A generalization of the Cauchy-Davenport theorem

Mantas Bakšys
University of Cambridge
mb2412@cam.ac.uk

October 28, 2024

Abstract

Cauchy-Davenport theorem is a famous theorem, which is known as one of the founding results in additive combinatorics. It was originally independently discovered by Cauchy [2] and Davenport [3] and has been formalized in the AFP entry [1] as a corollary of Kneser's theorem. More recently, many generalizations of this theorem have been found. In this entry, we formalise a generalization due to DeVos [4], which proves the theorem in a non-abelian setting.

Contents

1 Generalized Cauchy–Davenport Theorem: preliminaries

theory *Generalized-Cauchy-Davenport-preliminaries*
imports
Complex-Main
Jacobson-Basic-Algebra.Group-Theory

begin

1.1 Well-ordering lemmas

lemma *wf-prod-lex-fibers-inter*:
fixes *r* :: (*'a* × *'a*) *set* **and** *s* :: (*'b* × *'b*) *set* **and** *f* :: *'c* ⇒ *'a* **and** *g* :: *'c* ⇒ *'b* **and**
t :: (*'c* × *'c*) *set*
assumes *h1*: *wf* ((*inv-image r f*) ∩ *t*) **and**
h2: ⋀ *a*. *a* ∈ *range f* ⟹ *wf* (({*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ (*inv-image s g*)) ∩ *t*) **and**
h3: *trans t*
shows *wf* ((*inv-image* (*r* <∗*lex*∗> *s*) (λ *c*. (*f c*, *g c*))) ∩ *t*)
proof−
have *h4*: (⋃ *a* ∈ *range f*. ({*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ (*inv-image s g*)) ∩ *t*) =
(⋃ *a* ∈ *range f*. ({*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ (*inv-image s g*))) ∩ *t* **by** *blast*
have (*inv-image* (*r* <∗*lex*∗> *s*) (λ *c*. (*f c*, *g c*))) ∩ *t* = (*inv-image r f* ∩ *t*) ∪
(⋃ *a* ∈ *range f*. {*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ (*inv-image s g*) ∩ *t*)
proof
show *inv-image* (*r* <∗*lex*∗> *s*) (λ*c*. (*f c*, *g c*)) ∩ *t*
⊆ *inv-image r f* ∩ *t* ∪ (⋃*a*∈*range f*. {*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ *inv-image s g* ∩ *t*)
proof
fix *y* **assume** *hy*: *y* ∈ *inv-image* (*r* <∗*lex*∗> *s*) (λ*c*. (*f c*, *g c*)) ∩ *t*
then obtain *a b* **where** *hx*: *y* = (*a*, *b*) **and** (*f a*, *f b*) ∈ *r* ∨ (*f a* = *f b* ∧ (*g a*, *g b*) ∈ *s*)
using *in-inv-image in-lex-prod Int-iff SigmaE UNIV-Times-UNIV inf-top-right* **by** (*smt* (*z3*))
then show *y* ∈ *inv-image r f* ∩ *t* ∪ (⋃*a*∈*range f*. {*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ *inv-image s g* ∩ *t*)
using *hy* **by** *auto*
qed
show *inv-image r f* ∩ *t* ∪ (⋃*a*∈*range f*. {*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ *inv-image s g* ∩ *t*) ⊆
inv-image (*r* <∗*lex*∗> *s*) (λ*c*. (*f c*, *g c*)) ∩ *t* **using** *Int-iff SUP-le-iff SigmaD1 SigmaD2*
in-inv-image in-lex-prod mem-Collect-eq subrelI **by** *force*
qed
moreover have ((*inv-image r f*) ∩ *t*) *O*
(⋃ *a* ∈ *range f*. ({*x*. *f x* = *a*} × {*x*. *f x* = *a*} ∩ (*inv-image s g*)) ∩ *t*) ⊆

$(inv\text{-}image\ r\ f) \cap t$
  using *h3 trans-O-subset* **by** *fastforce*
  moreover have $wf\ (\bigcup\ a \in range\ f.\ \{x.\ f\ x = a\} \times \{x.\ f\ x = a\} \cap (inv\text{-}image\ s\ g) \cap t)$
    apply(*rule wf-UN*, *auto simp add*: *h2*)
    done
  ultimately show $wf\ (inv\text{-}image\ (r <*lex*> s)\ (\lambda\ c.\ (f\ c,\ g\ c)) \cap t)$
    using *wf-union-compatible*[*OF h1*] **by** *fastforce*
qed

lemma *wf-prod-lex-fibers*:
  fixes $r :: ('a \times 'a)\ set$ **and** $s :: ('b \times 'b)\ set$ **and** $f :: 'c \Rightarrow 'a$ **and** $g :: 'c \Rightarrow 'b$
  assumes *h1*: $wf\ (inv\text{-}image\ r\ f)$ **and**
  h2: $\bigwedge\ a.\ a \in range\ f \Longrightarrow wf\ (\{x.\ f\ x = a\} \times \{x.\ f\ x = a\} \cap (inv\text{-}image\ s\ g))$
  shows $wf\ (inv\text{-}image\ (r <*lex*> s)\ (\lambda\ c.\ (f\ c,\ g\ c)))$
  using *assms trans-def wf-prod-lex-fibers-inter*[*of r f UNIV s g*] *inf-top-right*
  by (*metis* (*mono-tags*, *lifting*) *iso-tuple-UNIV-I*)

context *monoid*

begin

1.2 Pointwise set multiplication in monoid: definition and key-lemmas

inductive-set $smul :: 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ **for** $A\ B$
  where
    smulI[*intro*]: $[\![a \in A;\ a \in M;\ b \in B;\ b \in M]\!] \Longrightarrow a \cdot b \in smul\ A\ B$

syntax $smul ::\ \ 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set\ ((\text{-}\ \cdots\ \text{-}))$

lemma *smul-eq*: $smul\ A\ B = \{c.\ \exists\, a \in A \cap M.\ \exists\, b \in B \cap M.\ c = a \cdot b\}$
  by (*auto simp*: *smul.simps elim*!: *smul.cases*)

lemma *smul*: $smul\ A\ B = (\bigcup a \in A \cap M.\ \bigcup b \in B \cap M.\ \{a \cdot b\})$
  by (*auto simp*: *smul-eq*)

lemma *smul-subset-carrier*: $smul\ A\ B \subseteq M$
  by (*auto simp*: *smul-eq*)

lemma *smul-Int-carrier* [*simp*]: $smul\ A\ B \cap M = smul\ A\ B$
  by (*simp add*: *Int-absorb2 smul-subset-carrier*)

lemma *smul-mono*: $[\![A' \subseteq A;\ B' \subseteq B]\!] \Longrightarrow smul\ A'\ B' \subseteq smul\ A\ B$
  by (*auto simp*: *smul-eq*)

lemma *smul-insert1*: $NO\text{-}MATCH\ \{\}\ A \Longrightarrow smul\ (insert\ x\ A)\ B = smul\ \{x\}\ B \cup smul\ A\ B$
  by (*auto simp*: *smul-eq*)

