *electoral-module n*
using *electing-n*
unfolding *electing-def*
by *simp*
show *electoral-module* $(m \triangleright n)$
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
w :: *'a*
assume
fin-A: *finite A* **and**
elect-w-in-p: $w \in$ *elect* $(m \triangleright n)$ *A p* **and**
lifted-w: *Profile.lifted A p q w*
have *finite-profile A p* $\wedge$ *finite-profile A q*
using *lifted-w*
unfolding *lifted-def*
by *metis*
thus $w \in$ *elect* $(m \triangleright n)$ *A q*
using *seq-comp-def-then-elect elect-w-in-p lifted-w*
def-monotone-m non-ele-m def-one-m electing-n
unfolding *defer-lift-invariance-def*
by *metis*
qed

Composing a defer-invariant-monotone electoral module in sequence before a non-electing, defer-monotone electoral module that defers exactly 1 alternative results in a defer-lift-invariant electoral module.

theorem *def-inv-mono-imp-def-lift-inv*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module*
assumes
strong-def-mon-m: *defer-invariant-monotonicity m* **and**
non-electing-n: *non-electing n* **and**
defers-1: *defers 1 n* **and**
defer-monotone-n: *defer-monotonicity n*
shows *defer-lift-invariance* $(m \triangleright n)$
proof (*unfold defer-lift-invariance-def*, *safe*)

have *electoral-mod-m*: *electoral-module m*
using *strong-def-mon-m*
unfolding *defer-invariant-monotonicity-def*
by *metis*
have *electoral-mod-n*: *electoral-module n*
using *defers-1*
unfolding *defers-def*
by *metis*
show *electoral-module* $(m \triangleright n)$
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$
assume
defer-a-p: $a \in$ *defer* $(m \triangleright n)$ A p **and**
lifted-a: *Profile.lifted* A p q a
from *strong-def-mon-m*
have *non-electing-m*: *non-electing m*
unfolding *defer-invariant-monotonicity-def*
by *simp*
have *electoral-mod-m*: *electoral-module m*
using *strong-def-mon-m*
unfolding *defer-invariant-monotonicity-def*
by *metis*
have *electoral-mod-n*: *electoral-module n*
using *defers-1*
unfolding *defers-def*
by *metis*
have *finite-profile-q*: *finite-profile* A q
using *lifted-a*
unfolding *Profile.lifted-def*
by *simp*
have *finite-profile-p*: *profile* A p
using *lifted-a*
unfolding *Profile.lifted-def*
by *simp*
show $(m \triangleright n)$ A p $=$ $(m \triangleright n)$ A q
proof (*cases*)
assume *not-unchanged*: *defer* m A $q \neq$ *defer* m A p
from *not-unchanged*
have *a-single-defer*: $\{a\}$ $=$ *defer* m A q
using *strong-def-mon-m electoral-mod-n defer-a-p*
lifted-a seq-comp-def-set-trans finite-profile-p
finite-profile-q
unfolding *defer-invariant-monotonicity-def*

by *metis*
moreover have
$\{a\} = defer\ m\ A\ q \longrightarrow defer\ (m \triangleright n)\ A\ q \subseteq \{a\}$
using *finite-profile-q electoral-mod-m electoral-mod-n seq-comp-def-set-sound*
by (*metis* (*no-types*, *opaque-lifting*))
ultimately have
$(a \in defer\ m\ A\ p) \longrightarrow defer\ (m \triangleright n)\ A\ q \subseteq \{a\}$
by *simp*
moreover have *def-card-one*:
$(a \in defer\ m\ A\ p) \longrightarrow card\ (defer\ (m \triangleright n)\ A\ q) = 1$
using *a-single-defer card-eq-0-iff card-insert-disjoint defers-1 electoral-mod-m empty-iff finite.emptyI seq-comp-defers-def-set order-refl def-presv-fin-prof finite-profile-q*
unfolding *One-nat-def defers-def*
by *metis*
moreover have *defer-a-in-m-p*:
$a \in defer\ m\ A\ p$
using *electoral-mod-m electoral-mod-n defer-a-p seq-comp-def-set-bounded finite-profile-p finite-profile-q*
by *blast*
ultimately have
$defer\ (m \triangleright n)\ A\ q = \{a\}$
using *Collect-mem-eq card-1-singletonE empty-Collect-eq insertCI subset-singletonD*
by *metis*
moreover have
$defer\ (m \triangleright n)\ A\ p = \{a\}$
proof (*safe*)
fix $x :: 'a$
assume
defer-x: $x \in defer\ (m \triangleright n)\ A\ p$ **and**
x-exists: $x \notin \{\}$
have *fin-defer*:
$\forall\ f\ (A::'a\ set)\ prof.$
$(electoral\text{-}module\ f \wedge finite\ A \wedge profile\ A\ prof) \longrightarrow$
$finite\text{-}profile\ (defer\ f\ A\ prof)$
$(limit\text{-}profile\ (defer\ f\ A\ prof)\ prof)$
using *def-presv-fin-prof*
by (*metis* (*no-types*))
have $finite\text{-}profile\ (defer\ m\ A\ p)\ (limit\text{-}profile\ (defer\ m\ A\ p)\ p)$
using *electoral-mod-m finite-profile-p finite-profile-q fin-defer*
by *blast*
hence $Suc\ (card\ (defer\ m\ A\ p - \{a\})) = card\ (defer\ m\ A\ p)$
using *card-Suc-Diff1 defer-a-in-m-p*
by *metis*
hence *min-card*:

Suc 0 $\leq$ *card* (*defer m A p*)
by *linarith*
have *emod-n-then-mn*:
electoral-module n $\longrightarrow$ *electoral-module* ($m \triangleright n$)
using *electoral-mod-m*
by *simp*
have *defers* (*Suc 0*) *n*
using *defers-1*
by *simp*
hence *defer-card-one*:
electoral-module n $\wedge$
($\forall$ *A prof*.
(*Suc 0* $\leq$ *card A* $\wedge$ *finite A* $\wedge$ *profile A prof*) $\longrightarrow$
card (*defer n A prof*) = *Suc 0*)
unfolding *defers-def*
by *simp*
hence *emod-mn*: *electoral-module* ($m \triangleright n$)
using *emod-n-then-mn*
by *blast*
have *nat-diff*:
$\forall$ (*i::nat*) *j*. $i \leq j \longrightarrow i - j = 0$
by *auto*
have *nat-comp*:
$\forall$ (*i::nat*) *j k*.
$i \leq j \wedge j \leq k \vee$
$j \leq i \wedge i \leq k \vee$
$i \leq k \wedge k \leq j \vee$
$k \leq j \wedge j \leq i \vee$
$j \leq k \wedge k \leq i \vee$
$k \leq i \wedge i \leq j$
using *le-cases3*
by *linarith*
have *fin-diff-card*:
$\forall$ *A a*.
(*finite A* $\wedge$ (*a::'a*) $\in$ *A*) $\longrightarrow$
card ($A - \{a\}$) = *card A* $- 1$
using *card-Diff-singleton*
by *metis*
with *fin-defer defer-card-one min-card*
have *card* (*defer* ($m \triangleright n$) *A p*) = *Suc 0*
using *electoral-mod-m seq-comp-defers-def-set*
finite-profile-p finite-profile-q
by *metis*
with *fin-diff-card nat-comp nat-diff emod-mn fin-defer*
have $\{a\} = \{x\}$
using *One-nat-def card-1-singletonE singletonD*
defer-a-p defer-x
by *metis*
thus $x = a$

by *force*
next
show $a \in$ *defer* $(m \triangleright n)$ A p
using *defer-a-p*
by *linarith*
qed
ultimately have *defer* $(m \triangleright n)$ A p $=$ *defer* $(m \triangleright n)$ A q
by *blast*
moreover have *elect* $(m \triangleright n)$ A p $=$ *elect* $(m \triangleright n)$ A q
using *finite-profile-p finite-profile-q*
non-electing-m non-electing-n
seq-comp-presv-non-electing
non-electing-def
by *metis*
thus *?thesis*
using *calculation eq-def-and-elect-imp-eq*
electoral-mod-m electoral-mod-n
finite-profile-p seq-comp-sound
finite-profile-q
by *metis*
next
assume *not-different-alternatives*:
$\neg($*defer* m A $q \neq$ *defer* m A $p)$
have *elect* m A $p = \{\}$
using *non-electing-m finite-profile-p finite-profile-q*
by (*simp add*: *non-electing-def*)
moreover have *elect* m A $q = \{\}$
using *non-electing-m finite-profile-q*
by (*simp add*: *non-electing-def*)
ultimately have *elect-m-equal*: *elect* m A p $=$ *elect* m A q
by *simp*
from *not-different-alternatives*
have *same-alternatives*: *defer* m A q $=$ *defer* m A p
by *simp*
hence
(*limit-profile* (*defer* m A p) p) $=$
(*limit-profile* (*defer* m A p) q) $\vee$
lifted (*defer* m A q)
(*limit-profile* (*defer* m A p) p)
(*limit-profile* (*defer* m A p) q) a
using *defer-in-alts electoral-mod-m*
lifted-a finite-profile-q
limit-prof-eq-or-lifted
by *metis*
thus *?thesis*
proof
assume
limit-profile (*defer* m A p) p $=$
limit-profile (*defer* m A p) q

hence *same-profile*:
limit-profile (*defer m A p*) *p* =
limit-profile (*defer m A q*) *q*
using *same-alternatives*
by *simp*
hence *results-equal-n*:
n (*defer m A q*) (*limit-profile* (*defer m A q*) *q*) =
n (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
by (*simp add*: *same-alternatives*)
moreover have *results-equal-m*: *m A p* = *m A q*
using *elect-m-equal same-alternatives*
finite-profile-p finite-profile-q
by (*simp add*: *electoral-mod-m eq-def-and-elect-imp-eq*)
hence $(m \triangleright n)$ *A p* = $(m \triangleright n)$ *A q*
using *same-profile*
by *auto*
thus *?thesis*
by *blast*
next
assume *still-lifted*:
lifted (*defer m A q*) (*limit-profile* (*defer m A p*) *p*)
(*limit-profile* (*defer m A p*) *q*) *a*
hence *a-in-def-p*:
a ∈ *defer n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*)
using *electoral-mod-m electoral-mod-n*
finite-profile-p defer-a-p
seq-comp-def-set-trans
finite-profile-q
by *metis*
hence *a-still-deferred-p*:
{*a*} ⊆ *defer n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*)
by *simp*
have *card-le-1-p*: *card* (*defer m A p*) ≥ *1*
using *One-nat-def Suc-leI card-gt-0-iff*
electoral-mod-m electoral-mod-n
equals0D finite-profile-p defer-a-p
seq-comp-def-set-trans def-presv-fin-prof
finite-profile-q
by *metis*
hence
card (*defer n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*)) = *1*
using *defers-1 electoral-mod-m*
finite-profile-p def-presv-fin-prof
finite-profile-q
unfolding *defers-def*
by *metis*

hence *def-set-is-a-p*:
  $\{a\}$ = *defer n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*)
  using *a-still-deferred-p card-1-singletonE*
      insert-subset singletonD
  by *metis*
have *a-still-deferred-q*:
  $a \in$ *defer n* (*defer m A q*)
    (*limit-profile* (*defer m A p*) *q*)
  using *still-lifted a-in-def-p*
      defer-monotone-n electoral-mod-m
      same-alternatives
      def-presv-fin-prof finite-profile-q
  unfolding *defer-monotonicity-def*
  by *metis*
have *card* (*defer m A q*) $\geq$ *1*
  using *card-le-1-p same-alternatives*
  by *simp*
hence
  card (*defer n* (*defer m A q*)
    (*limit-profile* (*defer m A q*) *q*)) = *1*
  using *defers-1 electoral-mod-m*
      finite-profile-q def-presv-fin-prof
  unfolding *defers-def*
  by *metis*
hence *def-set-is-a-q*:
  $\{a\}$ =
    defer n (*defer m A q*)
      (*limit-profile* (*defer m A q*) *q*)
  using *a-still-deferred-q card-1-singletonE*
      same-alternatives singletonD
  by *metis*
have
  defer n (*defer m A p*)
    (*limit-profile* (*defer m A p*) *p*) =
      defer n (*defer m A q*)
        (*limit-profile* (*defer m A q*) *q*)
  using *def-set-is-a-q def-set-is-a-p*
  by *auto*
thus *?thesis*
  using *seq-comp-presv-non-electing*
      eq-def-and-elect-imp-eq non-electing-def
      finite-profile-p finite-profile-q
      non-electing-m non-electing-n
      seq-comp-defers-def-set
  by *metis*
qed
qed
qed

end

4.4 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.

4.4.1 Definition

fun *parallel-composition* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒
    'a Aggregator ⇒ *'a Electoral-Module* **where**
  parallel-composition m n agg A p = *agg A* (*m A p*) (*n A p*)

abbreviation *parallel* :: *'a Electoral-Module* ⇒ *'a Aggregator* ⇒
    'a Electoral-Module ⇒ *'a Electoral-Module*
   (- $\|_{-}$ - [*50*, *1000*, *51*] *50*) **where**
  $m \|_a n$ == *parallel-composition m n a*

4.4.2 Soundness

theorem *par-comp-sound*[*simp*]:
  fixes
    m :: *'a Electoral-Module* **and**
    n :: *'a Electoral-Module* **and**
    a :: *'a Aggregator*
  assumes
    mod-m: *electoral-module m* **and**
    mod-n: *electoral-module n* **and**
    agg-a: *aggregator a*
  shows *electoral-module* ($m \|_a n$)
proof (*unfold electoral-module-def*, *safe*)
  fix
    A :: *'a set* **and**
    p :: *'a Profile*
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p*
  have *wf-quant*:
    ∀ *agg*. *aggregator agg* =

```
    (∀ A' e r d e' r' d'.
      (¬ well-formed (A'::'a set) (e, r', d) ∨
        ¬ well-formed A' (r, d', e')) ∨
      well-formed A'
        (agg A' (e, r', d) (r, d', e')))
    unfolding aggregator-def
    by blast
  have wf-imp:
    ∀ m' A' p'.
      (electoral-module m' ∧ finite (A'::'a set) ∧
        profile A' p') ⟶
        well-formed A' (m' A' p')
    using par-comp-result-sound
    by (metis (no-types))
  from mod-m mod-n fin-A prof-A agg-a
  have well-formed A (a A (m A p) (n A p))
    using agg-a combine-ele-rej-def fin-A
          mod-m mod-n prof-A wf-imp wf-quant
    by metis
  thus well-formed A ((m ∥a n) A p)
    by simp
qed
```

4.4.3 Composition Rule

Using a conservative aggregator, the parallel composition preserves the property non-electing.

```
theorem conserv-agg-presv-non-electing[simp]:
  fixes
    m :: 'a Electoral-Module and
    n :: 'a Electoral-Module and
    a :: 'a Aggregator
  assumes
    non-electing-m: non-electing m and
    non-electing-n: non-electing n and
    conservative: agg-conservative a
  shows non-electing (m ∥a n)
proof (unfold non-electing-def, safe)
  have emod-m: electoral-module m
    using non-electing-m
    unfolding non-electing-def
    by simp
  have emod-n: electoral-module n
    using non-electing-n
    unfolding non-electing-def
    by simp
  have agg-a: aggregator a
    using conservative
    unfolding agg-conservative-def
```

by *simp*
thus *electoral-module* $(m \parallel_a n)$
using *emod-m emod-n agg-a par-comp-sound*
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$
assume
fin-A: *finite* A **and**
prof-A: *profile* A p **and**
w-wins: $w \in$ *elect* $(m \parallel_a n)$ A p
have *emod-m*: *electoral-module* m
using *non-electing-m*
unfolding *non-electing-def*
by *simp*
have *emod-n*: *electoral-module* n
using *non-electing-n*
unfolding *non-electing-def*
by *simp*
have
$\forall$ r r' d d' e e' A' f.
(*well-formed* $(A'::'a$ *set*$)$ (e', r', d') $\wedge$ *well-formed* A' (e, r, d) $\longrightarrow$
elect-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e\ \wedge$
reject-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r\ \wedge$
defer-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d) =$
$((\neg$ *well-formed* A' (e', r', d') $\vee$ $\neg$ *well-formed* A' $(e, r, d))$ $\vee$
elect-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq e' \cup e\ \wedge$
reject-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq r' \cup r\ \wedge$
defer-r $(f\ A'\ (e', r', d')\ (e, r, d)) \subseteq d' \cup d)$
by *linarith*
hence
$\forall$ *agg*. *agg-conservative agg* $=$
(*aggregator agg* $\wedge$
$(\forall$ A' e e' d d' r r'. $(\neg$ *well-formed* $(A'::'a$ *set*$)$ (e, r, d) $\vee$
$\neg$ *well-formed* A' $(e', r', d'))$ $\vee$
elect-r $(agg\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e'\ \wedge$
reject-r $(agg\ A'\ (e, r, d)\ (e', r', d')) \subseteq r \cup r'\ \wedge$
defer-r $(agg\ A'\ (e, r, d)\ (e', r', d')) \subseteq d \cup d'))$
unfolding *agg-conservative-def*
by *simp*
hence
aggregator a $\wedge$
$(\forall$ A' e e' d d' r r'. $\neg$ *well-formed* A' (e, r, d) $\vee$
$\neg$ *well-formed* A' (e', r', d') $\vee$
elect-r $(a\ A'\ (e, r, d)\ (e', r', d')) \subseteq e \cup e'\ \wedge$
reject-r $(a\ A'\ (e, r, d)\ (e', r', d')) \subseteq r \cup r'\ \wedge$
defer-r $(a\ A'\ (e, r, d)\ (e', r', d')) \subseteq d \cup d')$

using *conservative*
by *presburger*
hence
let $c = (a\ A\ (m\ A\ p)\ (n\ A\ p))$ *in*
$(\mathit{elect\text{-}r}\ c \subseteq ((\mathit{elect}\ m\ A\ p) \cup (\mathit{elect}\ n\ A\ p)))$
using *emod-m emod-n fin-A par-comp-result-sound prod.collapse prof-A*
by *metis*
hence $w \in ((\mathit{elect}\ m\ A\ p) \cup (\mathit{elect}\ n\ A\ p))$
using *w-wins*
by *auto*
thus $w \in \{\}$
using *sup-bot-right fin-A prof-A non-electing-m non-electing-n*
unfolding *non-electing-def*
by (*metis* (*no-types*, *lifting*))
qed

end

4.5 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 (1) the termination condition is met or (2) no new decisions are made (i.e., a fixed point is reached).

