Verified Construction of Fair Voting Rules

Michael Kirsten

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

December 16, 2021

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, lemmata, operations on preference relations, etc.

1.1.1 Definition

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*:
  assumes *linear-order-on A r*
  shows *antisym r*
  using *assms linear-order-on-def partial-order-on-def*
  by *auto*

lemma *lin-imp-trans*:
  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*:
  assumes
    refl: $x \preceq_r x$ **and**
    fin: *finite r*
  shows *rank r x* $\geq$ *1*
proof $-$
  have $x \in \{y \in \textit{Field } r.\ (x, y) \in r\}$
    using *FieldI2 refl*
    by *fastforce*
  hence $\{y \in \textit{Field } r.\ (x, y) \in r\} \neq \{\}$
    by *blast*
  hence $\textit{card } \{y \in \textit{Field } r.\ (x, y) \in r\} \neq \textit{0}$
    by (*simp add*: *fin finite-Field*)
  moreover have $\textit{card}\{y \in \textit{Field } r.\ (x, y) \in r\} \geq \textit{0}$
    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**
  $\textit{limited } A\ r \equiv r \subseteq A \times A$

lemma *limitedI*:
  $(\bigwedge x\ y.\ [\![\ x \preceq_r y\ ]\!] \Longrightarrow x \in A \wedge y \in A) \Longrightarrow \textit{limited } A\ r$
  unfolding *limited-def*
  by *auto*

lemma *limited-dest*:
  $(\bigwedge x\ y.\ [\![\ x \preceq_r y;\ \textit{limited } A\ r\ ]\!] \Longrightarrow x \in A \wedge y \in A)$
  unfolding *limited-def*
  by *auto*

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

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

lemma *connex-imp-refl*:
  assumes *connex A r*
  shows *refl-on A r*
proof
  show $r \subseteq A \times A$
    using *assms connex-def limited-def*

by *metis*
next
fix
x :: $'a$
assume
x-in-A: $x \in A$
have $x \preceq_r x$
using *assms connex-def x-in-A*
by *metis*
thus $(x, x) \in r$
by *simp*
qed

lemma *lin-ord-imp-connex*:
assumes *linear-order-on* A r
shows *connex* A r
unfolding *connex-def limited-def*
proof (*safe*)
fix
a :: $'a$ **and**
b :: $'a$
assume
asm1: $(a, b) \in r$
show $a \in A$
using *asm1 assms partial-order-onD*(*1*) *order-on-defs*(*3*) *refl-on-domain*
by *metis*
next
fix
a :: $'a$ **and**
b :: $'a$
assume
asm1: $(a, b) \in r$
show $b \in A$
using *asm1 assms partial-order-onD*(*1*) *order-on-defs*(*3*) *refl-on-domain*
by *metis*
next
fix
x :: $'a$ **and**
y :: $'a$
assume
asm1: $x \in A$ **and**
asm2: $y \in A$ **and**
asm3: $\neg\ y \preceq_r x$
have $(y, x) \notin r$
using *asm3*
by *simp*
hence $(x, y) \in r$

```
    using asm1 asm2 assms partial-order-onD(1)
          linear-order-on-def refl-onD total-on-def
    by metis
  thus x ≼_r y
    by simp
qed

lemma connex-antsym-and-trans-imp-lin-ord:
  assumes
    connex-r: connex A r and
    antisym-r: antisym r and
    trans-r: trans r
  shows linear-order-on A r
  unfolding connex-def linear-order-on-def partial-order-on-def
            preorder-on-def refl-on-def total-on-def
proof (safe)
  fix
    a :: 'a and
    b :: 'a
  assume
    asm1: (a, b) ∈ r
  show a ∈ A
    using asm1 connex-r refl-on-domain connex-imp-refl
    by metis
next
  fix
    a :: 'a and
    b :: 'a
  assume
    asm1: (a, b) ∈ r
  show b ∈ A
    using asm1 connex-r refl-on-domain connex-imp-refl
    by metis
next
  fix
    x :: 'a
  assume
    asm1: x ∈ A
  show (x, x) ∈ r
    using asm1 connex-r connex-imp-refl refl-onD
  by metis
next
  show trans r
    using trans-r
    by simp
next
  show antisym r
    using antisym-r
    by simp
```

next
  fix
    $x :: 'a$ **and**
    $y :: 'a$
  assume
    asm1: $x \in A$ **and**
    asm2: $y \in A$ **and**
    asm3: $x \neq y$ **and**
    asm4: $(y, x) \notin r$
  have $x \preceq_r y \vee y \preceq_r x$
    using *asm1 asm2 connex-r connex-def*
    by *metis*
  hence $(x, y) \in r \vee (y, x) \in r$
    by *simp*
  thus $(x, y) \in r$
    using *asm4*
    by *metis*
qed

lemma *limit-to-limits*: *limited* A (*limit* A r)
  unfolding *limited-def*
  by *auto*

lemma *limit-presv-connex*:
  assumes
    connex: *connex* S r **and**
    subset: $A \subseteq S$
  shows *connex* A (*limit* A r)
  unfolding *connex-def limited-def*
proof (*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
    asm1: $x \in A$ **and**
    asm2: $y \in A$ **and**
    asm3: $(y, x) \notin r$
  have $y \preceq_r x \vee x \preceq_r y$
    using *asm1 asm2 connex connex-def in-mono subset*
    by *metis*
  hence
    $x \preceq_{?s} y \vee y \preceq_{?s} x$
    using *asm1 asm2*
    by *auto*
  hence $x \preceq_{?s} y$
    using *asm3*

```
    by simp
  thus (x, y) ∈ r
    by simp
qed

lemma limit-presv-antisym:
  assumes
    antisymmetric: antisym r and
    subset: A ⊆ S
  shows antisym (limit A r)
  using antisym-def antisymmetric
  by auto

lemma limit-presv-trans:
  assumes
    transitive: trans r and
    subset:      A ⊆ S
  shows trans (limit A r)
  unfolding trans-def
proof (simp, safe)
  fix
    x :: 'a and
    y :: 'a and
    z :: 'a
  assume
    asm1: (x, y) ∈ r and
    asm2: x ∈ A and
    asm3: y ∈ A and
    asm4: (y, z) ∈ r and
    asm5: z ∈ A
  show  (x, z) ∈ r
    using asm1 asm4 transE transitive
    by metis
qed

lemma limit-presv-lin-ord:
  assumes
    linear-order-on S r and
      A ⊆ 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) order-on-defs(2)
          order-on-defs(3)
  by metis

lemma limit-presv-prefs1:
  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-prefs2*:
assumes *x-less-y*: $(x, y) \in$ *limit* A r
shows $x \preceq_r y$
using *mem-Collect-eq x-less-y*
by *auto*

lemma *limit-trans*:
assumes
$B \subseteq A$ **and**
$C \subseteq B$ **and**
linear-order-on A r
shows *limit* C $r =$ *limit* C (*limit* B r)
using *assms*
by *auto*

lemma *lin-ord-not-empty*:
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*:
$\forall r.$ *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof
fix r :: *'a Preference-Relation*
show *linear-order-on* $\{a\}$ $r \longrightarrow r = \{(a, a)\}$
proof
assume *asm*: *linear-order-on* $\{a\}$ r
hence $a \preceq_r a$
using *connex-def lin-ord-imp-connex singletonI*
by *metis*
moreover have $\forall (x, y) \in r.\ x = a \wedge y = a$
using *asm connex-imp-refl lin-ord-imp-connex*
refl-on-domain split-beta
by *fastforce*
ultimately show $r = \{(a, a)\}$
by *auto*
qed
qed

1.1.4 Auxiliary Lemmata

lemma *above-trans*:
  assumes
    trans r **and**
    $(a, b) \in r$
  shows *above r b* $\subseteq$ *above r a*
  using *Collect-mono above-def assms transE*
  by *metis*

lemma *above-refl*:
  assumes
    refl-on A r **and**
    $a \in A$
  shows $a \in$ *above r a*
  using *above-def assms refl-onD*
  by *fastforce*

lemma *above-subset-geq-one*:
  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 *above-def assms connex-imp-refl above-refl insert-absorb*
    lin-ord-imp-connex mem-Collect-eq refl-on-domain
    singletonI subset-singletonD
  by *metis*

lemma *above-connex*:
  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*: $a \preceq_r b \longleftrightarrow b \in$ *above r a*
  by (*simp add*: *above-def*)

lemma *limit-presv-above*:
  assumes
    $b \in$ *above r a* **and**

    $a \in B \wedge b \in B$
  shows $b \in$ *above* (*limit B r*) *a*
  using *pref-imp-in-above assms limit-presv-prefs1*
  by *metis*

lemma *limit-presv-above2*:

```
  assumes
    b ∈ above (limit B r) a and
    linear-order-on A r and
    B ⊆ A and
    a ∈ B and
    b ∈ B
  shows b ∈ above r a
  unfolding above-def
  using above-def assms(1) limit-presv-prefs2
        mem-Collect-eq pref-imp-in-above
  by metis

lemma above-one:
  assumes
    linear-order-on A r and
    finite A ∧ A ≠ {}
  shows ∃ a∈A. above r a = {a} ∧ (∀ x∈A. above r x = {x} ⟶ x = a)
proof –
  obtain n::nat where n: n+1 = card A
    using Suc-eq-plus1 antisym-conv2 assms(2) card-eq-0-iff
          gr0-implies-Suc le0
    by metis
  have
    (linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ n+1 = card A)
          ⟶ (∃ a. a∈A ∧ above r a = {a})
  proof (induction n arbitrary: A r)
    case 0
    show ?case
    proof
      assume asm: linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ 0+1 = card A
      then obtain a where {a} = A
        using card-1-singletonE add.left-neutral
        by metis
      hence a ∈ A ∧ above r a = {a}
        using above-def asm connex-imp-refl above-refl
              lin-ord-imp-connex refl-on-domain
        by fastforce
      thus ∃ a. a∈A ∧ above r a = {a}
        by auto
    qed
  next
    case (Suc n)
    show ?case
    proof
      assume asm:
        linear-order-on A r ∧ finite A ∧ A ≠ {} ∧ Suc n+1 = card A
      then obtain B where B: card B = n+1 ∧ B ⊆ 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′ asm card-1-singletonE*
    card-Diff-subset finite-subset
  by *metis*
have $\exists\, b{\in}B.$ *above* (*limit* B r) $b = \{b\}$
  using *B One-nat-def Suc.IH add-diff-cancel-left′ asm*
    card-eq-0-iff diff-le-self finite-subset leD lessI
    limit-presv-lin-ord
  by *metis*
then obtain b **where** b: *above* (*limit* B r) $b = \{b\}$
  by *blast*
hence *b1*: $\{a.\ (b, a) \in$ *limit* $B\ r\} = \{b\}$
  using *above-def*
  by *metis*
hence *b2*: $b \preceq_r b$
  using *CollectD limit-presv-prefs2 singletonI*
  by (*metis* (*lifting*))
show $\exists\, a.\ a \in A \wedge$ *above* $r\ a = \{a\}$
proof *cases*
  assume
    asm1: $a \preceq_r b$
  have *f1*:
    $\forall\, A\ r\ a\ aa.$
      $\neg$ *refl-on* $A\ r \vee (a{::}'a,\ aa) \notin r \vee a \in A \wedge aa \in A$
    using *refl-on-domain*
    by *metis*
  have *f2*:
    $\forall\, A\ r.\ \neg$ *connex* ($A{::}'a$ *set*) $r \vee$ *refl-on* $A\ r$
    using *connex-imp-refl*
    by *metis*
  have *f3*:
    $\forall\, A\ r.\ \neg$ *linear-order-on* ($A{::}'a$ *set*) $r \vee$ *connex* $A\ r$
    by (*simp add*: *lin-ord-imp-connex*)
  hence *refl-on* $A\ r$
    using *f2 asm*
    by *metis*
  hence $a \in A \wedge b \in A$
    using *f1 asm1*
    by *simp*
  hence *f4*:
    $\forall\, a.\ a \notin A \vee b = a \vee (b, a) \in r \vee (a, b) \in r$
    using *asm order-on-defs*(*3*) *total-on-def*
    by *metis*
  have *f5*:
    $(b, b) \in$ *limit* $B\ r$
    using *above-def b mem-Collect-eq singletonI*
    by *metis*
  have *f6*:

$\forall\, a\ A\ Aa.\ (a{::}'a) \notin A - Aa \lor a \in A \land a \notin Aa$
by *simp*
have *ff1*:
$\{a.\ (b,\ a) \in \mathit{limit}\ B\ r\} = \{b\}$
using *above-def b*
by (*metis* (*no-types*))
have *ff2*:
$(b,\ b) \in \{(aa,\ a).\ (aa,\ a) \in r \land aa \in B \land a \in B\}$
using *f5*
by *simp*
moreover have *b-wins-B*:
$\forall\, x \in B.\ b \in \mathit{above}\ r\ x$
using *B above-def f4 ff1 ff2 CollectI*
Product-Type.Collect-case-prodD
by *fastforce*
moreover have $b \in \mathit{above}\ r\ a$
using *asm1 pref-imp-in-above*
by *metis*
ultimately have *b-wins*:
$\forall\, x \in A.\ b \in \mathit{above}\ r\ x$
using *Diff-iff a empty-iff insert-iff*
by (*metis* (*no-types*))
hence $\forall\, x \in A.\ x \in \mathit{above}\ r\ b \longrightarrow x = b$
using *CollectD above-def antisym-def asm lin-imp-antisym*
by *metis*
hence $\forall\, x \in A.\ x \in \mathit{above}\ r\ b \longleftrightarrow x = b$
using *b-wins*
by *blast*
moreover have *above-b-in-A*: $\mathit{above}\ r\ b \subseteq A$
using *above-def asm connex-imp-refl lin-ord-imp-connex*
mem-Collect-eq refl-on-domain subsetI
by *metis*
ultimately have $\mathit{above}\ r\ b = \{b\}$
using *above-def b*
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 *B DiffE a asm b limit-to-limits connex-def*
limited-dest singletonI subset-iff
lin-ord-imp-connex pref-imp-in-above
by *metis*
hence *b-smaller-a-0*: $(b,\ a) \in r$
by *simp*
have *g1*:
$\forall\, A\ r\ Aa.$

$\neg$ *linear-order-on* $(A::'a\ set)$ $r \vee$
$\neg$ $Aa \subseteq A \vee$
linear-order-on Aa (*limit* Aa r)
using *limit-presv-lin-ord*
by *metis*
have
$\{a.\ (b,\ a) \in$ *limit* $B\ r\} = \{b\}$
using *above-def* b
by *metis*
hence *g2*: $b \in B$
by *auto*
have *g3*:
partial-order-on B (*limit* B r) $\wedge$ *total-on* B (*limit* B r)
using *g1* B *asm order-on-defs*(*3*)
by *metis*
have
$\forall A\ r.$
total-on A r = $(\forall a.\ (a::'a) \notin A \vee$
$(\forall aa.\ (aa \notin A \vee a = aa) \vee (a,\ aa) \in r \vee (aa,\ a) \in r))$
using *total-on-def*
by *metis*
hence
$\forall a.\ a \notin B \vee$
$(\forall aa.\ aa \notin B \vee a = aa \vee$
$(a,\ aa) \in$ *limit* B $r \vee (aa,\ a) \in$ *limit* B r)
using *g3*
by *simp*
hence $\forall x \in B.\ b \in$ *above* r x
using *limit-presv-prefs2 pref-imp-in-above singletonD mem-Collect-eq*
asm b *b1* *b2* B *g2*
by (*metis* (*lifting*))
hence
$\forall x \in B.\ x \preceq_r b$
by (*simp add*: *above-def*)
hence *b-wins2*:
$\forall x \in B.\ (x,\ b) \in r$
by *simp*
have *trans* r
using *asm lin-imp-trans*
by *metis*
hence $\forall x \in B.\ (x,\ a) \in r$
using *transE b-smaller-a-0 b-wins2*
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 *asm lin-ord-imp-connex above-connex Diff-iff*
empty-iff insert-iff pref-imp-in-above
by *metis*

have $\forall x \in A.\ x \in$ *above r a* $\longleftrightarrow x = a$
using *antisym-def asm lin-imp-antisym nothing-above-a pref-imp-in-above CollectD above-def*
by *metis*
moreover have *above-a-in-A*: *above r a* $\subseteq A$
using *above-def asm connex-imp-refl lin-ord-imp-connex mem-Collect-eq refl-on-domain*
by *fastforce*
ultimately have *above r a* $= \{a\}$
using *above-def a*
by *auto*
thus *?thesis*
using *above-a-in-A*
by *blast*
qed
qed
qed
hence $\exists a.\ a{\in}A \wedge$ *above r a* $= \{a\}$
using *assms n*
by *blast*
thus *?thesis*
using *assms connex-def lin-ord-imp-connex pref-imp-in-above singletonD*
by *metis*
qed

lemma *above-one2*:
assumes
lin-ord: *linear-order-on A r* **and**
fin-not-emp: *finite A* $\wedge A \neq \{\}$ **and**
above1: *above r a* $= \{a\} \wedge$ *above r b* $= \{b\}$
shows $a = b$
proof $-$
have $a \preceq_r a \wedge b \preceq_r b$
using *above1 singletonI pref-imp-in-above*
by *metis*
also 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 *above1 connex-def limited-dest*
by *metis*
qed

lemma *above-presv-limit*:
  assumes *linear-order r*
  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* $\wedge$ $a \in A$ $\wedge$
    $(\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y)$

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*:
  assumes *order*: *linear-order-on A p*
  shows $\forall a \in A.$ *equiv-rel-except-a A p p a*
  by (*simp add*: *equiv-rel-except-a-def order*)

lemma *lifted-imp-equiv-rel-except-a*:
  assumes *lifted*: *lifted A r s a*
  shows *equiv-rel-except-a A r s a*
proof −
  from *lifted* **have**
    linear-order-on A r $\wedge$ *linear-order-on A s* $\wedge$ $a \in A$ $\wedge$
      $(\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y)$
    by (*simp add*: *lifted-def equiv-rel-except-a-def*)
  thus *?thesis*
    by (*simp add*: *equiv-rel-except-a-def*)
qed

lemma *lifted-mono*:
  assumes *lifted*: *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*
have *antisym r*
  using *equiv-rel-except-a-def lifted*
    lifted-imp-equiv-rel-except-a
    lin-imp-antisym
  by *metis*
hence *antisym-r*:
  $(\forall x\, y.\ (x, y) \in r \longrightarrow (y, x) \in r \longrightarrow x = y)$
  using *antisym-def*
  by *metis*
hence *imp-x-eq-a-0*:
  $[\![(x, a) \in r;\ (a, x) \in r]\!] \Longrightarrow x = a$
  by *simp*
have *lift-ex*: $\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
  using *lifted lifted-def*
  by *metis*
from *lift-ex* **obtain** $y :: 'a$ **where**
  f1: $y \in A - \{a\} \wedge a \preceq_r y \wedge y \preceq_s a$
  by *metis*
hence *f1-0*:
  $y \in A - \{a\} \wedge (a, y) \in r \wedge (y, a) \in s$
  by *simp*
have *f2*:
  equiv-rel-except-a $A\ r\ s\ a$
  using *lifted lifted-def*
  by *metis*
hence *f2-0*:
  $\forall\, x \in A - \{a\}.\ \forall\, y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
  using *equiv-rel-except-a-def*
  by *metis*
hence *f2-1*:
  $\forall\, x \in A - \{a\}.\ \forall\, y \in A - \{a\}.\ (x, y) \in r \longleftrightarrow (x, y) \in s$
  by *simp*
have *trans*: $\forall\, x\, y\, z.\ (x, y) \in r \longrightarrow (y, z) \in r \longrightarrow (x, z) \in r$
  using *f2 equiv-rel-except-a-def linear-order-on-def*
    partial-order-on-def preorder-on-def trans-def
  by *metis*
have *x-pref-y-0*: $(x, y) \in s$
  using *equiv-rel-except-a-def f1-0 f2 f2-1 insertE*
    insert-Diff x-in-A x-neq-a x-pref-a-0 trans
  by *metis*
have *a-pref-y-0*: $(a, y) \in s$
  using *a-pref-x-0 imp-x-eq-a-0 x-neq-a x-pref-a-0*
    equiv-rel-except-a-def f2 lin-imp-trans

transE x-pref-y-0
by *metis*
show *False*
using *a-pref-y-0 antisymD equiv-rel-except-a-def*
DiffD2 f1-0 f2 lin-imp-antisym singletonI
by *metis*
qed

lemma *lifted-mono2*:
assumes
lifted: *lifted* $A\ r\ s\ a$ **and**
x-pref-a: $x \preceq_r a$
shows $x \preceq_s a$
proof (*simp*)
have *x-pref-a-0*: $(x, a) \in r$
using *x-pref-a*
by *simp*
have *x-in-A*: $x \in A$
using *connex-imp-refl equiv-rel-except-a-def*
lifted lifted-def lin-ord-imp-connex
refl-on-domain x-pref-a-0
by *metis*
have $\forall x \in A - \{a\}. \forall y \in A - \{a\}. x \preceq_r y \longleftrightarrow x \preceq_s y$
using *lifted lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall x \in A - \{a\}. \forall y \in A - \{a\}. (x, y) \in r \longleftrightarrow (x, y) \in s$
by *simp*
have $\exists x \in A - \{a\}. a \preceq_r x \wedge x \preceq_s a$
using *lifted 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 equiv-rel-except-a-def refl-onD*
lifted lifted-def lin-ord-imp-connex
by *metis*
next
case *False*
thus *?thesis*
using *equiv-rel-except-a-def insertE insert-Diff*
lifted lifted-imp-equiv-rel-except-a x-in-A
x-pref-a-0 ex-lifted lin-imp-trans rest-eq
trans-def
by *metis*

qed
qed

lemma *lifted-above*:
assumes *lifted* $A\ r\ s\ a$
shows *above* $s\ a \subseteq$ *above* $r\ a$
unfolding *above-def*
proof (*safe*)
fix
$x :: 'a$
assume
a-pref-x: $(a, x) \in s$
have $\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
using *assms lifted-def*
by *metis*
hence *lifted-r*:
$\exists\, x \in A - \{a\}.\ (a, x) \in r \wedge (x, a) \in s$
by *simp*
have $\forall\, x \in A - \{a\}.\ \forall\, y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
using *assms lifted-def equiv-rel-except-a-def*
by *metis*
hence *rest-eq*:
$\forall\, x \in A - \{a\}.\ \forall\, y \in A - \{a\}.\ (x, y) \in r \longleftrightarrow (x, y) \in s$
by *simp*
have *trans-r*:
$\forall\, x\ y\ z.\ (x, y) \in r \longrightarrow (y, z) \in r \longrightarrow (x, z) \in r$
using *trans-def lifted-def lin-imp-trans*
equiv-rel-except-a-def assms
by *metis*
have *trans-s*:
$\forall\, x\ y\ z.\ (x, y) \in s \longrightarrow (y, z) \in s \longrightarrow (x, z) \in s$
using *trans-def lifted-def lin-imp-trans*
equiv-rel-except-a-def assms
by *metis*
have *refl-r*:
$(a, a) \in r$
using *assms connex-imp-refl equiv-rel-except-a-def*
lifted-def lin-ord-imp-connex refl-onD
by *metis*
have *x-in-A*: $x \in A$
using *a-pref-x assms connex-imp-refl equiv-rel-except-a-def*
lifted-def lin-ord-imp-connex refl-onD2
by *metis*
show $(a, x) \in r$
using *Diff-iff a-pref-x lifted-r rest-eq singletonD*
trans-r trans-s x-in-A refl-r
by (*metis* (*full-types*))
qed

lemma *lifted-above2*:
  assumes
    lifted A r s a **and**
    $x \in A-\{a\}$
  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 $\forall z \in A-\{a\}.\ x \preceq_r z \longleftrightarrow x \preceq_s z$
    using *assms lifted-def equiv-rel-except-a-def*
    by *metis*
  hence $\forall z \in A-\{a\}.\ (x, z) \in r \longleftrightarrow (x, z) \in s$
    by *simp*
  hence $\forall z \in A-\{a\}.\ z \in$ *above* r $x \longleftrightarrow z \in$ *above* s x
    by (*simp add*: *above-def*)
  hence $y \in$ *above* r $x \longleftrightarrow y \in$ *above* s x
    using *y-not-in-above-s assms*(*1*) *connex-def*
        equiv-rel-except-a-def lifted-def lifted-mono2
        limited-dest lin-ord-imp-connex member-remove
        pref-imp-in-above remove-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*:
  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**
    $\forall x \in S - \{a\}.\ \forall y \in S - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
    by (*simp add*: *lifted-def equiv-rel-except-a-def*)
  with *subset* **have** *temp*:
    $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
    by *auto*
  hence *eql-rs*:
      $\forall x \in A - \{a\}.\ \forall y \in A - \{a\}.$
      $(x, y) \in$ (*limit* A r) $\longleftrightarrow (x, y) \in$ (*limit* A s)
    using *DiffD1 limit-presv-prefs1 limit-presv-prefs2*
    by *auto*
  show *?thesis*
  proof *cases*

assume *a1*: $a \in A$
thus *?thesis*
proof *cases*

  assume *a1-1*: $\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a$
  from *lifted subset* **have**
    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*
  moreover from *a1 a1-1* **have** *keep-lift*:
    $\exists\, x \in A - \{a\}.$ (*let* q = *limit* A r *in* $a \preceq_q x$) $\wedge$
      (*let* u = *limit* A s *in* $x \preceq_u a$)
    using *DiffD1 limit-presv-prefs1*
    by *simp*
  ultimately show *?thesis*
    using *a1 temp*
    by (*simp add*: *lifted-def equiv-rel-except-a-def*)
next
  assume
    $\neg(\exists\, x \in A - \{a\}.\ a \preceq_r x \wedge x \preceq_s a)$
  hence *a1-2*:
    $\forall\, x \in A - \{a\}.\ \neg(a \preceq_r x \wedge x \preceq_s a)$
    by *auto*
  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 lifted-def equiv-rel-except-a-def*
        limit-presv-lin-ord lin-ord-imp-connex
    by *metis*
  moreover have *connex1*:
    $\forall\, A\ r.$ *connex* A r =
      (*limited* A r $\wedge$ ($\forall\, a.$ (a::$'a$) $\in A \longrightarrow$
        ($\forall\, aa.\ aa \in A \longrightarrow a \preceq_r aa \vee aa \preceq_r a$)))
    by (*simp add*: *Ball-def-raw connex-def*)
  hence *limit1*:
    limited A (*limit* A r) $\wedge$
      ($\forall\, a.\ a \notin A\ \vee$
        ($\forall\, aa.$
          $aa \notin A \vee (a, aa) \in$ *limit* A r $\vee$
            $(aa, a) \in$ *limit* A r))
    using *connex connex1*
    by *simp*
  have *limit2*:
    $\forall\, a\ aa\ A\ r.$ (a::$'a$, aa) $\notin$ *limit* A r $\vee$ $a \preceq_r aa$
    using *limit-presv-prefs2*
    by *metis*

have
limited A (*limit* A s) $\wedge$
($\forall\, a.\ a \notin A \vee$
($\forall\, aa.\ aa \notin A \vee$
(*let* q = *limit* A s *in* $a \preceq_q aa \vee aa \preceq_q a$)))
using *connex connex-def*
by *metis*
hence *connex2*:
limited A (*limit* A s) $\wedge$
($\forall\, a.\ a \notin A \vee$
($\forall\, aa.\ aa \notin A \vee$
((a, aa) $\in$ *limit* A s $\vee$ (aa, 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 limit1 limit-presv-prefs2 a1*
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) $\in$ (*limit* A r) $\longleftrightarrow$ (a, x) $\in$ (*limit* A s)
using *DiffD1 limit2 limit1 connex2 a1 a1-2 not-worse*
by *metis*
hence
$\forall\, x \in A - \{a\}$.
(*let* q = *limit* A r *in* $a \preceq_q x$) $\longleftrightarrow$ (*let* q = *limit* A s *in* $a \preceq_q x$)
by *simp*
moreover have
$\forall\, x \in A - \{a\}$. ($x$, a) $\in$ (*limit* A r) $\longleftrightarrow$ (x, a) $\in$ (*limit* A s)
using *a1 a1-2 not-worse DiffD1 limit-presv-prefs2 connex2 limit1*
by *metis*
moreover have
(a, a) $\in$ (*limit* A r) $\wedge$ (a, a) $\in$ (*limit* A s)
using *a1 connex connex-imp-refl refl-onD*
by *metis*
moreover have
limited A (*limit* A r) $\wedge$ *limited* A (*limit* A s)
using *limit-to-limits*
by *metis*
ultimately have
$\forall\, x\ y$. (x, y) $\in$ *limit* A r $\longleftrightarrow$ (x, y) $\in$ *limit* A s
using *eql-rs*
by *auto*
thus *?thesis*
by *simp*
qed
next
assume *a2*: $a \notin A$

with *eql-rs* **have**
$\forall x \in A.\ \forall y \in A.\ (x,\ y) \in (\mathit{limit}\ A\ r) \longleftrightarrow (x,\ y) \in (\mathit{limit}\ A\ s)$
by *simp*
thus *?thesis*
using *limit-to-limits limited-dest subrelI subset-antisym*
by *auto*
qed
qed

lemma *negl-diff-imp-eq-limit*:
assumes
change: *equiv-rel-except-a* $S\ r\ s\ a$ **and**
subset: $A \subseteq S$ **and**
notInA: $a \notin A$
shows $\mathit{limit}\ A\ r = \mathit{limit}\ A\ s$
proof −
have $A \subseteq S-\{a\}$
by (*simp add*: *notInA subset subset-Diff-insert*)
hence $\forall x \in A.\ \forall y \in A.\ x \preceq_r y \longleftrightarrow x \preceq_s y$
by (*meson change equiv-rel-except-a-def in-mono*)
thus *?thesis*
by *auto*
qed

theorem *lifted-above-winner*:
assumes
lifted-a: *lifted* $A\ r\ s\ a$ **and**
above-x: $\mathit{above}\ r\ x = \{x\}$ **and**
fin-A: *finite* A
shows $\mathit{above}\ s\ x = \{x\} \vee \mathit{above}\ s\ a = \{a\}$
proof *cases*
assume $x = a$
thus *?thesis*
using *above-subset-geq-one lifted-a above-x*
lifted-above lifted-def equiv-rel-except-a-def
by *metis*
next
assume *asm1*: $x \neq a$
thus *?thesis*
proof *cases*
assume $\mathit{above}\ s\ x = \{x\}$
thus *?thesis*
by *simp*
next
assume *asm2*: $\mathit{above}\ s\ x \neq \{x\}$
have $\forall y \in A.\ y \preceq_r x$
proof −
fix $aa :: 'a$
have *imp-a*: $x \preceq_r aa \longrightarrow aa \notin A \vee aa \preceq_r x$

using *singletonD pref-imp-in-above above-x*
by *metis*
also have *f1*:
$\forall A\ r.$
$(connex\ A\ r \lor$
$(\exists a.\ (\exists aa.\ \neg\ (aa::'a) \preceq_r a \land \neg\ a \preceq_r aa \land aa \in A) \land a \in A) \lor$
$\neg\ limited\ A\ r) \land$
$((\forall a.\ (\forall aa.\ aa \preceq_r a \lor a \preceq_r aa \lor aa \notin A) \lor a \notin A) \land limited\ A\ r \lor$
$\neg\ connex\ A\ r)$
using *connex-def*
by *metis*
moreover have *eq-exc-a*:
equiv-rel-except-a A r s a
using *lifted-def lifted-a*
by *metis*
ultimately have $aa \notin A \lor aa \preceq_r x$
using *pref-imp-in-above above-x equiv-rel-except-a-def*
lin-ord-imp-connex limited-dest insertCI
by *metis*
thus *?thesis*
using *f1 eq-exc-a above-one above-one2 above-x fin-A*
equiv-rel-except-a-def insert-not-empty pref-imp-in-above
lin-ord-imp-connex mk-disjoint-insert insertE
by *metis*
qed
moreover have *equiv-rel-except-a A r s a*
using *lifted-a lifted-def*
by *metis*
moreover have $x \in A-\{a\}$
using *above-one above-one2 asm1 assms calculation*
equiv-rel-except-a-def insert-not-empty
member-remove remove-def insert-absorb
by *metis*
ultimately have $\forall\, y \in A-\{a\}.\ y \preceq_s x$
using *DiffD1 lifted-a equiv-rel-except-a-def*
by *metis*
hence *not-others*: $\forall\, y \in A-\{a\}.\ above\ s\ y \neq \{y\}$
using *asm2 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 lifted-def*
equiv-rel-except-a-def above-one singleton-iff
by *metis*
thus *?thesis*
by *simp*
qed
qed

theorem *lifted-above-winner2*:

