The Incompatibility of Strategy-Proofness and Representation in Party-Approval Multi-Winner Elections

Théo Delemazure
Tom Demeulemeester
Manuel Eberl
Jonas Israel
Patrick Lederer

November 18, 2022

In party-approval multi-winner elections, the goal is to allocate the seats of a fixed-size committee to parties based on approval ballots of the voters over the parties. In particular, each voter can approve multiple parties and each party can be assigned multiple seats.

Three central requirements in this settings are:

Anonymity: The result is invariant under renaming the voters.

Representation: Every sufficiently large group of voters with similar preferences is represented by some committee members.

Strategy-proofness: No voter can benefit by misreporting her true preferences.

We show that these three basic axioms are incompatible for party-approval multi-winner voting rules, thus proving a far-reaching impossibility theorem.

The proof of this result is obtained by formulating the problem in propositional logic and then letting a SAT solver show that the formula is unsatisfiable. The DRUP proof output by the SAT solver is then converted into Lammich's GRAT format and imported into Isabelle/HOL with some custom-written ML code.

This transformation is proof-producing, so the final Isabelle/HOL theorem does not rely on any oracles or other trusted external trusted components.

Contents

1 Auxiliary Facts About Multisets

theory *PAPP-Multiset-Extras*
  imports *HOL−Library.Multiset*
begin

This section contains a number of not particularly interesting small facts about multisets.

lemma *mset-set-subset-iff*: $\textit{finite } A \Longrightarrow \textit{mset-set } A \subseteq\# \ B \longleftrightarrow A \subseteq \textit{set-mset } B$
  by (*metis finite-set-mset finite-set-mset-mset-set mset-set-set-mset-msubset*
      msubset-mset-set-iff set-mset-mono subset-mset.trans)

lemma *mset-subset-size-ge-imp-eq*:
  assumes $A \subseteq\# \ B$ $\textit{size } A \geq \textit{size } B$
  shows $A = B$
  using *assms*
proof (*induction A arbitrary*: *B*)
  case *empty*
  thus *?case* **by** *auto*
next
  case (*add x A B*)
  have [*simp*]: $x \in\# \ B$
    using *add.prems* **by** (*simp add*: *insert-subset-eq-iff*)
  define B' **where** $B' = B - \{\#x\#\}$
  have *B-eq*: $B = \textit{add-mset } x \ B'$
    using *add.prems* **unfolding** B'*-def* **by** (*auto simp*: *add-mset-remove-trivial-If*)
  have $A = B'$
    using *add.prems* **by** (*intro add.IH*) (*auto simp*: *B-eq*)
  thus *?case*
    by (*auto simp*: *B-eq*)
qed

lemma *mset-psubset-iff*:
  $X \subset\# \ Y \longleftrightarrow X \subseteq\# \ Y \wedge (\exists x.\ \textit{count } X\ x < \textit{count } Y\ x)$
  by (*meson less-le-not-le subset-mset.less-le-not-le subseteq-mset-def*)

lemma *count-le-size*: $\textit{count } A\ x \leq \textit{size } A$
  by (*induction A*) *auto*

lemma *size-filter-eq-conv-count* [*simp*]: $\textit{size } (\textit{filter-mset } (\lambda y.\ y = x)\ A) = \textit{count } A\ x$
  by (*induction A*) *auto*

lemma *multiset-filter-mono'*:
  assumes $\bigwedge x.\ x \in\# \ A \Longrightarrow P\ x \Longrightarrow Q\ x$
  shows $\textit{filter-mset } P\ A \subseteq\# \ \textit{filter-mset } Q\ A$
  using *assms* **by** (*induction A*) (*auto simp*: *subset-mset.absorb-iff1 add-mset-union*)

lemma *multiset-filter-mono''*:
  assumes $A \subseteq\# \ B$ $\bigwedge x.\ x \in\# \ A \Longrightarrow P\ x \Longrightarrow Q\ x$
  shows $\textit{filter-mset } P\ A \subseteq\# \ \textit{filter-mset } Q\ B$

using *assms multiset-filter-mono multiset-filter-mono'*
by (*metis subset-mset.order-trans*)

lemma *filter-mset-disjunction*:
assumes $\bigwedge x.\ x \in\# X \Longrightarrow P\ x \Longrightarrow Q\ x \Longrightarrow False$
shows $filter\text{-}mset\ (\lambda x.\ P\ x \vee Q\ x)\ X = filter\text{-}mset\ P\ X + filter\text{-}mset\ Q\ X$
using *assms* **by** (*induction X*) *auto*

lemma *size-mset-sum-mset*: $size\ (sum\text{-}mset\ X) = (\sum x{\in}\#X.\ size\ (x :: 'a\ multiset))$
by (*induction X*) *auto*

lemma *count-sum-mset*: $count\ (sum\text{-}mset\ X)\ x = (\sum Y{\in}\#X.\ count\ Y\ x)$
by (*induction X*) *auto*

lemma *replicate-mset-rec*: $n > 0 \Longrightarrow replicate\text{-}mset\ n\ x = add\text{-}mset\ x\ (replicate\text{-}mset\ (n - 1)\ x)$
by (*cases n*) *auto*

lemma *add-mset-neq*: $x \notin\# B \Longrightarrow add\text{-}mset\ x\ A \neq B$
by *force*

lemma *filter-replicate-mset*:
$filter\text{-}mset\ P\ (replicate\text{-}mset\ n\ x) = (if\ P\ x\ then\ replicate\text{-}mset\ n\ x\ else\ \{\#\})$
by (*induction n*) *auto*

lemma *filter-diff-mset'*: $filter\text{-}mset\ P\ (X - Y) = filter\text{-}mset\ P\ X - Y$
by (*rule multiset-eqI*) *auto*

lemma *in-diff-multiset-absorb2*: $x \notin\# B \Longrightarrow x \in\# A - B \longleftrightarrow x \in\# A$
by (*metis count-greater-zero-iff count-inI in-diff-count*)

end

2 Anonymous Party Approval Rules

theory *Anonymous-PAPP*
imports *Complex-Main Randomised-Social-Choice.Order-Predicates PAPP-Multiset-Extras*
begin

In this section we will define (anonymous) P-APP rules and some basic desirable properties of P-APP rules.

2.1 Definition of the General Setting

The following locale encapsulates an anonymous *party approval election*; that is:

- a number of voters
- a set of parties

- the size of the desired committee

The number of parties and voters is assumed to be finite and non-zero. As a modelling choice, we do not distinguish the voters at all; there is no explicit set of voters. We only care about their number.

locale *anon-papp-election* =
  fixes *n-voters* :: *nat* **and** *parties* :: *'a set* **and** *committee-size* :: *nat*
  assumes *finite-parties* [*simp*, *intro*]: *finite parties*
  assumes *n-voters-pos*: *n-voters* > *0*
  assumes *nonempty-parties* [*simp*]: *parties* ≠ {}
begin

The result of a P-APP election is a committee, i.e. a multiset of parties with the desired size.

definition *is-committee* :: *'a multiset* ⇒ *bool* **where**
  is-committee W ⟷ *set-mset W* ⊆ *parties* ∧ *size W* = *committee-size*

end

A *preference profile* for a P-APP collection consists of one approval list (i.e. a set of approved parties) for each voter. Since we are in an anonymous setting, this means that we have a *multiset* consisting of n sets of parties (where n is the number of voters).
Moreover, we make the usual assumption that the approval lists must be non-empty.

locale *anon-papp-profile* = *anon-papp-election* +
  fixes *A* :: *'a set multiset*
  assumes *A-subset*: ⋀*X*. *X* ∈# *A* ⟹ *X* ⊆ *parties*
  assumes *A-nonempty*: {} ∉# *A*
  assumes *size-A*: *size A* = *n-voters*
begin

lemma *A-nonempty'*: *A* ≠ {#}
  using *size-A n-voters-pos* **by** *auto*

end

context *anon-papp-election*
begin

abbreviation
  is-pref-profile **where** *is-pref-profile* ≡ *anon-papp-profile n-voters parties*

lemma *is-pref-profile-iff*:
  is-pref-profile A ⟷ *set-mset A* ⊆ *Pow parties* − {{}} ∧ *size A* = *n-voters*
  unfolding *anon-papp-profile-def anon-papp-profile-axioms-def*
  using *anon-papp-election-axioms* **by** *auto*

lemma *not-is-pref-profile-empty* [*simp*]: $\neg$*is-pref-profile* {#}
  using *anon-papp-profile.A-nonempty'*[*of n-voters*]
  by *auto*

The following relation is a key definition: it takes an approval list A and turns it into a preference relation on committees. A committee is to be at least as good as another if the number of approved parties in it is at least as big.

This relation is a reflexive, transitive, and total.

definition *committee-preference* :: *'a set* $\Rightarrow$ *'a multiset relation* (*Comm*) **where**
  $W1 \preceq[Comm(A)]\ W2 \longleftrightarrow size\ \{\#\ x{\in}\#W1.\ x \in A\ \#\} \le size\ \{\#\ x{\in}\#W2.\ x \in A\ \#\}$

lemma *not-strict-Comm* [*simp*]: $\neg(W1 \prec[Comm(A)]\ W2) \longleftrightarrow W1 \succeq[Comm(A)]\ W2$
  by (*auto simp*: *committee-preference-def strongly-preferred-def*)

lemma *not-weak-Comm* [*simp*]: $\neg(W1 \preceq[Comm(A)]\ W2) \longleftrightarrow W1 \succ[Comm(A)]\ W2$
  by (*auto simp*: *committee-preference-def strongly-preferred-def*)

sublocale *Comm*: *preorder Comm*(A) $\lambda x\ y.\ x \prec[Comm(A)]\ y$
  by *standard* (*auto simp*: *committee-preference-def strongly-preferred-def*)

lemma *strong-committee-preference-iff*:
  $W1 \prec[Comm(A)]\ W2 \longleftrightarrow size\ \{\#\ x{\in}\#W1.\ x \in A\ \#\} < size\ \{\#\ x{\in}\#W2.\ x \in A\ \#\}$
  by (*auto simp*: *committee-preference-def strongly-preferred-def*)

We also define the Pareto ordering on parties induced by a given preference profile: One party is at least as good (in the Pareto relation) as another if all voters agree that it is at least as good. That is, $y \succeq x$ in the Pareto ordering if all voters who approve x also approve y.

This relation is also reflexive and transitive.

definition *Pareto* :: *'a set multiset* $\Rightarrow$ *'a relation* **where**
  $x \preceq[Pareto(A)]\ y \longleftrightarrow x \in parties \wedge y \in parties \wedge (\forall X{\in}\#A.\ x \in X \longrightarrow y \in X)$

sublocale *Pareto*: *preorder-on parties Pareto A*
  by *standard* (*auto simp*: *Pareto-def*)

Pareto losers are parties that are (strictly) Pareto-dominated, i.e. there exists some other party that all voters consider to be at least as good and at least one voter considers it to be strictly better.

definition *pareto-losers* :: *'a set multiset* $\Rightarrow$ *'a set* **where**
  $pareto\text{-}losers\ A = \{x.\ \exists y.\ y \succ[Pareto(A)]\ x\}$

end

2.2 P-APP rules and Desirable Properties

The following locale describes a P-APP rule. This is simply a function that maps every preference profile to a committee of the desired size.

Note that in our setting, a P-APP rule has a fixed number of voters, a fixed set of parties, and a fixed desired committee size.

locale *anon-papp* = *anon-papp-election* +
  fixes r :: *'a set multiset* ⇒ *'a multiset*
  assumes *rule-wf*: *is-pref-profile* $A \Longrightarrow$ *is-committee* (r A)

2.3 Efficiency

Efficiency is a common notion in Social Choice Theory. The idea is that if a party is "obviously bad", then it should not be chosen. What "obviously bad" means depends on the precise notion of Efficiency that is used. We will talk about two notions: Weak Efficiency and Pareto Efficiency.

A P-APP rule is *weakly efficient* if a party that is approved by no one is never part of the output committee.

Note that approval lists must be non-empty, so there is always at least one party that is approved by at least one voter.

locale *weakly-efficient-anon-papp* = *anon-papp* +
  assumes *weakly-efficient*: *is-pref-profile* $A \Longrightarrow \forall X{\in}\#A.\ x \notin X \Longrightarrow x \notin\# \ r\ A$

A P-APP rule is *Pareto-efficient* if a Pareto-dominated party is never part of the output committee.

locale *pareto-optimal-anon-papp* = *anon-papp* +
  assumes *pareto-optimal*: *is-pref-profile* $A \Longrightarrow x \in$ *pareto-losers* $A \Longrightarrow x \notin\# \ r\ A$
begin

Pareto-efficiency implies weak efficiency:

sublocale *weakly-efficient-anon-papp*
proof
  fix A x
  assume A: *is-pref-profile* A **and** x: $\forall X{\in}\#A.\ x \notin X$
  interpret *anon-papp-profile n-voters parties committee-size A*
    by *fact*
  have $A \neq \{\#\}$
    using *A-nonempty'* **.**
  then obtain X **where** X: $X \in\# A$
    by *auto*
  with *A-nonempty* **have** $X \neq \{\}$
    by *auto*
  then obtain y **where** y: $y \in X$
    by *auto*
  show $x \notin\# \ r\ A$
  proof (*cases* $x \in$ *parties*)
    case *False*
    thus *?thesis*
      using *rule-wf*[*OF A*] **by** (*auto simp*: *is-committee-def*)
  next

case *True*
have $y \succ[Pareto(A)]\ x$
unfolding *Pareto-def* **using** *X x y True A-subset*[*of X*]
by (*auto simp*: *strongly-preferred-def*)
hence $x \in$ *pareto-losers A*
by (*auto simp*: *pareto-losers-def*)
thus *?thesis*
using *pareto-optimal*[*OF A*] **by** *auto*
qed
qed

end

2.4 Strategyproofness

Strategyproofness is another common notion in Social Choice Theory that generally encapsulates the notion that an voter should not be able to manipulate the outcome of an election in their favour by (unilaterally) submitting fake preferences; i.e. reporting one's preferences truthfully should always be the optimal choice.

A P-APP rule is called *cardinality-strategyproof* if an voter cannot obtain a better committee (i.e. one that contains strictly more of their approved parties) by submitting an approval list that is different from their real approval list.

To make the definition simpler, we first define the notion of *manipulability*: in the context of a particular P-APP rule r, a preference profile A is said to be manipulable by the voter i with the fake preference list Y if $r(A(i := Y))$ contains strictly more parties approved by i than $r(A)$.

Since we have anonymous profiles and do not talk about particular voters, we replace i with their approval list X. Since A is a multiset, the definition of manipulability becomes $r(A - \{X\} + \{Y\}) \succ_X r(A)$.

definition (**in** *anon-papp*) *card-manipulable* **where**
card-manipulable $A\ X\ Y \longleftrightarrow$
is-pref-profile $A \wedge X \in\# A \wedge Y \neq \{\} \wedge Y \subseteq$ *parties* $\wedge\ r\ (A - \{\#X\#\} + \{\#Y\#\})$ $\succ[Comm(X)]\ r\ A$

A technical (and fairly obvious) lemma: replacing an voter's approval list with a different approval list again yields a valid preference profile.

lemma (**in** *anon-papp*) *is-pref-profile-replace*:
assumes *is-pref-profile A* **and** $X \in\# A$ **and** $Y \neq \{\}$ **and** $Y \subseteq$ *parties*
shows *is-pref-profile* $(A - \{\#X\#\} + \{\#Y\#\})$
proof −
interpret *anon-papp-profile n-voters parties committee-size A*
by *fact*
show *?thesis*
using *assms A-subset A-nonempty* **unfolding** *is-pref-profile-iff*
by (*auto dest*: *in-diffD simp*: *size-Suc-Diff1*)
qed

```
locale card-stratproof-anon-papp = anon-papp +
  assumes not-manipulable: ¬card-manipulable A X Y
begin
```

The two following alternative versions of non-manipulability are somewhat nicer to use in practice.

```
lemma not-manipulable':
  assumes is-pref-profile A is-pref-profile A' A + {#Y#} = A' + {#X#}
  shows   ¬(r A' ≻[Comm(X)] r A)
proof (cases X = Y)
  case True
  thus ?thesis
    using assms by (simp add: strongly-preferred-def)
next
  case False
  interpret A: anon-papp-profile n-voters parties committee-size A
    by fact
  interpret A': anon-papp-profile n-voters parties committee-size A'
    by fact
  from assms(3) False have ∗: Y ∈# A' X ∈# A
    by (metis add-mset-add-single insert-noteq-member)+

  have ¬card-manipulable A X Y
    by (intro not-manipulable)
  hence ¬r (A − {#X#} + {#Y#}) ≻[Comm(X)] r A
    using assms ∗ A.A-subset A'.A-subset A.A-nonempty A'.A-nonempty
    by (auto simp: card-manipulable-def)
  also have A − {#X#} + {#Y#} = A'
    using assms(3) False by (metis add-eq-conv-diff add-mset-add-single)
  finally show ?thesis .
qed

lemma not-manipulable'':
  assumes is-pref-profile A is-pref-profile A' A + {#Y#} = A' + {#X#}
  shows   r A' ⪯[Comm(X)] r A
  using not-manipulable'[OF assms] by simp

end
```

2.5 Representation

Representation properties are in a sense the opposite of Efficiency properties: if a sufficiently high voters agree that certain parties are good, then these should, to some extent, be present in the result. For instance, if we have 20 voters and 5 of them approve parties A and B, then if the output committee has size 4, we would expect either A or B to be in the committee to ensure that these voters' preferences are represented fairly.