4.5.1 Definition

lemma *loop-termination-helper*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
not-term: $\neg t\ (acc\ A\ p)$ **and**
subset: $\mathit{defer}\ (acc \triangleright m)\ A\ p \subset \mathit{defer}\ acc\ A\ p$ **and**

not-inf: $\neg$*infinite* (*defer acc A p*)
shows
((*acc* $\triangleright$ *m*, *m*, *t*, *A*, *p*), (*acc*, *m*, *t*, *A*, *p*)) $\in$
measure (λ(*acc*, *m*, *t*, *A*, *p*). *card* (*defer acc A p*))
using *assms psubset-card-mono*
by *simp*

This function handles the accumulator for the following loop composition function.

function *loop-comp-helper* ::
'a Electoral-Module $\Rightarrow$ *'a Electoral-Module* $\Rightarrow$
'a Termination-Condition $\Rightarrow$ *'a Electoral-Module* **where**
t (*acc A p*) $\vee$ $\neg$((*defer* (*acc* $\triangleright$ *m*) *A p*) $\subset$ (*defer acc A p*)) $\vee$
infinite (*defer acc A p*) $\Longrightarrow$
loop-comp-helper acc m t A p = *acc A p* |
$\neg$ (*t* (*acc A p*) $\vee$ $\neg$((*defer* (*acc* $\triangleright$ *m*) *A p*) $\subset$ (*defer acc A p*)) $\vee$
infinite (*defer acc A p*)) $\Longrightarrow$
loop-comp-helper acc m t A p = *loop-comp-helper* (*acc* $\triangleright$ *m*) *m t A p*
proof –
fix
P :: *bool* **and**
x :: (*'a Electoral-Module*) $\times$ (*'a Electoral-Module*) $\times$
(*'a Termination-Condition*) $\times$ *'a set* $\times$ *'a Profile*
have *x-exists*: $\exists$ *f A p p2 g*. (*g*, *f*, *p*, *A*, *p2*) = *x*
using *prod-cases5*
by *metis*
assume
a1: $\bigwedge$ *t acc A p m*.
t (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$ $\neg$ *finite* (*defer acc A p*) $\Longrightarrow$
x = (*acc*, *m*, *t*, *A*, *p*) $\Longrightarrow$ *P* **and**
a2: $\bigwedge$ *t acc A p m*.
$\neg$ (*t* (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$ $\neg$ *finite* (*defer acc A p*)) $\Longrightarrow$
x = (*acc*, *m*, *t*, *A*, *p*) $\Longrightarrow$ *P*
thus *P*
using *x-exists*
by (*metis* (*no-types*))
next
show
$\bigwedge$ *t acc A p m ta acca Aa pa ma*.
t (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
$\neg$ *finite* (*defer acc A p*) $\Longrightarrow$
ta (*acca Aa pa*) $\vee$ $\neg$ *defer* (*acca* $\triangleright$ *ma*) *Aa pa* $\subset$ *defer acca Aa pa* $\vee$
$\neg$ *finite* (*defer acca Aa pa*) $\Longrightarrow$
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*) $\Longrightarrow$
acc A p = *acca Aa pa*
by *fastforce*
next

show
$\bigwedge$ *t acc A p m ta acca Aa pa ma.*
t (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
infinite (*defer acc A p*) $\Longrightarrow$
$\neg$ (*ta* (*acca Aa pa*) $\vee$ $\neg$ *defer* (*acca* $\triangleright$ *ma*) *Aa pa* $\subset$ *defer acca Aa pa* $\vee$
infinite (*defer acca Aa pa*)) $\Longrightarrow$
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*) $\Longrightarrow$
acc A p = *loop-comp-helper-sumC* (*acca* $\triangleright$ *ma*, *ma*, *ta*, *Aa*, *pa*)
proof −
fix
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
m :: *'a Electoral-Module* **and**
ta :: *'a Termination-Condition* **and**
acca :: *'a Electoral-Module* **and**
Aa :: *'a set* **and**
pa :: *'a Profile* **and**
ma :: *'a Electoral-Module*
assume
a1: *t* (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
infinite (*defer acc A p*) **and**
a2: $\neg$ (*ta* (*acca Aa pa*) $\vee$ $\neg$ *defer* (*acca* $\triangleright$ *ma*) *Aa pa* $\subset$ *defer acca Aa pa* $\vee$
infinite (*defer acca Aa pa*)) **and**
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*)
hence *False*
using *a2 a1*
by *force*
thus *acc A p* = *loop-comp-helper-sumC* (*acca* $\triangleright$ *ma*, *ma*, *ta*, *Aa*, *pa*)
by *auto*
qed
next
show
$\bigwedge$ *t acc A p m ta acca Aa pa ma.*
$\neg$ (*t* (*acc A p*) $\vee$ $\neg$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\vee$
infinite (*defer acc A p*)) $\Longrightarrow$
$\neg$ (*ta* (*acca Aa pa*) $\vee$ $\neg$ *defer* (*acca* $\triangleright$ *ma*) *Aa pa* $\subset$ *defer acca Aa pa* $\vee$
infinite (*defer acca Aa pa*)) $\Longrightarrow$
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*) $\Longrightarrow$
loop-comp-helper-sumC (*acc* $\triangleright$ *m*, *m*, *t*, *A*, *p*) =
loop-comp-helper-sumC (*acca* $\triangleright$ *ma*, *ma*, *ta*, *Aa*, *pa*)
by *force*
qed
termination
proof −
have *func-term*:
$\exists$ *r*. *wf r* $\wedge$
($\forall$ *p f* (*A*::*'a set*) *prof g*.

p (f A prof) ∨
¬ defer (f ⊳ g) A prof ⊂ defer f A prof ∨
infinite (defer f A prof) ∨
((f ⊳ g, g, p, A, prof), (f, g, p, A, prof)) ∈ r)
using loop-termination-helper wf-measure termination
by (metis (no-types))
hence
∀ r p.
Ex ((λ ra. ∀ f (A::'a set) prof pa g.
∃ prof' pb p-rel pc pd h (B::'a set) prof'' i pe.
¬ wf r ∨
loop-comp-helper-dom
(p::('a Electoral-Module) × (- Electoral-Module) ×
(- Termination-Condition) × - set × - Profile) ∨
infinite (defer f A prof) ∨
pa (f A prof) ∧
wf
(prof'::((
('a Electoral-Module) × ('a Electoral-Module) ×
('a Termination-Condition) × 'a set × 'a Profile) × -) set) ∧
¬ loop-comp-helper-dom (pb::
('a Electoral-Module) × (- Electoral-Module) ×
(- Termination-Condition) × - set × - Profile) ∨
wf p-rel ∧ ¬ defer (f ⊳ g) A prof ⊂ defer f A prof ∧
¬ loop-comp-helper-dom
(pc::('a Electoral-Module) × (- Electoral-Module) ×
(- Termination-Condition) × - set × - Profile) ∨
((f ⊳ g, g, pa, A, prof), f, g, pa, A, prof) ∈ p-rel ∧ wf p-rel ∧
¬ loop-comp-helper-dom
(pd::('a Electoral-Module) × (- Electoral-Module) ×
(- Termination-Condition) × - set × - Profile) ∨
finite (defer h B prof'') ∧
defer (h ⊳ i) B prof'' ⊂ defer h B prof'' ∧
¬ pe (h B prof'') ∧
((h ⊳ i, i, pe, B, prof''), h, i, pe, B, prof'') ∉ r)::
((('a Electoral-Module) × ('a Electoral-Module) ×
('a Termination-Condition) × 'a set × 'a Profile) ×
('a Electoral-Module) × ('a Electoral-Module) ×
('a Termination-Condition) × 'a set × 'a Profile) set ⇒ bool)
by metis
obtain
p-rel :: ((('a Electoral-Module) × ('a Electoral-Module) ×
('a Termination-Condition) × 'a set × 'a Profile) ×
('a Electoral-Module) × ('a Electoral-Module) ×
('a Termination-Condition) × 'a set × 'a Profile) set **where**
wf p-rel ∧
(∀ p f A prof g. p (f A prof) ∨
¬ defer (f ⊳ g) A prof ⊂ defer f A prof ∨
infinite (defer f A prof) ∨

$((f \triangleright g,\ g,\ p,\ A,\ prof),\ f,\ g,\ p,\ A,\ prof) \in p\text{-}rel)$
using *func-term*
by *presburger*
thus *?thesis*
using *termination*
by *metis*
qed

lemma *loop-comp-code-helper*[*code*]:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
shows
loop-comp-helper acc m t A p =
(*if* (t (*acc A p*) $\vee$ $\neg$((*defer* (*acc* $\triangleright$ m) *A p*) $\subset$ (*defer acc A p*)) $\vee$
infinite (*defer acc A p*))
then (*acc A p*) *else* (*loop-comp-helper* (*acc* $\triangleright$ m) *m t A p*))
by *simp*

function *loop-composition* ::
'a Electoral-Module $\Rightarrow$ *'a Termination-Condition* $\Rightarrow$ *'a Electoral-Module* **where**
t ({}, {}, A) $\Longrightarrow$ *loop-composition m t A p* = *defer-module A p* |
$\neg$(t ({}, {}, A)) $\Longrightarrow$ *loop-composition m t A p* = (*loop-comp-helper m m t*) *A p*
by (*fastforce*, *simp-all*)
termination
using *termination wf-empty*
by *blast*

abbreviation *loop* ::
'a Electoral-Module $\Rightarrow$ *'a Termination-Condition* $\Rightarrow$ *'a Electoral-Module*
(- $\circlearrowleft_{-}$ *50*) **where**
$m \circlearrowleft_t$ $\equiv$ *loop-composition m t*

lemma *loop-comp-code*[*code*]:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
A :: *'a set* **and**
p :: *'a Profile*
shows
loop-composition m t A p =
(*if* (t ({},{},A)) *then* (*defer-module A p*) *else* (*loop-comp-helper m m t*) *A p*)
by *simp*

lemma *loop-comp-helper-imp-partit*:
fixes

m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
n :: *nat*
assumes
module-m: *electoral-module m* **and**
profile: *finite-profile A p*
shows
electoral-module acc ∧ (*n* = *card* (*defer acc A p*)) ⟹
well-formed A (*loop-comp-helper acc m t A p*)
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
have
∀ (*f*::*'a set* ⇒ *'a Profile* ⇒ *'a Result*) *g*.
(*electoral-module f* ∧ *electoral-module g*) ⟶
electoral-module (*f* ▷ *g*)
by *auto*
hence *electoral-module* (*acc* ▷ *m*)
using *less.prems module-m*
by *metis*
hence *wf-acc*:
¬ *t* (*acc A p*) ∧ ¬ *t* (*acc A p*) ∧
defer (*acc* ▷ *m*) *A p* ⊂ *defer acc A p* ∧
finite (*defer acc A p*) ⟶
well-formed A (*loop-comp-helper acc m t A p*)
using *less.hyps less.prems loop-comp-helper.simps*(*2*)
psubset-card-mono
by *metis*
have *well-formed A* (*acc A p*)
using *less.prems profile*
unfolding *electoral-module-def*
by *blast*
thus *?case*
using *wf-acc loop-comp-helper.simps*(*1*)
by (*metis* (*no-types*))
qed

4.5.2 Soundness

theorem *loop-comp-sound*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition*
assumes *electoral-module m*
shows *electoral-module* ($m \circlearrowleft_t$)
using *def-mod-sound loop-composition.simps*(*1*, *2*) *loop-comp-helper-imp-partit*
assms

unfolding *electoral-module-def*
by *metis*

lemma *loop-comp-helper-imp-no-def-incr*:
fixes
m :: $'a$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
acc :: $'a$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
n :: *nat*
assumes
module-m: *electoral-module* m **and**
profile: *finite-profile* A p
shows
$(electoral\text{-}module\ acc \wedge n = card\ (defer\ acc\ A\ p)) \Longrightarrow$
$defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p \subseteq defer\ acc\ A\ p$
proof (*induct arbitrary*: *acc rule*: *less-induct*)
case (*less*)
have *emod-acc-m*: *electoral-module* $(acc \triangleright m)$
using *less.prems module-m*
by *simp*
have $\forall\ A\ A'.\ infinite\ (A::'a\ set) \vee \neg A' \subset A \vee card\ A' < card\ A$
using *psubset-card-mono*
by *metis*
hence
$\neg\ t\ (acc\ A\ p) \wedge defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p \wedge$
$finite\ (defer\ acc\ A\ p) \longrightarrow$
$defer\ (loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t)\ A\ p \subseteq defer\ acc\ A\ p$
using *emod-acc-m less.hyps less.prems*
by *blast*
hence
$\neg\ t\ (acc\ A\ p) \wedge defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p \wedge$
$finite\ (defer\ acc\ A\ p) \longrightarrow$
$defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p \subseteq defer\ acc\ A\ p$
using *loop-comp-helper.simps*(*2*)
by (*metis* (*no-types*))
thus *?case*
using *eq-iff loop-comp-helper.simps*(*1*)
by (*metis* (*no-types*))
qed

4.5.3 Lemmas

lemma *loop-comp-helper-def-lift-inv-helper*:
fixes
m :: $'a$ *Electoral-Module* **and**
t :: $'a$ *Termination-Condition* **and**
acc :: $'a$ *Electoral-Module* **and**

