Refinement of Fair Modular Voting Rules

Valentin Springsklee

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
valentin.springsklee@student.kit.edu

October 12, 2022

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.

In this article, the Isabelle Refinement Framework by Peter Lammich [3] is used to generate verified efficient code for voting rules that are specified in a framework for the construction of such fair voting rules using composable modules [1, 2].

Contents

theory *Preference-List*
  imports
    Verified-Voting-Rule-Construction.Preference-Relation
    List−Index.List-Index
begin

ordered from most to least preferred candidate

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

definition *well-formed-pl* :: $'a$ *Preference-List* $\Rightarrow$ *bool* **where**
  well-formed-pl pl $\equiv$ *length pl* $> 0 \wedge$ *distinct pl*

rank 1 is top prefernce, rank 0 is not in list

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

fun *is-less-preferred-than* ::
  $'a \Rightarrow$ $'a$ *Preference-List* $\Rightarrow$ $'a \Rightarrow$ *bool* (- $\lesssim_{-}$ - [*50*, *1000*, *51*] *50*) **where**
    $x \lesssim_r y$ $=$ ((*List.member r x*) $\wedge$ (*List.member r y*) $\wedge$ (*rank-l r x* $\geq$ *rank-l r y*))

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

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

lemma *rank-gt-zero*:
  assumes
    wf : *well-formed-pl r* **and**
    refl: $x \lesssim_r x$
  shows *rank-l r x* $\geq$ *1*
proof *auto*
  from *refl* **show** *List.member r x* **by** *auto*
qed

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

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

definition *trans* :: *'a Preference-List* ⇒ *bool* **where**
trans r ≡ $\forall (x, y, z) \in ((set\ r) \times (set\ r) \times (set\ r)).$
$x \lesssim_r y \wedge y \lesssim_r z \longrightarrow x \lesssim_r z$

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

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

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

lemma *lin-ord-imp-connex-l*:
assumes *linear-order-on-l A r*
shows *connex-l A r*
by (*metis Preference-List.connex-l-def Preference-List.is-less-preferred-than.simps*

linear-order-on-l-def preorder-on-l-def
total-on-l-def assms nle-le)

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

lemma *limited-dest*:
shows $(\bigwedge\ x\ y.\ [\![$ *is-less-preferred-than x r y*; *limited A r* $]\!] \Longrightarrow\ x \in A \wedge y \in A)$
unfolding *limited-def* **by** (*simp*)

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

lemma *above-trans*:
assumes
trans: *trans r* **and**
less: $a \lesssim_r b$
shows *set* (*above-l r b*) ⊆ *set* (*above-l r a*)
by (*metis Preference-List.above-l-def Preference-List.is-less-preferred-than.elims*(*2*)
less set-take-subset-set-take)

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

lemma *less-preffered-l-rel-eq*:
  shows $a \lesssim_l b \longleftrightarrow$ *Preference-Relation.is-less-preferred-than a* (*pl-*α *l*) *b*
  by (*simp add*: *pl-*α*-def*)

theorem *aboveeq*: **assumes** *wf*: *well-formed-pl l* **and** *lo*: *linear-order-on-l A l*
  shows *set* (*above-l l a*) = *Order-Relation.above* (*pl-*α *l*) *a*
proof
  show *set* (*above-l l a*) $\subseteq$ *above* (*pl-*α *l*) *a*
  proof *clarify*
    fix *x*
    assume $x \in$ *set* (*Preference-List.above-l l a*)
    have *length* (*above-l l a*) = *rank-l l a* **unfolding** *above-l-def*
      by (*simp add*: *Suc-le-eq in-set-member*)
    from *wf lo this* **have** *index l x* $\leq$ *index l a* **unfolding** *rank-l.simps*
      by (*metis Preference-List.above-l-def Preference-List.rank-l.simps Suc-eq-plus1 Suc-le-eq* ‹$x \in$ *set* (*Preference-List.above-l l a*)› *bot-nat-0.extremum-strict index-take linorder-not-less*)
    from *this* **have** $a \lesssim_l x$
      by (*metis One-nat-def Preference-List.above-l-def Preference-List.is-less-preferred-than.elims*(*3*) *Preference-List.rank-l.simps Suc-le-mono* ‹$x \in$ *set* (*Preference-List.above-l l a*)› *add.commute add-0 add-Suc empty-iff find-index-le-size in-set-member index-def le-antisym list.set*(*1*) *size-index-conv take-0*)
    from *this* **have** *Preference-Relation.is-less-preferred-than a* (*pl-*α *l*) *x*
      using *less-preffered-l-rel-eq* **by** (*metis*)
    from *this* **show** $x \in$ *Order-Relation.above* (*pl-*α *l*) *a*
      using *pref-imp-in-above* **by** (*metis*)
  qed
next
  show *above* (*pl-*α *l*) *a* $\subseteq$ *set* (*above-l l a*)
  proof *clarify*
    fix *x*
    assume $x \in$ *Order-Relation.above* (*pl-*α *l*) *a*
    from *this* **have** *Preference-Relation.is-less-preferred-than a* (*pl-*α *l*) *x*
      using *pref-imp-in-above* **by** (*metis*)
    from *this* **have** *alpx*: $a \lesssim_l x$
      using *less-preffered-l-rel-eq* **by** (*metis*)
    from *this* **have** *rank-l l x* $\leq$ *rank-l l a*
      by *auto*
    from *this alpx* **show** $x \in$ *set* (*Preference-List.above-l l a*)
      using *Preference-List.above-l-def Preference-List.is-less-preferred-than.simps Preference-List.rank-l.simps*
      by (*metis Suc-eq-plus1 Suc-le-eq in-set-member index-less-size-conv set-take-if-index*)
  qed
qed