lemma *smul-insert2*: $NO\text{-}MATCH\ \{\}\ B \Longrightarrow smul\ A\ (insert\ x\ B) = smul\ A\ \{x\} \cup smul\ A\ B$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-Un1*: $smul\ (A \cup A')\ B = smul\ A\ B \cup smul\ A'\ B$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-Un2*: $smul\ A\ (B \cup B') = smul\ A\ B \cup smul\ A\ B'$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-Union1*: $smul\ (\bigcup\ A)\ B = (\bigcup\ a \in A.\ smul\ a\ B)$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-Union2*: $smul\ A\ (\bigcup\ B) = (\bigcup\ b \in B.\ smul\ A\ b)$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-insert*: $smul\ A\ B \subseteq smul\ A\ (insert\ x\ B)\ smul\ A\ B \subseteq smul\ (insert\ x\ A)\ B$
by (*auto simp*: *smul-eq*)

lemma *smul-subset-Un*: $smul\ A\ B \subseteq smul\ A\ (B \cup C)\ smul\ A\ B \subseteq smul\ (A \cup C)\ B$
by (*auto simp*: *smul-eq*)

lemma *smul-empty* [*simp*]: $smul\ A\ \{\} = \{\}\ smul\ \{\}\ A = \{\}$
by (*auto simp*: *smul-eq*)

lemma *smul-empty′*:
assumes $A \cap M = \{\}$
shows $smul\ B\ A = \{\}\ smul\ A\ B = \{\}$
using *assms* **by** (*auto simp*: *smul-eq*)

lemma *smul-is-empty-iff* [*simp*]: $smul\ A\ B = \{\} \longleftrightarrow A \cap M = \{\} \lor B \cap M = \{\}$
by (*auto simp*: *smul-eq*)

lemma *smul-D* [*simp*]: $smul\ A\ \{\mathbf{1}\} = A \cap M\ smul\ \{\mathbf{1}\}\ A = A \cap M$
by (*auto simp*: *smul-eq*)

lemma *smul-Int-carrier-eq* [*simp*]: $smul\ A\ (B \cap M) = smul\ A\ B\ smul\ (A \cap M)\ B = smul\ A\ B$
by (*auto simp*: *smul-eq*)

lemma *smul-assoc*:
shows $smul\ (smul\ A\ B)\ C = smul\ A\ (smul\ B\ C)$
by (*fastforce simp add*: *smul-eq associative Bex-def*)

lemma *finite-smul*:
assumes *finite A finite B* **shows** *finite* (*smul A B*)

using *assms* **by** (*auto simp*: *smul-eq*)

lemma *finite-smul'*:
assumes *finite* ($A \cap M$) *finite* ($B \cap M$)
shows *finite* (*smul* A B)
using *assms* **by** (*auto simp*: *smul-eq*)

1.3 Exponentiation in a monoid: definitions and lemmas

primrec *power* :: $'a \Rightarrow nat \Rightarrow 'a$ (**infix** $\frown$ *100*)
where
power0: *power* g $0 = \mathbf{1}$
| *power-suc*: *power* g (*Suc* n) $=$ *power* g $n \cdot g$

lemma *power-one*:
assumes $g \in M$
shows *power* g $1 = g$ **using** *power-def power0 assms* **by** *simp*

lemma *power-mem-carrier*:
fixes n
assumes $g \in M$
shows $g \frown n \in M$
apply (*induction* n, *auto simp add*: *assms power-def*)
done

lemma *power-mult*:
assumes $g \in M$
shows $g \frown n \cdot g \frown m = g \frown (n + m)$
proof(*induction* m)
case *0*
then show *?case* **using** *assms power0 right-unit power-mem-carrier* **by** *simp*
next
case (*Suc* m)
assume $g \frown n \cdot g \frown m = g \frown (n + m)$
then show *?case* **using** *power-def* **by** (*smt* (*verit*) *add-Suc-right assms associative*
power-mem-carrier power-suc)
qed

lemma *mult-inverse-power*:
assumes $g \in M$ **and** *invertible* g
shows $g \frown n \cdot ((inverse\ g) \frown n) = \mathbf{1}$
proof(*induction* n)
case *0*
then show *?case* **using** *power-0* **by** *auto*
next
case (*Suc* n)
assume *hind*: $g \frown n \cdot local.inverse\ g \frown n = \mathbf{1}$
then have $g \frown Suc\ n \cdot inverse\ g \frown Suc\ n = (g \cdot g \frown n) \cdot (inverse\ g \frown n \cdot inverse$

g)
using *power-def power-mult assms* **by** (*smt* (*z3*) *add.commute invertible-inverse-closed*

invertible-right-inverse left-unit monoid.associative monoid-axioms power-mem-carrier power-suc)
then show *?case* **using** *associative power-mem-carrier assms hind* **by** (*smt* (*verit*, *del-insts*)
composition-closed invertible-inverse-closed invertible-right-inverse right-unit)
qed

lemma *inverse-mult-power*:
assumes $g \in M$ **and** *invertible g*
shows ((*inverse g*) $^\frown n) \cdot g ^\frown n = \mathbf{1}$ **using** *assms* **by** (*metis invertible-inverse-closed*

invertible-inverse-inverse invertible-inverse-invertible mult-inverse-power)

lemma *inverse-mult-power-eq*:
assumes $g \in M$ **and** *invertible g*
shows *inverse* ($g ^\frown n$) = (*inverse g*) $^\frown n$
using *assms inverse-equality* **by** (*simp add*: *inverse-mult-power mult-inverse-power power-mem-carrier*)

definition *power-int* :: $'a \Rightarrow int \Rightarrow 'a$ (**infixr** *powi 80*) **where**
power-int g n = (*if* $n \geq 0$ *then* $g ^\frown$(*nat n*) *else* (*inverse g*) $^\frown$(*nat* ($-n$)))

definition *nat-powers* :: $'a \Rightarrow 'a$ *set* **where** *nat-powers g* = ((λ *n*. $g ^\frown n$) ‘ *UNIV*)

lemma *nat-powers-eq-Union*: *nat-powers g* = ($\bigcup$ *n*. $\{g ^\frown n\}$) **using** *nat-powers-def*
by *auto*

definition *powers* :: $'a \Rightarrow 'a$ *set* **where** *powers g* = ((λ *n*. *g powi n*) ‘ *UNIV*)

lemma *nat-powers-subset*:
nat-powers g $\subseteq$ *powers g*
proof
fix *x* **assume** $x \in$ *nat-powers g*
then obtain *n* **where** $x = g ^\frown n$ **and** *nat n* = *n* **using** *nat-powers-def* **by** *auto*
then show $x \in$ *powers g* **using** *powers-def power-int-def*
by (*metis UNIV-I image-iff of-nat-0-le-iff*)
qed