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-winner3*:
assumes
lifted-a: *lifted A r s a* **and**
above-x: *above s x* = {*x*} **and**
fin-A: *finite A* **and**
x-not-a: $x \neq a$
shows *above r x* = {*x*}
proof (*rule ccontr*)
assume *asm*: *above r x* $\neq$ {*x*}
then obtain *y* **where** *y*: *above r y* = {*y*}
using *lifted-a fin-A insert-Diff insert-not-empty*
lifted-def equiv-rel-except-a-def above-one
by *metis*
hence *above s y* = {*y*} $\vee$ *above s a* = {*a*}
using *lifted-a fin-A lifted-above-winner*
by *metis*
moreover have $\forall$ *b*. *above s b* = {*b*} $\longrightarrow$ *b* = *x*
using *all-not-in-conv lifted-a above-x lifted-def*
fin-A equiv-rel-except-a-def above-one2
by *metis*
ultimately have *y* = *x*
using *x-not-a*
by *presburger*
moreover have $y \neq x$
using *asm 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

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

1.2.2 Auxiliary Functions

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-equals-partition $A\ (e, r, d) = (e \cup r \cup d = A)$

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

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

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

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

1.2.3 Auxiliary Lemmata

lemma *result-imp-rej*:
  assumes *well-formed* $A\ (e, r, d)$
  shows $A - (e \cup d) = r$
proof (*safe*)
  fix
    $x :: 'a$
  assume
    x-in-A: $x \in A$ **and**
    x-not-rej: $x \notin r$ **and**
    x-not-def: $x \notin d$
  from *assms* **have**

```
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧
    (r ∩ d = {}) ∧ (e ∪ r ∪ d = A)
    by simp
  thus x ∈ e
    using x-in-A x-not-rej x-not-def
    by auto
next
  fix
    x :: 'a
  assume
    x-rej:   x ∈ r
  from assms have
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧
    (r ∩ d = {}) ∧ (e ∪ r ∪ d = A)
    by simp
  thus x ∈ A
    using x-rej
    by auto
next
  fix
    x :: 'a
  assume
    x-rej:  x ∈ r and
    x-elec: x ∈ e
  from assms have
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧
    (r ∩ d = {}) ∧ (e ∪ r ∪ d = A)
    by simp
  thus False
    using x-rej x-elec
    by auto
next
  fix
    x :: 'a
  assume
    x-rej: x ∈ r and
    x-def: x ∈ d
  from assms have
    (e ∩ r = {}) ∧ (e ∩ d = {}) ∧
    (r ∩ d = {}) ∧ (e ∪ r ∪ d = A)
    by simp
  thus False
    using x-rej x-def
    by auto
qed

lemma result-count:
  assumes
    well-formed A (e, r, d) and
```

finite A
shows *card A = card e + card r + card d*
proof −
from *assms(1)* **have** *disj*:
$(e \cap r = \{\}) \wedge (e \cap d = \{\}) \wedge (r \cap d = \{\})$
by *simp*
from *assms(1)* **have** *set-partit*:
$e \cup r \cup d = A$
by *simp*
show *?thesis*
using *assms disj set-partit Int-Un-distrib2 finite-Un card-Un-disjoint sup-bot.right-neutral*
by *metis*
qed

lemma *defer-subset*:
assumes *well-formed A result*
shows *defer-r result* $\subseteq$ *A*
proof (*safe*)
fix *x* :: *'a*
assume *assm0*: $x \in$ *defer-r result*
obtain
AA :: *'a Result* ⇒ *'a set* ⇒ *'a set* **and**
pp :: *'a Result* ⇒ *'a set* ⇒ *'a Result* **where**
assm1: *A = AA result A* ∧ *result = pp result A* ∧
disjoint3 (*pp result A*) ∧
set-equals-partition (*AA result A*) (*pp result A*)
using *assms*
by *simp*
hence
∀ *p A*. ∃ *Aa Ab Ac*.
(¬ *set-equals-partition* (*A::'a set*) *p* ∨ (*Aa*, *Ab*, *Ac*) = *p*) ∧
(¬ *set-equals-partition A p* ∨ *Aa* ∪ *Ab* ∪ *Ac* = *A*)
by *auto*
thus $x \in A$
using *UnCI assm0 assm1 snd-conv*
by *metis*
qed

lemma *elect-subset*:
assumes *well-formed A result*
shows *elect-r result* $\subseteq$ *A*
proof (*safe*)
fix *x* :: *'a*
assume *assm0*: $x \in$ *elect-r result*
obtain
AA :: *'a Result* ⇒ *'a set* ⇒ *'a set* **and**
pp :: *'a Result* ⇒ *'a set* ⇒ *'a Result* **where**
assm1: *A = AA result A* ∧ *result = pp result A* ∧

```
        disjoint3 (pp result A) ∧
        set-equals-partition (AA result A) (pp result A)
    using assms
    by simp
  hence
    ∀ p A. ∃ Aa Ab Ac.
      (¬ set-equals-partition (A::'a set) p ∨ (Aa, Ab, Ac) = p) ∧
        (¬ set-equals-partition A p ∨ Aa ∪ Ab ∪ Ac = A)
    by auto
  thus x ∈ A
    using UnCI assm0 assm1 assms fst-conv
    by metis
qed

lemma reject-subset:
  assumes well-formed A result
  shows reject-r result ⊆ A
proof (safe)
  fix x :: 'a
  assume assm0: x ∈ reject-r result
  obtain
    AA :: 'a Result ⇒ 'a set ⇒ 'a set and
    pp :: 'a Result ⇒ 'a set ⇒ 'a Result where
    assm1: A = AA result A ∧ result = pp result A ∧
        disjoint3 (pp result A) ∧
        set-equals-partition (AA result A) (pp result A)
    using assms
    by simp
  hence
    ∀ p A. ∃ Aa Ab Ac.
      (¬ set-equals-partition (A::'a set) p ∨ (Aa, Ab, Ac) = p) ∧
        (¬ set-equals-partition A p ∨ Aa ∪ Ab ∪ Ac = A)
    by auto
  thus x ∈ A
    using UnCI assms assm0 assm1 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

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

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* : *profile A p* ≡ (∀ *b* ∈ (*set p*). *linear-order-on A b*)
  by (*simp add*: *all-set-conv-all-nth profile-def*)

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

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*]: *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) ∧ (above ((p@[a])!i) x = {x})} =
      {i. i < length p ∧ (above (p!i) x = {x})} ∪
        {i. i = length p ∧ (above ([a]!0) x = {x})}
  proof (safe, simp-all)
    fix
      xa :: nat and
      xaa :: 'a
    assume
      xa < Suc (length p) and
      above ((p@[a])!xa) x = {x} and
      xa ≠ length p and
      xaa ∈ above (p!xa) x
    thus xaa = x
      using less-antisym p-pos-in-ps singletonD
      by metis
  next
    fix
      xa :: nat
    assume
      xa < Suc (length p) and
      above ((p@[a])!xa) x = {x} and
      xa ≠ length p
    thus x ∈ above (p!xa) x
      using less-antisym insertI1 p-pos-in-ps
      by metis
  next
    fix
```

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

next
  fix
    xa :: *nat* **and**
    xb :: *'a*
  assume
    $xa < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!xa)\ x = \{x\}$ **and**
    $x \notin above\ a\ x$ **and**
    $xb \in above\ (p!xa)\ x$
  thus $xb = x$
    using *insertI1 less-antisym nth-append-length*
        p-pos-in-ps singletonD
    by *metis*
next
  fix
    xa :: *nat*
  assume
    $xa < Suc\ (length\ p)$ **and**
    $above\ ((p@[a])!xa)\ x = \{x\}$ **and**
    $x \notin above\ a\ x$
  thus $x \in above\ (p!xa)\ x$
    using *insertI1 less-antisym nth-append-length p-pos-in-ps*
    by *metis*
next
  fix
    xa :: *nat* **and**
    xaa :: *'a*
  assume
    $xa < length\ p$ **and**
    $above\ (p!xa)\ x = \{x\}$ **and**
    $xaa \in above\ ((p@[a])!xa)\ x$
  thus $xaa = x$
    by (*simp add*: *nth-append*)
next
  fix
    xa :: *nat*
  assume
    $xa < length\ p$ **and**
    $above\ (p!xa)\ x = \{x\}$
  thus $x \in above\ ((p@[a])!xa)\ x$
    by (*simp add*: *nth-append*)
qed
have *f1*:
  $finite\ \{n.\ n < length\ p \wedge (above\ (p!n)\ x = \{x\})\}$
  by *simp*
have *f2*:
  $finite\ \{n.\ n = length\ p \wedge (above\ ([a]!0)\ x = \{x\})\}$
  by *simp*
have

card ({*i*. *i* < *length p* ∧ (*above* (*p*!*i*) *x* = {*x*})} ∪
  {*i*. *i* = *length p* ∧ (*above* ([*a*]!*0*) *x* = {*x*})}) =
    card {*i*. *i* < *length p* ∧ (*above* (*p*!*i*) *x* = {*x*})} +
      card {*i*. *i* = *length p* ∧ (*above* ([*a*]!*0*) *x* = {*x*})}
using *f1 f2 card-Un-disjoint*
by *blast*
thus
card {*i*. *i* < *Suc* (*length p*) ∧ (*above* ((*p*@[*a*])!*i*) *x* = {*x*})} =
  card {*i*. *i* < *length p* ∧ (*above* (*p*!*i*) *x* = {*x*})} +
    card {*i*. *i* = *0* ∧ (*above* ([*a*]!*i*) *x* = {*x*})}
using *set-prof-eq shift-idx-a*
by *auto*
qed
have *pref-count-code-induct*:
win-count-code (*p*@[*a*]) *x* =
  win-count-code p x + *win-count-code* [*a*] *x*
proof (*induction p*, *simp*)
fix
aa :: ′*a Preference-Relation* **and**
p :: ′*a Profile*
assume
win-count-code (*p*@[*a*]) *x* =
  win-count-code p x + *win-count-code* [*a*] *x*
thus
win-count-code ((*aa*#*p*)@[*a*]) *x* =
  win-count-code (*aa*#*p*) *x* + *win-count-code* [*a*] *x*
by *simp*
qed
show *win-count* (*p*@[*a*]) *x* = *win-count-code* (*p*@[*a*]) *x*
using *pref-count-code-induct pref-count-induct*
  base-case one-ballot-equiv
by *presburger*
qed

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

fun *prefer-count-code* :: ′*a Profile* ⇒ ′*a* ⇒ ′*a* ⇒ *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*]: *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:
  $prefer\text{-}count\ p\ x\ y = prefer\text{-}count\text{-}code\ p\ x\ y$
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
  $prefer\text{-}count\ [a]\ x\ y =$
    $(let\ q = [a]\ in$
      $card\ \{i{::}nat.\ i < length\ q\ \wedge$
           $(let\ r = (q!i)\ in\ (y \preceq_r x))\})$
  by *simp*
hence *one-ballot-equiv*:
  $prefer\text{-}count\ [a]\ x\ y = prefer\text{-}count\text{-}code\ [a]\ x\ y$
  using *size-one*
  by (*simp add*: *nth-Cons′*)
have *pref-count-induct*:
  $prefer\text{-}count\ (p@[a])\ x\ y =$
    $prefer\text{-}count\ p\ x\ y + prefer\text{-}count\ [a]\ x\ y$
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 $xa < \textit{length } p$
using *less-antisym nth-append-length*
by *metis*
next
fix
$xa :: \textit{nat}$
assume
$xa < \textit{Suc } (\textit{length } p)$ **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
$xa :: \textit{nat}$
assume
$xa < \textit{length } p$ **and**
$(y, x) \in p!xa$
thus $(y, x) \in (p@[a])!xa$
using *less-antisym p-pos-in-ps*
by *metis*
qed
have *f1*:
$\textit{finite } \{n.\ n < \textit{length } p \wedge (y, x) \in p!n\}$
by *simp*
have *f2*:
$\textit{finite } \{n.\ n = \textit{length } p \wedge (y, x) \in [a]!\mathit{0}\}$
by *simp*
have
$\textit{card } (\{i.\ i < \textit{length } p \wedge (y, x) \in p!i\} \cup$
$\{i.\ i = \textit{length } p \wedge (y, x) \in [a]!\mathit{0}\}) =$
$\textit{card } \{i.\ i < \textit{length } p \wedge (y, x) \in p!i\} +$
$\textit{card } \{i.\ i = \textit{length } p \wedge (y, x) \in [a]!\mathit{0}\}$
using *f1 f2 card-Un-disjoint*
by *blast*
thus
$\textit{card } \{i.\ i < \textit{Suc } (\textit{length } p) \wedge (y, x) \in (p@[a])!i\} =$
$\textit{card } \{i.\ i < \textit{length } p \wedge (y, x) \in p!i\} +$
$\textit{card } \{i.\ i = \mathit{0} \wedge (y, x) \in [a]!i\}$
using *set-prof-eq shift-idx-a*
by *auto*
qed
have *pref-count-code-induct*:
$\textit{prefer-count-code } (p@[a])\ x\ y =$
$\textit{prefer-count-code } p\ x\ y + \textit{prefer-count-code } [a]\ x\ y$
proof (*simp*, *safe*)

```
    assume
      assm: (y, x) ∈ a
    show
      prefer-count-code (p@[a]) x y = Suc (prefer-count-code p x y)
    proof (induction p, simp-all)
      show (y, x) ∈ a
        using assm
        by simp
    qed
  next
    assume
      assm: (y, x) ∉ a
    show
      prefer-count-code (p@[a]) x y = prefer-count-code p x y
    proof (induction p, simp-all, safe)
      assume
        (y, x) ∈ a
      thus False
        using assm
        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: { f x | x . x ∈ S } = f ‘ S
  by auto

lemma pref-count-set-compr: {prefer-count p x y | y . y ∈ A−{x}} =
          (prefer-count p x) ‘ (A−{x})
  by auto

lemma pref-count:
  assumes prof: profile A p
  assumes x-in-A: x ∈ A
  assumes y-in-A: y ∈ A
  assumes neq: x ≠ y
  shows prefer-count p x y = (length p) − (prefer-count p y x)
proof −
  have 00: card {i::nat. i < length p} = length p
    by simp
  have 10:
    {i::nat. i < length p ∧ (let r = (p!i) in (y ⪯r x))} =
        {i::nat. i < length p} −
          {i::nat. i < length p ∧ ¬ (let r = (p!i) in (y ⪯r x))}
    by auto
```

have *0*: $\forall\ i{::}nat\ .\ i < length\ p \longrightarrow linear\text{-}order\text{-}on\ A\ (p!i)$
using *prof profile-def*
by *metis*
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 *connex-def x-in-A y-in-A*
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**
$\forall\ i{::}nat\ .\ i < length\ p \longrightarrow$
$\neg\ (let\ r = (p!i)\ in\ (y \preceq_r x)) = (let\ r = (p!i)\ in\ (x \preceq_r y))$
by *metis*
hence *2*:
$\{i{::}nat.\ i < length\ p \wedge \neg\ (let\ r = (p!i)\ in\ (y \preceq_r x))\} =$
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}$
by *metis*
hence *20*:
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\} =$
$\{i{::}nat.\ i < length\ p\}\ -$
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}$
using *10 2*
by *simp*
have
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\} \subseteq$
$\{i{::}nat.\ i < length\ p\}$
by (*simp add*: *Collect-mono*)
hence *30*:
$card\ (\{i{::}nat.\ i < length\ p\}\ -$
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\}) =$
$(card\ \{i{::}nat.\ i < length\ p\})\ -$
$card(\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$
by (*simp add*: *card-Diff-subset*)
have $prefer\text{-}count\ p\ x\ y =$
$card\ \{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (y \preceq_r x))\}$
by *simp*
also have
$... = card(\{i{::}nat.\ i < length\ p\}\ -$
$\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$

using *20*
by *simp*
also have
... $= (card\ \{i{::}nat.\ i < length\ p\}) -$
$card(\{i{::}nat.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r y))\})$
using *30*
by *metis*
also have
... $= length\ p - (prefer\text{-}count\ p\ y\ x)$
by *simp*
finally show *?thesis*
by (*simp add*: *20 30*)
qed

lemma *pref-count-sym*:
assumes *p1*: $prefer\text{-}count\ p\ a\ x \geq prefer\text{-}count\ p\ x\ b$
assumes *prof*: *profile A p*
assumes *a-in-A*: $a \in A$
assumes *b-in-A*: $b \in A$
assumes *x-in-A*: $x \in A$
assumes *neq1*: $a \neq x$
assumes *neq2*: $x \neq b$
shows $prefer\text{-}count\ p\ b\ x \geq prefer\text{-}count\ p\ x\ a$
proof −
from *prof a-in-A x-in-A neq1* **have** *0*:
$prefer\text{-}count\ p\ a\ x = (length\ p) - (prefer\text{-}count\ p\ x\ a)$
using *pref-count*
by *metis*
moreover from *prof x-in-A b-in-A neq2* **have** *1*:
$prefer\text{-}count\ p\ x\ b = (length\ p) - (prefer\text{-}count\ p\ b\ x)$
using *pref-count*
by (*metis* (*mono-tags*, *lifting*))
hence *2*: $(length\ p) - (prefer\text{-}count\ p\ x\ a) \geq$
$(length\ p) - (prefer\text{-}count\ p\ b\ x)$
using *calculation p1*
by *auto*
hence *3*: $(prefer\text{-}count\ p\ x\ a) - (length\ p) \leq$
$(prefer\text{-}count\ p\ b\ x) - (length\ p)$
using *a-in-A diff-is-0-eq diff-le-self neq1*
pref-count prof x-in-A
by (*metis* (*no-types*))
hence $(prefer\text{-}count\ p\ x\ a) \leq (prefer\text{-}count\ p\ b\ x)$
using *1 3 calculation p1*
by *linarith*
thus *?thesis*
by *linarith*
qed

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

assumes $p = []$
shows $\forall\ x\ y.$ *prefer-count* $p\ x\ y = 0$
using *assms*
by *simp*

lemma *empty-prof-imp-zero-pref-count-code*:
assumes $p = []$
shows $\forall\ x\ y.$ *prefer-count-code* $p\ x\ y = 0$
using *assms*
by *simp*

lemma *pref-count-code-incr*:
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*:
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* :: $'a \Rightarrow 'a$ *Profile* $\Rightarrow 'a \Rightarrow$ *bool* **where**
wins $x\ p\ y =$
(*prefer-count* $p\ x\ y >$ *prefer-count* $p\ y\ x$)

lemma *wins-antisym*:
assumes *wins* $a\ p\ b$
shows $\neg$ *wins* $b\ p\ a$
using *assms*
by *simp*

lemma *wins-irreflex*: $\neg$ *wins* $w\ p\ w$
using *wins-antisym*
by *metis*

1.3.3 Condorcet Winner

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

lemma *cond-winner-unique*:

assumes *winner-c*: *condorcet-winner A p c* **and**
winner-w: *condorcet-winner A p w*
shows $w = c$
proof (*rule ccontr*)
assume
assumption: $w \neq c$
from *winner-w*
have *wins w p c*
using *assumption insert-Diff insert-iff winner-c*
by *simp*
hence ¬ *wins c p w*
by (*simp add*: *wins-antisym*)
moreover from *winner-c*
have
c-wins-against-w: *wins c p w*
using *Diff-iff assumption*
singletonD winner-w
by *simp*
ultimately show *False*
by *simp*
qed

lemma *cond-winner-unique2*:
assumes *winner*: *condorcet-winner A p w* **and**
not-w: $x \neq w$ **and**
in-A: $x \in A$
shows ¬ *condorcet-winner A p x*
using *not-w cond-winner-unique winner*
by *metis*

lemma *cond-winner-unique3*:
assumes *condorcet-winner A p w*
shows $\{a \in A.\ \mathit{condorcet\text{-}winner}\ A\ p\ a\} = \{w\}$
proof (*safe*, *simp-all*, *safe*)
fix
$x :: 'a$
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
x-in-A: $x \in A$ **and**
x-wins:
$\forall xa \in A - \{x\}.$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (x, xa) \in p!i\} <$
$\mathit{card}\ \{i.\ i < \mathit{length}\ p \wedge (xa, x) \in p!i\}$
from *assms* **have** *assm*:
$\mathit{finite\text{-}profile}\ A\ p \wedge\ w \in A \wedge$
$(\forall x \in A - \{w\}.$
$\mathit{card}\ \{i{::}\mathit{nat}.\ i < \mathit{length}\ p \wedge (w, x) \in p!i\} <$
$\mathit{card}\ \{i{::}\mathit{nat}.\ i < \mathit{length}\ p \wedge (x, w) \in p!i\})$

    by *simp*
  hence
    $(\forall x \in A - \{w\}.$
      $card\ \{i{::}nat.\ i < length\ p \wedge (w, x) \in p!i\} <$
        $card\ \{i{::}nat.\ i < length\ p \wedge (x, w) \in p!i\})$
    by *simp*
  hence *w-beats-x*:
    $x \neq w \Longrightarrow$
      $card\ \{i{::}nat.\ i < length\ p \wedge (w, x) \in p!i\} <$
        $card\ \{i{::}nat.\ i < length\ p \wedge (x, w) \in p!i\}$
    using *x-in-A*
    by *simp*
  also from *assm* **have**
    finite-profile A p
    by *simp*
  moreover from *assm* **have**
    $w \in A$
    by *simp*
  hence *x-beats-w*:
    $x \neq w \Longrightarrow$
      $card\ \{i.\ i < length\ p \wedge (x, w) \in p!i\} <$
        $card\ \{i.\ i < length\ p \wedge (w, x) \in p!i\}$
    using *x-wins*
    by *simp*
  from *w-beats-x x-beats-w* **show**
    $x = w$
    by *linarith*
next
  fix
    $x :: 'a$
  from *assms* **show** $w \in A$
    by *simp*
next
  fix
    $x :: 'a$
  from *assms* **show** *finite A*
    by *simp*
next
  fix
    $x :: 'a$
  from *assms* **show** *profile A p*
    by *simp*
next
  fix
    $x :: 'a$
  from *assms* **show** $w \in A$
    by *simp*
next
  fix

x :: $'a$ **and**
xa :: $'a$
assume
xa-in-A: $xa \in A$ **and**
w-wins:
$\neg$ *card* $\{i.\ i < length\ p \land (w,\ xa) \in p!i\} <$
card $\{i.\ i < length\ p \land (xa,\ w) \in p!i\}$
from *assms* **have**
finite-profile $A\ p \land\ w \in A \land$
$(\forall\, x \in A - \{w\}$.
card $\{i{::}nat.\ i < length\ p \land (w,\ x) \in p!i\} <$
card $\{i{::}nat.\ i < length\ p \land (x,\ w) \in p!i\})$
by *simp*
thus $xa = w$
using *xa-in-A w-wins insert-Diff insert-iff*
by (*metis* (*no-types*, *lifting*))
qed

1.3.4 Limited Profile

fun *limit-profile* :: $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Profile* **where**
limit-profile $A\ p$ = *map* (*limit* A) p

lemma *limit-prof-trans*:
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*:
assumes
profile: *finite-profile* $S\ p$ **and**
subset: $A \subseteq S$
shows *finite-profile* A (*limit-profile* $A\ p$)
proof (*simp*)
from *profile*
show *finite-profile* A (*map* (*limit* A) p)
using *length-map limit-presv-lin-ord nth-map*
profile-def subset infinite-super
by *metis*
qed

lemma *limit-prof-presv-size*:
assumes *f-prof*: *finite-profile* $S\ p$ **and**
subset: $A \subseteq S$
shows *length* p = *length* (*limit-profile* $A\ p$)

by *simp*

1.3.5 Lifting Property

definition *equiv-prof-except-a* :: $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$
$'a \Rightarrow$ *bool* **where**
equiv-prof-except-a $A\ p\ q\ a \equiv$
finite-profile $A\ p \wedge$ *finite-profile* $A\ q \wedge$
$a \in A \wedge$ *length* $p =$ *length* $q \wedge$
$(\forall\, i{::}nat.$
$i <$ *length* $p \longrightarrow$
equiv-rel-except-a $A\ (p!i)\ (q!i)\ a)$

definition *lifted* :: $'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a \Rightarrow$ *bool* **where**
lifted $A\ p\ q\ a \equiv$
finite-profile $A\ p \wedge$ *finite-profile* $A\ q \wedge$
$a \in A \wedge$ *length* $p =$ *length* $q \wedge$
$(\forall\, i{::}nat.$
$(i <$ *length* $p \wedge \neg$*Preference-Relation.lifted* $A\ (p!i)\ (q!i)\ a) \longrightarrow$
$(p!i) = (q!i)) \wedge$
$(\exists\, i{::}nat.\ i <$ *length* $p \wedge$ *Preference-Relation.lifted* $A\ (p!i)\ (q!i)\ a)$

lemma *lifted-imp-equiv-prof-except-a*:
assumes *lifted*: *lifted* $A\ p\ q\ a$
shows *equiv-prof-except-a* $A\ p\ q\ a$
proof −
have
$\forall\, i{::}nat.\ i <$ *length* $p \longrightarrow$
equiv-rel-except-a $A\ (p!i)\ (q!i)\ a$
proof
fix i :: *nat*
show
$i <$ *length* $p \longrightarrow$
equiv-rel-except-a $A\ (p!i)\ (q!i)\ a$
proof
assume *i-ok*: $i <$ *length* p
show *equiv-rel-except-a* $A\ (p!i)\ (q!i)\ a$
using *lifted-def i-ok lifted profile-def trivial-equiv-rel*
lifted-imp-equiv-rel-except-a
by *metis*
qed
qed
thus *?thesis*
using *lifted-def lifted equiv-prof-except-a-def*
by *metis*
qed

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

```
  assumes
    change: equiv-prof-except-a S p q a and
    subset: A ⊆ S and
    notInA: a ∉ A
  shows limit-profile A p = limit-profile A q
proof −
  have
    ∀ i::nat. i < length p ⟶
      equiv-rel-except-a S (p!i) (q!i) a
    using change equiv-prof-except-a-def
    by metis
  hence ∀ i::nat. i < length p ⟶ limit A (p!i) = limit A (q!i)
    using notInA negl-diff-imp-eq-limit subset
    by metis
  hence 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))
  thus ?thesis
    by simp
qed

lemma limit-prof-eq-or-lifted:
  assumes
    lifted: lifted S p q a and
    subset: A ⊆ S
  shows
    limit-profile A p = limit-profile A q ∨
        lifted A (limit-profile A p) (limit-profile A q) a
proof cases
  assume inA: a ∈ A
  have
    ∀ i::nat. i < length p ⟶
        (Preference-Relation.lifted S (p!i) (q!i) a ∨ (p!i) = (q!i))
    using lifted-def lifted
    by metis
  hence one:
    ∀ i::nat. i < length p ⟶
         (Preference-Relation.lifted A (limit A (p!i)) (limit A (q!i)) a ∨
           (limit A (p!i)) = (limit A (q!i)))
    using limit-lifted-imp-eq-or-lifted subset
    by metis
  thus ?thesis
  proof cases
    assume ∀ i::nat. i < length p ⟶ (limit A (p!i)) = (limit A (q!i))
    thus ?thesis
      using lifted-def length-map lifted
            limit-profile.simps nth-equalityI nth-map
      by (metis (mono-tags, lifting))
```

next
  assume *assm*:
    $\neg(\forall\, i{::}nat.\ i < length\ p \longrightarrow (limit\ A\ (p!i)) = (limit\ A\ (q!i)))$
  let $?p = limit\text{-}profile\ A\ p$
  let $?q = limit\text{-}profile\ A\ q$
  have $profile\ A\ ?p \wedge profile\ A\ ?q$
    using *lifted-def lifted limit-profile-sound subset*
    by *metis*
  moreover have $length\ ?p = length\ ?q$
    using *lifted-def lifted*
    by *fastforce*
  moreover have
    $\exists\, i{::}nat.\ i < length\ ?p \wedge Preference\text{-}Relation.lifted\ A\ (?p!i)\ (?q!i)\ a$
    using *assm lifted-def length-map lifted*
        limit-profile.simps nth-map one
    by (*metis* (*no-types*, *lifting*))
  moreover have
    $\forall\, i{::}nat.$
      $(i < length\ ?p \wedge \neg Preference\text{-}Relation.lifted\ A\ (?p!i)\ (?q!i)\ a) \longrightarrow$
        $(?p!i) = (?q!i)$
    using *lifted-def length-map lifted*
        limit-profile.simps nth-map one
    by *metis*
  ultimately have $lifted\ A\ ?p\ ?q\ a$
    using *lifted-def inA lifted rev-finite-subset subset*
    by (*metis* (*no-types*, *lifting*))
  thus *?thesis*
    by *simp*
qed
next
  assume $a \notin A$
  thus *?thesis*
    using *lifted negl-diff-imp-eq-limit-prof subset*
        lifted-imp-equiv-prof-except-a
    by *metis*
qed

end

Chapter 2

Component Types

2.1 Electoral Module

theory *Electoral-Module*
  imports *Social-Choice-Types/Preference-Relation*
    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

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

2.1.2 Auxiliary Definitions

definition *electoral-module* :: $'a$ *Electoral-Module* $\Rightarrow$ *bool* **where**
  electoral-module $m \equiv \forall\, A\ p.$ *finite-profile* $A\ p \longrightarrow$ *well-formed* $A\ (m\ A\ p)$

lemma *electoral-modI*:
$((\bigwedge A\ p.\ [\![\ \textit{finite-profile}\ A\ p\ ]\!] \Longrightarrow \textit{well-formed}\ A\ (m\ A\ p)) \Longrightarrow$
$\quad \textit{electoral-module}\ m)$
unfolding *electoral-module-def*
by *auto*