theorem *rankeq*: **assumes** *wf*: *well-formed-pl l* **and** *lo*: *linear-order-on-l A l*
  shows *rank-l l a* = *Preference-Relation.rank* (*pl-*α *l*) *a*

proof (*simp*, *safe*)
  assume *air*: *List.member l a*
  from *assms* **have** *abe*: *Order-Relation.above* (*pl-α l*) *a* = *set* (*above-l l a*)
    using *aboveeq*
    by *metis*
  from *wf* **have** *dl*: *distinct* (*above-l l a*) **unfolding** *well-formed-pl-def above-l-def*
    using *distinct-take* **by** *blast*
  from *dl* **have** *ce*: *card* (*set* (*above-l l a*)) = *length* (*above-l l a*) **unfolding** *well-formed-pl-def*
    using *distinct-card*
    by *blast*
  have *length* (*above-l l a*) = *rank-l l a* **unfolding** *above-l-def*
    by (*simp add*: *Suc-le-eq in-set-member*)
  from *air abe dl ce this* **show** *Suc* (*index l a*) = *card* (*Order-Relation.above* (*pl-α l*) *a*)
    by *simp*
next
  assume *anr*: ¬ *List.member l a*
  from *anr* **have** *a* ∉ (*set l*) **unfolding** *pl-α-def*
    by (*simp add*: *in-set-member*)
  from *this* **have** *a* ∉ *Order-Relation.above* (*pl-α l*) *a*
      unfolding *Order-Relation.above-def pl-α-def*
      by (*simp add*: *anr*)
  from *this* **have** *Order-Relation.above* (*pl-α l*) *a* = {}
      unfolding *Order-Relation.above-def*
      using *anr less-preffered-l-rel-eq* **by** *fastforce*
  from *this* **show** *card* (*Order-Relation.above* (*pl-α l*) *a*) = *0* **by** *fastforce*
qed

theorem *linorder-l-imp-rel*:
  assumes *wf*: *well-formed-pl l* **and** *lo*: *linear-order-on-l A l*
  shows *Order-Relation.linear-order-on A* (*pl-α l*)
proof (*unfold Order-Relation.linear-order-on-def partial-order-on-def*
    Order-Relation.preorder-on-def, *clarsimp*, *safe*)
  from *wf* **have** *l* ≠ [] **using** *well-formed-pl-def*
    by *auto*
  from *lo* **have** *refl-on-l A l*
    by (*unfold linear-order-on-l-def preorder-on-l-def*, *simp*)
  from *this* **show** *refl-on A* (*pl-α l*)
  proof (*unfold refl-on-l-def pl-α-def refl-on-def*, *clarsimp*)
    fix *a* **and** *b*
    assume *ni*: ∀ *x*∈*A*. *List.member l x*
    assume *aA*: *List.member l a* **and** *bA*: *List.member l b*
    from *ni aA bA* **show** *a* ∈ *A* ∧ *b* ∈ *A*
      using *lo linear-order-on-l-def preorder-on-l-def Preference-List.limited-def* **by** (*metis*)
  qed
next
  show *Relation.trans* (*pl-α l*)

```
    by (unfold Preference-List.trans-def pl-α-def Relation.trans-def, simp)
next
  show antisym (pl-α l)
  proof (unfold antisym-def pl-α-def is-less-preferred-than.simps, clarsimp)
    fix x and y
    assume xm: List.member l x and ym: List.member l y
    assume si: index l x = index l y
    from xm ym si show x = y
      by (simp add: member-def)
  qed
next
  show total-on A (pl-α l)
    using connex-l-def lin-ord-imp-connex-l lo pl-α-def total-on-def by fastforce
qed

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

lemma rel-trans:
  shows Relation.trans (pl-α pl)
  unfolding Relation.trans-def pl-α-def
  by auto

lemma connex-imp-refl:
  assumes connex-l A pl
  shows refl-on-l A pl
  unfolding connex-l-def refl-on-l-def
proof clarsimp
```

```
  fix x
  assume x ∈ A
  from this assms show List.member pl x
   by (metis Preference-List.connex-l-def Preference-List.is-less-preferred-than.elims(1))
qed

lemma aconnex:
  assumes well-formed-pl pl and lo: linear-order-on-l A pl
  shows connex-l A pl
  using  Preference-List.connex-l-def Preference-List.is-less-preferred-than.simps
    linear-order-on-l-def preorder-on-l-def refl-on-l-def lo
  by (metis nle-le)

end
theory Profile-Array
  imports Verified-Voting-Rule-Construction.Profile
  Preference-List
  List−Index.List-Index
  CAVA-Base.CAVA-Base
  Collections.Diff-Array
begin

notation array-get (-[[-]] [900,0] 1000)

value list-of-array (array-of-list [1::nat,2])

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

fun pr1-α :: 'a Profile-List ⇒ 'a Profile where
  pr1-α pr1 = map (Preference-List.pl-α) pr1

type-synonym 'a Preference-Array = 'a array


definition is-less-pref-array ::'a ⇒ 'a Preference-Array ⇒ 'a ⇒ bool nres where
  is-less-pref-array x ballot y ≡ do {
    (i, rank) ← WHILET (λ(i, rank). (i < (array-length ballot)))
      (λ(i, rank). do {
      let c = ballot[[i]];
      let ret = (if (c = y) then True else False);
      let i = i + 1;
      RETURN (i, rank)
    })(0,0);
    RETURN (True)
  }

type-synonym 'a Profile-Array = ('a Preference-Array) array

definition profile-l :: 'a set ⇒ 'a Profile-List ⇒ bool where
```

