

Khovanskii's Theorem

Angeliki Koutsoukou-Argyraki and Lawrence C. Paulson

March 24, 2023

Abstract

We formalise the proof of an important theorem in additive combinatorics due to Khovanskii [2, 3], attesting that the cardinality of the set of all sums of n many elements of A , where A is a finite subset of an abelian group, is a polynomial in n for all sufficiently large n . We follow a proof of the theorem due to Nathanson and Ruzsa [4, 5] as presented in the notes “Introduction to Additive Combinatorics” by Timothy Gowers [1] for the University of Cambridge.

Contents

1	Product Operator for Commutative Monoids	3
1.1	Products over Finite Sets	3
1.2	Results for Abelian Groups	8
2	Khovanskii's Theorem	9
2.1	Arithmetic operations on lists, pointwise on the elements . .	10
2.2	The pointwise ordering on two equal-length lists of natural numbers	12
2.3	Pointwise minimum and maximum of a set of lists	14
2.4	A locale to fix the finite subset $A \subseteq G$	14
2.5	Adding one to a list element	17
2.6	The set of all r -tuples that sum to n	17
2.7	Lemma 2.7 in Gowers's notes	21
2.8	The set of minimal elements of a set of r -tuples is finite . . .	24
2.9	Towards Lemma 2.9 in Gowers's notes	25
2.10	Towards the main theorem	31

Acknowledgements The authors were supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council.

1 Product Operator for Commutative Monoids

theory *FiniteProduct*

imports

Jacobson-Basic-Algebra.Group-Theory

begin

1.1 Products over Finite Sets

context *commutative-monoid* **begin**

definition *M-ify* $x \equiv \text{if } x \in M \text{ then } x \text{ else } \mathbf{1}$

definition *fincomp* $f A \equiv \text{if finite } A \text{ then } \text{Finite-Set.fold } (\lambda x y. f x \cdot M\text{-ify } y) \mathbf{1} A \text{ else } \mathbf{1}$

lemma *fincomp-empty* [*simp*]: $\text{fincomp } f \ \{\} = \mathbf{1}$
by (*simp add: fincomp-def*)

lemma *fincomp-infinite*[*simp*]: $\text{infinite } A \implies \text{fincomp } f A = \mathbf{1}$
by (*simp add: fincomp-def*)

lemma *left-commute*: $\llbracket a \in M; b \in M; c \in M \rrbracket \implies b \cdot (a \cdot c) = a \cdot (b \cdot c)$
using *commutative* **by** *force*

lemma *comp-fun-commute-onI*:

assumes $f \in F \rightarrow M$

shows *comp-fun-commute-on* $F \ (\lambda x y. f x \cdot M\text{-ify } y)$

using *assms*

by (*auto simp add: comp-fun-commute-on-def Pi-iff M-ify-def left-commute*)

lemma *fincomp-closed* [*simp*]:

assumes $f \in F \rightarrow M$

shows $\text{fincomp } f F \in M$

proof –

interpret *comp-fun-commute-on* $F \ (\lambda x y. f x \cdot M\text{-ify } y)$

by (*simp add: assms comp-fun-commute-onI*)

show *?thesis*

unfolding *fincomp-def*

by (*smt (verit, ccfv-threshold) M-ify-def Pi-iff fold-graph-fold assms composition-closed equalityE fold-graph-closed-lemma unit-closed*)

qed

lemma *fincomp-insert* [*simp*]:

assumes $F: \text{finite } F \ a \notin F$ **and** $f: f \in F \rightarrow M \ f a \in M$

shows $\text{fincomp } f (\text{insert } a F) = f a \cdot \text{fincomp } f F$

proof –

interpret *comp-fun-commute-on* $\text{insert } a F \ (\lambda x y. f x \cdot M\text{-ify } y)$

by (simp add: comp-fun-commute-onI f)
 show ?thesis
 using assms fincomp-closed commutative-monoid.M-ify-def commutative-monoid-axioms
 by (fastforce simp add: fincomp-def)
 qed

lemma *fincomp-unit-eqI*: $(\bigwedge x. x \in A \implies f x = \mathbf{1}) \implies \text{fincomp } f A = \mathbf{1}$

proof (induct A rule: infinite-finite-induct)

case empty show ?case by simp

next

case (insert a A)

have $(\lambda i. \mathbf{1}) \in A \rightarrow M$ by auto

with insert show ?case by simp

qed simp

lemma *fincomp-unit [simp]*: $\text{fincomp } (\lambda i. \mathbf{1}) A = \mathbf{1}$

by (simp add: fincomp-unit-eqI)

lemma *funcset-Int-left [simp, intro]*:

$\llbracket f \in A \rightarrow C; f \in B \rightarrow C \rrbracket \implies f \in A \text{ Int } B \rightarrow C$

by fast

lemma *funcset-Un-left [iff]*:

$(f \in A \text{ Un } B \rightarrow C) = (f \in A \rightarrow C \wedge f \in B \rightarrow C)$

by fast

lemma *fincomp-Un-Int*:

$\llbracket \text{finite } A; \text{finite } B; g \in A \rightarrow M; g \in B \rightarrow M \rrbracket \implies$

$\text{fincomp } g (A \cup B) \cdot \text{fincomp } g (A \cap B) =$

$\text{fincomp } g A \cdot \text{fincomp } g B$

— The reversed orientation looks more natural, but LOOPS as a simprule!

proof (induct set: finite)

case empty then show ?case by simp

next

case (insert a A)

then have $g a \in M$ $g \in A \rightarrow M$ by blast+

with insert show ?case

by (simp add: Int-insert-left associative insert-absorb left-commute)

qed

lemma *fincomp-Un-disjoint*:

$\llbracket \text{finite } A; \text{finite } B; A \cap B = \{\}; g \in A \rightarrow M; g \in B \rightarrow M \rrbracket$

$\implies \text{fincomp } g (A \cup B) = \text{fincomp } g A \cdot \text{fincomp } g B$

by (metis Pi-split-domain fincomp-Un-Int fincomp-closed fincomp-empty right-unit)

lemma *fincomp-comp*:

$\llbracket f \in A \rightarrow M; g \in A \rightarrow M \rrbracket \implies \text{fincomp } (\lambda x. f x \cdot g x) A = (\text{fincomp } f A \cdot \text{fincomp } g A)$

proof (induct A rule: infinite-finite-induct)

```

  case empty show ?case by simp
next
  case (insert a A)
  then have  $f a \in M \ g \in A \rightarrow M \ g a \in M \ f \in A \rightarrow M \ (\lambda x. f x \cdot g x) \in A \rightarrow M$ 
    by blast+
  then show ?case
    by (simp add: insert associative left-commute)
qed simp

```

```

lemma fincomp-cong':
  assumes  $A = B \ g \in B \rightarrow M \ \bigwedge i. i \in B \implies f i = g i$ 
  shows  $\text{fincomp } f A = \text{fincomp } g B$ 
proof (cases finite B)
  case True
  then have ?thesis
    using assms
  proof (induct arbitrary: A)
    case empty thus ?case by simp
  next
    case (insert x B)
    then have  $\text{fincomp } f A = \text{fincomp } f (\text{insert } x B)$  by simp
    also from insert have  $\dots = f x \cdot \text{fincomp } f B$ 
      by (simp add: Pi-iff)
    also from insert have  $\dots = g x \cdot \text{fincomp } g B$  by fastforce
    also from insert have  $\dots = \text{fincomp } g (\text{insert } x B)$ 
      by (intro fincomp-insert [THEN sym]) auto
    finally show ?case .
  qed
  with assms show ?thesis by simp
next
  case False with assms show ?thesis by simp
qed

```

```

lemma fincomp-cong:
  assumes  $A = B \ g \in B \rightarrow M \ \bigwedge i. i \in B = \text{simp} \implies f i = g i$ 
  shows  $\text{fincomp } f A = \text{fincomp } g B$ 
  using assms unfolding simp-implies-def by (blast intro: fincomp-cong')

```

Usually, if this rule causes a failed congruence proof error, the reason is that the premise $g \in B \rightarrow M$ cannot be shown. Adding *Pi-def* to the simpset is often useful. For this reason, *fincomp-cong* is not added to the simpset by default.

```

lemma fincomp-0 [simp]:
   $f \in \{0::\text{nat}\} \rightarrow M \implies \text{fincomp } f \{..0\} = f 0$ 
  by (simp add: Pi-def)

```

```

lemma fincomp-0':  $f \in \{..n\} \rightarrow M \implies (f 0) \cdot \text{fincomp } f \{\text{Suc } 0..n\} = \text{fincomp } f \{..n\}$ 
  by (metis Pi-split-insert-domain Suc-n-not-le-n atLeastAtMost-iff atLeastAtMost-insertL)

```

atMost-atLeast0 finite-atLeastAtMost fincomp-insert le0)

lemma *fincomp-Suc* [*simp*]:

$f \in \{..Suc\ n\} \rightarrow M \implies fincomp\ f\ \{..Suc\ n\} = (f\ (Suc\ n) \cdot fincomp\ f\ \{..n\})$
by (*simp add: Pi-def atMost-Suc*)

lemma *fincomp-Suc2*:

$f \in \{..Suc\ n\} \rightarrow M \implies fincomp\ f\ \{..Suc\ n\} = (fincomp\ (\%i.\ f\ (Suc\ i))\ \{..n\} \cdot f\ 0)$

proof (*induct n*)

case 0 **thus** ?*case* **by** (*simp add: Pi-def*)

next

case *Suc* **thus** ?*case*

by (*simp add: associative Pi-def*)

qed

lemma *fincomp-Suc3*:

assumes $f \in \{..n :: nat\} \rightarrow M$

shows $fincomp\ f\ \{..n\} = (f\ n) \cdot fincomp\ f\ \{..< n\}$

proof (*cases n = 0*)

case *True* **thus** ?*thesis*

using *assms atMost-Suc* **by** *simp*

next

case *False*

then obtain *k* **where** $n = Suc\ k$

using *not0-implies-Suc* **by** *blast*

thus ?*thesis*

using *fincomp-Suc[of f k]* *assms atMost-Suc lessThan-Suc-atMost* **by** *simp*

qed

lemma *fincomp-reindex*:

$f \in (h\ 'A) \rightarrow M \implies$

$inj\text{-on}\ h\ A \implies fincomp\ f\ (h\ 'A) = fincomp\ (\lambda x.\ f\ (h\ x))\ A$

proof (*induct A rule: infinite-finite-induct*)

case (*infinite A*)

hence $\neg\ finite\ (h\ 'A)$

using *finite-imageD* **by** *blast*

with $\langle \neg\ finite\ A \rangle$ **show** ?*case* **by** *simp*

qed (*auto simp add: Pi-def*)

lemma *fincomp-const*:

assumes a [*simp*]: $a \in M$

shows $fincomp\ (\lambda x.\ a)\ A = rec\text{-nat}\ \mathbf{1}\ (\lambda u.\ (\cdot)\ a)\ (card\ A)$

by (*induct A rule: infinite-finite-induct*) *auto*

lemma *fincomp-singleton*:

assumes *i-in-A*: $i \in A$ **and** *fin-A*: *finite A* **and** *f-Pi*: $f \in A \rightarrow M$

shows $fincomp\ (\lambda j.\ if\ i = j\ then\ f\ j\ else\ \mathbf{1})\ A = f\ i$

using *i-in-A fincomp-insert [of A - {i} i (\lambda j. if i = j then f j else 1)]*