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

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

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

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

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

definition *eliminates* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
$\textit{eliminates}\ n\ m \equiv$
$\quad \textit{electoral-module}\ m\ \wedge$
$\quad\quad (\forall\ A\ p.\ (\textit{card}\ A > n \wedge \textit{finite-profile}\ A\ p) \longrightarrow \textit{card}\ (\textit{reject}\ m\ A\ p) = n)$

definition *elects* :: $\textit{nat} \Rightarrow\ 'a\ \textit{Electoral-Module} \Rightarrow \textit{bool}$ **where**
$\textit{elects}\ n\ m \equiv$
$\quad \textit{electoral-module}\ m\ \wedge$
$\quad\quad (\forall\ A\ p.\ (\textit{card}\ A \geq n \wedge \textit{finite-profile}\ A\ p) \longrightarrow \textit{card}\ (\textit{elect}\ m\ A\ p) = n)$

definition *indep-of-alt* :: $'a\ \textit{Electoral-Module} \Rightarrow\ 'a\ \textit{set} \Rightarrow\ 'a \Rightarrow \textit{bool}$ **where**
$\textit{indep-of-alt}\ m\ A\ a \equiv$

electoral-module $m \wedge (\forall\, p\ q.\ \textit{equiv-prof-except-a}\ A\ p\ q\ a \longrightarrow m\ A\ p = m\ A\ q)$

2.1.3 Equivalence Definitions

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

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

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

definition *mod-contains-result* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒ *'a set* ⇒ *'a Profile* ⇒ *'a* ⇒ *bool* **where**
mod-contains-result $m\ n\ A\ p\ a \equiv$
electoral-module $m \wedge$ *electoral-module* $n \wedge$ *finite-profile* $A\ p \wedge a \in A \wedge$
$(a \in \textit{elect}\ m\ A\ p \longrightarrow a \in \textit{elect}\ n\ A\ p) \wedge$
$(a \in \textit{reject}\ m\ A\ p \longrightarrow a \in \textit{reject}\ n\ A\ p) \wedge$
$(a \in \textit{defer}\ m\ A\ p \longrightarrow a \in \textit{defer}\ n\ A\ p)$

2.1.4 Auxiliary Lemmata

lemma *combine-ele-rej-def*:
assumes
ele: *elect* $m\ A\ p = e$ **and**
rej: *reject* $m\ A\ p = r$ **and**
def: *defer* $m\ A\ p = d$
shows $m\ A\ p = (e,\ r,\ d)$
using *def ele rej*
by *auto*

lemma *par-comp-result-sound*:
assumes
mod-m: *electoral-module* m **and**
f-prof: *finite-profile* $A\ p$
shows *well-formed* $A\ (m\ A\ p)$

using *electoral-module-def mod-m f-prof*
by *auto*

lemma *result-presv-alts*:
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
x :: $'a$
assume
asm: $x \in elect\ m\ A\ p$
have *partit*:
$\forall A\ p.$
$\neg$ *set-equals-partition* (A::$'a$ *set*) p $\vee$
$(\exists B\ C\ D\ E.\ A = B \wedge p = (C, D, E) \wedge C \cup D \cup E = B)$
by *simp*
from *e-mod f-prof* **have** *set-partit*:
set-equals-partition A ($m\ A\ p$)
using *electoral-module-def*
by *auto*
thus $x \in A$
using *UnI1 asm fstI set-partit partit*
by (*metis* (*no-types*))
next
fix
x :: $'a$
assume
asm: $x \in reject\ m\ A\ p$
have *partit*:
$\forall A\ p.$
$\neg$ *set-equals-partition* (A::$'a$ *set*) p $\vee$
$(\exists B\ C\ D\ E.\ A = B \wedge p = (C, D, E) \wedge C \cup D \cup E = B)$
by *simp*
from *e-mod f-prof* **have** *set-partit*:
set-equals-partition A ($m\ A\ p$)
using *electoral-module-def*
by *auto*
thus $x \in A$
using *UnI1 asm fstI set-partit partit*
sndI subsetD sup-ge2
by *metis*
next
fix
x :: $'a$
assume
asm: $x \in defer\ m\ A\ p$
have *partit*:

$\forall\, A\ p.$
  $\neg$ *set-equals-partition* $(A{::}'a\ set)\ p \vee$
    $(\exists\, B\ C\ D\ E.\ A = B \wedge p = (C,\ D,\ E) \wedge C \cup D \cup E = B)$
by *simp*
from *e-mod f-prof* **have** *set-partit*:
set-equals-partition $A\ (m\ A\ p)$
using *electoral-module-def*
by *auto*
thus $x \in A$
using *asm set-partit partit sndI subsetD sup-ge2*
by *metis*
next
fix
$x :: 'a$
assume
asm1: $x \in A$ **and**
asm2: $x \notin$ *defer* $m\ A\ p$ **and**
asm3: $x \notin$ *reject* $m\ A\ p$
have *partit*:
$\forall\, A\ p.$
  $\neg$ *set-equals-partition* $(A{::}'a\ set)\ p \vee$
    $(\exists\, B\ C\ D\ E.\ A = B \wedge p = (C,\ D,\ E) \wedge C \cup D \cup E = B)$
by *simp*
from *e-mod f-prof* **have** *set-partit*:
set-equals-partition $A\ (m\ A\ p)$
using *electoral-module-def*
by *auto*
show $x \in$ *elect* $m\ A\ p$
using *asm1 asm2 asm3 fst-conv partit*
set-partit snd-conv Un-iff
by *metis*
qed

lemma *result-disj*:
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
$x :: 'a$
assume
asm1: $x \in$ *elect* $m\ A\ p$ **and**
asm2: $x \in$ *reject* $m\ A\ p$
have *partit*:
$\forall\, A\ p.$

$\neg$ *set-equals-partition* $(A{::}'a\ set)\ p \vee$
$(\exists B\ C\ D\ E.\ A = B \wedge p = (C,\ D,\ E) \wedge C \cup D \cup E = B)$
by *simp*
from *module profile* **have** *set-partit*:
set-equals-partition $A\ (m\ A\ p)$
using *electoral-module-def*
by *auto*
from *profile* **have** *prof-p*:
finite $A \wedge$ *profile* $A\ p$
by *simp*
from *module prof-p* **have** *wf-A-m*:
well-formed $A\ (m\ A\ p)$
using *electoral-module-def*
by *metis*
show *False*
using *prod.exhaust-sel DiffE UnCI asm1 asm2*
module profile result-imp-rej wf-A-m
prof-p set-partit partit
by (*metis* (*no-types*))
next
fix
$x :: 'a$
assume
asm1: $x \in$ *elect* $m\ A\ p$ **and**
asm2: $x \in$ *defer* $m\ A\ p$
have *partit*:
$\forall A\ p.$
$\neg$ *set-equals-partition* $(A{::}'a\ set)\ p \vee$
$(\exists B\ C\ D\ E.\ A = B \wedge p = (C,\ D,\ E) \wedge C \cup D \cup E = B)$
by *simp*
have *disj*:
$\forall p.\ \neg$ *disjoint3* $p \vee$
$(\exists B\ C\ D.\ p = (B{::}'a\ set,\ C,\ D) \wedge$
$B \cap C = \{\} \wedge B \cap D = \{\} \wedge C \cap D = \{\})$
by *simp*
from *profile* **have** *prof-p*:
finite $A \wedge$ *profile* $A\ p$
by *simp*
from *module prof-p* **have** *wf-A-m*:
well-formed $A\ (m\ A\ p)$
using *electoral-module-def*
by *metis*
hence *wf-A-m-0*:
disjoint3 $(m\ A\ p) \wedge$ *set-equals-partition* $A\ (m\ A\ p)$
by *simp*
hence *disj3*:
disjoint3 $(m\ A\ p)$
by *simp*
have *set-partit*:

set-equals-partition A (*m A p*)
using *wf-A-m-0*
by *simp*
from *disj3* **obtain**
AA :: *'a Result* ⇒ *'a set* **and**
AAa :: *'a Result* ⇒ *'a set* **and**
AAb :: *'a Result* ⇒ *'a set*
where
m A p =
(*AA* (*m A p*), *AAa* (*m A p*), *AAb* (*m A p*)) ∧
AA (*m A p*) ∩ *AAa* (*m A p*) = {} ∧
AA (*m A p*) ∩ *AAb* (*m A p*) = {} ∧
AAa (*m A p*) ∩ *AAb* (*m A p*) = {}
using *asm1 asm2 disj*
by *metis*
hence ((*elect m A p*) ∩ (*reject m A p*) = {}) ∧
((*elect m A p*) ∩ (*defer m A p*) = {}) ∧
((*reject m A p*) ∩ (*defer m A p*) = {})
using *disj3 eq-snd-iff fstI*
by *metis*
thus *False*
using *asm1 asm2 module profile wf-A-m prof-p*
set-partit partit disjoint-iff-not-equal
by (*metis* (*no-types*))
next
fix
x :: *'a*
assume
asm1: *x* ∈ *reject m A p* **and**
asm2: *x* ∈ *defer m A p*
have *partit*:
∀ *A p*.
¬ *set-equals-partition* (*A*::*'a set*) *p* ∨
(∃ *B C D E*. *A* = *B* ∧ *p* = (*C*, *D*, *E*) ∧ *C* ∪ *D* ∪ *E* = *B*)
by *simp*
from *module profile* **have** *set-partit*:
set-equals-partition A (*m A p*)
using *electoral-module-def*
by *auto*
from *profile* **have** *prof-p*:
finite A ∧ *profile A p*
by *simp*
from *module prof-p* **have** *wf-A-m*:
well-formed A (*m A p*)
using *electoral-module-def*
by *metis*
show *False*
using *prod.exhaust-sel DiffE UnCI asm1 asm2*
module profile result-imp-rej wf-A-m

prof-p set-partit partit
by (*metis* (*no-types*))
qed

lemma *elect-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *elect m A p* $\subseteq$ *A*
using *le-supI1 e-mod f-prof result-presv-alts sup-ge1*
by *metis*

lemma *reject-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *reject m A p* $\subseteq$ *A*
using *le-supI1 e-mod f-prof result-presv-alts sup-ge2*
by *fastforce*

lemma *defer-in-alts*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p*
shows *defer m A p* $\subseteq$ *A*
using *e-mod f-prof result-presv-alts*
by *auto*

lemma *def-presv-fin-prof*:
assumes *module*: *electoral-module m* **and**
f-prof: *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 module f-prof
by *metis*

lemma *upper-card-bounds-for-result*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *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*
by (*simp add*: *card-mono defer-in-alts elect-in-alts*
e-mod f-prof reject-in-alts)

lemma *reject-not-elec-or-def*:
  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 −
  from *e-mod f-prof* **have** *0*: *well-formed A* (*m A p*)
    by (*simp add*: *electoral-module-def*)
  with *e-mod f-prof*
    have (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
      using *result-presv-alts*
      by *simp*
    moreover from *0* **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*:
  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** *0*: *well-formed A* (*m A p*)
    by (*simp add*: *electoral-module-def*)
  hence
    disjoint3 (*m A p*) ∧ *set-equals-partition A* (*m A p*)
    by *simp*
  with *e-mod f-prof*
  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 from *0* **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*:
  assumes
    e-mod: *electoral-module m* **and**

f-prof: *finite-profile A p*
shows *defer m A p* = *A* − (*elect m A p*) − (*reject m A p*)
proof −
from *e-mod f-prof* **have** *0*: *well-formed A* (*m A p*)
by (*simp add*: *electoral-module-def*)
hence (*elect m A p*) ∪ (*reject m A p*) ∪ (*defer m A p*) = *A*
using *e-mod f-prof result-presv-alts*
by *auto*
moreover from *0* **have**
(*elect m A p*) ∩ (*defer 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 *electoral-mod-defer-elem*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p* **and**
alternative: *x* ∈ *A* **and**
not-elected: *x* ∉ *elect m A p* **and**
not-rejected: *x* ∉ *reject m A p*
shows *x* ∈ *defer m A p*
using *DiffI e-mod f-prof alternative*
not-elected not-rejected
reject-not-elec-or-def
by *metis*

lemma *mod-contains-result-comm*:
assumes *mod-contains-result m n A p a*
shows *mod-contains-result n m A p a*
unfolding *mod-contains-result-def*
proof (*safe*)
show *electoral-module n*
using *assms mod-contains-result-def*
by *metis*
next
show *electoral-module m*
using *assms mod-contains-result-def*
by *metis*
next
show *finite A*
using *assms mod-contains-result-def*
by *metis*
next
show *profile A p*
using *assms mod-contains-result-def*

by *metis*
next
show $a \in A$
using *assms mod-contains-result-def*
by *metis*
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*:
assumes
e-mod: *electoral-module m* **and**
f-prof: *finite-profile A p* **and**
alternative: $x \in A$ **and**
not-rejected: $x \notin$ *reject m A p*
shows $x \in$ *elect m A p* $\vee$ $x \in$ *defer m A p*
using *alternative electoral-mod-defer-elem e-mod not-rejected f-prof*
by *metis*

lemma *single-elim-imp-red-def-set*:
assumes
eliminating: *eliminates 1 m* **and**
leftover-alternatives: *card A* $>$ *1* **and**
f-prof: *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 eliminating eq-iff leftover-alternatives not-one-le-zero f-prof psubsetI reject-not-elec-or-def*
by *metis*

lemma *eq-alts-in-profs-imp-eq-results*:
assumes

eq: $\forall a \in A.$ *prof-contains-result m A p q a* **and**

input: *electoral-module m* $\wedge$ *finite-profile A p* $\wedge$ *finite-profile A q*
shows *m A p* = *m A q*
proof −
have $\forall a \in$ *elect m A p*. $a \in$ *elect m A q*
using *elect-in-alts eq prof-contains-result-def input in-mono*
by *metis*
moreover have $\forall a \in$ *elect m A q*. $a \in$ *elect m A p*
using *contra-subsetD disjoint-iff-not-equal elect-in-alts electoral-mod-defer-elem eq prof-contains-result-def input result-disj*
proof −
fix *aa* :: *'a*
have $\forall$ *A Aa*. (*Aa*::*'a set*) $\cap$ *A* = {} $\vee$ *Aa* $\neq$ {}
by *blast*
moreover have *f1*: *elect m A q* − *A* = {}
using *Diff-eq-empty-iff elect-in-alts input*
by *metis*
moreover have *f2*: *defer m A q* $\cap$ *elect m A q* = {}
using *disjoint-iff-not-equal input result-disj*
by (*metis* (*no-types*))
moreover have *f3*: *reject m A q* $\cap$ *elect m A q* = {}
using *disjoint-iff-not-equal input result-disj*
by (*metis* (*no-types*))
ultimately have *f4*:
($\exists$ *Aa*. *Aa* $\cap$ *elect m A q* = {} $\wedge$ *aa* $\in$ *Aa*) $\vee$
aa $\notin$ *elect m A q* $\vee$ *aa* $\in$ *elect m A p*
using *DiffI electoral-mod-defer-elem eq prof-contains-result-def*
by (*metis* (*no-types*))
hence *f5*:
aa $\notin$ *elect m A q* $\vee$ *aa* $\in$ *elect m A p*
using *disjoint-iff-not-equal*
by *metis*
from *f1 f2 f3*
show *?thesis*
using *Diff-iff Int-iff empty-iff eq not-rej-imp-elec-or-def 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 input in-mono*
by *fastforce*
moreover have
$\forall a \in$ *reject m A q*. $a \in$ *reject m A p*
proof −
{
fix *aa* :: *'a*

have *ff1*:
  $\forall$ *f*. *reject f A q* $\subseteq$ *A* $\vee$ $\neg$ *electoral-module f*
  using *input reject-in-alts*
  by *metis*
have *ff2*:
  $\forall$ *f*. *reject f A q* $\cap$ *defer f A q* = {} $\vee$ $\neg$ *electoral-module f*
  using *input result-disj*
  by *metis*
have *electoral-module m* $\wedge$ *profile A p* $\wedge$ *finite-profile A q*
  using *input*
  by *blast*
hence *ff3*:
  elect m A q $\cap$ *reject m A q* = {} $\wedge$ *elect m A q* $\cap$ *defer m A q* = {} $\wedge$
    reject m A q $\cap$ *defer m A q* = {}
  by (*simp add*: *result-disj*)
hence
  aa $\in$ *elect m A q* $\longrightarrow$ *aa* $\notin$ *reject m A q* $\vee$ *aa* $\in$ *reject m A p*
  using *disjoint-iff-not-equal ff3*
  by *metis*
hence *aa* $\notin$ *reject m A q* $\vee$ *aa* $\in$ *reject m A p*
  using *ff2 ff1 electoral-mod-defer-elem eq*
      input prof-contains-result-def
  by *fastforce*
}
thus *?thesis*
  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 input in-mono*
  by *fastforce*
moreover have $\forall$ *a* $\in$ *defer m A q*. *a* $\in$ *defer m A p*
proof (*safe*)
  fix *a* :: *'a*
  assume *a* $\in$ *defer m A q*
  thus *a* $\in$ *defer m A p*
    using *calculation defer-not-elec-or-rej*
        input subsetI subset-antisym
    by (*metis*)
qed
ultimately show *?thesis*
  using *prod.collapse subsetI subset-antisym*
  by *metis*
qed

lemma *eq-def-and-elect-imp-eq*:
  assumes
    electoral-module m **and**
    electoral-module n **and**
    finite-profile A p **and**

finite-profile A q **and**
elect m A p = *elect n A q* **and**
defer m A p = *defer n A q*
shows *m A p* = *n A q*
proof −
have *disj-m*:
disjoint3 (*m A p*)
using *assms*(*1*) *assms*(*3*) *electoral-module-def*
by *auto*
have *disj-n*:
disjoint3 (*n A q*)
using *assms*(*2*) *assms*(*4*) *electoral-module-def*
by *auto*
have *set-partit-m*:
set-equals-partition A ((*elect m A p*), (*reject m A p*), (*defer m A p*))
using *assms*(*1*) *assms*(*3*) *electoral-module-def*
by *auto*
moreover have
disjoint3 ((*elect m A p*),(*reject m A p*),(*defer m A p*))
using *disj-m prod.collapse*
by *metis*
have *set-partit-n*:
set-equals-partition A ((*elect n A q*), (*reject n A q*), (*defer n A q*))
using *assms*(*2*) *assms*(*4*) *electoral-module-def*
by *auto*
moreover have
disjoint3 ((*elect n A q*),(*reject n A q*),(*defer n A q*))
using *disj-n prod.collapse*
by *metis*
have *reject-p*:
reject m A p = *A* − ((*elect m A p*) ∪ (*defer m A p*))
using *assms*(*1*) *assms*(*3*) *combine-ele-rej-def*
electoral-module-def result-imp-rej
by *metis*
have *reject-q*:
reject n A q = *A* − ((*elect n A q*) ∪ (*defer n A q*))
using *assms*(*2*) *assms*(*4*) *combine-ele-rej-def*
electoral-module-def result-imp-rej
by *metis*
from *reject-p reject-q*
show *?thesis*
by (*simp add*: *assms*(*5*) *assms*(*6*) *prod-eqI*)
qed

2.1.5 Non-Blocking

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

```
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:
  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 x :: 'a
  assume
    electX: x ∈ 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)
  with electing have elect m A p ⊆ A
    unfolding electing-def
    by metis
  with electX show x ∈ A
    by auto
next
  fix x :: 'a
  assume x ∈ A
  with electing f-prof one-alt
  show x ∈ elect m A p
    unfolding electing-def
    using One-nat-def Suc-leI card-seteq
          card-gt-0-iff elect-in-alts
          infinite-super
    by metis
qed

theorem electing-imp-non-blocking:
  assumes electing: electing m
  shows non-blocking m
  unfolding non-blocking-def
proof (safe)
  from electing
  show electoral-module m
    using electing-def
```

by *metis*
next
fix
$A :: 'a\ set$ **and**
$p :: 'a\ Profile$ **and**
$x :: 'a$
assume
finA: *finite A* **and**
profA: *profile A p* **and**
rejA: *reject m A p* $=$ *A* **and**
xInA: $x \in A$
from *electing* **have**
electoral-module m $\wedge$
$(\forall$ *A rs*. $A = \{\} \vee$ *infinite A* $\vee$
$\neg$ *profile A rs* $\vee$ *elect m A rs* $\neq \{\})$
unfolding *electing-def*
by *metis*
hence *f1*: $A = \{\}$
using *Diff-cancel Un-empty*
elec-and-def-not-rej
finA profA rejA
by (*metis* (*no-types*))
from *xInA*
have $x \in A$
by *metis*
with *f1* **show** $x \in \{\}$
by *metis*
qed

2.1.7 Properties

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*:
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 *rejects 1 m*
using *One-nat-def rejecting*
by *metis*
moreover have
electoral-module m $\wedge$
$(\forall$ *A rs*. *infinite A* $\vee \neg$ *profile A rs* $\vee$ *elect m A rs* $= \{\})$

```
    using non-electing
    unfolding non-electing-def
    by metis
  moreover from this have
    reject m A p ⊆ A
    using f-prof reject-in-alts
    by metis
  ultimately show ?thesis
    using f-prof 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-card2:
  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
    ∀ f. (non-electing f ∨
          (∃ A rs. ({}::'a set) ≠ elect f A rs ∧ profile A rs ∧ finite A) ∨
          ¬ electoral-module f) ∧
        ((∀ A rs. {} = elect f A rs ∨ ¬ profile A rs ∨ infinite A) ∧
          electoral-module f ∨ ¬ non-electing f)
    using non-electing-def
    by metis
  moreover from this have
    electoral-module m ∧
      (∀ A rs. infinite A ∨ ¬ profile A rs ∨ elect m A rs = {})
    using non-electing
    by (metis (no-types))
  moreover from this have
    reject m A p ⊆ A
    using f-prof reject-in-alts
    by metis
  moreover have 1 < card A
    using not-empty
    by presburger
  moreover have eliminates 1 m
    using One-nat-def eliminating
    by presburger
  moreover have eliminates 1 m
    using eliminating
    by force
  ultimately show ?thesis
```

using *f-prof*
by (*simp add*: *card-Diff-subset defer-not-elec-or-rej*
eliminates-def finite-subset)
qed

definition *defer-deciding* :: *'a Electoral-Module* ⇒ *bool* **where**
defer-deciding m ≡
electoral-module m ∧ *non-electing m* ∧ *defers 1 m*

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 w*. *condorcet-winner A p w* ∧ *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 w*. *condorcet-winner A p w* ∧ *finite A* ⟶
(*w* ∉ *reject m A p* ∧
(∀ *l*. ¬*condorcet-winner A p l* ⟶ *l* ∉ *elect m A p*) ∧
(*w* ∈ *elect m A p* ⟶
(∀ *l*. ¬*condorcet-winner A p l* ⟶ *l* ∈ *reject m A p*))))

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

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*)

```
definition disjoint-compatibility :: 'a Electoral-Module ⇒
                                     'a Electoral-Module ⇒ bool where
  disjoint-compatibility m n ≡
    electoral-module m ∧ electoral-module n ∧
        (∀ S. finite S ⟶
          (∃ A ⊆ S.
            (∀ a ∈ A. indep-of-alt m S a ∧
              (∀ p. finite-profile S p ⟶ a ∈ reject m S p)) ∧
            (∀ a ∈ S−A. indep-of-alt n S a ∧
              (∀ p. finite-profile S p ⟶ a ∈ reject n S p))))
```

```
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}))
```

```
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:
  assumes ccomp: condorcet-compatibility m and
          dd: defer-deciding m and
          winner: condorcet-winner A p w
  shows defer m A p = {w}
proof (rule ccontr)
  assume not-w: defer m A p ≠ {w}
  from dd have def-1:
    defers 1 m
    using defer-deciding-def
    by metis
  hence c-win:
    finite-profile A p ∧  w ∈ A ∧ (∀ x ∈ A − {w} . wins w p x)
    using winner
    by simp
  hence card (defer m A p) = 1
    using One-nat-def Suc-leI card-gt-0-iff
          def-1 defers-def equals0D
    by metis
  hence 0: ∃ x ∈ A . defer m A p ={x}
    using card-1-singletonE dd defer-deciding-def
```

```
        defer-in-alts insert-subset c-win
    by metis
  with not-w have ∃ l ∈ A . l ≠ w ∧ defer m A p = {l}
    by metis
  hence not-in-defer: w ∉ defer m A p
    by auto
  have non-electing m
    using dd defer-deciding-def
    by metis
  hence not-in-elect: w ∉ elect m A p
    using c-win equals0D non-electing-def
    by metis
  from not-in-defer not-in-elect have one-side:
    w ∈ reject m A p
    using ccomp condorcet-compatibility-def c-win
        electoral-mod-defer-elem
    by metis
  from ccomp have other-side: w ∉ reject m A p
    using condorcet-compatibility-def c-win winner
    by (metis (no-types, hide-lams))
  thus False
    by (simp add: one-side)
qed

theorem ccomp-and-dd-imp-dcc[simp]:
  assumes ccomp: condorcet-compatibility m and
        dd: defer-deciding m
  shows defer-condorcet-consistency m
proof (unfold defer-condorcet-consistency-def, auto)
  show electoral-module m
    using dd defer-deciding-def
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a
  assume
    prof-A: profile A p and
    w-in-A: w ∈ A and
    finiteness: finite A and
    assm: ∀ x∈A − {w}.
        card {i. i < length p ∧ (w, x) ∈ (p!i)} <
          card {i. i < length p ∧ (x, w) ∈ (p!i)}
  have winner: condorcet-winner A p w
    using assm finiteness prof-A w-in-A
    by simp
  hence
    m A p =
```

$(\{\},$
$A - \textit{defer}\ m\ A\ p,$
$\{d \in A.\ \textit{condorcet-winner}\ A\ p\ d\})$

proof −

from *dd* **have** *0*:
elect m A p = {}
using *defer-deciding-def non-electing-def*
winner
by *fastforce*

from *dd ccomp* **have** *1*: *defer m A p* = {*w*}
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 defer-deciding-def*
reject-not-elec-or-def winner
by *fastforce*
from *0 1 2* **have** *3*: *m A p* = ({}, *A* − *defer m A p*, {*w*})
using *combine-ele-rej-def*
by *metis*
have {*w*} = {*d* ∈ *A*. *condorcet-winner A p d*}
using *cond-winner-unique3 winner*
by *metis*
thus *?thesis*
using *3*
by *auto*
qed
hence
m A p =
({},
A − *defer m A p*,
$\{d \in A.\ \forall x \in A - \{d\}.\ \textit{wins}\ d\ p\ x\})$
using *finiteness prof-A winner Collect-cong*
by *auto*
hence
m A p =
({},
A − *defer m A p*,
$\{d \in A.\ \forall x \in A - \{d\}.$
$\textit{prefer-count}\ p\ x\ d < \textit{prefer-count}\ p\ d\ x\})$
by *simp*
hence
m A p =
({},
A − *defer m A p*,
$\{d \in A.\ \forall x \in A - \{d\}.$
$\textit{card}\ \{i.\ i < \textit{length}\ p \wedge (\textit{let}\ r = (p!i)\ \textit{in}\ (d \preceq_r x))\} <$

$card\ \{i.\ i < length\ p \wedge (let\ r = (p!i)\ in\ (x \preceq_r d))\}\})$
by *simp*
thus
$m\ A\ p =$
$(\{\},$
$A - defer\ m\ A\ p,$
$\{d \in A.\ \forall x{\in}A - \{d\}.$
$card\ \{i.\ i < length\ p \wedge (d,\ x) \in (p!i)\} <$
$card\ \{i.\ i < length\ p \wedge (x,\ d) \in (p!i)\}\})$
by *simp*
qed

theorem *disj-compat-comm*[*simp*]:
assumes *compatible*: *disjoint-compatibility m n*
shows *disjoint-compatibility n m*
proof −
have
$\forall S.\ finite\ S \longrightarrow$
$(\exists A \subseteq S.$
$(\forall a \in A.\ indep\text{-}of\text{-}alt\ n\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ n\ S\ p)) \wedge$
$(\forall a \in S{-}A.\ indep\text{-}of\text{-}alt\ m\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ m\ S\ p)))$
proof
fix
$S :: 'a\ set$
obtain A **where** *old-A*:
$finite\ S \longrightarrow$
$(A \subseteq S \wedge$
$(\forall a \in A.\ indep\text{-}of\text{-}alt\ m\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ m\ S\ p)) \wedge$
$(\forall a \in S{-}A.\ indep\text{-}of\text{-}alt\ n\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ n\ S\ p)))$
using *compatible disjoint-compatibility-def*
by *fastforce*
hence
$finite\ S \longrightarrow$
$(\exists A \subseteq S.$
$(\forall a \in S{-}A.\ indep\text{-}of\text{-}alt\ n\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ n\ S\ p)) \wedge$
$(\forall a \in A.\ indep\text{-}of\text{-}alt\ m\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ m\ S\ p)))$
by *auto*
hence
$finite\ S \longrightarrow$
$(\exists A \subseteq S.$
$(\forall a \in S{-}A.\ indep\text{-}of\text{-}alt\ n\ S\ a \wedge$
$(\forall p.\ finite\text{-}profile\ S\ p \longrightarrow a \in reject\ n\ S\ p)) \wedge$

```
          (∀ a ∈ S−(S−A). indep-of-alt m S a ∧
            (∀ p. finite-profile S p ⟶ a ∈ reject m S p)))
      using double-diff order-refl
      by metis
    thus
      finite S ⟶
        (∃ A ⊆ S.
          (∀ a ∈ A. indep-of-alt n S a ∧
            (∀ p. finite-profile S p ⟶ a ∈ reject n S p)) ∧
          (∀ a ∈ S−A. indep-of-alt m S a ∧
            (∀ p. finite-profile S p ⟶ a ∈ reject m S p)))
      by fastforce
  qed
  moreover have electoral-module m ∧ electoral-module n
    using compatible disjoint-compatibility-def
    by auto
  ultimately show ?thesis
    by (simp add: disjoint-compatibility-def)
qed


theorem dl-inv-imp-def-mono[simp]:
  assumes defer-lift-invariance m
  shows defer-monotonicity m
  using assms defer-monotonicity-def defer-lift-invariance-def
  by fastforce
```

2.1.9 Social Choice Properties

Condorcet Consistency

```
definition condorcet-consistency :: 'a Electoral-Module ⇒ bool where
  condorcet-consistency m ≡
    electoral-module m ∧
    (∀ A p w. condorcet-winner A p w ⟶
      (m A p =
        ({e ∈ A. condorcet-winner A p e},
          A − (elect m A p),
          {})))

lemma condorcet-consistency2:
  condorcet-consistency m ⟷
      electoral-module m ∧
        (∀ A p w. condorcet-winner A p w ⟶
            (m A p =
              ({w}, A − (elect m A p), {})))
proof (safe)
  assume condorcet-consistency m
  thus electoral-module m
    using condorcet-consistency-def
```

by *metis*
next
fix
A :: $'a$ *set* **and**
p :: $'a$ *Profile* **and**
w :: $'a$
assume
cc: *condorcet-consistency m* **and**
cwin: *condorcet-winner A p w*
show
$m\ A\ p = (\{w\},\ A - elect\ m\ A\ p,\ \{\})$
using *cond-winner-unique3 condorcet-consistency-def cc cwin*
by (*metis* (*mono-tags*, *lifting*))
next
assume
e-mod: *electoral-module m* **and**
cwin:
$\forall A\ p\ w.$ *condorcet-winner A p w* $\longrightarrow$
$m\ A\ p = (\{w\},\ A - elect\ m\ A\ p,\ \{\})$
have
$\forall f.$ *condorcet-consistency f* $=$
(*electoral-module f* $\wedge$
($\forall A\ rs\ a.\ \neg$ *condorcet-winner A rs* (a::$'a$) $\vee$
$f\ A\ rs = (\{a \in A.$ *condorcet-winner A rs a*$\}$,
$A - elect\ f\ A\ rs,\ \{\})))$
unfolding *condorcet-consistency-def*
by *blast*
moreover have
$\forall A\ rs\ a.\ \neg$ *condorcet-winner A rs* (a::$'a$) $\vee$
$\{a \in A.$ *condorcet-winner A rs a*$\} = \{a\}$
using *cond-winner-unique3*
by (*metis* (*full-types*))
ultimately show
condorcet-consistency m
unfolding *condorcet-consistency-def*
using *cond-winner-unique3 e-mod cwin*
by *presburger*
qed