profile-l A pr1 ≡ (∀ *i::nat. i* < *length pr1* ⟶
well-formed-pl (*pr1*!*i*) ∧ *linear-order-on-l A* (*pr1*!*i*))

definition *well-formed-prefa* :: *'a Preference-Array* ⇒ *bool* **where**
well-formed-prefa pa = ((*array-length pa* > *0*) ∧ *distinct* (*list-of-array pa*))

lemma *wfa-imp-wfl*[*simp*]: *well-formed-prefa pa* ⟶ *well-formed-pl* (*list-of-array pa*)
unfolding *well-formed-prefa-def well-formed-pl-def*
by (*simp add*: *array-length-list*)

definition *rank-array-invariant ballot-a a* ≡ λ (*i*, *rank*).
∀ *idx* < *i*. *ballot-a*[[*idx*]] ≠ *a* ∨ *rank* = *i*

definition *rank-array-mon* :: *'a Preference-Array* ⇒ *'a* ⇒ *nat nres* **where**
rank-array-mon ballot-a a ≡ *do* {
(*i*, *rank*) ← *WHILET* (λ(*i*, *rank*). (*i* < (*array-length ballot-a*) ∧ *rank* = *0*))
(λ(*i*, *rank*). *do* {
let rank = (*if* (*ballot-a*[[*i*]] = *a*) *then* (*i* + *1*) *else 0*);
let i = *i* + *1*;
RETURN (*i*, *rank*)
})(*0*,*0*);
RETURN rank
}

lemma *rank-array-mon-correct*: **assumes** *prof*: *well-formed-prefa a*
shows *rank-array-mon ballot-a a* ≤ *SPEC* (λ *r*. *r* = *rank* (*pl-α* (*list-of-array ballot-a*)) *a*)
unfolding *rank-array-mon-def rank-l.simps*
apply (*intro WHILET-rule*[**where** *I*=(*rank-array-invariant ballot-a a*) **and** *R*=*measure* (λ(*i*,-). (*array-length ballot-a*) − *i*)] *refine-vcg*)
unfolding *rank-array-invariant-def*
apply *auto*
proof −
fix *i*
assume *nir*: ¬ *i* < *array-length ballot-a*
from *nir prof* **have** ∀ *idx* < *array-length ballot-a*. *ballot-a*[[*idx*]] ≠ *a*
unfolding *well-formed-prefa-def*

oops

Profile Array abstraction functions

definition *pa-to-pl* :: *'a Profile-Array* ⇒ *'a Profile-List* **where**
pa-to-pl pa = *map* (*list-of-array*) (*list-of-array pa*)

definition *pa-to-pr* :: *'a Profile-Array* ⇒ *'a Profile* **where**
pa-to-pr pa = *pr1-α* (*pa-to-pl pa*)

definition *pl-to-pa* :: *'a Profile-List* ⇒ *'a Profile-Array* **where**

```
pl-to-pa pa = array-of-list (map (array-of-list) (pa))
```

Profile properties and refinement

```
definition profile-a :: 'a set ⇒ 'a Profile-Array ⇒ bool where
  profile-a A pa = profile-l A (pa-to-pl pa)

abbreviation finite-profile-a :: 'a set ⇒ 'a Profile-Array ⇒ bool where
  finite-profile-a A pa ≡ finite A ∧ profile-a A pa

lemma profile-data-refine:
  assumes (pl,pr)∈build-rel pr1-α (profile-l A)
  shows profile A pr
  unfolding profile-def
  apply(intro allI impI)
proof (−)
  fix i
  assume ir: i < length pr
  from ir assms have well-formed-pl (pl ! i) unfolding profile-l-def
    by (simp add: in-br-conv)
  from ir assms have linear-order-on-l A (pl ! i) unfolding profile-l-def
    by (simp add: in-br-conv)
  from assms this show linear-order-on A (pr ! i) unfolding profile-l-def
    using linorder-l-imp-rel
    by (metis (mono-tags, lifting) in-br-conv ir length-map nth-map pr1-α.simps)
qed

lemma profile-a-l: assumes profile-a A pa
  shows profile-l A (pa-to-pl pa)
  using assms profile-a-def by (metis)

lemma profile-a-rel: assumes profile-a A pa
  shows profile A (pa-to-pr pa)
  using profile-data-refine
  by (metis assms brI pa-to-pr-def profile-a-def)
```