lemma *inverse-nat-powers-subset*:
nat-powers (*inverse g*) $\subseteq$ *powers g*
proof
fix *x* **assume** $x \in$ *nat-powers* (*inverse g*)
then obtain *n* **where** *hx*: *x* = (*inverse g*) $^\frown n$ **using** *nat-powers-def* **by** *blast*
then show $x \in$ *powers g*
proof(*cases n* = *0*)
case *True*

```
    then show ?thesis using hx power0 powers-def
      by (metis nat-powers-def nat-powers-subset rangeI subsetD)
  next
    case False
    then have hpos: ¬ (− int n) ≥ 0 by auto
    then have x = g powi (− int n) using hx hpos power-int-def by simp
    then show ?thesis using powers-def by auto
  qed
qed

lemma powers-eq-union-nat-powers:
  powers g = nat-powers g ∪ nat-powers (inverse g)
proof
  show powers g ⊆ nat-powers g ∪ nat-powers (local.inverse g)
    using powers-def nat-powers-def power-int-def by auto
next
  show nat-powers g ∪ nat-powers (inverse g) ⊆ powers g
    by (simp add: inverse-nat-powers-subset nat-powers-subset)
qed

lemma one-mem-nat-powers: 1 ∈ nat-powers g
  using rangeI power0 nat-powers-def by metis

lemma nat-powers-subset-carrier:
  assumes g ∈ M
  shows nat-powers g ⊆ M
  using nat-powers-def power-mem-carrier assms by auto

lemma nat-powers-mult-closed:
  assumes g ∈ M
  shows ⋀ x y. x ∈ nat-powers g ⟹ y ∈ nat-powers g ⟹ x · y ∈ nat-powers g
  using nat-powers-def power-mult assms by auto

lemma nat-powers-inv-mult:
  assumes g ∈ M and invertible g
  shows ⋀ x y. x ∈ nat-powers g ⟹ y ∈ nat-powers (inverse g) ⟹ x · y ∈
powers g
proof−
  fix x y assume x ∈ nat-powers g and y ∈ nat-powers (inverse g)
  then obtain n m where hx: x = g ^ n and hy: y = (inverse g) ^ m using
nat-powers-def by blast
  show x · y ∈ powers g
  proof(cases n ≥ m)
    case True
    then obtain k where n = k + m using add.commute le-Suc-ex by blast
    then have g ^ n · (inverse g) ^ m = g ^ k using mult-inverse-power assms
associative
      by (smt (verit) invertible-inverse-closed local.power-mult power-mem-carrier
right-unit)
```

then show *?thesis* **using** *hx hy powers-eq-union-nat-powers nat-powers-def* **by** *auto*
next
case *False*
then obtain k **where** $m = n + k$ **by** (*metis leI less-imp-add-positive*)
then have g ^ $n \cdot$ (*inverse* g) ^ m = (*inverse* g) ^ k **using** *inverse-mult-power assms associative*
by (*smt* (*verit*) *left-unit local.power-mult monoid.invertible-inverse-closed monoid-axioms*
mult-inverse-power power-mem-carrier)
then show *?thesis* **using** *hx hy powers-eq-union-nat-powers nat-powers-def* **by** *auto*
qed
qed

lemma *inv-nat-powers-mult*:
assumes $g \in M$ **and** *invertible* g
shows $\bigwedge$ x y. $x \in$ *nat-powers* (*inverse* g) $\Longrightarrow$ $y \in$ *nat-powers* g $\Longrightarrow$ $x \cdot y \in$ *powers* g
by (*metis assms invertible-inverse-closed invertible-inverse-inverse invertible-inverse-invertible nat-powers-inv-mult powers-eq-union-nat-powers sup-commute*)

lemma *powers-mult-closed*:
assumes $g \in M$ **and** *invertible* g
shows $\bigwedge$ x y. $x \in$ *powers* g $\Longrightarrow$ $y \in$ *powers* g $\Longrightarrow$ $x \cdot y \in$ *powers* g
using *powers-eq-union-nat-powers assms*
nat-powers-mult-closed nat-powers-inv-mult inv-nat-powers-mult **by** *fastforce*

lemma *nat-powers-submonoid*:
assumes $g \in M$
shows *submonoid* (*nat-powers* g) M $(\cdot)$ **1**
apply(*unfold-locales*)
apply(*auto simp add*: *assms nat-powers-mult-closed one-mem-nat-powers nat-powers-subset-carrier*)
done

lemma *nat-powers-monoid*:
assumes $g \in M$
shows *monoid* (*nat-powers* g) $(\cdot)$ **1**
using *nat-powers-submonoid assms* **by** (*smt* (*verit*) *monoid.intro associative left-unit*
one-mem-nat-powers nat-powers-mult-closed right-unit submonoid.sub)

lemma *powers-submonoid*:
assumes $g \in M$ **and** *invertible* g
shows *submonoid* (*powers* g) M $(\cdot)$ **1**
proof
show *powers* $g \subseteq M$ **using** *powers-eq-union-nat-powers nat-powers-subset-carrier assms* **by** *auto*
next

show $\bigwedge a\ b.\ a \in powers\ g \Longrightarrow b \in powers\ g \Longrightarrow a \cdot b \in powers\ g$
using *powers-mult-closed assms* **by** *auto*
next
show $\mathbf{1} \in powers\ g$ **using** *powers-eq-union-nat-powers one-mem-nat-powers* **by** *auto*
qed

lemma *powers-monoid*:
assumes $g \in M$ **and** *invertible g*
shows *monoid* (*powers g*) $(\cdot)$ **1**
by (*smt* (*verit*) *monoid.intro Un-iff assms associative in-mono invertible-inverse-closed*

monoid.left-unit monoid.right-unit nat-powers-monoid powers-eq-union-nat-powers

powers-mult-closed powers-submonoid submonoid.sub-unit-closed submonoid.subset)

lemma *mem-nat-powers-invertible*:
assumes $g \in M$ **and** *invertible g* **and** $u \in nat\text{-}powers\ g$
shows *monoid.invertible* (*powers g*) $(\cdot)$ **1** *u*
proof−
obtain *n* **where** *hu*: $u = g \ \widehat{}\ n$ **using** *assms nat-powers-def* **by** *blast*
then have *inverse* $u \in powers\ g$ **using** *assms inverse-mult-power-eq powers-eq-union-nat-powers nat-powers-def* **by** *auto*
then show *?thesis* **using** *hu assms* **by** (*metis in-mono inverse-mult-power inverse-mult-power-eq*
monoid.invertibleI monoid.nat-powers-subset monoid.powers-monoid monoid-axioms mult-inverse-power)
qed

lemma *mem-nat-inv-powers-invertible*:
assumes $g \in M$ **and** *invertible g* **and** $u \in nat\text{-}powers$ (*inverse g*)
shows *monoid.invertible* (*powers g*) $(\cdot)$ **1** *u*
using *assms* **by** (*metis inf-sup-aci*(*5*) *invertible-inverse-closed invertible-inverse-inverse*

invertible-inverse-invertible mem-nat-powers-invertible powers-eq-union-nat-powers)

lemma *powers-group*:
assumes $g \in M$ **and** *invertible g*
shows *group* (*powers g*) $(\cdot)$ **1**
proof(*auto simp add*: *group-def Group-Theory.group-axioms-def assms powers-monoid*)
show $\bigwedge u.\ u \in powers\ g \Longrightarrow$ *monoid.invertible* (*powers g*) $(\cdot)$ **1** *u* **using** *assms mem-nat-inv-powers-invertible mem-nat-powers-invertible powers-eq-union-nat-powers*
by *auto*
qed

lemma *nat-powers-ne-one*:
assumes $g \in M$ **and** $g \neq \mathbf{1}$
shows $nat\text{-}powers\ g \neq \{\mathbf{1}\}$
proof−

have $g \in$ *nat-powers g* **using** *power-one nat-powers-def assms rangeI* **by** *metis*
then show *?thesis* **using** *assms* **by** *blast*
qed

lemma *powers-ne-one*:
assumes $g \in M$ **and** $g \neq \mathbf{1}$
shows *powers* $g \neq \{\mathbf{1}\}$ **using** *assms nat-powers-ne-one*
by (*metis all-not-in-conv nat-powers-subset one-mem-nat-powers subset-singleton-iff*)