A :: *'a set* **and**
p :: *'a Profile*
assumes
monotone-m: *defer-lift-invariance m* **and**
f-prof: *finite-profile A p*
shows
(*defer-lift-invariance acc* ∧ *n* = *card* (*defer acc A p*)) ⟶
(∀ *q a*.
(*a* ∈ (*defer* (*loop-comp-helper acc m t*) *A p*) ∧
lifted A p q a) ⟶
(*loop-comp-helper acc m t*) *A p* =
(*loop-comp-helper acc m t*) *A q*)
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
case (*less n*)
have *defer-card-comp*:
defer-lift-invariance acc ⟶
(∀ *q a*. (*a* ∈ (*defer* (*acc* ▷ *m*) *A p*) ∧ *lifted A p q a*) ⟶
card (*defer* (*acc* ▷ *m*) *A p*) = *card* (*defer* (*acc* ▷ *m*) *A q*))
using *monotone-m def-lift-inv-seq-comp-help*
by *metis*
have *defer-card-acc*:
defer-lift-invariance acc ⟶
(∀ *q a*. (*a* ∈ (*defer* (*acc*) *A p*) ∧ *lifted A p q a*) ⟶
card (*defer* (*acc*) *A p*) = *card* (*defer* (*acc*) *A q*))
unfolding *defer-lift-invariance-def*
by *simp*
hence *defer-card-acc-2*:
defer-lift-invariance acc ⟶
(∀ *q a*. (*a* ∈ (*defer* (*acc* ▷ *m*) *A p*) ∧ *lifted A p q a*) ⟶
card (*defer* (*acc*) *A p*) = *card* (*defer* (*acc*) *A q*))
using *monotone-m f-prof seq-comp-def-set-trans*
unfolding *defer-lift-invariance-def*
by *metis*
thus *?case*
proof (*cases*)
assume *card-unchanged*: *card* (*defer* (*acc* ▷ *m*) *A p*) = *card* (*defer acc A p*)
with *defer-card-comp defer-card-acc monotone-m*
have
defer-lift-invariance (*acc*) ⟶
(∀ *q a*. (*a* ∈ (*defer* (*acc*) *A p*) ∧ *lifted A p q a*) ⟶
(*loop-comp-helper acc m t*) *A q* = *acc A q*)
proof (*safe*)
fix
q :: *'a Profile* **and**
a :: *'a*
assume
def-card-eq:
card (*defer* (*acc* ▷ *m*) *A p*) = *card* (*defer acc A p*) **and**
dli-acc: *defer-lift-invariance acc* **and**

```
      def-seq-lift-card:
      ∀ q a. a ∈ defer (acc ▷ m) A p ∧ Profile.lifted A p q a ⟶
        card (defer (acc ▷ m) A p) = card (defer (acc ▷ m) A q) and
      a-in-def-acc: a ∈ defer acc A p and
      lifted-A: Profile.lifted A p q a
    have emod-m: electoral-module m
      using monotone-m
      unfolding defer-lift-invariance-def
      by simp
    have emod-acc: electoral-module acc
      using dli-acc
      unfolding defer-lift-invariance-def
      by simp
    have acc-eq-pq: acc A q = acc A p
      using a-in-def-acc dli-acc lifted-A
      unfolding defer-lift-invariance-def
      by (metis (full-types))
    with emod-acc emod-m
    have
      finite (defer acc A p) ⟶
        loop-comp-helper acc m t A q = acc A q
      using a-in-def-acc def-card-eq def-seq-lift-card
            dual-order.strict-iff-order f-prof lifted-A
            loop-comp-code-helper psubset-card-mono
            seq-comp-def-set-bounded
      by (metis (no-types))
    thus loop-comp-helper acc m t A q = acc A q
      using acc-eq-pq loop-comp-code-helper
      by (metis (full-types))
  qed
  moreover from card-unchanged
  have (loop-comp-helper acc m t) A p = acc A p
    using loop-comp-helper.simps(1) order.strict-iff-order psubset-card-mono
    by metis
  ultimately have
    (defer-lift-invariance (acc ▷ m) ∧ defer-lift-invariance acc) ⟶
      (∀ q a. (a ∈ (defer (loop-comp-helper acc m t) A p) ∧
          lifted A p q a) ⟶
            (loop-comp-helper acc m t) A p =
              (loop-comp-helper acc m t) A q)
    unfolding defer-lift-invariance-def
    by metis
  thus ?thesis
    using monotone-m seq-comp-presv-def-lift-inv
    by blast
next
  assume card-changed:
    ¬ (card (defer (acc ▷ m) A p) = card (defer acc A p))
  with f-prof seq-comp-def-card-bounded
```

have *card-smaller-for-p*:
  electoral-module (*acc*) $\longrightarrow$
    (*card* (*defer* (*acc* $\triangleright$ *m*) *A* *p*) $<$ *card* (*defer acc A p*))
  using *monotone-m order.not-eq-order-implies-strict*
  unfolding *defer-lift-invariance-def*
  by (*metis* (*full-types*))
with *defer-card-acc-2 defer-card-comp*
have *card-changed-for-q*:
  defer-lift-invariance (*acc*) $\longrightarrow$
    ($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A* *p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
      (*card* (*defer* (*acc* $\triangleright$ *m*) *A* *q*) $<$ *card* (*defer acc A q*)))
  unfolding *defer-lift-invariance-def*
  by (*metis* (*no-types*, *lifting*))
thus *?thesis*
proof (*cases*)
  assume *t-not-satisfied-for-p*: $\neg$ *t* (*acc A p*)
  hence *t-not-satisfied-for-q*:
    defer-lift-invariance (*acc*) $\longrightarrow$
      ($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A* *p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
        $\neg$ *t* (*acc A q*))
    using *monotone-m f-prof seq-comp-def-set-trans*
    unfolding *defer-lift-invariance-def*
    by *metis*
  from *card-changed defer-card-comp defer-card-acc*
  have *dli-card-def*:
    (*defer-lift-invariance* (*acc* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance* (*acc*)) $\longrightarrow$
      ($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A* *p*) $\wedge$ *Profile.lifted A p q a*) $\longrightarrow$
        card (*defer* (*acc* $\triangleright$ *m*) *A* *q*) $\neq$ (*card* (*defer acc A q*)))
  proof $-$
    have
      $\forall$ *f*.
        (*defer-lift-invariance f* $\vee$
          ($\exists$ *A prof prof2* (*a*::'*a*).
            f A prof $\neq$ *f A prof2* $\wedge$
              Profile.lifted A prof prof2 a $\wedge$
              a $\in$ *defer f A prof*) $\vee$ $\neg$ *electoral-module f*) $\wedge$
              (($\forall$ *A p1 p2 b*. *f A p1* $=$ *f A p2* $\vee$ $\neg$ *Profile.lifted A p1 p2 b* $\vee$
                b $\notin$ *defer f A p1*) $\wedge$
              electoral-module f $\vee$ $\neg$ *defer-lift-invariance f*)
      unfolding *defer-lift-invariance-def*
      by *blast*
    thus *?thesis*
      using *card-changed monotone-m f-prof seq-comp-def-set-trans*
      by (*metis* (*no-types*, *opaque-lifting*))
  qed
  hence *dli-def-subset*:
    defer-lift-invariance (*acc* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance* (*acc*) $\longrightarrow$
      ($\forall$ *q a*. (*a* $\in$ (*defer* (*acc* $\triangleright$ *m*) *A* *p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
        defer (*acc* $\triangleright$ *m*) *A* *q* $\subset$ *defer acc A q*)

proof –
  {
    fix
      alt :: *'a* **and**
      prof :: *'a Profile*
    have
      $(\neg$ *defer-lift-invariance* $($*acc* $\triangleright$ *m*$) \vee \neg$ *defer-lift-invariance acc*$) \vee$
        $($*alt* $\notin$ *defer* $($*acc* $\triangleright$ *m*$)$ *A p* $\vee \neg$ *lifted A p prof alt*$) \vee$
        defer $($*acc* $\triangleright$ *m*$)$ *A prof* $\subset$ *defer acc A prof*
      using *Profile.lifted-def dli-card-def defer-lift-invariance-def*
          monotone-m psubsetI seq-comp-def-set-bounded
      by (*metis* (*no-types*))
  }
  thus *?thesis*
    by *metis*
qed
with *t-not-satisfied-for-p*
have *rec-step-q*:
  $($*defer-lift-invariance* $($*acc* $\triangleright$ *m*$) \wedge$ *defer-lift-invariance* $($*acc*$)) \longrightarrow$
      $(\forall$ *q a*. $($*a* $\in ($*defer* $($*acc* $\triangleright$ *m*$)$ *A p*$) \wedge$ *lifted A p q a*$) \longrightarrow$
          loop-comp-helper acc m t A q =
            loop-comp-helper $($*acc* $\triangleright$ *m*$)$ *m t A q*$)$
proof (*safe*)
  fix
    q :: *'a Profile* **and**
    a :: *'a*
  assume
    a-in-def-impl-def-subset:
    $\forall$ *q a*. *a* $\in$ *defer* $($*acc* $\triangleright$ *m*$)$ *A p* $\wedge$ *lifted A p q a* $\longrightarrow$
      defer $($*acc* $\triangleright$ *m*$)$ *A q* $\subset$ *defer acc A q* **and**
    dli-acc: *defer-lift-invariance acc* **and**
    a-in-def-seq-acc-m: *a* $\in$ *defer* $($*acc* $\triangleright$ *m*$)$ *A p* **and**
    lifted-pq-a: *lifted A p q a*
  have *defer-subset-acc*:
    defer $($*acc* $\triangleright$ *m*$)$ *A q* $\subset$ *defer acc A q*
    using *a-in-def-impl-def-subset lifted-pq-a*
        a-in-def-seq-acc-m
    by *metis*
  have *electoral-module acc*
    using *dli-acc*
    unfolding *defer-lift-invariance-def*
    by *simp*
  hence *finite* $($*defer acc A q*$) \wedge \neg$ *t* $($*acc A q*$)$
    using *lifted-def dli-acc a-in-def-seq-acc-m*
        lifted-pq-a def-presv-fin-prof
        t-not-satisfied-for-q
    by *metis*
  with *defer-subset-acc*
  show

loop-comp-helper acc m t A q =
loop-comp-helper (*acc* ▷ *m*) *m t A q*
using *loop-comp-code-helper*
by *metis*
qed
have *rec-step-p*:
electoral-module acc ⟶
loop-comp-helper acc m t A p = *loop-comp-helper* (*acc* ▷ *m*) *m t A p*
proof (*safe*)
assume *emod-acc*: *electoral-module acc*
have *emod-implies-defer-subset*:
electoral-module m ⟶ *defer* (*acc* ▷ *m*) *A p* ⊆ *defer acc A p*
using *emod-acc f-prof seq-comp-def-set-bounded*
by *blast*
have *card-ineq*: *card* (*defer* (*acc* ▷ *m*) *A p*) < *card* (*defer acc A p*)
using *card-smaller-for-p emod-acc*
by *force*
have *fin-def-limited-acc*:
finite-profile (*defer acc A p*) (*limit-profile* (*defer acc A p*) *p*)
using *def-presv-fin-prof emod-acc f-prof*
by *metis*
have *defer* (*acc* ▷ *m*) *A p* ⊆ *defer acc A p*
using *emod-implies-defer-subset defer-lift-invariance-def monotone-m*
by *blast*
hence *defer* (*acc* ▷ *m*) *A p* ⊂ *defer acc A p*
using *fin-def-limited-acc card-ineq card-psubset*
by *metis*
with *fin-def-limited-acc*
show *loop-comp-helper acc m t A p* = *loop-comp-helper* (*acc* ▷ *m*) *m t A p*
using *loop-comp-code-helper t-not-satisfied-for-p*
by (*metis* (*no-types*))
qed
show *?thesis*
proof (*safe*)
fix
q :: *′a Profile* **and**
a :: *′a*
assume
dli-acc: *defer-lift-invariance acc* **and**
n-card-acc: *n* = *card* (*defer acc A p*) **and**
a-in-defer-lch: *a* ∈ *defer* (*loop-comp-helper acc m t*) *A p* **and**
a-lifted: *Profile.lifted A p q a*
hence *emod-acc*: *electoral-module acc*
unfolding *defer-lift-invariance-def*
by *metis*
have *defer-lift-invariance* (*acc* ▷ *m*) ∧ *a* ∈ *defer* (*acc* ▷ *m*) *A p*
using *a-in-defer-lch defer-lift-invariance-def dli-acc*
f-prof loop-comp-helper-imp-no-def-incr monotone-m
rec-step-p seq-comp-presv-def-lift-inv subsetD

by (*metis* (*no-types*))
with *emod-acc*
show *loop-comp-helper acc m t A p* = *loop-comp-helper acc m t A q*
using *a-in-defer-lch a-lifted card-smaller-for-p dli-acc*
less.hyps n-card-acc rec-step-p rec-step-q
by (*metis* (*full-types*))
qed
next
assume ¬ ¬*t* (*acc A p*)
thus *?thesis*
using *loop-comp-helper.simps*(*1*)
unfolding *defer-lift-invariance-def*
by *metis*
qed
qed
qed

lemma *loop-comp-helper-def-lift-inv*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-acc: *defer-lift-invariance acc* **and**
profile: *finite-profile A p*
shows
$\forall$ *q a*. (*lifted A p q a* $\wedge$ *a* $\in$ (*defer* (*loop-comp-helper acc m t*) *A p*)) $\longrightarrow$
(*loop-comp-helper acc m t*) *A p* = (*loop-comp-helper acc m t*) *A q*
using *loop-comp-helper-def-lift-inv-helper*
monotone-m monotone-acc profile
by *blast*

lemma *loop-comp-helper-def-lift-inv-2*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-acc: *defer-lift-invariance acc* **and**
finite-A-p: *finite-profile A p* **and**
lifted-A-pq: *lifted A p q a* **and**

a-in-defer-acc: $a \in$ *defer* (*loop-comp-helper acc m t*) *A p*
shows (*loop-comp-helper acc m t*) *A p* = (*loop-comp-helper acc m t*) *A q*
using *finite-A-p lifted-A-pq a-in-defer-acc*
loop-comp-helper-def-lift-inv
monotone-acc monotone-m
by *blast*

lemma *lifted-imp-fin-prof*:
fixes
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assumes *lifted A p q a*
shows *finite-profile A p*
using *assms*
unfolding *Profile.lifted-def*
by *simp*

lemma *loop-comp-helper-presv-def-lift-inv*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module*
assumes
monotone-m: *defer-lift-invariance m* **and**
monotone-acc: *defer-lift-invariance acc*
shows *defer-lift-invariance* (*loop-comp-helper acc m t*)
proof (*unfold defer-lift-invariance-def*, *safe*)
show *electoral-module* (*loop-comp-helper acc m t*)
using *electoral-modI loop-comp-helper-imp-partit monotone-acc monotone-m*
unfolding *defer-lift-invariance-def*
by (*metis* (*no-types*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assume
defer-a: $a \in$ *defer* (*loop-comp-helper acc m t*) *A p* **and**
lift-a: *Profile.lifted A p q a*
show *loop-comp-helper acc m t A p* = *loop-comp-helper acc m t A q*
using *defer-a lift-a lifted-imp-fin-prof loop-comp-helper-def-lift-inv*
monotone-acc monotone-m
by (*metis* (*full-types*))
qed

lemma *loop-comp-presv-non-electing-helper*:

fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
n :: *nat*
assumes
non-electing-m: *non-electing m* **and**
non-electing-acc: *non-electing acc* **and**
f-prof: *finite-profile A p* **and**
acc-defer-card: $n = card\ (defer\ acc\ A\ p)$
shows $elect\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p = \{\}$
using *acc-defer-card non-electing-acc*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)
case (*less n*)
thus *?case*
proof (*safe*)
fix *x* :: *'a*
assume
y-acc-no-elect:
$(\bigwedge y\ acc'.\ y < card\ (defer\ acc\ A\ p) \Longrightarrow$
$y = card\ (defer\ acc'\ A\ p) \Longrightarrow non\text{-}electing\ acc' \Longrightarrow$
$elect\ (loop\text{-}comp\text{-}helper\ acc'\ m\ t)\ A\ p = \{\})$ **and**
acc-non-elect: *non-electing acc* **and**
x-in-acc-elect: $x \in elect\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p$
have
$\forall\ (f{::}'a\ set \Rightarrow 'a\ Profile \Rightarrow 'a\ Result)\ g.$
$(non\text{-}electing\ f \wedge non\text{-}electing\ g) \longrightarrow$
$non\text{-}electing\ (f \triangleright g)$
by *simp*
hence *seq-acc-m-non-elect*: $non\text{-}electing\ (acc \triangleright m)$
using *acc-non-elect non-electing-m*
by *blast*
have $\forall\ A\ B.\ (finite\ (A{::}'a\ set) \wedge B \subset A) \longrightarrow card\ B < card\ A$
using *psubset-card-mono*
by *metis*
hence *card-ineq*:
$\forall\ A\ B.\ (finite\ (A{::}'a\ set) \wedge B \subset A) \longrightarrow card\ B < card\ A$
by *presburger*
have *no-elect-acc*: $elect\ acc\ A\ p = \{\}$
using *acc-non-elect f-prof non-electing-def*
by *auto*
have *card-n-no-elect*:
$\forall\ n\ f.$
$(n < card\ (defer\ acc\ A\ p) \wedge n = card\ (defer\ f\ A\ p) \wedge non\text{-}electing\ f) \longrightarrow$
$elect\ (loop\text{-}comp\text{-}helper\ f\ m\ t)\ A\ p = \{\}$
using *y-acc-no-elect*
by *blast*

