The Incompatibility of Fishburn-Strategyproofness and Pareto-Efficiency

Felix Brandt, Manuel Eberl, Christian Saile, Christian Stricker

February 23, 2021

Abstract

This formalisation contains the proof that there is no anonymous Social Choice Function for at least three agents and alternatives that satisfies both Pareto-Efficiency and Fishburn-Strategyproofness. It was derived from a proof of Brandt *et al.* [1], which relies on an unverified translation of a fixed finite instance of the original problem to SAT. This Isabelle proof contains a machine-checked version of both the statement for exactly three agents and alternatives and the lifting to the general case.

Contents

1 Social Choice Functions

theory *Social-Choice-Functions*
imports
Randomised-Social-Choice.Preference-Profile-Cmd
begin

1.1 Weighted majority graphs

definition *supporters* :: ('*agent*, '*alt*) *pref-profile* ⇒ '*alt* ⇒ '*alt* ⇒ '*agent set* **where**
supporters-auxdef: *supporters R x y* = {*i*. *x* ≻[*R i*] *y*}

definition *weighted-majority* :: ('*agent*, '*alt*) *pref-profile* ⇒ '*alt* ⇒ '*alt* ⇒ *int* **where**
weighted-majority R x y = *int* (*card* (*supporters R x y*)) − *int* (*card* (*supporters R y x*))

lemma *weighted-majority-refl* [*simp*]: *weighted-majority R x x* = *0*
by (*simp add*: *weighted-majority-def*)

lemma *weighted-majority-swap*: *weighted-majority R x y* = −*weighted-majority R y x*
by (*simp add*: *weighted-majority-def*)

lemma *eval-set-filter*:
Set.filter P {} = {}
P x ⟹ *Set.filter P* (*insert x A*) = *insert x* (*Set.filter P A*)
¬*P x* ⟹ *Set.filter P* (*insert x A*) = *Set.filter P A*
unfolding *Set.filter-def* **by** *auto*

context *election*
begin

lemma *supporters-def*:
assumes *is-pref-profile R*
shows *supporters R x y* = {*i*∈*agents*. *x* ≻[*R i*] *y*}
proof −
interpret *pref-profile-wf agents alts R* **by** *fact*
show *?thesis* **using** *not-outside* **unfolding** *supporters-auxdef* **by** *blast*
qed

lemma *supporters-nonagent1*:
assumes *is-pref-profile R x* ∉ *alts*
shows *supporters R x y* = {}
proof −
interpret *pref-profile-wf agents alts R* **by** *fact*
from *assms* **show** *?thesis* **by** (*auto simp*: *supporters-def dest*: *not-outside*)
qed

lemma *supporters-nonagent2*:
assumes *is-pref-profile R y* ∉ *alts*

shows *supporters R x y = {}*
proof −
interpret *pref-profile-wf agents alts R* **by** *fact*
from *assms* **show** *?thesis* **by** (*auto simp*: *supporters-def dest*: *not-outside*)
qed

lemma *weighted-majority-nonagent1*:
assumes *is-pref-profile R x ∉ alts*
shows *weighted-majority R x y = 0*
using *assms* **by** (*simp add*: *weighted-majority-def supporters-nonagent1 supporters-nonagent2*)

lemma *weighted-majority-nonagent2*:
assumes *is-pref-profile R y ∉ alts*
shows *weighted-majority R x y = 0*
using *assms* **by** (*simp add*: *weighted-majority-def supporters-nonagent1 supporters-nonagent2*)

lemma *weighted-majority-eq-iff*:
assumes *is-pref-profile R1 is-pref-profile R2*
shows *weighted-majority R1 = weighted-majority R2 ⟷*
(∀ x∈alts. ∀ y∈alts. weighted-majority R1 x y = weighted-majority R2 x y)
proof (*intro iffI ext*)
fix *x y* :: *'alt*
assume *∀ x∈alts. ∀ y∈alts. weighted-majority R1 x y = weighted-majority R2 x y*
with *assms* **show** *weighted-majority R1 x y = weighted-majority R2 x y*
by (*cases x ∈ alts*; *cases y ∈ alts*)
(*auto simp*: *fun-eq-iff weighted-majority-nonagent1 weighted-majority-nonagent2*)
qed *auto*

end

1.2 Definition of Social Choice Functions

locale *social-choice-function = election agents alts*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* +
fixes *scf* :: (*'agent*, *'alt*) *pref-profile ⇒ 'alt set*
assumes *scf-nonempty*: *is-pref-profile R ⟹ scf R ≠ {}*
assumes *scf-alts*: *is-pref-profile R ⟹ scf R ⊆ alts*

1.3 Anonymity

An SCF is anonymous if permuting the agents in the input does not change the result.

locale *anonymous-scf = social-choice-function agents alts scf*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf* +
assumes *anonymous*: *π permutes agents ⟹ is-pref-profile R ⟹ scf (R ∘ π) = scf R*

begin

lemma *anonymous′*:
assumes *anonymous-profile R1 = anonymous-profile R2*
assumes *is-pref-profile R1 is-pref-profile R2*
shows *scf R1 = scf R2*
proof −
from *anonymous-profile-agent-permutation*[*OF assms finite-agents*]
obtain π **where** π *permutes agents R1 = R2* ∘ π **by** *blast*
with *anonymous*[*of* π *R2*] *assms* **show** *?thesis* **by** *simp*
qed

lemma *anonymity-prefs-from-table*:
assumes *prefs-from-table-wf agents alts xs prefs-from-table-wf agents alts ys*
assumes *mset* (*map snd xs*) = *mset* (*map snd ys*)
shows *scf* (*prefs-from-table xs*) = *scf* (*prefs-from-table ys*)
proof −
from *prefs-from-table-agent-permutation*[*OF assms*] **guess** π **.**
with *anonymous*[*of* π, *of prefs-from-table xs*] *assms*(*1*) **show** *?thesis*
by (*simp add*: *pref-profile-from-tableI*)
qed

context
begin
qualified lemma *anonymity-prefs-from-table-aux*:
assumes *R1 = prefs-from-table xs prefs-from-table-wf agents alts xs*
assumes *R2 = prefs-from-table ys prefs-from-table-wf agents alts ys*
assumes *mset* (*map snd xs*) = *mset* (*map snd ys*)
shows *scf R1 = scf R2* **unfolding** *assms*(*1,3*)
by (*rule anonymity-prefs-from-table*) (*simp-all add*: *assms del*: *mset-map*)
end

end

1.4 Neutrality

An SCF is neutral if permuting the alternatives in the input does not change the result, modulo the equivalent permutation in the output lottery.

locale *neutral-scf* = *social-choice-function agents alts scf*
for *agents* :: *′agent set* **and** *alts* :: *′alt set* **and** *scf* +
assumes *neutral*: σ *permutes alts* ⟹ *is-pref-profile R* ⟹
scf (*permute-profile* σ *R*) = σ ‘ *scf R*
begin

Alternative formulation of neutrality that shows that our definition is equivalent to that in the paper.