1.4 Definition of the order of an element in a group

definition *order*
where $order\ g = (if\ (\exists\ n.\ n > 0 \land g\ \widehat{}\ n = \mathbf{1})\ then\ Min\ \{n.\ g\ \widehat{}\ n = \mathbf{1} \land n > 0\}\ else\ 0)$

definition *min-order* **where** $min\text{-}order = Min\ ((order\ `\ M) - \{0\})$

end

1.5 Sumset scalar multiplication cardinality lemmas

context *group*

begin

lemma *card-smul-singleton-right-eq*:
assumes *finite A* **shows** $card\ (smul\ A\ \{a\}) = (if\ a \in G\ then\ card\ (A \cap G)\ else\ 0)$
proof ($cases\ a \in G$)
case *True*
then have $smul\ A\ \{a\} = (\lambda x.\ x \cdot a)\ `\ (A \cap G)$
by (*auto simp*: *smul-eq*)
moreover have $inj\text{-}on\ (\lambda x.\ x \cdot a)\ (A \cap G)$
by (*auto simp*: *inj-on-def True*)
ultimately show *?thesis*
by (*metis True card-image*)
qed (*auto simp*: *smul-eq*)

lemma *card-smul-singleton-left-eq*:
assumes *finite A* **shows** $card\ (smul\ \{a\}\ A) = (if\ a \in G\ then\ card\ (A \cap G)\ else\ 0)$
proof ($cases\ a \in G$)
case *True*
then have $smul\ \{a\}\ A = (\lambda x.\ a \cdot x)\ `\ (A \cap G)$
by (*auto simp*: *smul-eq*)
moreover have $inj\text{-}on\ (\lambda x.\ a \cdot x)\ (A \cap G)$
by (*auto simp*: *inj-on-def True*)
ultimately show *?thesis*
by (*metis True card-image*)
qed (*auto simp*: *smul-eq*)

lemma *card-smul-sing-right-le*:
  assumes *finite A* **shows** *card* (*smul A* {*a*}) ≤ *card A*
  by (*simp add*: *assms card-mono card-smul-singleton-right-eq*)

lemma *card-smul-sing-left-le*:
  assumes *finite A* **shows** *card* (*smul* {*a*} *A*) ≤ *card A*
  by (*simp add*: *assms card-mono card-smul-singleton-left-eq*)

lemma *card-le-smul-right*:
  assumes *A*: *finite A a* ∈ *A a* ∈ *G*
    and *B*: *finite B B* ⊆ *G*
  shows *card B* ≤ *card* (*smul A B*)
proof −
  have *B* ⊆ (λ *x*. (*inverse a*) · *x*) ‘ *smul A B*
    using *A B*
    apply (*clarsimp simp*: *smul image-iff*)
    using *Int-absorb2 Int-iff invertible invertible-left-inverse2* **by** *metis*
  with *A B* **show** *?thesis*
    by (*meson finite-smul surj-card-le*)
qed

lemma *card-le-smul-left*:
  assumes *A*: *finite A b* ∈ *B b* ∈ *G*
    and *B*: *finite B A* ⊆ *G*
  shows *card A* ≤ *card* (*smul A B*)
proof −
  have *A* ⊆ (λ *x*. *x* · (*inverse b*)) ‘ *smul A B*
    using *A B*
    apply (*clarsimp simp*: *smul image-iff associative*)
    using *Int-absorb2 Int-iff invertible invertible-right-inverse assms*(*5*) **by** (*metis right-unit*)
  with *A B* **show** *?thesis*
    by (*meson finite-smul surj-card-le*)
qed

lemma *infinite-smul-right*:
  assumes *A* ∩ *G* ≠ {} **and** *infinite* (*B* ∩ *G*)
  shows *infinite* (*A* ··· *B*)
proof
  assume *hfin*: *finite* (*smul A B*)
  obtain *a* **where** *ha*: *a* ∈ *A* ∩ *G* **using** *assms* **by** *auto*
  then have *finite* (*smul* {*a*} *B*) **using** *hfin* **by** (*metis Int-Un-eq*(*1*) *finite-subset insert-is-Un*
    mk-disjoint-insert smul-subset-Un(*2*))
  moreover have *B* ∩ *G* ⊆ (λ *x*. *inverse a* · *x*) ‘ *smul* {*a*} *B*
  proof
    fix *b* **assume** *hb*: *b* ∈ *B* ∩ *G*

then have $b = inverse\ a \cdot (a \cdot b)$ **using** *associative ha* **by** (*simp add*: *invertible-left-inverse2*)
then show $b \in (\lambda\ x.\ inverse\ a \cdot x)$ ‘ *smul* $\{a\}$ B **using** *smul.simps hb ha* **by** *blast*
qed
ultimately show *False* **using** *assms* **using** *finite-surj* **by** *blast*
qed

lemma *infinite-smul-left*:
assumes $B \cap G \neq \{\}$ **and** *infinite* $(A \cap G)$
shows *infinite* $(A \cdots B)$
proof
assume *hfin*: *finite* (*smul* A B)
obtain b **where** *hb*: $b \in B \cap G$ **using** *assms* **by** *auto*
then have *finite* (*smul* A $\{b\}$) **using** *hfin* **by** (*simp add*: *rev-finite-subset smul-mono*)
moreover have $A \cap G \subseteq (\lambda\ x.\ x \cdot inverse\ b)$ ‘ *smul* A $\{b\}$
proof
fix a **assume** *ha*: $a \in A \cap G$
then have $a = (a \cdot b) \cdot inverse\ b$ **using** *associative hb*
by (*metis IntD2 invertible invertible-inverse-closed invertible-right-inverse right-unit*)
then show $a \in (\lambda\ x.\ x \cdot inverse\ b)$ ‘ *smul* A $\{b\}$ **using** *smul.simps hb ha* **by** *blast*
qed
ultimately show *False* **using** *assms* **using** *finite-surj* **by** *blast*
qed

