

A Generalization of the Cauchy–Davenport Theorem

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

February 6, 2026

Abstract

The Cauchy–Davenport theorem is a fundamental result 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 Preliminaries on well-orderings, groups, and sumsets	3
1.1 Well-ordering lemmas	3
1.2 Pointwise set multiplication in a monoid: definition and key lemmas	4
1.3 Exponentiation in a monoid: definitions and lemmas	6
1.4 Definition of the order of an element in a monoid	11
1.5 Sumset scalar multiplication cardinality lemmas	11
1.6 Pointwise set multiplication in a group: auxiliary lemmas	14
1.7 <i>ecard</i> – extended definition of cardinality of a set	16
2 Generalized Cauchy–Davenport theorem: main proof	16
2.1 The counterexample pair relation in [4]	16
2.2 $p(G)$ – the order of the smallest nontrivial finite subgroup of a group: definition and lemmas	18
2.3 Proof of the generalized Cauchy–Davenport theorem for (non-abelian) groups	18

1 Preliminaries on well-orderings, groups, and sum-sets

theory *Generalized-Cauchy-Davenport-preliminaries*

imports

Complex-Main

Jacobson-Basic-Algebra.Group-Theory

HOL-Library.Extended-Nat

begin

1.1 Well-ordering lemmas

lemma *wf-prod-lex-fibers-inter*:

fixes $r :: ('a \times 'a)$ set **and** $s :: ('b \times 'b)$ set **and** $f :: 'c \Rightarrow 'a$ **and** $g :: 'c \Rightarrow 'b$

and

$t :: ('c \times 'c)$ set

assumes $h1: wf ((inv-image\ r\ f) \cap t)$ **and**

$h2: \bigwedge a. a \in range\ f \implies wf (\{x. f\ x = a\} \times \{x. f\ x = a\} \cap (inv-image\ s\ g)) \cap t)$ **and**

$h3: trans\ t$

shows $wf ((inv-image\ (r\ <*\!lex*\!>\ s)\ (\lambda\ c. (f\ c, g\ c))) \cap t)$

proof –

have $h4: (\bigcup a \in range\ f. (\{x. f\ x = a\} \times \{x. f\ x = a\} \cap (inv-image\ s\ g)) \cap t)$

=

$(\bigcup a \in range\ f. (\{x. f\ x = a\} \times \{x. f\ x = a\} \cap (inv-image\ s\ g))) \cap t$ **by** *blast*

have $(inv-image\ (r\ <*\!lex*\!>\ s)\ (\lambda\ c. (f\ c, g\ c))) \cap t = (inv-image\ r\ f \cap t) \cup$

$(\bigcup a \in range\ f. \{x. f\ x = a\} \times \{x. f\ x = a\} \cap (inv-image\ s\ g) \cap t)$

proof

show $inv-image\ (r\ <*\!lex*\!>\ s)\ (\lambda\ c. (f\ c, g\ c)) \cap t$

$\subseteq inv-image\ r\ f \cap t \cup (\bigcup a \in range\ f. \{x. f\ x = a\} \times \{x. f\ x = a\} \cap inv-image\ s\ g \cap t)$

proof

fix y **assume** $hy: y \in inv-image\ (r\ <*\!lex*\!>\ s)\ (\lambda\ c. (f\ c, g\ c)) \cap t$

then obtain $a\ b$ **where** $hx: y = (a, b)$ **and** $(f\ a, f\ b) \in r \vee (f\ a = f\ b \wedge (g\ a, g\ b) \in s)$

using *in-inv-image in-lex-prod Int-iff SigmaE UNIV-Times-UNIV inf-top-right* **by** *(smt (z3))*

then show $y \in inv-image\ r\ f \cap t \cup (\bigcup a \in range\ f. \{x. f\ x = a\} \times \{x. f\ x = a\} \cap inv-image\ s\ g \cap t)$

using hy **by** *auto*

qed

show $inv-image\ r\ f \cap t \cup (\bigcup a \in range\ f. \{x. f\ x = a\} \times \{x. f\ x = a\} \cap inv-image\ s\ g \cap t) \subseteq$

$inv-image\ (r\ <*\!lex*\!>\ s)\ (\lambda\ c. (f\ c, g\ c)) \cap t$ **using** *Int-iff SUP-le-iff SigmaD1 SigmaD2*

in-inv-image in-lex-prod mem-Collect-eq subrelI **by** *force*

qed

moreover have $((inv-image\ r\ f) \cap t) \ O$

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

lemma *wf-prod-lex-fibers*:

fixes $r :: ('a \times 'a) \text{ set}$ **and** $s :: ('b \times 'b) \text{ set}$ **and** $f :: 'c \Rightarrow 'a$ **and** $g :: 'c \Rightarrow 'b$
assumes *h1: wf (inv-image r f)* **and**
h2: $\bigwedge a. a \in \text{range } f \implies wf (\{x. f x = a\} \times \{x. f x = a\} \cap (\text{inv-image } s g))$
shows $wf (\text{inv-image } (r <*\text{lex}*> s) (\lambda c. (f c, g c)))$
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 a monoid: definition and key lemmas

inductive-set $smul :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ **for** $A B$

where

$smul[\text{intro}]: \llbracket a \in A; a \in M; b \in B; b \in M \rrbracket \implies a \cdot b \in smul A B$