lemma *neutral′*:
assumes σ *permutes alts*

assumes *is-pref-profile R*
assumes $a \in alts$
shows $\sigma\ a \in scf\ (permute\text{-}profile\ \sigma\ R) \longleftrightarrow a \in scf\ R$
proof −
have $*$: $x = y$ **if** $\sigma\ x = \sigma\ y$ **for** $x\ y$
using *permutes-inj*[*OF assms*(*1*)] *that* **by** (*auto dest*: *injD*)
from *assms* **show** *?thesis* **by** (*auto simp*: *neutral dest*!: $*$)
qed

end

locale *an-scf* =
anonymous-scf agents alts scf + *neutral-scf agents alts scf*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf*
begin

lemma *scf-anonymous-neutral*:
assumes *perm*: σ *permutes alts* **and** *wf*: *is-pref-profile R1 is-pref-profile R2*
assumes *eq*: *anonymous-profile R1* =
image-mset (*map* ($\lambda A.\ \sigma$ ‘ A)) (*anonymous-profile R2*)
shows *scf R1* = σ ‘ *scf R2*
proof −
interpret *R1*: *pref-profile-wf agents alts R1* **by** *fact*
interpret *R2*: *pref-profile-wf agents alts R2* **by** *fact*
from *perm* **have** *wf'*: *is-pref-profile* (*permute-profile* σ *R2*)
by (*rule R2.wf-permute-alts*)
from *eq perm* **have** *anonymous-profile R1* = *anonymous-profile* (*permute-profile* σ *R2*)
by (*simp add*: *R2.anonymous-profile-permute*)
from *anonymous-profile-agent-permutation*[*OF this wf*(*1*) *wf'*]
obtain π **where** π *permutes agents permute-profile* σ *R2* $\circ\ \pi$ = *R1* **by** *auto*
have *scf* (*permute-profile* σ *R2* $\circ\ \pi$) = *scf* (*permute-profile* σ *R2*)
by (*rule anonymous*) *fact*+
also have ... = σ ‘ *scf R2*
by (*rule neutral*) *fact*+
also have *permute-profile* σ *R2* $\circ\ \pi$ = *R1* **by** *fact*
finally show *?thesis* **.**
qed

lemma *scf-anonymous-neutral'*:
assumes *perm*: σ *permutes alts* **and** *wf*: *is-pref-profile R1 is-pref-profile R2*
assumes *eq*: *anonymous-profile R1* =
image-mset (*map* ($\lambda A.\ \sigma$ ‘ A)) (*anonymous-profile R2*)
shows $\sigma\ x \in scf\ R1 \longleftrightarrow x \in scf\ R2$
proof −
have *scf R1* = σ ‘ *scf R2* **by** (*intro scf-anonymous-neutral*) *fact*+
also have $\sigma\ x \in \ldots \longleftrightarrow x \in scf\ R2$

by (*blast dest*: *injD*[*OF permutes-inj*[*OF perm*]])
finally show *?thesis* **.**
qed

lemma *scf-automorphism*:
assumes *perm*: σ *permutes alts* **and** *wf*: *is-pref-profile R*
assumes *eq*: *image-mset* (*map* ($\lambda A.$ σ ‘ *A*)) (*anonymous-profile R*) = *anonymous-profile R*
shows σ ‘ *scf R* = *scf R*
using *scf-anonymous-neutral*[*OF perm wf wf eq* [*symmetric*]] **..**

end

lemma *an-scf-automorphism-aux*:
assumes *wf*: *prefs-from-table-wf agents alts yss R* ≡ *prefs-from-table yss*
assumes *an*: *an-scf agents alts scf*
assumes *eq*: *mset* (*map* ((*map* ($\lambda A.$ *permutation-of-list xs* ‘ *A*)) ∘ *snd*) *yss*) = *mset* (*map snd yss*)
assumes *perm*: *set* (*map fst xs*) ⊆ *alts set* (*map snd xs*) = *set* (*map fst xs*)
distinct (*map fst xs*)
and *x*: *x* ∈ *alts y* = *permutation-of-list xs x*
shows *x* ∈ *scf R* ⟷ *y* ∈ *scf R*
proof −
note *perm* = *list-permutesI*[*OF perm*]
let *?σ* = *permutation-of-list xs*
note *perm′* = *permutation-of-list-permutes* [*OF perm*]
from *wf* **have** *wf′*: *pref-profile-wf agents alts R* **by** (*simp add*: *pref-profile-from-tableI*)
then interpret *R*: *pref-profile-wf agents alts R* **.**
from *perm′* **interpret** *R′*: *pref-profile-wf agents alts permute-profile ?σ R*
by (*simp add*: *R.wf-permute-alts*)
from *an* **interpret** *an-scf agents alts scf* **.**

from *eq wf* **have** *eq′*: *image-mset* (*map* ($\lambda A.$ *?σ* ‘ *A*)) (*anonymous-profile R*) = *anonymous-profile R*
by (*simp add*: *anonymise-prefs-from-table mset-map multiset.map-comp*)
have *x* ∈ *scf R* ⟷ *?σ x* ∈ *?σ* ‘ *scf R*
by (*blast dest*: *injD*[*OF permutes-inj*[*OF perm′*]])
also from *eq′ x wf′ perm′* **have** *?σ* ‘ *scf R* = *scf R*
by (*intro scf-automorphism*)
(*simp-all add*: *R.anonymous-profile-permute pref-profile-from-tableI*)
finally show *?thesis* **using** *x* **by** *simp*
qed

1.5 Weighted-majoritarian SCFs

locale *pairwise-scf* = *social-choice-function agents alts scf*
for *agents* :: *′agent set* **and** *alts* :: *′alt set* **and** *scf* +
assumes *pairwise*:
is-pref-profile R1 ⟹ *is-pref-profile R2* ⟹ *weighted-majority R1* = *weighted-majority*

R2 $\Longrightarrow$
scf R1 = scf R2

1.6 Pareto efficiency

locale *pareto-efficient-scf = social-choice-function agents alts scf*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf* +
assumes *pareto-efficient*:
is-pref-profile R $\Longrightarrow$ *scf R* $\cap$ *pareto-losers R* = {}
begin

lemma *pareto-efficient'*:
assumes *is-pref-profile R y* $\succ$[*Pareto*(*R*)] *x*
shows *x* $\notin$ *scf R*
using *pareto-efficient*[*of R*] *assms*
by (*auto simp*: *pareto-losers-def*)

lemma *pareto-efficient''*:
assumes *is-pref-profile R i* $\in$ *agents* $\forall$ *i*$\in$*agents*. *y* $\succeq$[*R i*] *x* $\neg$*y* $\preceq$[*R i*] *x*
shows *x* $\notin$ *scf R*
proof −
from *assms*(*1*) **interpret** *pref-profile-wf agents alts R* **.**
from *assms*(*2*−) **show** *?thesis*
by (*intro pareto-efficient'*[*OF assms*(*1*), *of - y*])
(*auto simp*: *Pareto-iff strongly-preferred-def*)
qed

end

1.7 Set extensions

type-synonym *'alt set-extension* = *'alt relation* $\Rightarrow$ *'alt set relation*

definition *Kelly* :: *'alt set-extension* **where**
A $\succeq$[*Kelly*(*R*)] *B* $\longleftrightarrow$ ($\forall$ *a*$\in$*A*. $\forall$ *b*$\in$*B*. *a* $\succeq$[*R*] *b*)