1.6 Pointwise set multiplication in group: auxiliary lemmas

lemma *set-inverse-composition-commute*:
assumes $X \subseteq G$ **and** $Y \subseteq G$
shows *inverse* ‘ $(X \cdots Y)$ = (*inverse* ‘ Y) $\cdots$ (*inverse* ‘ X)
proof
show *inverse* ‘ $(X \cdots Y) \subseteq$ (*inverse* ‘ Y) $\cdots$ (*inverse* ‘ X)
proof
fix z **assume** $z \in$ *inverse* ‘ $(X \cdots Y)$
then obtain x y **where** $z = inverse\ (x \cdot y)$ **and** $x \in X$ **and** $y \in Y$
by (*smt* (*verit*) *image-iff smul.cases*)
then show $z \in$ (*inverse* ‘ Y) $\cdots$ (*inverse* ‘ X)
using *inverse-composition-commute assms*
by (*smt* (*verit*) *image-eqI in-mono inverse-equality invertible invertibleE smul.simps*)
qed
show (*inverse* ‘ Y) $\cdots$ (*inverse* ‘ X) $\subseteq$ *inverse* ‘ $(X \cdots Y)$
proof
fix z **assume** $z \in$ (*inverse* ‘ Y) $\cdots$ (*inverse* ‘ X)
then obtain x y **where** $x \in X$ **and** $y \in Y$ **and** $z = inverse\ y \cdot inverse\ x$
using *smul.cases image-iff* **by** *blast*
then show $z \in$ *inverse* ‘ $(X \cdots Y)$ **using** *inverse-composition-commute assms*

```
smul.simps
      by (smt (verit) image-iff in-mono invertible)
  qed
qed

lemma smul-singleton-eq-contains-nat-powers:
  fixes n :: nat
  assumes X ⊆ G and g ∈ G and X ··· {g} = X
  shows X ··· {g ^ n} = X
proof(induction n)
  case 0
  then show ?case using power-def assms by auto
next
  case (Suc n)
  assume hXn: X ··· {g ^ n} = X
  moreover have X ··· {g ^ Suc n} = (X ··· {g ^ n}) ··· {g}
  proof
    show X ··· {g ^ Suc n} ⊆ (X ··· {g ^ n}) ··· {g}
    proof
      fix z assume z ∈ X ··· {g ^ Suc n}
      then obtain x where z = x · (g ^ Suc n) and hx: x ∈ X  using smul.simps
by auto
      then have z = (x · g ^ n) · g using assms associative by (simp add: in-mono
power-mem-carrier)
      then show z ∈ (X ··· {g ^ n}) ··· {g} using hx assms
        by (simp add: power-mem-carrier smul.smulI subsetD)
    qed
  next
    show (X ··· {g ^ n}) ··· {g} ⊆ X ··· {g ^ Suc n}
    proof
      fix z assume z ∈ (X ··· {g ^ n}) ··· {g}
      then obtain x where z = (x · g ^ n) · g and hx: x ∈ X using smul.simps
by auto
      then have z = x · g ^ Suc n
      using power-def associative power-mem-carrier assms by (simp add: in-mono)
      then show z ∈ X ··· {g ^ Suc n} using hx assms
        by (simp add: power-mem-carrier smul.smulI subsetD)
    qed
  qed
  ultimately show ?case using assms by simp
qed

lemma smul-singleton-eq-contains-inverse-nat-powers:
  fixes n :: nat
  assumes X ⊆ G and g ∈ G and X ··· {g} = X
  shows X ··· {(inverse g) ^ n} = X
proof−
  have (X ··· {g}) ··· {inverse g} = X
  proof
```

show $(X \cdots \{g\}) \cdots \{inverse\ g\} \subseteq X$
proof
fix z **assume** $z \in (X \cdots \{g\}) \cdots \{inverse\ g\}$
then obtain $y\ x$ **where** $y \in X \cdots \{g\}$ **and** $z = y \cdot inverse\ g$ **and** $x \in X$ **and** $y = x \cdot g$
using *assms smul.simps* **by** (*metis empty-iff insert-iff*)
then show $z \in X$ **using** *assms* **by** (*simp add*: *associative subset-eq*)
qed
next
show $X \subseteq (X \cdots \{g\}) \cdots \{inverse\ g\}$
proof
fix x **assume** *hx*: $x \in X$
then have $x = x \cdot g \cdot inverse\ g$ **using** *assms* **by** (*simp add*: *associative subset-iff*)
then show $x \in (X \cdots \{g\}) \cdots \{inverse\ g\}$ **using** *assms smul.simps hx* **by** *auto*
qed
qed
then have $X \cdots \{inverse\ g\} = X$ **using** *assms* **by** *auto*
then show *?thesis* **using** *assms* **by** (*simp add*: *smul-singleton-eq-contains-nat-powers*)
qed

lemma *smul-singleton-eq-contains-powers*:
fixes n :: *nat*
assumes $X \subseteq G$ **and** $g \in G$ **and** $X \cdots \{g\} = X$
shows $X \cdots (powers\ g) = X$ **using** *powers-eq-union-nat-powers smul-subset-Union2*

nat-powers-eq-Union smul-singleton-eq-contains-nat-powers
smul-singleton-eq-contains-inverse-nat-powers assms smul-subset-Un2 **by** *auto*

end

end

2 Generalized Cauchy–Davenport Theorem: main proof

theory *Generalized-Cauchy-Davenport-main-proof*
imports
Generalized-Cauchy-Davenport-preliminaries
begin

context *group*

begin

2.1 Definition of the counterexample pair relation in [4]

definition *devos-rel* **where**
$\textit{devos-rel} = (\lambda\ (A,\ B).\ \textit{card}(A \cdots B))\ {<}{*}\textit{mlex}{*}{>}\ (\textit{inv-image}\ (\{(n,\ m).\ n > m\}$
${<}{*}\textit{lex}{*}{>}$
$\textit{measure}\ (\lambda\ (A,\ B).\ \textit{card}\ A)))\ (\lambda\ (A,\ B).\ (\textit{card}\ A + \textit{card}\ B,\ (A,\ B)))$

2.2 Lemmas about the counterexample pair relation in [4]

lemma *devos-rel-iff*:
$((A,\ B),\ (C,\ D)) \in \textit{devos-rel} \longleftrightarrow \textit{card}(A \cdots B) < \textit{card}(C \cdots D) \lor$
$(\textit{card}(A \cdots B) = \textit{card}(C \cdots D) \land \textit{card}\ A + \textit{card}\ B > \textit{card}\ C + \textit{card}\ D) \lor$
$(\textit{card}(A \cdots B) = \textit{card}(C \cdots D) \land \textit{card}\ A + \textit{card}\ B = \textit{card}\ C + \textit{card}\ D \land \textit{card}$
$A < \textit{card}\ C)$
using *devos-rel-def mlex-iff*[*of - -* $\lambda\ (A,\ B).\ \textit{card}(A \cdots B)$] **by** *fastforce*

lemma *devos-rel-le-smul*:
$((A,\ B),\ (C,\ D)) \in \textit{devos-rel} \Longrightarrow \textit{card}(A \cdots B) \le \textit{card}(C \cdots D)$
using *devos-rel-iff* **by** *fastforce*