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

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*: $\llbracket A' \subseteq A; B' \subseteq B \rrbracket \implies smul A' B' \subseteq smul A B$
by (*auto simp: smul-eq*)

lemma *smul-insert1*: *NO-MATCH* $\{x\} A \implies smul (\text{insert } x A) B = smul \{x\} B \cup smul A B$

by (*auto simp: smul-eq*)

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

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

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

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

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

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

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

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

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

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

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

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

lemma *smul-assoc*:
shows $\text{smul } (\text{smul } A B) C = \text{smul } A (\text{smul } B C)$
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 ∩ M) finite (B ∩ 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* ⇒ *nat* ⇒ '*a* (**infix** ^ 100)
where
power0: *power g 0 = 1*
| *power-suc*: *power g (Suc n) = power g n · g*

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

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

lemma *power-mult*:
assumes *g ∈ M*
shows *g ^ n · g ^ m = g ^ (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 ^ n · g ^ m = g ^ (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 ∈ M* **and** *invertible g*
shows *g ^ n · ((inverse g) ^ n) = 1*
proof(*induction n*)
case 0
then show ?*case* **using** *power-0* **by** *auto*
next
case (*Suc n*)
assume *hind*: *g ^ n · local.inverse g ^ n = 1*

then have $g \wedge \text{Suc } n \cdot \text{inverse } g \wedge \text{Suc } n = (g \cdot g \wedge n) \cdot (\text{inverse } g \wedge n \cdot \text{inverse } g)$
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 $((\text{inverse } g) \wedge n) \cdot g \wedge n = \mathbf{1}$ **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 $\text{inverse } (g \wedge n) = (\text{inverse } g) \wedge n$
using *assms inverse-equality* **by** (*simp add: inverse-mult-power mult-inverse-power*
power-mem-carrier)

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

definition *nat-powers* $:: 'a \Rightarrow 'a \text{ set}$ **where** $\text{nat-powers } g = ((\lambda n. g \wedge n) \text{ ` UNIV})$

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

definition *powers* $:: 'a \Rightarrow 'a \text{ set}$ **where** $\text{powers } g = ((\lambda n. g \text{ powi } n) \text{ ` UNIV})$

lemma *nat-powers-subset*:

$\text{nat-powers } g \subseteq \text{powers } g$

proof

fix x **assume** $x \in \text{nat-powers } g$

then obtain n **where** $x = g \wedge n$ **and** $\text{nat } n = n$ **using** *nat-powers-def* **by** *auto*

then show $x \in \text{powers } g$ **using** *powers-def power-int-def*

by (*metis UNIV-I image-iff of-nat-0-le-iff*)

qed

lemma *inverse-nat-powers-subset*:

$\text{nat-powers } (\text{inverse } g) \subseteq \text{powers } g$

proof

fix x **assume** $x \in \text{nat-powers } (\text{inverse } g)$

then obtain n **where** $x = (\text{inverse } g) \wedge n$ **using** *nat-powers-def* **by** *blast*

then show $x \in \text{powers } g$

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:  $\neg (- \text{int } n) \geq 0$  by auto
  then have  $x = g \text{ powi } (- \text{int } n)$  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  $g = \text{nat-powers } g \cup \text{nat-powers } (\text{inverse } g)$ 
proof
  show powers  $g \subseteq \text{nat-powers } g \cup \text{nat-powers } (\text{local.inverse } g)$ 
    using powers-def nat-powers-def power-int-def by auto
  next
  show  $\text{nat-powers } g \cup \text{nat-powers } (\text{inverse } g) \subseteq \text{powers } g$ 
    by (simp add: inverse-nat-powers-subset nat-powers-subset)
qed

lemma one-mem-nat-powers:  $1 \in \text{nat-powers } g$ 
  using rangeI power0 nat-powers-def by metis

lemma nat-powers-subset-carrier:
  assumes  $g \in M$ 
  shows  $\text{nat-powers } g \subseteq M$ 
  using nat-powers-def power-mem-carrier assms by auto

lemma nat-powers-mult-closed:
  assumes  $g \in M$ 
  shows  $\bigwedge x y. x \in \text{nat-powers } g \implies y \in \text{nat-powers } g \implies x \cdot y \in \text{nat-powers } g$ 
  using nat-powers-def power-mult assms by auto