Monadic redifintion of counting functions.

```
definition wc-invar p0 a ≡ λ(r,ac).
  r ≤ length p0
  ∧ ac = card{i::nat. i < r ∧ above (p0!i) a = {a}}

definition win-count-mon :: 'a Profile ⇒ 'a ⇒ nat nres where
win-count-mon p a ≡ do {
  (r, ac) ← WHILET (λ(r, -). r < length p) (λ(r, ac). do {
    ASSERT (r < length p);
    let ac = ac + (if (above (p!r) a = {a}) then 1 else 0);
    let r = r + 1;
    RETURN (r, ac)
  })(0,0);
  RETURN ac
```

}

definition *win-count-mon-r* :: *'a Profile* ⇒ *'a* ⇒ *nat nres* **where**
win-count-mon-r p a ≡ *do* {
  (*r*, *ac*) ← *WHILET* (λ(*r*, -). *r* < *length p*) (λ(*r*, *ac*). *do* {
    ASSERT (*r* < *length p*);
    let ac = *ac* + (*if* (*rank* (*p*!*r*) *a* = *1*) *then 1 else 0*);
    let r = *r* + *1*;
    RETURN (*r*, *ac*)
  })(*0*,*0*);
  RETURN ac
}

lemma *win-count-mon-correct*:
  shows *win-count-mon p a* ≤ *SPEC* (λ *wc*. *wc* = *win-count p a*)
  unfolding *win-count-mon-def win-count.simps*
  apply (*intro WHILET-rule*[**where** *I*=(*wc-invar p a*) **and** *R*=*measure* (λ(*r*,-). (*length p*) − *r*)] *refine-vcg*)
  unfolding *wc-invar-def*
  apply (*simp-all*)
  apply (*erule subst*)
  apply (*simp*)
  apply (*intro conjI impI*)
proof (*simp-all*)
  fix *r* :: *nat*
  assume *le*: *r* < *length p*
  assume *atop*: *above* (*p* ! *r*) *a* = {*a*}
  with *atop* **have** *prep*:
      {*i*. *i* < *Suc r* ∧ *above* (*p* ! *i*) *a* = {*a*}}
     = {*i*. *i* < *r* ∧ *above* (*p* ! *i*) *a* = {*a*}} ∪ {*r*}
    by *fastforce*
  then show
      Suc (*card* {*i*. *i* < *r* ∧ *above* (*p* ! *i*) *a* = {*a*}}) =
       card {*i*. *i* < *Suc r* ∧ *above* (*p* ! *i*) *a* = {*a*}}
    by *fastforce*
next
  fix *r* :: *nat*
  assume *r* < *length p*
  assume *atop*: *above* (*p* ! *r*) *a* ≠ {*a*}
  then show
     card {*i*. *i* < *r* ∧ *above* (*p* ! *i*) *a* = {*a*}} =
     card {*i*. *i* < *Suc r* ∧ *above* (*p* ! *i*) *a* = {*a*}}
    by (*metis less-Suc-eq*)
qed

lemma *carde*: **assumes** *pprofile*: *profile A p*
  shows $\forall$ *r* < *length p*. (*card* (*above* (*p* ! *r*) *a*) = *1*) = (*above* (*p* ! *r*) *a* = {*a*})
  using *pprofile*
   by (*metis profile-def rank.simps Preference-Relation.rankone1 Preference-Relation.rankone2*)

lemma *win-count-mon-r-correct*:
  assumes *prof*: *profile A p*
  shows *win-count-mon-r p a* $\leq$ *SPEC* (λ *wc*. *wc* = *win-count p a*)
  unfolding *win-count-mon-r-def win-count.simps*
   apply (*intro WHILET-rule*[**where** *I*=(*wc-invar p a*) **and** *R*=*measure* (λ(*r*,-). (*length p*) − *r*)] *refine-vcg*)
  unfolding *wc-invar-def*
  apply (*simp-all*)
  apply (*erule subst*)
  apply (*simp*)
proof (*safe*, *simp-all*)
  fix *aa*
  assume *aail*: *aa* < *length p*
  assume *rank1*: *card* (*above* (*p* ! *aa*) *a*) = *Suc 0*
  from *aail prof rank1* **have** *above* (*p* ! *aa*) *a* = {*a*}
    by (*metis One-nat-def profile-def rank.simps rankone2*)
  from *this* **have** *prep*:
        {*i*. *i* < *Suc aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
       = {*i*. *i* < *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}} $\cup$ {*aa*}
    by *fastforce*
  then show
        Suc (*card* {*i*. *i* < *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}) =
         card {*i*. *i* < *Suc aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
    by *simp*
next
  fix *aa*
  assume *aail*: *aa* < *length p*
  assume *rank1*: *card* (*above* (*p* ! *aa*) *a*) $\neq$ *Suc 0*
  from *aail rank1* **have** *neq*: *above* (*p* ! *aa*) *a* $\neq$ {*a*}
    by *fastforce*
  have *seteq*:
      {*i*. *i* < *Suc aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
      ={*i*. *i* < *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}} $\cup$ {*i*. *i* = *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
    by *fastforce*
  from *neq* **have** *emp*: {*i*. *i* = *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}} = {} **by** *blast*
    from *seteq emp* **have**
    {*i*. *i* < *Suc aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}} = {*i*. *i* < *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
    by *simp*
  then show
      card {*i*. *i* < *aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}} =
      card {*i*. *i* < *Suc aa* $\wedge$ *above* (*p* ! *i*) *a* = {*a*}}
    by *simp*
qed