lemma *Kelly-strict-iff*: *A* $\succ$[*Kelly*(*R*)] *B* $\longleftrightarrow$ (($\forall$ *a*$\in$*A*. $\forall$ *b*$\in$*B*. *R b a*) $\wedge$ $\neg$ ($\forall$ *a*$\in$*B*. $\forall$ *b*$\in$*A*. *R b a*))
unfolding *strongly-preferred-def Kelly-def* **..**

lemmas *Kelly-eval* = *Kelly-def Kelly-strict-iff*

definition *Fishb* :: *'alt set-extension* **where**
A $\succeq$[*Fishb*(*R*)] *B* $\longleftrightarrow$ ($\forall$ *a*$\in$*A*. $\forall$ *b*$\in$*B*−*A*. *a* $\succeq$[*R*] *b*) $\wedge$ ($\forall$ *a*$\in$*A*−*B*. $\forall$ *b*$\in$*B*. *a* $\succeq$[*R*] *b*)

lemma *Fishb-strict-iff*:
A $\succ$[*Fishb*(*R*)] *B* $\longleftrightarrow$
(($\forall$ *a*$\in$*A*. $\forall$ *b*$\in$*B* − *A*. *R b a*) $\wedge$ ($\forall$ *a*$\in$*A* − *B*. $\forall$ *b*$\in$*B*. *R b a*)) $\wedge$
$\neg$ (($\forall$ *a*$\in$*B*. $\forall$ *b*$\in$*A* − *B*. *R b a*) $\wedge$ ($\forall$ *a*$\in$*B* − *A*. $\forall$ *b*$\in$*A*. *R b a*))

unfolding *strongly-preferred-def Fishb-def* **..**

lemmas *Fishb-eval* = *Fishb-def Fishb-strict-iff*

1.8 Strategyproofness

locale *strategyproof-scf* =
social-choice-function agents alts scf
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf* +
fixes *set-ext* :: *'alt set-extension*
assumes *strategyproof*:
is-pref-profile R ⟹ *total-preorder-on alts Ri'* ⟹ *i* ∈ *agents* ⟹
¬ *scf* (*R*(*i* := *Ri'*)) ≻[*set-ext*(*R i*)] *scf R*

locale *strategyproof-anonymous-scf* =
anonymous-scf agents alts scf + *strategyproof-scf agents alts scf set-ext*
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf* **and** *set-ext*
begin

lemma *strategyproof'*:
assumes *is-pref-profile R1 is-pref-profile R2 i* ∈ *agents j* ∈ *agents*
assumes *anonymous-profile R2* = *anonymous-profile R1* −
{#*weak-ranking* (*R1 i*)#} + {#*weak-ranking* (*R2 j*)#}
shows ¬*scf R2* ≻[*set-ext* (*R1 i*)] *scf R1*
proof −
let *?R3* = *R1*(*i* := *R2 j*)
interpret *R1*: *pref-profile-wf agents alts R1* **by** *fact*
interpret *R2*: *pref-profile-wf agents alts R2* **by** *fact*
from ‹*j* ∈ *agents*› **have** *total-preorder-on alts* (*R2 j*) **by** *simp*
from *strategyproof*[*OF assms*(*1*) *this* ‹*i* ∈ *agents*›]
have ¬*scf ?R3* ≻[*set-ext* (*R1 i*)] *scf R1* **.**
also from *assms* **have** *scf ?R3* = *scf R2*
by (*intro anonymous'*) (*simp-all add*: *R1.anonymous-profile-update*)
finally show *?thesis* **.**
qed

end

context *preorder-on*
begin

lemma *strict-not-outside*:
assumes *x* ≺[*le*] *y*
shows *x* ∈ *carrier y* ∈ *carrier*
using *assms not-outside*[*of x y*] **unfolding** *strongly-preferred-def* **by** *blast*+

end

1.9 Lifting preferences

Preference profiles can be lifted to a setting with more agents and alternatives by padding them with dummy agents and alternatives in such a way that every original agent prefers and original alternative over any dummy alternative and is indifferent between the dummy alternatives. Moreover, every dummy agent is completely indifferent.

definition *lift-prefs* ::
  'alt set ⇒ *'alt set* ⇒ *'alt relation* ⇒ *'alt relation* **where**
 lift-prefs alts alts' R = (λ*x y*.
   x ∈ *alts'* ∧ *y* ∈ *alts'* ∧ (*x* = *y* ∨ *x* ∉ *alts* ∨ (*y* ∈ *alts* ∧ *R x y*)))

lemma *lift-prefs-wf*:
 assumes *total-preorder-on alts R alts* ⊆ *alts'*
 shows *total-preorder-on alts' (lift-prefs alts alts' R)*
proof −
 interpret *R*: *total-preorder-on alts R* **by** *fact*
 show *?thesis*
  by *standard* (*insert R.total*, *auto dest*: *R.trans simp*: *lift-prefs-def*)
qed

definition *lift-pref-profile* ::
  'agent set ⇒ *'alt set* ⇒ *'agent set* ⇒ *'alt set* ⇒
    (*'agent*, *'alt*) *pref-profile* ⇒ (*'agent*, *'alt*) *pref-profile* **where**
 lift-pref-profile agents alts agents' alts' R = (λ*i x y*.
   x ∈ *alts'* ∧ *y* ∈ *alts'* ∧ *i* ∈ *agents'* ∧
   (*x* = *y* ∨ *x* ∉ *alts* ∨ *i* ∉ *agents* ∨ (*y* ∈ *alts* ∧ *R i x y*)))

lemma *lift-pref-profile-conv-vector*:
 assumes *i* ∈ *agents i* ∈ *agents'*
 shows *lift-pref-profile agents alts agents' alts' R i = lift-prefs alts alts' (R i)*
 using *assms* **by** (*auto simp*: *lift-pref-profile-def lift-prefs-def*)