Weak representation is a particularly weak variant of this that states that if at least one

k-th of the voters (where k is the size of the output committee) approve only a single party x, then x should be in the committee at least once:

locale *weak-rep-anon-papp* =
  anon-papp n-voters parties committee-size r
  for *n-voters* **and** *parties* :: *'alt set* **and** *committee-size* :: *nat* **and** *r* +
  assumes *weak-representation*:
    is-pref-profile A $\Longrightarrow$ *committee-size* $*$ *count A* $\{x\} \geq$ *n-voters* $\Longrightarrow$ $x \in\#$ *r A*

The following alternative definition of Weak Representation is a bit closer to the definition given in the paper.

lemma *weak-rep-anon-papp-altdef*:
  weak-rep-anon-papp n-voters parties committee-size r $\longleftrightarrow$
    anon-papp n-voters parties committee-size r $\wedge$ (*committee-size* = *0* $\vee$
    ($\forall$ *A x. anon-papp-profile n-voters parties A* $\longrightarrow$
        count A $\{x\} \geq$ *n-voters* / *committee-size* $\longrightarrow$ $x \in\#$ *r A*))
  by (*cases committee-size* = *0*)
    (*auto simp*: *field-simps weak-rep-anon-papp-def*
        weak-rep-anon-papp-axioms-def
        anon-papp-def anon-papp-axioms-def anon-papp-election-def
      simp flip: *of-nat-mult*)

Justified Representation is a stronger notion which demands that if there is a subgroup of voters that comprises at least one k-th of all voters and for which the intersection of their approval lists is some nonempty set X, then at least one of the parties approved by at least one voter in that subgroup must be in the result committee.

locale *justified-rep-anon-papp* =
  anon-papp n-voters parties committee-size r
  for *n-voters* **and** *parties* :: *'alt set* **and** *committee-size* :: *nat* **and** *r* +
  assumes *justified-representation*:
    is-pref-profile A $\Longrightarrow$ $G \subseteq\#$ *A* $\Longrightarrow$ *committee-size* $*$ *size G* $\geq$ *n-voters* $\Longrightarrow$
      ($\bigcap X \in$*set-mset G. X*) $\neq \{\}$ $\Longrightarrow$ $\exists X\ x.\ X \in\#\ G \wedge x \in X \wedge x \in\#$ *r A*
begin

Any rule that satisfies Justified Representation also satisfies Weak Representation

sublocale *weak-rep-anon-papp*
proof
  fix *A x*
  assume $*$: *is-pref-profile A n-voters* $\leq$ *committee-size* $*$ *count A* $\{x\}$
  define *G* **where** *G* = *replicate-mset* (*count A* $\{x\}$) $\{x\}$
  have [*simp*]: *size G* = *count A* $\{x\}$
    by (*auto simp*: *G-def*)
  have $**$: *set-mset G* $\subseteq \{\{x\}\}$
    by (*auto simp*: *G-def*)
  have $***$: $G \subseteq\#$ *A*
    unfolding *G-def* **by** (*meson count-le-replicate-mset-subset-eq order-refl*)
  have $\exists X\ x.\ X \in\#\ G \wedge x \in X \wedge x \in\#$ *r A*
    by (*rule justified-representation*) (*use* $*$ $**$ $***$ **in** *auto*)
  thus $x \in\#$ *r A*

using $**$ **by** *auto*
qed

end

locale *card-stratproof-weak-rep-anon-papp* =
card-stratproof-anon-papp + *weak-rep-anon-papp*

2.6 Proportional Representation

The notions of Representation we have seen so far are fairly week in that they only demand that certain parties be in the committee at least once if enough voters approve them. Notions of Proportional Representation strengthen this by demanding that if a sufficiently large subgroup of voters approve some parties, then these voters must be represented in the result committe not just once, but to a degree proportional to the size of that subgroup of voters.

For Weak Representation, the proportional generalization is fairly simple: if a fraction of at least $\frac{ln}{k}$ of the voters uniquely approve a party x, then x must be in the committee at least l times.

locale *weak-prop-rep-anon-papp* =
anon-papp n-voters parties committee-size r
for *n-voters* **and** *parties* :: *'alt set* **and** *committee-size* :: *nat* **and** *r* +
assumes *weak-proportional-representation*:
$\textit{is-pref-profile } A \Longrightarrow \textit{committee-size} * \textit{count } A\ \{x\} \geq l * \textit{n-voters} \Longrightarrow \textit{count } (r\ A)\ x \geq l$
begin

sublocale *weak-rep-anon-papp*
proof
fix *A x*
assume $\textit{is-pref-profile } A\ \textit{n-voters} \leq \textit{committee-size} * \textit{count } A\ \{x\}$
thus $x \in\#\ r\ A$
using *weak-proportional-representation*[*of A 1*] **by** *auto*
qed

end

Similarly, Justified *Proportional* Representation demands that if the approval lists of a subgroup of at least $\frac{ln}{k}$ voters have a non-empty intersection, then at least l parties in the result committee are each approved by at least one of the voters in the subgroup.

locale *justified-prop-rep-anon-papp* =
anon-papp n-voters parties committee-size r
for *n-voters* **and** *parties* :: *'alt set* **and** *committee-size* :: *nat* **and** *r* +
assumes *justified-proportional-representation*:
$\textit{is-pref-profile } A \Longrightarrow G \subseteq\# A \Longrightarrow \textit{committee-size} * \textit{size } G \geq l * \textit{n-voters} \Longrightarrow$
$(\bigcap X{\in}\textit{set-mset } G.\ X) \neq \{\} \Longrightarrow \textit{size } \{\#\ x \in\#\ r\ A.\ x \in (\bigcup X{\in}\textit{set-mset } G.\ X)\ \#\} \geq l$
begin

```
sublocale justified-rep-anon-papp
proof
  fix A G
  assume is-pref-profile A G ⊆# A n-voters ≤ committee-size ∗ size G
         (⋂ X∈set-mset G. X) ≠ {}
  hence size {#x ∈# r A. ∃X∈#G. x ∈ X#} ≥ 1
    using justified-proportional-representation[of A G 1] by auto
  hence {#x ∈# r A. ∃X∈#G. x ∈ X#} ≠ {#}
    by auto
  thus ∃X x. X ∈# G ∧ x ∈ X ∧ x ∈# r A
    by fastforce
qed

sublocale weak-prop-rep-anon-papp
proof
  fix A l x
  assume ∗: is-pref-profile A l ∗ n-voters ≤ committee-size ∗ count A {x}
  define G where G = replicate-mset (count A {x}) {x}
  from ∗ have size {#x ∈# r A. x ∈ (⋃ X∈set-mset G. X)#} ≥ l
    by (intro justified-proportional-representation)
       (auto simp: G-def simp flip: count-le-replicate-mset-subset-eq)
  also have size {#x ∈# r A. x ∈ (⋃ X∈set-mset G. X)#} ≤ count (r A) x
    by (auto simp: G-def)
  finally show count (r A) x ≥ l .
qed

end


locale card-stratproof-weak-prop-rep-anon-papp =
  card-stratproof-anon-papp + weak-prop-rep-anon-papp

end
```

3 The Base Case of the Impossibility

```
theory PAPP-Impossibility-Base-Case
  imports Anonymous-PAPP SAT-Replay
begin
```

In this section, we will prove the base case of our P-APP impossibility result, namely that there exists no anonymous P-APP rule f for 6 voters, 4 parties, and committee size 3 that satisfies Weak Representation and Cardinality Strategyproofness.

The proof works by looking at some (comparatively small) set of preference profiles and the set of all 20 possible output committees. Each proposition $f(A) = C$ (where A is a profile from our set and C is one of the 20 possible output committees) is considered as a Boolean variable.

All the conditions arising on these variables based on the fact that f is a function and the additional properties (Representation, Strategyproofness) are encoded as SAT clauses. This SAT problem is then proven unsatisfiable by an external SAT solver and the resulting proof re-imported into Isabelle/HOL.

3.1 Auxiliary Material

We define the set of committees of the given size k for a given set of parties P.

definition *committees* :: *nat* ⇒ *'a set* ⇒ *'a multiset set* **where**
  committees k P = {*W*. *set-mset W* ⊆ *P* ∧ *size W* = *k*}

We now prove a recurrence for this set so that we can more easily compute the set of all possible committees:

lemma *committees-0* [*simp*]: *committees 0 P* = {{#}}
  by (*auto simp*: *committees-def*)

lemma *committees-Suc*:
  committees (*Suc n*) *P* = (⋃*x*∈*P*. ⋃ *W*∈*committees n P*. {{#*x*#} + *W*})
proof *safe*
  fix *C* **assume** *C*: *C* ∈ *committees* (*Suc n*) *P*
  hence *size C* = *Suc n*
    by (*auto simp*: *committees-def*)
  hence *C* ≠ {#}
    by *auto*
  then obtain *x* **where** *x*: *x* ∈# *C*
    by *auto*
  define *C'* **where** *C'* = *C* − {#*x*#}
  have *C* = {#*x*#} + *C' x* ∈ *P C'* ∈ *committees n P*
    using *C x* **by** (*auto simp*: *committees-def C'-def size-Diff-singleton dest*: *in-diffD*)
  thus *C* ∈ (⋃*x*∈*P*. ⋃ *W*∈*committees n P*. {{#*x*#} + *W*})
    by *blast*
qed (*auto simp*: *committees-def*)

The following function takes a list $[a_1, \ldots, a_n]$ and computes the list of all pairs of the form (a_i, a_j) with $i < j$:

fun *pairs* :: *'a list* ⇒ (*'a* × *'a*) *list* **where**
  pairs [] = []
| *pairs* (*x* # *xs*) = *map* (λ*y*. (*x*, *y*)) *xs* @ *pairs xs*

lemma *distinct-conv-pairs*: *distinct xs* ⟷ *list-all* (λ(*x*,*y*). *x* ≠ *y*) (*pairs xs*)
  by (*induction xs*) (*auto simp*: *list-all-iff*)

lemma *list-ex-unfold*: *list-ex P* (*x* # *y* # *xs*) ⟷ *P x* ∨ *list-ex P* (*y* # *xs*) *list-ex P* [*x*] ⟷ *P x*
  by *simp-all*

lemma *list-all-unfold*: *list-all P* (*x* # *y* # *xs*) ⟷ *P x* ∧ *list-all P* (*y* # *xs*) *list-all P* [*x*] ⟷ *P x*

by *simp-all*

3.2 Setup for the Base Case

We define a locale for an anonymous P-APP rule for 6 voters, 4 parties, and committee size 3 that satisfies weak representation and cardinality strategyproofness. Our goal is to prove the theorem *False* inside this locale.

locale *papp-impossibility-base-case* =
  card-stratproof-weak-rep-anon-papp 6 *parties* 3 *r*
  for *parties* :: $'a$ *set* **and** *r* +
  assumes *card-parties*: *card parties* = 4
begin

A slightly more convenient version of Weak Representation:

lemma *weak-representation'*:
  assumes *is-pref-profile A* $A' \equiv A$ $\forall z{\in}Z.$ *count A* $\{z\} \geq 2$ $\neg Z \subseteq$ *set-mset W*
  shows $r\ A' \neq W$
  using *weak-representation*[*OF assms*(1)] *assms*(2–4) **by** *auto*

The following lemma (Lemma 2 in the appendix of the paper) is a strengthening of Weak Representation and Strategyproofness in our concrete setting:

Let A be a preference profile containing approval lists X and let Z be a set of parties such that each element of Z is uniquely approved by at least two voters in A. Due to Weak Representation, at least $|X \cap Z|$ members of the committee are then approved by X.

What the lemma now says is that if there exists another voter with approval list $Y \subseteq X$ and $Y \nsubseteq Z$, then there is an additional committee member that is approved by X.

This lemma will be used both in our symmetry-breaking argument and as a means to add more clauses to the SAT instance. Since these clauses are logical consequences of Strategyproofness and Weak Representation, they are technically redundant – but their presence allows us to use consider a smaller set of profiles and still get a contradiction. Without using the lemma, we would need to feed more profiles to the SAT solver to obtain the same information.

lemma *lemma2*:
  assumes *A*: *is-pref-profile A*
  assumes $X \in\# A$ **and** $Y \in\# A - \{\#X\#\}$ **and** $Y \subseteq X$ **and** $\neg Y \subseteq Z$
  assumes *Z*: $\forall z{\in}Z.$ *count A* $\{z\} \geq 2$
  shows *size* (*filter-mset* ($\lambda x.\ x \in X$) ($r\ A$)) > *card* ($X \cap Z$)
proof (*rule ccontr*)

For the sake of contradiction, suppose the number of elements approved by X were no larger than $|X \cap Z|$.

  assume $\neg$*size* (*filter-mset* ($\lambda x.\ x \in X$) ($r\ A$)) > *card* ($X \cap Z$)
  hence *le*: *size* (*filter-mset* ($\lambda x.\ x \in X$) ($r\ A$)) $\leq$ *card* ($X \cap Z$)
    by *linarith*

interpret *anon-papp-profile 6 parties 3 A*
  by *fact*
have $Z \subseteq$ *parties*
  using *assms*(*1*,*6*) **by** (*meson is-committee-def order.trans rule-wf weak-representation'*)
have [*simp*]: *finite Z*
  by (*rule finite-subset*[*OF - finite-parties*]) *fact*

Due to Weak Representation, each member of $X \cap Z$ must be chosen at least once. But due to the above, it cannot be chosen more than once. So it has to be chosen exactly once.

have *X-approved-A-eq*: *filter-mset* ($\lambda x.\ x \in X$) (r A) = *mset-set* ($X \cap Z$)
proof −
  have *mset-set Z* ⊆# *r A*
    using *Z weak-representation*[*OF A*] **by** (*subst mset-set-subset-iff*) *auto*
  hence *size* (*filter-mset* ($\lambda x.\ x \in X$) (*mset-set Z*)) $\leq$ *size* (*filter-mset* ($\lambda x.\ x \in X$) (r A))
    by (*intro size-mset-mono multiset-filter-mono*)
  also have *filter-mset* ($\lambda x.\ x \in X$) (*mset-set Z*) = *mset-set* $\{x{\in}Z.\ x \in X\}$
    by *simp*
  also have $\{x{\in}Z.\ x \in X\} = X \cap Z$
    by *auto*
  also have *size* (*mset-set* ($X \cap Z$)) = *card* ($X \cap Z$)
    by *simp*
  finally have *size* (*filter-mset* ($\lambda x.\ x \in X$) (r A)) = *card* ($X \cap Z$)
    using *le* **by** *linarith*
  moreover have *mset-set* ($X \cap Z$) ⊆# *filter-mset* ($\lambda x.\ x \in X$) (r A)
    using *Z weak-representation*[*OF A*] **by** (*subst mset-set-subset-iff*) *auto*
  ultimately show *filter-mset* ($\lambda x.\ x \in X$) (r A) = *mset-set* ($X \cap Z$)
    by (*intro mset-subset-size-ge-imp-eq* [*symmetric*]) *auto*
qed

have *count-eq-1*: *count* (r A) x = *1* **if** $x \in X \cap Z$ **for** x
  using *that X-approved-A-eq*
  by (*metis* ‹*finite Z*› *count-filter-mset count-mset-set' diff-is-0-eq diff-zero finite-subset inf-le2 not-one-le-zero*)

Let x be some element of Y that is not in Z.

obtain x **where** x: $x \in Y - Z$
  using ‹$\neg Y \subseteq Z$› **by** *blast*
with *assms* **have** x': $x \in X - Z$
  by *auto*
have [*simp*]: $x \in$ *parties*
  using *A-subset assms*(*2*) x' **by** *blast*

Let A' be the preference profile obtained by having voter X lying and pretending she only approves x.