have
$\bigwedge f.$
(*finite* (*defer acc A p*) $\wedge$ *defer f A p* $\subset$ *defer acc A p* $\wedge$ *non-electing f*) $\longrightarrow$
elect (*loop-comp-helper f m t*) *A p* = {}
using *card-n-no-elect psubset-card-mono*
by *metis*
hence *loop-helper-term*:
($\neg$ *t* (*acc A p*) $\wedge$ *defer* (*acc* $\triangleright$ *m*) *A p* $\subset$ *defer acc A p* $\wedge$
finite (*defer acc A p*)) $\wedge$
$\neg$ *t* (*acc A p*) $\longrightarrow$
elect (*loop-comp-helper acc m t*) *A p* = {}
using *loop-comp-code-helper seq-acc-m-non-elect*
by (*metis* (*no-types*))
obtain *set-func* :: *'a set* $\Rightarrow$ *'a* **where**
$\forall$ *A*. (*A* = {} $\longrightarrow$ ($\forall$ *a*. *a* $\notin$ *A*)) $\wedge$ (*A* $\neq$ {} $\longrightarrow$ *set-func A* $\in$ *A*)
using *all-not-in-conv*
by (*metis* (*no-types*))
thus *x* $\in$ {}
using *loop-comp-code-helper no-elect-acc x-in-acc-elect loop-helper-term*
by (*metis* (*no-types*))
qed
qed

lemma *loop-comp-helper-iter-elim-def-n-helper*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
n :: *nat* **and**
x :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall$ *r*. ((*t r*) = (*card* (*defer-r r*) = *x*)) **and**
x-greater-zero: *x* > *0* **and**
f-prof: *finite-profile A p* **and**
n-acc-defer-card: *n* = *card* (*defer acc A p*) **and**
n-ge-x: *n* $\geq$ *x* **and**
def-card-gt-one: *card* (*defer acc A p*) > *1* **and**
acc-nonelect: *non-electing acc*
shows *card* (*defer* (*loop-comp-helper acc m t*) *A p*) = *x*
using *n-ge-x def-card-gt-one acc-nonelect n-acc-defer-card*
proof (*induct n arbitrary*: *acc rule*: *less-induct*)

case (*less n*)
have *mod-acc*: *electoral-module acc*
using *less.prems*(*3*) *non-electing-def*

by *metis*
hence *step-reduces-defer-set*: *defer* (*acc* ▷ *m*) *A p* ⊂ *defer acc A p*
using *seq-comp-elim-one-red-def-set single-elimination*
f-prof less.prems(*2*)
by *metis*
thus *?case*
proof (*cases t* (*acc A p*))
case *True*
assume *term-satisfied*: *t* (*acc A p*)
thus *card* (*defer-r* (*loop-comp-helper acc m t A p*)) = *x*
using *loop-comp-helper.simps*(*1*) *term-satisfied terminate-if-n-left*
by *metis*
next
case *False*
hence *card-not-eq-x*: *card* (*defer acc A p*) ≠ *x*
using *terminate-if-n-left*
by *metis*
have ¬(*infinite* (*defer acc A p*))
using *def-presv-fin-prof f-prof mod-acc*
by (*metis* (*full-types*))
hence *rec-step*: *loop-comp-helper acc m t A p* = *loop-comp-helper* (*acc* ▷ *m*) *m*
t A p
using *False loop-comp-helper.simps*(*2*) *step-reduces-defer-set*
by *metis*
have *card-too-big*: *card* (*defer acc A p*) > *x*
using *card-not-eq-x dual-order.order-iff-strict less.prems*(*1*, *4*)
by *simp*
hence *enough-leftover*: *card* (*defer acc A p*) > *1*
using *x-greater-zero*
by *simp*
obtain *k* **where**
new-card-k: *k* = *card* (*defer* (*acc* ▷ *m*) *A p*)
by *metis*
have *defer acc A p* ⊆ *A*
using *defer-in-alts f-prof mod-acc*
by *metis*
hence *step-profile*:
finite-profile (*defer acc A p*) (*limit-profile* (*defer acc A p*) *p*)
using *f-prof limit-profile-sound*
by *metis*
hence
card (*defer m* (*defer acc A p*) (*limit-profile* (*defer acc A p*) *p*)) =
card (*defer acc A p*) − *1*
using *enough-leftover non-electing-m single-elim-decr-def-card-2*
single-elimination
by *metis*
hence *k-card*: *k* = *card* (*defer acc A p*) − *1*
using *mod-acc f-prof new-card-k non-electing-def*
non-electing-m seq-comp-defers-def-set

by *metis*
hence *new-card-still-big-enough*: $x \leq k$
using *card-too-big*
by *linarith*
show *?thesis*
proof (*cases* $x < k$)
case *True*
hence $1 < card$ (*defer* (*acc* ▷ *m*) *A p*)
using *new-card-k x-greater-zero*
by *linarith*
moreover have $k < n$
using *step-reduces-defer-set step-profile psubset-card-mono new-card-k less.prems*(*4*)
by *blast*
moreover have *electoral-module* (*acc* ▷ *m*)
using *mod-acc eliminates-def seq-comp-sound single-elimination*
by *metis*
moreover have *non-electing* (*acc* ▷ *m*)
using *less.prems*(*3*) *non-electing-m*
by *simp*
ultimately have
card (*defer* (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *A p*) = *x*
using *new-card-k new-card-still-big-enough less.hyps*
by *metis*
thus *?thesis*
using *rec-step*
by *presburger*
next
case *False*
thus *?thesis*
using *dual-order.strict-iff-order new-card-k new-card-still-big-enough rec-step terminate-if-n-left*
by *simp*
qed
qed
qed

lemma *loop-comp-helper-iter-elim-def-n*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
acc :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
x :: *nat*
assumes
non-electing-m: *non-electing m* **and**

single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: ∀ *r*. ((*t r*) = (*card* (*defer-r r*) = *x*)) **and**
x-greater-zero: *x* > *0* **and**
f-prof: *finite-profile A p* **and**
acc-defers-enough: *card* (*defer acc A p*) ≥ *x* **and**
non-electing-acc: *non-electing acc*
shows *card* (*defer* (*loop-comp-helper acc m t*) *A p*) = *x*
using *acc-defers-enough gr-implies-not0 le-neq-implies-less less-one linorder-neqE-nat loop-comp-helper.simps*(*1*) *loop-comp-helper-iter-elim-def-n-helper non-electing-acc non-electing-m f-prof single-elimination nat-neq-iff terminate-if-n-left x-greater-zero less-le*
by (*metis* (*no-types*, *lifting*))

lemma *iter-elim-def-n-helper*:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
x :: *nat*
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: ∀ *r*. ((*t r*) = (*card* (*defer-r r*) = *x*)) **and**
x-greater-zero: *x* > *0* **and**
f-prof: *finite-profile A p* **and**
enough-alternatives: *card A* ≥ *x*
shows *card* (*defer* (*m* $\circlearrowleft_t$) *A p*) = *x*
proof (*cases*)
assume *card A* = *x*
thus *?thesis*
by (*simp add*: *terminate-if-n-left*)
next
assume *card-not-x*: ¬ *card A* = *x*
thus *?thesis*
proof (*cases*)
assume *card A* < *x*
thus *?thesis*
using *enough-alternatives not-le*
by *blast*
next
assume ¬*card A* < *x*
hence *card-big-enough-A*: *card A* > *x*
using *card-not-x*
by *linarith*
hence *card-m*: *card* (*defer m A p*) = *card A* − *1*
using *non-electing-m f-prof single-elimination single-elim-decr-def-card-2 x-greater-zero*

by *fastforce*
hence *card-big-enough-m*: *card* (*defer m A p*) $\geq x$
using *card-big-enough-A*
by *linarith*
hence ($m \circlearrowleft_t$) *A p* = (*loop-comp-helper m m t*) *A p*
by (*simp add*: *card-not-x terminate-if-n-left*)
thus *?thesis*
using *card-big-enough-m non-electing-m f-prof single-elimination terminate-if-n-left x-greater-zero loop-comp-helper-iter-elim-def-n*
by *metis*
qed
qed

4.5.4 Composition Rules

The loop composition preserves defer-lift-invariance.

theorem *loop-comp-presv-def-lift-inv*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
t :: *'a Termination-Condition*
assumes *defer-lift-invariance m*
shows *defer-lift-invariance* ($m \circlearrowleft_t$)
proof (*unfold defer-lift-invariance-def*, *safe*)
from *assms*
have *electoral-module m*
unfolding *defer-lift-invariance-def*
by *simp*
thus *electoral-module* ($m \circlearrowleft_t$)
by (*simp add*: *loop-comp-sound*)
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
q :: *'a Profile* **and**
a :: *'a*
assume
a-in-loop-defer: $a \in$ *defer* ($m \circlearrowleft_t$) *A p* **and**
lifted-a: *Profile.lifted A p q a*
have *defer-lift-loop*:
$\forall$ *p q a*. ($a \in$ (*defer* ($m \circlearrowleft_t$) *A p*) $\wedge$ *lifted A p q a*) $\longrightarrow$
($m \circlearrowleft_t$) *A p* = ($m \circlearrowleft_t$) *A q*
using *assms lifted-imp-fin-prof loop-comp-helper-def-lift-inv-2 loop-composition.simps defer-module.simps*
by (*metis* (*full-types*))
show ($m \circlearrowleft_t$) *A p* = ($m \circlearrowleft_t$) *A q*
using *a-in-loop-defer lifted-a defer-lift-loop*
by *metis*
qed

The loop composition preserves the property non-electing.

theorem *loop-comp-presv-non-electing*[*simp*]:
  fixes
    m :: $'a$ *Electoral-Module* **and**
    t :: $'a$ *Termination-Condition*
  assumes *non-electing-m*: *non-electing m*
  shows *non-electing* $(m \circlearrowleft_t)$
proof (*unfold non-electing-def*, *safe*, *simp-all*)
  show *electoral-module* $(m \circlearrowleft_t)$
    using *loop-comp-sound non-electing-m*
    unfolding *non-electing-def*
    by *metis*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    x :: $'a$
  assume
    finite A **and**
    profile A p **and**
    $x \in$ *elect* $(m \circlearrowleft_t)$ A p
  thus *False*
    using *def-mod-non-electing loop-comp-presv-non-electing-helper*
        non-electing-m empty-iff loop-comp-code
    unfolding *non-electing-def*
    by (*metis* (*no-types*))
qed

theorem *iter-elim-def-n*[*simp*]:
  fixes
    m :: $'a$ *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 (*unfold defers-def*, *safe*)
  show *electoral-module* $(m \circlearrowleft_t)$
    using *loop-comp-sound non-electing-m*
    unfolding *non-electing-def*
    by *metis*
next
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile*
  assume

$n \leq$ *card A* **and**
finite A **and**
profile A p
thus *card* (*defer* (m $\circlearrowleft_t$) A p) $= n$
using *iter-elim-def-n-helper non-electing-m single-elimination terminate-if-n-left x-greater-zero*
by *metis*
qed

end

4.6 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.

4.6.1 Definition

fun *maximum-parallel-composition* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* **where**
maximum-parallel-composition m n $=$
(*let* a $=$ *max-aggregator in* ($m \parallel_a n$))

abbreviation *max-parallel* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* (**infix** $\parallel_\uparrow$ *50*) **where**
$m \parallel_\uparrow n ==$ *maximum-parallel-composition m n*

4.6.2 Soundness

theorem *max-par-comp-sound*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module*

assumes
  mod-m: *electoral-module m* **and**
  mod-n: *electoral-module n*
shows *electoral-module* $(m \parallel_\uparrow n)$
using *mod-m mod-n*
by *simp*

4.6.3 Lemmas

lemma *max-agg-eq-result*:
  fixes
    m :: $'a$ *Electoral-Module* **and**
    n :: $'a$ *Electoral-Module* **and**
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile* **and**
    a :: $'a$
  assumes
    module-m: *electoral-module m* **and**
    module-n: *electoral-module n* **and**
    f-prof: *finite-profile A p* **and**
    in-A: $a \in A$
  shows
    mod-contains-result $(m \parallel_\uparrow n)$ m A p a $\vee$
      mod-contains-result $(m \parallel_\uparrow n)$ n A p a
proof (*cases*)
  assume *a-elect*: $a \in$ *elect* $(m \parallel_\uparrow n)$ A p
    have *mod-contains-inst*:
      $\forall$ *p-mod q-mod a-set prof b*.
        mod-contains-result p-mod q-mod a-set prof $(b{::}'a)$ =
          (*electoral-module p-mod* $\wedge$ *electoral-module q-mod* $\wedge$
            finite a-set $\wedge$ *profile a-set prof* $\wedge$ $b \in$ *a-set* $\wedge$
            ($b \notin$ *elect p-mod a-set prof* $\vee$ $b \in$ *elect q-mod a-set prof*) $\wedge$
            ($b \notin$ *reject p-mod a-set prof* $\vee$ $b \in$ *reject q-mod a-set prof*) $\wedge$
            ($b \notin$ *defer p-mod a-set prof* $\vee$ $b \in$ *defer q-mod a-set prof*))
      unfolding *mod-contains-result-def*
      by *simp*
    have *module-mn*: *electoral-module* $(m \parallel_\uparrow n)$
      using *module-m module-n*
      by *simp*
  have *not-defer-mn*: $a \notin$ *defer* $(m \parallel_\uparrow n)$ A p
    using *module-mn IntI a-elect empty-iff f-prof result-disj*
    by (*metis* (*no-types*))
  have *not-reject-mn*: $a \notin$ *reject* $(m \parallel_\uparrow n)$ A p
    using *module-mn IntI a-elect empty-iff f-prof result-disj*
    by (*metis* (*no-types*))
  from *a-elect*
  have *let* (*e1*, *r1*, *d1*) = m A p;
        (*e2*, *r2*, *d2*) = n A p *in*
      $a \in$ *e1* $\cup$ *e2*