$fin-A$ $f-Pi$ $fincomp-unit$ [of $A - \{i\}$]
 $fincomp-cong$ [of $A - \{i\}$ $A - \{i\}$] ($\lambda j. \text{if } i = j \text{ then } f j \text{ else } \mathbf{1}$) ($\lambda i. \mathbf{1}$)
unfolding $Pi-def$ $simp-implies-def$ **by** ($force$ $simp$ $add: insert-absorb$)

lemma $fincomp-singleton-swap$:

assumes $i-in-A$: $i \in A$ **and** $fin-A$: $finite$ A **and** $f-Pi$: $f \in A \rightarrow M$
shows $fincomp$ ($\lambda j. \text{if } j = i \text{ then } f j \text{ else } \mathbf{1}$) $A = f i$
using $fincomp-singleton$ [OF $assms$] **by** ($simp$ $add: eq-commute$)

lemma $fincomp-mono-neutral-cong-left$:

assumes $finite$ B
and $A \subseteq B$
and 1 : $\bigwedge i. i \in B - A \implies h i = \mathbf{1}$
and gh : $\bigwedge x. x \in A \implies g x = h x$
and h : $h \in B \rightarrow M$
shows $fincomp$ g $A = fincomp$ h B

proof –

have eq : $A \cup (B - A) = B$ **using** $\langle A \subseteq B \rangle$ **by** $blast$
have d : $A \cap (B - A) = \{\}$ **using** $\langle A \subseteq B \rangle$ **by** $blast$
from $\langle finite$ $B \rangle$ $\langle A \subseteq B \rangle$ **have** f : $finite$ A $finite$ $(B - A)$
by ($auto$ $intro: finite-subset$)
have $h \in A \rightarrow M$ $h \in B - A \rightarrow M$
using $assms$ **by** ($auto$ $simp: image-subset-iff-funcset$)
moreover **have** $fincomp$ g $A = fincomp$ h $A \cdot fincomp$ h $(B - A)$

proof –

have $fincomp$ h $(B - A) = \mathbf{1}$
using 1 $fincomp-unit-eqI$ **by** $blast$
moreover **have** $fincomp$ g $A = fincomp$ h A
using $\langle h \in A \rightarrow M \rangle$ $fincomp-cong'$ gh **by** $blast$
ultimately **show** $?thesis$
by ($simp$ $add: \langle h \in A \rightarrow M \rangle$)

qed

ultimately **show** $?thesis$

by ($simp$ $add: fincomp-Un-disjoint$ [OF f d , $unfolded$ eq])

qed

lemma $fincomp-mono-neutral-cong-right$:

assumes $finite$ B
and $A \subseteq B$ $\bigwedge i. i \in B - A \implies g i = \mathbf{1}$ $\bigwedge x. x \in A \implies g x = h x$ $g \in B \rightarrow M$
shows $fincomp$ g $B = fincomp$ h A
using $assms$ **by** ($auto$ $intro!: fincomp-mono-neutral-cong-left$ [$symmetric$])

lemma $fincomp-mono-neutral-cong$:

assumes [$simp$]: $finite$ B $finite$ A
and $*$: $\bigwedge i. i \in B - A \implies h i = \mathbf{1}$ $\bigwedge i. i \in A - B \implies g i = \mathbf{1}$
and gh : $\bigwedge x. x \in A \cap B \implies g x = h x$
and g : $g \in A \rightarrow M$
and h : $h \in B \rightarrow M$
shows $fincomp$ g $A = fincomp$ h B

proof–
have $\text{fincomp } g \ A = \text{fincomp } g \ (A \cap B)$
by (*rule fincomp-mono-neutral-cong-right*) (*use assms in auto*)
also have $\dots = \text{fincomp } h \ (A \cap B)$
by (*rule fincomp-cong*) (*use assms in auto*)
also have $\dots = \text{fincomp } h \ B$
by (*rule fincomp-mono-neutral-cong-left*) (*use assms in auto*)
finally show *?thesis* .
qed

lemma *fincomp-UN-disjoint*:
assumes
 $\text{finite } I \ \wedge i. i \in I \implies \text{finite } (A \ i) \ \text{pairwise } (\lambda i \ j. \text{disjnt } (A \ i) \ (A \ j)) \ I$
 $\wedge i \ x. i \in I \implies x \in A \ i \implies g \ x \in M$
shows $\text{fincomp } g \ (\bigcup (A \ ' I)) = \text{fincomp } (\lambda i. \text{fincomp } g \ (A \ i)) \ I$
using *assms*
proof (*induction set: finite*)
case empty
then show *?case*
by force
next
case (insert i I)
then show *?case*
unfolding *pairwise-def disjnt-def*
apply *clarsimp*
apply (*subst fincomp-Un-disjoint*)
apply (*fastforce intro!: funcsetI fincomp-closed*)
done
qed

lemma *fincomp-Union-disjoint*:
 $\llbracket \text{finite } C; \wedge A. A \in C \implies \text{finite } A \ \wedge (\forall x \in A. f \ x \in M); \text{pairwise disjnt } C \rrbracket \implies$
 $\text{fincomp } f \ (\bigcup C) = \text{fincomp } (\text{fincomp } f) \ C$
by (*frule fincomp-UN-disjoint [of C id f]*) *auto*

end

1.2 Results for Abelian Groups

context *abelian-group* **begin**

lemma *fincomp-inverse*:
 $f \in A \rightarrow G \implies \text{fincomp } (\lambda x. \text{inverse } (f \ x)) \ A = \text{inverse } (\text{fincomp } f \ A)$
proof (*induct A rule: infinite-finite-induct*)
case empty show *?case* **by simp**
next
case (insert a A)
then have $f \ a \in G \ f \in A \rightarrow G \ (\lambda x. \text{inverse } (f \ x)) \in A \rightarrow G$


```

    by blast+
  with insert show ?case
    by (simp add: commutative inverse-composition-commute)
qed simp

```

Jeremy Avigad. This should be generalized to arbitrary groups, not just Abelian ones, using Lagrange’s theorem.

```

lemma power-order-eq-one:
  assumes fin [simp]: finite G
    and a [simp]: a ∈ G
  shows rec-nat 1 (λu. (·) a) (card G) = 1
proof -
  have rec-G: rec-nat 1 (λu. (·) a) (card G) ∈ G
    by (metis Pi-I' a fincomp-closed fincomp-const)
  have ∧x. x ∈ G ⇒ x ∈ (·) a ‘ G
    by (metis a composition-closed imageI invertible invertible-inverse-closed invert-
    ible-right-inverse2)
  with a have (·) a ‘ G = G by blast
  then have 1 · fincomp (λx. x) G = fincomp (λx. x) ((·) a ‘ G)
    by simp
  also have ... = fincomp (λx. a · x) G
    using fincomp-reindex
    by (subst (2) fincomp-reindex [symmetric]) (auto simp: inj-on-def)
  also have ... = fincomp (λx. a) G · fincomp (λx. x) G
    by (simp add: fincomp-comp)
  also have fincomp (λx. a) G = rec-nat 1 (λu. (·) a) (card G)
    by (simp add: fincomp-const)
  finally show ?thesis
    by (metis commutative fincomp-closed funcset-id invertible invertible-left-cancel
    rec-G unit-closed)
qed

end

end

```

2 Khovanskii’s Theorem

We formalise the proof of an important theorem in additive combinatorics due to Khovanskii, attesting that the cardinality of the set of all sums of n many elements of A , where A is a finite subset of an abelian group, is a polynomial in n for all sufficiently large n . We follow a proof due to Nathanson and Ruzsa as presented in the notes Introduction to Additive Combinatorics by Timothy Gowers for the University of Cambridge.

```

theory Khovanskii
  imports
    Complex-Main

```

FiniteProduct
HOL-Library.Equipollence
Pluenecke-Ruzsa-Inequality.Pluenecke-Ruzsa-Inequality
Bernoulli.Bernoulli — sums of a fixed power are polynomials
HOL-Analysis.Weierstrass-Theorems — needed for polynomial function
HOL-Library.List-Lenlexorder — lexicographic ordering for the type *nat*
list

begin

The sum of the elements of a list

abbreviation $\sigma \equiv \text{sum-list}$

Related to the nsets of Ramsey.thy, but simpler

definition $\text{finsets} :: ['a \text{ set}, \text{nat}] \Rightarrow 'a \text{ set set}$
where $\text{finsets } A \ n \equiv \{N. N \subseteq A \wedge \text{card } N = n\}$

lemma $\text{card-finsets: finite } N \Longrightarrow \text{card } (\text{finsets } N \ k) = \text{card } N \ \text{choose } k$
by (*simp add: finsets-def n-subsets*)

lemma $\text{sorted-map-plus-iff}$ [*simp*]:
fixes $a :: 'a :: \text{linordered-cancel-ab-semigroup-add}$
shows $\text{sorted } (\text{map } ((+) \ a) \ xs) \longleftrightarrow \text{sorted } xs$
by (*induction xs*) *auto*

lemma $\text{distinct-map-plus-iff}$ [*simp*]:
fixes $a :: 'a :: \text{linordered-cancel-ab-semigroup-add}$
shows $\text{distinct } (\text{map } ((+) \ a) \ xs) \longleftrightarrow \text{distinct } xs$
by (*induction xs*) *auto*

2.1 Arithmetic operations on lists, pointwise on the elements

Weak type class properties. Cancellation is difficult to arrange because of complications when lists differ in length.

instantiation $\text{list} :: (\text{plus}) \ \text{plus}$

begin

definition $\text{plus-list} \equiv \text{map2 } (+)$

instance..

end

lemma length-plus-list [*simp*]:
fixes $xs :: 'a :: \text{plus list}$
shows $\text{length } (xs+ys) = \min (\text{length } xs) (\text{length } ys)$
by (*simp add: plus-list-def*)

lemma plus-Nil [*simp*]: $[] + xs = []$
by (*simp add: plus-list-def*)

lemma $\text{plus-Cons: } (y \# ys) + (x \# xs) = (y+x) \# (ys+xs)$
by (*simp add: plus-list-def*)

```

lemma nth-plus-list [simp]:
  fixes xs :: 'a::plus list
  assumes i < length xs i < length ys
  shows (xs+ys)!i = xs!i + ys!i
  by (simp add: plus-list-def assms)

instantiation list :: (minus) minus
begin
definition minus-list ≡ map2 (-)
instance..
end

lemma length-minus-list [simp]:
  fixes xs :: 'a::minus list
  shows length (xs-y) = min (length xs) (length ys)
  by (simp add: minus-list-def)

lemma minus-Nil [simp]: [] - xs = []
  by (simp add: minus-list-def)

lemma minus-Cons: (y # ys) - (x # xs) = (y-x) # (ys-xs)
  by (simp add: minus-list-def)

lemma nth-minus-list [simp]:
  fixes xs :: 'a::minus list
  assumes i < length xs i < length ys
  shows (xs-y)!i = xs!i - ys!i
  by (simp add: minus-list-def assms)

instance list :: (ab-semigroup-add) ab-semigroup-add
proof
  have map2 (+) (map2 (+) xs ys) zs = map2 (+) xs (map2 (+) ys zs) for xs ys
  zs :: 'a list
  proof (induction xs arbitrary: ys zs)
    case (Cons x xs)
    show ?case
    proof (cases ys=[] ∨ zs=[])
      case False
      then obtain y ys' z zs' where ys = y#ys' zs = z # zs'
      by (meson list.exhaust)
      then show ?thesis
      by (simp add: Cons add.assoc)
    qed auto
  qed auto
  then show a + b + c = a + (b + c) for a b c :: 'a list
  by (auto simp: plus-list-def)
next