definition *winsr* :: *'a Preference-Relation* ⇒ *'a* ⇒ *nat* **where**
winsr r a ≡ (*if* (*rank r a* = *1*) *then 1 else 0*)

definition *win-count-mon-outer* :: *'a Profile* ⇒ *'a* ⇒ *nat nres* **where**
win-count-mon-outer p a ≡ *do* {
(*r*, *ac*) ← *WHILET* (λ(*r*, *-*). *r* < *length p*) (λ(*r*, *ac*). *do* {
ASSERT (*r* < *length p*);
let ac = *ac* + *winsr* (*p*!*r*) *a*;
let r = *r* + *1*;
RETURN (*r*, *ac*)
})(*0*,*0*);
RETURN ac
}

lemma *win-count-mon-outer-correct*:
assumes *prof*: *profile A p*
shows *win-count-mon-outer p a* ≤ *SPEC* (λ *wc*. *wc* = *win-count p a*)
proof −
have *eq*: *win-count-mon-outer p a* = *win-count-mon-r p a*
unfolding *win-count-mon-outer-def win-count-mon-r-def winsr-def*
by *fastforce*
from *eq* **show** *?thesis* **using** *win-count-mon-r-correct*
using *prof* **by** *fastforce*
qed

schematic-goal *wc-code-aux*: *RETURN ?wc-code* ≤ *win-count-mon p a*
unfolding *win-count-mon-def*
by (*refine-transfer*)

concrete-definition *win-count-code* **for** *p a* **uses** *wc-code-aux*

lemma *win-count-equiv*:
shows *win-count p a* = *win-count-code p a*
proof −
from *order-trans*[*OF win-count-code.refine win-count-mon-correct*]
have *win-count-code p a* = *win-count p a*
by *fastforce*
thus *?thesis* **by** *simp*
qed

export-code *win-count* **in** *Scala*

Data refinement

definition *winsr-imp* :: *'a Preference-List* ⇒ *'a* ⇒ *nat* **where**
winsr-imp l a ≡ (*if* (*rank-l l a* = *1*) *then 1 else 0*)

definition *winsr-imp'* :: *'a Preference-List* ⇒ *'a* ⇒ *nat* **where**
winsr-imp' l a ≡ (*if* (*l*!*0* = *a*) *then 1 else 0*)

lemma *winsr-imp-refine*:
  assumes *linear-order-on-l A l*
  assumes $(l,r) \in$ *build-rel pl-*α *well-formed-pl*
  shows *winsr-imp l a* = (*winsr r a*)
  unfolding *winsr-imp-def winsr-def*
  using *rankeq*
  by (*metis assms*(*1*) *assms*(*2*) *in-br-conv*)

lemma *winsr-imp′-eq*:
  assumes *well-formed-pl l*
  shows *winsr-imp′ l a* = (*winsr-imp l a*)
  unfolding *winsr-imp′-def winsr-imp-def*
proof (*simp*, *safe*)
  show *List-Index.index l* (*l* ! *0*) = *0*
    by (*simp add*: *index-eqI*)
next
  assume *amem*: *List.member l a*
  assume *anhd*: *l*!*0* ≠ *a*
  from *amem anhd* **show** *0* < *List-Index.index l a*
    by (*metis gr0I in-set-member nth-index*)
next
  assume *nmem*: ¬ *List.member l* (*l*!*0*)
  assume *fstex*: *a* = *l* ! *0*
  from *nmem* **have** *set l* = {}
    by (*metis length-greater-0-conv member-def nth-mem set-empty2*)
  from *this* **have** *l* = []
    using *set-empty* **by** *simp*
  from *assms this* **show** *False* **unfolding** *well-formed-pl-def* **by** *simp*
qed

definition *win-count-imp* :: ′*a Profile-List* ⇒ ′*a* ⇒ *nat nres* **where**
win-count-imp p a ≡ *do* {
  (*r*, *ac*) ← *WHILET* (λ(*r*, -). *r* < *length p*) (λ(*r*, *ac*). *do* {
    ASSERT (*r* < *length p*);
    let ac = *ac* + (*winsr-imp* (*p*!*r*) *a*);
    let r = *r* + *1*;
    RETURN (*r*, *ac*)
  })(*0*,*0*);
  RETURN ac
}

lemma *win-count-imp-refine*:
  assumes $(pl,pr) \in$ *build-rel pr1-*α (*profile-l A*)
  shows *win-count-imp pl a* ≤ ⇓*Id* (*win-count-mon-outer pr a*)
  using *assms* **unfolding** *win-count-imp-def win-count-mon-outer-def*
  apply (*refine-rcg*)

apply (*refine-dref-type*) — Type-based heuristics to instantiate data refinement goals
apply *simp*
apply (*auto simp add*:
refine-hsimp refine-rel-defs)
using *winsr-imp-refine*
by (*metis in-br-conv profile-l-def*)