by *auto*
hence *union-mn*: $a \in (\mathit{elect}\ m\ A\ p) \cup (\mathit{elect}\ n\ A\ p)$
by *auto*
thus *?thesis*
using *f-prof in-A module-m module-mn module-n*
not-defer-mn not-reject-mn union-mn
mod-contains-inst
by *blast*
next
assume *not-a-elect*: $a \notin \mathit{elect}\ (m \parallel_{\uparrow} n)\ A\ p$
thus *?thesis*
proof (*cases*)
assume *a-in-def*: $a \in \mathit{defer}\ (m \parallel_{\uparrow} n)\ A\ p$
thus *?thesis*
proof (*safe*)
assume *not-mod-cont-mn*:
$\neg$ *mod-contains-result* $(m \parallel_{\uparrow} n)\ n\ A\ p\ a$
have *par-emod*:
$\forall\ f\ g.$
(*electoral-module* $(f{::}'a\ \mathit{set} \Rightarrow {}'a\ \mathit{Profile} \Rightarrow {}'a\ \mathit{Result}) \wedge$
electoral-module $g) \longrightarrow$
electoral-module $(f \parallel_{\uparrow} g)$
using *max-par-comp-sound*
by *blast*
hence *electoral-module* $(m \parallel_{\uparrow} n)$
using *module-m module-n*
by *blast*
hence *max-par-emod*:
electoral-module $(m \parallel_m ax\text{-}aggregator\ n)$
by *simp*
have *set-intersect*:
$\forall\ (b{::}'a)\ A\ B.\ (b \in A \cap B) = (b \in A \wedge b \in B)$
by *blast*
obtain
s-func :: $('a\ \mathit{set} \Rightarrow {}'a\ \mathit{Profile} \Rightarrow {}'a\ \mathit{Result}) \Rightarrow {}'a\ \mathit{set}$ **and**
p-func :: $('a\ \mathit{set} \Rightarrow {}'a\ \mathit{Profile} \Rightarrow {}'a\ \mathit{Result}) \Rightarrow {}'a\ \mathit{Profile}$ **where**
well-f:
$\forall\ f.$
($\neg$ *electoral-module* $f \vee$
$(\forall\ A\ \mathit{prof}.\ (\mathit{finite}\ A \wedge \mathit{profile}\ A\ \mathit{prof}) \longrightarrow \mathit{well\text{-}formed}\ A\ (f\ A\ \mathit{prof}))) \wedge$
(*electoral-module* $f \vee \mathit{finite}\ (\mathit{s\text{-}func}\ f) \wedge \mathit{profile}\ (\mathit{s\text{-}func}\ f)\ (\mathit{p\text{-}func}\ f) \wedge$
$\neg\ \mathit{well\text{-}formed}\ (\mathit{s\text{-}func}\ f)\ (f\ (\mathit{s\text{-}func}\ f)\ (\mathit{p\text{-}func}\ f)))$
unfolding *electoral-module-def*
by *moura*
hence *wf-n*: *well-formed* $A\ (n\ A\ p)$
using *f-prof module-n*
by *blast*
have *wf-m*: *well-formed* $A\ (m\ A\ p)$
using *well-f f-prof module-m*

by *blast*
have *e-mod-par*: *electoral-module* $(m \parallel_\uparrow n)$
using *par-emod module-m module-n*
by *blast*
hence *electoral-module* $(m \parallel_m ax\text{-}aggregator\ n)$
by *simp*
hence *result-disj-max*:
$elect\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p \cap reject\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p = \{\}$
$\wedge$
$elect\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p \cap defer\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p = \{\}$
$\wedge$
$reject\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p \cap defer\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p = \{\}$
using *f-prof result-disj*
by *metis*
have *a-not-elect*:
$a \notin elect\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p$
using *result-disj-max a-in-def*
by *force*
have *result-m*:
$(elect\ m\ A\ p,\ reject\ m\ A\ p,\ defer\ m\ A\ p) = m\ A\ p$
by *auto*
have *result-n*:
$(elect\ n\ A\ p,\ reject\ n\ A\ p,\ defer\ n\ A\ p) = n\ A\ p$
by *auto*
have *max-pq*:
$\forall\ (B::'a\ set)\ p\ q.$
$elect\text{-}r\ (max\text{-}aggregator\ B\ p\ q) = elect\text{-}r\ p \cup elect\text{-}r\ q$
by *force*
have
$a \notin elect\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p$
using *a-not-elect*
by *blast*
with *max-pq*
have $a \notin elect\ m\ A\ p \cup elect\ n\ A\ p$
by (*simp add*: *max-pq*)
hence *b-not-elect-mn*:
$a \notin elect\ m\ A\ p \wedge a \notin elect\ n\ A\ p$
by *blast*
have *b-not-mpar-rej*:
$a \notin reject\ (m \parallel_m ax\text{-}aggregator\ n)\ A\ p$
using *result-disj-max a-in-def*
by *fastforce*
hence *b-not-par-rej*:
$a \notin reject\ (m \parallel_\uparrow n)\ A\ p$
by *auto*
have *mod-cont-res-fg*:
$\forall\ f\ g\ B\ prof\ (b::'a).$
$mod\text{-}contains\text{-}result\ f\ g\ B\ prof\ b =$
$(electoral\text{-}module\ f \wedge electoral\text{-}module\ g \wedge$

finite B ∧ *profile B prof* ∧ $b \in B$ ∧
($b \notin$ *elect f B prof* ∨ $b \in$ *elect g B prof*) ∧
($b \notin$ *reject f B prof* ∨ $b \in$ *reject g B prof*) ∧
($b \notin$ *defer f B prof* ∨ $b \in$ *defer g B prof*))
by (*simp add*: *mod-contains-result-def*)
have *max-agg-res*:
max-aggregator A (*elect m A p*, *reject m A p*, *defer m A p*)
(*elect n A p*, *reject n A p*, *defer n A p*) = (*m* $\|_m$*ax-aggregator n*) *A p*
by *simp*
have *well-f-max*:
∀ *r2 r1 e2 e1 d2 d1 B*.
well-formed B (*e1*, *r1*, *d1*) ∧ *well-formed B* (*e2*, *r2*, *d2*) ⟶
reject-r (*max-aggregator B* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) = *r1* ∩ *r2*
using *max-agg-rej-set*
by *metis*
have *e-mod-disj*:
∀ *f* (*B*::′*a set*) *prof*.
(*electoral-module f* ∧ *finite* (*B*::′*a set*) ∧ *profile B prof*) ⟶
elect f B prof ∪ *reject f B prof* ∪ *defer f B prof* = *B*
using *result-presv-alts*
by *blast*
hence *e-mod-disj-n*:
elect n A p ∪ *reject n A p* ∪ *defer n A p* = *A*
using *f-prof module-n*
by *metis*
have
∀ *f g B prof* (*b*::′*a*).
mod-contains-result f g B prof b =
(*electoral-module f* ∧ *electoral-module g* ∧
finite B ∧ *profile B prof* ∧ $b \in B$ ∧
($b \notin$ *elect f B prof* ∨ $b \in$ *elect g B prof*) ∧
($b \notin$ *reject f B prof* ∨ $b \in$ *reject g B prof*) ∧
($b \notin$ *defer f B prof* ∨ $b \in$ *defer g B prof*))
by (*simp add*: *mod-contains-result-def*)
with *e-mod-disj-n*
have $a \in$ *reject n A p*
using *e-mod-par f-prof in-A module-n not-mod-cont-mn*
a-not-elect b-not-elect-mn b-not-mpar-rej
by *auto*
hence $a \notin$ *reject m A p*
using *well-f-max max-agg-res result-m result-n*
set-intersect wf-m wf-n b-not-mpar-rej
by (*metis* (*no-types*))
with *max-agg-res*
have $a \notin$ *defer* (*m* $\|_\uparrow$ *n*) *A p* ∨ $a \in$ *defer m A p*
using *e-mod-disj f-prof in-A module-m b-not-elect-mn*
by *blast*
with *b-not-mpar-rej*
show *mod-contains-result* (*m* $\|_\uparrow$ *n*) *m A p a*

using *mod-cont-res-fg b-not-par-rej e-mod-par f-prof in-A module-m a-not-elect*
by *auto*
qed
next
assume *not-a-defer*: $a \notin$ *defer* $(m \parallel_\uparrow n)$ *A p*
have *el-rej-defer*:
(*elect m A p*, *reject m A p*, *defer m A p*) = *m A p*
by *auto*
from *not-a-elect not-a-defer*
have *a-reject*: $a \in$ *reject* $(m \parallel_\uparrow n)$ *A p*
using *electoral-mod-defer-elem in-A module-m module-n f-prof max-par-comp-sound*
by *metis*
hence
case snd (*m A p*) *of* (*Aa*, *Ab*) ⇒
case n A p of (*Ac*, *Ad*, *Ae*) ⇒
$a \in$ *reject-r*
(*max-aggregator A*
(*elect m A p*, *Aa*, *Ab*) (*Ac*, *Ad*, *Ae*))
using *el-rej-defer*
by *force*
hence
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
$a \in$ *reject-r* (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*))
by (*simp add*: *case-prod-unfold*)
hence
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
$a \in A - (e1 \cup e2 \cup d1 \cup d2)$
by *simp*
hence $a \notin$ *elect m A p* ∪ (*defer n A p* ∪ *defer m A p*)
by *force*
thus *?thesis*
using *mod-contains-result-comm mod-contains-result-def Un-iff a-reject f-prof in-A module-m module-n max-par-comp-sound*
by (*metis* (*no-types*))
qed
qed

lemma *max-agg-rej-iff-both-reject*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a*
assumes

f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n*
shows
$(a \in \mathit{reject}\ (m \parallel_{\uparrow} n)\ A\ p) =$
$(a \in \mathit{reject}\ m\ A\ p \land a \in \mathit{reject}\ n\ A\ p)$
proof
assume *rej-a*: $a \in \mathit{reject}\ (m \parallel_{\uparrow} n)\ A\ p$
hence
case n A p of $(Aa, Ab, Ac) \Rightarrow$
$a \in$ *reject-r* (*max-aggregator A*
(*elect m A p*, *reject m A p*, *defer m A p*) $(Aa, Ab, Ac))$
by *auto*
hence
case snd (*m A p*) *of* $(Aa, Ab) \Rightarrow$
case n A p of $(Ac, Ad, Ae) \Rightarrow$
$a \in$ *reject-r* (*max-aggregator A*
(*elect m A p*, *Aa*, *Ab*) (*Ac*, *Ad*, *Ae*))
by *force*
with *rej-a*
have *let* (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
$a \in$ *reject-r* (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*))
by (*simp add*: *prod.case-eq-if*)
hence
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
$a \in A -$ (*e1* $\cup$ *e2* $\cup$ *d1* $\cup$ *d2*)
by *simp*
hence
$a \in A -$ (*elect m A p* $\cup$ *elect n A p* $\cup$ *defer m A p* $\cup$ *defer n A p*)
by *auto*
thus $a \in$ *reject m A p* $\land$ $a \in$ *reject n A p*
using *Diff-iff Un-iff electoral-mod-defer-elem*
f-prof module-m module-n
by *metis*
next
assume *a*: $a \in$ *reject m A p* $\land$ $a \in$ *reject n A p*
hence
$a \notin$ *elect m A p* $\land$ $a \notin$ *defer m A p* $\land$
$a \notin$ *elect n A p* $\land$ $a \notin$ *defer n A p*
using *IntI empty-iff module-m module-n f-prof result-disj*
by *metis*
thus $a \in \mathit{reject}\ (m \parallel_{\uparrow} n)\ A\ p$
using *DiffD1 a f-prof max-agg-eq-result module-m module-n*
mod-contains-result-comm mod-contains-result-def
reject-not-elec-or-def
by (*metis* (*no-types*))
qed

lemma *max-agg-rej-1*:
  fixes
    m :: *'a Electoral-Module* **and**
    n :: *'a Electoral-Module* **and**
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    a :: *'a*
  assumes
    f-prof: *finite-profile A p* **and**
    module-m: *electoral-module m* **and**
    module-n: *electoral-module n* **and**
    rejected: *a* ∈ *reject n A p*
  shows *mod-contains-result m* (*m* $\|_\uparrow$ *n*) *A p a*
proof (*unfold mod-contains-result-def*, *safe*)
  show *electoral-module m*
    using *module-m*
    by *simp*
next
  show *electoral-module* (*m* $\|_\uparrow$ *n*)
    using *module-m module-n*
    by *simp*
next
  show *finite A*
    using *f-prof*
    by *simp*
next
  show *profile A p*
    using *f-prof*
    by *simp*
next
  show *a* ∈ *A*
    using *f-prof module-n reject-in-alts rejected*
    by *auto*
next
  assume *a-in-elect*: *a* ∈ *elect m A p*
  hence *a-not-reject*: *a* ∉ *reject m A p*
    using *disjoint-iff-not-equal f-prof module-m result-disj*
    by *metis*
  have *rej-in-A*: *reject n A p* ⊆ *A*
    using *f-prof module-n*
    by (*simp add*: *reject-in-alts*)
  have *a-in-A*: *a* ∈ *A*
    using *rej-in-A in-mono rejected*
    by *metis*
  with *a-in-elect a-not-reject*
  show *a* ∈ *elect* (*m* $\|_\uparrow$ *n*) *A p*
    using *f-prof max-agg-eq-result module-m module-n rejected*
        max-agg-rej-iff-both-reject mod-contains-result-comm

mod-contains-result-def
by *metis*
next
assume $a \in$ *reject* $m\ A\ p$
hence $a \in$ *reject* $m\ A\ p \wedge a \in$ *reject* $n\ A\ p$
using *rejected*
by *simp*
thus $a \in$ *reject* $(m \parallel_{\uparrow} n)\ A\ p$
using *f-prof max-agg-rej-iff-both-reject module-m module-n*
by (*metis* (*no-types*))
next
assume *a-in-defer*: $a \in$ *defer* $m\ A\ p$
hence *defer-a*:
$\exists\ b.\ b \in$ *defer* $m\ A\ p \wedge b = a$
by *simp*
then obtain *a-inst* :: $'a$ **where**
inst-a: $a =$ *a-inst* $\wedge$ *a-inst* $\in$ *defer* $m\ A\ p$
by *metis*
hence *a-not-rej*: $a \notin$ *reject* $m\ A\ p$
using *disjoint-iff-not-equal f-prof inst-a module-m result-disj*
by (*metis* (*no-types*))
have
$\forall\ f\ A$ *prof*.
(*electoral-module* $f \wedge$ *finite* $(A{::}'a\ set) \wedge$ *profile* A *prof*) $\longrightarrow$
elect $f\ A$ *prof* $\cup$ *reject* $f\ A$ *prof* $\cup$ *defer* $f\ A$ *prof* $= A$
using *result-presv-alts*
by *metis*
with *a-in-defer*
have $a \in A$
using *f-prof module-m*
by *blast*
with *inst-a a-not-rej*
show $a \in$ *defer* $(m \parallel_{\uparrow} n)\ A\ p$
using *f-prof max-agg-eq-result max-agg-rej-iff-both-reject*
mod-contains-result-comm mod-contains-result-def
module-m module-n rejected
by *metis*
qed

lemma *max-agg-rej-2*:
fixes
m :: $'a$ *Electoral-Module* **and**
n :: $'a$ *Electoral-Module* **and**
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
a :: $'a$
assumes
f-prof: *finite-profile* $A\ p$ **and**
module-m: *electoral-module* m **and**

module-n: *electoral-module n* **and**
rejected: $a \in$ *reject n A p*
shows *mod-contains-result* $(m \parallel_\uparrow n)$ *m A p a*
using *mod-contains-result-comm max-agg-rej-1*
module-m module-n f-prof rejected
by *metis*

lemma *max-agg-rej-3*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a*
assumes
f-prof: *finite-profile A p* **and**
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
rejected: $a \in$ *reject m A p*
shows *mod-contains-result n* $(m \parallel_\uparrow n)$ *A p a*
proof (*unfold mod-contains-result-def*, *safe*)
show *electoral-module n*
using *module-n*
by *simp*
next
show *electoral-module* $(m \parallel_\uparrow n)$
using *module-m module-n*
by *simp*
next
show *finite A*
using *f-prof*
by *simp*
next
show *profile A p*
using *f-prof*
by *simp*
next
show $a \in A$
using *f-prof in-mono module-m reject-in-alts rejected*
by (*metis* (*no-types*))
next
assume $a \in$ *elect n A p*
thus $a \in$ *elect* $(m \parallel_\uparrow n)$ *A p*
using *Un-iff combine-ele-rej-def fst-conv*
maximum-parallel-composition.simps
max-aggregator.simps
unfolding *parallel-composition.simps*
by (*metis* (*mono-tags*, *lifting*))
next

assume $a \in$ *reject* n A p
thus $a \in$ *reject* $(m \parallel_\uparrow n)$ A p
using *f-prof max-agg-rej-iff-both-reject module-m module-n rejected*
by *metis*
next
assume *a-in-def*: $a \in$ *defer* n A p
have $a \in A$
using *f-prof max-agg-rej-1 mod-contains-result-def module-m rejected*
by *metis*
thus $a \in$ *defer* $(m \parallel_\uparrow n)$ A p
using *a-in-def disjoint-iff-not-equal f-prof*
max-agg-eq-result max-agg-rej-iff-both-reject
mod-contains-result-comm mod-contains-result-def
module-m module-n rejected result-disj
by *metis*
qed

lemma *max-agg-rej-4*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
a :: *'a*
assumes
f-prof: *finite-profile* A p **and**
module-m: *electoral-module* m **and**
module-n: *electoral-module* n **and**
rejected: $a \in$ *reject* m A p
shows *mod-contains-result* $(m \parallel_\uparrow n)$ n A p a
using *mod-contains-result-comm max-agg-rej-3*
module-m module-n f-prof rejected
by *metis*

lemma *max-agg-rej-intersect*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile*
assumes
module-m: *electoral-module* m **and**
module-n: *electoral-module* n **and**
f-prof: *finite-profile* A p
shows *reject* $(m \parallel_\uparrow n)$ A p = (*reject* m A p) $\cap$ (*reject* n A p)
proof –
have
A = (*elect* m A p) $\cup$ (*reject* m A p) $\cup$ (*defer* m A p) $\wedge$
A = (*elect* n A p) $\cup$ (*reject* n A p) $\cup$ (*defer* n A p)

using *module-m module-n f-prof result-presv-alts*
by *metis*
hence
$A - ((elect\ m\ A\ p) \cup (defer\ m\ A\ p)) = (reject\ m\ A\ p) \wedge$
$A - ((elect\ n\ A\ p) \cup (defer\ n\ A\ p)) = (reject\ n\ A\ p)$
using *module-m module-n f-prof reject-not-elec-or-def*
by *auto*
hence
$A - ((elect\ m\ A\ p) \cup (elect\ n\ A\ p) \cup (defer\ m\ A\ p) \cup (defer\ n\ A\ p)) =$
$(reject\ m\ A\ p) \cap (reject\ n\ A\ p)$
by *blast*
hence
let $(e1,\ r1,\ d1) = m\ A\ p;$
$(e2,\ r2,\ d2) = n\ A\ p$ *in*
$A - (e1 \cup e2 \cup d1 \cup d2) = r1 \cap r2$
by *fastforce*
thus *?thesis*
by *auto*
qed

lemma *dcompat-dec-by-one-mod*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
a :: *'a*
assumes
compatible: *disjoint-compatibility* $m\ n$ **and**
in-A: $a \in A$
shows
$(\forall\ p.$ *finite-profile* $A\ p \longrightarrow$
mod-contains-result $m\ (m \parallel_{\uparrow} n)\ A\ p\ a) \vee$
$(\forall\ p.$ *finite-profile* $A\ p \longrightarrow$
mod-contains-result $n\ (m \parallel_{\uparrow} n)\ A\ p\ a)$
using *DiffI compatible in-A max-agg-rej-1 max-agg-rej-3*
unfolding *disjoint-compatibility-def*
by *metis*