```

```

have map2 (+) xs ys = map2 (+) ys xs for xs ys :: 'a list
proof (induction xs arbitrary: ys)
  case (Cons x xs)
  show ?case
  proof (cases ys)
    case (Cons y ys^)
    then show ?thesis
      by (simp add: Cons.IH add.commute)
  qed auto
qed auto
then show a + b = b + a for a b :: 'a list
  by (auto simp: plus-list-def)
qed

```

2.2 The pointwise ordering on two equal-length lists of natural numbers

Gowers uses the usual symbol (\leq) for his pointwise ordering. In our development, \leq is the lexicographic ordering and \sqsubseteq is the pointwise ordering.

definition *pointwise-le* :: [nat list, nat list] \Rightarrow bool (**infixr** \sqsubseteq 50)
where $x \sqsubseteq y \equiv \text{list-all2 } (\leq) x y$

definition *pointwise-less* :: [nat list, nat list] \Rightarrow bool (**infixr** \triangleleft 50)
where $x \triangleleft y \equiv x \sqsubseteq y \wedge x \neq y$

lemma *pointwise-le-iff-nth*:
 $x \sqsubseteq y \longleftrightarrow \text{length } x = \text{length } y \wedge (\forall i < \text{length } x. x!i \leq y!i)$
by (simp add: list-all2-conv-all-nth pointwise-le-def)

lemma *pointwise-le-iff*:
 $x \sqsubseteq y \longleftrightarrow \text{length } x = \text{length } y \wedge (\forall (i,j) \in \text{set } (\text{zip } x y). i \leq j)$
by (simp add: list-all2-iff pointwise-le-def)

lemma *pointwise-append-le-iff* [simp]: $u @ x \sqsubseteq u @ y \longleftrightarrow x \sqsubseteq y$
by (auto simp: pointwise-le-iff-nth nth-append)

lemma *pointwise-le-refl* [iff]: $x \sqsubseteq x$
by (simp add: list.rel-refl pointwise-le-def)

lemma *pointwise-le-antisym*: $\llbracket x \sqsubseteq y; y \sqsubseteq x \rrbracket \Longrightarrow x = y$
by (metis antisym list-all2-antisym pointwise-le-def)

lemma *pointwise-le-trans*: $\llbracket x \sqsubseteq y; y \sqsubseteq z \rrbracket \Longrightarrow x \sqsubseteq z$
by (smt (verit, del-insts) le-trans list-all2-trans pointwise-le-def)

lemma *pointwise-le-Nil* [simp]: $\text{Nil} \sqsubseteq x \longleftrightarrow x = \text{Nil}$
by (simp add: pointwise-le-def)

lemma *pointwise-le-Nil2* [simp]: $x \sqsubseteq \text{Nil} \longleftrightarrow x = \text{Nil}$

by (*simp add: pointwise-le-def*)

lemma *pointwise-le-iff-less-equal*: $x \sqsubseteq y \longleftrightarrow x \triangleleft y \vee x = y$
using *pointwise-less-def* by *blast*

lemma *pointwise-less-iff*:
 $x \triangleleft y \longleftrightarrow x \sqsubseteq y \wedge (\exists (i,j) \in \text{set } (\text{zip } x \ y). i < j)$
using *list-eq-iff-zip-eq pointwise-le-iff pointwise-less-def* by *fastforce*

lemma *pointwise-less-iff2*: $x \triangleleft y \longleftrightarrow x \sqsubseteq y \wedge (\exists k < \text{length } x. x!k < y ! k)$
unfolding *pointwise-less-def pointwise-le-iff-nth*
by (*fastforce intro!: nth-equalityI*)

lemma *pointwise-less-Nil* [*simp*]: $\neg \text{Nil} \triangleleft x$
by (*simp add: pointwise-less-def*)

lemma *pointwise-less-Nil2* [*simp*]: $\neg x \triangleleft \text{Nil}$
by (*simp add: pointwise-less-def*)

lemma *zero-pointwise-le-iff* [*simp*]: $\text{replicate } r \ 0 \sqsubseteq x \longleftrightarrow \text{length } x = r$
by (*auto simp: pointwise-le-iff-nth*)

lemma *pointwise-le-imp- σ* :
assumes $xs \sqsubseteq ys$ **shows** $\sigma \ xs \leq \sigma \ ys$
using *assms*
proof (*induction ys arbitrary: xs*)
case *Nil*
then show *?case*
by (*simp add: pointwise-le-iff*)
next
case (*Cons y ys*)
then obtain $x \ xs'$ **where** $x \leq y \ xs = x \# \ xs' \ xs' \sqsubseteq ys$
by (*auto simp: pointwise-le-def list-all2-Cons2*)
then show *?case*
by (*simp add: Cons.IH add-le-mono*)
qed

lemma *sum-list-plus*:
fixes $xs :: 'a::\text{comm-monoid-add}$ *list*
assumes $\text{length } xs = \text{length } ys$ **shows** $\sigma \ (xs + ys) = \sigma \ xs + \sigma \ ys$
using *assms* **by** (*simp add: plus-list-def case-prod-unfold sum-list-addr*)

lemma *sum-list-minus*:
assumes $xs \sqsubseteq ys$ **shows** $\sigma \ (ys - xs) = \sigma \ ys - \sigma \ xs$
using *assms*
proof (*induction ys arbitrary: xs*)
case (*Cons y ys*)
then obtain $x \ xs'$ **where** $x \leq y \ xs = x \# \ xs' \ xs' \sqsubseteq ys$
by (*auto simp: pointwise-le-def list-all2-Cons2*)

then show *?case*
using *pointwise-le-imp-σ* **by** (*auto simp: Cons minus-Cons*)
qed (*auto simp: in-set-conv-nth*)

2.3 Pointwise minimum and maximum of a set of lists

definition *min-pointwise* :: $[nat, nat\ list\ set] \Rightarrow nat\ list$
where *min-pointwise* $\equiv \lambda r\ U. map\ (\lambda i. Min\ ((\lambda u. u!i)\ 'U))\ [0..<r]$

lemma *min-pointwise-le*: $\llbracket u \in U; finite\ U \rrbracket \Longrightarrow min-pointwise\ (length\ u)\ U \trianglelefteq u$
by (*simp add: min-pointwise-def pointwise-le-iff-nth*)

lemma *min-pointwise-ge-iff*:
assumes *finite U U ≠ {}* $\bigwedge u. u \in U \Longrightarrow length\ u = r\ length\ x = r$
shows $x \trianglelefteq min-pointwise\ r\ U \longleftrightarrow (\forall u \in U. x \trianglelefteq u)$
by (*auto simp: min-pointwise-def pointwise-le-iff-nth assms*)

definition *max-pointwise* :: $[nat, nat\ list\ set] \Rightarrow nat\ list$
where *max-pointwise* $\equiv \lambda r\ U. map\ (\lambda i. Max\ ((\lambda u. u!i)\ 'U))\ [0..<r]$

lemma *max-pointwise-ge*: $\llbracket u \in U; finite\ U \rrbracket \Longrightarrow u \trianglelefteq max-pointwise\ (length\ u)\ U$
by (*simp add: max-pointwise-def pointwise-le-iff-nth*)

lemma *max-pointwise-le-iff*:
assumes *finite U U ≠ {}* $\bigwedge u. u \in U \Longrightarrow length\ u = r\ length\ x = r$
shows $max-pointwise\ r\ U \trianglelefteq x \longleftrightarrow (\forall u \in U. u \trianglelefteq x)$
by (*auto simp: max-pointwise-def pointwise-le-iff-nth assms*)

lemma *max-pointwise-mono*:
assumes $X' \subseteq X\ finite\ X\ X' \neq \{\}$
shows $max-pointwise\ r\ X' \trianglelefteq max-pointwise\ r\ X$
using *assms* **by** (*simp add: max-pointwise-def pointwise-le-iff-nth Max-mono image-mono*)

lemma *pointwise-le-plus*: $\llbracket xs \trianglelefteq ys; length\ ys \leq length\ zs \rrbracket \Longrightarrow xs \trianglelefteq ys+zs$

proof (*induction xs arbitrary: ys zs*)

case (*Cons x xs*)

then obtain $y\ ys'\ z\ zs'$ **where** $ys = y\#\ ys'\ zs = z\#\ zs'$

unfolding *pointwise-le-iff* **by** (*metis Suc-le-length-iff le-refl length-Cons*)

with *Cons* **show** *?case*

by (*auto simp: plus-list-def pointwise-le-def*)

qed (*simp add: pointwise-le-iff*)

lemma *pairwise-minus-cancel*: $\llbracket z \trianglelefteq x; z \trianglelefteq y; x - z = y - z \rrbracket \Longrightarrow x = y$
unfolding *pointwise-le-iff-nth* **by** (*metis eq-diff-iff nth-equalityI nth-minus-list*)

2.4 A locale to fix the finite subset $A \subseteq G$

locale *Khovanskii = additive-abelian-group* +
fixes $A :: 'a\ set$

assumes $AsubG: A \subseteq G$ **and** $finA: finite\ A$

begin

finite products of a group element

definition $Gmult :: 'a \Rightarrow nat \Rightarrow 'a$
where $Gmult\ a\ n \equiv (((\oplus)a) \overset{\sim}{\sim} n)\ \mathbf{0}$

lemma $Gmult-0$ [*simp*]: $Gmult\ a\ 0 = \mathbf{0}$
by (*simp add: Gmult-def*)

lemma $Gmult-1$ [*simp*]: $a \in G \implies Gmult\ a\ (Suc\ 0) = a$
by (*simp add: Gmult-def*)

lemma $Gmult-Suc$ [*simp*]: $Gmult\ a\ (Suc\ n) = a \oplus Gmult\ a\ n$
by (*simp add: Gmult-def*)

lemma $Gmult-in-G$ [*simp,intro*]: $a \in G \implies Gmult\ a\ n \in G$
by (*induction n*) *auto*

lemma $Gmult-add-add$:
assumes $a \in G$
shows $Gmult\ a\ (m+n) = Gmult\ a\ m \oplus Gmult\ a\ n$
by (*induction m*) (*use assms local.associative in fastforce*)⁺

lemma $Gmult-add-diff$:
assumes $a \in G$
shows $Gmult\ a\ (n+k) \ominus Gmult\ a\ n = Gmult\ a\ k$
by (*metis Gmult-add-add Gmult-in-G assms commutative inverse-closed invertible invertible-left-inverse2*)

lemma $Gmult-diff$:
assumes $a \in G\ n \leq m$
shows $Gmult\ a\ m \ominus Gmult\ a\ n = Gmult\ a\ (m-n)$
by (*metis Gmult-add-diff assms le-add-diff-inverse*)