define A' **where** $A' = A - \{\#X\#\} + \{\#\{x\}\#\}$
have A': *is-pref-profile* A'
  using *is-pref-profile-replace*[*OF A* ‹X ∈# A›, *of* $\{x\}$] **by** (*auto simp*: A'-*def*)

We now show that even with this manipulated profile, the committee members approved by X are exactly the same as before:

have *X-approved-A′-eq*: *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A') =$ *mset-set* $(X \cap Z)$
proof −

Every element of Z must still be in the result committee due to Weak Representation.

have *mset-set* $Z \subseteq\#\ r\ A'$
proof (*subst mset-set-subset-iff*)
show $Z \subseteq$ *set-mset* $(r\ A')$
proof
fix z **assume** z: $z \in Z$
from x' z **have** [*simp*]: $x \neq z$
by *auto*
have [*simp*]: $X \neq \{z\}$
using x' **by** *auto*
show $z \in\#\ r\ A'$
using Z *weak-representation*[*OF* A', *of* z] z x x' **by** (*auto simp*: *A′-def*)
qed
qed *auto*

Thus the parties in $X \cap Z$ must be in the committee (and they are approved by X).

have *mset-set* $(X \cap Z) \subseteq\#$ *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A')$
proof −
have *filter-mset* $(\lambda x.\ x \in X)$ (*mset-set* Z) $\subseteq\#$ *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A')$
using ‹*mset-set* $Z \subseteq\#\ r\ A'$› **by** (*intro multiset-filter-mono*) *auto*
also have *filter-mset* $(\lambda x.\ x \in X)$ (*mset-set* Z) $=$ *mset-set* $(X \cap Z)$
by *auto*
finally show *mset-set* $(X \cap Z) \subseteq\#$ *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A')$ **.**
qed

Due to Strategyproofness, no additional committee members can be approved by X, so indeed only $X \cap Z$ is approved by X, and they each occur only once.

moreover have $\neg$*card-manipulable* A X $\{x\}$
using *not-manipulable* **by** *blast*
hence *size* (*mset-set* $(X \cap Z)$) $\geq$ *size* (*filter-mset* $(\lambda x.\ x \in X)$ $(r\ A')$) **using** *assms*
by (*simp add*: *card-manipulable-def A′-def strong-committee-preference-iff not-less X-approved-A-eq*)
ultimately show *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A') =$ *mset-set* $(X \cap Z)$
by (*metis mset-subset-size-ge-imp-eq*)
qed

Next, we show that the set of committee members approved by Y in the committee returned for the manipulated profile is exactly $Y \cap Z$ (and again, each party only occurs once).

have *Y-approved-A′-eq*: *filter-mset* $(\lambda x.\ x \in Y)$ $(r\ A') =$ *mset-set* $(Y \cap Z)$
proof −
have *filter-mset* $(\lambda x.\ x \in Y)$ (*filter-mset* $(\lambda x.\ x \in X)$ $(r\ A')$) $=$

filter-mset $(\lambda x.\ x \in Y)$ (*mset-set* $(X \cap Z)$)
by (*simp only*: *X-approved-A′-eq*)
also have *filter-mset* $(\lambda x.\ x \in Y)$ (*filter-mset* $(\lambda x.\ x \in X)$ $(r\ A'))$ $=$
filter-mset $(\lambda x.\ x \in Y \wedge x \in X)$ $(r\ A')$
by (*simp add*: *filter-filter-mset conj-commute*)
also have $(\lambda x.\ x \in Y \wedge x \in X) = (\lambda x.\ x \in Y)$
using *assms* **by** *auto*
also have *filter-mset* $(\lambda x.\ x \in Y)$ (*mset-set* $(X \cap Z)$) $=$ *mset-set* $(Y \cap Z)$
using *assms* **by** *auto*
finally show *?thesis* **.**
qed

Next, define the profile A'' obtained from A' by also having Y pretend to approve only x.

define A'' **where** $A'' = A' - \{\#Y\#\} + \{\#\{x\}\#\}$
have $Y \in\# A'$
using *assms* **by** (*auto simp*: *A′-def*)
hence A'': *is-pref-profile* A''
using *is-pref-profile-replace*[*OF* A', *of* Y $\{x\}$] **by** (*auto simp*: *A″-def*)

Again, the elements of Z must be chosen due to Weak Representation.

have $Z \subseteq$ *set-mset* $(r\ A'')$
proof
fix z **assume** z: $z \in Z$
from x' z **have** [*simp*]: $x \neq z$
by *auto*
have [*simp*]: $X \neq \{z\}$ $Y \neq \{z\}$
using x x' **by** *auto*
show $z \in\# r\ A''$
using Z *weak-representation*[*OF* A'', *of* z] z x x'
by (*auto simp*: *A″-def A′-def*)
qed

But now additionally, x must be chosen, since both X and Y uniquely approve it.

moreover have $x \in\# r\ A''$
using x x' ‹$Y \in\# A - \{\#X\#\}$› **by** (*intro weak-representation* A'') (*auto simp*: *A″-def A′-def*)
ultimately have *insert* x $(Y \cap Z) \subseteq$ *set-mset* $(r\ A'') \cap Y$
using x **by** *blast*

Now we have a contradiction due to Strategyproofness, since Y can force the additional member x into the committee by lying.

hence *mset-set* (*insert* x $(Y \cap Z)$) $\subseteq\#$ *filter-mset* $(\lambda w.\ w \in Y)$ $(r\ A'')$
by (*subst mset-set-subset-iff*) *auto*
hence *size* (*mset-set* (*insert* x $(Y \cap Z)$)) $\leq$ *size* (*filter-mset* $(\lambda w.\ w \in Y)$ $(r\ A''))$
by (*rule size-mset-mono*)
hence *size* (*filter-mset* $(\lambda x.\ x \in Y)$ $(r\ A''))$ $>$ *size* (*filter-mset* $(\lambda x.\ x \in Y)$ $(r\ A'))$
using x **by** (*simp add*: *Y-approved-A′-eq*)

hence *card-manipulable* A' Y $\{x\}$
using A' x ‹$Y \in\# A'$›
unfolding *card-manipulable-def strong-committee-preference-iff* A''*-def* **by** *auto*
thus *False*
using *not-manipulable* **by** *blast*
qed

The following are merely reformulation of the above lemma for technical reasons.

lemma *lemma2'*:
assumes *is-pref-profile A*
assumes $\forall z{\in}Z.\ count\ A\ \{z\} \geq 2$
assumes $X \in\# A \wedge (\exists\, Y.\ Y \in\# A - \{\#X\#\} \wedge Y \subseteq X \wedge \neg Y \subseteq Z)$
shows $\neg$*filter-mset* $(\lambda x.\ x \in X)$ $(r\ A)$ $\subseteq\#$ *mset-set* $(X \cap Z)$
proof
assume *subset*: *filter-mset* $(\lambda x.\ x \in X)$ $(r\ A)$ $\subseteq\#$ *mset-set* $(X \cap Z)$
from *assms*(*3*) **obtain** Y **where** Y: $X \in\# A$ $Y \in\# A - \{\#X\#\}$ $Y \subseteq X$ $\neg Y \subseteq Z$
by *blast*
have *card* $(X \cap Z) <$ *size* $\{\#x \in\# r\ A.\ x \in X\#\}$
by (*rule lemma2*[**where** $Y = Y$]) (*use Y assms*(*1*,*2*) **in** *auto*)
with *size-mset-mono*[*OF subset*] **show** *False*
by *simp*
qed

lemma *lemma2''*:
assumes *is-pref-profile A*
assumes $A' \equiv A$
assumes $\forall z{\in}Z.\ count\ A\ \{z\} \geq 2$
assumes $X \in\# A \wedge (\exists\, Y{\in}set\text{-}mset\ (A - \{\#X\#\}).\ Y \subseteq X \wedge \neg Y \subseteq Z)$
assumes *filter-mset* $(\lambda x.\ x \in X)$ $W \subseteq\#$ *mset-set* $(X \cap Z)$
shows $r\ A' \neq W$
using *lemma2'*[*of A Z X*] *assms* **by** *auto*

3.3 Symmetry Breaking

In the following, we formalize the symmetry-breaking argument that shows that we can reorder the four alternatives C_1 to C_4 in such a way that the preference profile

$$\{C_1\}\quad \{C_2\}\quad \{C_1, C_2\}\quad \{C_3\}\quad \{C_3\}\quad \{C_3, C_4\}$$

is mapped to one of the committees $[C_1, C_1, C_3]$ or $[C_1, C_2, C_3]$.

We start with a simple technical lemma that states that if we have a multiset A of size 3 consisting of the elements x and y and x occurs at least as often as y, then $A = [x, x, y]$.

lemma *papp-multiset-3-aux*:
assumes *size* $A = 3$ $x \in\# A$ $y \in\# A$ *set-mset* $A \subseteq \{x, y\}$ $x \neq y$ *count* $A\ x \geq$ *count* $A\ y$
shows $A = \{\#x, x, y\#\}$
proof –
have *count* $A\ x > 0$
using *assms* **by** *force*

```
  have size A = (∑z∈set-mset A. count A z)
    by (rule size-multiset-overloaded-eq)
  also have set-mset A = {x, y}
    using assms by auto
  also have (∑z∈. . . . count A z) = count A x + count A y
    using assms by auto
  finally have count A x + count A y = 3
    by (simp add: assms(1))
  moreover from assms have count A x > 0 count A y > 0
    by auto
  ultimately have ∗: count A x = 2 ∧ count A y = 1
    using ‹count A x ≥ count A y› by linarith
  show ?thesis
  proof (rule multiset-eqI)
    fix z show count A z = count {#x, x, y#} z
    proof (cases z ∈ {x, y})
      case False
      with assms have z ∉ set-mset A
        by auto
      hence count A z = 0
        by (simp add: Multiset.not-in-iff)
      thus ?thesis
        using False by auto
    qed (use ∗ in auto)
  qed
qed
```

The following is the main symmetry-breaking result. It shows that we can find parties C_1 to C_4 with the desired property.

This is a somewhat ad-hoc argument; in the appendix of the paper this is done more systematically in Lemma 3.

```
lemma symmetry-break-aux:
  obtains C1 C2 C3 C4 where
    parties = {C1, C2, C3, C4} distinct [C1, C2, C3, C4]
    r ({#{C1}, {C2}, {C1, C2}, {C3}, {C4}, {C3, C4}#}) ∈ {{#C1, C1, C3#}, {#C1, C2,
C3#}}
proof −
  note I = that
  have ∃xs. set xs = parties ∧ distinct xs
    using finite-distinct-list[of parties] by blast
  then obtain xs where xs: set xs = parties distinct xs
    by blast
  from xs have length xs = 4
    using card-parties distinct-card[of xs] by auto
  then obtain C1 C2 C3 C4 where xs-eq: xs = [C1, C2, C3, C4]
    by (auto simp: eval-nat-numeral length-Suc-conv)
  have parties-eq: parties = {C1, C2, C3, C4}
    by (subst xs(1) [symmetric], subst xs-eq) auto
  have [simp]:
```

```
    C1 ≠ C2 C1 ≠ C3 C1 ≠ C4
    C2 ≠ C1 C2 ≠ C3 C2 ≠ C4
    C3 ≠ C1 C3 ≠ C2 C3 ≠ C4
    C4 ≠ C1 C4 ≠ C2 C4 ≠ C3
  using ‹distinct xs› unfolding xs-eq by auto

define A where A = {#{C1}, {C2}, {C1, C2}, {C3}, {C4}, {C3, C4}#}
define m where m = Max (count (r A) ‘ parties)

have A: is-pref-profile A
  unfolding A-def is-pref-profile-iff by (simp add: parties-eq)
hence is-committee (r A)
  by (rule rule-wf)
hence rA: size (r A) = 3 set-mset (r A) ⊆ parties
  unfolding is-committee-def by auto
define X where X = set-mset (r A)
have X ≠ {} X ⊆ parties
  using rA by (auto simp: X-def)

have m > 0
proof −
  obtain x where x ∈ X
    using ‹X ≠ {}› by blast
  with ‹X ⊆ parties› have C1 ∈ X ∨ C2 ∈ X ∨ C3 ∈ X ∨ C4 ∈ X
    unfolding parties-eq by blast
  thus ?thesis
    unfolding m-def X-def by (subst Max-gr-iff) (auto simp: parties-eq)
qed

have m ≤ 3
proof −
  have m ≤ size (r A)
    unfolding m-def by (subst Max-le-iff) (auto simp: count-le-size)
  also have ... = 3
    by fact
  finally show ?thesis .
qed

have m ∈ (count (r A) ‘ parties)
  unfolding m-def by (intro Max-in) auto
then obtain C1' where C1': count (r A) C1' = m C1' ∈ parties
  by blast
have C1' ∈# r A
  using ‹m > 0› C1'(1) by auto

have ∃ C2'∈parties−{C1'}. {C1', C2'} ∈# A
  using C1' unfolding A-def parties-eq
  by (elim insertE; simp add: insert-Diff-if insert-commute)
then obtain C2' where C2': C2' ∈ parties − {C1'} {C1', C2'} ∈# A
```

by *blast*
have [*simp*]: *C1′* ≠ *C2′* *C2′* ≠ *C1′*
using *C2′* **by** *auto*
have *disj*: *C1′* = *C1* ∧ *C2′* = *C2* ∨ *C1′* = *C2* ∧ *C2′* = *C1* ∨ *C1′* = *C3* ∧ *C2′* = *C4* ∨ *C1′* = *C4* ∧ *C2′* = *C3*
using *C1′*(*2*) *C2′* **unfolding** *A-def parties-eq*
by (*elim insertE*; *force simp*: *insert-commute*)

obtain *C3′* **where** *C3′*: *C3′* ∈ *parties*−{*C1′*, *C2′*}
using *C1′*(*2*) *C2′* **unfolding** *parties-eq* **by** (*fastforce simp*: *insert-Diff-if*)
obtain *C4′* **where** *C4′*: *C4′* ∈ *parties*−{*C1′*, *C2′*, *C3′*}
using *C1′*(*2*) *C2′* *C3′* **unfolding** *parties-eq* **by** (*fastforce simp*: *insert-Diff-if*)
have *A-eq*: *A* = {#{*C1′*}, {*C2′*}, {*C1′*, *C2′*}, {*C3′*}, {*C4′*}, {*C3′*, *C4′*}#}
using *disj C3′ C4′*
by (*elim disjE*) (*auto simp*: *A-def parties-eq insert-commute*)
have *distinct*:
C1′ ≠ *C2′* *C1′* ≠ *C3′* *C1′* ≠ *C4′*
C2′ ≠ *C1′* *C2′* ≠ *C3′* *C2′* ≠ *C4′*
C3′ ≠ *C1′* *C3′* ≠ *C2′* *C3′* ≠ *C4′*
C4′ ≠ *C1′* *C4′* ≠ *C2′* *C4′* ≠ *C3′*
using *C1′* *C2′* *C3′* *C4′* **by** *blast+*
have *parties-eq′*: *parties* = {*C1′*, *C2′*, *C3′*, *C4′*}
using *C1′*(*2*) *C2′*(*1*) *C3′* *C4′* *distinct* **unfolding** *parties-eq* **by** (*elim insertE*) *auto*

have ¬{#*x* ∈# *r A*. *x* ∈ {*C3′*, *C4′*}#} ⊆# *mset-set* ({*C3′*, *C4′*} ∩ {})
by (*rule lemma2′*[*OF A*]) (*auto simp*: *A-eq*)
hence *C34′*: *C3′* ∈# *r A* ∨ *C4′* ∈# *r A*
by *auto*
then consider *C3′* ∈# *r A* *C4′* ∈# *r A* | *C3′* ∈# *r A* *C4′* ∉# *r A* | *C3′* ∉# *r A* *C4′* ∈# *r A*
by *blast*

thus *?thesis*
proof *cases*
assume ∗: *C3′* ∈# *r A* *C4′* ∈# *r A*
have *r A* = {#*C3′*, *C4′*, *C1′*#}
by (*rule sym*, *rule mset-subset-size-ge-imp-eq*)
(*use* ∗ ‹*C1′* ∈# *r A*› *distinct* **in**
‹*auto simp*: ‹*size* (*r A*) = *3*› *Multiset.insert-subset-eq-iff in-diff-multiset-absorb2*›)
thus *?thesis* **using** *distinct*
by (*intro that*[*of C3′ C4′ C1′ C2′*])
(*auto simp*: *parties-eq′ A-eq add-mset-commute insert-commute*)