4.6.4 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 Electoral-Module* **and**
n :: *'a Electoral-Module*
assumes
non-electing-m: *non-electing* m **and**
non-electing-n: *non-electing* n

shows *non-electing* $(m \parallel_\uparrow n)$
using *non-electing-m non-electing-n*
by *simp*

Using the max aggregator, composing two compatible electoral modules in parallel preserves defer-lift-invariance.

theorem *par-comp-def-lift-inv*[*simp*]:
fixes
m :: $'a$ *Electoral-Module* **and**
n :: $'a$ *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 (*unfold defer-lift-invariance-def*, *safe*)
have *electoral-mod-m*: *electoral-module* m
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *simp*
have *electoral-mod-n*: *electoral-module* n
using *monotone-n*
unfolding *defer-lift-invariance-def*
by *simp*
show *electoral-module* $(m \parallel_\uparrow n)$
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
q :: $'a$ *Profile* **and**
a :: $'a$
assume
defer-a: $a \in$ *defer* $(m \parallel_\uparrow n)$ A p **and**
lifted-a: *Profile.lifted* A p q a
hence *f-profs*: *finite-profile* A p $\wedge$ *finite-profile* A q
unfolding *lifted-def*
by *simp*
from *compatible*
obtain B :: $'a$ *set* **where**
alts: $B \subseteq A \wedge (\forall\ x \in B.$ *indep-of-alt* m A x $\wedge$
$(\forall\ p.$ *finite-profile* A $p \longrightarrow x \in$ *reject* m A $p)) \wedge$
$(\forall\ x \in A - B.$ *indep-of-alt* n A x $\wedge$
$(\forall\ p.$ *finite-profile* A $p \longrightarrow x \in$ *reject* n A $p))$
using *f-profs*
unfolding *disjoint-compatibility-def*
by (*metis* (*no-types*, *lifting*))
have $\forall\ x \in A.$ *prof-contains-result* $(m \parallel_\uparrow n)$ A p q x

proof (*cases*)
  assume *a0*: $a \in B$
  hence $a \in$ *reject* $m\ A\ p$
    using *alts f-profs*
    by *blast*
  with *defer-a*
  have *defer-n*: $a \in$ *defer* $n\ A\ p$
    using *compatible f-profs max-agg-rej-4*
    unfolding *disjoint-compatibility-def mod-contains-result-def*
    by *metis*
  have $\forall\ x \in B.$ *mod-contains-result* $(m \parallel_{\uparrow} n)\ n\ A\ p\ x$
    using *alts compatible max-agg-rej-4 f-profs*
    unfolding *disjoint-compatibility-def*
    by *metis*
  moreover have $\forall\ x \in A.$ *prof-contains-result* $n\ A\ p\ q\ x$
  proof (*unfold prof-contains-result-def*, *clarify*)
    fix $b :: 'a$
    assume *b-in-A*: $b \in A$
    show
      electoral-module $n\ \wedge$
       finite-profile $A\ p\ \wedge$
       finite-profile $A\ q\ \wedge$
       $b \in A\ \wedge$
       $(b \in$ *elect* $n\ A\ p \longrightarrow b \in$ *elect* $n\ A\ q)\ \wedge$
       $(b \in$ *reject* $n\ A\ p \longrightarrow b \in$ *reject* $n\ A\ q)\ \wedge$
       $(b \in$ *defer* $n\ A\ p \longrightarrow b \in$ *defer* $n\ A\ q)$
    proof (*safe*)
      show *electoral-module* n
        using *monotone-n*
        unfolding *defer-lift-invariance-def*
        by *metis*
    next
      show *finite* A
        using *f-profs*
        by *simp*
    next
      show *profile* $A\ p$
        using *f-profs*
        by *simp*
    next
      show *finite* A
        using *f-profs*
        by *simp*
    next
      show *profile* $A\ q$
        using *f-profs*
        by *simp*
    next
      show $b \in A$

```
      using b-in-A
      by simp
  next
    assume b ∈ elect n A p
    thus b ∈ elect n A q
      using defer-n lifted-a monotone-n f-profs
      unfolding defer-lift-invariance-def
      by metis
  next
    assume b ∈ reject n A p
    thus b ∈ reject n A q
      using defer-n lifted-a monotone-n f-profs
      unfolding defer-lift-invariance-def
      by metis
  next
    assume b ∈ defer n A p
    thus b ∈ defer n A q
      using defer-n lifted-a monotone-n f-profs
      unfolding defer-lift-invariance-def
      by metis
  qed
qed
moreover have
  ∀ x ∈ B. mod-contains-result n (m ∥↑ n) A q x
  using alts compatible max-agg-rej-3 f-profs
  unfolding disjoint-compatibility-def
  by metis
ultimately have 00:
  ∀ x ∈ B. prof-contains-result (m ∥↑ n) A p q x
  unfolding mod-contains-result-def prof-contains-result-def
  by simp
have
  ∀ x ∈ A − B. mod-contains-result (m ∥↑ n) m A p x
  using alts max-agg-rej-2 monotone-m monotone-n f-profs
  unfolding defer-lift-invariance-def
  by metis
moreover have ∀ x ∈ A. prof-contains-result m A p q x
proof (unfold prof-contains-result-def, clarify)
  fix b :: 'a
  assume b-in-A: b ∈ A
  show
    electoral-module m ∧
     finite-profile A p ∧
     finite-profile A q ∧
     b ∈ A ∧
     (b ∈ elect m A p ⟶ b ∈ elect m A q) ∧
     (b ∈ reject m A p ⟶ b ∈ reject m A q) ∧
     (b ∈ defer m A p ⟶ b ∈ defer m A q)
  proof (safe)
```

show *electoral-module m*
using *monotone-m*
unfolding *defer-lift-invariance-def*
by *metis*
next
show *finite A*
using *f-profs*
by *simp*
next
show *profile A p*
using *f-profs*
by *simp*
next
show *finite A*
using *f-profs*
by *simp*
next
show *profile A q*
using *f-profs*
by *simp*
next
show $b \in A$
using *b-in-A*
by *simp*
next
assume $b \in$ *elect m A p*
thus $b \in$ *elect m A q*
using *alts a0 lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
next
assume $b \in$ *reject m A p*
thus $b \in$ *reject m A q*
using *alts a0 lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
next
assume $b \in$ *defer m A p*
thus $b \in$ *defer m A q*
using *alts a0 lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
qed
qed
moreover have
$\forall\ x \in A - B.$ *mod-contains-result m* $(m \parallel_\uparrow n)$ *A q x*
using *alts max-agg-rej-1 monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*

ultimately have *01*:
  $\forall\ x \in A - B.$ *prof-contains-result* $(m \parallel_\uparrow n)\ A\ p\ q\ x$
  unfolding *mod-contains-result-def prof-contains-result-def*
  by *simp*
from *00 01*
show *?thesis*
  by *blast*
next
assume $a \notin B$
hence *a-in-set-diff*: $a \in A - B$
  using *DiffI lifted-a compatible f-profs*
  unfolding *Profile.lifted-def*
  by (*metis* (*no-types*, *lifting*))
hence $a \in$ *reject* $n\ A\ p$
  using *alts f-profs*
  by *blast*
with *defer-a*
have *defer-m*: $a \in$ *defer* $m\ A\ p$
  using *DiffD1 DiffD2 compatible dcompat-dec-by-one-mod f-profs*
    defer-not-elec-or-rej max-agg-sound par-comp-sound
    disjoint-compatibility-def not-rej-imp-elec-or-def
    mod-contains-result-def
  unfolding *maximum-parallel-composition.simps*
  by *metis*
have
  $\forall\ x \in B.$ *mod-contains-result* $(m \parallel_\uparrow n)\ n\ A\ p\ x$
  using *alts compatible max-agg-rej-4 f-profs*
  unfolding *disjoint-compatibility-def*
  by *metis*
moreover have $\forall\ x \in A.$ *prof-contains-result* $n\ A\ p\ q\ x$
proof (*unfold prof-contains-result-def*, *clarify*)
  fix $b :: 'a$
  assume *b-in-A*: $b \in A$
  show
    electoral-module $n\ \wedge$
    finite-profile $A\ p\ \wedge$
    finite-profile $A\ q\ \wedge$
    $b \in A\ \wedge$
    $(b \in$ *elect* $n\ A\ p \longrightarrow b \in$ *elect* $n\ A\ q)\ \wedge$
    $(b \in$ *reject* $n\ A\ p \longrightarrow b \in$ *reject* $n\ A\ q)\ \wedge$
    $(b \in$ *defer* $n\ A\ p \longrightarrow b \in$ *defer* $n\ A\ q)$
  proof (*safe*)
    show *electoral-module* n
      using *monotone-n*
      unfolding *defer-lift-invariance-def*
      by *metis*
  next
    show *finite* A
      using *f-profs*

by *simp*
next
show *profile A p*
using *f-profs*
by *simp*
next
show *finite A*
using *f-profs*
by *simp*
next
show *profile A q*
using *f-profs*
by *simp*
next
show $b \in A$
using *b-in-A*
by *simp*
next
assume $b \in$ *elect n A p*
thus $b \in$ *elect n A q*
using *alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
next
assume $b \in$ *reject n A p*
thus $b \in$ *reject n A q*
using *alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
next
assume $b \in$ *defer n A p*
thus $b \in$ *defer n A q*
using *alts a-in-set-diff lifted-a lifted-imp-equiv-prof-except-a*
unfolding *indep-of-alt-def*
by *metis*
qed
qed
moreover have $\forall\ x \in B.$ *mod-contains-result n* $(m \parallel_\uparrow n)$ *A q x*
using *alts compatible max-agg-rej-3 f-profs*
unfolding *disjoint-compatibility-def*
by *metis*
ultimately have *10*:
$\forall\ x \in B.$ *prof-contains-result* $(m \parallel_\uparrow n)$ *A p q x*
unfolding *mod-contains-result-def prof-contains-result-def*
by *simp*
have $\forall\ x \in A - B.$ *mod-contains-result* $(m \parallel_\uparrow n)$ *m A p x*
using *alts max-agg-rej-2 monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*

moreover have $\forall\ x \in A.$ *prof-contains-result* $m\ A\ p\ q\ x$
proof (*unfold prof-contains-result-def*, *clarify*)
  fix $b :: 'a$
  assume *b-in-A*: $b \in A$
  show
    electoral-module $m \wedge$
      finite-profile $A\ p \wedge$
      finite-profile $A\ q \wedge$
      $b \in A \wedge$
      $(b \in$ *elect* $m\ A\ p \longrightarrow b \in$ *elect* $m\ A\ q) \wedge$
      $(b \in$ *reject* $m\ A\ p \longrightarrow b \in$ *reject* $m\ A\ q) \wedge$
      $(b \in$ *defer* $m\ A\ p \longrightarrow b \in$ *defer* $m\ A\ q)$
  proof (*safe*)
    show *electoral-module m*
      using *monotone-m*
      unfolding *defer-lift-invariance-def*
      by *simp*
  next
    show *finite A*
      using *f-profs*
      by *simp*
  next
    show *profile A p*
      using *f-profs*
      by *simp*
  next
    show *finite A*
      using *f-profs*
      by *simp*
  next
    show *profile A q*
      using *f-profs*
      by *simp*
  next
    show $b \in A$
      using *b-in-A*
      by *simp*
  next
    assume $b \in$ *elect* $m\ A\ p$
    thus $b \in$ *elect* $m\ A\ q$
      using *defer-m lifted-a monotone-m*
      unfolding *defer-lift-invariance-def*
      by *metis*
  next
    assume $b \in$ *reject* $m\ A\ p$
    thus $b \in$ *reject* $m\ A\ q$
      using *defer-m lifted-a monotone-m*
      unfolding *defer-lift-invariance-def*
      by *metis*

next
assume $b \in$ *defer m A p*
thus $b \in$ *defer m A q*
using *defer-m lifted-a monotone-m*
unfolding *defer-lift-invariance-def*
by *metis*
qed
qed
moreover have
$\forall\ x \in A - B.$ *mod-contains-result m* $(m \parallel_\uparrow n)$ *A q x*
using *alts max-agg-rej-1 monotone-m monotone-n f-profs*
unfolding *defer-lift-invariance-def*
by *metis*
ultimately have *11*:
$\forall\ x \in A - B.$ *prof-contains-result* $(m \parallel_\uparrow n)$ *A p q x*
using *electoral-mod-defer-elem*
unfolding *mod-contains-result-def prof-contains-result-def*
by *simp*
from *10 11*
show *?thesis*
by *blast*
qed
thus $(m \parallel_\uparrow n)$ *A p* $=$ $(m \parallel_\uparrow n)$ *A q*
using *compatible f-profs eq-alts-in-profs-imp-eq-results*
max-par-comp-sound
unfolding *disjoint-compatibility-def*
by *metis*
qed

lemma *par-comp-rej-card*:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module* **and**
A :: *'a set* **and**
p :: *'a Profile* **and**
c :: *nat*
assumes
compatible: *disjoint-compatibility m n* **and**
f-prof: *finite-profile A p* **and**
reject-sum: *card* (*reject m A p*) $+$ *card* (*reject n A p*) $=$ *card A* $+$ *c*
shows *card* (*reject* $(m \parallel_\uparrow n)$ *A p*) $= c$
proof $-$
from *compatible*
obtain *B* **where**
alt-set: $B \subseteq A \land$
$(\forall\ a \in B.$ *indep-of-alt m A a* $\land$
$(\forall\ q.$ *finite-profile A q* $\longrightarrow a \in$ *reject m A q*$)) \land$
$(\forall\ a \in A - B.$ *indep-of-alt n A a* $\land$
$(\forall\ q.$ *finite-profile A q* $\longrightarrow a \in$ *reject n A q*$))$

using *f-prof*
unfolding *disjoint-compatibility-def*
by *metis*
from *f-prof compatible*
have *reject-representation*:
$reject\ (m \parallel_{\uparrow} n)\ A\ p = (reject\ m\ A\ p) \cap (reject\ n\ A\ p)$
using *max-agg-rej-intersect*
unfolding *disjoint-compatibility-def*
by *metis*
have $electoral\text{-}module\ m \wedge electoral\text{-}module\ n$
using *compatible*
unfolding *disjoint-compatibility-def*
by *simp*
hence *subsets*: $(reject\ m\ A\ p) \subseteq A \wedge (reject\ n\ A\ p) \subseteq A$
by (*simp add*: *f-prof reject-in-alts*)
hence $finite\ (reject\ m\ A\ p) \wedge finite\ (reject\ n\ A\ p)$
using *rev-finite-subset f-prof*
by *metis*
hence *0*:
$card\ (reject\ (m \parallel_{\uparrow} n)\ A\ p) =$
$card\ A + c -$
$card\ ((reject\ m\ A\ p) \cup (reject\ n\ A\ p))$
using *card-Un-Int reject-representation reject-sum*
by *fastforce*
have $\forall\ a \in A.\ a \in (reject\ m\ A\ p) \vee a \in (reject\ n\ A\ p)$
using *alt-set f-prof*
by *blast*
hence $A = reject\ m\ A\ p \cup reject\ n\ A\ p$
using *subsets*
by *force*
hence *1*: $card\ ((reject\ m\ A\ p) \cup (reject\ n\ A\ p)) = card\ A$
by *presburger*
from *0 1*
show $card\ (reject\ (m \parallel_{\uparrow} n)\ A\ p) = c$
by *simp*
qed