Mapping elements of A to their numeric subscript

abbreviation $idx \equiv to-nat-on\ A$

The elements of A in order

definition $aA :: 'a\ list$
where $aA \equiv map\ (from-nat-into\ A)\ [0..<card\ A]$

definition $\alpha :: nat\ list \Rightarrow 'a$
where $\alpha \equiv \lambda x. fincomp\ (\lambda i. Gmult\ (aA!i)\ (x!i))\ \{..<card\ A\}$

The underlying assumption is $length\ y = length\ x$

definition $useless :: nat\ list \Rightarrow bool$
where $useless\ x \equiv \exists y < x. \sigma\ y = \sigma\ x \wedge \alpha\ y = \alpha\ x \wedge length\ y = length\ x$

abbreviation $useful\ x \equiv \neg\ useless\ x$

lemma $alpha\ replicate\ 0$ [simp]: $\alpha\ (replicate\ (card\ A)\ 0) = \mathbf{0}$
by (auto simp: $\alpha\text{-def}$ intro: $fincomp\ unit\ eqI$)

lemma $idx\ less\ cardA$:
assumes $a \in A$ shows $idx\ a < card\ A$
by (metis *assms* $bij\ betw\ def\ finA\ imageI\ lessThan\ iff\ to\ nat\ on\ finite$)

lemma $aA\ idx\ eq$ [simp]:
assumes $a \in A$ shows $aA\ !\ (idx\ a) = a$
by (simp add: $aA\ def\ assms\ countable\ finite\ finA\ idx\ less\ cardA$)

lemma $set\ aA$: $set\ aA = A$
using $bij\ betw\ from\ nat\ into\ finite$ [OF $finA$]
by (simp add: $aA\ def\ atLeast0LessThan\ bij\ betw\ def$)

lemma $nth\ aA\ in\ G$ [simp]: $i < card\ A \implies aA!i \in G$
using $AsubG\ aA\ def\ set\ aA$ by auto

lemma $alpha\ in\ G$ [iff]: $\alpha\ x \in G$
using $nth\ aA\ in\ G\ fincomp\ closed$ by (simp add: $\alpha\ def$)

lemma $Gmult\ in\ PiG$ [simp]: $(\lambda i. Gmult\ (aA!i)\ (f\ i)) \in \{..\ < card\ A\} \rightarrow G$
by *simp*

lemma $alpha\ plus$:
assumes $length\ x = card\ A\ length\ y = card\ A$
shows $\alpha\ (x + y) = \alpha\ x \oplus \alpha\ y$
proof –
have $\alpha\ (x + y) = fincomp\ (\lambda i. Gmult\ (aA!i)\ (map2\ (+)\ x\ y!i)) \{..\ < card\ A\}$
by (simp add: $\alpha\ def\ plus\ list\ def$)
also have $\dots = fincomp\ (\lambda i. Gmult\ (aA!i)\ (x!i + y!i)) \{..\ < card\ A\}$
by (intro $fincomp\ cong'$; simp add: *assms*)
also have $\dots = fincomp\ (\lambda i. Gmult\ (aA!i)\ (x!i) \oplus Gmult\ (aA!i)\ (y!i)) \{..\ < card\ A\}$
by (intro $fincomp\ cong'$; simp add: $Gmult\ add\ add$)
also have $\dots = \alpha\ x \oplus \alpha\ y$
by (simp add: $\alpha\ def\ fincomp\ comp$)
finally show ?thesis .

qed

lemma $alpha\ minus$:
assumes $y \sqsubseteq x\ length\ y = card\ A$
shows $\alpha\ (x - y) = \alpha\ x \ominus \alpha\ y$
proof –
have $\alpha\ (x - y) = fincomp\ (\lambda i. Gmult\ (aA!i)\ (map2\ (-)\ x\ y!i)) \{..\ < card\ A\}$
by (simp add: $\alpha\ def\ minus\ list\ def$)

also have $\dots = \text{fincomp } (\lambda i. \text{Gmult } (aA!i) (x!i - y!i)) \{..<\text{card } A\}$
using *assms* **by** (*intro fincomp-cong'*) (*auto simp: pointwise-le-iff*)
also have $\dots = \text{fincomp } (\lambda i. \text{Gmult } (aA!i) (x!i) \ominus \text{Gmult } (aA!i) (y!i)) \{..<\text{card } A\}$
using *assms*
by (*intro fincomp-cong'*) (*simp add: pointwise-le-iff-nth Gmult-diff*)+
also have $\dots = \alpha x \ominus \alpha y$
by (*simp add: α -def fincomp-comp fincomp-inverse*)
finally show *?thesis* .
qed

2.5 Adding one to a list element

definition *list-incr* :: $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$
where *list-incr* $i x \equiv x[i := \text{Suc } (x!i)]$

lemma *list-incr-Nil* [*simp*]: *list-incr* $i [] = []$
by (*simp add: list-incr-def*)

lemma *list-incr-Cons* [*simp*]: *list-incr* (*Suc* i) ($k\#ks$) = $k \# \text{list-incr } i ks$
by (*simp add: list-incr-def*)

lemma *sum-list-incr* [*simp*]: $i < \text{length } x \Longrightarrow \sigma (\text{list-incr } i x) = \text{Suc } (\sigma x)$
by (*auto simp: list-incr-def sum-list-update*)

lemma *length-list-incr* [*simp*]: $\text{length } (\text{list-incr } i x) = \text{length } x$
by (*auto simp: list-incr-def*)

lemma *nth-le-list-incr*: $i < \text{card } A \Longrightarrow x!i \leq \text{list-incr } (idx a) x!i$
unfolding *list-incr-def*
by (*metis Suc-leD linorder-not-less list-update-beyond nth-list-update-eq nth-list-update-neq order-refl*)

lemma *list-incr-nth-diff*: $i < \text{length } x \Longrightarrow \text{list-incr } j x!i - x!i = (\text{if } i = j \text{ then } 1 \text{ else } 0)$
by (*simp add: list-incr-def*)

2.6 The set of all r -tuples that sum to n

definition *length-sum-set* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat list set}$
where *length-sum-set* $r n \equiv \{x. \text{length } x = r \wedge \sigma x = n\}$

lemma *length-sum-set-Nil* [*simp*]: *length-sum-set* $0 n = (\text{if } n=0 \text{ then } \{[]\} \text{ else } \{\})$
by (*auto simp: length-sum-set-def*)

lemma *length-sum-set-Suc* [*simp*]: $k\#ks \in \text{length-sum-set } (Suc r) n \longleftrightarrow (\exists m. ks \in \text{length-sum-set } r m \wedge n = m+k)$
by (*auto simp: length-sum-set-def*)

lemma *length-sum-set-Suc-epoll*: $\text{length-sum-set } (\text{Suc } r) \ n \approx \text{Sigma } \{..n\} \ (\lambda i. \text{length-sum-set } r \ (n-i))$ (is ?L \approx ?R)

unfolding *epoll-def*

proof

let ?f = ($\lambda l. (\text{hd } l, \text{tl } l)$)

show *bij-betw* ?f ?L ?R

proof (*intro* *bij-betw-imageI*)

show *inj-on* ?f ?L

by (*force simp: inj-on-def length-sum-set-def intro: list.expand*)

show ?f ' ?L = ?R

by (*force simp: length-sum-set-def length-Suc-conv*)

qed

qed

lemma *finite-length-sum-set*: *finite* ($\text{length-sum-set } r \ n$)

proof (*induction* *r* *arbitrary: n*)

case 0

then show ?case

by (*auto simp: length-sum-set-def*)

next

case (*Suc r*)

then show ?case

using *length-sum-set-Suc-epoll* *epoll-finite-iff* **by** *blast*

qed

lemma *card-length-sum-set*: $\text{card } (\text{length-sum-set } (\text{Suc } r) \ n) = (\sum i \leq n. \text{card } (\text{length-sum-set } r \ (n-i)))$

proof –

have $\text{card } (\text{length-sum-set } (\text{Suc } r) \ n) = \text{card } (\text{Sigma } \{..n\} \ (\lambda i. \text{length-sum-set } r \ (n-i)))$

by (*metis* *epoll-finite-iff* *epoll-iff-card* *finite-length-sum-set* *length-sum-set-Suc-epoll*)

also have $\dots = (\sum i \leq n. \text{card } (\text{length-sum-set } r \ (n-i)))$

by (*simp add: finite-length-sum-set*)

finally show ?thesis .

qed

lemma *sum-up-index-split'*:

assumes $N \leq n$ **shows** $(\sum i \leq n. f \ i) = (\sum i \leq n-N. f \ i) + (\sum i = \text{Suc } (n-N)..n. f \ i)$

by (*metis* *assms* *diff-add* *sum-up-index-split*)

lemma *sum-invert*: $N \leq n \implies (\sum i = \text{Suc } (n - N)..n. f \ (n - i)) = (\sum j < N. f \ j)$

proof (*induction* *N*)

case (*Suc N*)

then show ?case

apply (*auto simp: Suc-diff-Suc*)

by (*metis* *sum.atLeast-Suc-atMost* *Suc-leD* *add commute* *diff-diff-cancel* *diff-le-self*)

qed *auto*

lemma *sum-diff-split*:
assumes $N \leq n$
shows $(\sum_{i \leq n - N}. \text{real } (n - i) \wedge j) = (\sum_{i \leq n}. \text{real } i \wedge j) - (\sum_{i < N}. \text{real } i \wedge j)$
proof –
have *inj*: *inj-on* $((-) n) \{N..n\}$
by (*auto simp: inj-on-def*)
have $(\sum_{i \leq n - N}. \text{real } (n - i) \wedge j) = (\sum_{i \in (-) n \text{ ' } \{N..n\}}. \text{real } (n - i) \wedge j)$
proof (*rule sum.cong*)
have $\bigwedge x. x \leq n - N \implies \exists m \geq N. m \leq n \wedge x = n - m$
by (*metis assms diff-diff-cancel diff-le-mono2 diff-le-self le-trans*)
then show $\{..n - N\} = (-) n \text{ ' } \{N..n\}$
by (*auto simp: image-iff Bex-def*)
qed *auto*
also have $\dots = (\sum_{i=N..n}. \text{real } i \wedge j)$
using *sum.reindex* [*OF inj, of $\lambda i. \text{real } (n - i) \wedge j, \text{symmetric}$*]
by (*simp add:*)
also have $\dots = (\sum_{i \leq n}. \text{real } i \wedge j) - (\sum_{i < N}. \text{real } i \wedge j)$
proof (*cases N*)
case 0
then show *?thesis*
using *atMost-atLeast0* **by** *auto*
next
case (*Suc N'*)
then show *?thesis*
using *assms sum-up-index-split* [*of - N' n - N'*]
by (*smt (verit, best) Suc-diff-le add-Suc-shift diff-Suc-Suc le-add-diff-inverse lessThan-Suc-atMost*)
qed
finally show *?thesis* .
qed