(Weak) Monotonicity

definition *monotonicity* :: $'a$ *Electoral-Module* $\Rightarrow$ *bool* **where**
monotonicity m $\equiv$
electoral-module m $\wedge$
($\forall A\ p\ q\ w.$
(*finite A* $\wedge$ $w \in$ *elect m A p* $\wedge$ *lifted A p q w*) $\longrightarrow$ $w \in$ *elect m A q*)

Homogeneity

fun *times* :: *nat* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *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* ⇒ *'a set* ⇒ *'a Profile* ⇒ *nat*

2.2.2 Property

definition *condorcet-rating* :: *'a Evaluation-Function* ⇒ *bool* **where**
  condorcet-rating f ≡
    ∀ *A p w* . *condorcet-winner A p w* ⟶
      (∀ *l* ∈ *A* . *l* ≠ *w* ⟶ *f l A p* < *f w A p*)

2.2.3 Theorems

theorem *cond-winner-imp-max-eval-val*:
  assumes
    rating: *condorcet-rating e* **and**
    f-prof: *finite-profile A p* **and**
    winner: *condorcet-winner A p w*
  shows *e w A p* = *Max* {*e a A p* | *a*. *a* ∈ *A*}
proof −

  let *?set* = {*e a A p* | *a*. *a* ∈ *A*} **and**
      ?eMax = *Max* {*e a A p* | *a*. *a* ∈ *A*} **and**
      ?eW = *e w 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 *a* :: *'a*
  assume *aInA*: *a* $\in$ *A*
  have $\forall$ *n na*. (*n*::*nat*) $\neq$ *na* $\vee$ *n* $\leq$ *na*
    by *simp*
  with *aInA* **show** *e a A p* $\leq$ *e w A p*
    using *condorcet-rating-def*
        less-imp-le rating winner
    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

theorem *non-cond-winner-not-max-eval*:
  assumes
    rating: *condorcet-rating e* **and**
    f-prof: *finite-profile A p* **and**
    winner: *condorcet-winner A p w* **and**
    linA: *l* $\in$ *A* **and**
    loser: *w* $\neq$ *l*
  shows *e l A p* < *Max* {*e a A p* | *a*. *a* $\in$ *A*}
proof −
  have *e l A p* < *e w A p*
    using *condorcet-rating-def linA loser rating winner*
    by *metis*
  also have *e w A p* = *Max* {*e a A p* |*a*. *a* $\in$ *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-Value* = *nat*

type-synonym *Threshold-Relation* = *nat* ⇒ *nat* ⇒ *bool*

type-synonym *'a Electoral-Set* = *'a set* ⇒ *'a Profile* ⇒ *'a set*

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

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

2.3.2 Common Eliminators

fun *less-eliminator* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
    'a Electoral-Module **where**
  less-eliminator e t A p = *elimination-module e t* (<) *A p*

fun *max-eliminator* :: *'a Evaluation-Function* ⇒ *'a Electoral-Module* **where**
  max-eliminator e A p =
    less-eliminator e (*Max* {*e x A p* | *x*. *x* ∈ *A*}) *A p*

fun *leq-eliminator* :: *'a Evaluation-Function* ⇒ *Threshold-Value* ⇒
'a Electoral-Module **where**
leq-eliminator e t A p = *elimination-module e t* (≤) *A p*

fun *min-eliminator* :: *'a Evaluation-Function* ⇒ *'a Electoral-Module* **where**
min-eliminator e A p =
leq-eliminator e (*Min* {*e x A p* | *x*. *x* ∈ *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*]: *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*]: *electoral-module* (*less-eliminator e t*)
unfolding *electoral-module-def*
proof (*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*]: *electoral-module* (*leq-eliminator e t*)
unfolding *electoral-module-def*
proof (*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*]: *electoral-module* (*max-eliminator e*)
unfolding *electoral-module-def*
proof (*safe*, *simp*)
fix
A :: *'a set* **and**
p :: *'a Profile*
show
$\{a \in A.\ e\ a\ A\ p < Max\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \neq A \longrightarrow$
$\{a \in A.\ e\ a\ A\ p < Max\ \{e\ x\ A\ p\ |x.\ x \in A\}\} \cup A = A$
by *safe*
qed

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

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

lemma *leq-avg-elim-sound*[*simp*]: *electoral-module* (*leq-average-eliminator e*)
unfolding *electoral-module-def*
proof (*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*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*elimination-module e t r*)
by (*simp add*: *non-electing-def*)

lemma *less-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*less-eliminator e t*)
using *elim-mod-non-electing profile less-elim-sound*
by (*simp add*: *non-electing-def*)

lemma *leq-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*leq-eliminator e t*)
proof –
have *non-electing* (*elimination-module e t* ($\leq$))
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *max-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*max-eliminator e*)
proof –
have *non-electing* (*elimination-module e t* ($<$))
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

lemma *min-elim-non-electing*:
assumes *profile*: *finite-profile A p*
shows *non-electing* (*min-eliminator e*)
proof –
have *non-electing* (*elimination-module e t* ($<$))
by (*simp add*: *non-electing-def*)
thus *?thesis*
by (*simp add*: *non-electing-def*)
qed

```
lemma less-avg-elim-non-electing:
  assumes profile: finite-profile A p
  shows non-electing (less-average-eliminator e)
proof −
  have non-electing (elimination-module e t (<))
    by (simp add: non-electing-def)
  thus ?thesis
    by (simp add: non-electing-def)
qed

lemma leq-avg-elim-non-electing:
  assumes profile: finite-profile A p
  shows non-electing (leq-average-eliminator e)
proof −
  have non-electing (elimination-module e t (≤))
    by (simp add: non-electing-def)
  thus ?thesis
    by (simp add: non-electing-def)
qed
```

2.3.5 Inference Rules

```
theorem cr-eval-imp-ccomp-max-elim[simp]:
  assumes
    profile: finite-profile A p and
    rating: condorcet-rating e
  shows
    condorcet-compatibility (max-eliminator e)
  unfolding condorcet-compatibility-def
proof (auto)
  have f1:
    ⋀A p w x. condorcet-winner A p w ⟹
      finite A ⟹ w ∈ A ⟹ e w A p < Max {e x A p |x. x ∈ A} ⟹
        x ∈ A ⟹ e x A p < Max {e x A p |x. x ∈ A}
    using rating
    by (simp add: cond-winner-imp-max-eval-val)
  thus
    ⋀A p w x.
      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)} ⟹
              finite A ⟹ e w A p < Max {e x A p | x. x ∈ A} ⟹
                x ∈ A ⟹ e x A p < Max {e x A p | x. x ∈ A}
    by simp
qed

lemma cr-eval-imp-dcc-max-elim-helper1:
```

```
  assumes
    f-prof: finite-profile A p and
    rating: condorcet-rating e and
    winner: condorcet-winner A p w
  shows elimination-set e (Max {e x A p | x. x ∈ A}) (<) A p = A − {w}
proof (safe, simp-all, safe)
  assume
    w-in-A: w ∈ A and
    max: e w A p < Max {e x A p |x. x ∈ A}
  show False
    using cond-winner-imp-max-eval-val
          rating winner f-prof max
    by fastforce
next
  fix
    x :: 'a
  assume
    x-in-A: x ∈ A and
    not-max: ¬ e x A p < Max {e y A p |y. y ∈ A}
  show x = w
    using non-cond-winner-not-max-eval x-in-A
          rating winner f-prof not-max
    by (metis (mono-tags, lifting))
qed


theorem cr-eval-imp-dcc-max-elim[simp]:
  assumes rating: condorcet-rating e
  shows defer-condorcet-consistency (max-eliminator e)
  unfolding defer-condorcet-consistency-def
proof (safe, simp)
  fix
    A :: 'a set and
    p :: 'a Profile and
    w :: 'a
  assume
    winner: condorcet-winner A p w and
    finite: finite A
  let ?trsh = (Max {e y A p | y. y ∈ A})
  show
    max-eliminator e A p =
      ({},
        A − defer (max-eliminator e) A p,
        {a ∈ A. condorcet-winner A p a})
  proof (cases elimination-set e (?trsh) (<) A p ≠ A)
    case True
    have profile: finite-profile A p
      using winner
      by simp
```

```
  with rating winner have 0:
    (elimination-set e ?trsh (<) A p) = A − {w}
    using cr-eval-imp-dcc-max-elim-helper1
    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 − {w}, A − (A − {w}))
    using 0
    by presburger
  also have ... = ({}, A − {w}, {w})
    using winner
    by auto
  also have ... = ({},A − defer (max-eliminator e) A p, {w})
    using calculation
    by auto
  also have
    ... =
      ({},
        A − defer (max-eliminator e) A p,
        {d ∈ A. condorcet-winner A p d})
    using cond-winner-unique3 winner Collect-cong
    by (metis (no-types, lifting))
  finally show ?thesis
    using finite winner
    by metis
next
  case False
  thus ?thesis
  proof −
    have f1:
      finite A ∧ profile A p ∧ w ∈ A ∧ (∀ a. a ∉ A − {w} ∨ wins w p a)
      using winner
      by auto
    hence
      ?trsh = e w A p
      using rating winner
      by (simp add: cond-winner-imp-max-eval-val)
    hence False
      using f1 False
      by auto
    thus ?thesis
      by simp
  qed
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 ∪ e2,
     A − (e1 ∪ e2 ∪ d1 ∪ d2),
     (d1 ∪ d2) − (e1 ∪ e2))
```

2.5.2 Auxiliary Lemma

```
lemma max-agg-rej-set: (well-formed A (e1, r1, d1) ∧
                         well-formed A (e2, r2, d2)) ⟶
          reject-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) = r1 ∩ r2
proof −
  have well-formed A (e1, r1, d1) ⟶  A − (e1 ∪ d1) = r1
    by (simp add: result-imp-rej)
  moreover have
    well-formed A (e2, r2, d2) ⟶  A − (e2 ∪ d2) = r2
    by (simp add: result-imp-rej)
  ultimately have
    (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
        A − (e1 ∪ e2 ∪ d1 ∪ d2) = r1 ∩ r2
    by blast
  moreover have
    {l ∈ A. l ∉ e1 ∪ e2 ∪ d1 ∪ d2} = A − (e1 ∪ e2 ∪ d1 ∪ d2)
    by (simp add: set-diff-eq)
  ultimately show ?thesis
    by simp
qed
```

2.5.3 Soundness

```
theorem max-agg-sound[simp]: aggregator max-aggregator
```

unfolding *aggregator-def*
proof (*simp*, *safe*)
fix
A :: $'a$ *set* **and**
e1 :: $'a$ *set* **and**
e2 :: $'a$ *set* **and**
d1 :: $'a$ *set* **and**
d2 :: $'a$ *set* **and**
r1 :: $'a$ *set* **and**
r2 :: $'a$ *set* **and**
x :: $'a$
assume
asm1: $e2 \cup r2 \cup d2 = e1 \cup r1 \cup d1$ **and**
asm2: $x \notin d1$ **and**
asm3: $x \notin r1$ **and**
asm4: $x \in e2$
show $x \in e1$
using *asm1 asm2 asm3 asm4*
by *auto*
next
fix
A :: $'a$ *set* **and**
e1 :: $'a$ *set* **and**
e2 :: $'a$ *set* **and**
d1 :: $'a$ *set* **and**
d2 :: $'a$ *set* **and**
r1 :: $'a$ *set* **and**
r2 :: $'a$ *set* **and**
x :: $'a$
assume
asm1: $e2 \cup r2 \cup d2 = e1 \cup r1 \cup d1$ **and**
asm2: $x \notin d1$ **and**
asm3: $x \notin r1$ **and**
asm4: $x \in d2$
show $x \in e1$
using *asm1 asm2 asm3 asm4*
by *auto*
qed

2.5.4 Properties

theorem *max-agg-consv*[*simp*]: *agg-conservative max-aggregator*
proof –
have
$\forall$ *A e1 e2 d1 d2 r1 r2*.
(*well-formed A* (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*)) $\longrightarrow$
reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) = $r1 \cap r2$
using *max-agg-rej-set*
by *blast*

```
  hence
    ∀ A e1 e2 d1 d2 r1 r2.
            (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
        reject-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ r1 ∩ r2
    by blast
  moreover have
    ∀ A e1 e2 d1 d2 r1 r2.
        (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
            elect-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (e1 ∪ e2)
    by (simp add: subset-eq)
  ultimately have
    ∀ A e1 e2 d1 d2 r1 r2.
        (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
            (elect-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (e1 ∪ e2) ∧
             reject-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (r1 ∪ r2))
    by blast
  moreover have
    ∀ A e1 e2 d1 d2 r1 r2.
        (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
            defer-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (d1 ∪ d2)
    by auto
  ultimately have
    ∀ A e1 e2 d1 d2 r1 r2.
        (well-formed A (e1, r1, d1) ∧ well-formed A (e2, r2, d2)) ⟶
            (elect-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (e1 ∪ e2) ∧
            reject-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (r1 ∪ r2) ∧
            defer-r (max-aggregator A (e1, r1, d1) (e2, r2, d2)) ⊆ (d1 ∪ d2))
    by blast
  thus ?thesis
    by (simp add: agg-conservative-def)
qed


theorem max-agg-comm[simp]: agg-commutative max-aggregator
  unfolding agg-commutative-def
proof (safe)
  show aggregator max-aggregator
    by simp
next
  fix
    A :: 'a set and
    e1 :: 'a set and
    e2 :: 'a set and
    d1 :: 'a set and
    d2 :: 'a set and
    r1 :: 'a set and
    r2 :: 'a set
  show
    max-aggregator A (e1, r1, d1) (e2, r2, d2) =
```

max-aggregator A (e2, r2, d2) (e1, r1, d1)
by *auto*
qed

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

3.2.2 Soundness

theorem *drop-mod-sound*[*simp*]:
  assumes *order*: *linear-order r*
  shows *electoral-module* (*drop-module n r*)
proof −
  let *?mod* = *drop-module n r*
  have
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $(\forall a \in A.\ a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\} \lor$
        $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\})$
    by *auto*
  hence
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} \cup$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} = A$
    by *blast*
  hence *0*:
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      set-equals-partition A (*drop-module n r A p*)
    by *simp*
  have
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $(\forall a \in A.\ \neg(a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) \leq n\} \land$
        $a \in \{x \in A.\ card(above\ (limit\ A\ r)\ x) > n\}))$
    by *auto*
  hence
    $\forall A\ p.\ finite\text{-}profile\ A\ p \longrightarrow$
      $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq n\} \cap$

$\{a \in A.\ card(above\ (limit\ A\ r)\ a) > n\} = \{\}$
by *blast*
hence *1*: $\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$ *disjoint3* (*?mod* $A\ p$)
by *simp*
from *0 1* **have**
$\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$
well-formed A (*?mod* $A\ p$)
by *simp*
hence
$\forall A\ p.$ *finite-profile* $A\ p \longrightarrow$
well-formed A (*?mod* $A\ p$)
by *simp*
thus *?thesis*
using *electoral-modI*
by *metis*
qed

3.2.3 Non-Electing

theorem *drop-mod-non-electing*[*simp*]:
assumes *order*: *linear-order r*
shows *non-electing* (*drop-module n r*)
by (*simp add*: *non-electing-def order*)

3.2.4 Properties

theorem *drop-mod-def-lift-inv*[*simp*]:
assumes *order*: *linear-order r*
shows *defer-lift-invariance* (*drop-module n r*)
by (*simp add*: *order defer-lift-invariance-def*)

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 ∈ A. card(above (limit A r) a) > n},
    {a ∈ A. card(above (limit A r) a) ≤ n})
```

3.3.2 Soundness

```
theorem pass-mod-sound[simp]:
  assumes order: linear-order r
  shows electoral-module (pass-module n r)
proof −
  let ?mod = pass-module n r
  have
    ∀ A p. finite-profile A p ⟶
        (∀ a ∈ A. a ∈ {x ∈ A. card(above (limit A r) x) > n} ∨
                 a ∈ {x ∈ A. card(above (limit A r) x) ≤ n})
    using CollectI not-less
    by metis
  hence
    ∀ A p. finite-profile A p ⟶
        {a ∈ A. card(above (limit A r) a) > n} ∪
        {a ∈ A. card(above (limit A r) a) ≤ n} = A
    by blast
  hence 0:
    ∀ A p. finite-profile A p ⟶ set-equals-partition A (pass-module n r A p)
    by simp
  have
    ∀ A p. finite-profile A p ⟶
      (∀ 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 p. finite-profile A p ⟶
      {a ∈ A. card(above (limit A r) a) > n} ∩
      {a ∈ A. card(above (limit A r) a) ≤ n} = {}
    by blast
  hence 1:
    ∀ A p. finite-profile A p ⟶ disjoint3 (?mod A p)
    by simp
  from 0 1
  have
    ∀ A p. finite-profile A p ⟶ well-formed A (?mod A p)
    by simp
  hence
    ∀ A p. finite-profile A p ⟶ well-formed A (?mod A p)
    by simp
  thus ?thesis
```

using *electoral-modI*
by *metis*
qed

3.3.3 Non-Blocking

theorem *pass-mod-non-blocking*[*simp*]:
assumes *order*: *linear-order r* **and**
g0-n: $n > 0$
shows *non-blocking* (*pass-module n r*)
unfolding *non-blocking-def*
proof (*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**
x :: *'a*
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
card-A:
$\{a \in A.\ n <$
card (*above*
$\{(a, b).\ (a, b) \in r \wedge$
$a \in A \wedge b \in A\}\ a)\} = A$ **and**
x-in-A: $x \in A$
have *lin-ord-A*:
linear-order-on A (*limit A r*)
using *limit-presv-lin-ord order top-greatest*
by *metis*
have
$\exists\, a{\in}A.$ *above* (*limit A r*) $a = \{a\} \wedge$
$(\forall\, x{\in}A.$ *above* (*limit A r*) $x = \{x\} \longrightarrow x = a)$
using *above-one fin-A lin-ord-A x-in-A*
by *blast*
hence *not-all*:
$\{a \in A.$ *card*(*above* (*limit A r*) $a) > n\} \neq A$
using *One-nat-def Suc-leI assms*(*2*) *is-singletonI*
is-singleton-altdef leD mem-Collect-eq
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

theorem *pass-mod-non-electing*[*simp*]:
  assumes *order*: *linear-order r*
  shows *non-electing* (*pass-module n r*)
  by (*simp add*: *non-electing-def order*)

3.3.5 Properties

theorem *pass-mod-dl-inv*[*simp*]:
  assumes *order*: *linear-order r*
  shows *defer-lift-invariance* (*pass-module n r*)
  by (*simp add*: *order defer-lift-invariance-def*)

theorem *pass-zero-mod-def-zero*[*simp*]:
  assumes *order*: *linear-order r*
  shows *defers 0* (*pass-module 0 r*)
  unfolding *defers-def*
proof (*safe*)
  show *electoral-module* (*pass-module 0 r*)
    using *pass-mod-sound order*
    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*
  show
    $card\ (defer\ (pass\text{-}module\ 0\ r)\ A\ p) = 0$
  proof −
    have *lin-ord-on-A*:
      linear-order-on A (*limit A r*)
      using *order limit-presv-lin-ord*
      by *blast*
    have *f1*: *connex A* (*limit A r*)
      using *lin-ord-imp-connex lin-ord-on-A*
      by *simp*
    obtain $aa :: ('a \Rightarrow bool) \Rightarrow 'a$ **where**
      f2:
      $\forall p.\ (Collect\ p = \{\} \longrightarrow (\forall a.\ \neg\ p\ a)) \wedge$
        $(Collect\ p \neq \{\} \longrightarrow p\ (aa\ p))$
      by *moura*
    have $\forall n.\ \neg\ (n{::}nat) \leq 0 \vee n = 0$
      by *blast*
    hence
      $\forall a\ Aa.\ \neg\ connex\ Aa\ (limit\ A\ r) \vee a \notin Aa \vee a \notin A \vee$
        $\neg\ card\ (above\ (limit\ A\ r)\ a) \leq 0$

using *above-connex above-presv-limit card-eq-0-iff*
equals0D finite-A order rev-finite-subset
by (*metis* (*no-types*))
hence $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq 0\} = \{\}$
using *f1*
by *auto*
hence $card\ \{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq 0\} = 0$
using *card.empty*
by *metis*
thus $card\ (defer\ (pass\text{-}module\ 0\ r)\ A\ p) = 0$
by *simp*
qed
qed

theorem *pass-one-mod-def-one*[*simp*]:
assumes *order*: *linear-order r*
shows *defers 1* (*pass-module 1 r*)
unfolding *defers-def*
proof (*safe*)
show *electoral-module* (*pass-module 1 r*)
using *pass-mod-sound order*
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
$card\ (defer\ (pass\text{-}module\ 1\ r)\ A\ p) = 1$
proof −
have $A \neq \{\}$
using *card-pos*
by *auto*
moreover have *lin-ord-on-A*:
linear-order-on A (*limit A r*)
using *order limit-presv-lin-ord*
by *blast*
ultimately have *winner-exists*:
$\exists\, a{\in}A.\ above\ (limit\ A\ r)\ a = \{a\} \land$
$(\forall\, x{\in}A.\ above\ (limit\ A\ r)\ x = \{x\} \longrightarrow x = a)$
using *finite-A*
by (*simp add*: *above-one*)
then obtain *w* **where** *w-unique-top*:
$above\ (limit\ A\ r)\ w = \{w\} \land$
$(\forall\, x{\in}A.\ above\ (limit\ A\ r)\ x = \{x\} \longrightarrow x = w)$

using *above-one*
by *auto*
hence $\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq 1\} = \{w\}$
proof
assume
w-top: $above\ (limit\ A\ r)\ w = \{w\}$ **and**
w-unique: $\forall\, x{\in}A.\ above\ (limit\ A\ r)\ x = \{x\} \longrightarrow x = w$
have $card\ (above\ (limit\ A\ r)\ w) \leq 1$
using *w-top*
by *auto*
hence $\{w\} \subseteq \{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq 1\}$
using *winner-exists w-unique-top*
by *blast*
moreover have
$\{a \in A.\ card(above\ (limit\ A\ r)\ a) \leq 1\} \subseteq \{w\}$
proof
fix
$x :: 'a$
assume *x-in-winner-set*:
$x \in \{a \in A.\ card\ (above\ (limit\ A\ r)\ a) \leq 1\}$
hence *x-in-A*: $x \in A$
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* $x \preceq_q x$
using *connex-limit above-connex*
pref-imp-in-above x-in-A
by *metis*
hence $(x,x) \in limit\ A\ r$
by *simp*
hence *x-above-x*: $x \in above\ (limit\ A\ r)\ x$
by (*simp add*: *above-def*)
have $above\ (limit\ A\ r)\ x \subseteq A$
using *above-presv-limit order*
by *fastforce*
hence *above-finite*: *finite* (*above* (*limit A r*) *x*)
by (*simp add*: *finite-A finite-subset*)
have $card\ (above\ (limit\ A\ r)\ x) \leq 1$
using *x-in-winner-set*
by *simp*
moreover have
$card\ (above\ (limit\ A\ r)\ x) \geq 1$
using *One-nat-def Suc-leI above-finite card-eq-0-iff*
equals0D neq0-conv x-above-x
by *metis*
ultimately have
$card\ (above\ (limit\ A\ r)\ x) = 1$

by *simp*
hence $\{x\}$ = *above* (*limit A r*) *x*
using *is-singletonE is-singleton-altdef singletonD x-above-x*
by *metis*
hence $x = w$
using *w-unique*
by (*simp add*: *x-in-A*)
thus $x \in \{w\}$
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*:
assumes *order*: *linear-order r*
shows *defers 2* (*pass-module 2 r*)
unfolding *defers-def*
proof (*safe*)
show *electoral-module* (*pass-module 2 r*)
using *order*
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 order*
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 order 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*: $\forall\, b \in A.\ a \in above\ (limit\ A\ r)\ b$
by (*simp add*: *above-def*)
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*: $a \in defer\ (pass\text{-}module\ 2\ r)\ A\ p$
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 order 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 order 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 subset-UNIV above-presv-limit insert-subset order limit-presv-above limit-presv-above2*
by *metis*
moreover have *above-subset*: $above\ (limit\ A\ r)\ b \subseteq A$
using *above-presv-limit order*
by *metis*

moreover have *b-above-b*: $b \in above\ (limit\ A\ r)\ b$
  using *above-def b b-best above-presv-limit*
    mem-Collect-eq order insert-subset
  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*: $b \in defer\ (pass\text{-}module\ 2\ r)\ A\ p$
  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 *above-def mem-Collect-eq*
  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 One-nat-def 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
  by *metis*
ultimately have
  $\forall\, c \in A-\{a, b\}.\ card\ (above\ (limit\ A\ r)\ c) \geq 3$
  using *card-mono finA finite-subset above-presv-limit order*
  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 $\forall\, c \in A-\{a, b\}.\ c \notin defer\ (pass\text{-}module\ 2\ r)\ A\ p$
  by (*simp add*: *not-le*)
moreover have $defer\ (pass\text{-}module\ 2\ r)\ A\ p \subseteq A$
  by *auto*
ultimately have $defer\ (pass\text{-}module\ 2\ r)\ A\ p \subseteq \{a, b\}$
  by *blast*
with *a-in-defer b-in-defer* **have**
  $defer\ (pass\text{-}module\ 2\ r)\ A\ p = \{a, b\}$
  by *fastforce*
thus $card\ (defer\ (pass\text{-}module\ 2\ r)\ A\ p) = 2$
  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 =
({$a \in A$. $\forall x \in A$. *win-count p x* $\leq$ *win-count p a*},
{$a \in A$. $\exists x \in A$. *win-count p x* $>$ *win-count p a*},
{})

3.5.2 Soundness

theorem *plurality-sound*[*simp*]: *electoral-module plurality*
proof –
have
$\forall A\ p$.
let elect = {$a \in (A::'a\ set)$. $\forall x \in A$. *win-count p x* $\leq$ *win-count p a*};
reject = {$a \in A$. $\exists x \in A$. *win-count p x* $>$ *win-count p a*} *in*
elect $\cap$ *reject* = {}
by *auto*
hence *disjoint*:
$\forall A\ p$.
let elect = {$a \in (A::'a\ set)$. $\forall x \in A$. *win-count p x* $\leq$ *win-count p a*};
reject = {$a \in A$. $\exists x \in A$. *win-count p x* $>$ *win-count p a*} *in*
disjoint3 (*elect*, *reject*, {})
by *simp*
have
$\forall A\ p$.
let elect = {$a \in (A::'a\ set)$. $\forall x \in A$. *win-count p x* $\leq$ *win-count p a*};
reject = {$a \in A$. $\exists x \in A$. *win-count p x* $>$ *win-count p a*} *in*
elect $\cup$ *reject* = *A*
using *not-le-imp-less*
by *auto*
hence *unity*:
$\forall A\ p$.
let elect = {$a \in (A::'a\ set)$. $\forall x \in A$. *win-count p x* $\leq$ *win-count p a*};
reject = {$a \in A$. $\exists x \in A$. *win-count p x* $>$ *win-count p a*} *in*
set-equals-partition A (*elect*, *reject*, {})
by *simp*
from *disjoint unity* **show** *?thesis*
by (*simp add*: *electoral-modI*)
qed

3.5.3 Electing

lemma *plurality-electing2*: $\forall A\ p$.