lemma *lift-pref-profile-wf*:
 assumes *pref-profile-wf agents alts R*
 assumes *agents* ⊆ *agents' alts* ⊆ *alts' finite alts'*
 defines *R'* ≡ *lift-pref-profile agents alts agents' alts' R*
 shows *pref-profile-wf agents' alts' R'*
proof −
 from *assms* **interpret** *R*: *pref-profile-wf agents alts* **by** *simp*
 have *finite-total-preorder-on alts' (R' i)*
  if *i*: *i* ∈ *agents'* **for** *i*
 proof (*cases i* ∈ *agents*)
  case *True*
  then interpret *finite-total-preorder-on alts R i* **by** *simp*
  from *True assms* **show** *?thesis*
   by *unfold-locales* (*auto simp*: *lift-pref-profile-def dest*: *total intro*: *trans*)
 next

```
    case False
    with assms i show ?thesis
      by unfold-locales (simp-all add: lift-pref-profile-def)
  qed
  moreover have R' i = (λ- -. False) if i ∉ agents' for i
    unfolding lift-pref-profile-def R'-def using that by simp
  ultimately show ?thesis unfolding pref-profile-wf-def using assms by auto
qed

lemma lift-pref-profile-permute-agents:
  assumes π permutes agents agents ⊆ agents'
  shows   lift-pref-profile agents alts agents' alts' (R ∘ π) =
          lift-pref-profile agents alts agents' alts' R ∘ π
  using assms permutes-subset[OF assms]
  by (auto simp add: lift-pref-profile-def o-def permutes-in-image)

lemma lift-pref-profile-permute-alts:
  assumes σ permutes alts alts ⊆ alts'
  shows   lift-pref-profile agents alts agents' alts' (permute-profile σ R) =
          permute-profile σ (lift-pref-profile agents alts agents' alts' R)
proof −
  from assms have inv: inv σ permutes alts by (intro permutes-inv)
  from this assms(2) have inv σ permutes alts' by (rule permutes-subset)
  with inv show ?thesis using assms permutes-inj[OF ‹inv σ permutes alts›]
    by (fastforce simp add: lift-pref-profile-def permutes-in-image
        permute-profile-def fun-eq-iff dest: injD)
qed


context
  fixes agents alts R agents' alts' R'
  assumes R-wf: pref-profile-wf agents alts R
  assumes election: agents ⊆ agents' alts ⊆ alts' alts ≠ {} agents ≠ {} finite alts'
  defines R' ≡ lift-pref-profile agents alts agents' alts' R
begin

interpretation R: pref-profile-wf agents alts R by fact
interpretation R': pref-profile-wf agents' alts' R'
  using election R-wf by (simp add: R'-def lift-pref-profile-wf)

lemma lift-pref-profile-strict-iff:
  x ≺[lift-pref-profile agents alts agents' alts' R i] y ⟷
    i ∈ agents ∧ ((y ∈ alts ∧ x ∈ alts' − alts) ∨ x ≺[R i] y)
proof (cases i ∈ agents)
  case True
  then interpret total-preorder-on alts R i by simp
  show ?thesis using not-outside election
    by (auto simp: lift-pref-profile-def strongly-preferred-def)
qed (simp-all add: lift-pref-profile-def strongly-preferred-def)
```

lemma *preferred-alts-lift-pref-profile*:
assumes *i*: $i \in agents'$ **and** *x*: $x \in alts'$
shows $preferred\text{-}alts\ (R'\ i)\ x =$
$(if\ i \in agents \wedge x \in alts\ then\ preferred\text{-}alts\ (R\ i)\ x\ else\ alts')$
proof (*cases* $i \in agents$)
assume *i*: $i \in agents$
then interpret *Ri*: *total-preorder-on alts R i* **by** *simp*
show *?thesis*
using *i x election Ri.not-outside*
by (*auto simp*: *preferred-alts-def R'-def lift-pref-profile-def Ri.refl*)
qed (*auto simp*: *preferred-alts-def R'-def lift-pref-profile-def i x*)

lemma *lift-pref-profile-Pareto-iff*:
$x \preceq[Pareto(R')]\ y \longleftrightarrow x \in alts' \wedge y \in alts' \wedge (x \notin alts \vee x \preceq[Pareto(R)]\ y)$
proof −
from *R.nonempty-agents* **obtain** *i* **where** *i*: $i \in agents$ **by** *blast*
then interpret *Ri*: *finite-total-preorder-on alts R i* **by** *simp*
show *?thesis* **unfolding** *R'.Pareto-iff R.Pareto-iff* **unfolding** *R'-def lift-pref-profile-def*
using *election i* **by** (*auto simp*: *preorder-on.refl*[*OF R.in-dom*]
simp del: *R.nonempty-alts R.nonempty-agents intro*: *Ri.not-outside*)
qed

lemma *lift-pref-profile-Pareto-strict-iff*:
$x \prec[Pareto(R')]\ y \longleftrightarrow x \in alts' \wedge y \in alts' \wedge (x \notin alts \wedge y \in alts \vee x \prec[Pareto(R)]\ y)$
by (*auto simp*: *strongly-preferred-def lift-pref-profile-Pareto-iff R.Pareto.not-outside*)

lemma *pareto-losers-lift-pref-profile*:
shows $pareto\text{-}losers\ R' = pareto\text{-}losers\ R \cup (alts' - alts)$
proof −
have *A*: $x \in alts\ y \in alts$ **if** $x \prec[Pareto(R)]\ y$ **for** *x y*
using *that R.Pareto.not-outside* **unfolding** *strongly-preferred-def* **by** *auto*
have *B*: $x \in alts'$ **if** $x \in alts$ **for** *x* **using** *election that* **by** *blast*
from *R.nonempty-alts* **obtain** *x* **where** *x*: $x \in alts$ **by** *blast*
thus *?thesis* **unfolding** *pareto-losers-def lift-pref-profile-Pareto-strict-iff* [*abs-def*]
by (*auto dest*: *A B*)
qed

end

1.10 Lowering SCFs

Using the preference lifting, we can now *lower* an SCF to a setting with fewer agents and alternatives under mild conditions to the original SCF. This preserves many nice properties, such as anonymity, neutrality, and strategyproofness.

locale *scf-lowering* =

pareto-efficient-scf agents alts scf
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf* +
fixes *agents' alts'*
assumes *agents'-subset*: *agents'* ⊆ *agents* **and** *alts'-subset*: *alts'* ⊆ *alts*
and *agents'-nonempty* [*simp*]: *agents'* ≠ {} **and** *alts'-nonempty* [*simp*]: *alts'* ≠ {}
begin

lemma *finite-agents'* [*simp*]: *finite agents'*
using *agents'-subset finite-agents* **by** (*rule finite-subset*)

lemma *finite-alts'* [*simp*]: *finite alts'*
using *alts'-subset finite-alts* **by** (*rule finite-subset*)

abbreviation *lift* :: (*'agent*, *'alt*) *pref-profile* ⇒ (*'agent*, *'alt*) *pref-profile* **where**
lift ≡ *lift-pref-profile agents' alts' agents alts*

definition *lowered* :: (*'agent*, *'alt*) *pref-profile* ⇒ *'alt set* **where**
lowered = *scf* ∘ *lift*

lemma *lift-wf* [*simp*, *intro*]:
pref-profile-wf agents' alts' R ⟹ *is-pref-profile* (*lift R*)
using *alts'-subset agents'-subset* **by** (*intro lift-pref-profile-wf*) *simp-all*

sublocale *lowered*: *election agents' alts'*
by *unfold-locales simp-all*

lemma *preferred-alts-lift*:
lowered.is-pref-profile R ⟹ *i* ∈ *agents* ⟹ *x* ∈ *alts* ⟹
preferred-alts (*lift R i*) *x* =
(*if i* ∈ *agents'* ∧ *x* ∈ *alts' then preferred-alts* (*R i*) *x else alts*)
using *alts'-subset agents'-subset*
by (*intro preferred-alts-lift-pref-profile*) *simp-all*

lemma *pareto-losers-lift*:
lowered.is-pref-profile R ⟹ *pareto-losers* (*lift R*) = *pareto-losers R* ∪ (*alts* − *alts'*)
using *agents'-subset alts'-subset* **by** (*intro pareto-losers-lift-pref-profile*) *simp-all*