lemma nat-powers-inv-mult:
  assumes  $g \in M$  and invertible  $g$ 
  shows  $\bigwedge x y. x \in \text{nat-powers } g \implies y \in \text{nat-powers } (\text{inverse } g) \implies x \cdot y \in \text{powers } g$ 
proof -
  fix  $x y$  assume  $x \in \text{nat-powers } g$  and  $y \in \text{nat-powers } (\text{inverse } g)$ 
  then obtain  $n m$  where  $hx: x = g \wedge n$  and  $hy: y = (\text{inverse } g) \wedge m$  using
  nat-powers-def by blast
  show  $x \cdot y \in \text{powers } g$ 
  proof (cases  $n \geq m$ )
    case True
    then obtain  $k$  where  $n = k + m$  using add.commute le-Suc-ex by blast
    then have  $g \wedge n \cdot (\text{inverse } g) \wedge m = g \wedge k$  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  $g^{\wedge n} \cdot (\text{inverse } g)^{\wedge m} = (\text{inverse } g)^{\wedge k}$  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  $\bigwedge x y. x \in \text{nat-powers } (\text{inverse } g) \implies y \in \text{nat-powers } g \implies x \cdot y \in$ 
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  $\bigwedge x y. x \in \text{powers } g \implies y \in \text{powers } g \implies x \cdot y \in \text{powers } g$ 
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 Group-Theory.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  $\bigwedge a b. a \in \text{powers } g \implies b \in \text{powers } g \implies a \cdot b \in \text{powers } g$ 
    using powers-mult-closed assms by auto
next
  show  $1 \in \text{powers } g$  using powers-eq-union-nat-powers one-mem-nat-powers by
  auto
qed

lemma powers-monoid:
  assumes  $g \in M$  and invertible  $g$ 
  shows Group-Theory.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  $u \in \text{nat-powers } g$ 
  shows monoid.invertible (powers  $g$ ) ( $\cdot$ )  $1$   $u$ 
  proof -
    obtain  $n$  where  $hu: u = g \wedge n$  using assms nat-powers-def by blast
    then have inverse  $u \in \text{powers } g$  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 in-
      verse-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  $u \in \text{nat-powers } (\text{inverse } g)$ 
  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-Theory.group (powers  $g$ ) ( $\cdot$ )  $1$ 
  proof -
    have  $\bigwedge u. u \in \text{powers } g \implies \text{monoid.invertible } (\text{powers } g) (\cdot) 1 u$  using assms
      mem-nat-inv-powers-invertible mem-nat-powers-invertible powers-eq-union-nat-powers
    by auto
    then show ?thesis using group-def Group-Theory.group-axioms-def assms pow-
      ers-monoid by metis
  qed

lemma nat-powers-ne-one:

```

```

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

end

```

```

context group

```

```

begin

```

```

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)

```

```

end

```

```

context monoid

```

```

begin

```

1.4 Definition of the order of an element in a monoid

```

definition order

```

```

  where  $\text{order } g = (\text{if } (\exists n. n > 0 \wedge g \wedge n = \mathbf{1}) \text{ then } \text{Min } \{n. g \wedge n = \mathbf{1} \wedge n > 0\} \text{ else } \infty)$ 

```

```

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

```

```

end

```

1.5 Sumset scalar multiplication cardinality lemmas

```

context group

```

```

begin

```

```

lemma card-smul-singleton-right-eq:

```

```

  assumes finite  $A$  shows  $\text{card } (\text{smul } A \{a\}) = (\text{if } a \in G \text{ then } \text{card } (A \cap G) \text{ else } 0)$ 

```