next

assume ∗: *C3′* ∈# *r A* *C4′* ∉# *r A*
show *?thesis*
proof (*cases C2′* ∈# *r A*)
case *True*

```
    have r A = {#C1', C2', C3'#}
      by (rule sym, rule mset-subset-size-ge-imp-eq)
        (use * ‹C1' ∈# r A› distinct True in
          ‹auto simp: ‹size (r A) = 3› Multiset.insert-subset-eq-iff in-diff-multiset-absorb2›)
    thus ?thesis using distinct
      by (intro that[of C1' C2' C3' C4'])
        (auto simp: parties-eq' A-eq add-mset-commute insert-commute)
  next
    case False
    have r A = {#C1', C1', C3'#}
    proof (rule papp-multiset-3-aux)
      show set-mset (r A) ⊆ {C1', C3'}
        using ‹set-mset (r A) ⊆ -› * False unfolding parties-eq' by auto
    next
      have count (r A) C3' ≤ m
        unfolding m-def by (subst Max-ge-iff) (auto simp: parties-eq')
      also have m = count (r A) C1'
        by (simp add: C1')
      finally show count (r A) C3' ≤ count (r A) C1' .
    qed (use C1' * False ‹C1' ∈# r A› distinct in ‹auto simp: ‹size (r A) = 3››)
    thus ?thesis using distinct
      by (intro that[of C1' C2' C3' C4'])
        (auto simp: parties-eq' insert-commute add-mset-commute A-eq)
  qed

next

  assume *: C3' ∉# r A C4' ∈# r A
  show ?thesis
  proof (cases C2' ∈# r A)
    case True
    have r A = {#C1', C2', C4'#}
      by (rule sym, rule mset-subset-size-ge-imp-eq)
        (use * ‹C1' ∈# r A› distinct True in
          ‹auto simp: ‹size (r A) = 3› Multiset.insert-subset-eq-iff in-diff-multiset-absorb2›)
    thus ?thesis using distinct
      by (intro that[of C1' C2' C4' C3'])
        (auto simp: parties-eq' A-eq add-mset-commute insert-commute)
  next
    case False
    have r A = {#C1', C1', C4'#}
    proof (rule papp-multiset-3-aux)
      show set-mset (r A) ⊆ {C1', C4'}
        using ‹set-mset (r A) ⊆ -› * False unfolding parties-eq' by auto
    next
      have count (r A) C4' ≤ m
        unfolding m-def by (subst Max-ge-iff) (auto simp: parties-eq')
      also have m = count (r A) C1'
        by (simp add: C1')
```

finally show *count* (*r A*) *C4′* ≤ *count* (*r A*) *C1′* **.**
qed (*use C1′* ∗ *False* ‹*C1′* ∈# *r A*› *distinct* **in** ‹*auto simp*: ‹*size* (*r A*) = *3*››)
thus *?thesis* **using** *distinct*
by (*intro that*[*of C1′ C2′ C4′ C3′*])
(*auto simp*: *parties-eq′ insert-commute add-mset-commute A-eq*)
qed
qed
qed

We now use the choice operator to get our hands on such values C_1 to C_4.

definition *C1234* **where**
C1234 = (*SOME xs. set xs* = *parties* ∧ *distinct xs* ∧
(*case xs of* [*C1*, *C2*, *C3*, *C4*] ⇒
r ({#{*C1*}, {*C2*}, {*C1*, *C2*}, {*C3*}, {*C4*}, {*C3*, *C4*}#}) ∈ {{#*C1*, *C1*, *C3*#}, {#*C1*, *C2*, *C3*#}}))

definition *C1* **where** *C1* = *C1234* ! *0*
definition *C2* **where** *C2* = *C1234* ! *1*
definition *C3* **where** *C3* = *C1234* ! *2*
definition *C4* **where** *C4* = *C1234* ! *3*

lemma *distinct*: *distinct* [*C1*, *C2*, *C3*, *C4*]
and *parties-eq*: *parties* = {*C1*, *C2*, *C3*, *C4*}
and *symmetry-break*:
r ({#{*C1*}, {*C2*}, {*C1*, *C2*}, {*C3*}, {*C4*}, {*C3*, *C4*}#}) ∈ {{#*C1*, *C1*, *C3*#}, {#*C1*, *C2*, *C3*#}}
proof −
have *C1234*:
set C1234 = *parties* ∧ *distinct C1234* ∧
(*case C1234 of* [*C1′*, *C2′*, *C3′*, *C4′*] ⇒
r ({#{*C1′*}, {*C2′*}, {*C1′*, *C2′*}, {*C3′*}, {*C4′*}, {*C3′*, *C4′*}#}) ∈
{{#*C1′*, *C1′*, *C3′*#}, {#*C1′*, *C2′*, *C3′*#}})
unfolding *C1234-def*
proof (*rule someI-ex*)
obtain *C1′ C2′ C3′ C4′* **where** ∗:
parties = {*C1′*, *C2′*, *C3′*, *C4′*} *distinct* [*C1′*, *C2′*, *C3′*, *C4′*]
r ({#{*C1′*}, {*C2′*}, {*C1′*, *C2′*}, {*C3′*}, {*C4′*}, {*C3′*, *C4′*}#}) ∈
{{#*C1′*, *C1′*, *C3′*#}, {#*C1′*, *C2′*, *C3′*#}}
using *symmetry-break-aux* **by** *blast*
show ∃ *xs. set xs* = *parties* ∧ *distinct xs* ∧
(*case xs of* [*C1′*, *C2′*, *C3′*, *C4′*] ⇒
r ({#{*C1′*}, {*C2′*}, {*C1′*, *C2′*}, {*C3′*}, {*C4′*}, {*C3′*, *C4′*}#}) ∈
{{#*C1′*, *C1′*, *C3′*#}, {#*C1′*, *C2′*, *C3′*#}})
by (*intro exI*[*of* - [*C1′*, *C2′*, *C3′*, *C4′*]]) (*use* ∗ **in** *auto*)
qed

have *length C1234* = *4*
using *C1234 card-parties distinct-card*[*of C1234*] **by** *simp*
then obtain *C1′ C2′ C3′ C4′* **where** *C1234-eq*: *C1234* = [*C1′*, *C2′*, *C3′*, *C4′*]

by (*auto simp*: *eval-nat-numeral length-Suc-conv*)
show *distinct* [*C1*, *C2*, *C3*, *C4*] *parties* = {*C1*, *C2*, *C3*, *C4*}
r ({#{*C1*}, {*C2*}, {*C1*, *C2*}, {*C3*}, {*C4*}, {*C3*, *C4*}#}) ∈ {{#*C1*, *C1*, *C3*#}, {#*C1*, *C2*, *C3*#}}
using *C1234* **by** (*simp-all add*: *C1234-eq C1-def C2-def C3-def C4-def*)
qed

lemma *distinct′* [*simp*]:
C1 ≠ *C2* *C1* ≠ *C3* *C1* ≠ *C4* *C2* ≠ *C1* *C2* ≠ *C3* *C2* ≠ *C4*
C3 ≠ *C1* *C3* ≠ *C2* *C3* ≠ *C4* *C4* ≠ *C1* *C4* ≠ *C2* *C4* ≠ *C3*
using *distinct* **by** *auto*

lemma *in-parties* [*simp*]: *C1* ∈ *parties* *C2* ∈ *parties* *C3* ∈ *parties* *C4* ∈ *parties*
by (*subst* (*2*) *parties-eq*; *simp*; *fail*)+

3.4 The Set of Possible Committees

Next, we compute the set of the 20 possible committees.

abbreviation *COM* **where** *COM* ≡ *committees 3 parties*

definition *COM′* **where** *COM′* =
[{#*C1*, *C1*, *C1*#}, {#*C1*, *C1*, *C2*#}, {#*C1*, *C1*, *C3*#}, {#*C1*, *C1*, *C4*#},
{#*C1*, *C2*, *C2*#}, {#*C1*, *C2*, *C3*#}, {#*C1*, *C2*, *C4*#}, {#*C1*, *C3*, *C3*#},
{#*C1*, *C3*, *C4*#}, {#*C1*, *C4*, *C4*#}, {#*C2*, *C2*, *C2*#}, {#*C2*, *C2*, *C3*#},
{#*C2*, *C2*, *C4*#}, {#*C2*, *C3*, *C3*#}, {#*C2*, *C3*, *C4*#}, {#*C2*, *C4*, *C4*#},
{#*C3*, *C3*, *C3*#}, {#*C3*, *C3*, *C4*#}, {#*C3*, *C4*, *C4*#},
{#*C4*, *C4*, *C4*#}]

lemma *distinct-COM′*: *distinct COM′*
by (*simp add*: *COM′-def add-mset-neq*)

lemma *COM-eq*: *COM* = *set COM′*
by (*subst parties-eq*)
(*simp-all add*: *COM′-def numeral-3-eq-3 committees-Suc add-ac insert-commute add-mset-commute*)

lemma *r-in-COM*:
assumes *is-pref-profile A*
shows *r A* ∈ *COM*
using *rule-wf*[*OF assms*] **unfolding** *committees-def is-committee-def* **by** *auto*

lemma *r-in-COM′*:
assumes *is-pref-profile A A′* ≡ *A*
shows *list-ex* (λ*W*. *r A′* = *W*) *COM′*
using *r-in-COM*[*OF assms*(*1*)] *assms*(*2*) **by** (*auto simp*: *list-ex-iff COM-eq*)

lemma *r-right-unique*:
list-all (λ(*W1*,*W2*). *r A* ≠ *W1* ∨ *r A* ≠ *W2*) (*pairs COM′*)
proof −
have *list-all* (λ(*W1*,*W2*). *W1* ≠ *W2*) (*pairs COM′*)

using *distinct-COM′* **unfolding** *distinct-conv-pairs* **by** *blast*
thus *?thesis*
unfolding *list-all-iff* **by** *blast*
qed

end

3.5 Generating Clauses and Replaying the SAT Proof

We now employ some custom-written ML code to generate all the SAT clauses arising from the given profiles (read from an external file) as Isabelle/HOL theorems. From these, we then derive *False* by replaying an externally found SAT proof (also written from an external file).

The proof was found with the glucose SAT solver, which outputs proofs in the DRUP format (a subset of the more powerful DRAT format). We then used the *DRAT-trim* tool by Wetzler et al. [2] to make the proof smaller. This was done repeatedly until the proof size did not decrease any longer. Then, the proof was converted into the *GRAT* format introduced by Lammich [1], which is easier to check (or in our case replay) than the less explicit DRAT (or DRUP) format.

external-file *sat-data/profiles*
external-file *sat-data/papp-impossibility.grat.xz*

context *papp-impossibility-base-case*
begin

ML-file ‹*papp-impossibility.ML*›

This invocation proves a theorem called *contradiction* whose statement is *False*. Note that the DIMACS version of the SAT file that is being generated can be viewed by clicking on "See theory exports" in the messages output by the invocation below.

On a 2021 desktop PC with 12 cores, proving all the clauses takes 8.4 s (multithreaded; CPU time 55 s). Replaying the proof takes 130 s (singlethreaded).

```
local-setup ‹fn lthy =>
  let
    val thm =
      PAPP-Impossibility.derive-false lthy
        (master-dir + path ‹sat-data/profiles›)
        (master-dir + path ‹sat-data/papp-impossibility.grat.xz›)
  in
    Local-Theory.note ((binding ‹contradiction›, []), [thm]) lthy |> snd
  end
›
```

end

With this, we can now prove the impossibility result:

lemma *papp-impossibility-base-case*:
  assumes *card parties* = *4*
  shows $\neg$*card-stratproof-weak-rep-anon-papp 6 parties 3 r*
proof
  assume *card-stratproof-weak-rep-anon-papp 6 parties 3 r*
  then interpret *card-stratproof-weak-rep-anon-papp 6 parties 3 r* **.**
  interpret *papp-impossibility-base-case parties r*
    by *unfold-locales fact+*
  show *False*
    by (*rule contradiction*)
qed

end

4 Lowering P-APP Rules to Smaller Settings

theory *Anonymous-PAPP-Lowering*
  imports *Anonymous-PAPP*
begin

In this section, we prove a number of lemmas (corresponding to Lemma 1 in the paper) that allow us to take an anonymous P-APP rule with some additional properties (typically Cardinality-Strategyproofness and Weak Representation or Weak Proportional Representation) and construct from it an anonymous P-APP rule for a different setting, i.e. different number of voters, parties, and/or result committee size.

In the reverse direction, this also allows us to lift impossibility results from one setting to another.

4.1 Preliminary Lemmas

context *card-stratproof-anon-papp*
begin

The following lemma is obtained by applying Strategyproofness repeatedly. It shows that if we have l voters with identical approval lists, then this entire group of voters has no incentive to submit wrong preferences. That is, the outcome they obtain by submitting their genuine approval lists is weakly preferred by them over all outcomes obtained where these l voters submit any other preferences (and the remaining $n - l$ voters submit the same preferences as before).

This is stronger than regular Strategyproofness, where we only demand that no voter has an incentive to submit wrong preferences *unilaterally* (and everyone else keeps the same preferences). Here we know that the entire group of l voters has no incentive to submit wrong preferences in coordination with one another.

lemma *proposition2*:
  assumes *size B* = *l size A* + *l* = *n-voters*
  assumes $X \neq \{\}$ $X \subseteq$ *parties* $\{\} \notin\#$ *A*+*B* $\forall X' \in\# A+B.\ X' \subseteq$ *parties*

  shows r (*replicate-mset* l X + A) $\succeq$[*Comm*(X)] r (B + A)
  using *assms*
proof (*induction* l *arbitrary*: A B)
  case *0*
  thus *?case*
    by *simp*
next
  case (*Suc* l A B)
  from *Suc.prems* **have** *set-mset* $B \neq \{\}$
    by *auto*
  then obtain Y **where** Y: $Y \in\# B$
    by *blast*
  define B' **where** $B' = B - \{\#Y\#\}$
  define A' **where** $A' = A + \{\#Y\#\}$
  have [*simp*]: *size* $B' = l$
    using *Suc.prems* Y **by** (*simp add*: B'*-def size-Diff-singleton*)
  have [*simp*]: *size* A' = *n-voters* $- l$
    using *Suc.prems* Y **by** (*simp add*: A'*-def*)

  have r (B' + A') $\preceq$[*Comm*(X)] r (*replicate-mset* l X + A')
    by (*rule Suc.IH*) (*use Suc.prems* Y **in** ‹*auto simp*: A'*-def* B'*-def size-Diff-singleton*›)
  also have $B' + A' = B + A$
    using Y **by** (*simp add*: B'*-def* A'*-def*)
  also have r (*replicate-mset* l X + A') $\preceq$[*Comm*(X)] r (*replicate-mset* (*Suc* l) X + A)
  proof (*rule not-manipulable''*)
    show *replicate-mset* (*Suc* l) X + A + $\{\#Y\#\}$ = *replicate-mset* l X + A' + $\{\#X\#\}$
      by (*simp add*: A'*-def*)
  next
    show *is-pref-profile* (*replicate-mset* (*Suc* l) X + A)
     using *Suc.prems* **by** *unfold-locales* (*auto split*: *if-splits*)
  next
    show *is-pref-profile* (*replicate-mset* l X + A')
      using *Suc.prems* Y **by** *unfold-locales* (*auto split*: *if-splits simp*: A'*-def*)
  qed
  finally show *?case* **.**
qed

end

context *card-stratproof-weak-rep-anon-papp*
begin

In a setting with Weak Representation and Cardinality-Strategyproofness, Proposition 2 allows us to strengthen Weak Representation in the following way: Suppose we at least $l\lfloor n/k \rfloor$ voters with the same approval list X, and X consists of at least l parties. Then at least l of the members of the result committee are in X.