sublocale *lowered*: *social-choice-function agents' alts' lowered*
proof
fix *R* **assume** *R-wf*: *pref-profile-wf agents' alts' R*
from *R-wf* **have** *R'-wf*: *pref-profile-wf agents alts* (*lift R*) **by** (*rule lift-wf*)
show *lowered R* ⊆ *alts'*
proof *safe*
fix *x* **assume** *x* ∈ *lowered R*
hence *x* ∈ *scf* (*lift R*) **by** (*auto simp*: *o-def lowered-def*)
moreover have *scf* (*lift R*) ∩ *pareto-losers* (*lift R*) = {}

```
    by (intro pareto-efficient R'-wf)
  ultimately show x ∈ alts' using scf-alts[of lift R]
    by (auto simp: pareto-losers-lift R-wf R'-wf)
 qed
 show lowered R ≠ {}
  using R'-wf by (auto simp: lowered-def scf-nonempty)
qed

sublocale lowered: pareto-efficient-scf agents' alts' lowered
proof
 fix R assume R-wf: pref-profile-wf agents' alts' R
 from R-wf have R'-wf: pref-profile-wf agents alts (lift R) by (rule lift-wf)
 have lowered R ∩ pareto-losers (lift R) = {} unfolding lowered-def o-def
  by (intro pareto-efficient R'-wf)
 with R-wf show lowered R ∩ pareto-losers R = {}
  by (auto simp: pareto-losers-lift)
qed

end


locale scf-lowering-anonymous =
 anonymous-scf agents alts scf +
 scf-lowering agents alts scf agents' alts'
 for agents :: 'agent set and alts :: 'alt set and scf agents' alts'
begin

sublocale lowered: anonymous-scf agents' alts' lowered
proof
 fix π R
 assume π permutes agents' and lowered.is-pref-profile R
 thus lowered (R ∘ π) = lowered R using agents'-subset
    by (auto simp: lowered-def lift-pref-profile-permute-agents anonymous per-
mutes-subset)
qed

end


locale scf-lowering-neutral =
 neutral-scf agents alts scf +
 scf-lowering agents alts scf agents' alts'
 for agents :: 'agent set and alts :: 'alt set and scf agents' alts'
begin

sublocale lowered: neutral-scf agents' alts' lowered
proof
 fix σ R
 assume σ permutes alts' and lowered.is-pref-profile R
```

thus *lowered* (*permute-profile* σ *R*) = σ ‘ *lowered R* **using** *alts′-subset*
by (*auto simp*: *lowered-def lift-pref-profile-permute-alts neutral permutes-subset*)
qed

end

The following is a technical condition that we need from a set extensions in order for strategyproofness to survive the lowering. The condition could probably be weakened a bit, but it is good enough for our purposes the way it is.

locale *liftable-set-extension* =
fixes *alts′ alts* :: *′alt set* **and** *set-ext* :: *′alt relation* ⇒ *′alt set relation*
assumes *set-ext-strong-lift*:
total-preorder-on alts′ R ⟹ *A* ≠ {} ⟹ *B* ≠ {} ⟹ *A* ⊆ *alts′* ⟹ *B* ⊆ *alts′* ⟹
A ≺[*set-ext R*] *B* ⟹ *A* ≺[*set-ext* (*lift-prefs alts′ alts R*)] *B*

lemma *liftable-set-extensionI-weak*:
assumes ⋀*R A B*. *total-preorder-on alts′ R* ⟹ *A* ≠ {} ⟹ *B* ≠ {} ⟹
A ⊆ *alts′* ⟹ *B* ⊆ *alts′* ⟹
A ⪯[*set-ext R*] *B* ⟷ *A* ⪯[*set-ext* (*lift-prefs alts′ alts R*)] *B*
shows *liftable-set-extension alts′ alts set-ext*
proof (*standard*, *goal-cases*)
case (*1 R A B*)
from *assms*[*of R A B*] **and** *assms*[*of R B A*] **and** *1* **show** *?case*
by (*auto simp*: *strongly-preferred-def*)
qed

lemma *Kelly-liftable*:
assumes *alts′* ⊆ *alts*
shows *liftable-set-extension alts′ alts Kelly*
proof (*rule liftable-set-extensionI-weak*, *goal-cases*)
case (*1 R A B*)
interpret *R*: *total-preorder-on alts′ R* **by** *fact*
from *1*(*2−5*) **show** *?case* **using** *assms R.refl*
by (*force simp*: *Kelly-def lift-prefs-def*)
qed

lemma *Fishburn-liftable*:
assumes *alts′* ⊆ *alts*
shows *liftable-set-extension alts′ alts Fishb*
proof (*rule liftable-set-extensionI-weak*, *goal-cases*)
case (*1 R A B*)
interpret *R*: *total-preorder-on alts′ R* **by** *fact*
have *conj-cong*: *P1* ∧ *Q1* ⟷ *P2* ∧ *Q2* **if** *P1* ⟷ *P2 Q1* ⟷ *Q2* **for** *P1 P2 Q1 Q2*
using *that* **by** *blast*
from *1*(*2−5*) **show** *?case* **using** *assms*
unfolding *Fishb-def lift-prefs-def* **by** (*intro conj-cong ball-cong refl*) *auto*

qed

locale *scf-lowering-strategyproof* =
strategyproof-scf agents alts scf set-ext +
liftable-set-extension alts′ alts set-ext +
scf-lowering agents alts scf agents′ alts′
for *agents* :: *′agent set* **and** *alts* :: *′alt set* **and** *scf agents′ alts′ set-ext*
begin

sublocale *lowered*: *strategyproof-scf agents′ alts′ lowered*
proof
fix *R Ri′ i*
assume *R-wf*: *lowered.is-pref-profile R* **and** *Ri′-wf*: *total-preorder-on alts′ Ri′*
and *i*: *i* ∈ *agents′*
interpret *R*: *pref-profile-wf agents′ alts′ R* **by** *fact*
interpret *Ri′*: *total-preorder-on alts′ Ri′* **by** *fact*
from *R-wf* **have** *R′-wf*: *is-pref-profile* (*lift R*) **by** (*rule lift-wf*)

— We lift the alternative preference for the agent i in R to preferences in the lifted profile.
define *Ri″* **where** *Ri″* = *lift-prefs alts′ alts Ri′*

have ¬*scf* (*lift R*) ≺[*set-ext* (*lift R i*)] *scf* ((*lift R*)(*i* := *Ri″*))
using *i agents′-subset alts′-subset* **unfolding** *Ri″-def*
by (*intro strategyproof R′-wf Ri′-wf lift-prefs-wf*) *auto*
also have (*lift R*)(*i* := *Ri″*) = *lift* (*R*(*i* := *Ri′*)) **using** *i agents′-subset*
by (*auto simp*: *fun-eq-iff Ri″-def lift-pref-profile-def lift-prefs-def*)
finally have *not-less*: ¬*scf* (*lift R*) ≺[*set-ext* (*lift R i*)] *scf* (*lift* (*R*(*i* := *Ri′*))) **.**

show ¬*lowered R* ≺[*set-ext* (*R i*)] *lowered* (*R*(*i* := *Ri′*))
proof
assume *lowered R* ≺[*set-ext* (*R i*)] *lowered* (*R*(*i* := *Ri′*))
hence *lowered R* ≺[*set-ext* (*lift-prefs alts′ alts* (*R i*))] *lowered* (*R*(*i* := *Ri′*))
by (*intro set-ext-strong-lift R.prefs-wf′*(*1*) *i lowered.scf-nonempty lowered.scf-alts*
R.wf-update R-wf Ri′-wf)
also have *lift-prefs alts′ alts* (*R i*) = *lift R i*
using *agents′-subset i* **by** (*subst lift-pref-profile-conv-vector*) *auto*
finally show *False* **using** *not-less* **unfolding** *lowered-def o-def* **by** *contradiction*
qed
qed

end

end

2 Main impossibility result

theory *Fishburn-Impossibility*
imports

Social-Choice-Functions
begin

2.1 Setting of the base case

Suppose we have an anonymous, Fishburn-strategyproof, and Pareto-efficient SCF for three agents $A1$ to $A3$ and three alternatives a, b, and c. We will derive a contradiction from this.