```

proof (cases  $a \in G$ )

```

```

  case True

```

then have $smul\ A\ \{a\} = (\lambda x. x \cdot a) \text{ ` } (A \cap G)$
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 ∈ G*)

case *True*

then have $smul\ \{a\}\ A = (\lambda x. a \cdot x) \text{ ` } (A \cap G)$

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** $card\ (smul\ A\ \{a\}) \leq card\ A$

by (*simp add: assms card-mono card-smul-singleton-right-eq*)

lemma *card-smul-sing-left-le:*

assumes *finite A* **shows** $card\ (smul\ \{a\}\ A) \leq card\ A$

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 $card\ B \leq card\ (smul\ A\ B)$

proof –

have $B \subseteq (\lambda x. (inverse\ a) \cdot x) \text{ ` } smul\ A\ B$

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 $card\ A \leq card\ (smul\ A\ B)$

proof –

have $A \subseteq (\lambda x. x \cdot (inverse\ b)) \text{ ` } smul\ A\ B$

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 $A \cap G \neq \{\}$ **and** *infinite* ($B \cap G$)
shows *infinite* ($A \cdots B$)
proof
assume *hfin: finite* (*smul* $A B$)
obtain *a* **where** *ha: a* $\in A \cap G$ **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 $B \cap G \subseteq (\lambda x. \text{inverse } a \cdot x) \text{ ` } \text{smul } \{a\} B$
proof
fix *b* **assume** *hb: b* $\in B \cap G$
then have $b = \text{inverse } a \cdot (a \cdot b)$ **using** *associative ha* **by** (*simp add: invertible-left-inverse2*)
then show $b \in (\lambda x. \text{inverse } a \cdot x) \text{ ` } \text{smul } \{a\} B$ **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 $A \cap G \subseteq (\lambda x. x \cdot \text{inverse } b) \text{ ` } \text{smul } A \{b\}$
proof
fix *a* **assume** *ha: a* $\in A \cap G$
then have $a = (a \cdot b) \cdot \text{inverse } b$ **using** *associative hb*
by (*metis IntD2 invertible invertible-inverse-closed invertible-right-inverse right-unit*)
then show $a \in (\lambda x. x \cdot \text{inverse } b) \text{ ` } \text{smul } A \{b\}$ **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 a group: auxiliary lemmas

lemma *set-inverse-composition-commute*:

assumes $X \subseteq G$ and $Y \subseteq G$

shows $\text{inverse } '(X \cdots Y) = (\text{inverse } ' Y) \cdots (\text{inverse } ' X)$

proof

show $\text{inverse } '(X \cdots Y) \subseteq (\text{inverse } ' Y) \cdots (\text{inverse } ' X)$

proof

fix z assume $z \in \text{inverse } '(X \cdots Y)$

then obtain $x y$ where $z = \text{inverse } (x \cdot y)$ and $x \in X$ and $y \in Y$

by (*smt (verit) image-iff smul.cases*)

then show $z \in (\text{inverse } ' Y) \cdots (\text{inverse } ' X)$

using *inverse-composition-commute assms*

by (*smt (verit) image-eqI in-mono inverse-equality invertible invertibleE smul.simps*)

qed

show $(\text{inverse } ' Y) \cdots (\text{inverse } ' X) \subseteq \text{inverse } '(X \cdots Y)$

proof

fix z assume $z \in (\text{inverse } ' Y) \cdots (\text{inverse } ' X)$

then obtain $x y$ where $x \in X$ and $y \in Y$ and $z = \text{inverse } y \cdot \text{inverse } x$

using *smul.cases image-iff* by *blast*

then show $z \in \text{inverse } '(X \cdots Y)$ 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 $n :: \text{nat}$

assumes $X \subseteq G$ and $g \in G$ and $X \cdots \{g\} = X$

shows $X \cdots \{g \wedge n\} = X$

proof(*induction n*)

case 0

then show *?case* using *power-def assms* by *auto*

next

case (*Suc n*)

assume *hXn*: $X \cdots \{g \wedge n\} = X$

moreover have $X \cdots \{g \wedge \text{Suc } n\} = (X \cdots \{g \wedge n\}) \cdots \{g\}$

proof

show $X \cdots \{g \wedge \text{Suc } n\} \subseteq (X \cdots \{g \wedge n\}) \cdots \{g\}$

proof

fix z assume $z \in X \cdots \{g \wedge \text{Suc } n\}$

then obtain x where $z = x \cdot (g \wedge \text{Suc } n)$ and *hx*: $x \in X$ using *smul.simps*

by *auto*

then have $z = (x \cdot g \wedge n) \cdot g$ using *assms associative* by (*simp add: in-mono power-mem-carrier*)

then show $z \in (X \cdots \{g \wedge n\}) \cdots \{g\}$ using *hx assms*

by (*simp add: power-mem-carrier smul.smulI subsetD*)

qed

next

```

show  $(X \cdots \{g \wedge n\}) \cdots \{g\} \subseteq X \cdots \{g \wedge \text{Suc } n\}$ 
proof
  fix  $z$  assume  $z \in (X \cdots \{g \wedge n\}) \cdots \{g\}$ 
  then obtain  $x$  where  $z = (x \cdot g \wedge n) \cdot g$  and  $hx: x \in X$  using smul.simps
by auto
  then have  $z = x \cdot g \wedge \text{Suc } n$ 
  using power-def associative power-mem-carrier assms by (simp add: in-mono)
  then show  $z \in X \cdots \{g \wedge \text{Suc } n\}$  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  $n :: \text{nat}$ 
  assumes  $X \subseteq G$  and  $g \in G$  and  $X \cdots \{g\} = X$ 
  shows  $X \cdots \{(inverse\ g) \wedge n\} = X$ 
proof -
  have  $(X \cdots \{g\}) \cdots \{inverse\ g\} = X$ 
  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  $n :: \text{nat}$ 
  assumes  $X \subseteq G$  and  $g \in G$  and  $X \cdots \{g\} = X$ 
  shows  $X \cdots (\text{powers } g) = X$  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

1.7 *ecard* – extended definition of cardinality of a set

ecard – definition of a cardinality of a set taking values in *enat* – extended natural numbers, defined to be ∞ for infinite sets

definition *ecard* **where** $ecard\ A = (if\ finite\ A\ then\ card\ A\ else\ \infty)$

lemma *ecard-eq-card-finite*:

assumes *finite* *A*

shows $ecard\ A = card\ A$

using *assms ecard-def* **by** *metis*

context *monoid*

begin

orderOf – abbreviation for the order of a monoid

abbreviation *orderOf* **where** $orderOf == ecard$

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 The counterexample pair relation in [4]

definition *devos-rel* **where**

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

$measure\ (\lambda\ (A,\ B).\ card\ A)))\ (\lambda\ (A,\ B).\ (card\ A + card\ B,\ (A,\ B)))$

lemma *devos-rel-iff*:

$((A,\ B),\ (C,\ D)) \in devos-rel \iff card(A \cdots B) < card(C \cdots D) \vee$

$(card(A \cdots B) = card(C \cdots D) \wedge card\ A + card\ B > card\ C + card\ D) \vee$

$(\text{card}(A \cdots B) = \text{card}(C \cdots D) \wedge \text{card } A + \text{card } B = \text{card } C + \text{card } D \wedge \text{card } A < \text{card } C)$

using *devos-rel-def mlex-iff*[of - - $\lambda (A, B). \text{card}(A \cdots B)$] **by** *fastforce*

lemma *devos-rel-le-smul*:

$((A, B), (C, D)) \in \text{devos-rel} \implies \text{card}(A \cdots B) \leq \text{card}(C \cdots D)$

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*

$\{(A, B). \text{finite } A \wedge A \neq \{\} \wedge A \subseteq G \wedge \text{finite } B \wedge B \neq \{\} \wedge B \subseteq G\}$) (**is** *wf (Restr devos-rel ?fin)*)

proof –

define *f* **where** $f = (\lambda (A, B). \text{card}(A \cdots B))$

define *g* **where** $g = (\lambda (A :: 'a \text{ set}, B :: 'a \text{ set}). (\text{card } A + \text{card } B, (A, B)))$

define *h* **where** $h = (\lambda (A :: 'a \text{ set}, B :: 'a \text{ set}). \text{card } A + \text{card } B)$

define *s* **where** $s = (\{(n :: \text{nat}, m :: \text{nat}). n > m\} <*\text{lex}*\text{> measure } (\lambda (A :: 'a \text{ set}, B :: 'a \text{ set}). \text{card } A))$

have *hle2f*: $\bigwedge x. x \in ?\text{fin} \implies h \ x \leq 2 * f \ x$

proof –

fix *x* **assume** *hx*: $x \in ?\text{fin}$

then obtain *A B* **where** *hxAB*: $x = (A, B)$ **by** *blast*

then have $\text{card } A \leq \text{card } (A \cdots B)$ **and** $\text{card } B \leq \text{card } (A \cdots B)$

using *card-le-smul-left card-le-smul-right hx* **by** *auto*

then show $h \ x \leq 2 * f \ x$ **using** *hxAB h-def f-def* **by** *force*

qed

have *wf (Restr (measure f) ?fin)* **by** (*simp add: wf-Int1*)

moreover have $\bigwedge a. a \in \text{range } f \implies \text{wf } (\text{Restr } (\text{Restr } (\text{inv-image } s \ g) \ \{x. f \ x = a\}) \ ?\text{fin})$

proof –

fix *y* **assume** $y \in \text{range } f$

then show *wf (Restr (Restr (inv-image s g) {x. f x = y}) ?fin)*

proof –

have $\text{Restr } (\{x. f \ x = y\} \times \{x. f \ x = y\} \cap (\text{inv-image } s \ g)) \ ?\text{fin} \subseteq$

$\text{Restr } (((\lambda x. 2 * f \ x - h \ x) <*\text{mlex}*\text{> measure } (\lambda (A :: 'a \text{ set}, B :: 'a \text{ set}). \text{card } A)) \cap$

$\{x. f \ x = y\} \times \{x. f \ x = y\}) \ ?\text{fin}$

proof

fix *z* **assume** *hz*: $z \in \text{Restr } (\{x. f \ x = y\} \times \{x. f \ x = y\} \cap (\text{inv-image } s \ g))$

then obtain *a b* **where** *hzab*: $z = (a, b)$ **and** $f \ a = y$ **and** $f \ b = y$ **and**

$h \ a > h \ b \vee h \ a = h \ b \wedge (a, b) \in \text{measure } (\lambda (A, B). \text{card } A)$ **and**

$a \in ?\text{fin}$ **and** $b \in ?\text{fin}$

using *s-def g-def h-def* **by** *force*

then obtain $2 * f \ a - h \ a < 2 * f \ b - h \ b \vee$

$2 * f \ a - h \ a = 2 * f \ b - h \ b \wedge (a, b) \in \text{measure } (\lambda (A, B). \text{card } A)$

using *hle2f* **by** (*smt (verit) diff-less-mono2 le-antisym nat-less-le*)

then show $z \in \text{Restr } (((\lambda x. 2 * f \ x - h \ x) <*\text{mlex}*\text{> measure } (\lambda (A, B). \text{card } A)) \cap$