lemma *real-polynomial-function-length-sum-set*:
 $\exists p. \text{real-polynomial-function } p \wedge (\forall n > 0. \text{real } (\text{card } (\text{length-sum-set } r \ n))) = p$
(real n)
proof (*induction r*)
case 0
have $\forall n > 0. \text{real } (\text{card } (\text{length-sum-set } 0 \ n)) = 0$
by *auto*
then show *?case*
by *blast*
next
case (*Suc r*)
then obtain *p* **where** *poly: real-polynomial-function p*
and $p: \bigwedge n. n > 0 \implies \text{real } (\text{card } (\text{length-sum-set } r \ n)) = p$ *(real n)*
by *blast*
then obtain *a n* **where** *p-eq: $p = (\lambda x. \sum_{i \leq n}. a \ i * x \wedge i)$*

using *real-polynomial-function-iff-sum* **by** *auto*
define q **where** $q \equiv \lambda x. \sum_{j \leq n}. a \ j * ((\text{bernpoly } (\text{Suc } j) (1 + x) - \text{bernpoly } (\text{Suc } j) 0) / (1 + \text{real } j) - 0 \wedge j)$
have $rp\text{-}q$: *real-polynomial-function* q
by (*fastforce simp: bernpoly-def p-eq q-def*)
have $q\text{-eq}$: $(\sum_{x \leq n - 1}. p (\text{real } (n - x))) = q (\text{real } n)$ **if** $n > 0$ **for** n
using *that*
by (*simp add: p-eq q-def sum.swap sum-diff-split add commute sum-of-powers flip: sum-distrib-left*)
define p' **where** $p' \equiv \lambda x. q \ x + \text{real } (\text{card } (\text{length-sum-set } r \ 0))$
have *real-polynomial-function* p'
using $rp\text{-}q$ **by** (*force simp: p'-def*)
moreover **have** $(\sum_{x \leq n - \text{Suc } 0}. p (\text{real } (n - x))) + \text{real } (\text{card } (\text{length-sum-set } r \ 0)) = p' (\text{real } n)$ **if** $n > 0$ **for** n
using *that q-eq* **by** (*auto simp: p'-def*)
ultimately show *?case*
unfolding *card-length-sum-set*
by (*force simp: sum-up-index-split' [of 1] p sum-invert*)
qed

lemma *all-zeroes-replicate: length-sum-set r 0 = {replicate r 0}*
by (*auto simp: length-sum-set-def replicate-eqI*)

lemma *length-sum-set-Suc-eq-UN: length-sum-set r (Suc n) = ($\bigcup_{i < r}. \text{list-incr } i$ ' length-sum-set r n)*

proof –
have $\exists i < r. x \in \text{list-incr } i$ ' *length-sum-set r n*
if $\sigma \ x = \text{Suc } n$ **and** $r = \text{length } x$ **for** x
proof –
have $x \neq \text{replicate } r \ 0$
using *that* **by** (*metis sum-list-replicate Zero-not-Suc mult-zero-right*)
then obtain i **where** $i < r$ $x!i \neq 0$
by (*metis <r = length x> in-set-conv-nth replicate-eqI*)
with *that* **have** $x[i := x!i - 1] \in \text{length-sum-set } r \ n$
by (*simp add: sum-list-update length-sum-set-def*)
with i **that** **show** *?thesis*
unfolding *list-incr-def* **by** *force*
qed
then show *?thesis*
by (*auto simp: length-sum-set-def Bex-def*)
qed

lemma *alpha-list-incr:*
assumes $a \in A$ $x \in \text{length-sum-set } (\text{card } A) \ n$
shows $\alpha (\text{list-incr } (\text{id } a) \ x) = a \oplus \alpha \ x$
proof –
have $\text{len } x = \text{card } A$
using *assms length-sum-set-def* **by** *blast*

have α (*list-incr* (*idx a*) *x*) \ominus α *x* = *fincomp* ($\lambda i. \text{Gmult } (aA!i) (\text{list-incr } (idx a) x!i) \ominus \text{Gmult } (aA!i) (x!i)) \{..<card A\}$
by (*simp add: α -def fincomp-comp fincomp-inverse*)
also have $\dots = \text{fincomp } (\lambda i. \text{Gmult } (aA!i) (\text{list-incr } (idx a) x!i - x!i)) \{..<card A\}$
by (*intro fincomp-cong; simp add: Gmult-diff nth-le-list-incr*)
also have $\dots = \text{fincomp } (\lambda i. \text{if } i = idx a \text{ then } (aA!i) \text{ else } \mathbf{0}) \{..<card A\}$
by (*intro fincomp-cong'; simp add: list-incr-nth-diff lenx*)
also have $\dots = a$
using *assms* **by** (*simp add: fincomp-singleton-swap idx-less-cardA*)
finally have α (*list-incr* (*idx a*) *x*) \ominus α *x* = *a* .
then show *?thesis*
by (*metis alpha-in-G associative inverse-closed invertible invertible-left-inverse right-unit*)
qed

lemma *sumset-iterated-enum:*

defines $r \equiv card A$
shows *sumset-iterated* $A n = \alpha$ ‘ *length-sum-set* $r n$
proof (*induction n*)
case 0
then show *?case*
by (*simp add: all-zeroes-replicate r-def*)
next
case (*Suc n*)
have *eq*: $\{..<r\} = idx$ ‘ A
by (*metis bij-betw-def finA r-def to-nat-on-finite*)
have *sumset-iterated* $A (Suc n) = (\bigcup a \in A. (\lambda i. a \oplus \alpha i))$ ‘ *length-sum-set* $r n$
using *AsubG* **by** (*auto simp: Suc sumset*)
also have $\dots = (\bigcup a \in A. (\lambda i. \alpha (\text{list-incr } (idx a) i)))$ ‘ *length-sum-set* $r n$
by (*simp add: alpha-list-incr r-def*)
also have $\dots = \alpha$ ‘ *length-sum-set* $r (Suc n)$
by (*simp add: image-UN image-comp length-sum-set-Suc-eq-UN eq*)
finally show *?case* .
qed

2.7 Lemma 2.7 in Gowers’s notes

The following lemma corresponds to a key fact about the cardinality of the set of all sums of n many elements of A , stated before Gowers’s Lemma 2.7.

lemma *card-sumset-iterated-length-sum-set-useful:*

defines $r \equiv card A$
shows $card(\text{sumset-iterated } A n) = card(\text{length-sum-set } r n \cap \{x. \text{useful } x\})$
(is card ?L = card ?R)
proof –
have $\alpha x \in \alpha$ ‘ (*length-sum-set* $r n \cap \{x. \text{useful } x\}$)
if $x \in \text{length-sum-set } r n$ **for** x
proof –
define y **where** $y \equiv LEAST y. y \in \text{length-sum-set } r n \wedge \alpha y = \alpha x$

```

have  $y$ :  $y \in \text{length-sum-set } (\text{card } A) \ n \wedge \alpha \ y = \alpha \ x$ 
  by (metis (mono-tags, lifting) LeastI r-def y-def that)
moreover
have useful  $y$ 
proof (clarsimp simp: useless-def)
  show False
    if  $\sigma \ z = \sigma \ y$   $\text{length } z = \text{length } y$  and  $z < y$   $\alpha \ z = \alpha \ y$  for  $z$ 
    using that Least-le length-sum-set-def not-less-Least r-def y y-def by fastforce
qed
ultimately show ?thesis
  unfolding image-iff length-sum-set-def r-def by (smt (verit) Int-Collect)
qed
then have sumset-iterated  $A \ n = \alpha \ ' (\text{length-sum-set } r \ n \cap \{x. \text{useful } x\})$ 
  by (auto simp: sumset-iterated-enum length-sum-set-def r-def)
moreover have inj-on  $\alpha \ (\text{length-sum-set } r \ n \cap \{x. \text{useful } x\})$ 
apply (simp add: image-iff length-sum-set-def r-def inj-on-def useless-def Ball-def)
  by (metis linorder-less-linear)
ultimately show ?thesis
  by (simp add: card-image length-sum-set-def)
qed

```

The following lemma corresponds to Lemma 2.7 in Gowers's notes.

```

lemma useless-leq-useless:
  defines  $r \equiv \text{card } A$ 
  assumes useless  $x$  and  $x \leq y$  and  $\text{length } x = r$ 
  shows useless  $y$ 
proof –
  have leny:  $\text{length } y = r$ 
    using pointwise-le-iff assms by auto
  obtain  $x'$  where  $x' < x$  and  $\sigma x'$ :  $\sigma \ x' = \sigma \ x$  and  $\alpha x'$ :  $\alpha \ x' = \alpha \ x$  and lenx':
 $\text{length } x' = \text{length } x$ 
    using assms useless-def by blast
  obtain  $i$  where  $i < \text{card } A$  and  $x_i$ :  $x^!i < x^!i$  and takex':  $\text{take } i \ x' = \text{take } i \ x$ 
    using  $\langle x' < x \rangle$  lenx' assms by (auto simp: list-less-def lenlex-def elim!: lex-take-index)
  define  $y'$  where  $y' \equiv y + x' - x$ 
  have leny':  $\text{length } y' = \text{length } y$ 
    using assms lenx' pointwise-le-iff by (simp add: y'-def)
  have  $x^!i \leq y^!i$ 
    using  $\langle x \leq y \rangle$   $\langle i < \text{card } A \rangle$  assms by (simp add: pointwise-le-iff-nth)
  then have  $y^!i < y^!i$ 
    using  $\langle i < \text{card } A \rangle$  assms lenx' xi pointwise-le-iff by (simp add: y'-def plus-list-def minus-list-def)
  moreover have  $\text{take } i \ y' = \text{take } i \ y$ 
proof (intro nth-equalityI)
  show  $\text{length } (\text{take } i \ y') = \text{length } (\text{take } i \ y)$ 
    by (simp add: leny')
  show  $\bigwedge k. k < \text{length } (\text{take } i \ y') \implies \text{take } i \ y'^!k = \text{take } i \ y^!k$ 
    using takex' by (simp add: y'-def plus-list-def minus-list-def take-map take-zip)
qed