```
                              (A ≠ {} ∧ finite-profile A p) ⟶
                                elect plurality A p ≠ {}
proof (intro allI impI conjI)
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    assm0: A ≠ {} ∧ finite-profile A p
  show
    elect plurality A p ≠ {}
  proof
    obtain max where
      max: max = Max(win-count p ‘ A)
      by simp
    then obtain a where
      a: win-count p a = max ∧ a ∈ A
      using Max-in assm0 empty-is-image
            finite-imageI imageE
      by (metis (no-types, lifting))
    hence
      ∀ x ∈ A. win-count p x ≤ win-count p a
      by (simp add: max assm0)
    moreover have
      a ∈ A
      using a
      by simp
    ultimately have
      a ∈ {a ∈ A. ∀ x ∈ A. win-count p x ≤ win-count p a}
      by blast
    hence a-elem:
      a ∈ elect plurality A p
      by simp
    assume
      assm1: elect plurality A p = {}
    thus False
      using a-elem assm1 all-not-in-conv
      by metis
  qed
qed


theorem plurality-electing[simp]: electing plurality
proof −
  have electoral-module plurality ∧
      (∀ A p. (A ≠ {} ∧ finite-profile A p) ⟶ elect plurality A p ≠ {})
  proof
    show electoral-module plurality
      by simp
  next
```

show ($\forall A\ p.\ (A \neq \{\} \wedge$ *finite-profile* $A\ p) \longrightarrow$ *elect plurality* $A\ p \neq \{\})$
using *plurality-electing2*
by *metis*
qed
thus *?thesis*
by (*simp add*: *electing-def*)
qed

3.5.4 Property

lemma *plurality-inv-mono2*: $\forall A\ p\ q\ a.$
$(a \in$ *elect plurality* $A\ p \wedge$ *lifted* $A\ p\ q\ a) \longrightarrow$
(*elect plurality* $A\ q =$ *elect plurality* $A\ p\ \vee$
elect plurality $A\ q = \{a\})$
proof (*intro allI impI*)
fix
$A :: {'a}$ *set* **and**
$p :: {'a}$ *Profile* **and**
$q :: {'a}$ *Profile* **and**
$a :: {'a}$
assume *asm1*:
$a \in$ *elect plurality* $A\ p \wedge$ *lifted* $A\ p\ q\ a$
have *ab1*: $\forall a\ b.\ (a{::}{'a}) \notin \{b\} \vee a = b$
by *force*
have *lifted-winner*:
$\forall x \in A.$
$\forall i{::}nat.\ i <$ *length* $p \longrightarrow$
(*above* $(p!i)\ x = \{x\} \longrightarrow$
(*above* $(q!i)\ x = \{x\} \vee$ *above* $(q!i)\ a = \{a\}))$
using *asm1 Profile.lifted-def lifted-above-winner*
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 *asm1*
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 *asm1 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 asm1 Profile.lifted-def
  by metis
hence
  ∀ x ∈ A−{a}.
    ∀ i::nat. i < length p ⟶
      (above (q!i) a = {a} ⟶ above (q!i) x ≠ {x})
  using DiffE Profile.lifted-def above-one2
        asm1 insertCI insert-absorb insert-not-empty
        profile-def sizes
  by metis
with lifted-winner
have above-QtoP:
  ∀ x ∈ A−{a}.
    ∀ i::nat. i < length p ⟶
      (above (q!i) x = {x} ⟶ above (p!i) x = {x})
  using lifted-above-winner3 asm1
        Profile.lifted-def
  by metis
hence
  ∀ x ∈ A−{a}.
    {i::nat. i < length p ∧ above (q!i) x = {x}} ⊆
      {i::nat. i < length p ∧ above (p!i) x = {x}}
  by (simp add: Collect-mono)
hence win-count-other:
  ∀ x ∈ A−{a}. win-count p x ≥ win-count q x
  by (simp add: card-mono sizes)
show
  elect plurality A q = elect plurality A p ∨
    elect plurality A q = {a}
proof cases
  assume win-count p a = win-count q a
  hence
    card {i::nat. i < length p ∧ above (p!i) a = {a}} =
      card {i::nat. i < length p ∧ above (q!i) a = {a}}
    by (simp add: sizes)
  moreover have
    finite {i::nat. i < length p ∧ above (q!i) a = {a}}
    by simp
  ultimately have
    {i::nat. i < length p ∧ above (p!i) a = {a}} =
      {i::nat. i < length p ∧ above (q!i) a = {a}}
    using a-win-subset
    by (simp add: card-subset-eq)
  hence above-pq:
    ∀ i::nat. i < length p ⟶
      above (p!i) a = {a} ⟷ above (q!i) a = {a}
    by blast
  hence above-pq2:
    ∀ n. ¬ n < length p ∨
```

```
    (above (p!n) a = {a}) = (above (q!n) a = {a})
  by presburger
moreover have
  ∀ x ∈ A−{a}.
    ∀ i::nat. i < length p ⟶
      (above (p!i) x = {x} ⟶
        (above (q!i) x = {x} ∨ above (q!i) a = {a}))
  using lifted-winner
  by auto
moreover have
  ∀ x ∈ A−{a}.
    ∀ i::nat. i < length p ⟶
      (above (p!i) x = {x} ⟶ above (p!i) a ≠ {a})
proof (rule ccontr, simp, safe, simp)
  fix
    x :: 'a and
    i :: nat
  assume
    x-in-A: x ∈ A and
    i-in-range: i < length p and
    abv-x: above (p!i) x = {x} and
    abv-a: above (p!i) a = {a}
  have not-empty:
    A ≠ {}
    using x-in-A
    by auto
  have
    linear-order-on A (p!i)
    using Profile.lifted-def asm1 i-in-range profile-def
    by (metis (no-types))
  thus x = a
    using not-empty abv-a abv-x fin-A
    by (simp add: above-one2)
qed
ultimately have above-PtoQ:
  ∀ x ∈ A−{a}.
    ∀ i::nat. i < length p ⟶
      (above (p!i) x = {x} ⟶ above (q!i) x = {x})
  by (simp add: above-pq)
hence
  ∀ x ∈ A.
    card {i::nat. i < length p ∧ above (p!i) x = {x}} =
      card {i::nat. i < length q ∧ above (q!i) x = {x}}
proof (safe)
  fix
    x :: 'a
  assume
    ∀ y ∈ A−{a}. ∀  i<length p.
      above (p!i) y = {y} ⟶ above (q!i) y = {y} and
```

```
      x-in-A: x ∈ A
    show
      card {i. i < length p ∧ above (p ! i) x = {x}} =
        card {i. i < length q ∧ above (q ! i) x = {x}}
      using DiffI x-in-A ab1 above-PtoQ above-QtoP above-pq2 sizes
      by (metis (no-types, lifting))
  qed
  hence ∀ x ∈ A. win-count p x = win-count q x
    by simp
  hence
    {a ∈ A. ∀ x ∈ A. win-count p x ≤ win-count p a} =
      {a ∈ A. ∀ x ∈ A. win-count q x ≤ win-count q a}
    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 auto
  have a-in-win-p:
    a ∈ {a ∈ A. ∀ x ∈ A. win-count p x ≤ win-count p a}
    using asm1
    by auto
  hence ∀ x ∈ A. win-count p x ≤ win-count p a
    by simp
  with strict-less win-count-other
  have less:
    ∀ x ∈ A−{a}. win-count q x < win-count q a
    using DiffD1 antisym dual-order.trans
          not-le-imp-less win-count-a
    by metis
  hence
    ∀ x ∈ A−{a}. ¬(∀ y ∈ A. win-count q y ≤ win-count q x)
    using asm1 Profile.lifted-def not-le
    by metis
  hence
    ∀ x ∈ A−{a}.
      x ∉ {a ∈ A. ∀ x ∈ A. win-count q x ≤ win-count q a}
    by blast
  hence
    ∀ x ∈ A−{a}. x ∉ elect plurality A q
    by simp
  moreover have
    a ∈ elect plurality A q
  proof −
    from less
    have
```

$\forall\, x \in A{-}\{a\}.$ *win-count q x* $\leq$ *win-count q a*
using *less-imp-le*
by *metis*
moreover have
win-count q a $\leq$ *win-count q a*
by *simp*
ultimately have
$\forall\, x \in A.$ *win-count q x* $\leq$ *win-count q a*
by *auto*
moreover have
$a \in A$
using *a-in-win-p*
by *auto*
ultimately have
$a \in \{a \in A.$
$\forall\, x \in A.$ *win-count q x* $\leq$ *win-count q a*$\}$
by *auto*
thus *?thesis*
by *simp*
qed
moreover have
elect plurality A q $\subseteq$ *A*
by *simp*
ultimately show *?thesis*
by *auto*
qed
qed

theorem *plurality-inv-mono*[*simp*]: *invariant-monotonicity plurality*
proof −
have
electoral-module plurality $\wedge$
($\forall$ *A p q a*.
($a \in$ *elect plurality A p* $\wedge$ *lifted A p q a*) $\longrightarrow$
(*elect plurality A q* $=$ *elect plurality A p* $\vee$
elect plurality A q $= \{a\}$))
proof
show *electoral-module plurality*
by *simp*
next
show
$\forall$ *A p q a*. ($a \in$ *elect plurality A p* $\wedge$ *lifted A p q a*) $\longrightarrow$
(*elect plurality A q* $=$ *elect plurality A p* $\vee$
elect plurality A q $= \{a\}$)
using *plurality-inv-mono2*
by *metis*
qed
thus *?thesis*

by (*simp add*: *invariant-monotonicity-def*)
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**
  $borda\text{-}score\ x\ A\ p = (\sum y \in A.\ (prefer\text{-}count\ p\ x\ y))$

fun *borda* :: *'a Electoral-Module* **where**
  $borda\ A\ p = max\text{-}eliminator\ borda\text{-}score\ A\ p$

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 Property

theorem *condorcet-score-is-condorcet-rating*: *condorcet-rating condorcet-score*
proof −
  have
    $\forall f.$
      $(\neg$ *condorcet-rating f* $\longrightarrow$
        $(\exists A\ rs\ a.$
          condorcet-winner A rs a $\wedge$
            $(\exists aa.\ \neg\ f\ (aa{::}'a)\ A\ rs < f\ a\ A\ rs \wedge a \neq aa \wedge aa \in A))) \wedge$
       (*condorcet-rating f* $\longrightarrow$
         $(\forall A\ rs\ a.$ *condorcet-winner A rs a* $\longrightarrow$
           $(\forall aa.\ f\ aa\ A\ rs < f\ a\ A\ rs \vee a = aa \vee aa \notin A)))$
    unfolding *condorcet-rating-def*
    by (*metis* (*mono-tags*, *hide-lams*))
  thus *?thesis*
    using *cond-winner-unique condorcet-score.simps zero-less-one*
    by (*metis* (*no-types*))
qed

theorem *condorcet-is-dcc*: *defer-condorcet-consistency condorcet*
unfolding *defer-condorcet-consistency-def electoral-module-def*
proof (*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 electoral-module-def max-elim-sound*
    by *metis*
  thus *well-formed A* (*condorcet 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-cscore-dcc:
    defer-condorcet-consistency (max-eliminator condorcet-score)
    using cr-eval-imp-dcc-max-elim
    by (simp add: condorcet-score-is-condorcet-rating)
  from defer-condorcet-consistency-def
  have
    max-eliminator condorcet-score A p =
  ({},
  A − defer (max-eliminator condorcet-score) A p,
  {a ∈ A. condorcet-winner A p a})
    using cwin-w finA max-cscore-dcc
    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 Lemmata

lemma *cond-winner-imp-win-count*:
  assumes *winner*: *condorcet-winner A p w*
  shows $card\ \{y \in A\ .\ wins\ w\ p\ y\} = card\ A\ -1$
proof −
  from *winner*
  have *0*: $\forall\, x \in A - \{w\}\ .\ wins\ w\ p\ x$
    by *simp*
  have *1*: $\forall\, M\ .\ \{x \in M\ .\ True\} = M$
    by *blast*
  from *0 1*
  have $\{x \in A - \{w\}\ .\ wins\ w\ p\ x\} = A - \{w\}$
    by *blast*
  hence *10*: $card\ \{x \in A - \{w\}\ .\ wins\ w\ p\ x\} = card\ (A - \{w\})$
    by *simp*
  from *winner*
  have *11*: $w \in A$
    by *simp*
  hence $card\ (A - \{w\}) = card\ A - 1$
    using *card-Diff-singleton condorcet-winner.simps winner*
    by *metis*
  hence *amount1*:
    $card\ \{x \in A - \{w\}\ .\ wins\ w\ p\ x\} = card\ (A) - 1$
    using *10*
    by *linarith*
  have *2*: $\forall\, x \in \{w\}\ .\ \neg\ wins\ x\ p\ x$
    by (*simp add*: *wins-irreflex*)
  have *3*: $\forall\, M\ .\ \{x \in M\ .\ False\} = \{\}$
    by *blast*
  from *2 3*
  have $\{x \in \{w\}\ .\ wins\ w\ p\ x\} = \{\}$
    by *blast*
  hence *amount2*: $card\ \{x \in \{w\}\ .\ wins\ w\ p\ x\} = 0$
    by *simp*
  have *disjunct*:
    $\{x \in A - \{w\}\ .\ wins\ w\ p\ x\} \cap \{x \in \{w\}\ .\ wins\ w\ p\ x\} = \{\}$
    by *blast*
  have *union*:
    $\{x \in A - \{w\}\ .\ wins\ w\ p\ x\} \cup \{x \in \{w\}\ .\ wins\ w\ p\ x\} =$
      $\{x \in A\ .\ wins\ w\ p\ x\}$
    using *2*
    by *blast*
  have *finiteness1*: *finite* $\{x \in A - \{w\}\ .\ wins\ w\ p\ x\}$
    using *condorcet-winner.simps winner*
    by *fastforce*
  have *finiteness2*: *finite* $\{x \in \{w\}\ .\ wins\ w\ p\ x\}$
    by *simp*
  from *finiteness1 finiteness2 disjunct card-Un-disjoint*
  have

card ({*x* ∈ *A* − {*w*} . *wins w p x*} ∪ {*x* ∈ {*w*} . *wins w p x*}) =
card {*x* ∈ *A* − {*w*} . *wins w p x*} + *card* {*x* ∈ {*w*} . *wins w p x*}
by *blast*
with *union*
have *card* {*x* ∈ *A* . *wins w p x*} =
card {*x* ∈ *A* − {*w*} . *wins w p x*} + *card* {*x* ∈ {*w*} . *wins w p x*}
by *simp*
with *amount1 amount2*
show *?thesis*
by *linarith*
qed

lemma *cond-winner-imp-loss-count*:
assumes *winner*: *condorcet-winner A p w*
shows *card* {*y* ∈ *A* . *wins y p w*} = *0*
using *Collect-empty-eq card-eq-0-iff condorcet-winner.simps*
insert-Diff insert-iff wins-antisym winner
by (*metis* (*no-types*, *lifting*))

lemma *cond-winner-imp-copeland-score*:
assumes *winner*: *condorcet-winner A p w*
shows *copeland-score w A p* = *card A* −*1*
unfolding *copeland-score.simps*
proof −
show
card {*y* ∈ *A*. *wins w p y*} − *card* {*y* ∈ *A*. *wins y p w*} =
card A − *1*
using *cond-winner-imp-loss-count*
cond-winner-imp-win-count winner
proof −
have *f1*: *card* {*a* ∈ *A*. *wins w p a*} = *card A* − *1*
using *cond-winner-imp-win-count winner*
by *simp*
have *f2*: *card* {*a* ∈ *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 *?thesis*
using *f2 f1*
by *simp*
qed
qed

lemma *non-cond-winner-imp-win-count*:
assumes

winner: *condorcet-winner A p w* **and**
loser: $l \neq w$ **and**
l-in-A: $l \in A$
shows *card* $\{y \in A$. *wins l p y*$\}$ <= *card* $A - 2$
proof −
from *winner loser l-in-A*
have *wins w p l*
by *simp*
hence *0*: ¬ *wins l p w*
by (*simp add*: *wins-antisym*)
have *1*: ¬ *wins l p l*
by (*simp add*: *wins-irreflex*)
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 *condorcet-winner.simps finite-Diff winner*
by *metis*
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 *condorcet-winner.simps winner*
by *metis*
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* $A - 2$
by (*simp add*: *loser*)
with *5 2*
show *?thesis*
by *simp*
qed

3.8.3 Property

theorem *copeland-score-is-cr*: *condorcet-rating copeland-score*
unfolding *condorcet-rating-def*
proof (*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: $l \neq w$
show
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*$\}$
proof $-$
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*
condorcet-winner.simps 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*
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*
with *0*
show *?thesis*
by *linarith*
qed
qed

theorem *copeland-is-dcc*: *defer-condorcet-consistency copeland*
unfolding *defer-condorcet-consistency-def electoral-module-def*
proof (*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 *electoral-module-def finA max-elim-sound profA*
by *blast*
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*
with *defer-condorcet-consistency-def*
show
copeland A p =
({},
A − *defer copeland A p*,
{*d* ∈ *A*. *condorcet-winner A p d*})
using *Collect-cong cwin-w finA max-cplscore-dcc*
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 Lemma

lemma *non-cond-winner-minimax-score*:
assumes
prof: *profile A p* **and**
winner: *condorcet-winner A p w* **and**
l-in-A: $l \in A$ **and**
l-neq-w: $l \neq w$
shows *minimax-score l A p* $\leq$ *prefer-count p l w*
proof −
let
?set = {*prefer-count p l y* |*y* . $y \in A-\{l\}$} **and**
?lscore = *minimax-score l A p*
have *finite A*
using *prof condorcet-winner.simps winner*
by *metis*
hence *finite* $(A-\{l\})$
using *finite-Diff*
by *simp*
hence *finite*: *finite ?set*
by *simp*
have $w \in A$
using *condorcet-winner.simps winner*
by *metis*
hence *0*: $w \in A-\{l\}$
using *l-neq-w*
by *force*
hence *not-empty*: *?set* $\neq$ {}
by *blast*

have *?lscore* = *Min ?set*
by *simp*
hence *1*: *?lscore* $\in$ *?set* $\wedge$ ($\forall$ *p* $\in$ *?set*. *?lscore* $\leq$ *p*)
using *local.finite not-empty Min-le Min-eq-iff*
by (*metis* (*no-types*, *lifting*))
thus *?thesis*
using *0*
by *auto*
qed

3.9.3 Property

theorem *minimax-score-cond-rating*: *condorcet-rating minimax-score*
proof (*unfold condorcet-rating-def minimax-score.simps prefer-count.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:$l \neq w$
show
Min $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r l))\}$ |
$y.\ y \in A - \{l\}\} <$
Min $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r w))\}$ |
$y.\ y \in A - \{w\}\}$
proof (*rule ccontr*)
assume
$\neg$ *Min* $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r l))\}$ |
$y.\ y \in A - \{l\}\} <$
Min $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r w))\}$ |
$y.\ y \in A - \{w\}\}$
hence
Min $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r l))\}$ |
$y.\ y \in A - \{l\}\} \geq$
Min $\{$*card* $\{i.\ i <$ *length p* $\wedge$ (*let* $r = (p!i)$ *in* $(y \preceq_r w))\}$ |
$y.\ y \in A - \{w\}\}$
by *linarith*
hence *000*:
Min $\{$*prefer-count p l y* $|y\ .\ y \in A-\{l\}\} \geq$
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 ∈ A−{w}} and
      6: n ∈ A−{w}
      by metis
    hence n-in-A: n ∈ A
      using DiffE
      by metis
    from 6
    have n-neq-w: n ≠ w
      by auto
    from winner
    have w-in-A: w ∈ 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 ≥ 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 ≥ 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 condorcet-winner.simps l-in-A-without-w
            wins-antisym winner
      by metis
  qed
qed

theorem minimax-is-dcc: defer-condorcet-consistency minimax
unfolding defer-condorcet-consistency-def electoral-module-def
proof (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*)
with *defer-condorcet-consistency-def*
have
max-eliminator minimax-score A p =
({},
A − *defer* (*max-eliminator minimax-score*) *A p*,
{$a \in A$. *condorcet-winner A p a*})
using *cwin-w finA*
by (*metis* (*no-types*))
thus
minimax A p =
({},
A − *defer minimax A p*,
{$d \in A$. *condorcet-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*]:
  assumes *order*: *linear-order r*
  shows *rejects 0* (*drop-module 0 r*)
  unfolding *rejects-def*
proof (*safe*)
  show *electoral-module* (*drop-module 0 r*)
    using *order*
    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 *f1*: *connex UNIV r*
    using *assms lin-ord-imp-connex*
    by *auto*
  obtain *aa* :: $('a \Rightarrow bool) \Rightarrow 'a$ **where**
    f2:
    $\forall p.$ (*Collect* $p = \{\} \longrightarrow (\forall a. \neg\ p\ a)) \wedge$

```
        (Collect p ≠ {} ⟶ p (aa p))
    by moura
  have f3: ∀ a. (a::'a) ∉ {}
    using empty-iff
    by simp
  have connex:
    connex A (limit A r)
    using f1 limit-presv-connex subset-UNIV
    by metis
  have
    ∀ A a. A ≠ {} ∨ (a::'a) ∉ A
    by simp
  hence f4:
    ∀ a Aa.
      ¬ connex Aa (limit A r) ∨ a ∉ Aa ∨ a ∉ A ∨
        ¬ card (above (limit A r) a) ≤ 0
    using above-connex above-presv-limit card-eq-0-iff
          finite-A finite-subset le-0-eq order
    by (metis (no-types))
  have {a ∈ A. card(above (limit A r) a) ≤ 0} = {}
    using connex f4
    by auto
  hence card {a ∈ A. card(above (limit A r) a) ≤ 0} = 0
    using card.empty
    by (metis (full-types))
  thus card (reject (drop-module 0 r) A p) = 0
    by simp
qed


theorem drop-two-mod-rej-two[simp]:
  assumes order: linear-order r
  shows rejects 2 (drop-module 2 r)
proof −
  have rej-drop-eq-def-pass:
    reject (drop-module 2 r) = defer (pass-module 2 r)
    by simp
  thus ?thesis
  proof −
    obtain
      AA :: ('a Electoral-Module) ⇒ nat ⇒ 'a set and
      rrs :: ('a Electoral-Module) ⇒ nat ⇒ 'a Profile where
      ∀ x0 x1. (∃ v2 v3. (x1 ≤ card v2 ∧ finite-profile v2 v3) ∧
          card (reject x0 v2 v3) ≠ x1) =
              ((x1 ≤ card (AA x0 x1) ∧
                finite-profile (AA x0 x1) (rrs x0 x1)) ∧
                card (reject x0 (AA x0 x1) (rrs x0 x1)) ≠ x1)
      by moura
    hence
```

```
    ∀ n f. (¬ rejects n f ∨ electoral-module f ∧
        (∀ A rs. (¬ n ≤ card A ∨ infinite A ∨ ¬ profile A rs) ∨
            card (reject f A rs) = n)) ∧
        (rejects n f ∨ ¬ electoral-module f ∨ (n ≤ card (AA f n) ∧
            finite-profile (AA f n) (rrs f n)) ∧
            card (reject f (AA f n) (rrs f n)) ≠ n)
    using rejects-def
    by force
  hence f1:
    ∀ n f. (¬ rejects n f ∨ electoral-module f ∧
      (∀ A rs. ¬ n ≤ card A ∨ infinite A ∨ ¬ profile A rs ∨
          card (reject f A rs) = n)) ∧
      (rejects n f ∨ ¬ electoral-module f ∨ n ≤ card (AA f n) ∧
          finite (AA f n) ∧ profile (AA f n) (rrs f n) ∧
          card (reject f (AA f n) (rrs f n)) ≠ n)
    by presburger
  have
    ¬ 2 ≤ card (AA (drop-module 2 r) 2) ∨
      infinite (AA (drop-module 2 r) 2) ∨
      ¬ profile (AA (drop-module 2 r) 2) (rrs (drop-module 2 r) 2) ∨
      card (reject (drop-module 2 r) (AA (drop-module 2 r) 2)
          (rrs (drop-module 2 r) 2)) = 2
    using rej-drop-eq-def-pass defers-def order
          pass-two-mod-def-two
    by (metis (no-types))
  thus ?thesis
    using f1 drop-mod-sound order
    by blast
 qed
qed


theorem drop-pass-disj-compat[simp]:
 assumes order: linear-order r
 shows disjoint-compatibility (drop-module n r) (pass-module n r)
 unfolding disjoint-compatibility-def
proof (safe)
 show electoral-module (drop-module n r)
  using order
  by simp
next
 show electoral-module (pass-module n r)
  using order
  by simp
next
 fix
  S :: 'a set
 assume
  fin: finite S
```

```
  obtain
    p :: 'a Profile
    where finite-profile S p
    using empty-iff empty-set fin profile-set
    by metis
  show
    ∃ A ⊆ S.
      (∀ a ∈ A. indep-of-alt (drop-module n r) S a ∧
        (∀ p. finite-profile S p ⟶
          a ∈ reject (drop-module n r) S p)) ∧
      (∀ a ∈ S−A. indep-of-alt (pass-module n r) S a ∧
        (∀ p. finite-profile S p ⟶
          a ∈ reject (pass-module n r) S p))
  proof
    have same-A:
      ∀ p q. (finite-profile S p ∧ finite-profile S q) ⟶
        reject (drop-module n r) S p =
          reject (drop-module n r) S q
      by auto
    let ?A = reject (drop-module n r) S p
    have ?A ⊆ S
      by auto
    moreover have
      (∀ a ∈ ?A. indep-of-alt (drop-module n r) S a)
      using order
      by (simp add: indep-of-alt-def)
    moreover have
      ∀ a ∈ ?A. ∀ p. finite-profile S p ⟶
        a ∈ reject (drop-module n r) S p
      by auto
    moreover have
      (∀ a ∈ S−?A. indep-of-alt (pass-module n r) S a)
      using order
      by (simp add: indep-of-alt-def)
    moreover have
      ∀ a ∈ S−?A. ∀ p. finite-profile S p ⟶
        a ∈ reject (pass-module n r) S p
      by auto
    ultimately show
      ?A ⊆ S ∧
        (∀ a ∈ ?A. indep-of-alt (drop-module n r) S a ∧
          (∀ p. finite-profile S p ⟶
            a ∈ reject (drop-module n r) S p)) ∧
        (∀ a ∈ S−?A. indep-of-alt (pass-module n r) S a ∧
          (∀ p. finite-profile S p ⟶
            a ∈ reject (pass-module n r) S 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 $\Rightarrow$ *'a Electoral-Module* (-$\downarrow$ *50*) **where**
  m$\downarrow$ == *revision-composition m*

4.2.2 Soundness

theorem *rev-comp-sound*[*simp*]:
  assumes *module*: *electoral-module m*
  shows *electoral-module* (*revision-composition m*)
proof $-$
  from *module* **have** $\forall$ *A p. finite-profile A p* $\longrightarrow$ *elect m A p* $\subseteq$ *A*
    using *elect-in-alts*
    by *auto*
  hence $\forall$ *A p. finite-profile A p* $\longrightarrow$ (*A* $-$ *elect m A p*) $\cup$ *elect m A p* = *A*
    by *blast*
  hence *unity*:
    $\forall$ *A p. finite-profile A p* $\longrightarrow$
      set-equals-partition A (*revision-composition m A p*)
    by *simp*
  have $\forall$ *A p. finite-profile A p* $\longrightarrow$ (*A* $-$ *elect m A p*) $\cap$ *elect m A p* = {}
    by *blast*
  hence *disjoint*:
    $\forall$ *A p. finite-profile A p* $\longrightarrow$ *disjoint3* (*revision-composition m A p*)
    by *simp*
  from *unity disjoint* **show** *?thesis*
    by (*simp add*: *electoral-modI*)
qed

4.2.3 Composition Rules

theorem *rev-comp-non-electing*[*simp*]:
  assumes *electoral-module m*
  shows *non-electing* (*m*↓)
  by (*simp add*: *assms non-electing-def*)

theorem *rev-comp-non-blocking*[*simp*]:
  assumes *electing m*
  shows *non-blocking* (*m*↓)
  unfolding *non-blocking-def*
proof (*safe*, *simp-all*)
  show *electoral-module* (*m*↓)
    using *assms electing-def rev-comp-sound*
    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 electing-def empty-iff*
        Diff-disjoint Int-absorb2 elect-in-alts
    by (*metis* (*no-types*, *lifting*))
  show *False*
    using *non-elect assms electing-def empty-iff fin-A*
        non-electing-def prof-A x-in-A
    by (*metis* (*no-types*, *lifting*))
qed

theorem *rev-comp-def-inv-mono*[*simp*]:
  assumes *invariant-monotonicity m*
  shows *defer-invariant-monotonicity* (*m*↓)
proof −
  have $\forall A\ p\ q\ w.$ ($w \in$ *defer* (*m*↓) *A p* $\wedge$ *lifted A p q w*) $\longrightarrow$
        (*defer* (*m*↓) *A q* $=$ *defer* (*m*↓) *A p* $\vee$ *defer* (*m*↓) *A q* $= \{w\}$)
    using *assms*
    by (*simp add*: *invariant-monotonicity-def*)
  moreover have *electoral-module* (*m*↓)
    using *assms rev-comp-sound invariant-monotonicity-def*
    by *auto*
  moreover have *non-electing* (*m*↓)

using *assms rev-comp-non-electing invariant-monotonicity-def*
by *auto*
ultimately have *electoral-module* $(m\downarrow) \wedge$ *non-electing* $(m\downarrow) \wedge$
$(\forall A\ p\ q\ w.\ (w \in \mathit{defer}\ (m\downarrow)\ A\ p \wedge \mathit{lifted}\ A\ p\ q\ w) \longrightarrow$
$(\mathit{defer}\ (m\downarrow)\ A\ q = \mathit{defer}\ (m\downarrow)\ A\ p \vee \mathit{defer}\ (m\downarrow)\ A\ q = \{w\}))$
by *blast*
thus *?thesis*
using *defer-invariant-monotonicity-def*
by (*simp add*: *defer-invariant-monotonicity-def*)
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*:
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*:
    ∀ *prof f a A*.
      (*profile A prof* ∧ *finite A* ∧ *electoral-module f* ∧
        (*a*::′*a*) ∈ *defer f A prof*) ⟶
          a ∈ *A*
    using *UnCI result-presv-alts*
    by (*metis* (*mono-tags*))
  from *module-m f-prof*
  have *disjoint-m*: *disjoint3* (*m A p*)
    using *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*))
    using *electoral-module-def well-formed.simps*
    by *metis*
  have *disj-n*:
    elect m A p ∩ *reject m A p* = {} ∧
      elect m A p ∩ *defer m A p* = {} ∧
      reject m A p ∩ *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*) ⊆ *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* ∩ *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*) ⊆ *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*:
$(elect\ m\ A\ p \cap defer\ n\ ?new\text{-}A\ ?new\text{-}p) = \{\}$
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*:
$(reject\ m\ A\ p \cap reject\ n\ ?new\text{-}A\ ?new\text{-}p) = \{\}$
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 $\forall\, A\ Aa.\ \neg\ (A{::}'a\ set) \subseteq Aa \vee A = A \cap Aa$
by *blast*
with *disjoint-m disjoint-n def-presv-fin-prof f-prof*
module-m module-n elec-n-in-def-m
have *3*:
$(reject\ m\ A\ p \cap elect\ n\ ?new\text{-}A\ ?new\text{-}p) = \{\}$
using *disj-n*
by *blast*
have
$(elect\ m\ A\ p \cup elect\ n\ ?new\text{-}A\ ?new\text{-}p) \cap$
$(reject\ m\ A\ p \cup reject\ n\ ?new\text{-}A\ ?new\text{-}p) = \{\}$
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 ∈ {}
    using disj-n
    by simp
next
  fix x :: 'a
  assume
    elec-x: x ∈ elect m A p and
    rej-lim-x:
    x ∈ reject n (defer m A p)
      (limit-profile (defer m A p) p)
  from elec-x rej-lim-x
  show x ∈ {}
    using 0
    by blast
next
  fix x :: 'a
  assume
    elec-lim-x:
    x ∈ elect n (defer m A p) (limit-profile (defer m A p) p) and
    rej-x: x ∈ reject m A p
  from elec-lim-x rej-x
  show x ∈ {}
    using 3
    by blast
next
  fix x :: 'a
  assume
    elec-lim-x:
    x ∈ elect n (defer m A p) (limit-profile (defer m A p) p) and
    rej-lim-x:
    x ∈ reject n (defer m A p) (limit-profile (defer m A p) p)
  from elec-lim-x rej-lim-x
  show x ∈ {}
    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 ∪ elect n ?new-A ?new-p) ∩
        (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 ∪ reject n ?new-A ?new-p) ∩
        (defer n ?new-A ?new-p) = {}
```

proof (*safe*)
  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 ∈ *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 ∈ defer n (defer m A p) (limit-profile (defer m A p) p) and
    x-in-rej:
    x ∈ reject n (defer m A p) (limit-profile (defer m A p) p)
  from x-in-def x-in-rej
  show x ∈ {}
    using fin-def module-n prof-def-lim reject-not-elec-or-def
    by fastforce
 qed
 ultimately have
   disjoint3 (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
   using sequential-composition.simps
   by metis
qed

lemma seq-comp-presv-alts:
 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 ▷ 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)
   by (simp add: electoral-module-def)
 with module-m f-prof have 0:
   elect m A p ∪ reject m A p ∪ ?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)
   using electoral-module-def well-formed.simps
   by metis
 with module-m module-n f-prof have 1:
   elect n ?new-A ?new-p ∪
      reject n ?new-A ?new-p ∪
      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*
using *sequential-composition.simps*
by *metis*
qed