Lemma stating that the above relation due to DeVos is well-founded

lemma *devos-rel-wf* : *wf* (*Restr devos-rel*
$\{(A,\ B).\ \textit{finite}\ A \land A \neq \{\} \land A \subseteq G \land \textit{finite}\ B \land B \neq \{\} \land B \subseteq G\}$) (**is** *wf*
(*Restr devos-rel ?fin*))
proof−
define f **where** $f = (\lambda\ (A,\ B).\ \textit{card}(A{\cdots}B))$
define g **where** $g = (\lambda\ (A :: {'a}\ \textit{set},\ B :: {'a}\ \textit{set}).\ (\textit{card}\ A + \textit{card}\ B,\ (A,\ B)))$
define h **where** $h = (\lambda\ (A :: {'a}\ \textit{set},\ B :: {'a}\ \textit{set}).\ \textit{card}\ A + \textit{card}\ B)$
define s **where** $s = (\{(n :: \textit{nat},\ m :: \textit{nat}).\ n > m\}\ {<}{*}\textit{lex}{*}{>}\ \textit{measure}\ (\lambda\ (A :: {'a}$
$\textit{set},\ B :: {'a}\ \textit{set}).\ \textit{card}\ A))$
have *hle2f*: $\bigwedge x.\ x \in\ ?\textit{fin} \Longrightarrow h\ x \le 2 * f\ x$
proof−
fix x **assume** *hx*: $x \in\ ?\textit{fin}$
then obtain $A\ B$ **where** *hxAB*: $x = (A,\ B)$ **by** *blast*
then have $\textit{card}\ A \le \textit{card}\ (A \cdots B)$ **and** $\textit{card}\ B \le \textit{card}\ (A \cdots B)$
using *card-le-smul-left card-le-smul-right hx* **by** *auto*
then show $h\ x \le 2 * f\ x$ **using** *hxAB h-def f-def* **by** *force*
qed
have *wf* (*Restr* (*measure f*) *?fin*) **by** (*simp add*: *wf-Int1*)
moreover have $\bigwedge a.\ a \in \textit{range}\ f \Longrightarrow \textit{wf}\ (\textit{Restr}\ (\textit{Restr}\ (\textit{inv-image}\ s\ g)\ \{x.\ f\ x$
$= a\})\ ?\textit{fin})$
proof−
fix y **assume** $y \in \textit{range}\ f$
then show $\textit{wf}\ (\textit{Restr}\ (\textit{Restr}\ (\textit{inv-image}\ s\ g)\ \{x.\ f\ x = y\})\ ?\textit{fin})$
proof−
have $\textit{Restr}\ (\{x.\ f\ x = y\} \times \{x.\ f\ x = y\} \cap (\textit{inv-image}\ s\ g))\ ?\textit{fin} \subseteq$
$\textit{Restr}\ (((\lambda\ x.\ 2 * f\ x - h\ x)\ {<}{*}\textit{mlex}{*}{>}\ \textit{measure}\ (\lambda\ (A :: {'a}\ \textit{set},\ B :: {'a}\ \textit{set}).$
$\textit{card}\ A)) \cap$
$\{x.\ f\ x = y\} \times \{x.\ f\ x = y\})\ ?\textit{fin}$
proof

fix z **assume** *hz*: $z \in$ *Restr* $(\{x.\ f\ x = y\} \times \{x.\ f\ x = y\} \cap$ (*inv-image* $s\ g$)) *?fin*
then obtain $a\ b$ **where** *hzab*: $z = (a, b)$ **and** $f\ a = y$ **and** $f\ b = y$ **and**
$h\ a > h\ b \vee h\ a = h\ b \wedge (a, b) \in$ *measure* $(\lambda\ (A, B).$ *card* $A)$ **and**
$a \in$ *?fin* **and** $b \in$ *?fin*
using *s-def g-def h-def* **by** *force*
then obtain $2 * f\ a - h\ a < 2 * f\ b - h\ b \vee$
$2 * f\ a - h\ a = 2 * f\ b - h\ b \wedge (a, b) \in$ *measure* $(\lambda\ (A, B).$ *card* $A)$
using *hle2f* **by** (*smt* (*verit*) *diff-less-mono2 le-antisym nat-less-le*)
then show $z \in$ *Restr* $(((\lambda\ x.\ 2 * f\ x - h\ x)$ <*mlex*> *measure* $(\lambda\ (A, B).$ *card* $A)) \cap$
$\{x.\ f\ x = y\} \times \{x.\ f\ x = y\})$ *?fin* **using** *hzab hz* **by** (*simp add*: *mlex-iff*)
qed
moreover have *wf* $((\lambda\ x.\ 2 * f\ x - h\ x)$ <*mlex*> *measure* $(\lambda\ (A, B).$ *card* $A))$
by (*simp add*: *wf-mlex*)
ultimately show *?thesis* **by** (*simp add*: *Int-commute wf-Int1 wf-subset*)
qed
qed
moreover have *trans* (*?fin* $\times$ *?fin*) **using** *trans-def* **by** *fast*
ultimately have *wf* (*Restr* (*inv-image* (*less-than* <*lex*> s) $(\lambda\ c.\ (f\ c, g\ c))$) *?fin*)
using *wf-prod-lex-fibers-inter*[*of less-than f ?fin* $\times$ *?fin s g*] *measure-def*
by (*metis* (*no-types*, *lifting*) *inf-sup-aci*(*1*))
moreover have (*inv-image* (*less-than* <*lex*> s) $(\lambda\ c.\ (f\ c, g\ c))) =$ *devos-rel*
using *s-def f-def g-def devos-rel-def mlex-prod-def* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed

2.3 Definition of $p(G)$ in [4] with associated lemmas

definition p **where** $p =$ *Inf* (*card* ' $\{H.$ *subgroup* $H\ G\ (\cdot)\ \mathbf{1} \wedge$ *finite* $H \wedge H \neq \{\mathbf{1}\}\})$

lemma *powers-subgroup*:
assumes $g \in G$
shows *subgroup* (*powers* g) $G\ (\cdot)\ \mathbf{1}$
by (*simp add*: *assms powers-group powers-submonoid subgroup.intro*)

lemma *subgroup-finite-ge*:
assumes *subgroup* $H\ G\ (\cdot)\ \mathbf{1}$ **and** $H \neq \{\mathbf{1}\}$ **and** *finite* H
shows *card* $H \geq p$
using *p-def assms* **by** (*simp add*: *wellorder-Inf-le1*)

lemma *subgroup-infinite-or-ge*:
assumes *subgroup* $H\ G\ (\cdot)\ \mathbf{1}$ **and** $H \neq \{\mathbf{1}\}$
shows *infinite* $H \vee$ *card* $H \geq p$ **using** *subgroup-finite-ge assms* **by** *auto*

end

2.4 Proof of the Generalized Cauchy-Davenport for (non-abelian) groups

Generalized Cauchy-Davenport theorem for (non-abelian) groups due to Matt DeVos [4]