```

ultimately have $y' < y$
using $leny' \langle i < card\ A \rangle\ assms\ pointwise-le-iff$
by $(auto\ simp:\ list-less-def\ lenlex-def\ lexord-lex\ lexord-take-index-conv)$
moreover have $\sigma\ y' = \sigma\ y$
using $assms$
by $(simp\ add:\ \sigma x'\ lenx'\ leny\ pointwise-le-plus\ sum-list-minus\ sum-list-plus\ y'-def)$
moreover have $\alpha\ y' = \alpha\ y$
using $assms\ lenx'\ \alpha x'\ leny$
by $(fastforce\ simp:\ y'-def\ pointwise-le-plus\ alpha-minus\ alpha-plus\ local.associative)$
ultimately show $?thesis$
using $leny'\ useless-def$ **by** $blast$
qed

inductive-set $minimal-elements$ **for** U
where $\llbracket x \in U; \bigwedge y. y \in U \implies \neg y \triangleleft x \rrbracket \implies x \in minimal-elements\ U$

lemma $pointwise-less-imp-\sigma$:
assumes $xs \triangleleft ys$ **shows** $\sigma\ xs < \sigma\ ys$
proof –
have $eq:\ length\ ys = length\ xs$ **and** $xs \trianglelefteq ys$
using $assms$ **by** $(auto\ simp:\ pointwise-le-iff\ pointwise-less-iff)$
have $\forall k < length\ xs. xs!k \leq ys!k$
using $\langle xs \trianglelefteq ys \rangle\ list-all2-nthD\ pointwise-le-def$ **by** $auto$
moreover have $\exists k < length\ xs. xs!k < ys!k$
using $assms\ pointwise-less-iff2$ **by** $force$
ultimately show $?thesis$
by $(force\ simp:\ eq\ sum-list-sum-nth\ intro:\ sum-strict-mono-ex1)$
qed

lemma $wf-measure-\sigma$: $wf\ (inv-image\ less-than\ \sigma)$
by $blast$

lemma WFP : $wfP\ (\triangleleft)$
by $(auto\ simp:\ wfP-def\ pointwise-less-imp-\sigma\ intro:\ wf-subset\ [OF\ wf-measure-\sigma])$

The following is a direct corollary of the above lemma, i.e. a corollary of Lemma 2.7 in Gowers's notes.

corollary $useless-iff$:
assumes $length\ x = card\ A$
shows $useless\ x \iff (\exists x' \in minimal-elements\ (Collect\ useless).\ x' \trianglelefteq x)$ **(is**
 $-=?R)$

proof
assume $useless\ x$
obtain z **where** $z:\ useless\ z\ z \trianglelefteq x$ **and** $zmin:\ \bigwedge y. y \triangleleft z \implies y \trianglelefteq x \implies useful\ y$
using $wfE-min\ [to-pred,\ \mathbf{where}\ Q = \{z.\ useless\ z \wedge z \trianglelefteq x\},\ OF\ WFP]$
by $(metis\ (no-types,\ lifting)\ \langle useless\ x \rangle\ mem-Collect-eq\ pointwise-le-refl)$

```

then show ?R
  by (smt (verit) mem-Collect-eq minimal-elements.intros pointwise-le-trans point-
wise-less-def)
next
  assume ?R
  with useless-leq-useless minimal-elements.cases show useless x
    by (metis assms mem-Collect-eq pointwise-le-iff)
qed

```

2.8 The set of minimal elements of a set of r -tuples is finite

The following general finiteness claim corresponds to Lemma 2.8 in Gowers's notes and is key to the main proof.

lemma *minimal-elements-set-tuples-finite:*

assumes $Ur: \bigwedge x. x \in U \implies \text{length } x = r$

shows *finite (minimal-elements U)*

using *assms*

proof (*induction r arbitrary: U*)

case 0

then have $U \subseteq \{\emptyset\}$

by *auto*

then show ?case

by (metis *finite.simps minimal-elements.cases finite-subset subset-eq*)

next

case (*Suc r*)

show ?case

proof (*cases U={}*)

case *True*

with *Suc.IH* **show** ?thesis **by** *blast*

next

case *False*

then obtain u **where** $u: u \in U$ **and** $zmin: \bigwedge y. y \triangleleft u \implies y \notin U$

using *wfE-min [to-pred, where Q = U, OF WFP]* **by** *blast*

define V **where** $V = \{v \in U. \neg u \trianglelefteq v\}$

define VF **where** $VF \equiv \lambda i t. \{v \in V. v!i = t\}$

have [*simp*]: $\text{length } v = \text{Suc } r$ **if** $v \in VF$ i t **for** v i t

using *that* **by** (*simp add: Suc.prem VF-def V-def*)

have *: $\exists i \leq r. v!i < u!i$ **if** $v \in V$ **for** v

using *that* u *Suc.prem*

by (*force simp: V-def pointwise-le-iff-nth not-le less-Suc-eq-le*)

with u **have** *minimal-elements* $U \leq \text{insert } u (\bigcup_{i \leq r}. \bigcup_{t < u!i}. \text{minimal-elements } (VF \ i \ t))$

by (*force simp: VF-def V-def minimal-elements.simps pointwise-less-def*)

moreover

have *finite (minimal-elements (VF i t))* **if** $i \leq r$ $t < u!i$ **for** i t

proof –

define *delete* **where** $\text{delete} \equiv \lambda v::\text{nat list}. \text{take } i \ v \ @ \ \text{drop } (\text{Suc } i) \ v$ — deletion of i

have *len-delete[simp]: length (delete u) = r* **if** $u \in VF$ i t **for** u


```

    using Suc.premis VF-def V-def ⟨i ≤ r⟩ delete-def that by auto
  have nth-delete: delete u!k = (if k < i then u!k else u!Suc k) if u ∈ VF i t k < r
for u k
  using that by (simp add: delete-def nth-append)
  have delete-le-iff [simp]: delete u ⊑ delete v ↔ u ⊑ v if u ∈ VF i t v ∈ VF
i t for u v
  proof
    assume delete u ⊑ delete v
    then have ∀j. (j < i → u!j ≤ v!j) ∧ (j < r → i ≤ j → u!Suc j ≤
v!Suc j)
      using that ⟨i ≤ r⟩
      by (force simp: pointwise-le-iff-nth nth-delete split: if-split-asm cong:
conj-cong)
    then show u ⊑ v
      using that ⟨i ≤ r⟩
      apply (simp add: pointwise-le-iff-nth VF-def)
      by (metis eq-iff le-Suc-eq less-Suc-eq-0-disj linorder-not-less)
  next
    assume u ⊑ v then show delete u ⊑ delete v
      using that by (simp add: pointwise-le-iff-nth nth-delete)
  qed
  then have delete-eq-iff: delete u = delete v ↔ u = v if u ∈ VF i t v ∈ VF
i t for u v
    by (metis that pointwise-le-antisym pointwise-le-refl)
  have delete-less-iff: delete u < delete v ↔ u < v if u ∈ VF i t v ∈ VF i t
for u v
    by (metis delete-le-iff pointwise-le-antisym pointwise-less-def that)
  have length (delete v) = r if v ∈ V for v
    using id-take-nth-drop Suc.premis V-def ⟨i ≤ r⟩ delete-def that by auto
  then have finite (minimal-elements (delete ‘ V))
    by (metis (mono-tags, lifting) Suc.IH image-iff)
  moreover have inj-on delete (minimal-elements (VF i t))
    by (simp add: delete-eq-iff inj-on-def minimal-elements.simps)
  moreover have delete ‘ (minimal-elements (VF i t)) ⊆ minimal-elements
(delete ‘ (VF i t))
    by (auto simp: delete-less-iff minimal-elements.simps)
  ultimately show ?thesis
    by (metis (mono-tags, lifting) Suc.IH image-iff inj-on-finite len-delete)
  qed
  ultimately show ?thesis
    by (force elim: finite-subset)
  qed
qed

```

2.9 Towards Lemma 2.9 in Gowers’s notes

Increasing sequences

```

fun augmentum :: nat list ⇒ nat list
  where augmentum [] = []

```

| $augmentum (n\#ns) = n \# map ((+)n) (augmentum ns)$

definition *dementum*:: $nat\ list \Rightarrow nat\ list$
where $dementum\ xs \equiv xs - (0\#xs)$

lemma *dementum-Nil* [simp]: $dementum\ [] = []$
by (*simp add: dementum-def*)

lemma *zero-notin-augmentum* [simp]: $0 \notin set\ ns \Longrightarrow 0 \notin set\ (augmentum\ ns)$
by (*induction ns*) *auto*

lemma *length-augmentum* [simp]: $length\ (augmentum\ xs) = length\ xs$
by (*induction xs*) *auto*

lemma *sorted-augmentum* [simp]: $0 \notin set\ ns \Longrightarrow sorted\ (augmentum\ ns)$
by (*induction ns*) *auto*

lemma *distinct-augmentum* [simp]: $0 \notin set\ ns \Longrightarrow distinct\ (augmentum\ ns)$
by (*induction ns*) (*simp-all add: image-iff*)

lemma *augmentum-subset-sum-list*: $set\ (augmentum\ ns) \subseteq \{..\sigma\ ns\}$
by (*induction ns*) *auto*

lemma *sum-list-augmentum*: $\sigma\ ns \in set\ (augmentum\ ns) \longleftrightarrow length\ ns > 0$
by (*induction ns*) *auto*

lemma *length-dementum* [simp]: $length\ (dementum\ xs) = length\ xs$
by (*simp add: dementum-def*)

lemma *sorted-imp-pointwise*:
assumes $sorted\ (xs@[n])$
shows $0 \# xs \trianglelefteq xs\ @ [n]$
using *assms*
by (*simp add: pointwise-le-iff-nth nth-Cons' nth-append sorted-append sorted-wrt-append sorted-wrt-nth-less*)

lemma *sum-list-dementum*:
assumes $sorted\ (xs@[n])$
shows $\sigma\ (dementum\ (xs@[n])) = n$
proof –
have $dementum\ (xs@[n]) = (xs@[n]) - (0\#xs)$
by (*rule nth-equalityI; simp add: nth-append dementum-def nth-Cons'*)
then show *?thesis*
by (*simp add: sum-list-minus sorted-imp-pointwise assms*)
qed

lemma *augmentum-cancel*: $map\ ((+)k)\ (augmentum\ ns) - (k\ \# map\ ((+)k)\ (augmentum\ ns)) = ns$
proof (*induction ns arbitrary: k*)

```

    case Nil
  then show ?case
    by simp
next
  case (Cons n ns)
  have (+) k ∘ (+) n = (+) (k+n) by auto
  then show ?case
    by (simp add: minus-Cons Cons)
qed

lemma dementum-augmentum [simp]:
  assumes 0 ∉ set ns
  shows (dementum ∘ sorted-list-of-set) ((set ∘ augmentum) ns) = ns (is ?L ns =
-)
  using assms augmentum-cancel [of 0]
  by (simp add: dementum-def map-idI sorted-list-of-set.idem-if-sorted-distinct)

lemma dementum-nonzero:
  assumes ns: sorted-wrt (<) ns and 0: 0 ∉ set ns
  shows 0 ∉ set (dementum ns)
  unfolding dementum-def minus-list-def
  using sorted-wrt-nth-less [OF ns] 0
  by (auto simp: in-set-conv-nth image-iff set-zip nth-Cons' dest: leD)

lemma nth-augmentum [simp]: i < length ns ⇒ augmentum ns!i = (∑ j ≤ i. ns!j)
proof (induction ns arbitrary: i)
  case Nil
  then show ?case
    by simp
next
  case (Cons a ns)
  show ?case
  proof (cases i=0)
    case False
    then have augmentum (a # ns)!i = a + sum (!) ns {..i-1}
      using Cons.IH Cons.prem by auto
    also have ... = a + (∑ j ∈ {0 <.. i}. ns!(j-1))
      using sum.reindex [of Suc {..i - Suc 0} λj. ns!(j-1), symmetric] False
      by (simp add: image-Suc-atMost atLeastSucAtMost-greaterThanAtMost del:
sum.cl-ivl-Suc)
    also have ... = (∑ j = 0..i. if j=0 then a else ns!(j-1))
      by (simp add: sum.head)
    also have ... = sum (!) (a # ns) {..i}
      by (simp add: nth-Cons' atMost-atLeast0)
    finally show ?thesis .
  qed auto
qed