locale *fb-impossibility-3-3* =
strategyproof-anonymous-scf agents alts scf Fishb +
pareto-efficient-scf agents alts scf
for *agents* :: *'agent set* **and** *alts* :: *'alt set* **and** *scf A1 A2 A3 a b c* +
assumes *agents-eq*: *agents* = {*A1*, *A2*, *A3*}
assumes *alts-eq*: *alts* = {*a*, *b*, *c*}
assumes *distinct-agents*: *distinct* [*A1*, *A2*, *A3*]
assumes *distinct-alts*: *distinct* [*a*, *b*, *c*]
begin

We first give some simple rules that will allow us to break down the strategyproofness and support conditions more easily later.

lemma *agents-neq* [*simp*]: *A1* ≠ *A2 A2* ≠ *A1 A1* ≠ *A3 A3* ≠ *A1 A2* ≠ *A3 A3* ≠ *A2*
using *distinct-agents* **by** *auto*

lemma *alts-neq* [*simp*]: *a* ≠ *b a* ≠ *c b* ≠ *c b* ≠ *a c* ≠ *a c* ≠ *b*
using *distinct-alts* **by** *auto*

lemma *agent-in-agents* [*simp*]: *A1* ∈ *agents A2* ∈ *agents A3* ∈ *agents*
by (*simp-all add*: *agents-eq*)

lemma *alt-in-alts* [*simp*]: *a* ∈ *alts b* ∈ *alts c* ∈ *alts*
by (*simp-all add*: *alts-eq*)

lemma *Bex-alts*: (∃ *x*∈*alts*. *P x*) ⟷ *P a* ∨ *P b* ∨ *P c*
by (*simp add*: *alts-eq*)

lemma *eval-pareto-loser-aux*:
assumes *is-pref-profile R*
shows *x* ∈ *pareto-losers R* ⟷ (∃ *y*∈{*a*,*b*,*c*}. *x* ≺[*Pareto*(*R*)] *y*)
proof −
interpret *pref-profile-wf agents alts R* **by** *fact*
have ∗: *y* ∈ {*a*,*b*,*c*} **if** *x* ≺[*Pareto*(*R*)] *y* **for** *y*
using *Pareto.strict-not-outside*[*of x y*] *that* **by** (*simp add*: *alts-eq*)
show *?thesis* **by** (*auto simp*: *pareto-losers-def dest*: ∗)
qed

lemma *eval-Pareto*:
assumes *is-pref-profile R*

shows $x \prec[Pareto(R)]\ y \longleftrightarrow (\forall\, i{\in}\{A1,A2,A3\}.\ x \preceq[R\ i]\ y) \wedge (\exists\, i{\in}\{A1,A2,A3\}.\ \neg x \succeq[R\ i]\ y)$
proof −
interpret *R*: *pref-profile-wf agents alts* **by** *fact*
show *?thesis* **unfolding** *R.Pareto-strict-iff* **by** (*auto simp*: *strongly-preferred-def agents-eq*)
qed

lemmas *eval-pareto* = *eval-pareto-loser-aux eval-Pareto*
lemma *pareto-efficiency*: *is-pref-profile R* $\Longrightarrow$ *x* $\in$ *pareto-losers R* $\Longrightarrow$ *x* $\notin$ *scf R*
using *pareto-efficient*[*of R*] **by** *blast*

lemma *Ball-scf*:
assumes *is-pref-profile R*
shows $(\forall\, x{\in}scf\ R.\ P\ x) \longleftrightarrow (a \notin scf\ R \vee P\ a) \wedge (b \notin scf\ R \vee P\ b) \wedge (c \notin scf\ R \vee P\ c)$
using *scf-alts*[*OF assms*] **unfolding** *alts-eq* **by** *blast*

lemma *Ball-scf-diff*:
assumes *is-pref-profile R1 is-pref-profile R2*
shows $(\forall\, x{\in}scf\ R1 - scf\ R2.\ P\ x) \longleftrightarrow$
$(a \in scf\ R2 \vee a \notin scf\ R1 \vee P\ a) \wedge (b \in scf\ R2 \vee b \notin scf\ R1 \vee P\ b) \wedge$
$(c \in scf\ R2 \vee c \notin scf\ R1 \vee P\ c)$
using *assms*[*THEN scf-alts*] **unfolding** *alts-eq* **by** *blast*

lemma *scf-nonempty′*:
assumes *is-pref-profile R*
shows $\exists\, x{\in}alts.\ x \in scf\ R$
using *scf-nonempty*[*OF assms*] *scf-alts*[*OF assms*] **by** *blast*

2.2 Definition of Preference Profiles and Fact Gathering

We now define the 21 preference profile that will lead to the impossibility result.

preference-profile
agents: *agents*
alts: *alts*
where *R1* = *A1*: [*a*, *c*], *b* *A2*: [*a*, *c*], *b* *A3*: *b*, *c*, *a*
and *R2* = *A1*: *c*, [*a*, *b*] *A2*: *b*, *c*, *a* *A3*: *c*, *b*, *a*
and *R3* = *A1*: [*a*, *c*], *b* *A2*: *b*, *c*, *a* *A3*: *c*, *b*, *a*
and *R4* = *A1*: [*a*, *c*], *b* *A2*: *a*, *b*, *c* *A3*: *b*, *c*, *a*
and *R5* = *A1*: *c*, [*a*, *b*] *A2*: *a*, *b*, *c* *A3*: *b*, *c*, *a*
and *R6* = *A1*: *b*, [*a*, *c*] *A2*: *c*, [*a*, *b*] *A3*: *b*, *c*, *a*
and *R7* = *A1*: [*a*, *c*], *b* *A2*: *b*, [*a*, *c*] *A3*: *b*, *c*, *a*
and *R8* = *A1*: [*b*, *c*], *a* *A2*: *a*, [*b*, *c*] *A3*: *a*, *c*, *b*
and *R9* = *A1*: [*b*, *c*], *a* *A2*: *b*, [*a*, *c*] *A3*: *a*, *b*, *c*
and *R10* = *A1*: *c*, [*a*, *b*] *A2*: *a*, *b*, *c* *A3*: *c*, *b*, *a*
and *R11* = *A1*: [*a*, *c*], *b* *A2*: *a*, *b*, *c* *A3*: *c*, *b*, *a*
and *R12* = *A1*: *c*, [*a*, *b*] *A2*: *b*, *a*, *c* *A3*: *c*, *b*, *a*