theorem *win-count-imp-correct*:
assumes $(pl,pr) \in build\text{-}rel\ pr1\text{-}\alpha\ (profile\text{-}l\ A)$
shows $win\text{-}count\text{-}imp\ pl\ a \leq SPEC\ (\lambda\ wc.\ wc = win\text{-}count\ pr\ a)$
using *ref-two-step*[*OF win-count-imp-refine win-count-mon-outer-correct*] *assms*
profile-data-refine **by** *fastforce*

definition *win-count-imp'* :: *'a Profile-List* ⇒ *'a* ⇒ *nat nres* **where**
win-count-imp' p a ≡ *do* {
(r, ac) ← *WHILET* (λ(r, -). r < *length p*) (λ(r, ac). *do* {
ASSERT (r < *length p*);
let ballot = (p!r);
let ac = *ac* + *winsr-imp' ballot a*;
let r = *r* + *1*;
RETURN (r, ac)
})(*0*,*0*);
RETURN ac
}

lemma *win-count-imp'-refine*: **assumes** *profile-l A pl*
shows *win-count-imp' pl a* ≤ ⇓*Id* (*win-count-imp pl a*)
unfolding *win-count-imp'-def win-count-imp-def winsr-imp-def wc-invar-def*
apply (*refine-rcg*)
apply (*refine-dref-type*) — Type-based heuristics to instantiate data refinement goals
apply *simp-all*
apply (*auto simp add*:
refine-hsimp refine-rel-defs)
proof (*unfold winsr-imp'-def*, *simp-all*)
fix *x1*
assume *range*: *x1* < *length pl*
assume *mem*: *List.member* (*pl* ! *x1*) *a*
assume *fst*: *List-Index.index* (*pl* ! *x1*) *a* = *0*
from *mem fst* **show** *pl*!*x1*!*0* = *a*
by (*metis in-set-member nth-index*)
next
fix *x1*
assume *range*: *x1* < *length pl*
assume *mem*: *List.member* (*pl* ! *x1*) *a*
assume *nfst*: *List-Index.index* (*pl* ! *x1*) *a* > *0*

from *mem nfst* **show** *pl*!*x1*!*0* ≠ *a*
  by (*metis index-eq-iff*)
next
  fix *x1*
  assume *range*: *x1* < *length pl*
  assume *nmem*: ¬ *List.member* (*pl* ! *x1*) *a*
  from *assms range* **have** *nonempty-ballot*: (*pl*!*x1*) ≠ [] **unfolding** *profile-l-def well-formed-pl-def*
    by (*metis len-greater-imp-nonempty*)
  have *l*≠[] ∧ (*l*!*0* = *a*) ⟶ *List.member l a*
    by (*metis length-greater-0-conv member-def nth-mem*)
  from *this nonempty-ballot nmem* **show** *pl*!*x1*!*0* ≠ *a*
    by (*metis length-greater-0-conv member-def nth-mem*)
qed

theorem *win-count-imp′-correct*:
  assumes (*pl*,*pr*)∈*build-rel pr1-α* (*profile-l A*)
  shows *win-count-imp′ pl a* ≤ *SPEC* (λ *wc*. *wc* = *win-count pr a*)
  using *ref-two-step*[*OF win-count-imp′-refine win-count-imp-correct*] *assms refine-IdD*
  by (*metis in-br-conv*)

Moving from Lists to Arrays

definition *win-count-imp2* :: ′*a Profile-Array* ⇒ ′*a* ⇒ *nat nres* **where**
win-count-imp2 p a ≡ *do* {
  (*i*, *ac*) ← *WHILET* (λ(*i*, -). *i* < *array-length p*) (λ(*i*, *ac*). *do* {
    ASSERT (*i* < *array-length p*);
    let ballot = (*p*[[*i*]]);
    let ac = *ac* + (*if* (*ballot*[[*0*]] = *a*) *then 1 else 0*);
    let i = *i* + *1*;
    RETURN (*i*, *ac*)
  })(*0*,*0*);
  RETURN ac
}

lemma *win-count-imp2-refine*:
  assumes (*pa*, *pl*) ∈ *br pa-to-pl* (*profile-a A*)
  shows *win-count-imp2 pa a* ≤ ⇓*Id* (*win-count-imp′ pl a*)
  unfolding *win-count-imp2-def win-count-imp′-def winsr-imp′-def*
  apply (*refine-rcg*)
  apply (*refine-dref-type*)
  apply (*simp-all*, *safe*)
proof (*simp-all*)
  fix *x1*
  assume *ir*: *x1* < *array-length pa*
  have *array-length pa* = *length* (*list-of-array pa*)
    by (*simp add*: *array-length-list*)
  from *assms ir this* **show** *x1* < *length pl* **unfolding** *pa-to-pl-def*
    by (*simp add*: *in-br-conv*)