lemma *proposition3*:
  assumes *is-pref-profile* A $X \subseteq$ *parties card* $X \geq l$

assumes *committee-size* > 0
assumes *count A X* $\geq l * \lceil$*n-voters* / *committee-size*$\rceil$
shows *size* {# $x \in$# *r A*. $x \in X$ #} $\geq l$
using *assms*
proof (*induction l arbitrary*: *A X rule*: *less-induct*)
case (*less l A X*)
interpret *A*: *anon-papp-profile n-voters parties committee-size A*
by *fact*
consider $l = 0 \mid l = 1 \mid l > 1$
by *force*
thus *?case*
proof *cases*
assume $l = 0$
thus *?thesis* **by** *simp*
next
assume [*simp*]: $l = 1$
define *n* **where** *n* = *count A X*
with *less.prems* **have** $X \neq$ {}
by *auto*
then obtain *x* **where** *x*: $x \in X$
by *blast*
have $n \leq$ *size A*
unfolding *n-def* **by** (*rule count-le-size*)
hence $n \leq$ *n-voters*
by (*simp add*: *A.size-A*)

have *count A X* > 0
by (*rule Nat.gr0I*) (*use n-voters-pos less.prems* **in** ‹*auto simp*: *field-simps*›)
hence $X \in$# *A*
by *force*
have [*simp*]: *replicate-mset n X* ⊆# *A*
by (*simp add*: *n-def flip*: *count-le-replicate-mset-subset-eq*)

define *A''* **where** *A''* = *A* − *replicate-mset n X*
define *A'* **where** *A'* = *A''* + *replicate-mset n* {*x*}
interpret *A'*: *anon-papp-profile n-voters parties committee-size A'*
using *A.A-nonempty A.A-subset A.size-A x* ‹$X \in$# *A*›
by *unfold-locales*
(*fastforce simp*: *A'-def A''-def size-Diff-submset subset-mset.add-increasing2*
split: *if-splits dest*!: *in-diffD*)+

have $x \in$# *r A'*
proof (*rule weak-representation*)
show *is-pref-profile A'*
by (*fact A'.anon-papp-profile-axioms*)
next
have *n-voters* $\leq$ *committee-size* $* n$
using *less.prems* **by** (*simp add*: *n-def ceiling-le-iff field-simps flip*: *of-nat-mult*)
also have $n \leq$ *count A'* {*x*}

```
      by (simp add: A′-def)
    finally show n-voters ≤ committee-size ∗ count A′ {x}
      by simp
  qed
  hence 1 ≤ count (r A′) x
    by simp
  also have . . . = size {# y ∈# r A′. y = x #}
    by simp
  also have . . . ≤ size {# y ∈# r A′. y ∈ X #}
    by (intro size-mset-mono multiset-filter-mono′) (use x in auto)

  also have r A′ ⪯[Comm(X)] r A
  proof −
    have r (replicate-mset n {x} + A′′) ⪯[Comm(X)] r (replicate-mset n X + A′′)
    proof (rule proposition2)
      show {} ∉# A′′ + replicate-mset n {x}
        using A′.A-nonempty by (auto simp: A′-def)
      show ∀ X′∈#A′′ + replicate-mset n {x}. X′ ⊆ parties
        using A′.A-subset x by (auto simp: A′-def dest: in-diffD)
      show size A′′ + n = n-voters
        using ‹n ≤ n-voters› by (auto simp: A′′-def size-Diff-submset A.size-A)
    qed (use less.prems in auto)
    also have replicate-mset n X + A′′ = A
      by (simp add: A′′-def n-def flip: count-le-replicate-mset-subset-eq)
    finally show ?thesis
      by (simp add: A′-def add-ac)
  qed
  hence size {# y ∈# r A′. y ∈ X #} ≤ size {# y ∈# r A. y ∈ X #}
    by (simp add: committee-preference-def)

  finally show ?thesis
    by simp

next
  assume l: l > 1

  define n where n = count A X
  have n ≤ size A
    unfolding n-def by (rule count-le-size)
  hence n ≤ n-voters
    by (simp add: A.size-A)

  define m where m = nat (ceiling (n-voters / committee-size))
  have n-voters / committee-size ≤ m
    unfolding m-def by linarith
  hence m: n-voters ≤ committee-size ∗ m
    using ‹committee-size > 0› by (simp add: field-simps flip: of-nat-mult)
  have real n-voters / real committee-size > 0
    using n-voters-pos less.prems by auto
```

hence m': $\lceil$*real n-voters / real committee-size*$\rceil$ = *int m*
  by (*simp add*: *m-def*)
have $1 * m \leq l * m$
  using *l* **by** (*intro mult-right-mono*) *auto*
also have $l * m \leq n$
  using *less.prems* **by** (*simp add*: *m′ n-def flip*: *of-nat-mult*)
finally have $m \leq n$
  by *simp*

with *less.prems l* **have** $X \neq \{\}$
  by *auto*
then obtain *x* **where** *x*: $x \in X$
  by *blast*
have *card* $(X - \{x\}) > 0$
  using *less.prems x l* **by** *simp*
hence $X - \{x\} \neq \{\}$
  by *force*

have *count A X* > 0
  by (*rule Nat.gr0I*) (*use n-voters-pos less.prems l* **in** ‹*auto simp*: *field-simps mult-le-0-iff*›)
hence $X \in\# A$
  by *force*
have [*simp*]: *replicate-mset n X* $\subseteq\#$ *A*
  by (*simp add*: *n-def flip*: *count-le-replicate-mset-subset-eq*)

define A'' **where** $A'' = A -$ *replicate-mset n X*
define A' **where** $A' = A'' +$ *replicate-mset m* $\{x\}$ + *replicate-mset* $(n - m)$ $(X - \{x\})$
interpret A': *anon-papp-profile n-voters parties committee-size* A'
proof
  show $Y \subseteq$ *parties* **if** $Y \in\# A'$ **for** Y
    using *that A.A-subset x* ‹$X \in\# A$›
    by (*fastforce simp*: *A′-def A′′-def dest*!: *in-diffD split*: *if-splits*)
next
  show $\{\} \notin\# A'$
    using *A.A-nonempty x* ‹$X \in\# A$› ‹$X - \{x\} \neq \{\}$›
    by (*auto simp*: *A′-def A′′-def dest*!: *in-diffD split*: *if-splits*)
next
  show *size* A' = *n-voters*
    using ‹$m \leq n$›
    by (*auto simp*: *A′-def A′′-def A.size-A subset-mset.add-increasing2 size-Diff-submset*)
qed

have $x \in\#$ *r* A'
proof (*rule weak-representation*)
  show *is-pref-profile* A'
    by (*fact A′.anon-papp-profile-axioms*)
next
  have *n-voters* $\leq$ *committee-size* $*$ *m*
    by (*fact m*)

also have $m \le$ *count* A' $\{x\}$
by (*simp add*: A'*-def*)
finally show *n-voters* $\le$ *committee-size* $*$ *count* A' $\{x\}$
by *simp*
qed
hence $1 \le$ *count* $(r\ A')\ x$
by *simp*
also have $\ldots =$ *size* $\{\#\ y \in\#\ r\ A'.\ y = x\ \#\}$
by *simp*
finally have *1*: *size* $\{\#y \in\#\ r\ A'.\ y = x\#\} \ge 1$ **.**

have *2*: *size* $\{\#\ y \in\#\ r\ A'.\ y \in X - \{x\}\ \#\} \ge l - 1$
proof (*rule less.IH*)
have *int* $(l - 1) * \lceil$*real n-voters* / *real committee-size*$\rceil =$ *int* $((l - 1) * m)$
by (*auto simp add*: m' *not-less*)
also have $(l - 1) * m = l * m - m$
by (*simp add*: *algebra-simps*)
also have $l * m \le n$
using *less.prems* **by** (*simp add*: m' *n-def flip*: *of-nat-mult*)
hence $l * m - m \le n - m$
by (*meson diff-le-mono*)
also have $n - m \le$ *count* A' $(X - \{x\})$
by (*simp add*: A'*-def* A''*-def*)
finally show *int* $(l - 1) * \lceil$*real n-voters* / *real committee-size*$\rceil \le$ *int* (*count* A' $(X -$
$\{x\}))$
by *simp*
qed (*use l* A'*.anon-papp-profile-axioms x less.prems* **in** ‹*auto*›)

have $1 + (l - 1) \le$ *size* $\{\#y \in\#\ r\ A'.\ y = x\#\}$ + *size* $\{\#y \in\#\ r\ A'.\ y \in X - \{x\}\#\}$
by (*intro add-mono 1 2*)
also have $\ldots =$ *size* $(\{\#y \in\#\ r\ A'.\ y = x\#\} + \{\#y \in\#\ r\ A'.\ y \in X - \{x\}\#\})$
by *simp*
also have $\{\#y \in\#\ r\ A'.\ y = x\#\} + \{\#y \in\#\ r\ A'.\ y \in X - \{x\}\#\} =$
$\{\#y \in\#\ r\ A'.\ y = x \lor y \in X - \{x\}\#\}$
by (*rule filter-mset-disjunction* [*symmetric*]) *auto*
also have $(\lambda y.\ y = x \lor y \in X - \{x\}) = (\lambda y.\ y \in X)$
using x **by** *auto*
also have $1 + (l - 1) = l$
using l **by** *simp*

also have $r\ A' \preceq[Comm(X)]\ r\ A$
proof −
have r (*replicate-mset* m $\{x\}$ + *replicate-mset* $(n - m)$ $(X - \{x\}) + A''$) $\preceq[Comm(X)]$
r (*replicate-mset* n $X + A''$)
proof (*rule proposition2*)
show $\{\} \notin\#\ A''$ + (*replicate-mset* m $\{x\}$ + *replicate-mset* $(n - m)$ $(X - \{x\})$)
using A'*.A-nonempty* **by** (*auto simp*: A'*-def*)
show $\forall X' \in\#\ A''$ + (*replicate-mset* m $\{x\}$ + *replicate-mset* $(n - m)$ $(X - \{x\})$). $X' \subseteq$
parties

```
      using A'.A-subset x by (auto simp: A'-def dest: in-diffD)
    show size A'' + n = n-voters
      using ‹n ≤ n-voters› by (auto simp: A''-def size-Diff-submset A.size-A)
  qed (use less.prems l ‹m ≤ n› in auto)
  also have replicate-mset n X + A'' = A
    by (simp add: A''-def n-def flip: count-le-replicate-mset-subset-eq)
  finally show ?thesis
    by (simp add: A'-def add-ac)
 qed
 hence size {# y ∈# r A'. y ∈ X #} ≤ size {# y ∈# r A. y ∈ X #}
  by (simp add: committee-preference-def)

 finally show ?thesis
  by simp
 qed
qed

end
```

4.2 Dividing the number of voters

If we have a PAPP rule that satisfies weak representation and cardinality strategyproofness, for ln voters, we can turn it into one for n voters. This is done by simply cloning each voter l times.

Consequently, if we have an impossibility result for n voters, it also holds for any integer multiple of n.

```
locale divide-voters-card-stratproof-weak-rep-anon-papp =
  card-stratproof-weak-rep-anon-papp l * n-voters parties committee-size r
  for l n-voters parties committee-size r
begin

definition lift-profile :: 'a set multiset ⇒ 'a set multiset where
  lift-profile A = (∑ X∈#A. replicate-mset l X)

sublocale lowered: anon-papp-election n-voters parties
  by standard (use n-voters-pos in auto)

lemma l-pos: l > 0
  using n-voters-pos by auto

lemma empty-in-lift-profile-iff [simp]: {} ∈# lift-profile A ⟷ {} ∈# A
  using l-pos by (auto simp: lift-profile-def)

lemma set-mset-lift-profile [simp]: set-mset (lift-profile A) = set-mset A
  using l-pos by (auto simp: lift-profile-def)

lemma size-lift-profile: size (lift-profile A) = l * size A
  by (simp add: size-mset-sum-mset lift-profile-def image-mset.compositionality o-def)
```

```
lemma count-lift-profile [simp]: count (lift-profile A) x = l * count A x
  unfolding lift-profile-def by (induction A) auto

lemma is-pref-profile-lift-profile [intro]:
  assumes lowered.is-pref-profile A
  shows   is-pref-profile (lift-profile A)
proof -
  interpret anon-papp-profile n-voters parties committee-size A
    by fact
  show ?thesis
    using A-nonempty A-subset size-A
    by unfold-locales
       (auto simp: lift-profile-def size-mset-sum-mset image-mset.compositionality o-def)
qed

sublocale lowered: anon-papp n-voters parties committee-size r ∘ lift-profile
proof
  fix A assume lowered.is-pref-profile A
  hence is-pref-profile (lift-profile A)
    by blast
  hence is-committee (r (lift-profile A))
    using rule-wf by blast
  thus lowered.is-committee ((r ∘ lift-profile) A)
    by simp
qed

sublocale lowered: weak-rep-anon-papp n-voters parties committee-size r ∘ lift-profile
proof
  fix A x
  assume A: lowered.is-pref-profile A and x: n-voters ≤ committee-size * count A {x}
  from A have A': is-pref-profile (lift-profile A)
    by blast
  from x have l * n-voters ≤ l * (committee-size * count A {x})
    by (rule mult-left-mono) auto
  also have ... = committee-size * count (lift-profile A) {x}
    by simp
  finally have x ∈# r (lift-profile A)
    by (intro weak-representation A')
  thus x ∈# (r ∘ lift-profile) A
    by simp
qed

sublocale lowered: card-stratproof-anon-papp n-voters parties committee-size r ∘ lift-profile
proof
  fix A X Y
  show ¬lowered.card-manipulable A X Y
    unfolding lowered.card-manipulable-def
  proof (rule notI, elim conjE)
```

```
    assume A: lowered.is-pref-profile A and XY: X ∈# A  Y ≠ {}  Y ⊆ parties
    assume ∗: (r ∘ lift-profile) A ≺[lowered.committee-preference X]
              (r ∘ lift-profile) (A − {#X#} + {#Y#})
    interpret anon-papp-profile n-voters parties committee-size A
      by fact
    have X: X ≠ {} X ⊆ parties
      using XY A-nonempty A-subset by auto

    define A′ where A′ = A − {#X#}
    have A′: A = A′ + {#X#}
      using XY by (simp add: A′-def)

    have r (lift-profile A) ≺[committee-preference X]
         r (lift-profile (A − {#X#} + {#Y#}))
      using ∗ by simp
    also have r (lift-profile (A − {#X#} + {#Y#})) ⪯[committee-preference X]
              r (lift-profile A − {#X#} + {#Y#})
    proof −
      have r (replicate-mset (l − 1) Y + (lift-profile A′ + {#Y#})) ⪯[committee-preference X]
           r (replicate-mset (l − 1) X + (lift-profile A′ + {#Y#}))
      proof (rule proposition2)
        show size (lift-profile A′ + {#Y#}) + (l − 1) = l ∗ n-voters
          using XY l-pos n-voters-pos
          by (simp add: A′-def size-lift-profile size-Diff-singleton
                        algebra-simps Suc-diff-le size-A)
      next
        show {} ∉# lift-profile A′ + {#Y#} + replicate-mset (l − 1) Y
          using XY A-nonempty by (auto simp: A′-def dest: in-diffD)
      next
        show ∀ X′∈#lift-profile A′ + {#Y#} + replicate-mset (l − 1) Y. X′ ⊆ parties
          using XY A-subset by (auto simp: A′-def dest: in-diffD)
      qed (use X in auto)
      thus ?thesis
        by (simp add: A′ replicate-mset-rec l-pos lift-profile-def)
    qed
    finally have card-manipulable (lift-profile A) X Y
      unfolding card-manipulable-def using XY A by auto
    with not-manipulable show False
      by blast
  qed
qed

sublocale lowered: card-stratproof-weak-rep-anon-papp n-voters parties committee-size r ∘ lift-profile
  ..

end


locale divide-voters-card-stratproof-weak-prop-rep-anon-papp =
```

card-stratproof-weak-prop-rep-anon-papp $l * $ *n-voters parties committee-size r*
for *l n-voters parties committee-size r*
begin

sublocale *divide-voters-card-stratproof-weak-rep-anon-papp* **..**

sublocale *lowered*: *card-stratproof-weak-prop-rep-anon-papp*
n-voters parties committee-size $r \circ$ *lift-profile*
proof
fix A x l'
assume A: *lowered.is-pref-profile* A **and** x: $l' *$ *n-voters* $\leq$ *committee-size* $*$ *count* A $\{x\}$
from A **have** A': *is-pref-profile* (*lift-profile* A)
by *blast*
from x **have** $l * (l' *$ *n-voters*$) \leq l * ($*committee-size* $*$ *count* A $\{x\})$
by (*rule mult-left-mono*) *auto*
also have $\ldots =$ *committee-size* $*$ *count* (*lift-profile* A) $\{x\}$
by *simp*
also have $l * (l' *$ *n-voters*$) = l' * (l *$ *n-voters*$)$
by (*simp add*: *algebra-simps*)
finally have *count* (*r* (*lift-profile* A)) $x \geq l'$
by (*intro weak-proportional-representation* A')
thus *count* ((*r* $\circ$ *lift-profile*) A) $x \geq l'$
by *simp*
qed

end

4.3 Decreasing the number of parties

If we have a PAPP rule that satisfies weak representation and cardinality strategyproofness, for m parties, we can turn it into one for $m - 1$ parties. This is done by simply duplicating one particular party (say x) in the preference profile, i.e. whenever x is part of an approval list, we add a clone of x (say y) as well. Should y then end up in the committee, we simply replace it with x.