$\{x. f x = y\} \times \{x. f x = y\}$ *?fin* **using** *hzab hz* **by** (*simp add: mlex-iff*)
qed
moreover have *wf ((λ x. 2 * f x - h x) <*mlex*> measure (λ (A, B). card A))*
by (*simp add: wf-mlex*)
ultimately show *?thesis* **by** (*simp add: Int-commute wf-Int1 wf-subset*)
qed
moreover have *trans (?fin × ?fin)* **using** *trans-def* **by** *fast*
ultimately have *wf (Restr (inv-image (less-than <*lex*> s) (λ c. (f c, g c))) ?fin)*
using *wf-prod-lex-fibers-inter[of less-than f ?fin × ?fin s g] measure-def*
by (*metis (no-types, lifting) inf-sup-aci(1)*)
moreover have (*inv-image (less-than <*lex*> s) (λ 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

2.2 $p(G)$ – the order of the smallest nontrivial finite subgroup of a group: definition and lemmas

$p(G)$ – the size of the smallest nontrivial finite subgroup of G , set to ∞ if none exist

definition $p :: \text{enat}$ **where** $p = \text{Inf} (\text{orderOf} \{ H. \text{subgroup } H \ G \ (\cdot) \ \mathbf{1} \wedge H \neq \{ \mathbf{1} \} \})$

lemma *subgroup-finite-ge:*

assumes *subgroup H G (·) 1 and H ≠ {1} and finite H*
shows $\text{card } H \geq p$
using *assms p-def wellorder-Inf-le1 ecard-eq-card-finite*
by (*metis (mono-tags, lifting) image-eqI mem-Collect-eq*)

lemma *subgroup-infinite-or-card-ge:*

assumes *subgroup H G (·) 1 and H ≠ {1}*
shows $\text{infinite } H \vee \text{card } H \geq p$ **using** *subgroup-finite-ge assms* **by** *auto*

end

2.3 Proof of the generalized Cauchy–Davenport theorem 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 ≠ {} and hBne: B ≠ {} and hAG: A ⊆ G and hBG: B ⊆ G and*
hAfn: finite A and hBfn: finite B
shows $\text{card } (A \cdots B) \geq \min p (\text{card } A + \text{card } B - 1)$

proof(*rule ccontr*)

We will prove the theorem by contradiction

assume $hcontr: \neg \min p (\text{card } A + \text{card } B - 1) \leq \text{card } (A \cdots B)$
let $?fin = \{(A, B). \text{finite } A \wedge A \neq \{\} \wedge A \subseteq G \wedge \text{finite } B \wedge B \neq \{\} \wedge B \subseteq G\}$
define M **where** $M = \{(A, B). \text{card } (A \cdots B) < \min p (\text{card } A + \text{card } B - 1)\} \cap ?fin$
have $hmemM: (A, B) \in M$ **using** *assms hcontr M-def not-le* **by** *blast*

Firstly, extract sets X and Y , which are minimal counterexamples of the DeVos relation defined above

then obtain $X Y$ **where** $hXYM: (X, Y) \in M$ **and** $hmin: \bigwedge Z. Z \in M \implies (Z, (X, Y)) \notin \text{Restr devos-rel } ?fin$
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** $hXfin: \text{finite } X$ **and** $hYfin: \text{finite } Y$
and
 $hXYlt: \text{card } (X \cdots Y) < \min p (\text{card } X + \text{card } Y - 1)$ **using** $hXYM$ *M-def*
by *auto*
have $hXY: \text{card } X \leq \text{card } Y$
proof(*rule ccontr*)
assume $hcontr: \neg \text{card } X \leq \text{card } Y$
have $hinvinj: \text{inj-on inverse } G$ **using** *inj-onI invertible invertible-inverse-inverse*
by *metis*
let $?M = \text{inverse } ' X$
let $?N = \text{inverse } ' Y$
have $?N \cdots ?M = \text{inverse } '(X \cdots Y)$ **using** *set-inverse-composition-commute*
 $hXYM$ *M-def* **by** *auto*
then have $hNM: \text{card } (?N \cdots ?M) = \text{card } (X \cdots Y)$
using $hinvinj$ *card-image inj-on-subset smul-subset-carrier* **by** *metis*
moreover have $hM: \text{card } ?M = \text{card } X$
using $hinvinj$ hXG hYG *card-image inj-on-subset* **by** *metis*
moreover have $hN: \text{card } ?N = \text{card } Y$
using $hinvinj$ hYG *card-image inj-on-subset* **by** *metis*
moreover have $hNplusM: \text{card } ?N + \text{card } ?M = \text{card } X + \text{card } Y$ **using** hM
 hN **by** *auto*
ultimately have $\text{card } (?N \cdots ?M) < \min p (\text{card } ?N + \text{card } ?M - 1)$
using $hXYM$ *M-def hXYlt* **by** *argo*
then have $(?N, ?M) \in M$ **using** *M-def hXYM* **by** *blast*
then have $((?N, ?M), (X, Y)) \notin \text{devos-rel}$ **using** $hmin$ $hXYM$ *M-def* **by** *blast*
then have $\neg \text{card } Y < \text{card } X$ **using** hN hNM $hNplusM$ *devos-rel-iff* **by** *simp*
then show *False* **using** $hcontr$ **by** *simp*
qed