next
  fix *x1*
  assume *ir*: $x1 < length\ pl$
  from *assms ir* **show** *g2*: $x1 < array\text{-}length\ pa$ **unfolding** *pa-to-pl-def*
    by (*simp add*: *array-length-list in-br-conv*)
next
  fix *x1*
  assume *ir*: $x1 < length\ pl$
  assume *afst*: $a = (pa[[x1]])[[0]]$
  from *ir* **have** *arrayac*: $(pa[[x1]])[[0]] = list\text{-}of\text{-}array((list\text{-}of\text{-}array\ pa)!x1)!0$
    by (*metis Diff-Array.array.exhaust array-get.simps list-of-array.simps*)
  from *assms ir arrayac* **show** $pl!\ x1\ !\ 0 = (pa[[x1]])[[0]]$
    unfolding *pa-to-pl-def well-formed-pl-def*
    by (*simp add*: *in-br-conv*)
next
  fix *x1*
  assume *ir*: $x1 < length\ pl$
  assume *neq*: $pa[[x1]][[0]] \neq pl\ !\ x1\ !\ 0$
  from *ir* **have** *arrayac*: $pa[[x1]][[0]] = list\text{-}of\text{-}array((list\text{-}of\text{-}array\ pa)!x1)!0$
    by (*metis Diff-Array.array.exhaust array-get.simps list-of-array.simps*)
  from *assms ir arrayac* **have** $pl!\ x1\ !\ 0 = pa[[x1]][[0]]$
    unfolding *pa-to-pl-def well-formed-pl-def*
    by (*simp add*: *in-br-conv*)
  from *neq this* **show** *False* **by** *simp*
qed

lemma *a-l-r-step*: $(pr1\text{-}\alpha \circ pa\text{-}to\text{-}pl) = pa\text{-}to\text{-}pr$
  by (*simp add*: *fun-comp-eq-conv pa-to-pr-def*)

lemma *win-count-imp2-correct*:
  assumes $(pa, pr) \in br\ pa\text{-}to\text{-}pr\ (profile\text{-}a\ A)$
  shows $win\text{-}count\text{-}imp2\ pa\ a \leq SPEC\ (\lambda ac.\ ac = win\text{-}count\ pr\ a)$
  using *ref-two-step*[*OF win-count-imp2-refine win-count-imp′-correct*]
proof −
  assume *td*: $\bigwedge pa\ pl\ A\ pr\ Aa\ a.$
        $(pa, pl) \in br\ pa\text{-}to\text{-}pl\ (profile\text{-}a\ A) \Longrightarrow$
        $(pl, pr) \in br\ pr1\text{-}\alpha\ (profile\text{-}l\ Aa) \Longrightarrow$
 $win\text{-}count\text{-}imp2\ pa\ a \leq \Downarrow nat\text{-}rel\ (SPEC\ (\lambda wc.\ wc = win\text{-}count\ pr\ a))$
  obtain *pl* **where** *r1*: $(pa, pl) \in br\ pa\text{-}to\text{-}pl\ (profile\text{-}a\ A)$
    by (*metis assms in-br-conv*)
  from *r1* **have** *r2*: $(pl, pr) \in br\ pr1\text{-}\alpha\ (profile\text{-}l\ A)$ **using** *a-l-r-step assms profile-a-l*
    by (*metis comp-def in-br-conv*)
  from *r1 r2 td* **show** *?thesis*
    using *assms*
   by (*metis conc-trans-additional*(*5*) *singleton-conv win-count-imp′-correct win-count-imp2-refine*)
qed

schematic-goal *wc-code-refine-aux*: $RETURN\ ?wc\text{-}code \leq win\text{-}count\text{-}imp2\ p\ a$

unfolding *win-count-imp2-def*
by (*refine-transfer*)

concrete-definition *win-count-imp-code* **for** p a **uses** *wc-code-refine-aux*

lemma *win-count-array*[*simp*]:
assumes *lg*: (*profile-a* A *pa*)
shows *win-count-imp-code pa* a = *win-count* (*pa-to-pr pa*) a
using *lg order-trans*[*OF win-count-imp-code.refine win-count-imp2-correct*,
of pa (*pa-to-pr pa*)]
by (*auto simp*: *refine-rel-defs*)

lemma *win-count-array-code-correct*:
assumes *lg*: (*profile-a* A *pa*)
shows *win-count* (*pa-to-pr pa*) a = *win-count-imp-code pa* a
by (*metis lg win-count-array*)

definition *prefer-count-invariant* $p\ x\ y \equiv \lambda(r,\ ac).$
$r \leq$ *length* $p \land$
$ac =$ *card* $\{i{::}nat.\ i < r \land (let\ r = (p!i)\ in\ (y \preceq_r x))\}$

definition *prefer-count-mon* :: *'a Profile* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *nat nres* **where**
prefer-count-mon $p\ x\ y \equiv$ *do* {
$(i,\ ac) \leftarrow$ *WHILET* $(\lambda(i,\ \text{-}).\ i <$ *length* $p)$ $(\lambda(i,\ ac).$ *do* {
ASSERT ($i <$ *length* p);
let $b = (p!i)$;
let $ac = ac + ($*if* $y \preceq_b x$ *then* 1 *else* 0$)$;
let $i = i + 1$;
RETURN (i, ac)
})(0,0);
RETURN ac
}

lemma *prefer-count-mon-correct*:
shows *prefer-count-mon* $p\ a\ b \leq$ *SPEC* (λ *wc*. *wc* = *prefer-count* $p\ a\ b$)
unfolding *prefer-count-mon-def prefer-count.simps*
apply (*intro WHILET-rule*[**where** I=(*prefer-count-invariant* $p\ a\ b$)
and R=*measure* $(\lambda(r,\text{-}).\ (length\ p) - r)$] *refine-vcg*)
unfolding *prefer-count-invariant-def*
apply (*simp-all*)
apply (*erule subst*)
apply (*simp*)
apply (*intro conjI impI*)
proof (*simp-all*)
fix r
assume *ir*: $r <$ *length* p
assume *blpa*: $(b,\ a) \in p!r$
with *blpa* **have** *prep*:

$\{i.\ i < Suc\ r \wedge (b, a) \in p\ !\ i\}$
$= \{i.\ i < r \wedge (b, a) \in p\ !\ i\} \cup \{r\}$
by *fastforce*
thus $Suc\ (card\ \{i.\ i < r \wedge (b, a) \in p\ !\ i\}) = card\ \{i.\ i < Suc\ r \wedge (b, a) \in p\ !\ i\}$
by *fastforce*
next
fix *r*
assume *ir*: $r < length\ p$
assume *bnlpa*: $(b, a) \notin p!r$
with *bnlpa ir* **show** *prep*:
$card\ \{i.\ i < r \wedge (b, a) \in p\ !\ i\}$
$= card\ \{i.\ i < Suc\ r \wedge (b, a) \in p\ !\ i\}$
using *less-Suc-eq* **by** *metis*
qed

end
theory *Result-Ref*
imports *Verified-Voting-Rule-Construction.Result*

begin

type-synonym $'a\ Result\text{-}Ref = 'a\ list * 'a\ list * 'a\ list$

end
theory *Electoral-Module-Ref*
imports *Social-Choice-Types/Profile-Array*
Social-Choice-Types/Result-Ref
Verified-Voting-Rule-Construction.Electoral-Module
begin

type-synonym $'a\ Electoral\text{-}Module\text{-}Ref = 'a\ set \Rightarrow 'a\ Profile\text{-}Array \Rightarrow 'a\ Result$

definition *electoral-module-r* :: $'a\ Electoral\text{-}Module\text{-}Ref \Rightarrow bool$ **where**
$electoral\text{-}module\text{-}r\ mr \equiv \forall\ A\ p.\ finite\text{-}profile\text{-}a\ A\ p \longrightarrow well\text{-}formed\ A\ (mr\ A\ p)$

end
theory *Plurality-Module-Ref*
imports *Verified-Voting-Rule-Construction.Electoral-Module*
Component-Types/Electoral-Module-Ref
Verified-Voting-Rule-Construction.Plurality-Module
begin

fun *plurality-r* :: *'a Electoral-Module-Ref* **where**
plurality-r A p =
({*a* ∈ *A*. ∀ *x* ∈ *A*. *win-count-imp-code p x* ≤ *win-count-imp-code p a*},
{*a* ∈ *A*. ∃ *x* ∈ *A*. *win-count-imp-code p x* > *win-count-imp-code p a*},
{})

lemma *datarefplurality*:
shows ∀ *A*. (*plurality-r A*, *plurality A*) ∈ (*br pa-to-pr* (*profile-a A*)) →
Id
apply (*refine-rcg*)
apply (*auto simp add*:
refine-rel-defs)
done

type-synonym *'a Electoral-Module-Ref-T* = *'a set* ⇒ *'a Profile-Array* ⇒ *'a Result-Ref nres*

definition *initmap* :: *'a set* ⇒ *'a* ⇀ *nat* **where**
initmap A = (*SOME m*. (∀ *a*∈*A*. ((*m a*) = *Some* (*0*::*nat*))))

definition *computewcforcands* :: *'a set* ⇒ *'a Profile-Array* ⇒ (*'a* ⇀ *nat*) *nres*
where
computewcforcands A p ≡ *do* {
(*i*, *wcmap*) ← *WHILET* (λ(*i*, -). *i* < *array-length p*) (λ(*i*, *wcmap*). *do* {
ASSERT (*i* < *array-length p*);
let ballot = (*p*[[*i*]]);
let winner = (*ballot*[[*0*]]);
let wcmap = (*if* (*wcmap winner* = *None*) *then wcmap* (*winner* ↦ *0*)
else wcmap (*winner* ↦ (*the* (*wcmap winner*) + *1*)));
let i = *i* + *1*;
RETURN (*i*, *wcmap*)
})(*0*, *Map.empty*);
RETURN wcmap
}

lemma *wcmap-correc* : **assumes** *profile-a A p*
shows *computewcforcands A p* ≤ *SPEC* (λ *m*. ∀ *a* ∈ *A*. (*the* (*m a*)) = *win-count-imp-code p a*)
unfolding *computewcforcands-def initmap-def*
apply (*intro WHILET-rule*[**where** *R*=*measure* (λ(*i*,-). (*array-length p*) − *i*)]
refine-vcg)
apply *auto*
oops

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.

[3] P. Lammich. Refinement for monadic programs. *Archive of Formal Proofs*, January 2012. https://isa-afp.org/entries/Refine_Monadic.html, Formal proof development.