Consequently, if we have an impossibility result for k parties, it also holds for $\geq m$ parties.

locale *remove-alt-card-stratproof-weak-rep-anon-papp* =
card-stratproof-weak-rep-anon-papp n-voters parties committee-size r
for *n-voters* **and** *parties* :: *'a set* **and** *committee-size r* +
fixes x y :: *'a*
assumes *xy*: $x \in$ *parties* $y \in$ *parties* $x \neq y$
begin

definition *lift-applist* :: *'a set* $\Rightarrow$ *'a set* **where**
lift-applist X = (*if* $x \in X$ *then insert* y X *else* X)

definition *lift-profile* :: *'a set multiset* $\Rightarrow$ *'a set multiset* **where**
lift-profile A = *image-mset lift-applist* A

definition *lower-result* **where** *lower-result* C = *image-mset* (λz. *if* $z = y$ *then* x *else* z) C

definition *lowered* **where** *lowered* = *lower-result* $\circ$ r $\circ$ *lift-profile*

lemma *lift-profile-empty* [*simp*]: *lift-profile* $\{\#\} = \{\#\}$
by (*simp add*: *lift-profile-def*)

lemma *lift-profile-add-mset* [*simp*]:
lift-profile (*add-mset* X A) = *add-mset* (*lift-applist* X) (*lift-profile* A)
by (*simp add*: *lift-profile-def*)

lemma *empty-in-lift-profile-iff* [*simp*]: $\{\} \in\#$ *lift-profile* $A \longleftrightarrow \{\} \in\# A$
by (*auto simp*: *lift-applist-def lift-profile-def*)

lemma *size-lift-profile* [*simp*]: *size* (*lift-profile* A) = *size* A
by (*simp add*: *lift-profile-def*)

lemma *lift-applist-eq-self-iff* [*simp*]: *lift-applist* $X = X \longleftrightarrow x \notin X \vee y \in X$
by (*auto simp*: *lift-applist-def*)

lemma *lift-applist-eq-self-iff'* [*simp*]: *lift-applist* $(X - \{y\}) = X \longleftrightarrow (x \in X \longleftrightarrow y \in X)$
by (*cases* $y \in X$) (*auto simp*: *lift-applist-def xy*)

lemma *in-lift-applist-iff*: $z \in$ *lift-applist* $X \longleftrightarrow z \in X \vee (z = y \wedge x \in X)$
by (*auto simp*: *lift-applist-def*)

lemma *count-lift-profile*:
assumes $\forall\, Y \in\# A.\ y \notin Y$
shows *count* (*lift-profile* A) X = (*if* $x \in X \longleftrightarrow y \in X$ *then* *count* A $(X - \{y\})$ *else* 0)
using *assms xy* **by** (*induction* A) (*auto simp*: *lift-applist-def*)

lemma *y-notin-lower-result* [*simp*]: $y \notin\#$ *lower-result* C
using *xy* **by** (*auto simp*: *lower-result-def*)

lemma *lower-result-subset*: *set-mset* (*lower-result* C) $\subseteq$ *insert* x (*set-mset* $C - \{y\}$)
using *xy* **by** (*auto simp*: *lower-result-def*)

lemma *lower-result-subset'*: *set-mset* $C \subseteq$ *parties* $\Longrightarrow$ *set-mset* (*lower-result* C) $\subseteq$ *parties*
using *xy* **by** (*auto simp*: *lower-result-def*)

lemma *size-lower-result* [*simp*]: *size* (*lower-result* C) = *size* C
by (*simp add*: *lower-result-def*)

lemma *count-lower-result*:
count (*lower-result* C) z =
(*if* $z = y$ *then* 0
else if $z = x$ *then* *count* C x + *count* C y

```
    else count C z)
  using xy by (induction C) (auto simp: lower-result-def)

lemma in-lower-result-iff:
  z ∈# lower-result C ⟷ z ≠ y ∧ (z ∈# C ∨ (z = x ∧ y ∈# C))
  unfolding lower-result-def using xy by (induction C) auto


sublocale lowered: anon-papp-election n-voters parties − {y}
  by standard (use n-voters-pos xy in auto)

lemma is-pref-profile-lift-profile [intro]:
  assumes lowered.is-pref-profile A
  shows   is-pref-profile (lift-profile A)
proof −
  interpret anon-papp-profile n-voters parties − {y} committee-size A
    by fact
  show ?thesis
    using A-nonempty A-subset size-A
    by unfold-locales
       (auto simp: lift-profile-def lift-applist-def xy
                size-mset-sum-mset image-mset.compositionality o-def)
qed

sublocale lowered: anon-papp n-voters parties − {y} committee-size lowered
proof
  fix A assume lowered.is-pref-profile A
  hence is-pref-profile (lift-profile A)
    by blast
  hence is-committee (r (lift-profile A))
    using rule-wf by blast
  thus lowered.is-committee (lowered A)
    unfolding lowered.is-committee-def is-committee-def lowered-def
    using lower-result-subset'[of r (lift-profile A)] by auto
qed

sublocale lowered: weak-rep-anon-papp n-voters parties − {y} committee-size lowered
proof
  fix A z
  assume A: lowered.is-pref-profile A and z: n-voters ≤ committee-size ∗ count A {z}
  interpret A: anon-papp-profile n-voters parties − {y} committee-size A
    by fact
  have committee-size > 0
    using z n-voters-pos by (intro Nat.gr0I) auto

  from A have A': is-pref-profile (lift-profile A)
    by blast
  have count A {z} > 0
    using z n-voters-pos by (intro Nat.gr0I) auto
```

```
  hence {z} ∈# A
    by simp
  hence z′: z ∈ parties − {y}
    using A.A-subset z by auto

  define C where C = r (lift-profile A)

  show z ∈# lowered A
  proof (cases z = x)
    case False
    have n-voters ≤ committee-size ∗ count A {z}
      by fact
    also have count A {z} ≤ count (lift-profile A) {z}
      using A.A-subset z′ False by (subst count-lift-profile) auto
    hence committee-size ∗ count A {z} ≤ committee-size ∗ count (lift-profile A) {z}
      by (intro mult-left-mono) auto
    finally have z ∈# r (lift-profile A)
      by (intro weak-representation A′)
    thus z ∈# lowered A
      using False z′ by (simp add: lowered-def in-lower-result-iff)
  next
    case [simp]: True
    have 1 ≤ size {#z ∈# C. z ∈ {x, y}#}
      unfolding C-def
    proof (rule proposition3)
      have [simp]: {x, y} − {y} = {x}
        using xy by auto
      hence n-voters ≤ committee-size ∗ count (lift-profile A) {x, y}
        using xy A.A-subset z by (subst count-lift-profile) auto
      thus int 1 ∗ ⌈real n-voters / real committee-size⌉ ≤ int (count (lift-profile A) {x, y})
        using ‹committee-size > 0›
        by (auto simp: ceiling-le-iff field-simps simp flip: of-nat-mult)
    qed (use A′ xy ‹committee-size > 0› in auto)
    also have . . . = count C x + count C y
      using xy by (induction C) auto
    also have . . . = count (lowered A) x
      using xy by (simp add: lowered-def count-lower-result C-def)
    finally show z ∈# lowered A
      by simp
  qed
qed

lemma filter-lower-result-eq:
  y ∉ X =⇒ {#x ∈# lower-result C. x ∈ X#} = lower-result {#x ∈# C. x ∈ lift-applist X#}
  by (induction C) (auto simp: lower-result-def lift-applist-def)

sublocale lowered: card-stratproof-anon-papp n-voters parties − {y} committee-size lowered
proof
  fix A X Y
```

```
  show ¬lowered.card-manipulable A X Y
    unfolding lowered.card-manipulable-def
  proof (rule notI, elim conjE)
    assume A: lowered.is-pref-profile A and XY: X ∈# A  Y ≠ {}  Y ⊆ parties − {y}
    assume ∗: lowered A ≺[lowered.committee-preference X] lowered (A − {#X#} + {#Y#})
    interpret anon-papp-profile n-voters parties − {y} committee-size A
      by fact
    have X: X ≠ {} X ⊆ parties − {y}
      using XY A-nonempty A-subset by auto
    define A′ where A′ = A − {#X#}
    have A′: A = A′ + {#X#}
      using XY by (simp add: A′-def)

    from ∗ have size {#x ∈# lower-result (r (lift-profile A′ + {#lift-applist X#})). x ∈ X#}
<
                 size {#x ∈# lower-result (r (lift-profile A′ + {#lift-applist Y#})). x ∈ X#}
      by (simp add: lowered-def A′ lowered.strong-committee-preference-iff)
    also have {#x ∈# lower-result (r (lift-profile A′ + {#lift-applist X#})). x ∈ X#} =
               lower-result {#x ∈# r (lift-profile A′ + {#lift-applist X#}). x ∈ lift-applist X#}
      using X by (subst filter-lower-result-eq) auto
    also have {#x ∈# lower-result (r (lift-profile A′ + {#lift-applist Y#})). x ∈ X#} =
               lower-result {#x ∈# r (lift-profile A′ + {#lift-applist Y#}). x ∈ lift-applist X#}
      using X by (subst filter-lower-result-eq) auto
    finally have size {#x ∈# r (lift-profile A′ + {#lift-applist X#}). x ∈ lift-applist X#} <
                  size {#x ∈# r (lift-profile A′ + {#lift-applist Y#}). x ∈ lift-applist X#}
      by simp
    hence r (lift-profile A′ + {#lift-applist X#}) ≺[committee-preference (lift-applist X)]
          r (lift-profile A′ + {#lift-applist Y#})
      by (simp add: strong-committee-preference-iff)
   moreover have ¬r (lift-profile A′ + {#lift-applist X#}) ≺[committee-preference (lift-applist
X)]
                  r (lift-profile A′ + {#lift-applist Y#})
    proof (rule not-manipulable′ [where Y = lift-applist Y])
      have is-pref-profile (lift-profile A)
        using A by blast
      thus is-pref-profile (lift-profile A′ + {#lift-applist X#})
        using A by (simp add: A′)
    next
      have is-pref-profile (lift-profile (A − {#X#} + {#Y#}))
        using A  XY lowered.is-pref-profile-replace by blast
      thus is-pref-profile (lift-profile A′ + {#lift-applist Y#})
        by (simp add: A′)
    qed auto
    ultimately show False
      by contradiction
  qed
qed

sublocale lowered: card-stratproof-weak-rep-anon-papp n-voters parties − {y} committee-size
```

```
lowered
  ..

end
```

The following lemma is now simply an iterated application of the above. This allows us to restrict a P-APP rule to any non-empty subset of parties.

```
lemma card-stratproof-weak-rep-anon-papp-restrict-parties:
  assumes card-stratproof-weak-rep-anon-papp n parties k r parties′ ⊆ parties parties′ ≠ {}
  shows   ∃ r. card-stratproof-weak-rep-anon-papp n parties′ k r
proof −
  have finite parties
  proof −
    interpret card-stratproof-weak-rep-anon-papp n parties k r
      by fact
    show ?thesis
      by (rule finite-parties)
  qed
  thus ?thesis
    using assms
  proof (induction parties arbitrary: r rule: finite-psubset-induct)
    case (psubset parties r)
    show ?thesis
    proof (cases parties = parties′)
      case True
      thus ?thesis
        using psubset.prems by auto
    next
      case False
      obtain x where x: x ∈ parties′
        using psubset.prems by blast
      from False and psubset.prems have parties − parties′ ≠ {}
        by auto
      then obtain y where y: y ∈ parties − parties′
        by blast

      interpret card-stratproof-weak-rep-anon-papp n parties k r
        by fact
      interpret remove-alt-card-stratproof-weak-rep-anon-papp n parties k r x y
        by standard (use x y psubset.prems in auto)
      show ?thesis
      proof (rule psubset.IH)
        show parties − {y} ⊂ parties and parties′ ⊆ parties − {y} parties′ ≠ {}
          using x y psubset.prems by auto
      next
        show card-stratproof-weak-rep-anon-papp n (parties − {y}) k lowered
          using lowered.card-stratproof-weak-rep-anon-papp-axioms .
      qed
    qed
```

qed
qed

4.4 Decreasing the committee size

If we have a PAPP rule that satisfies weak representation and cardinality strategyproofness, for $l(k+1)$ voters, $m+1$ parties, and a committee size of $k+1$, we can turn it into one for lk voters, m parties, and a committee size of k.

This is done by again cloning a party x into a new party y and additionally adding l new voters whose preferences are $\{x, y\}$. We again replace any y occuring in the output committee with x. Weak representation then ensures that x occurs in the output at least once, and we then simply remove one x from the committee to obtain an output committee of size $k - 1$.

Consequently, if we have an impossibility result for a committee size of m, we can extend it to a larger committee size, but at the cost of introducing a new party and new voters, and with a restriction on the number of voters.

locale *decrease-committee-card-stratproof-weak-rep-anon-papp* =
card-stratproof-weak-rep-anon-papp $l * (k + 1)$ *insert y parties* $k + 1$ *r*
for *l k y* **and** *parties* :: *'a set* **and** *r* +
fixes *x* :: *'a*
assumes *xy*: $x \in$ *parties* $y \notin$ *parties*
assumes *k*: $k > 0$
begin

definition *lift-applist* :: *'a set* ⇒ *'a set* **where**
lift-applist X = (*if* $x \in X$ *then insert y X else X*)

definition *lift-profile* :: *'a set multiset* ⇒ *'a set multiset* **where**
lift-profile A = *image-mset lift-applist A* + *replicate-mset l* $\{x, y\}$

definition *lower-result*
where *lower-result C* = *image-mset* ($\lambda z.$ *if* $z = y$ *then x else z*) $C - \{\#x\#\}$

definition *lowered* **where** *lowered* = *lower-result* ∘ *r* ∘ *lift-profile*

lemma *l*: $l > 0$
using *n-voters-pos* **by** *auto*

lemma *x-neq-y* [*simp*]: $x \neq y$ $y \neq x$
using *xy* **by** *auto*

lemma *lift-profile-empty* [*simp*]: *lift-profile* $\{\#\}$ = *replicate-mset l* $\{x, y\}$
by (*simp add*: *lift-profile-def*)

lemma *lift-profile-add-mset* [*simp*]:
lift-profile (*add-mset X A*) = *add-mset* (*lift-applist X*) (*lift-profile A*)

by (*simp add*: *lift-profile-def*)

lemma *empty-in-lift-profile-iff* [*simp*]: $\{\} \in\# \textit{lift-profile } A \longleftrightarrow \{\} \in\# A$
by (*auto simp*: *lift-applist-def lift-profile-def*)

lemma *size-lift-profile* [*simp*]: $\textit{size } (\textit{lift-profile } A) = \textit{size } A + l$
by (*simp add*: *lift-profile-def*)

lemma *lift-applist-eq-self-iff* [*simp*]: $\textit{lift-applist } X = X \longleftrightarrow x \notin X \lor y \in X$
by (*auto simp*: *lift-applist-def*)

lemma *lift-applist-eq-self-iff'* [*simp*]: $\textit{lift-applist } (X - \{y\}) = X \longleftrightarrow (x \in X \longleftrightarrow y \in X)$
by (*cases* $y \in X$) (*auto simp*: *lift-applist-def xy*)

lemma *in-lift-applist-iff*: $z \in \textit{lift-applist } X \longleftrightarrow z \in X \lor (z = y \land x \in X)$
by (*auto simp*: *lift-applist-def*)

lemma *count-lift-profile*:
assumes $\forall Y{\in}\#A.\ y \notin Y$
shows $\textit{count } (\textit{lift-profile } A)\ X =$
$(\textit{if } x \in X \longleftrightarrow y \in X \textit{ then count } A\ (X - \{y\}) \textit{ else } 0) +$
$(\textit{if } X = \{x, y\} \textit{ then } l \textit{ else } 0)$
using *assms xy* **by** (*induction A*) (*auto simp*: *lift-applist-def*)

lemma *y-notin-lower-result* [*simp*]: $y \notin\# \textit{lower-result } C$
using *xy* **by** (*auto simp*: *lower-result-def dest*: *in-diffD*)

lemma *lower-result-subset*: $\textit{set-mset } (\textit{lower-result } C) \subseteq \textit{insert } x\ (\textit{set-mset } C - \{y\})$
using *xy* **by** (*auto simp*: *lower-result-def dest*: *in-diffD*)

lemma *lower-result-subset'*: $\textit{set-mset } C \subseteq \textit{insert } y \textit{ parties} \Longrightarrow \textit{set-mset } (\textit{lower-result } C) \subseteq \textit{parties}$
by (*rule order.trans*[*OF lower-result-subset*]) (*use xy* **in** *auto*)