Observe that X contains at least 2 elements, otherwise the proof is easy

have $hX2: 2 \leq \text{card } X$
proof(*rule ccontr*)
assume $\neg 2 \leq \text{card } X$
moreover have $\text{card } X > 0$ **using** $hXYM$ *M-def card-gt-0-iff* **by** *blast*
ultimately have $hX1: \text{card } X = 1$ **by** *auto*

then obtain x **where** $X = \{x\}$ **and** $x \in G$ **using** hXG **by** (*metis card-1-singletonE insert-subset*)
then have $\text{card } (X \cdots Y) = \text{card } X + \text{card } Y - 1$ **using** *card-smul-singleton-left-eq hYG hXYM M-def*
by (*simp add: Int-absorb2*)
then show False **using** $hXYlt$ **by** *simp*
qed
then obtain $a \ b$ **where** $\text{habX}: \{a, b\} \subseteq X$ **and** $\text{habne}: a \neq b$ **by** (*metis card-2-iff obtain-subset-with-card-n*)
moreover have $b \in X \cdots \{\text{inverse } a \cdot b\}$ **by** (*smt (verit) habX composition-closed hXG insert-subset*
invertible invertible-inverse-closed invertible-right-inverse2 singletonI smul.smulI subsetD)

From this, obtain an element $g \in G$ such that $Xg \cap X \neq \emptyset$

then obtain g **where** $\text{hgG}: g \in G$ **and** $\text{hg1}: g \neq 1$ **and** $\text{hXgne}: (X \cdots \{g\}) \cap X \neq \{\}$
using $\text{habne } \text{habX } hXG$ **by** (*metis composition-closed insert-disjoint(2) insert-subset invertible*
invertible-inverse-closed invertible-right-inverse2 mk-disjoint-insert right-unit)

Now we show that $Xg \cap X$ is strict subset of X

have $\text{hpsubX}: (X \cdots \{g\}) \cap X \subset X$
proof(*rule ccontr*)
assume $\neg (X \cdots \{g\}) \cap X \subset X$
then have $\text{hXsub}: X \subseteq X \cdots \{g\}$ **by** *auto*
then have $\text{card } X \cdots \{g\} = \text{card } X$ **using** *card-smul-sing-right-le hXYM M-def*

Int-absorb2 $\langle g \in G \rangle$ *card.infinite card-smul-singleton-right-eq finite-Int hXG*
by *metis*

moreover have $\text{hXfin}: \text{finite } X$ **using** $hXYM$ *M-def* **by** *auto*

ultimately have $X \cdots \{g\} = X$ **using** hXsub *card-subset-eq finite.emptyI finite.insertI*

finite-smul **by** *metis*

then have $\text{hXpow}: X \cdots (\text{powers } g) = X$ **by** (*simp add: hXG hgG smul-singleton-eq-contains-powers*)

moreover have $\text{hfinpowers}: \text{finite } (\text{powers } g)$

proof(*rule ccontr*)

assume *infinite* $(\text{powers } g)$

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 $\text{card } (\text{powers } g) \leq \text{card } X$ **using** *card-le-smul-right*

powers-submonoid submonoid.subset hXYM M-def habX hXG hXfin hgG insert-subset invertible

subsetD **by** (*metis (no-types, lifting)*)

then have $p \leq \text{card } X$

using $\text{hfinpowers } \text{hg1 } \text{hgG}$ *le-trans powers-ne-one powers-subgroup subgroup-infinite-or-card-ge*

```

    by (smt (verit) enat-ile enat-ord-simps(1))
  then have  $p \leq \text{card } (X \cdots Y)$  using card-le-smul-left hXYM M-def
    ⟨ $b \in \text{smul } X \{ \text{inverse } a \cdot b \}$ ⟩ bot-nat-0.extremum-uniqueI card.infinite
    card-0-eq card-le-smul-right empty-iff hXY hXfin hYG le-trans smul.cases
  by (smt (verit) enat-ile enat-ord-simps(1))
  then show False using hXYlt by auto
qed