```

lemma augmentum-dementum [simp]:

```

assumes 0 ∉ set ns sorted ns
shows augmentum (dementum ns) = ns
proof (rule nth-equalityI)
  fix i
  assume i < length (augmentum (dementum ns))
  then have i: i < length ns
    by simp
  show augmentum (dementum ns)!i = ns!i
  proof (cases i=0)
    case True
      then show ?thesis
        using nth-augmentum dementum-def i by auto
    next
      case False
        have ns-le: ∧j. [0 < j; j ≤ i] ⇒ ns ! (j - Suc 0) ≤ ns ! j
          using ‹sorted ns› i by (simp add: sorted-iff-nth-mono)
        have augmentum (dementum ns)!i = (∑ j≤i. ns!j - (if j = 0 then 0 else
ns!(j-1)))
          using i by (simp add: dementum-def nth-Cons^)
        also have ... = (∑ j=0..i. if j = 0 then ns!0 else ns!j - ns!(j-1))
          by (smt (verit, del-insts) diff-zero sum.cong atMost-atLeast0)
        also have ... = ns!0 + (∑ j∈{0<..i}. ns!j - ns!(j-1))
          by (simp add: sum.head)
        also have ... = ns!0 + ((∑ j∈{0<..i}. ns!j) - (∑ j∈{0<..i}. ns!(j-1)))
          by (auto simp: ns-le intro: sum-subtractf-nat)
        also have ... = ns!0 + (∑ j∈{0<..i}. ns!j) - (∑ j∈{0<..i}. ns!(j-1))
          proof -
            have (∑ j∈{0<..i}. ns ! (j - 1)) ≤ sum (!) ns {0<..i}
              by (metis One-nat-def greaterThanAtMost-iff ns-le sum-mono)
            then show ?thesis by simp
        qed
        also have ... = ns!0 + (∑ j∈{0<..i}. ns!j) - (∑ j≤i-Suc 0. ns!j)
          using sum.reindex [of Suc {..i - Suc 0} λj. ns!(j-1), symmetric] False
          by (simp add: image-Suc-atMost atLeastSucAtMost-greaterThanAtMost)
        also have ... = (∑ j=0..i. ns!j) - (∑ j≤i-Suc 0. ns!j)
          by (simp add: sum.head [of 0 i])
        also have ... = (∑ j=0..i-Suc 0. ns!j) + ns!i - (∑ j≤i-Suc 0. ns!j)
          by (metis False Suc-pred less-Suc0 not-less-eq sum.atLeast0-atMost-Suc)
        also have ... = ns!i
          by (simp add: atLeast0AtMost)
        finally show augmentum (dementum ns)!i = ns!i .
    qed
qed auto

```

The following lemma corresponds to Lemma 2.9 in Gowers's notes. The proof involves introducing bijective maps between r-tuples that fulfill certain properties/r-tuples and subsets of naturals, so as to show the cardinality claim.

lemma bound-sum-list-card:

assumes $r > 0$ **and** $n: n \geq \sigma x'$ **and** $lenx'$: $length\ x' = r$
defines $S \equiv \{x. x' \trianglelefteq x \wedge \sigma x = n\}$
shows $card\ S = (n - \sigma x' + r - 1)$ *choose* $(r-1)$
proof –
define m **where** $m \equiv n - \sigma x'$
define f **where** $f \equiv \lambda x::nat\ list. x - x'$
have f : *bij-betw* $f\ S$ (*length-sum-set* $r\ m$)
proof (*intro* *bij-betw-imageI*)
show *inj-on* $f\ S$
using *pairwise-minus-cancel* **by** (*force simp: S-def f-def inj-on-def*)
have $\bigwedge x. x \in S \implies f\ x \in length-sum-set\ r\ m$
by (*simp add: S-def f-def length-sum-set-def lenx' m-def pointwise-le-iff sum-list-minus*)
moreover **have** $x \in f\ 'S$ **if** $x \in length-sum-set\ r\ m$ **for** x
proof
have x [*simp*]: $length\ x = r\ \sigma x = m$
using *that* **by** (*auto simp: length-sum-set-def*)
have $x = x' + x - x'$
by (*rule nth-equalityI; simp add: lenx'*)
then **show** $x = f\ (x' + x)$
unfolding *f-def* **by** *fastforce*
have $x' \trianglelefteq x' + x$
by (*simp add: lenx' pointwise-le-plus*)
moreover **have** $\sigma (x' + x) = n$
by (*simp add: lenx' m-def n sum-list-plus*)
ultimately **show** $x' + x \in S$
using *S-def* **by** *blast*
qed
ultimately **show** $f\ 'S = length-sum-set\ r\ m$ **by** *auto*
qed
define g **where** $g \equiv \lambda x::nat\ list. map\ Suc\ x$
define g' **where** $g' \equiv \lambda x::nat\ list. x - replicate\ (length\ x)\ 1$
define T **where** $T \equiv length-sum-set\ r\ (m+r) \cap lists(-\{0\})$
have g : *bij-betw* g (*length-sum-set* $r\ m$) T
proof (*intro* *bij-betw-imageI*)
show *inj-on* g (*length-sum-set* $r\ m$)
by (*auto simp: g-def inj-on-def*)
have $\bigwedge x. x \in length-sum-set\ r\ m \implies g\ x \in T$
by (*auto simp: g-def length-sum-set-def sum-list-Suc T-def*)
moreover **have** $x \in g\ 'length-sum-set\ r\ m$ **if** $x \in T$ **for** x
proof
have [*simp*]: $length\ x = r$
using *length-sum-set-def* **that** *T-def* **by** *auto*
have $r1$ - x : $replicate\ r\ (Suc\ 0) \trianglelefteq x$
using *that* **unfolding** *T-def pointwise-le-iff-nth*
by (*simp add: lists-def in-listsp-conv-set Suc-leI*)
show $x = g\ (g'\ x)$
using *that* **by** (*intro nth-equalityI*) (*auto simp: g-def g'-def T-def*)
show $g'\ x \in length-sum-set\ r\ m$

```

    using that T-def by (simp add: g'-def r1-x sum-list-minus length-sum-set-def
sum-list-replicate)
  qed
  ultimately show g ' (length-sum-set r m) = T by auto
  qed
  define U where U ≡ (insert (m+r)) ' finsets {0<..

```

```

    by simp
  show (dementum ∘ sorted-list-of-set) u ∈ T
  apply (simp add: T-def length-sum-set-def)
  using sum-list-dementum [OF ssN] sortd-wrt-u 0 by (force simp: so-u
dementum-nonzero)+
  qed
  ultimately show (set ∘ augmentum) ‘ T = U by auto
  qed
  obtain φ where bij-betw φ S U
  by (meson bij-betw-trans f g h)
  moreover have card U = (n - σ x' + r - 1) choose (r - 1)
  proof -
  have inj-on (insert (m + r)) (finsets {0 <..< m + r} (r - Suc 0))
  by (simp add: inj-on-def finsets-def subset-iff) (meson insert-ident order-less-le)
  then have card U = card (finsets {0 <..< m + r} (r - 1))
  unfolding U-def by (simp add: card-image)
  also have ... = (n - σ x' + r - 1) choose (r - 1)
  by (simp add: card-finsets m-def)
  finally show ?thesis .
  qed
  ultimately show ?thesis
  by (metis bij-betw-same-card)
  qed

```

2.10 Towards the main theorem

lemma *extend-tuple*:

```

  assumes σ xs ≤ n length xs ≠ 0
  obtains ys where σ ys = n xs ≤ ys
  proof -
  obtain x xs' where xs: xs = x#xs'
  using assms list.exhaust by auto
  define y where y ≡ x + n - σ xs
  show thesis
  proof
  show σ (y#xs') = n
  using assms xs y-def by auto
  show xs ≤ y#xs'
  using assms y-def pointwise-le-def xs by auto
  qed
  qed

```

lemma *extend-preserving*:

```

  assumes σ xs ≤ n length xs > 1 i < length xs
  obtains ys where σ ys = n xs ≤ ys ys!i = xs!i
  proof -
  define j where j ≡ Suc i mod length xs
  define xs1 where xs1 = take j xs
  define xs2 where xs2 = drop (Suc j) xs

```

```

define  $x$  where  $x = xs!j$ 
have  $xs: xs = xs1 @ [x] @ xs2$ 
  using  $assms$ 
  apply ( $simp$   $add: Cons-nth-drop-Suc$   $assms$   $x-def$   $xs1-def$   $xs2-def$   $j-def$ )
  by ( $meson$   $Suc-lessD$   $id-take-nth-drop$   $mod-less-divisor$ )
define  $y$  where  $y \equiv x + n - \sigma xs$ 
define  $ys$  where  $ys \equiv xs1 @ [y] @ xs2$ 
have  $x \leq y$ 
  using  $assms$   $y-def$  by  $linarith$ 
show  $thesis$ 
proof
  show  $\sigma ys = n$ 
    using  $assms(1)$   $xs$   $y-def$   $ys-def$  by  $auto$ 
  show  $xs \triangleleft ys$ 
    using  $xs$   $ys-def$   $\langle x \leq y \rangle$   $pointwise-append-le-iff$   $pointwise-le-def$  by  $fastforce$ 
  have  $length\ xs1 \neq i$ 
    using  $assms$  by ( $simp$   $add: xs1-def$   $j-def$   $min-def$   $mod-Suc$ )
  then show  $ys!i = xs!i$ 
    by ( $auto$   $simp: ys-def$   $xs$   $nth-append$   $nth-Cons'$ )
qed
qed

```

The proof of the main theorem will make use of the inclusion-exclusion formula, in addition to the previously shown results.

theorem *Khovanskii*:

```

assumes  $card\ A > 1$ 
defines  $f \equiv \lambda n. card(sumset-iterated\ A\ n)$ 
obtains  $N\ p$  where  $real-polynomial-function\ p \wedge n. n \geq N \implies real\ (f\ n) = p$ 
( $real\ n$ )