lemma *size-lower-result* [*simp*]:
assumes $x \in\# C \lor y \in\# C$
shows $\textit{size } (\textit{lower-result } C) = \textit{size } C - 1$
using *assms* **by** (*auto simp*: *lower-result-def size-Diff-singleton*)

lemma *size-lower-result'*: $\textit{size } (\textit{lower-result } C) = \textit{size } C - (\textit{if } x \in\# C \lor y \in\# C \textit{ then } 1 \textit{ else } 0)$
proof −
define *f* **where** $f = (\lambda C.\ \textit{image-mset } (\lambda z.\ \textit{if } z = y \textit{ then } x \textit{ else } z)\ C)$
have $\textit{size } (\textit{lower-result } C) = \textit{size } (f\ C - \{\#x\#\})$
by (*simp add*: *lower-result-def f-def*)
also have $\ldots = \textit{size } (f\ C) - (\textit{if } x \in\# f\ C \textit{ then } 1 \textit{ else } 0)$
by (*simp add*: *diff-single-trivial size-Diff-singleton*)

finally show *?thesis*
by (*auto simp*: *f-def*)
qed

lemma *count-lower-result*:
$count\ (lower\text{-}result\ C)\ z =$
$(if\ z = y\ then\ 0$
$else\ if\ z = x\ then\ count\ C\ x + count\ C\ y - 1$
$else\ count\ C\ z)$ (**is** - = *?rhs*)
proof −
define f **where** $f = (\lambda C.\ image\text{-}mset\ (\lambda z.\ if\ z = y\ then\ x\ else\ z)\ C)$
have $count\ (lower\text{-}result\ C)\ z = count\ (f\ C - \{\#x\#\})\ z$
by (*simp add*: *lower-result-def f-def*)
also have $\ldots = count\ (f\ C)\ z - (if\ z = x\ then\ 1\ else\ 0)$
by *auto*
also have $count\ (f\ C)\ z = (if\ z = y\ then\ 0\ else\ if\ z = x\ then\ count\ C\ x + count\ C\ y\ else\ count\ C\ z)$
using *xy* **by** (*induction C*) (*auto simp*: *f-def*)
also have $\ldots - (if\ z = x\ then\ 1\ else\ 0) = ?rhs$
by *auto*
finally show *?thesis* **.**
qed

lemma *in-lower-resultD*:
$z \in\#\ lower\text{-}result\ C \Longrightarrow z = x \lor z \in\#\ C$
using *xy* **by** (*auto simp*: *lower-result-def dest*!: *in-diffD*)

lemma *in-lower-result-iff*:
$z \in\#\ lower\text{-}result\ C \longleftrightarrow z \neq y \land (if\ z = x\ then\ count\ C\ x + count\ C\ y > 1\ else\ z \in\#\ C)$
(**is** - = *?rhs*)
proof −
have $z \in\#\ lower\text{-}result\ C \longleftrightarrow count\ (lower\text{-}result\ C)\ z > 0$
by *auto*
also have $\ldots \longleftrightarrow ?rhs$
by (*subst count-lower-result*) *auto*
finally show *?thesis* **.**
qed

lemma *filter-lower-result-eq*:
assumes $y \notin X$
shows $\{\#z \in\#\ lower\text{-}result\ C.\ z \in X\#\} = lower\text{-}result\ \{\#z \in\#\ C.\ z \in lift\text{-}applist\ X\#\}$
proof −
define f **where** $f = (\lambda C.\ \{\#if\ z = y\ then\ x\ else\ z.\ z \in\#\ C\#\})$
have $lower\text{-}result\ \{\#z \in\#\ C.\ z \in lift\text{-}applist\ X\#\} = f\ \{\#z \in\#\ C.\ z \in lift\text{-}applist\ X\#\} - \{\#x\#\}$
by (*simp add*: *f-def lower-result-def*)
also have $f\ \{\#z \in\#\ C.\ z \in lift\text{-}applist\ X\#\} = \{\#z \in\#\ f\ C.\ z \in X\#\}$
using *assms* **by** (*induction C*) (*auto simp*: *f-def lift-applist-def*)
also have $\ldots - \{\#x\#\} = \{\#z \in\#\ f\ C - \{\#x\#\}.\ z \in X\#\}$

```
    by (subst filter-diff-mset′) auto
  also have f C − {#x#} = lower-result C
    by (simp add: f-def lower-result-def)
  finally show ?thesis ..
qed


sublocale lowered: anon-papp-election l ∗ k parties k
  by standard (use n-voters-pos xy finite-parties k in auto)

lemma is-pref-profile-lift-profile [intro]:
  assumes lowered.is-pref-profile A
  shows   is-pref-profile (lift-profile A)
proof −
  interpret A: anon-papp-profile l ∗ k parties k A
    by fact
  show ?thesis
    using A.A-nonempty A.A-subset A.size-A l
    by unfold-locales
      (auto simp: lift-profile-def lift-applist-def xy
                size-mset-sum-mset image-mset.compositionality o-def)
qed

lemma is-committee-lower-result:
  assumes is-committee C x ∈# C ∨ y ∈# C
  shows   lowered.is-committee (lower-result C)
  using assms unfolding is-committee-def lowered.is-committee-def
  using lower-result-subset′[of C] by auto

lemma x-or-y-in-r-lift-profile:
  assumes lowered.is-pref-profile A
  shows   x ∈# r (lift-profile A) ∨ y ∈# r (lift-profile A)
proof −
  interpret A: anon-papp-profile l ∗ k parties k A
    by fact
  have size {#z ∈# r (lift-profile A). z ∈ {x, y}#} ≥ 1
  proof (rule proposition3)
    have real (l ∗ (k + 1)) / real (k + 1) = real l
      by (simp add: field-simps)
    also have int 1 ∗ ⌈. . .⌉ = int l
      by simp
    also have l ≤ count (lift-profile A) {x, y}
      using xy A.A-subset by (subst count-lift-profile) auto
    finally show int 1 ∗ ⌈real (l ∗ (k + 1)) / real (k + 1)⌉ ≤ int (count (lift-profile A) {x, y})
      by simp
  next
    show is-pref-profile (lift-profile A)
      by (intro is-pref-profile-lift-profile) fact
  qed (use xy in auto)
```

hence {#*z* ∈# *r* (*lift-profile A*). *z* ∈ {*x*, *y*}#} ≠ {#}
  by *force*
thus *?thesis*
  by *auto*
qed

sublocale *lowered*: *anon-papp l* ∗ *k parties k lowered*
proof
  fix *A* **assume** *A*: *lowered.is-pref-profile A*
  hence *is-pref-profile* (*lift-profile A*)
    by *blast*
  hence *is-committee* (*r* (*lift-profile A*))
    using *rule-wf* **by** *blast*
  thus *lowered.is-committee* (*lowered A*)
    unfolding *lowered-def o-def* **using** *x-or-y-in-r-lift-profile*[*of A*] *A*
    by (*intro is-committee-lower-result*) *auto*
qed

sublocale *lowered*: *weak-rep-anon-papp l* ∗ *k parties k lowered*
proof
  fix *A z*
  assume *A*: *lowered.is-pref-profile A* **and** *z*: *l* ∗ *k* ≤ *k* ∗ *count A* {*z*}
  interpret *A*: *anon-papp-profile l* ∗ *k parties k A*
    by *fact*

  from *A* **have** *A′*: *is-pref-profile* (*lift-profile A*)
    by *blast*
  have *count A* {*z*} > *0*
    using *z k n-voters-pos* **by** (*intro Nat.gr0I*) *auto*
  hence {*z*} ∈# *A*
    by *simp*
  hence *z′*: *z* ∈ *parties*
    using *A.A-subset z* **by** *auto*
  hence [*simp*]: *y* ≠ *z z* ≠ *y*
    using *xy* **by** *auto*

  define *C* **where** *C* = *r* (*lift-profile A*)

  show *z* ∈# *lowered A*
  proof (*cases z* = *x*)
    case *False*
    have *l* ≤ *count A* {*z*}
      using *z k* **by** (*simp add*: *algebra-simps*)
    hence *l* ∗ (*k* + *1*) ≤ (*k* + *1*) ∗ *count A* {*z*}
      by (*subst mult.commute*, *intro mult-right-mono*) *auto*
    also have *count A* {*z*} = *count* (*lift-profile A*) {*z*}
      using *False A.A-subset xy* **by** (*subst count-lift-profile*) *auto*
    finally have *z* ∈# *r* (*lift-profile A*)
      by (*intro weak-representation A′*)

thus $z \in\#$ *lowered A*
using *False* **by** (*auto simp*: *lowered-def in-lower-result-iff*)
next
case [*simp*]: *True*
from *xy* **have** [*simp*]: $\{x, y\} - \{y\} = \{x\}$
by *auto*

have *size* $\{\#z \in\# C.\ z \in \{x, y\}\#\} \geq 2$
unfolding *C-def*
proof (*rule proposition3*)
have *real* $(l * (k + 1))$ / *real* $(k + 1) = l$
unfolding *of-nat-mult* **using** *k* **by** (*simp add*: *divide-simps*)
also have *int* $2 * \lceil \ldots \rceil =$ *int* $(2 * l)$
by *simp*
also have $\ldots \leq$ *count* (*lift-profile A*) $\{x, y\}$
using *z k xy A.A-subset* **by** (*subst count-lift-profile*) *auto*
finally show *int* $2 * \lceil$*real* $(l * (k + 1))$ / *real* $(k + 1)\rceil \leq \ldots$ **.**
qed (*use* A' *xy* **in** *auto*)
also have *size* $\{\#z \in\# C.\ z \in \{x, y\}\#\} =$ *count C x* + *count C y*
by (*induction C*) *auto*
finally have $x \in\#$ *lower-result C*
by (*subst in-lower-result-iff*) *auto*
thus $z \in\#$ *lowered A*
by (*simp add*: *lowered-def C-def*)
qed
qed

sublocale *lowered*: *card-stratproof-anon-papp l* $*$ *k parties k lowered*
proof
fix *A X Y*
show $\neg$*lowered.card-manipulable A X Y*
unfolding *lowered.card-manipulable-def*
proof (*rule notI*, *elim conjE*)
assume *A*: *lowered.is-pref-profile A* **and** *XY*: $X \in\# A$ $Y \neq \{\}$ $Y \subseteq$ *parties*
assume $*$: *lowered A* $\prec$[*lowered.committee-preference X*] *lowered* $(A - \{\#X\#\} + \{\#Y\#\})$
interpret *anon-papp-profile l* $*$ *k parties k A*
by *fact*
have *X*: $X \neq \{\}$ $X \subseteq$ *parties*
using *XY A-nonempty A-subset* **by** *auto*
define A' **where** $A' = A - \{\#X\#\}$
have A': $A = A' + \{\#X\#\}$
using *XY* **by** (*simp add*: A'*-def*)
from *xy X XY* **have** [*simp*]: $y \notin X$ $y \notin Y$
by *auto*

define *Al1* **where** *Al1* = *lift-profile A*
define *Al2* **where** *Al2* = *lift-profile* $(A' + \{\#Y\#\})$
have A'*-plus-Y*: *lowered.is-pref-profile* $(A' + \{\#Y\#\})$
unfolding A'*-def* **using** *A XY lowered.is-pref-profile-replace* **by** *blast*

have *Al1*: *is-pref-profile Al1*
unfolding *Al1-def* **using** *A* **by** *blast*
have *Al2*: *is-pref-profile Al2*
unfolding *Al2-def* **unfolding** *A′-def* **using** *A XY lowered.is-pref-profile-replace* **by** *blast*

have *size-aux*: *size* (*lower-result* {#*x* ∈# *r* (*lift-profile A*). *x* ∈ *lift-applist X*#}) =
size {#*x* ∈# *r* (*lift-profile A*). *x* ∈ *lift-applist X*#} − (*if x* ∈ *X then 1 else 0*)
if *A*: *lowered.is-pref-profile A* **for** *A*
using *x-or-y-in-r-lift-profile*[*OF A*]
by (*subst size-lower-result′*) (*auto simp*: *lift-applist-def*)

from ∗ **have** *size* {#*x* ∈# *lower-result* (*r Al1*). *x* ∈ *X*#} <
size {#*x* ∈# *lower-result* (*r Al2*). *x* ∈ *X*#}
by (*simp add*: *Al1-def Al2-def lowered-def A′ lowered.strong-committee-preference-iff*)
also have {#*x* ∈# *lower-result* (*r Al1*). *x* ∈ *X*#} = *lower-result* {#*x* ∈# *r Al1*. *x* ∈ *lift-applist X*#}
using *X xy* **by** (*subst filter-lower-result-eq*) *auto*
also have {#*x* ∈# *lower-result* (*r Al2*). *x* ∈ *X*#} = *lower-result* {#*x* ∈# *r Al2*. *x* ∈ *lift-applist X*#}
using *X xy* **by** (*subst filter-lower-result-eq*) *auto*
also have *size* (*lower-result* {#*x* ∈# *r Al1*. *x* ∈ *lift-applist X*#}) =
size {#*x* ∈# *r Al1*. *x* ∈ *lift-applist X*#} − (*if x* ∈ *X then 1 else 0*)
unfolding *Al1-def* **by** (*rule size-aux*) *fact*
also have *size* (*lower-result* {#*x* ∈# *r Al2*. *x* ∈ *lift-applist X*#}) =
size {#*x* ∈# *r Al2*. *x* ∈ *lift-applist X*#} − (*if x* ∈ *X then 1 else 0*)
unfolding *Al2-def* **by** (*rule size-aux*) *fact*
finally have *size* {#*x* ∈# *r Al1*. *x* ∈ *lift-applist X*#} < *size* {#*x* ∈# *r Al2*. *x* ∈ *lift-applist X*#}
by *auto*
hence *r Al1* ≺[*committee-preference* (*lift-applist X*)] *r Al2*
by (*simp add*: *strong-committee-preference-iff*)

moreover have ¬*r Al1* ≺[*committee-preference* (*lift-applist X*)] *r Al2*
by (*rule not-manipulable′* [**where** *Y* = *lift-applist Y*])
(*use Al1 Al2* **in** ‹*auto simp*: *Al1-def Al2-def A′*›)

ultimately show *False*
by *contradiction*
qed
qed

sublocale *lowered*: *card-stratproof-weak-rep-anon-papp l* ∗ *k parties k lowered*
..

end

For Weak *Proportional* Representation, the lowering argument to decrease the committee size is somewhat easier since it does not involve adding a new party; instead, we simply add l new voters whose preferences are $\{x\}$.