4.3.2 Soundness

theorem *seq-comp-sound*[*simp*]:
assumes *module-m*: *electoral-module m* **and**
module-n: *electoral-module n*
shows *electoral-module* ($m \triangleright n$)
unfolding *electoral-module-def*
proof (*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* (($m \triangleright n$) *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 Lemmata

lemma *seq-comp-dec-only-def*:
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 ($m \triangleright n$) *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 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 ▷ n) A p) = snd (m A p)
    using rej-empty empty-defer module-n prof-no-alt
    by auto
qed

lemma seq-comp-def-then-elect:
  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 electing-def non-electing-def seq-comp-sound*
by *metis*
next
assume *assm*: $A \neq \{\}$
from *n-electing-m f-prof*
have *ele*: *elect* m A $p = \{\}$
using *non-electing-def*
by *auto*
from *assm def-one-m f-prof finite*
have *def-card*:
card (*defer* m A p) = *1*
by (*simp add*: *Suc-leI card-gt-0-iff defers-def*)
with *n-electing-m f-prof*
have *def*:
$\exists\, a \in A.$ *defer* m A $p = \{a\}$
using *card-1-singletonE defer-in-alts non-electing-def singletonI subsetCE*
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 defers-def f-prof reject-not-elec-or-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 non-electing-def reject-not-elec-or-def*
by *metis*
hence
$\exists\, a \in A.$ *elect* $(m \triangleright n)$ A p =
elect n $\{a\}$ (*limit-profile* $\{a\}$ p)
using *prod.sel*(*1*) *prod.sel*(*2*) *sequential-composition.simps sup-bot.left-neutral*
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 non-electing-def prod.sel sequential-composition.simps def-presv-fin-prof sup-bot.left-neutral*
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 non-electing-def sequential-composition.simps sup-bot.left-neutral*
by *metis*
qed

lemma *seq-comp-def-card-bounded*:
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 sequential-composition.simps def-presv-fin-prof snd-conv*
by *metis*

lemma *seq-comp-def-set-bounded*:
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 sequential-composition.simps def-presv-fin-prof*
by *metis*

lemma *seq-comp-defers-def-set*:
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows
$defer\ (m \triangleright n)\ A\ p =$
$defer\ n\ (defer\ m\ A\ p)\ (limit\text{-}profile\ (defer\ m\ A\ p)\ p)$
using *sequential-composition.simps snd-conv*
by *metis*

lemma *seq-comp-def-then-elect-elec-set*:
assumes
module-m: *electoral-module m* **and**
module-n: *electoral-module n* **and**
f-prof: *finite-profile A p*
shows
$elect\ (m \triangleright n)\ A\ p =$
$elect\ n\ (defer\ m\ A\ p)\ (limit\text{-}profile\ (defer\ m\ A\ p)\ p) \cup$
$(elect\ m\ A\ p)$
using *Un-commute fst-conv sequential-composition.simps*

by *metis*

lemma *seq-comp-elim-one-red-def-set*:
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*
sequential-composition.simps def-presv-fin-prof
single-elim-imp-red-def-set snd-conv
by *metis*

lemma *seq-comp-def-set-sound*:
assumes
electoral-module m **and**
electoral-module n **and**
finite-profile A p
shows *defer* ($m \triangleright n$) *A p* ⊆ *defer m A p*
proof −
have ∀ *A p*. *finite-profile A p* ⟶ *well-formed A* (*n A p*)
using *assms*(*2*) *electoral-module-def*
by *auto*
hence
finite-profile (*defer m A p*) (*limit-profile* (*defer m A p*) *p*) ⟶
well-formed (*defer m A p*)
(*n* (*defer m A p*) (*limit-profile* (*defer m A p*) *p*))
by *simp*
hence
well-formed (*defer m A p*) (*n* (*defer m A p*)
(*limit-profile* (*defer m A p*) *p*))
using *assms*(*1*) *assms*(*3*) *def-presv-fin-prof*
by *metis*
thus *?thesis*
using *assms seq-comp-def-set-bounded*
by *blast*
qed

lemma *seq-comp-def-set-trans*:
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*(*1*) *assms*(*2*)

    assms(*3*) *in-mono seq-comp-defers-def-set*
  by (*metis* (*no-types*, *hide-lams*))

4.3.4 Composition Rules

theorem *seq-comp-presv-non-blocking*[*simp*]:
  assumes
    non-blocking-m: *non-blocking m* **and**
    non-blocking-n: *non-blocking n*
  shows *non-blocking* $(m \triangleright n)$
proof −
  fix
    A :: $'a$ *set* **and**
    p :: $'a$ *Profile*
  let *?input-sound* = ((*A*::*'a set*) $\neq$ {} $\wedge$ *finite-profile A p*)
  from *non-blocking-m* **have**
    ?input-sound $\longrightarrow$ *reject m A p* $\neq A$
    by (*simp add*: *non-blocking-def*)
  with *non-blocking-m* **have** *0*:
    ?input-sound $\longrightarrow A -$ *reject m A p* $\neq$ {}
    using *Diff-eq-empty-iff non-blocking-def*
        reject-in-alts subset-antisym
    by *metis*
  from *non-blocking-m* **have**
    ?input-sound $\longrightarrow$ *well-formed A* (*m A p*)
    by (*simp add*: *electoral-module-def non-blocking-def*)
  hence
    ?input-sound $\longrightarrow$
      elect m A p $\cup$ *defer m A p* $= A -$ *reject m A p*
    using *non-blocking-def non-blocking-m elec-and-def-not-rej*
    by *metis*
  with *0* **have**
    ?input-sound $\longrightarrow$ *elect m A p* $\cup$ *defer m A p* $\neq$ {}
    by *auto*
  hence *?input-sound* $\longrightarrow$ (*elect m A p* $\neq$ {} $\vee$ *defer m A p* $\neq$ {})
    by *simp*
  with *non-blocking-m non-blocking-n*
  show *?thesis*
  proof (*unfold non-blocking-def*)
    assume
      emod-reject-m:
      electoral-module m $\wedge$
        ($\forall$ *A p*. $A \neq$ {} $\wedge$ *finite-profile A p* $\longrightarrow$
          reject m A p $\neq A$) **and**
      emod-reject-n:
      electoral-module n $\wedge$
        ($\forall$ *A p*. $A \neq$ {} $\wedge$ *finite-profile A p* $\longrightarrow$
          reject n A p $\neq A$)
    show

```
  electoral-module (m ▷ n) ∧
    (∀ A p.
      A ≠ {} ∧ finite-profile A p ⟶
        reject (m ▷ n) A p ≠ A)
proof (safe)
  show electoral-module (m ▷ 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 ▷ n) A p = A and
    x-in-A: x ∈ 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 ▷ n) A p =
      elect n (defer m A p) (limit-profile (defer m A p) p) ∪
        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 ▷ 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 ▷ n) A p ∪ defer (m ▷ n) A p = A − reject (m ▷ 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) ∪
      elect m A p ∪
      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*
using *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

theorem *seq-comp-presv-non-electing*[*simp*]:
assumes
m-elect: *non-electing m* **and**
n-elect: *non-electing n*
shows *non-electing* (*m* ▷ *n*)
unfolding *non-electing-def*
proof (*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

theorem *seq-comp-electing*[*simp*]:
  assumes *def-one-m1*: *defers 1 m1* **and**
          electing-m2: *electing m2*
  shows *electing* (*m1* ▷ *m2*)
proof −
  have
    ∀ *A p*. (*card A* ≥ *1* ∧ *finite-profile A p*) ⟶
      card (*defer m1 A p*) = *1*
    using *def-one-m1 defers-def*
    by *blast*
  hence *def-m1-not-empty*:
    ∀ *A p*. (*A* ≠ {} ∧ *finite-profile A p*) ⟶
      defer m1 A p ≠ {}
    using *One-nat-def Suc-leI card-eq-0-iff*
          card-gt-0-iff zero-neq-one
    by *metis*
  thus *?thesis*
    using *Un-empty def-one-m1 defers-def electing-def*
          electing-m2 seq-comp-def-then-elect-elec-set
          seq-comp-sound def-presv-fin-prof
  proof −
    obtain
      f-set ::
      (′*a set* ⇒ ′*a Profile* ⇒ ′*a Result*) ⇒ ′*a set* **and**
      f-prof ::
      (′*a set* ⇒ ′*a Profile* ⇒ ′*a Result*) ⇒ ′*a Profile* **where**
      f-mod:
      ∀ *f*.
        (¬ *electing f* ∨ *electoral-module f* ∧
          (∀ *A prof*.
            (*A* ≠ {} ∧ *finite A* ∧ *profile A prof*) ⟶
              elect f A prof ≠ {})) ∧
        (*electing f* ∨ ¬ *electoral-module f* ∨ *f-set f* ≠ {} ∧ *finite* (*f-set f*) ∧
          profile (*f-set f*) (*f-prof f*) ∧ *elect f* (*f-set f*) (*f-prof f*) = {})
      unfolding *electing-def*
      by *moura*
    hence *f-elect*:
      electoral-module m2 ∧
        (∀ *A prof*. (*A* ≠ {} ∧ *finite A* ∧ *profile A prof*) ⟶ *elect m2 A prof* ≠ {})
      using *electing-m2*
      by *metis*
    have *def-card-one*:
      electoral-module m1 ∧
        (∀ *A prof*.
          (*1* ≤ *card A* ∧ *finite A* ∧ *profile A prof*) ⟶
            card (*defer m1 A prof*) = *1*)

using *def-one-m1 defers-def*
by *blast*
hence *electoral-module* $(m1 \triangleright m2)$
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*:
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 $?new\text{-}Ap = defer\ m\ A\ p$
let $?new\text{-}Aq = defer\ m\ A\ q$
let $?new\text{-}p = limit\text{-}profile\ ?new\text{-}Ap\ p$
let $?new\text{-}q = limit\text{-}profile\ ?new\text{-}Aq\ q$
from *monotone-m monotone-n* **have** *modules*:
electoral-module m $\wedge$ *electoral-module n*
unfolding *defer-lift-invariance-def*
by *simp*
hence $finite\text{-}profile\ A\ p \longrightarrow defer\ (m \triangleright n)\ A\ p \subseteq defer\ m\ A\ p$
using *seq-comp-def-set-bounded*
by *metis*
moreover have *profile-p*: $lifted\ A\ p\ q\ a \longrightarrow finite\text{-}profile\ A\ p$
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 defer-lift-invariance-def def-and-lifted modules profile-p seq-comp-def-set-trans*
by *metis*
hence *new-A-eq*: $?new\text{-}Ap = ?new\text{-}Aq$
by *presburger*
have *defer-eq*:
$defer\ (m \triangleright n)\ A\ p = defer\ n\ ?new\text{-}Ap\ ?new\text{-}p$
using *sequential-composition.simps snd-conv*
by *metis*
hence *mono-n*:
$n\ ?new\text{-}Ap\ ?new\text{-}p = n\ ?new\text{-}Aq\ ?new\text{-}q$

proof *cases*
  assume *lifted ?new-Ap ?new-p ?new-q a*
  thus *?thesis*
    using *defer-eq mono-m monotone-n*
        defer-lift-invariance-def def-and-lifted
    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*
    by (*simp add*: *lifted-def*)
  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*: $a \in$ *?new-Ap*
    by *blast*
  moreover from *def-and-lifted*
  have *eql-lengths*:
    length ?new-p = *length ?new-q*
    by (*simp add*: *lifted-def*)
  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 lifted-def*
    by (*metis* (*no-types*, *lifting*))
  from *def-and-lifted modules* **have**
    ∀ *i*. (*0* ≤ *i* ∧ *i* < *length ?new-p*) ⟶
        (*Preference-Relation.lifted A* (*p*!*i*) (*q*!*i*) *a* ∨ (*p*!*i*) = (*q*!*i*))
    using *defer-in-alts Profile.lifted-def limit-prof-presv-size*
    by *metis*
  with *def-and-lifted modules mono-m* **have**
    ∀ *i*. (*0* ≤ *i* ∧ *i* < *length ?new-p*) ⟶
        (*Preference-Relation.lifted ?new-Ap* (*?new-p*!*i*) (*?new-q*!*i*) *a* ∨
         (*?new-p*!*i*) = (*?new-q*!*i*))
    using *limit-lifted-imp-eq-or-lifted defer-in-alts*
        Profile.lifted-def limit-prof-presv-size
        limit-profile.simps nth-map
    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
    using sequential-composition.simps
    by (metis (full-types))
qed


theorem seq-comp-presv-def-lift-inv[simp]:
  assumes
    monotone-m: defer-lift-invariance m and
    monotone-n: defer-lift-invariance n
  shows defer-lift-invariance (m ▷ n)
  using monotone-m monotone-n def-lift-inv-seq-comp-help
        seq-comp-sound defer-lift-invariance-def
  by (metis (full-types))


theorem seq-comp-def-one[simp]:
  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 ▷ n)
  unfolding defers-def
proof (safe)
  have electoral-mod-m: electoral-module m
    using non-electing-m
    by (simp add: non-electing-def)
  have electoral-mod-n: electoral-module n
    using def-1-n
    by (simp add: defers-def)
  show electoral-module (m ▷ n)
    using electoral-mod-m electoral-mod-n
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    pos-card: 1 ≤ card A and
    fin-A: finite A and
    prof-A: profile A p
  from pos-card have
    A ≠ {}
```

```
    by auto
  with fin-A prof-A have m-non-blocking:
    reject m A p ≠ A
    using non-blocking-m non-blocking-def
    by metis
  hence
    ∃ a. a ∈ A ∧ a ∉ reject m A p
    using pos-card non-electing-def non-electing-m
          reject-in-alts subset-antisym subset-iff
          fin-A prof-A subsetI
    by metis
  hence defer m A p ≠ {}
    using electoral-mod-defer-elem empty-iff pos-card
          non-electing-def non-electing-m fin-A prof-A
    by (metis (no-types))
  hence defer-non-empty:
    card (defer m A p) ≥ 1
    using One-nat-def Suc-leI card-gt-0-iff pos-card fin-A prof-A
          non-blocking-def non-blocking-m def-presv-fin-prof
    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 defers-def fin-A non-blocking-def non-blocking-m
          prof-A seq-comp-defers-def-set
    by (metis (no-types, hide-lams))
  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))
    using 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-def
          non-blocking-m def-presv-fin-prof
    by metis
  thus card (defer (m ▷ n) A p) = 1
    using defer-fun
    by auto
qed


theorem disj-compat-seq[simp]:
```

```
  assumes
    compatible: disjoint-compatibility m n and
    module-m2: electoral-module m2
  shows disjoint-compatibility (m ▷ m2) n
  unfolding disjoint-compatibility-def
proof (safe)
  show electoral-module (m ▷ m2)
    using compatible disjoint-compatibility-def
          module-m2 seq-comp-sound
    by metis
next
  show electoral-module n
    using compatible disjoint-compatibility-def
    by metis
next
  fix
    S :: 'a set
  assume
    fin-S: finite S
  have modules:
    electoral-module (m ▷ m2) ∧ electoral-module n
    using compatible disjoint-compatibility-def
          module-m2 seq-comp-sound
    by metis
  obtain A where A:
    A ⊆ S ∧
      (∀ a ∈ A. indep-of-alt m S a ∧
        (∀ p. finite-profile S p ⟶ a ∈ reject m S p)) ∧
      (∀ a ∈ S−A. indep-of-alt n S a ∧
        (∀ p. finite-profile S p ⟶ a ∈ reject n S p))
    using compatible disjoint-compatibility-def fin-S
    by (metis (no-types, lifting))
  show
    ∃ A ⊆ S.
      (∀ a ∈ A. indep-of-alt (m ▷ m2) S a ∧
        (∀ p. finite-profile S p ⟶ a ∈ reject (m ▷ m2) S p)) ∧
      (∀ a ∈ S−A. indep-of-alt n S a ∧
        (∀ p. finite-profile S p ⟶ a ∈ reject n S p))
  proof
    have
      ∀ a p q.
        a ∈ A ∧ equiv-prof-except-a S p q a ⟶
          (m ▷ m2) S p = (m ▷ m2) 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 indep-of-alt-def*
by *metis*
from *a b* **have** *profiles*:
finite-profile S p $\wedge$ *finite-profile S q*
using *equiv-prof-except-a-def*
by *fastforce*
hence (*defer m S p*) $\subseteq$ *S*
using *compatible defer-in-alts disjoint-compatibility-def*
by *blast*
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*
disjoint-compatibility-def eq-def profiles
negl-diff-imp-eq-limit-prof
by (*metis* (*no-types*, *lifting*))
with *eq-def* **have**
m2 (*defer m S p*) (*limit-profile* (*defer m S p*) *p*) =
m2 (*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 indep-of-alt-def*
by *metis*
ultimately show
($m \triangleright m2$) *S p* = ($m \triangleright m2$) *S q*
using *sequential-composition.simps*
by (*metis* (*full-types*))
qed
moreover have
$\forall a \in A.\ \forall p.$ *finite-profile S p* $\longrightarrow a \in$ *reject* ($m \triangleright m2$) *S p*
using *A UnI1 prod.sel sequential-composition.simps*
by *metis*
ultimately show
$A \subseteq S \wedge$
($\forall a \in A.$ *indep-of-alt* ($m \triangleright m2$) *S a* $\wedge$
($\forall p.$ *finite-profile S p* $\longrightarrow a \in$ *reject* ($m \triangleright m2$) *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

theorem *seq-comp-mono*[*simp*]:

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* $(m \triangleright n)$
unfolding *monotonicity-def*
proof (*safe*)
have *electoral-mod-m*: *electoral-module m*
using *non-ele-m*
by (*simp add*: *non-electing-def*)
have *electoral-mod-n*: *electoral-module n*
using *electing-n*
by (*simp add*: *electing-def*)
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 lifted-def*
by *metis*
thus $w \in$ *elect* $(m \triangleright n)$ *A q*
using *seq-comp-def-then-elect defer-lift-invariance-def*
elect-w-in-p lifted-w def-monotone-m non-ele-m
def-one-m electing-n
by *metis*
qed

theorem *def-inv-mono-imp-def-lift-inv*[*simp*]:
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* $(m \triangleright n)$
unfolding *defer-lift-invariance-def*
proof (*safe*)
have *electoral-mod-m*: *electoral-module m*

using *defer-invariant-monotonicity-def*
strong-def-mon-m
by *auto*
have *electoral-mod-n*: *electoral-module n*
using *defers-1 defers-def*
by *auto*
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*
by (*simp add*: *defer-invariant-monotonicity-def*)
have *electoral-mod-m*: *electoral-module m*
using *strong-def-mon-m defer-invariant-monotonicity-def*
by *auto*
have *electoral-mod-n*: *electoral-module n*
using *defers-1 defers-def*
by *auto*
have *finite-profile-q*: *finite-profile* A q
using *lifted-a*
by (*simp add*: *Profile.lifted-def*)
have *finite-profile-p*: *profile* A p
using *lifted-a*
by (*simp add*: *Profile.lifted-def*)
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*
defer-invariant-monotonicity-def lifted-a
seq-comp-def-set-trans finite-profile-p
finite-profile-q
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*, *hide-lams*))
ultimately have

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

```
      by simp
    have defers (Suc 0) n
      using defers-1
      by auto
    hence defer-card-one:
      electoral-module n ∧
        (∀ A prof.
          (Suc 0 ≤ card A ∧ finite A ∧ profile A prof) ⟶
            card (defer n A prof) = Suc 0)
      by (simp add: defers-def)
    hence emod-mn: electoral-module (m ▷ n)
      using emod-n-then-mn
      by blast
    have nat-diff:
      ∀ (i::nat) j. i ≤ j ⟶ i − j = 0
      by auto
    have nat-comp:
      ∀ (i::nat) j k.
        i ≤ j ∧ j ≤ k ∨
          j ≤ i ∧ i ≤ k ∨
          i ≤ k ∧ k ≤ j ∨
          k ≤ j ∧ j ≤ i ∨
          j ≤ k ∧ k ≤ i ∨
          k ≤ i ∧ i ≤ j
      using le-cases3
      by linarith
    have fin-diff-card:
      ∀ A a.
        (finite A ∧ (a::'a) ∈ A) ⟶
          card (A − {a}) = card A − 1
      using card-Diff-singleton
      by metis
    with fin-defer defer-card-one min-card
    have card (defer (m ▷ 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 ?thesis
      by force
  qed
next
  show a ∈ defer (m ▷ 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 defers-def electoral-mod-m*
finite-profile-p def-presv-fin-prof
finite-profile-q
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 ∈ defer n (defer m A q)
        (limit-profile (defer m A p) q)
      using still-lifted a-in-def-p
          defer-monotonicity-def
          defer-monotone-n electoral-mod-m
          same-alternatives
          def-presv-fin-prof finite-profile-q
      by metis
    have card (defer m A q) ≥ 1
      using card-le-1-p same-alternatives
      by auto
    hence
      card (defer n (defer m A q)
        (limit-profile (defer m A q) q)) = 1
      using defers-1 defers-def electoral-mod-m
          finite-profile-q def-presv-fin-prof
      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]:
  assumes
    mod-m: electoral-module m and
    mod-n: electoral-module n and
    agg-a: aggregator a
  shows electoral-module (m ‖_a n)
  unfolding electoral-module-def
proof (safe)
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    fin-A: finite A and
    prof-A: profile A p
  have wf-quant:
    ∀ f. aggregator f =
      (∀ a-set elec1 rej1 def1 elec2 rej2 def2.
        (¬ well-formed (a-set::'a set) (elec1, rej2, def1) ∨
          ¬ well-formed a-set (rej1, def2, elec2)) ∨
        well-formed a-set
          (f a-set (elec1, rej2, def1) (rej1, def2, elec2)))
    unfolding aggregator-def
    by blast
```

have *wf-imp*:
$\forall$ *e-mod a-set prof*.
(*electoral-module e-mod* $\wedge$ *finite* (*a-set*::$'a$ *set*) $\wedge$
profile a-set prof) $\longrightarrow$
well-formed a-set (*e-mod a-set prof*)
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* (($m \parallel_a n$) *A p*)
by *simp*
qed

4.4.3 Composition Rule

theorem *conserv-agg-presv-non-electing*[*simp*]:
assumes
non-electing-m: *non-electing m* **and**
non-electing-n: *non-electing n* **and**
conservative: *agg-conservative a*
shows *non-electing* ($m \parallel_a n$)
unfolding *non-electing-def*
proof (*safe*)
have *emod-m*: *electoral-module m*
using *non-electing-m*
by (*simp add*: *non-electing-def*)
have *emod-n*: *electoral-module n*
using *non-electing-n*
by (*simp add*: *non-electing-def*)
have *agg-a*: *aggregator a*
using *conservative*
by (*simp add*: *agg-conservative-def*)
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**
x :: $'a$
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
x-wins: $x \in$ *elect* ($m \parallel_a n$) *A p*
have *emod-m*: *electoral-module m*
using *non-electing-m*

```
    by (simp add: non-electing-def)
  have emod-n: electoral-module n
    using non-electing-n
    by (simp add: non-electing-def)
  have
    ∀ x0 x1 x2 x3 x4 x5 x6 x7.
      (well-formed (x6::'a set) (x5, x1, x3) ∧ well-formed x6 (x4, x0, x2) ⟶
        elect-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x5 ∪ x4 ∧
          reject-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x1 ∪ x0 ∧
          defer-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x3 ∪ x2) =
            ((¬ well-formed x6 (x5, x1, x3) ∨ ¬ well-formed x6 (x4, x0, x2)) ∨
              elect-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x5 ∪ x4 ∧
                reject-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x1 ∪ x0 ∧
                defer-r (x7 x6 (x5, x1, x3) (x4, x0, x2)) ⊆ x3 ∪ x2)
    by linarith
  hence
    ∀ f. agg-conservative f =
      (aggregator f ∧
        (∀ A Aa Ab Ac Ad Ae Af. (¬ well-formed (A::'a set) (Aa, Ae, Ac) ∨
            ¬ well-formed A (Ab, Af, Ad)) ∨
          elect-r (f A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Aa ∪ Ab ∧
            reject-r (f A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Ae ∪ Af ∧
            defer-r (f A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Ac ∪ Ad))
    by (simp add: agg-conservative-def)
  hence
    aggregator a ∧
      (∀ A Aa Ab Ac Ad Ae Af. ¬ well-formed A (Aa, Ae, Ac) ∨
          ¬ well-formed A (Ab, Af, Ad) ∨
          elect-r (a A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Aa ∪ Ab ∧
            reject-r (a A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Ae ∪ Af ∧
            defer-r (a A (Aa, Ae, Ac) (Ab, Af, Ad)) ⊆ Ac ∪ Ad)
    using conservative
    by presburger
  hence
    let c = (a A (m A p) (n A p)) in
      (elect-r c ⊆ ((elect m A p) ∪ (elect n A p)))
    using emod-m emod-n fin-A par-comp-result-sound
          prod.collapse prof-A
    by metis
  hence x ∈ ((elect m A p) ∪ (elect n A p))
    using x-wins
    by auto
  thus x ∈ {}
    using sup-bot-right non-electing-def fin-A
          non-electing-m non-electing-n prof-A
    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*:
  assumes
    not-term: $\neg t\ (acc\ A\ p)$ **and**
    subset: $defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p$ **and**
    not-inf: $\neg infinite\ (defer\ acc\ A\ p)$
  shows
    $((acc \triangleright m,\ m,\ t,\ A,\ p),\ (acc,\ m,\ t,\ A,\ p)) \in$
      $measure\ (\lambda(acc,\ m,\ t,\ A,\ p).\ card\ (defer\ acc\ A\ p))$
  using *assms psubset-card-mono*
  by *auto*

function *loop-comp-helper* ::
    $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Electoral-Module* $\Rightarrow$
      $'a$ *Termination-Condition* $\Rightarrow$ $'a$ *Electoral-Module* **where**
  $t\ (acc\ A\ p) \lor \neg((defer\ (acc \triangleright m)\ A\ p) \subset (defer\ acc\ A\ p)) \lor$
    $infinite\ (defer\ acc\ A\ p) \Longrightarrow$
      $loop\text{-}comp\text{-}helper\ acc\ m\ t\ A\ p = acc\ A\ p\ |$
  $\neg(t\ (acc\ A\ p) \lor \neg((defer\ (acc \triangleright m)\ A\ p) \subset (defer\ acc\ A\ p)) \lor$
    $infinite\ (defer\ acc\ A\ p)) \Longrightarrow$
      $loop\text{-}comp\text{-}helper\ acc\ m\ t\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ A\ p$
proof −
  fix
    P :: *bool* **and**
    x :: $('a\ Electoral\text{-}Module) \times ('a\ Electoral\text{-}Module) \times$
      $('a\ Termination\text{-}Condition) \times\ 'a\ set \times\ 'a\ Profile$
  assume
    a1: $\bigwedge t\ acc\ A\ p\ m.$
      $[\![t\ (acc\ A\ p) \lor \neg\ defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p \lor$

infinite (*defer acc A p*);
x = (*acc*, *m*, *t*, *A*, *p*)⟧ ⟹ *P* **and**
a2: ⋀*t acc A p m*.
⟦¬ (*t* (*acc A p*) ∨ ¬ *defer* (*acc* ⊳ *m*) *A p* ⊂ *defer acc A p* ∨
infinite (*defer acc A p*));
x = (*acc*, *m*, *t*, *A*, *p*)⟧ ⟹ *P*
have ∃*f A p p2 g*. (*g*, *f*, *p*, *A*, *p2*) = *x*
using *prod-cases5*
by *metis*
then show *P*
using *a2 a1*
by (*metis* (*no-types*))
next
show
⋀*t acc A p m ta acca Aa pa ma*.
t (*acc A p*) ∨ ¬ *defer* (*acc* ⊳ *m*) *A p* ⊂ *defer acc A p* ∨
infinite (*defer acc A p*) ⟹
ta (*acca Aa pa*) ∨ ¬ *defer* (*acca* ⊳ *ma*) *Aa pa* ⊂ *defer acca Aa pa* ∨
infinite (*defer acca Aa pa*) ⟹
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*) ⟹
acc A p = *acca Aa pa*
by *fastforce*
next
show
⋀*t acc A p m ta acca Aa pa ma*.
t (*acc A p*) ∨ ¬ *defer* (*acc* ⊳ *m*) *A p* ⊂ *defer acc A p* ∨
infinite (*defer acc A p*) ⟹
¬ (*ta* (*acca Aa pa*) ∨ ¬ *defer* (*acca* ⊳ *ma*) *Aa pa* ⊂ *defer acca Aa pa* ∨
infinite (*defer acca Aa pa*)) ⟹
(*acc*, *m*, *t*, *A*, *p*) = (*acca*, *ma*, *ta*, *Aa*, *pa*) ⟹
acc A p = *loop-comp-helper-sumC* (*acca* ⊳ *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*) ∨ ¬ *defer* (*acc* ⊳ *m*) *A p* ⊂ *defer acc A p* ∨
infinite (*defer acc A p*) **and**
a2: ¬ (*ta* (*acca Aa pa*) ∨ ¬ *defer* (*acca* ⊳ *ma*) *Aa pa* ⊂ *defer acca Aa pa* ∨
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 ▷ ma, ma, ta, Aa, pa)
    by auto
qed
next
  show
    ⋀t acc A p m ta acca Aa pa ma.
      ¬ (t (acc A p) ∨ ¬ defer (acc ▷ m) A p ⊂ defer acc A p ∨
        infinite (defer acc A p)) ⟹
         ¬ (ta (acca Aa pa) ∨ ¬ defer (acca ▷ ma) Aa pa ⊂ defer acca Aa pa ∨
          infinite (defer acca Aa pa)) ⟹
           (acc, m, t, A, p) = (acca, ma, ta, Aa, pa) ⟹
             loop-comp-helper-sumC (acc ▷ m, m, t, A, p) =
               loop-comp-helper-sumC (acca ▷ ma, ma, ta, Aa, pa)
    by force
qed
termination
proof −
  have f0:
    ∃ r. wf r ∧
        (∀ 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 ((λra. ∀ f (A::'a set) prof pa g.
            ∃ prof2 pb p-rel pc pd h (B::'a set) prof3 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
              (prof2::((
                ('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 prof3) ∧
         defer (h ▷ i) B prof3 ⊂ defer h B prof3 ∧
         ¬ pe (h B prof3) ∧
         ((h ▷ i, i, pe, B, prof3), h, i, pe, B, prof3) ∉ 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 ▷ g, g, p, A, prof), f, g, p, A, prof) ∈ p-rel)
    using f0
    by presburger
  thus ?thesis
    using termination
    by metis
qed

lemma loop-comp-code-helper[code]:
  loop-comp-helper acc m t A p =
    (if (t (acc A p) ∨ ¬((defer (acc ▷ m) A p) ⊂ (defer acc A p)) ∨
      infinite (defer acc A p))
    then (acc A p) else (loop-comp-helper (acc ▷ m) m t A p))
  by simp

function loop-composition ::
    'a Electoral-Module ⇒ 'a Termination-Condition ⇒
        'a Electoral-Module where
  t ({}, {}, A) ⟹
    loop-composition m t A p = defer-module A p |
  ¬(t ({}, {}, A)) ⟹
    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 ⇒ *'a Termination-Condition* ⇒ *'a Electoral-Module*
(- $\circlearrowleft_{-}$ *50*) **where**
$m \circlearrowleft_t$ ≡ *loop-composition m t*

lemma *loop-comp-code*[*code*]:
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*:
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 *electoral-module-def less.prems profile*
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*:
assumes *m-module*: *electoral-module m*

shows *electoral-module* $(m \circlearrowleft_t)$
using *def-mod-sound electoral-module-def loop-composition.simps*(*1*)
loop-composition.simps(*2*) *loop-comp-helper-imp-partit m-module*
by *metis*