```

Define auxiliary transformations of sets X and Y to reach a contradiction

```

let ?X1 =  $(X \cdots \{g\}) \cap X$ 
let ?X2 =  $(X \cdots \{g\}) \cup X$ 
let ?Y1 =  $(\{\text{inverse } g\} \cdots Y) \cup Y$ 
let ?Y2 =  $(\{\text{inverse } g\} \cdots Y) \cap Y$ 
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  $y = \text{inverse } g \cdot w$  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** $y = \text{inverse } g \cdot w$ **and** $w \in Y$ **using** $hy \text{ smul.cases}$
by *blast*
ultimately show $?thesis$ **using** $hz \text{ hXG hYG hgG associative invertible-right-inverse2}$
by (*simp add: smul.smulI subsetD*)
qed
qed
have $hY2ne: ?Y2 \neq \{\}$
proof
assume $hY2: ?Y2 = \{\}$
have $\text{card } X + \text{card } Y \leq \text{card } Y + \text{card } Y$ **by** (*simp add: hXY*)
also have $\dots = \text{card } ?Y1$ **using** $\text{card-Un-disjoint hYfin hYG hgG finite-smul}$
inf.orderE invertible
by (*metis hY2 card-smul-singleton-left-eq finite.emptyI finite.insertI invertible-inverse-closed*)
also have $\dots \leq \text{card } (?X1 \dots ?Y1)$ **using** $\text{card-le-smul-right}[OF \dots hY1fin \text{ hY1G}]$
 $hXgne \text{ hXG Int-assoc Int-commute ex-in-conv finite-Int hXfin smul.simps}$
 $\text{smul-D}(2)$
 $\text{smul-Int-carrier unit-closed}$ **by** *auto*
also have $\dots \leq \text{card } (X \dots Y)$ **using** $hXY1 \text{ finite-smul card-mono}$ **by** (*metis hXfin hYfin*)
finally show *False* **using** $hXYlt$ **by** *auto*
qed
have $hXadd: \text{card } ?X1 + \text{card } ?X2 = 2 * \text{card } X$
using $\text{card-smul-singleton-right-eq hgG hXfin hXG card-Un-Int}$
by (*metis Un-Int-eq(3) add commute finite.emptyI finite.insertI finite-smul mult-2 subset-Un-eq*)
have $hYadd: \text{card } ?Y1 + \text{card } ?Y2 = 2 * \text{card } Y$
using $\text{card-smul-singleton-left-eq hgG hYfin hYG card-Un-Int finite-smul}$
by (*metis Int-lower1 Un-Int-eq(3) card-0-eq card-Un-le card-seteq finite.emptyI finite.insertI*)
 $hY2ne \text{ le-add-same-cancel1 mult-2 subset-Un-eq}$

Split the contradiction proof into the cases based on whether $|?X2| + |?Y2| > |X| + |Y|$ holds

show *False*
proof (*cases card ?X2 + card ?Y2 > card X + card Y*)
case *hcase: True*
then have $h : \text{card } X + \text{card } Y - 1 \leq \text{card } ?X2 + \text{card } ?Y2 - 1$ **by** *simp*
have $hXY2le: \text{enat } (\text{card } (?X2 \dots ?Y2)) \leq \text{card } (X \dots Y)$
using $hXY2 \text{ finite-smul card-mono hXfin hYfin enat-ile}$ **by** (*metis enat-ord-simps(1)*)
moreover have $\dots < \min p (\text{card } X + \text{card } Y - 1)$ **using** $hXYlt$ **by** *auto*
moreover have $\dots \leq \min p (\text{card } ?X2 + \text{card } ?Y2 - 1)$
using $h \text{ enat-ile enat-ord-simps(1) min-def}$
by (*smt (verit, ccv-SIG) linorder-not-le order-le-less order-le-less-subst2*)
ultimately have $\text{card } (?X2 \dots ?Y2) < \min p (\text{card } ?X2 + \text{card } ?Y2 - 1)$
by *order*
then have $hXY1M: (?X2, ?Y2) \in M$ **using** $hY2ne \text{ hX2fin hX2G hXYM M-def}$
by *blast*

Show that $(?X2, ?Y2)$ is a smaller counterexample for the DeVos relation

moreover have $((?X2, ?Y2), (X, Y)) \in \text{Restr devos-rel } ?fin$ **using** $hXYM$
M-def hXY1M h hXY2le
devos-rel-iff hcase by auto
ultimately show *False* **using** *hmin* **by** *blast*
next
case *hcase: False*
then have *h: card ?X1 + card ?Y1 - 1 ≥ card X + card Y - 1* **using** *hXadd*
hYadd by linarith
have *hX1lt: card ?X1 < card X* **using** *hXfin* **by** *(simp add: hpsubX psub-*
set-card-mono)
have *hXY1le: enat (card (?X1 ... ?Y1)) ≤ card (X ... Y)*
using *hXY1 finite-smul card-mono hYfin hXfin* **by** *(metis enat-ord-simps(1))*
also have *... < min p (card X + card Y - 1)* **using** *hXYlt* **by** *auto*
also have *... ≤ min p (card ?X1 + card ?Y1 - 1)* **using** *h enat-ile enat-ord-simps(1)*
min-def
by *(smt (verit, ccfv-threshold) linorder-le-less-linear order.asym order-le-less-trans)*
finally have *hXY1M: (?X1, ?Y1) ∈ M* **using** *M-def hXgne hY1fin hY1G*
hXYM **by** *blast*

Show that $(?X1, ?Y1)$ is a smaller counterexample for the DeVos relation

moreover have $((?X1, ?Y1), (X, Y)) \in \text{Restr devos-rel } ?fin$ **using** $hXYM$
M-def hXY1M h hXY1le
devos-rel-iff hX1lt hXY1le hcase by force
ultimately show *?thesis* **using** *hmin* **by** *blast*
qed
qed
end

References

- [1] M. Bakšys and A. Koutsoukou-Argraki. Kneser’s Theorem and the Cauchy–Davenport 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.