Using the max-aggregator for composing two compatible modules in parallel, whereof the first one is non-electing and defers exactly one alternative, and the second one rejects exactly two alternatives, the composition results in an electoral module that eliminates exactly one alternative.

theorem *par-comp-elim-one*[*simp*]:
fixes
m :: *'a Electoral-Module* **and**
n :: *'a Electoral-Module*
assumes
defers-m-one: *defers 1 m* **and**
non-elec-m: *non-electing m* **and**
rejec-n-two: *rejects 2 n* **and**

disj-comp: *disjoint-compatibility m n*
shows *eliminates 1* ($m \parallel_\uparrow n$)
proof (*unfold eliminates-def*, *safe*)
have *electoral-mod-m*: *electoral-module m*
using *non-elec-m*
unfolding *non-electing-def*
by *simp*
have *electoral-mod-n*: *electoral-module n*
using *rejec-n-two*
unfolding *rejects-def*
by *simp*
show *electoral-module* ($m \parallel_\uparrow n$)
using *electoral-mod-m electoral-mod-n*
by *simp*
next
fix
A :: *'a set* **and**
p :: *'a Profile*
assume
min-card-two: *1* < *card A* **and**
fin-A: *finite A* **and**
prof-A: *profile A p*
have *card-geq-one*: *card A* ≥ *1*
using *min-card-two dual-order.strict-trans2 less-imp-le-nat*
by *blast*
have *module*: *electoral-module m*
using *non-elec-m*
unfolding *non-electing-def*
by *simp*
have *elec-card-zero*: *card* (*elect m A p*) = *0*
using *fin-A prof-A non-elec-m card-eq-0-iff*
unfolding *non-electing-def*
by *simp*
moreover from *card-geq-one*
have *def-card-one*: *card* (*defer m A p*) = *1*
using *defers-m-one module fin-A prof-A*
unfolding *defers-def*
by *simp*
ultimately have *card-reject-m*:
card (*reject m A p*) = *card A* − *1*
proof −
have *finite A*
using *fin-A*
by *simp*
moreover have *well-formed A* (*elect m A p*, *reject m A p*, *defer m A p*)
using *fin-A prof-A module*
unfolding *electoral-module-def*
by *simp*
ultimately have

card A = *card* (*elect m A p*) + *card* (*reject m A p*) + *card* (*defer m A p*)
using *result-count*
by *blast*
thus *?thesis*
using *def-card-one elec-card-zero*
by *simp*
qed
have *case-1*: *card A* $\geq$ *2*
using *min-card-two*
by *simp*
from *case-1*
have *card-reject-n*: *card* (*reject n A p*) = *2*
using *fin-A prof-A rejec-n-two*
unfolding *rejects-def*
by *blast*
from *card-reject-m card-reject-n*
have *card* (*reject m A p*) + *card* (*reject n A p*) = *card A* + *1*
using *card-geq-one*
by *linarith*
with *disj-comp prof-A fin-A card-reject-m card-reject-n*
show *card* (*reject* (*m* $\|_\uparrow$ *n*) *A p*) = *1*
using *par-comp-rej-card*
by *blast*
qed

end

4.7 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.

4.7.1 Definition

fun *elector* :: *'a Electoral-Module* $\Rightarrow$ *'a Electoral-Module* **where**
elector m = (*m* $\triangleright$ *elect-module*)

4.7.2 Soundness

```
theorem elector-sound[simp]:
  fixes m :: 'a Electoral-Module
  assumes electoral-module m
  shows electoral-module (elector m)
  using assms
  by simp
```

4.7.3 Electing

```
theorem elector-electing[simp]:
  fixes m :: 'a Electoral-Module
  assumes
    module-m: electoral-module m and
    non-block-m: non-blocking m
  shows electing (elector m)
proof -
  have non-block:
    non-blocking
      (elect-module::'a set ⇒ - Profile ⇒ - Result)
    by (simp add: electing-imp-non-blocking)
  obtain
    alts :: 'a Electoral-Module ⇒ 'a set and
    prof :: 'a Electoral-Module ⇒ 'a Profile where
    electing-func:
    ∀ f.
      (¬ electing f ∧ electoral-module f ⟶
        profile (alts f) (prof f) ∧ finite (alts f) ∧
          {} = elect f (alts f) (prof f)  ∧ {} ≠ alts f) ∧
      (electing f ∧ electoral-module f ⟶
        (∀ A p. (A ≠ {} ∧ profile A p ∧ finite A) ⟶ elect f A p ≠ {}))
    using electing-def
    by metis
  obtain
    ele :: 'a Result ⇒ 'a set and
    rej :: 'a Result ⇒ 'a set and
    def :: 'a Result ⇒ 'a set where
    result: ∀ r. (ele r, rej r, def r) = r
    using disjoint3.cases
    by (metis (no-types))
  hence r-func:
    ∀ r. (elect-r r, rej r, def r) = r
    by simp
  hence def-empty:
    profile (alts (elector m)) (prof (elector m)) ∧ finite (alts (elector m)) ⟶
      def (elector m (alts (elector m)) (prof (elector m))) = {}
    by simp
  have elec-mod:
    electoral-module (elector m)
```

using *elector-sound module-m*
by *simp*
have
finite (*alts* (*elector m*)) ∧
profile (*alts* (*elector m*)) (*prof* (*elector m*)) ∧
elect (*elector m*) (*alts* (*elector m*)) (*prof* (*elector m*)) = {} ∧
def (*elector m* (*alts* (*elector m*)) (*prof* (*elector m*))) = {} ∧
reject (*elector m*) (*alts* (*elector m*)) (*prof* (*elector m*)) =
rej (*elector m* (*alts* (*elector m*)) (*prof* (*elector m*))) ⟶
electing (*elector m*)
using *result electing-func Diff-empty elector.simps non-block-m snd-conv non-blocking-def reject-not-elec-or-def non-block seq-comp-presv-non-blocking*
by *metis*
thus *?thesis*
using *r-func def-empty elec-mod electing-func fst-conv snd-conv*
by *metis*
qed

4.7.4 Composition Rule

If m is defer-Condorcet-consistent, then elector(m) is Condorcet consistent.

lemma *dcc-imp-cc-elector*:
fixes *m* :: *'a Electoral-Module*
assumes *defer-condorcet-consistency m*
shows *condorcet-consistency* (*elector m*)
proof (*unfold defer-condorcet-consistency-def condorcet-consistency-def*, *auto*)
show *electoral-module* (*m* ▷ *elect-module*)
using *assms elect-mod-sound seq-comp-sound*
unfolding *defer-condorcet-consistency-def*
by *metis*
next
show
$\bigwedge$ *A p w x*.
finite A ⟹ *profile A p* ⟹ *w* ∈ *A* ⟹
∀ *x* ∈ *A* − {*w*}. *card* {*i*. *i* < *length p* ∧ (*w*, *x*) ∈ (*p*!*i*)} <
card {*i*. *i* < *length p* ∧ (*x*, *w*) ∈ (*p*!*i*)} ⟹
x ∈ *elect m A p* ⟹ *x* ∈ *A*
proof −
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
w :: *'a* **and**
x :: *'a*
assume
finite: *finite A* **and**
prof-A: *profile A p*
show

$\forall\ y \in A - \{w\}.$
$\quad card\ \{i.\ i < length\ p \wedge (w, y) \in (p!i)\} <$
$\quad\quad card\ \{i.\ i < length\ p \wedge (y, w) \in (p!i)\} \Longrightarrow$
$\quad\quad x \in elect\ m\ A\ p \Longrightarrow x \in A$
using *assms elect-in-alts subset-eq finite prof-A*
unfolding *defer-condorcet-consistency-def*
by *metis*
qed
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$w :: 'a$ **and**
$x :: 'a$ **and**
$xa :: 'a$
assume
finite: *finite* A **and**
prof-A: *profile* $A\ p$ **and**
w-in-A: $w \in A$ **and**
1: $x \in elect\ m\ A\ p$ **and**
2: $\forall\ y \in A - \{w\}.$
$\quad card\ \{i.\ i < length\ p \wedge (w, y) \in (p!i)\} <$
$\quad\quad card\ \{i.\ i < length\ p \wedge (y, w) \in (p!i)\}$
have *condorcet-winner* $A\ p\ w$
using *finite prof-A w-in-A 2*
by *simp*
thus $xa = x$
using *condorcet-winner.simps assms fst-conv insert-Diff 1 insert-not-empty*
unfolding *defer-condorcet-consistency-def*
by (*metis* (*no-types*, *lifting*))
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$w :: 'a$ **and**
$x :: 'a$
assume
finite: *finite* A **and**
prof-A: *profile* $A\ p$ **and**
w-in-A: $w \in A$ **and**
0: $\forall\ y \in A - \{w\}.$
$\quad card\ \{i.\ i < length\ p \wedge (w, y) \in (p!i)\} <$
$\quad\quad card\ \{i.\ i < length\ p \wedge (y, w) \in (p!i)\}$ **and**
1: $x \in defer\ m\ A\ p$
have *condorcet-winner* $A\ p\ w$
using *finite prof-A w-in-A 0*
by *simp*
thus $x \in A$
using *0 1 condorcet-winner.simps assms defer-in-alts*

order-trans subset-Compl-singleton
unfolding *defer-condorcet-consistency-def*
by (*metis* (*no-types*, *lifting*))
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$ **and**
x :: $'a$ **and**
xa :: $'a$
assume
finite: *finite* A **and**
prof-A: *profile* A p **and**
w-in-A: $w \in A$ **and**
1: $x \in$ *defer* m A p **and**
xa-in-A: $xa \in A$ **and**
2: $\forall\ y \in A - \{w\}$.
card $\{i.\ i <$ *length* $p \land (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \land (y, w) \in (p!i)\}$ **and**
3: $\neg$ *card* $\{i.\ i <$ *length* $p \land (x, xa) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \land (xa, x) \in (p!i)\}$
have *condorcet-winner* A p w
using *finite prof-A w-in-A 2*
by *simp*
thus $xa = x$
using *1 2 condorcet-winner.simps assms empty-iff xa-in-A*
defer-condorcet-consistency-def 3 DiffI
cond-winner-unique-3 insert-iff prod.sel(*2*)
by (*metis* (*no-types*, *lifting*))
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$ **and**
x :: $'a$
assume
finite: *finite* A **and**
prof-A: *profile* A p **and**
w-in-A: $w \in A$ **and**
x-in-A: $x \in A$ **and**
1: $x \notin$ *defer* m A p **and**
2: $\forall\ y \in A - \{w\}$.
card $\{i.\ i <$ *length* $p \land (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \land (y, w) \in (p!i)\}$ **and**
3: $\forall\ y \in A - \{x\}$.
card $\{i.\ i <$ *length* $p \land (x, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \land (y, x) \in (p!i)\}$
have *condorcet-winner* A p w
using *finite prof-A w-in-A 2*

by *simp*
also have *condorcet-winner A p x*
using *finite prof-A x-in-A 3*
by *simp*
ultimately show $x \in$ *elect m A p*
using *1 condorcet-winner.simps assms*
defer-condorcet-consistency-def
cond-winner-unique-3 insert-iff eq-snd-iff
by (*metis* (*no-types*, *lifting*))
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$ **and**
x :: $'a$
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
1: $x \in$ *reject m A p* **and**
2: $\forall\ y \in A - \{w\}$.
card $\{i.\ i <$ *length* $p \wedge (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (y, w) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*
thus $x \in A$
using *1 assms finite prof-A reject-in-alts subsetD*
unfolding *defer-condorcet-consistency-def*
by *metis*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$ **and**
x :: $'a$
assume
finite: *finite A* **and**
prof-A: *profile A p* **and**
w-in-A: $w \in A$ **and**
0: $x \in$ *reject m A p* **and**
1: $x \in$ *elect m A p* **and**
2: $\forall\ y \in A - \{w\}$.
card $\{i.\ i <$ *length* $p \wedge (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (y, w) \in (p!i)\}$
have *condorcet-winner A p w*
using *finite prof-A w-in-A 2*
by *simp*
thus *False*

```
    using 0 1 assms IntI empty-iff result-disj
    unfolding condorcet-winner.simps defer-condorcet-consistency-def
    by (metis (no-types, opaque-lifting))
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a and
    x :: 'a
  assume
    finite: finite A and
    prof-A: profile A p and
    w-in-A: w ∈ A and
    0: x ∈ reject m A p and
    1: x ∈ defer m A p and
    2: ∀ y ∈ A − {w}.
          card {i. i < length p ∧ (w, y) ∈ (p!i)} <
            card {i. i < length p ∧ (y, w) ∈ (p!i)}
  have condorcet-winner A p w
    using finite prof-A w-in-A 2
    by simp
  thus False
    using 0 1 assms IntI Diff-empty Diff-iff finite prof-A result-disj
    unfolding defer-condorcet-consistency-def
    by (metis (no-types, opaque-lifting))
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a and
    x :: 'a
  assume
    finite: finite A and
    prof-A: profile A p and
    w-in-A: w ∈ A and
    x-in-A: x ∈ A and
    0: x ∉ reject m A p and
    1: x ∉ defer m A p and
    2: ∀ y ∈ A − {w}.
          card {i. i < length p ∧ (w, y) ∈ (p!i)} <
            card {i. i < length p ∧ (y, w) ∈ (p!i)}
  have condorcet-winner A p w
    using finite prof-A w-in-A 2
    by simp
  thus x ∈ elect m A p
    using 0 1 assms x-in-A electoral-mod-defer-elem
    unfolding condorcet-winner.simps defer-condorcet-consistency-def
    by (metis (no-types, lifting))
qed
```

end

4.8 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.

4.8.1 Definition

fun *iter* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Electoral-Module* **where**
  iter m =
    (*let t* = *defer-equal-condition 1 in*
      (m $\circlearrowleft_t$))

abbreviation *defer-one-loop* ::
  $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Electoral-Module*
    (-$\circlearrowleft_{\exists !d}$ *50*) **where**
  m $\circlearrowleft_{\exists !d}$ $\equiv$ *iter m*

fun *iterelect* :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Electoral-Module* **where**
  iterelect m = *elector* (m $\circlearrowleft_{\exists !d}$)

end

Chapter 5

Voting Rules

5.1 Borda Rule

theory *Borda-Rule*
imports *Compositional-Structures/Basic-Modules/Borda-Module*
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.

5.1.1 Definition

fun *borda-rule* :: *'a Electoral-Module* **where**
borda-rule A p = *elector borda A p*

5.1.2 Soundness

theorem *borda-rule-sound*: *electoral-module borda-rule*
unfolding *borda-rule.simps*
using *elector-sound borda-sound*
by *metis*

end

5.2 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.