lemma *loop-comp-helper-imp-no-def-incr*:
assumes
module-m: *electoral-module m* **and**
profile: *finite-profile A p*
shows
(*electoral-module acc* $\wedge$ *n* = *card* (*defer acc A p*)) $\Longrightarrow$
defer (*loop-comp-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 Aa. infinite* (*A*::*'a set*) $\vee$ $\neg$ *Aa* $\subset$ *A* $\vee$ *card Aa* < *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-comp-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-comp-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 Lemmata

lemma *loop-comp-helper-def-lift-inv-helper*:
assumes
monotone-m: *defer-lift-invariance m* **and**
f-prof: *finite-profile A p*
shows
(*defer-lift-invariance acc* $\wedge$ *n* = *card* (*defer acc A p*)) $\longrightarrow$
($\forall$ *q a*.
(*a* $\in$ (*defer* (*loop-comp-helper acc m t*) *A p*) $\wedge$
lifted A p q a) $\longrightarrow$
(*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))
    by (simp add: defer-lift-invariance-def)
  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 defer-lift-invariance-def seq-comp-def-set-trans
    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 defer-lift-invariance-def monotone-m
        by auto
      have emod-acc: electoral-module acc
        using defer-lift-invariance-def dli-acc
        by blast
      have acc-eq-pq: acc A q = acc A p
        using a-in-def-acc defer-lift-invariance-def dli-acc lifted-A
```

by (*metis* (*full-types*))
with *emod-acc emod-m*
have
finite (*defer acc A p*) $\longrightarrow$
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* $\triangleright$ *m*) $\wedge$ *defer-lift-invariance acc*) $\longrightarrow$
($\forall$ *q a*. (*a* $\in$ (*defer* (*loop-comp-helper acc m t*) *A p*) $\wedge$
lifted A p q a) $\longrightarrow$
(*loop-comp-helper acc m t*) *A p* =
(*loop-comp-helper acc m t*) *A q*)
using *defer-lift-invariance-def*
by *metis*
thus *?thesis*
using *monotone-m seq-comp-presv-def-lift-inv*
by *blast*
next
assume *card-changed*:
$\neg$ (*card* (*defer* (*acc* $\triangleright$ *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*
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*)))
using *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) ⟶
    (∀ q a. (a ∈ (defer (acc ▷ m) A p) ∧ lifted A p q a) ⟶
      ¬ t (acc A q))
  using monotone-m f-prof defer-lift-invariance-def seq-comp-def-set-trans
  by metis
from card-changed defer-card-comp defer-card-acc
have dli-card-def:
  (defer-lift-invariance (acc ▷ m) ∧ defer-lift-invariance (acc)) ⟶
    (∀ q a. (a ∈ (defer (acc ▷ m) A p) ∧ Profile.lifted A p q a) ⟶
      card (defer (acc ▷ m) A q) ≠ (card (defer acc A q)))
proof –
  have
    ∀ f.
      (defer-lift-invariance f ∨
        (∃ A prof prof2 (a::'a).
          f A prof ≠ f A prof2 ∧
            Profile.lifted A prof prof2 a ∧
            a ∈ defer f A prof) ∨ ¬ electoral-module f) ∧
            ((∀ A p1 p2 b. f A p1 = f A p2 ∨ ¬ Profile.lifted A p1 p2 b ∨
              b ∉ defer f A p1) ∧
            electoral-module f ∨ ¬ defer-lift-invariance f)
    using defer-lift-invariance-def
    by blast
  thus ?thesis
    using card-changed monotone-m f-prof seq-comp-def-set-trans
    by (metis (no-types, hide-lams))
qed
hence dli-def-subset:
  defer-lift-invariance (acc ▷ m) ∧ defer-lift-invariance (acc) ⟶
    (∀ q a. (a ∈ (defer (acc ▷ m) A p) ∧ lifted A p q a) ⟶
      defer (acc ▷ m) A q ⊂ defer acc A q)
proof –
  {
    fix
      alt :: 'a and
      prof :: 'a Profile
    have
      (¬ defer-lift-invariance (acc ▷ m) ∨ ¬ defer-lift-invariance acc) ∨
        (alt ∉ defer (acc ▷ m) A p ∨ ¬ lifted A p prof alt) ∨
        defer (acc ▷ m) A prof ⊂ 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*:
$(defer\text{-}lift\text{-}invariance\ (acc \triangleright m) \wedge defer\text{-}lift\text{-}invariance\ (acc)) \longrightarrow$
$(\forall\, q\ a.\ (a \in (defer\ (acc \triangleright m)\ A\ p) \wedge lifted\ A\ p\ q\ a) \longrightarrow$
$loop\text{-}comp\text{-}helper\ acc\ m\ t\ A\ q =$
$loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ A\ q)$
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: $defer\text{-}lift\text{-}invariance\ acc$ **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 defer-lift-invariance-def*
by *auto*
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
$loop\text{-}comp\text{-}helper\ acc\ m\ t\ A\ q =$
$loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ A\ q$
using *loop-comp-code-helper*
by *metis*
qed
have *rec-step-p*:
$electoral\text{-}module\ acc \longrightarrow$
$loop\text{-}comp\text{-}helper\ acc\ m\ t\ A\ p = loop\text{-}comp\text{-}helper\ (acc \triangleright m)\ m\ t\ A\ p$
proof (*safe*)
assume *emod-acc*: *electoral-module acc*
have *emod-implies-defer-subset*:
$electoral\text{-}module\ m \longrightarrow defer\ (acc \triangleright m)\ A\ p \subseteq defer\ acc\ A\ p$
using *emod-acc f-prof seq-comp-def-set-bounded*
by *blast*
have *card-ineq*: $card\ (defer\ (acc \triangleright m)\ A\ p) < card\ (defer\ acc\ A\ p)$
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
          using 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)
      with defer-lift-invariance-def
      show ?thesis
        using loop-comp-helper.simps(1)
        by metis
    qed
  qed
qed

lemma loop-comp-helper-def-lift-inv:
```

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-inv2*:
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*:
assumes *lifted A p q a*
shows *finite-profile A p*
using *assms Profile.lifted-def*
by *fastforce*

lemma *loop-comp-helper-presv-def-lift-inv*:
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 −
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 prof prof2 a*. *f A prof* = *f A prof2* $\vee$
$\neg$ *Profile.lifted A prof prof2 a* $\vee$
a $\notin$ *defer f A prof*) $\wedge$
electoral-module f $\vee$ $\neg$ *defer-lift-invariance f*)
using *defer-lift-invariance-def*

```
    by blast
  thus ?thesis
    using electoral-module-def lifted-imp-fin-prof
          loop-comp-helper-def-lift-inv loop-comp-helper-imp-partit
          monotone-acc monotone-m
    by (metis (full-types))
qed

lemma loop-comp-presv-non-electing-helper:
  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-comp-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:
      (⋀y acc'. y < card (defer acc A p) ⟹
        y = card (defer acc' A p) ⟹ non-electing acc' ⟹
          elect (loop-comp-helper acc' m t) A p = {}) and
      acc-non-elect:
      non-electing acc and
      x-in-acc-elect:
      x ∈ elect (loop-comp-helper acc m t) A p
    have
      ∀(f::'a set ⇒ 'a Profile ⇒ 'a Result) g.
        (non-electing f ∧ non-electing g) ⟶
          non-electing (f ▷ g)
      by simp
    hence seq-acc-m-non-elect: non-electing (acc ▷ m)
      using acc-non-elect non-electing-m
      by blast
    have ∀ A B. (finite (A::'a set) ∧ B ⊂ A) ⟶ card B < card A
      using psubset-card-mono
      by metis
    hence card-ineq:
      ∀ A B. (finite (A::'a set) ∧ B ⊂ A) ⟶ 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:
      ∀ 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
$\bigwedge f.$
$(finite\ (defer\ acc\ A\ p) \wedge defer\ f\ A\ p \subset defer\ acc\ A\ p \wedge non\text{-}electing\ f) \longrightarrow$
$elect\ (loop\text{-}comp\text{-}helper\ f\ m\ t)\ A\ p = \{\}$
using *card-n-no-elect psubset-card-mono*
by *metis*
hence *f0*:
$(\neg\ t\ (acc\ A\ p) \wedge defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p \wedge$
$finite\ (defer\ acc\ A\ p)) \wedge$
$\neg\ t\ (acc\ A\ p) \longrightarrow$
$elect\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p = \{\}$
using *loop-comp-code-helper seq-acc-m-non-elect*
by (*metis* (*no-types*))
obtain *set-func* :: $'a\ set \Rightarrow 'a$ **where**
$\forall A.\ (A = \{\} \longrightarrow (\forall a.\ a \notin A)) \wedge (A \neq \{\} \longrightarrow set\text{-}func\ A \in A)$
using *all-not-in-conv*
by (*metis* (*no-types*))
thus $x \in \{\}$
using *loop-comp-code-helper no-elect-acc x-in-acc-elect f0*
by (*metis* (*no-types*))
qed
qed

lemma *loop-comp-helper-iter-elim-def-n-helper*:
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall\ r.\ ((t\ r) \longleftrightarrow (card\ (defer\text{-}r\ r) = x))$ **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 $card\ (defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p) = x$
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 *subset*:
$(card\ (defer\ acc\ A\ p) > 1 \wedge finite\text{-}profile\ A\ p \wedge electoral\text{-}module\ acc) \longrightarrow$
$defer\ (acc \triangleright m)\ A\ p \subset defer\ acc\ A\ p$
using *seq-comp-elim-one-red-def-set single-elimination*
by *blast*
hence *step-reduces-defer-set*:
$(card\ (defer\ acc\ A\ p) > 1 \wedge finite\text{-}profile\ A\ p \wedge non\text{-}electing\ acc) \longrightarrow$

defer (*acc* ⊳ *m*) *A p* ⊂ *defer acc A p*
using *non-electing-def*
by *auto*
thus *?case*
proof *cases*
assume *term-satisfied*: *t* (*acc A p*)
have *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*
thus *?case*
by *blast*
next
assume *term-not-satisfied*: ¬(*t* (*acc A p*))
hence *card-not-eq-x*: *card* (*defer acc A p*) ≠ *x*
by (*simp add*: *terminate-if-n-left*)
have *rec-step*:
(*card* (*defer acc A p*) > *1* ∧ *finite-profile A p* ∧ *non-electing acc*) ⟶
loop-comp-helper acc m t A p =
loop-comp-helper (*acc* ⊳ *m*) *m t A p*
using *loop-comp-helper.simps*(*2*) *non-electing-def def-presv-fin-prof*
step-reduces-defer-set term-not-satisfied
by *metis*
thus *?case*
proof *cases*
assume *card-too-small*:
card (*defer acc A p*) < *x*
thus *?thesis*
using *not-le card-too-small less.prems*(*1*) *less.prems*(*4*) *not-le*
by (*metis* (*no-types*))
next
assume *old-card-at-least-x*: ¬(*card* (*defer acc A p*) < *x*)
obtain *i* **where** *i-is-new-card*: *i* = *card* (*defer* (*acc* ⊳ *m*) *A p*)
by *blast*
with *card-not-eq-x*
have *card-too-big*:
card (*defer acc A p*) > *x*
using *nat-neq-iff old-card-at-least-x*
by *blast*
hence *enough-leftover*:
card (*defer acc A p*) > *1*
using *x-greater-zero*
by *auto*
have *electoral-module acc* ⟶ (*defer acc A p*) ⊆ *A*
by (*simp add*: *defer-in-alts f-prof*)
hence *step-profile*:
electoral-module acc ⟶
finite-profile (*defer acc A p*)
(*limit-profile* (*defer acc A p*) *p*)
using *f-prof limit-profile-sound*

```
  by auto
hence
  electoral-module acc ⟶
      card (defer m (defer acc A p)
        (limit-profile (defer acc A p) p)) =
          card (defer acc A p) − 1
  using non-electing-m single-elimination
        single-elim-decr-def-card2 enough-leftover
  by blast
hence electoral-module acc ⟶ i = card (defer acc A p) − 1
  using sequential-composition.simps snd-conv i-is-new-card
  by metis
hence electoral-module acc ⟶ i ≥ x
  using card-too-big
  by linarith
hence new-card-still-big-enough: electoral-module acc ⟶ x ≤ i
  by blast
have
  electoral-module acc ∧ electoral-module m ⟶
      defer (acc ▷ m) A p ⊆ defer acc A p
  using enough-leftover f-prof subset
  by blast
hence
  electoral-module acc ∧ electoral-module m ⟶
      i ≤ card (defer acc A p)
  using card-mono i-is-new-card step-profile
  by blast
hence i-geq-x:
  electoral-module acc ∧ electoral-module m ⟶ (i = x ∨ i > x)
  using nat-less-le new-card-still-big-enough
  by blast
thus ?thesis
proof cases
  assume new-card-greater-x: electoral-module acc ⟶ i > x
  hence electoral-module acc ⟶ 1 < card (defer (acc ▷ m) A p)
    using x-greater-zero i-is-new-card
    by linarith
  moreover have new-card-still-big-enough2:
    electoral-module acc ⟶ x ≤ i
    using i-is-new-card new-card-still-big-enough
    by blast
  moreover have
    n = card (defer acc A p) ⟶
        (electoral-module acc ⟶ i < n)
    using subset step-profile enough-leftover f-prof psubset-card-mono
          i-is-new-card
    by blast
  moreover have
    electoral-module acc ⟶
```

electoral-module (*acc* ▷ *m*)
using *non-electing-def non-electing-m seq-comp-sound*
by *blast*
moreover have *non-electing-new*:
non-electing acc ⟶ *non-electing* (*acc* ▷ *m*)
by (*simp add*: *non-electing-m*)
ultimately have *card-x*:
(*n* = *card* (*defer acc A p*) ∧ *non-electing acc* ∧
electoral-module acc) ⟶
card (*defer* (*loop-comp-helper* (*acc* ▷ *m*) *m t*) *A p*) = *x*
using *less.hyps i-is-new-card new-card-greater-x*
by *blast*
have *f1*: *loop-comp-helper acc m t A p* = *loop-comp-helper* (*acc* ▷ *m*) *m t A*
p
using *enough-leftover f-prof less.prems*(*3*) *rec-step*
by *blast*
have *electoral-module acc*
using *less.prems*(*3*) *non-electing-def*
by *blast*
thus *?thesis*
using *f1 card-x less.prems*(*3*) *less.prems*(*4*)
by *presburger*
next
assume *i-not-gt-x*: ¬(*electoral-module acc* ⟶ *i* > *x*)
hence *electoral-module acc* ∧ *electoral-module m* ⟶ *i* = *x*
using *i-geq-x*
by *blast*
hence *electoral-module acc* ∧ *electoral-module m* ⟶ *t* ((*acc* ▷ *m*) *A p*)
using *i-is-new-card terminate-if-n-left*
by *blast*
hence
electoral-module acc ∧ *electoral-module m* ⟶
card (*defer-r* (*loop-comp-helper* (*acc* ▷ *m*) *m t A p*)) = *x*
using *loop-comp-helper.simps*(*1*) *terminate-if-n-left*
by *metis*
thus *?thesis*
using *i-not-gt-x dual-order.strict-iff-order i-is-new-card*
loop-comp-helper.simps(*1*) *new-card-still-big-enough*
f-prof rec-step terminate-if-n-left
enough-leftover less.prems(*3*)
by *metis*
qed
qed
qed
qed

lemma *loop-comp-helper-iter-elim-def-n*:
assumes
non-electing-m: *non-electing m* **and**

single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall\ r.\ ((t\ r) \longleftrightarrow (card\ (defer\text{-}r\ r) = x))$ **and**
x-greater-zero: $x > 0$ **and**
f-prof: *finite-profile A p* **and**
acc-defers-enough: $card\ (defer\ acc\ A\ p) \geq x$ **and**
non-electing-acc: *non-electing acc*
shows $card\ (defer\ (loop\text{-}comp\text{-}helper\ acc\ m\ t)\ A\ p) = x$
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*:
assumes
non-electing-m: *non-electing m* **and**
single-elimination: *eliminates 1 m* **and**
terminate-if-n-left: $\forall\ r.\ ((t\ r) \longleftrightarrow (card\ (defer\text{-}r\ r) = x))$ **and**
x-greater-zero: $x > 0$ **and**
f-prof: *finite-profile A p* **and**
enough-alternatives: $card\ A \geq x$
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*: $\neg\ card\ A = x$
thus *?thesis*
proof *cases*
assume $card\ A < x$
thus *?thesis*
using *enough-alternatives not-le*
by *blast*
next
assume $\neg card\ A < x$
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-card2 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 $(m\ \circlearrowleft_t)\ A\ p = (loop\text{-}comp\text{-}helper\ m\ m\ t)\ A\ p$
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

theorem *loop-comp-presv-def-lift-inv*[*simp*]:
  assumes *monotone-m*: *defer-lift-invariance m*
  shows *defer-lift-invariance* (m ↻$_t$)
  unfolding *defer-lift-invariance-def*
proof (*safe*)
  from *monotone-m*
  have *electoral-module m*
    unfolding *defer-lift-invariance-def*
    by *simp*
  thus *electoral-module* (m ↻$_t$)
    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 ↻$_t$) A p **and**
    lifted-a: *Profile.lifted* A p q a
  have *defer-lift-loop*:
    $\forall$ p q a. ($a \in$ (*defer* (m ↻$_t$) A p) $\wedge$ *lifted* A p q a) $\longrightarrow$
      (m ↻$_t$) A p = (m ↻$_t$) A q
    using *monotone-m lifted-imp-fin-prof loop-comp-helper-def-lift-inv2*
      loop-composition.simps defer-module.simps
    by (*metis* (*full-types*))
  show (m ↻$_t$) A p = (m ↻$_t$) A q
    using *a-in-loop-defer lifted-a defer-lift-loop*
    by *metis*
qed

theorem *loop-comp-presv-non-electing*[*simp*]:
  assumes *non-electing-m*: *non-electing m*
  shows *non-electing* (m ↻$_t$)
  unfolding *non-electing-def*
proof (*safe*, *simp-all*)
  show *electoral-module* (m ↻$_t$)
    using *loop-comp-sound non-electing-def non-electing-m*

```
    by metis
next
  fix
    A :: 'a set and
    p :: 'a Profile and
    x :: 'a
  assume
    fin-A: finite A and
    prof-A: profile A p and
    x-elect: x ∈ elect (m ↺_t) A p
  show False
  using def-mod-non-electing loop-comp-presv-non-electing-helper
        non-electing-m empty-iff fin-A loop-comp-code
        non-electing-def prof-A x-elect
  by metis
qed

theorem iter-elim-def-n[simp]:
  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) = n)) and
    x-greater-zero: n > 0
  shows defers n (m ↺_t)
proof –
  have
    ∀ A p. finite-profile A p ∧ card A ≥ n ⟶
        card (defer (m ↺_t) A p) = n
    using iter-elim-def-n-helper non-electing-m single-elimination
          terminate-if-n-left x-greater-zero
    by blast
  moreover have electoral-module (m ↺_t)
    using loop-comp-sound eliminates-def single-elimination
    by blast
  thus ?thesis
    by (simp add: calculation defers-def)
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* ⇒
    'a Electoral-Module ⇒ *'a Electoral-Module* **where**
  maximum-parallel-composition m n =
   (*let a* = *max-aggregator in* ($m \parallel_a n$))

abbreviation *max-parallel* :: *'a Electoral-Module* ⇒ *'a Electoral-Module* ⇒
    '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*:
  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 Lemmata

lemma *max-agg-eq-result*:
  assumes
   module-m: *electoral-module m* **and**
   module-n: *electoral-module n* **and**
   f-prof: *finite-profile A p* **and**
   in-A: $x \in A$
  shows
   mod-contains-result ($m \parallel_\uparrow n$) *m A p x* ∨
    mod-contains-result ($m \parallel_\uparrow n$) *n A p x*
proof *cases*
  assume *a1*: $x \in$ *elect* ($m \parallel_\uparrow n$) *A p*
   have *mod-contains-inst*:
    ∀ *p-mod q-mod a-set prof a*.
     mod-contains-result p-mod q-mod a-set prof (*a*::*'a*) =
      (*electoral-module p-mod* ∧ *electoral-module q-mod* ∧

finite a-set ∧ *profile a-set prof* ∧ *a* ∈ *a-set* ∧
(*a* ∉ *elect p-mod a-set prof* ∨ *a* ∈ *elect q-mod a-set prof*) ∧
(*a* ∉ *reject p-mod a-set prof* ∨ *a* ∈ *reject q-mod a-set prof*) ∧
(*a* ∉ *defer p-mod a-set prof* ∨ *a* ∈ *defer q-mod a-set prof*))
by (*simp add*: *mod-contains-result-def*)
have *module-mn*: *electoral-module* ($m \parallel_\uparrow n$)
by (*simp add*: *module-m module-n*)
have *not-defer-mn*: *x* ∉ *defer* ($m \parallel_\uparrow n$) *A p*
using *module-mn IntI a1 empty-iff f-prof result-disj*
by (*metis* (*no-types*))
have *not-reject-mn*: *x* ∉ *reject* ($m \parallel_\uparrow n$) *A p*
using *module-mn IntI a1 empty-iff f-prof result-disj*
by (*metis* (*no-types*))
from *a1* **have**
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
x ∈ *e1* ∪ *e2*
by *auto*
hence *union-mn*: *x* ∈ (*elect m A p*) ∪ (*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-a1*: *x* ∉ *elect* ($m \parallel_\uparrow n$) *A p*
thus *?thesis*
proof *cases*
assume *x-in-def*: *x* ∈ *defer* ($m \parallel_\uparrow n$) *A p*
thus *?thesis*
proof (*safe*)
assume *not-mod-cont-mn*:
¬ *mod-contains-result* ($m \parallel_\uparrow n$) *n A p x*
have *par-emod*:
∀ *f g*.
(*electoral-module* (*f*::'*a set* ⇒ '*a Profile* ⇒ '*a Result*) ∧
electoral-module g) ⟶
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*:
∀ (*a*::'*a*) *A B*. (*a* ∈ *A* ∩ *B*) = (*a* ∈ *A* ∧ *a* ∈ *B*)

by *blast*
obtain
s-func :: ($'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Result*) $\Rightarrow$ $'a$ *set* **and**
p-func :: ($'a$ *set* $\Rightarrow$ $'a$ *Profile* $\Rightarrow$ $'a$ *Result*) $\Rightarrow$ $'a$ *Profile* **where**
well-f:
$\forall f$.
($\neg$ *electoral-module* f $\vee$
($\forall$ A *prof*. (*finite* A $\wedge$ *profile* A *prof*) $\longrightarrow$ *well-formed* A (f A *prof*))) $\wedge$
(*electoral-module* f $\vee$ *finite* (*s-func* f) $\wedge$ *profile* (*s-func* f) (*p-func* f) $\wedge$
$\neg$ *well-formed* (*s-func* f) (f (*s-func* f) (*p-func* f)))
using *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 *a-exists*: $\forall (a::'a)$. $a \notin \{\}$
by *blast*
have *e-mod-par*:
electoral-module (m $\|_\uparrow$ n)
using *par-emod module-m module-n*
by *blast*
hence *electoral-module* (m $\|_m$*ax-aggregator* n)
by *simp*
hence *result-disj-max*:
elect (m $\|_m$*ax-aggregator* n) A p $\cap$ *reject* (m $\|_m$*ax-aggregator* n) A p $= \{\}$
$\wedge$
elect (m $\|_m$*ax-aggregator* n) A p $\cap$ *defer* (m $\|_m$*ax-aggregator* n) A p $= \{\}$
$\wedge$
reject (m $\|_m$*ax-aggregator* n) A p $\cap$ *defer* (m $\|_m$*ax-aggregator* n) A p $= \{\}$
using *f-prof result-disj*
by *metis*
have *x-not-elect*:
$x \notin$ *elect* (m $\|_m$*ax-aggregator* n) A p
using *result-disj-max x-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*:
$\forall (A::'a$ *set*) p q.
elect-r (*max-aggregator* A p q) $=$ *elect-r* p $\cup$ *elect-r* q
by *force*
have

  $x \notin$ *elect* (m $\|_m$*ax-aggregator* n) A p
  using *x-not-elect*
  by *blast*
with *max-pq*
have $x \notin$ *elect* m A p $\cup$ *elect* n A p
  by (*simp add*: *max-pq*)
hence *x-not-elect-mn*:
  $x \notin$ *elect* m A p $\wedge$ $x \notin$ *elect* n A p
  by *blast*
have *x-not-mpar-rej*:
  $x \notin$ *reject* (m $\|_m$*ax-aggregator* n) A p
  using *result-disj-max x-in-def*
  by *fastforce*
hence *x-not-par-rej*:
  $x \notin$ *reject* (m $\|_\uparrow$ n) A p
  by *auto*
have *mod-cont-res-fg*:
  $\forall$ *f g A prof* (a::$'a$).
    mod-contains-result f g A prof a =
      (*electoral-module f* $\wedge$ *electoral-module g* $\wedge$
        finite A $\wedge$ *profile A prof* $\wedge$ $a \in A$ $\wedge$
          ($a \notin$ *elect f A prof* $\vee$ $a \in$ *elect g A prof*) $\wedge$
          ($a \notin$ *reject f A prof* $\vee$ $a \in$ *reject g A prof*) $\wedge$
          ($a \notin$ *defer f A prof* $\vee$ $a \in$ *defer g A 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*:
  $\forall$ *r2 r1 e2 e1 d2 d1 A*.
    well-formed A (*e1*, *r1*, *d1*) $\wedge$ *well-formed A* (*e2*, *r2*, *d2*) $\longrightarrow$
      reject-r (*max-aggregator A* (*e1*, *r1*, *d1*) (*e2*, *r2*, *d2*)) = *r1* $\cap$ *r2*
  using *max-agg-rej-set*
  by *metis*
have *e-mod-disj*:
  $\forall$ *f* (A::$'a$ *set*) *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 *blast*
hence *e-mod-disj-n*:
  elect n A p $\cup$ *reject n A p* $\cup$ *defer n A p* = A
  using *f-prof module-n*
  by *metis*
have
  $\forall$ *f g A prof* (a::$'a$).
    mod-contains-result f g A prof a =
      (*electoral-module f* $\wedge$ *electoral-module g* $\wedge$

```
          finite A ∧ profile A prof ∧ a ∈ A ∧
          (a ∉ elect f A prof ∨ a ∈ elect g A prof) ∧
          (a ∉ reject f A prof ∨ a ∈ reject g A prof) ∧
          (a ∉ defer f A prof ∨ a ∈ defer g A prof))
      by (simp add: mod-contains-result-def)
    with e-mod-disj-n
    have x ∈ reject n A p
      using e-mod-par f-prof in-A module-n not-mod-cont-mn
            x-not-elect x-not-elect-mn x-not-mpar-rej
      by auto
    hence x ∉ reject m A p
      using well-f-max max-agg-res result-m result-n
            set-intersect wf-m wf-n x-not-mpar-rej
      by (metis (no-types))
    with max-agg-res
    have
      x ∉ defer (m ∥↑ n) A p ∨ x ∈ defer m A p
        using e-mod-disj f-prof in-A module-m x-not-elect-mn
        by blast
    with x-not-mpar-rej
    show mod-contains-result (m ∥↑ n) m A p x
      using mod-cont-res-fg x-not-par-rej e-mod-par f-prof
            in-A module-m x-not-elect
      by auto
  qed
next
  assume not-a2: x ∉ defer (m ∥↑ 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-a1 not-a2 have a3:
    x ∈ reject (m ∥↑ 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) ⇒
        x ∈ 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
      x ∈ fst (snd (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*
$x \in A - (e1 \cup e2 \cup d1 \cup d2)$
by *simp*
hence
$x \notin$ *elect* $m\ A\ p \cup$ (*defer* $n\ A\ p \cup$ *defer* $m\ A\ p$)
by *force*
thus *?thesis*
using *mod-contains-result-comm mod-contains-result-def Un-iff a3 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*:
assumes
f-prof: *finite-profile* $A\ p$ **and**
module-m: *electoral-module* m **and**
module-n: *electoral-module* n
shows
$x \in$ *reject* $(m \parallel_\uparrow n)\ A\ p \longleftrightarrow$
$(x \in$ *reject* $m\ A\ p \wedge x \in$ *reject* $n\ A\ p)$
proof
assume *a*: $x \in$ *reject* $(m \parallel_\uparrow n)\ A\ p$
hence
case $n\ A\ p$ *of* $(Aa,\ Ab,\ Ac) \Rightarrow$
$x \in$ *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* $(Ac,\ Ad,\ Ae) \Rightarrow$
$x \in$ *reject-r* (*max-aggregator* A
(*elect* $m\ A\ p$, Aa, Ab) $(Ac,\ Ad,\ Ae)$)
by *force*
with *a* **have**
let $(e1,\ r1,\ d1) = m\ A\ p;$
$(e2,\ r2,\ d2) = n\ A\ p$ *in*
$x \in$ *fst* (*snd* (*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*
$x \in A - (e1 \cup e2 \cup d1 \cup d2)$
by *simp*
hence
$x \in A -$ (*elect* $m\ A\ p \cup$ *elect* $n\ A\ p \cup$ *defer* $m\ A\ p \cup$ *defer* $n\ A\ p$)
by *auto*

thus $x \in$ *reject* $m\ A\ p \wedge x \in$ *reject* $n\ A\ p$
  using *Diff-iff Un-iff electoral-mod-defer-elem*
    f-prof module-m module-n
  by *metis*
next
assume *a*: $x \in$ *reject* $m\ A\ p \wedge x \in$ *reject* $n\ A\ p$
hence
  $x \notin$ *elect* $m\ A\ p \wedge x \notin$ *defer* $m\ A\ p\ \wedge$
    $x \notin$ *elect* $n\ A\ p \wedge x \notin$ *defer* $n\ A\ p$
  using *IntI empty-iff module-m module-n f-prof result-disj*
  by *metis*
thus $x \in$ *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-rej1*:
  assumes
    f-prof: *finite-profile* $A\ p$ **and**
    module-m: *electoral-module* m **and**
    module-n: *electoral-module* n **and**
    rejected: $x \in$ *reject* $n\ A\ p$
  shows
    mod-contains-result $m\ (m \parallel_\uparrow n)\ A\ p\ x$
  unfolding *mod-contains-result-def*
proof (*safe*)
  show *electoral-module* m
    using *module-m*
    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 $x \in A$
    using *f-prof module-n reject-in-alts rejected*
    by *auto*
next
  assume

x-in-elect: $x \in$ *elect m A p*
hence *x-not-reject*:
x $\notin$ *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 $\subseteq$ *A*
using *f-prof module-n*
by (*simp add*: *reject-in-alts*)
have *x-in-A*: $x \in A$
using *rej-in-A in-mono rejected*
by *metis*
with *x-in-elect x-not-reject*
show $x \in$ *elect* ($m \parallel_\uparrow n$) *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 $x \in$ *reject m A p*
hence
$x \in$ *reject m A p* $\wedge$ $x \in$ *reject n A p*
using *rejected*
by *simp*
thus $x \in$ *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 *x-in-defer*: $x \in$ *defer m A p*
hence *defer-a*:
$\exists\, a.$ $a \in$ *defer m A p* $\wedge$ $x = a$
by *simp*
then obtain *x-inst* :: *'a* **where**
inst-x: $x =$ *x-inst* $\wedge$ *x-inst* $\in$ *defer m A p*
by *metis*
hence *x-not-rej*:
x $\notin$ *reject m A p*
using *disjoint-iff-not-equal f-prof inst-x 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 *x-in-defer*
have $x \in A$
using *f-prof module-m*
by *blast*