and *R13* = *A1*: [*a*, *c*], *b* *A2*: *b*, *a*, *c* *A3*: *c*, *b*, *a*
and *R14* = *A1*: *a*, [*b*, *c*] *A2*: *c*, [*a*, *b*] *A3*: *a*, *c*, *b*
and *R15* = *A1*: [*b*, *c*], *a* *A2*: *a*, [*b*, *c*] *A3*: *a*, *b*, *c*
and *R16* = *A1*: [*a*, *b*], *c* *A2*: *c*, [*a*, *b*] *A3*: *a*, *b*, *c*
and *R17* = *A1*: *a*, [*b*, *c*] *A2*: *a*, *b*, *c* *A3*: *b*, *c*, *a*
and *R18* = *A1*: [*a*, *c*], *b* *A2*: *b*, [*a*, *c*] *A3*: *b*, *a*, *c*
and *R19* = *A1*: *a*, [*b*, *c*] *A2*: *c*, [*a*, *b*] *A3*: *a*, *b*, *c*
and *R20* = *A1*: *b*, [*a*, *c*] *A2*: *a*, *b*, *c* *A3*: *b*, *a*, *c*
and *R21* = *A1*: [*b*, *c*], *a* *A2*: *a*, *b*, *c* *A3*: *b*, *c*, *a*
by (*simp-all add*: *agents-eq alts-eq*)

lemmas *R-wfs* =
R1.wf R2.wf R3.wf R4.wf R5.wf R6.wf R7.wf R8.wf R9.wf R10.wf R11.wf R12.wf R13.wf R14.wf R15.wf
R16.wf R17.wf R18.wf R19.wf R20.wf R21.wf

lemmas *R-evals* =
R1.eval R2.eval R3.eval R4.eval R5.eval R6.eval R7.eval R8.eval R9.eval R10.eval R11.eval R12.eval R13.eval
R14.eval R15.eval R16.eval R17.eval R18.eval R19.eval R20.eval R21.eval

lemmas *nonemptiness* = *R-wfs* [*THEN scf-nonempty′*, *unfolded Bex-alts*]

We show the support conditions from Pareto efficiency

lemma [*simp*]: *a* ∉ *scf R1* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R1.eval*)
lemma [*simp*]: *a* ∉ *scf R2* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R2.eval*)
lemma [*simp*]: *a* ∉ *scf R3* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R3.eval*)
lemma [*simp*]: *a* ∉ *scf R6* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R6.eval*)
lemma [*simp*]: *a* ∉ *scf R7* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R7.eval*)
lemma [*simp*]: *b* ∉ *scf R8* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R8.eval*)
lemma [*simp*]: *c* ∉ *scf R9* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R9.eval*)
lemma [*simp*]: *a* ∉ *scf R12* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R12.eval*)
lemma [*simp*]: *b* ∉ *scf R14* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R14.eval*)
lemma [*simp*]: *c* ∉ *scf R15* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R15.eval*)
lemma [*simp*]: *b* ∉ *scf R16* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R16.eval*)
lemma [*simp*]: *c* ∉ *scf R17* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R17.eval*)
lemma [*simp*]: *c* ∉ *scf R18* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto*

R18.eval)
lemma [*simp*]: $b \notin$ *scf R19* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R19.eval*)
lemma [*simp*]: $c \notin$ *scf R20* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R20.eval*)
lemma [*simp*]: $c \notin$ *scf R21* **by** (*rule pareto-efficiency*) (*simp-all add*: *eval-pareto R21.eval*)

We derive the strategyproofness conditions:

lemma *s41*: $\neg$ *scf R4* $\succ$[*Fishb*(*R1 A2*)] *scf R1*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R4.eval R1.eval*)

lemma *s32*: $\neg$ *scf R3* $\succ$[*Fishb*(*R2 A1*)] *scf R2*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *R3.eval R2.eval*)

lemma *s122*: $\neg$ *scf R12* $\succ$[*Fishb*(*R2 A2*)] *scf R2*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R12.eval R2.eval*)

lemma *s133*: $\neg$ *scf R13* $\succ$[*Fishb*(*R3 A2*)] *scf R3*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R13.eval R3.eval*)

lemma *s102*: $\neg$ *scf R10* $\succ$[*Fishb*(*R2 A2*)] *scf R2*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R10.eval R2.eval*)

lemma *s13*: $\neg$ *scf R1* $\succ$[*Fishb*(*R3 A3*)] *scf R3*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R1.eval R3.eval*)

lemma *s54*: $\neg$ *scf R5* $\succ$[*Fishb*(*R4 A1*)] *scf R4*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *R5.eval R4.eval*)

lemma *s174*: $\neg$ *scf R17* $\succ$[*Fishb*(*R4 A1*)] *scf R4*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *R17.eval R4.eval*)

lemma *s74*: $\neg$ *scf R7* $\succ$[*Fishb*(*R4 A2*)] *scf R4*
by (*intro strategyproof'*[**where** $j = A2$]) (*simp-all add*: *R7.eval R4.eval*)

lemma *s114*: $\neg$ *scf R11* $\succ$[*Fishb*(*R4 A3*)] *scf R4*
by (*intro strategyproof'*[**where** $j = A3$]) (*simp-all add*: *R11.eval R4.eval*)

lemma *s45*: $\neg$ *scf R4* $\succ$[*Fishb*(*R5 A1*)] *scf R5*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *R4.eval R5.eval*)

lemma *s65*: $\neg$ *scf R6* $\succ$[*Fishb*(*R5 A2*)] *scf R5*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R5.eval R6.eval*)

lemma *s105*: $\neg$ *scf R10* $\succ$[*Fishb*(*R5 A3*)] *scf R5*
by (*intro strategyproof'*[**where** $j = A3$]) (*simp-all add*: *R10.eval R5.eval*)

lemma *s67*: ¬ *scf R6* ≻[*Fishb*(*R7 A1*)] *scf R7*
by (*intro strategyproof*′[**where** *j* = *A2*]) (*simp-all add*: *insert-commute R6.eval R7.eval*)

lemma *s187*: ¬ *scf R18* ≻[*Fishb*(*R7 A3*)] *scf R7*
by (*intro strategyproof*′[**where** *j* = *A3*]) (*simp-all add*: *insert-commute R7.eval R18.eval*)

lemma *s219*: ¬ *scf R21* ≻[*Fishb*(*R9 A2*)] *scf R9*
by (*intro strategyproof*′[**where** *j* = *A3*]) (*simp-all add*: *insert-commute R9.eval R21.eval*)

lemma *s1011*: ¬ *scf R10* ≻[*Fishb*(*R11 A1*)] *scf R11*
by (*intro strategyproof*′[**where** *j* = *A1*]) (*simp-all add*: *insert-commute R10.eval R11.eval*)

lemma *s1012*: ¬ *scf R10* ≻[*Fishb*(*R12 A2*)] *scf R12*
by (*intro strategyproof*′[**where** *j* = *A2*]) (*simp-all add*: *insert-commute R10.eval R12.eval*)