5.2.1 Definition

fun *pairwise-majority-rule* :: *'a Electoral-Module* **where**
pairwise-majority-rule A p = elector condorcet A p

fun *condorcet'* :: *'a Electoral-Module* **where**
condorcet' A p =
$((\textit{min-eliminator condorcet-score}) \circlearrowleft_{\exists !d})\ A\ p$

fun *pairwise-majority-rule'* :: *'a Electoral-Module* **where**
pairwise-majority-rule' A p = iterelect condorcet' A p

5.2.2 Soundness

theorem *pairwise-majority-rule-sound*: *electoral-module pairwise-majority-rule*
unfolding *pairwise-majority-rule.simps*
using *condorcet-sound elector-sound*
by *metis*

theorem *condorcet'-rule-sound*: *electoral-module condorcet'*
unfolding *condorcet'.simps*
by (*simp add*: *loop-comp-sound*)

theorem *pairwise-majority-rule'-sound*: *electoral-module pairwise-majority-rule'*
unfolding *pairwise-majority-rule'.simps*
using *condorcet'-rule-sound elector-sound iter.simps iterelect.simps loop-comp-sound*
by *metis*

5.2.3 Condorcet Consistency Property

theorem *condorcet-condorcet*: *condorcet-consistency pairwise-majority-rule*
proof (*unfold pairwise-majority-rule.simps*)
show *condorcet-consistency* (*elector condorcet*)
using *condorcet-is-dcc dcc-imp-cc-elector*
by *metis*
qed

end

5.3 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.

5.3.1 Definition

fun *copeland-rule* :: *'a Electoral-Module* **where**
  copeland-rule A p = elector copeland A p

5.3.2 Soundness

theorem *copeland-rule-sound*: *electoral-module copeland-rule*
  unfolding *copeland-rule.simps*
  using *elector-sound copeland-sound*
  by *metis*

5.3.3 Condorcet Consistency Property

theorem *copeland-condorcet*: *condorcet-consistency copeland-rule*
proof (*unfold copeland-rule.simps*)
  show *condorcet-consistency* (*elector copeland*)
    using *copeland-is-dcc dcc-imp-cc-elector*
    by *metis*
qed

end

5.4 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.

5.4.1 Definition

fun *minimax-rule* :: *'a Electoral-Module* **where**
minimax-rule A p = elector minimax A p

5.4.2 Soundness

theorem *minimax-rule-sound*: *electoral-module minimax-rule*
unfolding *minimax-rule.simps*
using *elector-sound minimax-sound*
by *metis*

5.4.3 Condorcet Consistency Property

theorem *minimax-condorcet*: *condorcet-consistency minimax-rule*
proof (*unfold minimax-rule.simps*)
show *condorcet-consistency* (*elector minimax*)
using *minimax-is-dcc dcc-imp-cc-elector*
by *metis*
qed

end

5.5 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.

5.5.1 Definition

fun *blacks-rule* :: *'a Electoral-Module* **where**
blacks-rule A p = (*pairwise-majority-rule* $\triangleright$ *borda-rule*) *A p*

5.5.2 Soundness

theorem *blacks-rule-sound*: *electoral-module blacks-rule*
unfolding *blacks-rule.simps*
using *pairwise-majority-rule-sound borda-rule-sound seq-comp-sound*
by *metis*

end

5.6 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.

5.6.1 Definition

fun *nanson-baldwin-rule* :: *'a Electoral-Module* **where**
nanson-baldwin-rule A p =
((*min-eliminator borda-score*) $\circlearrowleft_{\exists !d}$) *A p*

5.6.2 Soundness

theorem *nanson-baldwin-rule-sound*: *electoral-module nanson-baldwin-rule*
unfolding *nanson-baldwin-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

5.7 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.

5.7.1 Definition

fun *classic-nanson-rule* :: *'a Electoral-Module* **where**
classic-nanson-rule A p =
((*leq-average-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) *A p*

5.7.2 Soundness

theorem *classic-nanson-rule-sound*: *electoral-module classic-nanson-rule*
unfolding *classic-nanson-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

5.8 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.

5.8.1 Definition

fun *schwartz-rule* :: *'a Electoral-Module* **where**
schwartz-rule A p =
((*less-average-eliminator borda-score*) $\circlearrowleft_{\exists!d}$) *A p*

5.8.2 Soundness

theorem *schwartz-rule-sound*: *electoral-module schwartz-rule*
unfolding *schwartz-rule.simps*
by (*simp add*: *loop-comp-sound*)

end

5.9 Sequential Majority Comparison

theory *Sequential-Majority-Comparison*
  imports *Compositional-Structures/Basic-Modules/Plurality-Module*
    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.

5.9.1 Definition

fun *smc* :: *'a Preference-Relation* ⇒ *'a Electoral-Module* **where**
  smc x A p =
    $((((((\textit{pass-module } 2\ x) \triangleright ((\textit{plurality}\downarrow) \triangleright (\textit{pass-module } 1\ x))) \parallel_{\uparrow}$
    $(\textit{drop-module } 2\ x)) \circlearrowleft_{\exists !d}) \triangleright \textit{elect-module})\ A\ p)$

5.9.2 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*
  assumes *order*: *linear-order x*
  shows *electoral-module* (*smc x*)
proof (*unfold electoral-module-def*, *simp*, *safe*, *simp-all*)
  fix
    A :: *'a set* **and**
    p :: *'a Profile* **and**
    xa :: *'a*
  let *?a* = *max-aggregator*
  let *?t* = *defer-equal-condition*
  let *?smc* =
    pass-module 2 x ▷
      $((\textit{plurality}\downarrow) \triangleright \textit{pass-module } (\textit{Suc } 0)\ x) \parallel_{?a}$
        $\textit{drop-module } 2\ x \circlearrowleft_{?t} (\textit{Suc } 0)$
  assume
    fin-A: *finite A* **and**
    prof-A: *profile A p* **and**
    reject-xa: *xa* ∈ *reject* (*?smc*) *A p* **and**
    elect-xa: *xa* ∈ *elect* (*?smc*) *A p*
  show *False*

using *IntI drop-mod-sound elect-xa emptyE fin-A*
loop-comp-sound max-agg-sound order prof-A
par-comp-sound pass-mod-sound reject-xa
plurality-sound result-disj rev-comp-sound
seq-comp-sound
by *metis*
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$xa :: 'a$
let $?a = max\text{-}aggregator$
let $?t = defer\text{-}equal\text{-}condition$
let $?smc =$
$pass\text{-}module\ 2\ x \triangleright$
$((plurality\downarrow) \triangleright pass\text{-}module\ (Suc\ 0)\ x) \parallel_{?}a$
$drop\text{-}module\ 2\ x \circlearrowleft_{?}t\ (Suc\ 0)$
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
reject-xa: $xa \in reject\ (?smc)\ A\ p$ **and**
defer-xa: $xa \in defer\ (?smc)\ A\ p$
show *False*
using *IntI drop-mod-sound defer-xa emptyE fin-A*
loop-comp-sound max-agg-sound order prof-A
par-comp-sound pass-mod-sound reject-xa
plurality-sound result-disj rev-comp-sound
seq-comp-sound
by *metis*
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$xa :: 'a$
let $?a = max\text{-}aggregator$
let $?t = defer\text{-}equal\text{-}condition$
let $?smc =$
$pass\text{-}module\ 2\ x \triangleright$
$((plurality\downarrow) \triangleright pass\text{-}module\ (Suc\ 0)\ x) \parallel_{?}a$
$drop\text{-}module\ 2\ x \circlearrowleft_{?}t\ (Suc\ 0)$
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
elect-xa:
$xa \in elect\ (?smc)\ A\ p$
show $xa \in A$
using *drop-mod-sound elect-in-alts elect-xa fin-A*
in-mono loop-comp-sound max-agg-sound order
par-comp-sound pass-mod-sound plurality-sound

prof-A rev-comp-sound seq-comp-sound
by *metis*
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
xa :: *'a*
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x ↺$_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
defer-xa: *xa* ∈ *defer* (*?smc*) *A p*
show *xa* ∈ *A*
using *drop-mod-sound defer-in-alts defer-xa fin-A*
in-mono loop-comp-sound max-agg-sound order
par-comp-sound pass-mod-sound plurality-sound
prof-A rev-comp-sound seq-comp-sound
by (*metis* (*no-types*, *lifting*))
next
fix
A :: *'a set* **and**
p :: *'a Profile* **and**
xa :: *'a*
let *?a* = *max-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x ↺$_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
reject-xa:
xa ∈ *reject* (*?smc*) *A p*
have *plurality-rev-sound*:
electoral-module
(*plurality*::*'a set* ⇒ (- × -) *set list* ⇒ - *set* × - *set* × - *set*↓)
by *simp*
have *par1-sound*:
electoral-module (*pass-module 2 x* ▷ ((*plurality*↓) ▷ *pass-module 1 x*))
using *order*
by *simp*
also have *par2-sound*:
electoral-module (*drop-module 2 x*)

```
    using order
    by simp
  show xa ∈ A
    using reject-in-alts reject-xa fin-A in-mono
          loop-comp-sound max-agg-sound order
          par-comp-sound pass-mod-sound prof-A
          seq-comp-sound pass-mod-sound par1-sound
          par2-sound plurality-rev-sound
    by (metis (no-types))
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    xa :: 'a
  let ?a = max-aggregator
  let ?t = defer-equal-condition
  let ?smc =
    pass-module 2 x ▹
      ((plurality↓) ▹ pass-module (Suc 0) x) ∥?a
        drop-module 2 x ↻?t (Suc 0)
  assume
    fin-A: finite A and
    prof-A: profile A p and
    xa-in-A: xa ∈ A and
    not-defer-xa:
      xa ∉ defer (?smc) A p and
    not-reject-xa:
      xa ∉ reject (?smc) A p
  show xa ∈ elect (?smc) A p
    using drop-mod-sound loop-comp-sound max-agg-sound
          order par-comp-sound pass-mod-sound xa-in-A
          plurality-sound rev-comp-sound seq-comp-sound
          electoral-mod-defer-elem fin-A not-defer-xa
          not-reject-xa prof-A
    by metis
qed
```

5.9.3 Electing

The sequential majority comparison electoral module is electing. This property is needed to convert electoral modules to a social choice function. Apart from the very last proof step, it is a part of the monotonicity proof below.

```
theorem smc-electing:
  fixes x :: 'a Preference-Relation
  assumes linear-order x
  shows electing (smc x)
proof −
  let ?pass2 = pass-module 2 x
  let ?tie-breaker = (pass-module 1 x)
```

let *?plurality-defer* = (*plurality*↓) ▷ *?tie-breaker*
let *?compare-two* = *?pass2* ▷ *?plurality-defer*
let *?drop2* = *drop-module 2 x*
let *?eliminator* = *?compare-two* $\|_\uparrow$ *?drop2*
let *?loop* =
  let t = *defer-equal-condition 1 in* (*?eliminator* $\circlearrowleft_t$)

have *00011*: *non-electing* (*plurality*↓)
  by *simp*
have *00012*: *non-electing ?tie-breaker*
  using *assms*
  by *simp*
have *00013*: *defers 1 ?tie-breaker*
  using *assms pass-one-mod-def-one*
  by *simp*
have *20000*: *non-blocking* (*plurality*↓)
  by *simp*

have *0020*: *disjoint-compatibility ?pass2 ?drop2*
  using *assms*
  by *simp*
have *1000*: *non-electing ?pass2*
  using *assms*
  by *simp*
have *1001*: *non-electing ?plurality-defer*
  using *00011 00012*
  by *simp*
have *2000*: *non-blocking ?pass2*
  using *assms*
  by *simp*
have *2001*: *defers 1 ?plurality-defer*
  using *20000 00011 00013 seq-comp-def-one*
  by *blast*

have *002*: *disjoint-compatibility ?compare-two ?drop2*
  using *assms 0020*
  by *simp*
have *100*: *non-electing ?compare-two*
  using *1000 1001*
  by *simp*
have *101*: *non-electing ?drop2*
  using *assms*
  by *simp*
have *102*: *agg-conservative max-aggregator*
  by *simp*
have *200*: *defers 1 ?compare-two*
  using *2000 1000 2001 seq-comp-def-one*
  by *auto*
have *201*: *rejects 2 ?drop2*

using *assms*
by *simp*

have *10*: *non-electing ?eliminator*
using *100 101 102*
by *simp*
have *20*: *eliminates 1 ?eliminator*
using *200 100 201 002 par-comp-elim-one*
by *metis*

have *2*: *defers 1 ?loop*
using *10 20*
by *simp*
have *3*: *electing elect-module*
by *simp*

show *?thesis*
using *2 3 assms seq-comp-electing smc-sound*
unfolding *Defer-One-Loop-Composition.iter.simps smc.simps electing-def*
by *metis*
qed

5.9.4 (Weak) Monotonicity Property

The following proof is a fully modular proof for weak monotonicity of sequential majority comparison. It is composed of many small steps.

theorem *smc-monotone*:
fixes x :: *'a Preference-Relation*
assumes *linear-order x*
shows *monotonicity* (*smc x*)
proof −
let *?pass2* = *pass-module 2 x*
let *?tie-breaker* = (*pass-module 1 x*)
let *?plurality-defer* = (*plurality*↓) ▷ *?tie-breaker*
let *?compare-two* = *?pass2* ▷ *?plurality-defer*
let *?drop2* = *drop-module 2 x*
let *?eliminator* = *?compare-two* $\|_\uparrow$ *?drop2*
let *?loop* =
let t = defer-equal-condition 1 in (*?eliminator* $\circlearrowleft_t$)

have *00010*: *defer-invariant-monotonicity* (*plurality*↓)
by *simp*
have *00011*: *non-electing* (*plurality*↓)
by *simp*
have *00012*: *non-electing ?tie-breaker*
using *assms*
by *simp*
have *00013*: *defers 1 ?tie-breaker*

using *assms pass-one-mod-def-one*
by *simp*
have *00014*: *defer-monotonicity ?tie-breaker*
using *assms*
by *simp*
have *20000*: *non-blocking* (*plurality*↓)
by *simp*

have *0000*: *defer-lift-invariance ?pass2*
using *assms*
by *simp*
have *0001*: *defer-lift-invariance ?plurality-defer*
using *00010 00011 00012 00013 00014*
by *simp*
have *0020*: *disjoint-compatibility ?pass2 ?drop2*
using *assms*
by *simp*
have *1000*: *non-electing ?pass2*
using *assms*
by *simp*
have *1001*: *non-electing ?plurality-defer*
using *00011 00012*
by *simp*
have *2000*: *non-blocking ?pass2*
using *assms*
by *simp*
have *2001*: *defers 1 ?plurality-defer*
using *20000 00011 00013 seq-comp-def-one*
by *blast*

have *000*: *defer-lift-invariance ?compare-two*
using *0000 0001*
by *simp*
have *001*: *defer-lift-invariance ?drop2*
using *assms*
by *simp*
have *002*: *disjoint-compatibility ?compare-two ?drop2*
using *assms 0020*
by *simp*

have *100*: *non-electing ?compare-two*
using *1000 1001*
by *simp*
have *101*: *non-electing ?drop2*
using *assms*
by *simp*
have *102*: *agg-conservative max-aggregator*
by *simp*
have *200*: *defers 1 ?compare-two*

```
    using 2000 1000 2001 seq-comp-def-one
    by auto
  have 201: rejects 2 ?drop2
    using assms
    by simp

  have 00: defer-lift-invariance ?eliminator
    using 000 001 002 par-comp-def-lift-inv
    by simp
  have 10: non-electing ?eliminator
    using 100 101 102
    by simp
  have 20: eliminates 1 ?eliminator
    using 200 100 201 002 par-comp-elim-one
    by simp

  have 0: defer-lift-invariance ?loop
    using 00
    by simp
  have 1: non-electing ?loop
    using 10
    by simp
  have 2: defers 1 ?loop
    using 10 20
    by simp
  have 3: electing elect-module
    by simp

  show ?thesis
    using 0 1 2 3 assms seq-comp-mono
    unfolding Electoral-Module.monotonicity-def
          Defer-One-Loop-Composition.iter.simps
          smc-sound smc.simps
    by (metis (full-types))
qed

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.