locale *decrease-committee-card-stratproof-weak-prop-rep-anon-papp* =
card-stratproof-weak-prop-rep-anon-papp $l * (k + 1)$ *parties* $k + 1$ *r*
for *l k* **and** *parties* :: *'a set* **and** *r* +
fixes *x* :: *'a*
assumes *x*: $x \in$ *parties*
assumes *k*: $k > 0$
begin

definition *lift-profile* :: *'a set multiset* ⇒ *'a set multiset* **where**
lift-profile A = *A* + *replicate-mset l* {*x*}

definition *lower-result*
where *lower-result C* = *C* − {#*x*#}

definition *lowered* **where** *lowered* = *lower-result* ∘ *r* ∘ *lift-profile*

lemma *l*: $l > 0$
using *n-voters-pos* **by** *auto*

lemma *lift-profile-empty* [*simp*]: *lift-profile* {#} = *replicate-mset l* {*x*}
by (*simp add*: *lift-profile-def*)

lemma *lift-profile-add-mset* [*simp*]:
lift-profile (*add-mset X A*) = *add-mset X* (*lift-profile A*)
by (*simp add*: *lift-profile-def*)

lemma *empty-in-lift-profile-iff* [*simp*]: {} ∈# *lift-profile A* ⟷ {} ∈# *A*
by (*auto simp*: *lift-profile-def*)

lemma *size-lift-profile* [*simp*]: *size* (*lift-profile A*) = *size A* + *l*
by (*simp add*: *lift-profile-def*)

lemma *count-lift-profile*:
count (*lift-profile A*) *X* = *count A X* + (*if X* = {*x*} *then l else 0*)
by (*auto simp*: *lift-profile-def*)

lemma *size-lower-result* [*simp*]:
assumes *x* ∈# *C*
shows *size* (*lower-result C*) = *size C* − *1*
using *assms* **by** (*auto simp*: *lower-result-def size-Diff-singleton*)

lemma *size-lower-result'*: *size* (*lower-result C*) = *size C* − (*if x* ∈# *C then 1 else 0*)
by (*induction C*) (*auto simp*: *lower-result-def size-Diff-singleton*)

lemma *count-lower-result*:
count (*lower-result C*) *z* = *count C z* − (*if z* = *x then 1 else 0*)
by (*auto simp*: *lower-result-def*)

lemma *in-lower-resultD*:
z ∈# *lower-result C* ⟹ *z* ∈# *C*
by (*auto simp*: *lower-result-def dest*!: *in-diffD*)

lemma *in-lower-result-iff*:
z ∈# *lower-result C* ⟷ (*if z* = *x then count C x* > *1 else z* ∈# *C*)
(**is** - = *?rhs*)
proof −
have *z* ∈# *lower-result C* ⟷ *count* (*lower-result C*) *z* > *0*
by *auto*
also have . . . ⟷ *?rhs*
by (*subst count-lower-result*) *auto*
finally show *?thesis* **.**
qed

sublocale *lowered*: *anon-papp-election l* ∗ *k parties k*
by *standard* (*use n-voters-pos finite-parties k* **in** *auto*)

lemma *is-pref-profile-lift-profile* [*intro*]:
assumes *lowered.is-pref-profile A*
shows *is-pref-profile* (*lift-profile A*)
proof −
interpret *A*: *anon-papp-profile l* ∗ *k parties k A*
by *fact*
show *?thesis*
using *A.A-nonempty A.A-subset A.size-A l*
by *unfold-locales*
(*auto simp*: *lift-profile-def x size-mset-sum-mset image-mset.compositionality o-def*)
qed

lemma *is-committee-lower-result*:
assumes *is-committee C x* ∈# *C*
shows *lowered.is-committee* (*lower-result C*)
using *assms* **unfolding** *is-committee-def lowered.is-committee-def*
by (*auto simp*: *lower-result-def size-Diff-singleton dest*: *in-diffD*)

lemma *x-in-r-lift-profile*:
assumes *lowered.is-pref-profile A*
shows *x* ∈# *r* (*lift-profile A*)
proof (*rule weak-representation*)
show *is-pref-profile* (*lift-profile A*)
using *assms* **by** *blast*
next
have (*k* + *1*) ∗ *l* ≤ (*k* + *1*) ∗ *count* (*lift-profile A*) {*x*}
by (*intro mult-left-mono*) (*auto simp*: *count-lift-profile*)
thus *l* ∗ (*k* + *1*) ≤ (*k* + *1*) ∗ *count* (*lift-profile A*) {*x*}
by (*simp add*: *algebra-simps*)

qed

sublocale *lowered*: *anon-papp* $l * k$ *parties* k *lowered*
proof
  fix A **assume** A: *lowered.is-pref-profile* A
  hence *is-pref-profile* (*lift-profile* A)
    by *blast*
  hence *is-committee* (r (*lift-profile* A))
    using *rule-wf* **by** *blast*
  thus *lowered.is-committee* (*lowered* A)
    unfolding *lowered-def o-def* **using** *x-in-r-lift-profile*[*of* A] A
    by (*intro is-committee-lower-result*) *auto*
qed

sublocale *lowered*: *weak-prop-rep-anon-papp* $l * k$ *parties* k *lowered*
proof
  fix A z l'
  assume A: *lowered.is-pref-profile* A **and** z: $l' * (l * k) \leq k * count\ A\ \{z\}$
  interpret A: *anon-papp-profile* $l * k$ *parties* k A
    by *fact*

  show *count* (*lowered* A) $z \geq l'$
  proof (*cases* $l' > 0$)
    case *False*
    thus *?thesis* **by** *auto*
  next
    case l: *True*

    from A **have** A': *is-pref-profile* (*lift-profile* A)
      by *blast*
    have *count* A $\{z\} > 0$
      using z k *n-voters-pos* l **by** (*intro Nat.gr0I*) *auto*
    hence $\{z\} \in\#\ A$
      by *simp*
    hence z': $z \in$ *parties*
      using *A.A-subset* z **by** *auto*

    define C **where** $C = r$ (*lift-profile* A)

    show *count* (*lowered* A) $z \geq l'$
    proof (*cases* $z = x$)
      case *False*
      have $l' * l \leq count\ A\ \{z\}$
        using z k **by** (*simp add*: *algebra-simps*)
      hence $l' * l * (k + 1) \leq (k + 1) * count\ A\ \{z\}$
        by (*subst mult.commute*, *intro mult-right-mono*) *auto*
      also have $count\ A\ \{z\} = count$ (*lift-profile* A) $\{z\}$
        using *False A.A-subset* **by** (*subst count-lift-profile*) *auto*
      finally have *count* (r (*lift-profile* A)) $z \geq l'$

```
      by (intro weak-proportional-representation A′) (auto simp: algebra-simps)
    thus l′ ≤ count (lowered A) z
      using False by (simp add: lowered-def lower-result-def)
  next
    case [simp]: True
    have l′ ∗ l ≤ count A {x}
      using z k by (simp add: algebra-simps)
    hence l′ ∗ l ∗ (k + 1) ≤ (k + 1) ∗ count A {x}
      by (subst mult.commute, intro mult-right-mono) auto
    also have ... + (k + 1) ∗ l = (k + 1) ∗ count (lift-profile A) {x}
      by (simp add: count-lift-profile algebra-simps)
    finally have (l′ + 1) ∗ (l ∗ (k + 1)) ≤ (k + 1) ∗ count (lift-profile A) {x}
      by (simp add: algebra-simps)
    hence count (r (lift-profile A)) x ≥ l′ + 1
      by (intro weak-proportional-representation A′)
    thus l′ ≤ count (lowered A) z
      by (simp add: lowered-def lower-result-def)
  qed
 qed
qed

sublocale lowered: card-stratproof-anon-papp l ∗ k parties k lowered
proof
 fix A X Y
 show ¬lowered.card-manipulable A X Y
   unfolding lowered.card-manipulable-def
 proof (rule notI, elim conjE)
   assume A: lowered.is-pref-profile A and XY: X ∈# A Y ≠ {} Y ⊆ parties
   assume ∗: lowered A ≺[lowered.committee-preference X] lowered (A − {#X#} + {#Y#})
   interpret anon-papp-profile l ∗ k parties k A
     by fact
   have X: X ≠ {} X ⊆ parties
     using XY A-nonempty A-subset by auto
   define A′ where A′ = A − {#X#}
   have A′: A = A′ + {#X#}
     using XY by (simp add: A′-def)

   define Al1 where Al1 = lift-profile A
   define Al2 where Al2 = lift-profile (A′ + {#Y#})
   have A′-plus-Y: lowered.is-pref-profile (A′ + {#Y#})
     unfolding A′-def using A XY lowered.is-pref-profile-replace by blast
   have Al1: is-pref-profile Al1
     unfolding Al1-def using A by blast
   have Al2: is-pref-profile Al2
     unfolding Al2-def unfolding A′-def using A XY lowered.is-pref-profile-replace by blast

   have size-aux: size (lower-result {#x ∈# r (lift-profile A). x ∈ X#}) =
                 size {#x ∈# r (lift-profile A). x ∈ X#} − (if x ∈ X then 1 else 0)
     if A: lowered.is-pref-profile A for A
```

using *x-in-r-lift-profile*[*OF A*] **by** (*subst size-lower-result′*) *auto*

from $*$ **have** *size* $\{\#x \in\#$ *lower-result* $(r\ Al1).\ x \in X\#\} <$
size $\{\#x \in\#$ *lower-result* $(r\ Al2).\ x \in X\#\}$
by (*simp add*: *Al1-def Al2-def lowered-def A′ lowered.strong-committee-preference-iff*)
also have $\{\#x \in\#$ *lower-result* $(r\ Al1).\ x \in X\#\} =$ *lower-result* $\{\#x \in\# r\ Al1.\ x \in X\#\}$
using $X\ x$ **unfolding** *lower-result-def* **by** (*subst filter-diff-mset′*) *auto*
also have $\{\#x \in\#$ *lower-result* $(r\ Al2).\ x \in X\#\} =$ *lower-result* $\{\#x \in\# r\ Al2.\ x \in X\#\}$
using $X\ x$ **unfolding** *lower-result-def* **by** (*subst filter-diff-mset′*) *auto*
also have *size* (*lower-result* $\{\#x \in\# r\ Al1.\ x \in X\#\}) =$
size $\{\#x \in\# r\ Al1.\ x \in X\#\} - ($*if* $x \in X$ *then 1 else 0*)
unfolding *Al1-def* **by** (*rule size-aux*) *fact*
also have *size* (*lower-result* $\{\#x \in\# r\ Al2.\ x \in X\#\}) =$
size $\{\#x \in\# r\ Al2.\ x \in X\#\} - ($*if* $x \in X$ *then 1 else 0*)
unfolding *Al2-def* **by** (*rule size-aux*) *fact*
finally have *size* $\{\#x \in\# r\ Al1.\ x \in X\#\} <$ *size* $\{\#x \in\# r\ Al2.\ x \in X\#\}$
by *auto*
hence *r Al1* $\prec$[*committee-preference X*] *r Al2*
by (*simp add*: *strong-committee-preference-iff*)

moreover have $\neg$*r Al1* $\prec$[*committee-preference X*] *r Al2*
by (*rule not-manipulable′* [**where** $Y = Y$])
(*use Al1 Al2* **in** ‹*auto simp*: *Al1-def Al2-def A′*›)

ultimately show *False*
by *contradiction*
qed
qed

sublocale *lowered*: *card-stratproof-weak-prop-rep-anon-papp l* $*$ *k parties k lowered*
..

end

end

5 Lifting the Impossibility Result to Larger Settings

theory *PAPP-Impossibility*
imports *PAPP-Impossibility-Base-Case Anonymous-PAPP-Lowering*
begin

In this section, we now prove the main results of this work by combining the base case with the lifting arguments formalized earlier.

First, we prove the following very simple technical lemma: a set that is infinite or finite with cardinality at least 2 contains two different elements x and y.

lemma *obtain-2-elements*:
assumes *infinite* $X \lor$ *card* $X \geq 2$

```
  obtains x y where x ∈ X y ∈ X x ≠ y
proof −
  from assms have X ≠ {}
    by auto
  then obtain x where x ∈ X
    by blast
  with assms have infinite X ∨ card (X − {x}) > 0
    by (subst card-Diff-subset) auto
  hence X − {x} ≠ {}
    by (metis card-gt-0-iff finite.emptyI infinite-remove)
  then obtain y where y ∈ X − {x}
    by blast
  with ‹x ∈ X› show ?thesis
    using that[of x y] by blast
qed
```

We now have all the ingredients to formalise the first main impossibility result: There is no P-APP rule that satisfies Anonymity, Cardinality-Strategyproofness, and Weak Representation if $k \geq 3$ and $m \geq k + 1$ and n is a multiple of $2k$.

The proof simply uses the lowering lemmas we proved earlier to first reduce the committee size to 3, then reduce the voters to 6, and finally restrict the parties to 4. At that point, the base case we proved with SAT solving earlier kicks in.

This corresponds to Theorem 1 in the paper.

```
theorem papp-impossibility1:
  assumes k ≥ 3 and card parties ≥ k + 1 and finite parties
  shows   ¬card-stratproof-weak-rep-anon-papp (2 ∗ k ∗ l) parties k r
  using assms
proof (induction k arbitrary: parties r rule: less-induct)
  case (less k parties r)
  show ?case
  proof (cases k = 3)
    assume [simp]: k = 3
```

If the committee size is 3, we first use our voter-division lemma to go from a P-APP rule for $6l$ voters to one with just 6 voters. Next, we choose 4 arbitrary parties and use our party-restriction lemma to obtain a P-APP rule for just 4 parties.

But this is exactly our base case, which we already know to be infeasible.

```
    show ?thesis
    proof
      assume card-stratproof-weak-rep-anon-papp (2 ∗ k ∗ l) parties k r
      then interpret card-stratproof-weak-rep-anon-papp l ∗ 6 parties 3 r
        by (simp add: mult-ac)
      interpret divide-voters-card-stratproof-weak-rep-anon-papp l 6 parties 3 r ..

      have card parties ≥ 4
        using less.prems by auto
      then obtain parties' where parties': parties' ⊆ parties card parties' = 4
```

```
      by (metis obtain-subset-with-card-n)
    have ∃ r. card-stratproof-weak-rep-anon-papp 6 parties′ 3 r
    proof (rule card-stratproof-weak-rep-anon-papp-restrict-parties)
      show card-stratproof-weak-rep-anon-papp 6 parties 3 (r ∘ lift-profile)
        by (rule lowered.card-stratproof-weak-rep-anon-papp-axioms)
    qed (use parties′ in auto)
    thus False
      using papp-impossibility-base-case[OF parties′(2)] by blast
  qed
next
  assume k ≠ 3
```

If the committee size is greater than 3, we use our other lowering lemma to reduce the committee size by 1 (while also reducing the number of voters by $2l$ and the number of parties by 1).

```
  with less.prems have k > 3
    by simp

  obtain x y where xy: x ∈ parties y ∈ parties x ≠ y
    using obtain-2-elements[of parties] less.prems by auto
  define parties′ where parties′ = parties − {y}
  have [simp]: card parties′ = card parties − 1
    unfolding parties′-def using xy by (subst card-Diff-subset) auto

  show ?thesis
  proof
    assume card-stratproof-weak-rep-anon-papp (2 ∗ k ∗ l) parties k r
    then interpret card-stratproof-weak-rep-anon-papp
      2 ∗ l ∗ (k − 1 + 1) insert y parties′ k − 1 + 1 r
      using ‹k > 3› xy by (simp add: parties′-def insert-absorb mult-ac)
   interpret decrease-committee-card-stratproof-weak-rep-anon-papp 2 ∗ l k − 1 y parties′ r x
      by unfold-locales (use ‹k > 3› xy in ‹auto simp: parties′-def›)
    have ¬card-stratproof-weak-rep-anon-papp (2 ∗ (k − 1) ∗ l) parties′ (k − 1) lowered
      by (rule less.IH) (use ‹k > 3› xy less.prems in auto)
    with lowered.card-stratproof-weak-rep-anon-papp-axioms show False
      by (simp add: mult-ac)
  qed
 qed
qed
```

If Weak Representation is replaced with Weak Proportional Representation, we can strengthen the impossibility result by relaxing the conditions on the number of parties to $m \geq 4$.

This works because with Weak Proportional Representation, we can reduce the size of the committee without changing the number of parties. We use this to again bring k down to 3 without changing m, at which point we can simply apply our previous impossibility result for Weak Representation.

This corresponds to Theorem 2 in the paper.

```
corollary papp-impossibility2:
  assumes k ≥ 3 and card parties ≥ 4 and finite parties
  shows  ¬card-stratproof-weak-prop-rep-anon-papp (2 * k * l) parties k r
  using assms
proof (induction k arbitrary: parties r rule: less-induct)
  case (less k parties r)
  show ?case
  proof (cases k = 3)
    assume [simp]: k = 3
```

For committee size 3, we simply employ our previous impossibility result:

```
    show ?thesis
    proof
      assume card-stratproof-weak-prop-rep-anon-papp (2 * k * l) parties k r
      then interpret card-stratproof-weak-prop-rep-anon-papp l * 6 parties 3 r
        by (simp add: mult-ac)
      have card-stratproof-weak-rep-anon-papp (l * 6) parties 3 r ..
      moreover have ¬card-stratproof-weak-rep-anon-papp (l * 6) parties 3 r
        using papp-impossibility1[of 3 parties l r] less.prems by (simp add: mult-ac)
      ultimately show False
        by contradiction
    qed
  next
    assume k ≠ 3
```

If the committee size is greater than 3, we use our other lowering lemma to reduce the committee size by 1 (while also reducing the number of voters by $2l$).

```
    with less.prems have k > 3
      by simp

    have parties ≠ {}
      using less.prems by auto
    then obtain x where x: x ∈ parties
      by blast

    show ?thesis
    proof
      assume card-stratproof-weak-prop-rep-anon-papp (2 * k * l) parties k r
      then interpret card-stratproof-weak-prop-rep-anon-papp
        2 * l * (k − 1 + 1) parties k − 1 + 1 r
        using ‹k > 3› by (simp add: mult-ac)
      interpret decrease-committee-card-stratproof-weak-prop-rep-anon-papp 2 * l k − 1 parties
r x
        by unfold-locales (use ‹k > 3› x in auto)
      have ¬card-stratproof-weak-prop-rep-anon-papp (2 * (k − 1) * l) parties (k − 1) lowered
        by (rule less.IH) (use ‹k > 3› less.prems in auto)
      with lowered.card-stratproof-weak-prop-rep-anon-papp-axioms show False
        by (simp add: mult-ac)
    qed
```

qed
qed

end

References

[1] P. Lammich. The GRAT tool chain – efficient (UN)SAT certificate checking with formal correctness guarantees. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 457–463. Springer, 2017.

[2] N. Wetzler, M. Heule, and W. A. H. Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014, Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.