lemma *s1213*: ¬ *scf R12* ≻[*Fishb*(*R13 A1*)] *scf R13*
by (*intro strategyproof*′[**where** *j* = *A1*]) (*simp-all add*: *insert-commute R12.eval R13.eval*)

lemma *s1113*: ¬ *scf R11* ≻[*Fishb*(*R13 A2*)] *scf R13*
by (*intro strategyproof*′[**where** *j* = *A2*]) (*simp-all add*: *insert-commute R11.eval R13.eval*)

lemma *s1813*: ¬ *scf R18* ≻[*Fishb*(*R13 A3*)] *scf R13*
by (*intro strategyproof*′[**where** *j* = *A2*]) (*simp-all add*: *insert-commute R18.eval R13.eval*)

lemma *s814*: ¬ *scf R8* ≻[*Fishb*(*R14 A2*)] *scf R14*
by (*intro strategyproof*′[**where** *j* = *A1*]) (*simp-all add*: *insert-commute R8.eval R14.eval*)

lemma *s1914*: ¬ *scf R19* ≻[*Fishb*(*R14 A3*)] *scf R14*
by (*intro strategyproof*′[**where** *j* = *A3*]) (*simp-all add*: *insert-commute R19.eval R14.eval*)

lemma *s1715*: ¬ *scf R17* ≻[*Fishb*(*R15 A1*)] *scf R15*
by (*intro strategyproof*′[**where** *j* = *A3*]) (*simp-all add*: *insert-commute R17.eval R15.eval*)

lemma *s815*: ¬ *scf R8* ≻[*Fishb*(*R15 A3*)] *scf R15*
by (*intro strategyproof*′[**where** *j* = *A3*]) (*simp-all add*: *insert-commute R8.eval R15.eval*)

lemma *s516*: ¬ *scf R5* ≻[*Fishb*(*R16 A1*)] *scf R16*

by (*intro strategyproof'*[**where** $j = A3$]) (*simp-all add*: *insert-commute R5.eval R16.eval*)

lemma *s517*: ¬ *scf R5* ≻[*Fishb*(*R17 A1*)] *scf R17*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R5.eval R17.eval*)

lemma *s1619*: ¬ *scf R16* ≻[*Fishb*(*R19 A1*)] *scf R19*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R16.eval R19.eval*)

lemma *s1820*: ¬ *scf R18* ≻[*Fishb*(*R20 A2*)] *scf R20*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R18.eval R20.eval*)

lemma *s920*: ¬ *scf R9* ≻[*Fishb*(*R20 A3*)] *scf R20*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R20.eval R9.eval*)

lemma *s521*: ¬ *scf R5* ≻[*Fishb*(*R21 A1*)] *scf R21*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R21.eval R5.eval*)

lemma *s421*: ¬ *scf R4* ≻[*Fishb*(*R21 A1*)] *scf R21*
by (*intro strategyproof'*[**where** $j = A1$]) (*simp-all add*: *insert-commute R21.eval R4.eval*)

lemmas *sp* = *s41 s32 s122 s102 s133 s13 s54 s174 s54 s74 s114 s45 s65 s105 s67 s187 s219 s1011 s1012 s1213 s1113 s1813 s814 s1914 s1715 s815 s516 s517 s1619 s1820 s920 s521 s421*

We now use the simplifier to break down the strategyproofness conditions into SAT formulae. This takes a few seconds, so we use some low-level ML code to at least do the simplification in parallel.

local-setup ‹*fn lthy* =>
let
val lthy' = *lthy addsimps* @{*thms Fishb-strict-iff Ball-scf Ball-scf-diff R-evals*}
val thms = *Par-List.map* (*Simplifier.asm-full-simplify lthy'*) @{*thms sp*}
in
Local-Theory.notes [((@{*binding sp'*}, []), [(*thms*, [])])] *lthy* |> *snd*
end
›

We show that the strategyproofness conditions, the non-emptiness conditions (i. e. every SCF must return at least one winner), and the efficiency conditions are not satisfiable together, which means that the SCF whose existence we assumed simply cannot exist.

theorem *absurd*: *False*

using *sp′* **and** *nonemptiness* [*simplified*] **by** *satx*

end

2.3 Lifting to more than 3 agents and alternatives

We now employ the standard lifting argument outlined before to lift this impossibility from 3 agents and alternatives to any setting with at least 3 agents and alternatives.

locale *fb-impossibility* =
  strategyproof-anonymous-scf agents alts scf Fishb +
  pareto-efficient-scf agents alts scf
  for *agents* :: *′agent set* **and** *alts* :: *′alt set* **and** *scf* +
  assumes *card-agents-ge*: *card agents* ≥ *3*
    and *card-alts-ge*: *card alts* ≥ *3*
begin

lemma *finite-list′*:
  assumes *finite A*
  obtains *xs* **where** *A* = *set xs distinct xs length xs* = *card A*
proof −
  from *assms* **obtain** *xs* **where** *set xs* = *A* **using** *finite-list* **by** *blast*
  thus *?thesis* **using** *distinct-card*[*of remdups xs*]
    by (*intro that*[*of remdups xs*]) *simp-all*
qed

lemma *finite-list-subset*:
  assumes *finite A card A* ≥ *n*
  obtains *xs* **where** *set xs* ⊆ *A distinct xs length xs* = *n*
proof −
  obtain *xs* **where** *A* = *set xs distinct xs length xs* = *card A*
    using *finite-list′*[*OF assms*(*1*)] **by** *blast*
  with *assms* **show** *?thesis*
    by (*intro that*[*of take n xs*]) (*simp-all add*: *set-take-subset*)
qed

lemma *card-ge-3E*:
  assumes *finite A card A* ≥ *3*
  obtains *a b c* **where** *distinct* [*a*,*b*,*c*] {*a*,*b*,*c*} ⊆ *A*
proof −
  from *finite-list-subset*[*OF assms*] **guess** *xs* **.**
  moreover then obtain *a b c* **where** *xs* = [*a*, *b*, *c*]
    by (*auto simp*: *eval-nat-numeral length-Suc-conv*)
  ultimately show *?thesis* **by** (*intro that*[*of a b c*]) *simp-all*
qed

theorem *absurd*: *False*
proof −

```
  from card-ge-3E[OF finite-agents card-agents-ge] guess A1 A2 A3 .
  note agents = this
  let ?agents' = {A1, A2, A3}
  from card-ge-3E[OF finite-alts card-alts-ge] guess a b c .
  note alts = this
  let ?alts' = {a, b, c}

  interpret scf-lowering-anonymous agents alts scf ?agents' ?alts'
    by standard (use agents alts in auto)
  interpret liftable-set-extension ?alts' alts Fishb
    by (intro Fishburn-liftable alts)
  interpret scf-lowering-strategyproof agents alts scf ?agents' ?alts' Fishb ..
  interpret fb-impossibility-3-3 ?agents' ?alts' lowered A1 A2 A3 a b c
    by standard (use agents alts in simp-all)
  from absurd show False .
qed

end

end
```

References

[1] F. Brandt, C. Saile, and C. Stricker. Voting with ties: Strong impossibilities via SAT solving. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, 2018. Forthcoming.