theorem (**in** *group*) *Generalized-Cauchy-Davenport*:
  assumes *hAne*: $A \neq \{\}$ **and** *hBne*: $B \neq \{\}$ **and** *hAG*: $A \subseteq G$ **and** *hBG*: $B \subseteq G$ **and**
  hAfin: *finite A* **and** *hBfin*: *finite B* **and**
  hsub: $\{H.\ \textit{subgroup-of-group}\ H\ G\ (\cdot)\ \mathbf{1} \wedge \textit{finite}\ H \wedge H \neq \{\mathbf{1}\}\} \neq \{\}$
  shows $\textit{card}\ (A \cdots B) \geq \textit{min}\ p\ (\textit{card}\ A + \textit{card}\ B - 1)$
proof(*rule ccontr*)
  assume *hcontr*: $\neg\ \textit{min}\ p\ (\textit{card}\ A + \textit{card}\ B - 1) \leq \textit{card}\ (A \cdots B)$
  let $?\textit{fin} = \{(A, B).\ \textit{finite}\ A \wedge A \neq \{\} \wedge A \subseteq G \wedge \textit{finite}\ B \wedge B \neq \{\} \wedge B \subseteq G\}$
  define *M* **where** $M = \{(A, B).\ \textit{card}\ (A \cdots B) < \textit{min}\ p\ (\textit{card}\ A + \textit{card}\ B - 1)\} \cap\ ?\textit{fin}$
  have *hmemM*: $(A, B) \in M$ **using** *assms hcontr M-def* **by** *auto*
  then obtain *X Y* **where** *hXYM*: $(X, Y) \in M$ **and** *hmin*: $\bigwedge Z.\ Z \in M \Longrightarrow (Z, (X, Y)) \notin \textit{Restr devos-rel}\ ?\textit{fin}$
    using *devos-rel-wf wfE-min* **by** (*smt* (*verit*, *del-insts*) *old.prod.exhaust*)
  have *hXG*: $X \subseteq G$ **and** *hYG*: $Y \subseteq G$ **and** *hXfin*: *finite X* **and** *hYfin*: *finite Y* **and**
    hXYlt: $\textit{card}\ (X \cdots Y) < \textit{min}\ p\ (\textit{card}\ X + \textit{card}\ Y - 1)$ **using** *hXYM M-def* **by** *auto*
  have *hXY*: $\textit{card}\ X \leq \textit{card}\ Y$
  proof(*rule ccontr*)
    assume *hcontr*: $\neg\ \textit{card}\ X \leq \textit{card}\ Y$
   have *hinvinj*: *inj-on inverse G* **using** *inj-onI invertible invertible-inverse-inverse* **by** *metis*
    let $?M = \textit{inverse}\ `\ X$
    let $?N = \textit{inverse}\ `\ Y$
   have $?N \cdots ?M = \textit{inverse}\ `\ (X \cdots Y)$ **using** *set-inverse-composition-commute hXYM M-def* **by** *auto*
    then have *hNM*: $\textit{card}\ (?N \cdots ?M) = \textit{card}\ (X \cdots Y)$
      using *hinvinj card-image subset-inj-on smul-subset-carrier* **by** *metis*
    moreover have *hM*: $\textit{card}\ ?M = \textit{card}\ X$
      using *hinvinj hXG hYG card-image subset-inj-on* **by** *metis*
    moreover have *hN*: $\textit{card}\ ?N = \textit{card}\ Y$
      using *hinvinj hYG card-image subset-inj-on* **by** *metis*
    moreover have *hNplusM*: $\textit{card}\ ?N + \textit{card}\ ?M = \textit{card}\ X + \textit{card}\ Y$ **using** *hM hN* **by** *auto*
    ultimately have $\textit{card}\ (?N \cdots ?M) < \textit{min}\ p\ (\textit{card}\ ?N + \textit{card}\ ?M - 1)$
      using *hXYM M-def* **by** *auto*
    then have $(?N, ?M) \in M$ **using** *M-def hXYM* **by** *blast*
   then have $((?N, ?M), (X, Y)) \notin \textit{devos-rel}$ **using** *hmin hXYM M-def* **by** *blast*
   then have $\neg\ \textit{card}\ Y < \textit{card}\ X$ **using** *hN hNM hNplusM devos-rel-iff* **by** *simp*
    then show *False* **using** *hcontr* **by** *linarith*
  qed
  have *hX2*: $2 \leq \textit{card}\ X$

proof(*rule ccontr*)
assume $\neg$ *2* $\leq$ *card X*
moreover have *card X* $>$ *0* **using** *hXYM M-def card-gt-0-iff* **by** *blast*
ultimately have *hX1*: *card X* $=$ *1* **by** *auto*
then obtain x **where** $X = \{x\}$ **and** $x \in G$ **using** *hXG* **by** (*metis card-1-singletonE insert-subset*)
then have *card* $(X \cdots Y) =$ *card* $X +$ *card* $Y -$ *1* **using** *card-smul-singleton-left-eq hYG hXYM M-def*
by (*simp add*: *Int-absorb2*)
then show *False* **using** *hXYlt* **by** *linarith*
qed
then obtain *a b* **where** *habX*: $\{a, b\} \subseteq X$ **and** *habne*: $a \neq b$ **by** (*metis card-2-iff obtain-subset-with-card-n*)
moreover have $b \in X \cdots \{$*inverse* $a \cdot b\}$ **by** (*smt* (*verit*) *habX composition-closed hXG insert-subset*
invertible invertible-inverse-closed invertible-right-inverse2 singletonI smul.smulI subsetD)
then obtain g **where** *hgG*: $g \in G$ **and** *hg1*: $g \neq$ **1** **and** *hXgne*: $(X \cdots \{g\}) \cap X \neq \{\}$
using *habne habX hXG* **by** (*metis composition-closed insert-disjoint*(*2*) *insert-subset invertible*
invertible-inverse-closed invertible-right-inverse2 mk-disjoint-insert right-unit)
have *hpsubX*: $(X \cdots \{g\}) \cap X \subset X$
proof(*rule ccontr*)
assume $\neg$ $(X \cdots \{g\}) \cap X \subset X$
then have *hXsub*: $X \subseteq X \cdots \{g\}$ **by** *auto*
then have *card* $X \cdots \{g\} =$ *card* X **using** *card-smul-sing-right-le hXYM M-def*
by (*metis Int-absorb2* ‹$g \in G$› *card.infinite card-smul-singleton-right-eq finite-Int hXG*)
moreover have *hXfin*: *finite X* **using** *hXYM M-def* **by** *auto*
ultimately have $X \cdots \{g\} = X$ **using** *hXsub*
by (*metis card-subset-eq finite.emptyI finite.insertI finite-smul*)
then have *hXpow*: $X \cdots$ (*powers g*) $= X$ **by** (*simp add*: *hXG hgG smul-singleton-eq-contains-powers*)
moreover have *hfinpowers*: *finite* (*powers g*)
proof(*rule ccontr*)
assume *infinite* (*powers g*)
then have *infinite X* **using** *hXG hX2 hXpow* **by** (*metis Int-absorb1 hXgne hXsub hgG*
infinite-smul-right invertible le-iff-inf powers-submonoid submonoid.subset)
then show *False* **using** *hXYM M-def* **by** *auto*
qed
ultimately have *card* (*powers g*) $\leq$ *card X* **using** *card-le-smul-right*
powers-submonoid submonoid.subset hXYM M-def habX hXG
by (*metis* (*no-types*, *lifting*) *hXfin hgG insert-subset invertible subsetD*)
then have $p \leq$ *card X*
by (*meson hfinpowers hg1 hgG le-trans powers-ne-one powers-subgroup subgroup-infinite-or-ge*)
then have $p \leq$ *card* $(X \cdots Y)$ **using** *card-le-smul-left hXYM M-def*
by (*metis* (*full-types*) ‹$b \in$ *smul* X $\{$*inverse* $a \cdot b\}$› *bot-nat-0.extremum-uniqueI*