```
  with inst-x x-not-rej
  show x ∈ defer (m ‖↑ 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-rej2:
  assumes
    f-prof: finite-profile A p and
    module-m: electoral-module m and
    module-n: electoral-module n and
    rejected: x ∈ reject n A p
  shows
    mod-contains-result (m ‖↑ n) m A p x
  using mod-contains-result-comm max-agg-rej1
        module-m module-n f-prof rejected
  by metis

lemma max-agg-rej3:
  assumes
    f-prof: finite-profile A p and
    module-m: electoral-module m and
    module-n: electoral-module n and
    rejected: x ∈ reject m A p
  shows
    mod-contains-result n (m ‖↑ n) A p x
  unfolding mod-contains-result-def
proof (safe)
  show electoral-module n
    using module-n
    by simp
next
  show electoral-module (m ‖↑ 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 x ∈ A
```

    using *f-prof in-mono module-m reject-in-alts rejected*
    by (*metis* (*no-types*))
next
  assume $x \in$ *elect n A p*
  thus $x \in$ *elect* ($m \parallel_{\uparrow} n$) *A p*
    using *Un-iff combine-ele-rej-def fst-conv*
          maximum-parallel-composition.simps
          max-aggregator.simps
          parallel-composition.simps
    by (*metis* (*mono-tags*, *lifting*))
next
  assume $x \in$ *reject n A p*
  thus $x \in$ *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 *x-in-def*: $x \in$ *defer n A p*
  have $x \in A$
    using *f-prof max-agg-rej1 mod-contains-result-def module-m rejected*
    by *metis*
  thus $x \in$ *defer* ($m \parallel_{\uparrow} n$) *A p*
    using *x-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-rej4*:
  assumes
    f-prof: *finite-profile A p* **and**
    module-m: *electoral-module m* **and**
    module-n: *electoral-module n* **and**
    rejected: $x \in$ *reject m A p*
  shows
    mod-contains-result ($m \parallel_{\uparrow} n$) *n A p x*
  using *mod-contains-result-comm max-agg-rej3*
        module-m module-n f-prof rejected
  by *metis*

lemma *max-agg-rej-intersect*:
  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*) ∪ (*reject m A p*) ∪ (*defer m A p*) ∧
A = (*elect n A p*) ∪ (*reject n A p*) ∪ (*defer n A p*)
by (*simp add*: *module-m module-n f-prof result-presv-alts*)
hence
A − ((*elect m A p*) ∪ (*defer m A p*)) = (*reject m A p*) ∧
A − ((*elect n A p*) ∪ (*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*) ∪ (*elect n A p*) ∪ (*defer m A p*) ∪ (*defer n A p*)) =
(*reject m A p*) ∩ (*reject n A p*)
by *blast*
hence
let (*e1*, *r1*, *d1*) = *m A p*;
(*e2*, *r2*, *d2*) = *n A p in*
A − (*e1* ∪ *e2* ∪ *d1* ∪ *d2*) = *r1* ∩ *r2*
by *fastforce*
thus *?thesis*
by *auto*
qed

lemma *dcompat-dec-by-one-mod*:
assumes
compatible: *disjoint-compatibility m n* **and**
in-A: *x* ∈ *A*
shows
(∀ *p*. *finite-profile A p* ⟶
mod-contains-result m (*m* $\|_\uparrow$ *n*) *A p x*) ∨
(∀ *p*. *finite-profile A p* ⟶
mod-contains-result n (*m* $\|_\uparrow$ *n*) *A p x*)
using *DiffI compatible disjoint-compatibility-def*
in-A max-agg-rej1 max-agg-rej3
by *metis*

4.6.4 Composition Rules

theorem *conserv-max-agg-presv-non-electing*[*simp*]:
assumes
non-electing-m: *non-electing m* **and**
non-electing-n: *non-electing n*
shows *non-electing* (*m* $\|_\uparrow$ *n*)
using *non-electing-m non-electing-n*
by *simp*

theorem *par-comp-def-lift-inv*[*simp*]:
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 ∥↑ n)
  unfolding defer-lift-invariance-def
proof (safe)
  have electoral-mod-m: electoral-module m
    using monotone-m
    by (simp add: defer-lift-invariance-def)
  have electoral-mod-n: electoral-module n
    using monotone-n
    by (simp add: defer-lift-invariance-def)
  show electoral-module (m ∥↑ n)
    using electoral-mod-m electoral-mod-n
    by simp
next
  fix
    S :: 'a set and
    p :: 'a Profile and
    q :: 'a Profile and
    x :: 'a
  assume
    defer-x: x ∈ defer (m ∥↑ n) S p and
    lifted-x: Profile.lifted S p q x
  hence f-profs: finite-profile S p ∧ finite-profile S q
    by (simp add: lifted-def)
  from compatible
  obtain A::'a set where A:
    A ⊆ S ∧ (∀ x ∈ A. indep-of-alt m S x ∧
      (∀ p. finite-profile S p ⟶ x ∈ reject m S p)) ∧
        (∀ x ∈ S−A. indep-of-alt n S x ∧
      (∀ p. finite-profile S p ⟶ x ∈ reject n S p))
    using disjoint-compatibility-def f-profs
    by (metis (no-types, lifting))
  have
    ∀ x ∈ S. prof-contains-result (m ∥↑ n) S p q x
  proof cases
    assume a0: x ∈ A
    hence x ∈ reject m S p
      using A f-profs
      by blast
    with defer-x
    have defer-n: x ∈ defer n S p
      using compatible disjoint-compatibility-def
            mod-contains-result-def f-profs max-agg-rej4
      by metis
    have
      ∀ x ∈ A. mod-contains-result (m ∥↑ n) n S p x
      using A compatible disjoint-compatibility-def
            max-agg-rej4 f-profs
```

```
    by metis
  moreover have ∀ x ∈ S. prof-contains-result n S p q x
    unfolding prof-contains-result-def
  proof (clarify)
    fix x :: 'a
    assume
      x-in-S: x ∈ S
    show
      electoral-module n ∧
       finite-profile S p ∧
       finite-profile S q ∧
       x ∈ S ∧
       (x ∈ elect n S p ⟶ x ∈ elect n S q) ∧
       (x ∈ reject n S p ⟶ x ∈ reject n S q) ∧
       (x ∈ defer n S p ⟶ x ∈ defer n S q)
    proof (safe)
      show electoral-module n
        using monotone-n defer-lift-invariance-def
        by metis
    next
      show finite S
        using f-profs
        by simp
    next
      show profile S p
        using f-profs
        by simp
    next
      show finite S
        using f-profs
        by simp
    next
      show profile S q
        using f-profs
        by simp
    next
      show x ∈ S
        using x-in-S
        by simp
    next
      assume x ∈ elect n S p
      thus x ∈ elect n S q
        using defer-n lifted-x monotone-n
              f-profs defer-lift-invariance-def
        by metis
    next
      assume x ∈ reject n S p
      thus x ∈ reject n S q
        using defer-n lifted-x monotone-n
```

f-profs defer-lift-invariance-def
by *metis*
next
assume $x \in$ *defer n S p*
thus $x \in$ *defer n S q*
using *defer-n lifted-x monotone-n*
f-profs defer-lift-invariance-def
by *metis*
qed
qed
moreover have
$\forall x \in A$. *mod-contains-result n* $(m \parallel_{\uparrow} n)$ *S q x*
using *A compatible disjoint-compatibility-def*
max-agg-rej3 f-profs
by *metis*
ultimately have *00*:
$\forall x \in A$. *prof-contains-result* $(m \parallel_{\uparrow} n)$ *S p q x*
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
have
$\forall x \in S - A$. *mod-contains-result* $(m \parallel_{\uparrow} n)$ *m S p x*
using *A max-agg-rej2 monotone-m monotone-n f-profs*
defer-lift-invariance-def
by *metis*
moreover have $\forall x \in S$. *prof-contains-result m S p q x*
unfolding *prof-contains-result-def*
proof (*clarify*)
fix $x :: 'a$
assume
x-in-S: $x \in S$
show
electoral-module m $\wedge$
finite-profile S p $\wedge$
finite-profile S q $\wedge$
$x \in S \wedge$
$(x \in$ *elect m S p* $\longrightarrow x \in$ *elect m S q*$) \wedge$
$(x \in$ *reject m S p* $\longrightarrow x \in$ *reject m S q*$) \wedge$
$(x \in$ *defer m S p* $\longrightarrow x \in$ *defer m S q*$)$
proof (*safe*)
show *electoral-module m*
using *monotone-m defer-lift-invariance-def*
by *metis*
next
show *finite S*
using *f-profs*
by *simp*
next
show *profile S p*
using *f-profs*
by *simp*

next
  show *finite S*
    using *f-profs*
    by *simp*
next
  show *profile S q*
    using *f-profs*
    by *simp*
next
  show $x \in S$
    using *x-in-S*
    by *simp*
next
  assume $x \in$ *elect m S p*
  thus $x \in$ *elect m S q*
    using *A a0 indep-of-alt-def lifted-x*
      lifted-imp-equiv-prof-except-a
    by *metis*
next
  assume $x \in$ *reject m S p*
  thus $x \in$ *reject m S q*
    using *A a0 indep-of-alt-def lifted-x*
      lifted-imp-equiv-prof-except-a
    by *metis*
next
  assume $x \in$ *defer m S p*
  thus $x \in$ *defer m S q*
    using *A a0 indep-of-alt-def lifted-x*
      lifted-imp-equiv-prof-except-a
    by *metis*
qed
qed
moreover have
  $\forall x \in S{-}A$. *mod-contains-result m* $(m \parallel_{\uparrow} n)$ *S q x*
  using *A max-agg-rej1 monotone-m monotone-n f-profs*
    defer-lift-invariance-def
  by *metis*
ultimately have *01*:
  $\forall x \in S{-}A$. *prof-contains-result* $(m \parallel_{\uparrow} n)$ *S p q x*
  by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
from *00 01*
show *?thesis*
  by *blast*
next
  assume $x \notin A$
  hence *a1*: $x \in S{-}A$
    using *DiffI lifted-x compatible f-profs*
      Profile.lifted-def
    by (*metis* (*no-types*, *lifting*))

hence $x \in$ *reject n S p*
  using *A f-profs*
  by *blast*
with *defer-x*
have *defer-m*: $x \in$ *defer m S p*
  using *DiffD1 DiffD2 compatible dcompat-dec-by-one-mod*
      defer-not-elec-or-rej disjoint-compatibility-def
      not-rej-imp-elec-or-def mod-contains-result-def
      max-agg-sound par-comp-sound f-profs
      maximum-parallel-composition.simps
  by *metis*
have
  $\forall x \in A.$ *mod-contains-result* $(m \parallel_\uparrow n)$ *n S p x*
  using *A compatible disjoint-compatibility-def*
      max-agg-rej4 f-profs
  by *metis*
moreover have $\forall x \in S.$ *prof-contains-result n S p q x*
  unfolding *prof-contains-result-def*
proof (*clarify*)
  fix $x :: 'a$
    assume
      x-in-S: $x \in S$
    show
      electoral-module n $\wedge$
       finite-profile S p $\wedge$
       finite-profile S q $\wedge$
       $x \in S \wedge$
       $(x \in$ *elect n S p* $\longrightarrow x \in$ *elect n S q*$) \wedge$
       $(x \in$ *reject n S p* $\longrightarrow x \in$ *reject n S q*$) \wedge$
       $(x \in$ *defer n S p* $\longrightarrow x \in$ *defer n S q*$)$
    proof (*safe*)
      show *electoral-module n*
        using *monotone-n defer-lift-invariance-def*
        by *metis*
    next
      show *finite S*
        using *f-profs*
        by *simp*
    next
      show *profile S p*
        using *f-profs*
        by *simp*
    next
      show *finite S*
        using *f-profs*
        by *simp*
    next
      show *profile S q*
        using *f-profs*

```
        by simp
      next
        show x ∈ S
          using x-in-S
          by simp
      next
        assume x ∈ elect n S p
        thus x ∈ elect n S q
          using A a1 indep-of-alt-def lifted-x
                lifted-imp-equiv-prof-except-a
          by metis
      next
        assume x ∈ reject n S p
        thus x ∈ reject n S q
          using A a1 indep-of-alt-def lifted-x
                lifted-imp-equiv-prof-except-a
          by metis
      next
        assume x ∈ defer n S p
        thus x ∈ defer n S q
          using A a1 indep-of-alt-def lifted-x
                lifted-imp-equiv-prof-except-a
          by metis
      qed
  qed
  moreover have
    ∀ x ∈ A. mod-contains-result n (m ‖↑ n) S q x
    using A compatible disjoint-compatibility-def
          max-agg-rej3 f-profs
    by metis
  ultimately have 10:
    ∀ x ∈ A. prof-contains-result (m ‖↑ n) S p q x
    by (simp add: mod-contains-result-def prof-contains-result-def)
  have
    ∀ x ∈ S−A. mod-contains-result (m ‖↑ n) m S p x
    using A max-agg-rej2 monotone-m monotone-n
          f-profs defer-lift-invariance-def
    by metis
  moreover have ∀ x ∈ S. prof-contains-result m S p q x
    unfolding prof-contains-result-def
  proof (clarify)
    fix x :: 'a
      assume
        x-in-S: x ∈ S
      show
        electoral-module m ∧
         finite-profile S p ∧
         finite-profile S q ∧
         x ∈ S ∧
```

```
      (x ∈ elect m S p ⟶ x ∈ elect m S q) ∧
      (x ∈ reject m S p ⟶ x ∈ reject m S q) ∧
      (x ∈ defer m S p ⟶ x ∈ defer m S q)
    proof (safe)
      show electoral-module m
        using monotone-m defer-lift-invariance-def
        by metis
    next
      show finite S
        using f-profs
        by simp
    next
      show profile S p
        using f-profs
        by simp
    next
      show finite S
        using f-profs
        by simp
    next
      show profile S q
        using f-profs
        by simp
    next
      show x ∈ S
        using x-in-S
        by simp
    next
      assume x ∈ elect m S p
      thus x ∈ elect m S q
        using defer-lift-invariance-def defer-m
            lifted-x monotone-m
        by metis
    next
      assume x ∈ reject m S p
      thus x ∈ reject m S q
        using defer-lift-invariance-def defer-m
            lifted-x monotone-m
        by metis
    next
      assume x ∈ defer m S p
      thus x ∈ defer m S q
        using defer-lift-invariance-def defer-m
            lifted-x monotone-m
        by metis
    qed
qed
moreover have
  ∀ x ∈ S−A. mod-contains-result m (m ∥↑ n) S q x
```

using *A max-agg-rej1 monotone-m monotone-n*
f-profs defer-lift-invariance-def
by *metis*
ultimately have *11*:
$\forall x \in S-A.$ *prof-contains-result* $(m \parallel_{\uparrow} n)$ S p q x
using *electoral-mod-defer-elem*
by (*simp add*: *mod-contains-result-def prof-contains-result-def*)
from *10 11*
show *?thesis*
by *blast*
qed
thus $(m \parallel_{\uparrow} n)$ S p $=$ $(m \parallel_{\uparrow} n)$ S q
using *compatible disjoint-compatibility-def f-profs*
eq-alts-in-profs-imp-eq-results max-par-comp-sound
by *metis*
qed

lemma *par-comp-rej-card*:
assumes
compatible: *disjoint-compatibility* x y **and**
f-prof: *finite-profile* S p **and**
reject-sum: *card* (*reject* x S p) $+$ *card* (*reject* y S p) $=$ *card* $S + n$
shows *card* (*reject* $(x \parallel_{\uparrow} y)$ S p) $= n$
proof $-$
from *compatible* **obtain** A **where** A:
$A \subseteq S \wedge$
$(\forall a \in A.$ *indep-of-alt* x S a $\wedge$
$(\forall p.$ *finite-profile* S p $\longrightarrow$ $a \in$ *reject* x S $p)) \wedge$
$(\forall a \in S-A.$ *indep-of-alt* y S a $\wedge$
$(\forall p.$ *finite-profile* S p $\longrightarrow$ $a \in$ *reject* y S $p))$
using *disjoint-compatibility-def f-prof*
by *metis*
from *f-prof compatible*
have *reject-representation*:
reject $(x \parallel_{\uparrow} y)$ S p $=$ (*reject* x S p) $\cap$ (*reject* y S p)
using *max-agg-rej-intersect disjoint-compatibility-def*
by *blast*
have *electoral-module* x $\wedge$ *electoral-module* y
using *compatible disjoint-compatibility-def*
by *auto*
hence *subsets*: (*reject* x S p) $\subseteq S \wedge$ (*reject* y S p) $\subseteq S$
by (*simp add*: *f-prof reject-in-alts*)
hence *finite* (*reject* x S p) $\wedge$ *finite* (*reject* y S p)
using *rev-finite-subset f-prof reject-in-alts*
by *auto*
hence *0*:
card (*reject* $(x \parallel_{\uparrow} y)$ S p) $=$
card $S + n -$
card ((*reject* x S p) $\cup$ (*reject* y S p))

```
    using card-Un-Int reject-representation reject-sum
    by fastforce
  have
    ∀ a ∈ S. a ∈ (reject x S p) ∨ a ∈ (reject y S p)
    using A f-prof
    by blast
  hence S = reject x S p ∪ reject y S p
    using subsets
    by force
  hence 1: card ((reject x S p) ∪ (reject y S p)) = card S
    by presburger
  from 0 1
  show card (reject (x ‖↑ y) S p) = n
    by simp
qed


theorem par-comp-elim-one[simp]:
  assumes
    defers-m-1: defers 1 m and
    non-elec-m: non-electing m and
    rejec-n-2: rejects 2 n and
    disj-comp: disjoint-compatibility m n
  shows eliminates 1 (m ‖↑ n)
  unfolding eliminates-def
proof (safe)
  have electoral-mod-m: electoral-module m
    using non-elec-m
    by (simp add: non-electing-def)
  have electoral-mod-n: electoral-module n
    using rejec-n-2
    by (simp add: rejects-def)
  show electoral-module (m ‖↑ n)
    using electoral-mod-m electoral-mod-n
    by simp
next
  fix
    A :: 'a set and
    p :: 'a Profile
  assume
    min-2-card: 1 < card A and
    fin-A: finite A and
    prof-A: profile A p
  have card-geq-1: card A ≥ 1
    using min-2-card dual-order.strict-trans2 less-imp-le-nat
    by blast
  have module: electoral-module m
    using non-elec-m non-electing-def
    by auto
```

```
  have elec-card-0: card (elect m A p) = 0
    using fin-A prof-A non-elec-m card-eq-0-iff non-electing-def
    by metis
  moreover
  from card-geq-1 have def-card-1:
    card (defer m A p) = 1
    using defers-m-1 module fin-A prof-A
    by (simp add: defers-def)
  ultimately have card-reject-m:
    card (reject m A p) = card A − 1
  proof −
    have finite A
      by (simp add: fin-A)
    moreover have
      well-formed A
        (elect m A p, reject m A p, defer m A p)
      using fin-A prof-A electoral-module-def module
      by auto
    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-1 elec-card-0
      by simp
  qed
  have case1: card A ≥ 2
    using min-2-card
    by auto
  from case1 have card-reject-n:
    card (reject n A p) = 2
    using fin-A prof-A rejec-n-2 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-1
    by linarith
  with disj-comp prof-A fin-A card-reject-m card-reject-n
  show
    card (reject (m ∥↑ 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 $elector :: 'a\ Electoral\text{-}Module \Rightarrow 'a\ Electoral\text{-}Module$ **where**
  $elector\ m = (m \triangleright elect\text{-}module)$

4.7.2 Soundness

theorem *elector-sound*[*simp*]:
  assumes *module-m*: *electoral-module m*
  shows *electoral-module* (*elector m*)
  by (*simp add*: *module-m*)

4.7.3 Electing

theorem *elector-electing*[*simp*]:
  assumes
    module-m: *electoral-module m* **and**
    non-block-m: *non-blocking m*
  shows *electing* (*elector m*)
proof −
  obtain
    $AA :: 'a\ Electoral\text{-}Module \Rightarrow 'a\ set$ **and**
    $rrs :: 'a\ Electoral\text{-}Module \Rightarrow 'a\ Profile$ **where**
    f1:
    $\forall f.$
      $(electing\ f\ \vee$
        $\{\} = elect\ f\ (AA\ f)\ (rrs\ f) \wedge profile\ (AA\ f)\ (rrs\ f) \wedge$
          $finite\ (AA\ f) \wedge \{\} \neq AA\ f\ \vee$
        $\neg\ electoral\text{-}module\ f) \wedge$
          $((\forall A\ rs.\ \{\} \neq elect\ f\ A\ rs \vee \neg\ profile\ A\ rs\ \vee$

infinite $A \lor \{\} = A) \land$
electoral-module $f \lor$
$\neg$ *electing* f)
using *electing-def*
by *metis*
have *non-block*:
non-blocking
(*elect-module*::$'a$ *set* $\Rightarrow$ - *Profile* $\Rightarrow$ - *Result*)
by (*simp add*: *electing-imp-non-blocking*)
thus *?thesis*

proof –
obtain
AAa :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *set* **and**
rrsa :: $'a$ *Electoral-Module* $\Rightarrow$ $'a$ *Profile* **where**
f1:
$\forall f$.
(*electing* $f \lor$
$\{\} =$ *elect* f (*AAa* f) (*rrsa* f) $\land$ *profile* (*AAa* f) (*rrsa* f) $\land$
finite (*AAa* f) $\land \{\} \neq$ *AAa* $f \lor$
$\neg$ *electoral-module* f) $\land$ (($\forall A$ *rs*. $\{\} \neq$ *elect* f A *rs* $\lor$
$\neg$ *profile* A *rs* $\lor$ *infinite* $A \lor \{\} = A) \land$ *electoral-module* $f \lor$
$\neg$ *electing* f)
using *electing-def*
by *metis*
obtain
AAb :: $'a$ *Result* $\Rightarrow$ $'a$ *set* **and**
AAc :: $'a$ *Result* $\Rightarrow$ $'a$ *set* **and**
AAd :: $'a$ *Result* $\Rightarrow$ $'a$ *set* **where**
f2:
$\forall p$. (*AAb* p, *AAc* p, *AAd* p) $= p$
using *disjoint3.cases*
by (*metis* (*no-types*))
have *f3*:
electoral-module (*elector* m)
using *elector-sound module-m*
by *simp*
have *f4*:
$\forall p$. (*elect-r* p, *AAc* p, *AAd* p) $= p$
using *f2*
by *simp*
have
finite (*AAa* (*elector* m)) $\land$
profile (*AAa* (*elector* m)) (*rrsa* (*elector* m)) $\land$
$\{\} =$ *elect* (*elector* m) (*AAa* (*elector* m)) (*rrsa* (*elector* m)) $\land$
$\{\} =$ *AAd* (*elector* m (*AAa* (*elector* m)) (*rrsa* (*elector* m))) $\land$
reject (*elector* m) (*AAa* (*elector* m)) (*rrsa* (*elector* m)) $=$
AAc (*elector* m (*AAa* (*elector* m)) (*rrsa* (*elector* m))) $\longrightarrow$
electing (*elector* m)

```
    using f2 f1 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
  moreover
  {
    assume
      {} ≠ AAd (elector m (AAa (elector m)) (rrsa (elector m)))
    hence
      ¬ profile (AAa (elector m)) (rrsa (elector m)) ∨
        infinite (AAa (elector m))
      using f4
      by simp
  }
  ultimately show ?thesis
    using f4 f3 f1 fst-conv snd-conv
    by metis
 qed
qed
```

4.7.4 Composition Rule

```
lemma dcc-imp-cc-elector:
  assumes dcc: 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 dcc defer-condorcet-consistency-def
        elect-mod-sound seq-comp-sound
    by metis
next
  show
    ⋀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
      ∀ y∈A − {w}.
```

$card \{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
$card \{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\} \Longrightarrow$
$x \in elect\ m\ A\ p \Longrightarrow x \in A$
using *dcc defer-condorcet-consistency-def*
elect-in-alts subset-eq finite prof-A
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: $\forall\, y{\in}A - \{w\}.$
$card \{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
$card \{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$
have *condorcet-winner* $A\ p\ w$
using *finite prof-A w-in-A 2*
by *simp*
thus $xa = x$
using *condorcet-winner.simps dcc fst-conv insert-Diff 1*
defer-condorcet-consistency-def insert-not-empty
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: $\forall\, y{\in}A - \{w\}.$
$card \{i.\ i < length\ p \wedge (w,\ y) \in (p!i)\} <$
$card \{i.\ i < length\ p \wedge (y,\ w) \in (p!i)\}$ **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 dcc defer-in-alts*
defer-condorcet-consistency-def order-trans

subset-Compl-singleton
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* $p \wedge (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (y, w) \in (p!i)\}$ **and**
3: $\neg$ *card* $\{i.\ i <$ *length* $p \wedge (x, xa) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (xa, x) \in (p!i)\}$
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 dcc empty-iff xa-in-A*
defer-condorcet-consistency-def 3 DiffI
cond-winner-unique3 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* $p \wedge (w, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (y, w) \in (p!i)\}$ **and**
3: $\forall y \in A - \{x\}$.
card $\{i.\ i <$ *length* $p \wedge (x, y) \in (p!i)\} <$
card $\{i.\ i <$ *length* $p \wedge (y, x) \in (p!i)\}$
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 dcc*
    defer-condorcet-consistency-def
    cond-winner-unique3 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 dcc defer-condorcet-consistency-def finite*
    prof-A reject-in-alts subsetD
  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 condorcet-winner.simps dcc IntI empty-iff*

```
        defer-condorcet-consistency-def result-disj
    by (metis (no-types, hide-lams))
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 dcc defer-condorcet-consistency-def IntI
          Diff-empty Diff-iff finite prof-A result-disj
    by (metis (no-types, hide-lams))
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 condorcet-winner.simps dcc x-in-A
          defer-condorcet-consistency-def electoral-mod-defer-elem
    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**
  $iter\ m =$
    $(let\ t = defer\text{-}equal\text{-}condition\ 1\ in$
      $(m\ \circlearrowleft_t))$

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

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 =
  ((min-eliminator condorcet-score) ↻∃!d) A p

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

5.2.2 Condorcet Consistency Property

```
theorem condorcet-condorcet: condorcet-consistency pairwise-majority-rule
proof −
  have
    condorcet-consistency (elector condorcet)
    using condorcet-is-dcc dcc-imp-cc-elector
    by metis
  thus ?thesis
    using condorcet-consistency2 electoral-module-def
          pairwise-majority-rule.simps
    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 Condorcet Consistency Property

```
theorem copeland-condorcet: condorcet-consistency copeland-rule
```

proof −
  have
    condorcet-consistency (*elector copeland*)
    using *copeland-is-dcc dcc-imp-cc-elector*
    by *metis*
  thus *?thesis*
    using *condorcet-consistency2 electoral-module-def*
        copeland-rule.simps
    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 Condorcet Consistency Property

theorem *minimax-condorcet*: *condorcet-consistency minimax-rule*
proof −
  have
    condorcet-consistency (*elector minimax*)
    using *minimax-is-dcc dcc-imp-cc-elector*
    by *metis*
  thus *?thesis*
    using *condorcet-consistency2 electoral-module-def*
        minimax-rule.simps
    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*

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*

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*

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*

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 =
((((((*pass-module 2 x*) ▷ ((*plurality*↓) ▷ (*pass-module 1 x*))) $\|_\uparrow$
(*drop-module 2 x*)) $\circlearrowleft_{\exists!d}$) ▷ *elect-module*) *A p*)

5.9.2 Soundness

theorem *smc-sound*:
assumes *order*: *linear-order x*
shows *electoral-module* (*smc x*)
unfolding *electoral-module-def*
proof (*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 ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ (*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-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x $\circlearrowleft_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
reject-xa:
xa ∈ *reject* (*?smc*) *A p* **and**
defer-xa:
xa ∈ *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-aggregator*
let *?t* = *defer-equal-condition*
let *?smc* =
pass-module 2 x ▷
((*plurality*↓) ▷ *pass-module* (*Suc 0*) *x*) $\|_?$*a*
drop-module 2 x $\circlearrowleft_?$*t* (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
elect-xa:
xa ∈ *elect* (*?smc*) *A p*
show *xa* ∈ *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 $\circlearrowleft_{?t}$ (*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 $\circlearrowleft_{?t}$ (*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* $\in$ *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 $\triangleright$
((*plurality*$\downarrow$) $\triangleright$ *pass-module* (*Suc 0*) *x*) $\|_{?a}$
drop-module 2 x $\circlearrowleft_{?t}$ (*Suc 0*)
assume
fin-A: *finite A* **and**
prof-A: *profile A p* **and**
xa-in-A: *xa* $\in$ *A* **and**
not-defer-xa:
xa $\notin$ *defer* (*?smc*) *A p* **and**
not-reject-xa:
xa $\notin$ *reject* (*?smc*) *A p*
show *xa* $\in$ *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

theorem *smc-electing*:
assumes *order*: *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*$\downarrow$) $\triangleright$ *?tie-breaker*
let *?compare-two* = *?pass2* $\triangleright$ *?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 *order*
  by *simp*
have *00013*: *defers 1 ?tie-breaker*
  using *order pass-one-mod-def-one*
  by *simp*
have *20000*: *non-blocking* (*plurality*↓)
  by *simp*

have *0020*: *disjoint-compatibility ?pass2 ?drop2*
  using *order*
  by *simp*
have *1000*: *non-electing ?pass2*
  using *order*
  by *simp*
have *1001*: *non-electing ?plurality-defer*
  using *00011 00012*
  by *simp*
have *2000*: *non-blocking ?pass2*
  using *order*
  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 *order 0020*
  by *simp*
have *100*: *non-electing ?compare-two*
  using *1000 1001*
  by *simp*
have *101*: *non-electing ?drop2*
  using *order*
  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 *order*
  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 smc-sound smc.simps electing-def*
    Defer-One-Loop-Composition.iter.simps
    order seq-comp-electing
  by *metis*
qed

5.9.4 (Weak) Monotonicity Property

theorem *smc-monotone*:
  assumes *order*: *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 *order*
  by *simp*
have *00013*: *defers 1 ?tie-breaker*
  using *order pass-one-mod-def-one*
  by *simp*
have *00014*: *defer-monotonicity ?tie-breaker*
  using *order*
  by *simp*
have *20000*: *non-blocking* (*plurality*↓)
  by *simp*

```
have 0000: defer-lift-invariance ?pass2
  using order
  by simp
have 0001: defer-lift-invariance ?plurality-defer
  using 00010 00011 00012 00013 00014
  by simp
have 0020: disjoint-compatibility ?pass2 ?drop2
  using order
  by simp
have 1000: non-electing ?pass2
  using order
  by simp
have 1001: non-electing ?plurality-defer
  using 00011 00012
  by simp
have 2000: non-blocking ?pass2
  using order
  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 order
  by simp
have 002: disjoint-compatibility ?compare-two ?drop2
  using order 0020
  by simp

have 100: non-electing ?compare-two
  using 1000 1001
  by simp
have 101: non-electing ?drop2
  using order
  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 order
  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
          Electoral-Module.monotonicity-def
          Defer-One-Loop-Composition.iter.simps
          smc-sound smc.simps order seq-comp-mono
    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.