card.infinite
card-0-eq card-le-smul-right empty-iff hXY hXfin hYG le-trans smul.cases)
then show *False* **using** *hXYlt* **by** *auto*
qed
let *?X1* = $(X \cdots \{g\}) \cap X$
let *?X2* = $(X \cdots \{g\}) \cup X$
let *?Y1* = $(\{inverse\ g\} \cdots Y) \cup Y$
let *?Y2* = $(\{inverse\ g\} \cdots Y) \cap Y$
have *hY1G*: *?Y1* $\subseteq G$ **and** *hY1fin*: *finite ?Y1* **and** *hX2G*: *?X2* $\subseteq G$ **and** *hX2fin*: *finite ?X2*
using *hYfin hYG hXG finite-smul hXfin smul-subset-carrier* **by** *auto*
have *hXY1*: *?X1* $\cdots$ *?Y1* $\subseteq X \cdots Y$
proof
fix z **assume** $z \in$ *?X1* $\cdots$ *?Y1*
then obtain x y **where** *hz*: $z = x \cdot y$ **and** *hx*: $x \in$ *?X1* **and** *hy*: $y \in$ *?Y1* **by** (*meson smul.cases*)
show $z \in X \cdots Y$
proof(*cases* $y \in Y$)
case *True*
then show *?thesis* **using** *hz hx smulI hXG hYG* **by** *auto*
next
case *False*
then obtain w **where** $y = inverse\ g \cdot\ w$ **and** $w \in Y$ **using** *hy smul.cases* **by** (*metis UnE singletonD*)
moreover obtain v **where** $x = v \cdot g$ **and** $v \in X$ **using** *hx smul.cases* **by** *blast*
ultimately show *?thesis* **using** *hz hXG hYG hgG associative invertible-right-inverse2*
by (*simp add*: *smul.smulI subsetD*)
qed
qed
have *hXY2*: *?X2* $\cdots$ *?Y2* $\subseteq X \cdots Y$
proof
fix z **assume** $z \in$ *?X2* $\cdots$ *?Y2*
then obtain x y **where** *hz*: $z = x \cdot y$ **and** *hx*: $x \in$ *?X2* **and** *hy*: $y \in$ *?Y2* **by** (*meson smul.cases*)
show $z \in X \cdots Y$
proof(*cases* $x \in X$)
case *True*
then show *?thesis* **using** *hz hy smulI hXG hYG* **by** *auto*
next
case *False*
then obtain v **where** $x = v \cdot g$ **and** $v \in X$ **using** *hx smul.cases* **by** (*metis UnE singletonD*)
moreover obtain w **where** $y = inverse\ g \cdot w$ **and** $w \in Y$ **using** *hy smul.cases* **by** *blast*
ultimately show *?thesis* **using** *hz hXG hYG hgG associative invertible-right-inverse2*
by (*simp add*: *smul.smulI subsetD*)
qed
qed

have *hY2ne*: *?Y2* $\neq$ {}
proof
assume *hY2*: *?Y2* = {}
have *card X* + *card Y* $\leq$ *card Y* + *card Y* **by** (*simp add*: *hXY*)
also have ... = *card ?Y1* **using** *card-Un-disjoint hYfin hYG hgG finite-smul inf.orderE invertible*
by (*metis hY2 card-smul-singleton-left-eq finite.emptyI finite.insertI invertible-inverse-closed*)
also have ... $\leq$ *card* (*?X1* $\cdots$ *?Y1*) **using** *card-le-smul-right*[*OF* - - - *hY1fin hY1G*]
hXgne hXG Int-assoc Int-commute ex-in-conv finite-Int hXfin smul.simps smul-D(*2*)
smul-Int-carrier unit-closed **by** *auto*
also have ... $\leq$ *card* (*X* $\cdots$ *Y*) **using** *hXY1 finite-smul card-mono* **by** (*metis hXfin hYfin*)
finally show *False* **using** *hXYlt* **by** *linarith*
qed
have *hXadd*: *card ?X1* + *card ?X2* = *2* $*$ *card X*
using *card-smul-singleton-right-eq hgG hXfin hXG card-Un-Int*
by (*metis Un-Int-eq*(*3*) *add.commute finite.emptyI finite.insertI finite-smul mult-2 subset-Un-eq*)
have *hYadd*: *card ?Y1* + *card ?Y2* = *2* $*$ *card Y*
using *card-smul-singleton-left-eq hgG hYfin hYG card-Un-Int finite-smul*
by (*metis Int-lower1 Un-Int-eq*(*3*) *card-0-eq card-Un-le card-seteq finite.emptyI finite.insertI*
hY2ne le-add-same-cancel1 mult-2 subset-Un-eq)
show *False*
proof (*cases card ?X2* + *card ?Y2* > *card X* + *card Y*)
case *h*: *True*
have *hXY2le*: *card* (*?X2* $\cdots$ *?Y2*) $\leq$ *card* (*X* $\cdots$ *Y*) **using** *hXY2 finite-smul card-mono* **by** (*metis hXfin hYfin*)
also have ... < *min p* (*card X* + *card Y* − *1*) **using** *hXYlt* **by** *auto*
also have ... $\leq$ *min p* (*card ?X2* + *card ?Y2* − *1*) **using** *h* **by** *simp*
finally have *hXY1M*: (*?X2*, *?Y2*) $\in$ *M* **using** *M-def hY2ne hX2fin hX2G hXYM* **by** *auto*
moreover have ((*?X2*, *?Y2*), (*X*, *Y*)) $\in$ *Restr devos-rel ?fin* **using** *hXYM M-def hXY1M h hXY2le*
devos-rel-iff **by** *auto*
ultimately show *False* **using** *hmin* **by** *blast*
next
case *False*
then have *h*: *card ?X1* + *card ?Y1* $\geq$ *card X* + *card Y* **using** *hXadd hYadd* **by** *linarith*
have *hX1lt*: *card ?X1* < *card X* **using** *hXfin* **by** (*simp add*: *hpsubX psubset-card-mono*)
have *hXY1le*: *card* (*?X1* $\cdots$ *?Y1*) $\leq$ *card* (*X* $\cdots$ *Y*) **using** *hXY1 finite-smul card-mono hYfin hXfin* **by** *metis*
also have ... < *min p* (*card X* + *card Y* − *1*) **using** *hXYlt* **by** *auto*
also have ... $\leq$ *min p* (*card ?X1* + *card ?Y1* − *1*) **using** *h* **by** *simp*

finally have *hXY1M*: (*?X1*, *?Y1*) ∈ *M* **using** *M-def hXgne hY1fin hY1G hXYM* **by** *auto*
moreover have ((*?X1*, *?Y1*), (*X*, *Y*)) ∈ *Restr devos-rel ?fin* **using** *hXYM M-def hXY1M h hXY1le*
devos-rel-iff hX1lt hXY1le h **by** *force*
ultimately show *?thesis* **using** *hmin* **by** *blast*
qed
qed

end

References

[1] M. Bakys and A. Koutsoukou-Argyraki. Kneser's theorem and the cauchydavenport theorem. *Archive of Formal Proofs*, November 2022. https://isa-afp.org/entries/Kneser_Cauchy_Davenport.html, Formal proof development.

[2] A. L. B. Cauchy. Recherches sur les nombres. *J. École Polytech.*, 9:99–116, 1813.

[3] H. Davenport. On the Addition of Residue Classes. *Journal of the London Mathematical Society*, s1-10(1):30–32, 01 1935.

[4] M. DeVos. On a generalization of the cauchy-davenport theorem. *Integers*, 16:A7, 2016.