```

proof –

```

define  $r$  where  $r \equiv card\ A$ 
define  $C$  where  $C \equiv \lambda n\ x'. \{x. x' \leq x \wedge \sigma x = n\}$ 
define  $X$  where  $X \equiv minimal-elements\ \{x. useless\ x \wedge length\ x = r\}$ 
have  $r > 1\ r \neq 0$ 
  using  $assms$   $r-def$  by  $auto$ 
have  $Csub: C\ n\ x' \subseteq length-sum-set\ (length\ x')\ n$  for  $n\ x'$ 
  by ( $auto$   $simp: C-def$   $length-sum-set-def$   $pointwise-le-iff$ )
then have  $finC: finite\ (C\ n\ x')$  for  $n\ x'$ 
  by ( $meson$   $finite-length-sum-set$   $finite-subset$ )
have  $finite\ X$ 
  using  $minimal-elements-set-tuples-finite$   $X-def$  by  $force$ 
then have  $max-X: \bigwedge x'. x' \in X \implies \sigma\ x' \leq \sigma\ (max-pointwise\ r\ X)$ 
  using  $X-def$   $max-pointwise-ge$   $minimal-elements.simps$   $pointwise-le-imp-\sigma$  by
 $force$ 
let  $?z0 = replicate\ r\ 0$ 
have  $Cn0: C\ n\ ?z0 = length-sum-set\ r\ n$  for  $n$ 
  by ( $auto$   $simp: C-def$   $length-sum-set-def$ )
then obtain  $p0$  where  $pf-p0: real-polynomial-function\ p0$  and  $p0: \bigwedge n. n > 0$ 
 $\implies p0\ (real\ n) = real\ (card\ (C\ n\ ?z0))$ 

```



```

  by (metis real-polynomial-function-length-sum-set)
  obtain q where pf-q: real-polynomial-function q and q:  $\bigwedge x. q\ x = x$  gchoose
  (r-1)
  using real-polynomial-function-gchoose by metis
  define p where p  $\equiv \lambda x::real. p0\ x - (\sum Y \mid Y \subseteq X \wedge Y \neq \{\}. (-1) \wedge (card\ Y + 1) * q((x - real(\sigma (max-pointwise\ r\ Y)) + real\ r - 1)))$ 
  show thesis
  proof
    note pf-q' = real-polynomial-function-compose [OF - pf-q, unfolded o-def]
    note pf-intros = real-polynomial-function-sum real-polynomial-function-diff
    real-polynomial-function.intros
    show real-polynomial-function p
      unfolding p-def using  $\langle finite\ X \rangle$  by (intro pf-p0 pf-q' pf-intros | force)+
    next
      fix n
      assume n  $\geq max\ 1\ (\sigma (max-pointwise\ r\ X))$ 
      then have nlarge: n  $\geq \sigma (max-pointwise\ r\ X)$  and n > 0
      by auto
      define U where U  $\equiv \lambda n. length-sum-set\ r\ n \cap \{x. useful\ x\}$ 
      have 2: (length-sum-set r n  $\cap \{x. useless\ x\}$ ) =  $(\bigcup x' \in X. C\ n\ x')$ 
      unfolding C-def X-def length-sum-set-def r-def
      using useless-leq-useless by (force simp: minimal-elements.simps pointwise-le-iff
      useless-iff)
      define SUM1 where SUM1  $\equiv \sum I \mid I \subseteq C\ n\ ' X \wedge I \neq \{\}. (-1) \wedge (card\ I + 1) * int (card (\bigcap I))$ 
      define SUM2 where SUM2  $\equiv \sum Y \mid Y \subseteq X \wedge Y \neq \{\}. (-1) \wedge (card\ Y + 1) * int (card (\bigcap (C\ n\ ' Y)))$ 
      have SUM1-card: card(length-sum-set r n  $\cap \{x. useless\ x\}$ ) = nat SUM1
      unfolding SUM1-def using  $\langle finite\ X \rangle$  by (simp add: finC 2 card-UNION)
      have SUM1  $\geq 0$ 
      unfolding SUM1-def using card-UNION-nonneg finC  $\langle finite\ X \rangle$  by auto
      have C-empty-iff: C n x' =  $\{\} \iff \sigma\ x' > n$  if length x'  $\neq 0$  for x'
      by (simp add: set-eq-iff C-def) (meson extend-tuple linorder-not-le point-
      wise-le-imp- $\sigma$  that)
      have C-eq-1: C n x' =  $\{[n]\}$  if  $\sigma\ x' \leq n$  length x' = 1 for x'
      using that by (auto simp: C-def length-Suc-conv pointwise-le-def elim!:
      list.rel-cases)
      have n-ge-X:  $\sigma\ x \leq n$  if  $x \in X$  for x
      by (meson le-trans max-X nlarge that)
      have len-X-r:  $x \in X \implies length\ x = r$  for x
      by (auto simp: X-def minimal-elements.simps)

      have min-pointwise r (C n x') = x' if  $r > 1$  x'  $\in X$  for x'
    proof (rule pointwise-le-antisym)
      have [simp]: length x' = r  $\sigma\ x' \leq n$ 
      using X-def minimal-elements.cases that(2) n-ge-X by auto
      have [simp]: length (min-pointwise r (C n x')) = r
      by (simp add: min-pointwise-def)
      show min-pointwise r (C n x')  $\preceq x'$ 
    end
  end

```

```

proof (clarsimp simp add: pointwise-le-iff-nth)
  fix i
  assume i < r
  then obtain y where  $\sigma y = n \wedge x' \sqsubseteq y \wedge y!i \leq x'!i$ 
    by (metis extend-preserving ⟨1 < r⟩ ⟨length x' = r⟩ ⟨x' ∈ X⟩ order.refl
n-ge-X)
  then have  $\exists y \in C \ n \ x'. \ y!i \leq x'!i$ 
    using C-def by blast
  with ⟨i < r⟩ show min-pointwise r (C n x')!i ≤ x'!i
    by (simp add: min-pointwise-def Min-le-iff finC C-empty-iff leD)
qed
have  $x' \sqsubseteq \text{min-pointwise } r \ (C \ n \ x')$  if  $\sigma x' \leq n$  length  $x' = r$  for  $x'$ 
  by (smt (verit, del-Insts) C-def C-empty-iff ⟨r ≠ 0⟩ finC leD mem-Collect-eq
min-pointwise-ge-iff pointwise-le-iff that)
  then show  $x' \sqsubseteq \text{min-pointwise } r \ (C \ n \ x')$ 
    using X-def minimal-elements.cases that by force
qed
then have inj-C: inj-on (C n) X
  by (smt (verit, best) inj-onI mem-Collect-eq ⟨r > 1⟩)
have inj-on-imageC: inj-on (image (C n)) (Pow X - {{{}})
  by (simp add: inj-C inj-on-diff inj-on-image-Pow)

have  $\text{Pow } (C \ n \ 'X) - \{\{\}\} \subseteq (\text{image } (C \ n)) \ '(\text{Pow } X - \{\{\}\})$ 
  by (metis Pow-empty image-Pow-surj image-diff-subset image-empty)
then have  $(\text{image } (C \ n)) \ '(\text{Pow } X - \{\{\}\}) = \text{Pow } (C \ n \ 'X) - \{\{\}\}$ 
  by blast
then have  $\text{SUM1} = \text{sum } (\lambda I. (-1) ^ (\text{card } I + 1) * \text{int } (\text{card } (\bigcap I))) ((\text{image } (C \ n)) \ '(\text{Pow } X - \{\{\}\}))$ 
  unfolding SUM1-def by (auto intro: sum.cong)
  also have  $\dots = \text{sum } ((\lambda I. (-1) ^ (\text{card } I + 1) * \text{int } (\text{card } (\bigcap I))) \circ (\text{image } (C \ n))) (\text{Pow } X - \{\{\}\})$ 
  by (simp add: sum.reindex inj-on-imageC)
  also have  $\dots = \text{SUM2}$ 
  unfolding SUM2-def using subset-inj-on [OF inj-C] by (force simp: card-image
intro: sum.cong)
  finally have  $\text{SUM1} = \text{SUM2}$  .

have  $\text{length-sum-set } r \ n = (\text{length-sum-set } r \ n \cap \{x. \text{useful } x\}) \cup (\text{length-sum-set } r \ n \cap \{x. \text{useless } x\})$ 
  by auto
then have  $\text{card } (\text{length-sum-set } r \ n) =$ 
   $\text{card } (\text{length-sum-set } r \ n \cap \{x. \text{useful } x\}) +$ 
   $\text{card } (\text{length-sum-set } r \ n \cap \text{Collect } \text{useless})$ 
  by (simp add: finite-length-sum-set disjnt-iff flip: card-Un-disjnt)
moreover have  $C \ n \ ?z0 = \text{length-sum-set } r \ n$ 
  by (auto simp: C-def length-sum-set-def)
ultimately have  $\text{card } (C \ n \ ?z0) = \text{card } (U \ n) + \text{nat } \text{SUM2}$ 
  by (simp add: U-def flip: ⟨SUM1 = SUM2⟩ SUM1-card)
then have  $\text{SUM2-le: } \text{nat } \text{SUM2} \leq \text{card } (C \ n \ ?z0)$ 

```

by *arith*
 have σ -max-pointwise-le: $\bigwedge Y. [Y \subseteq X; Y \neq \{\}] \implies \sigma (\text{max-pointwise } r Y)$
 $\leq n$
 by (*meson* $\langle \text{finite } X \rangle$ *le-trans max-pointwise-mono nlarge pointwise-le-imp- σ*)

have *card-C-max*: $\text{card } (C n (\text{max-pointwise } r Y)) =$
 $(n - \sigma (\text{max-pointwise } r Y) + r - \text{Suc } 0 \text{ choose } (r - \text{Suc } 0))$
 if $Y \subseteq X$ $Y \neq \{\}$ for Y
proof -
 have [*simp*]: $\text{length } (\text{max-pointwise } r Y) = r$
 by (*simp add: max-pointwise-def*)
 then show *?thesis*
 using $\langle r \neq 0 \rangle$ that *C-def* by (*simp add: bound-sum-list-card [of r]*)
 σ -max-pointwise-le)
qed

define *SUM3* where $SUM3 \equiv (\sum Y \mid Y \subseteq X \wedge Y \neq \{\}.$
 $- ((- 1) \wedge (\text{card } Y) * ((n - \sigma (\text{max-pointwise } r Y) + r - 1 \text{ choose } (r -$
 $1))))))$
 have $\bigcap (C n \text{ ' } Y) = C n (\text{max-pointwise } r Y)$ if $Y \subseteq X$ $Y \neq \{\}$ for Y
proof
 show $\bigcap (C n \text{ ' } Y) \subseteq C n (\text{max-pointwise } r Y)$
 unfolding *C-def*
proof *clarsimp*
fix x
assume $\forall y \in Y. y \triangleleft x \wedge \sigma x = n$
moreover have *finite* Y
 using $\langle \text{finite } X \rangle$ *infinite-super* that by *blast*
moreover have $\bigwedge u. u \in Y \implies \text{length } u = r$
 using *len-X-r* that by *blast*
ultimately show $\text{max-pointwise } r Y \triangleleft x \wedge \sigma x = n$
 by (*smt* (*verit, del-insts*) *all-not-in-conv max-pointwise-le-iff point-*
wise-le-iff-nth that(2))
qed
next
 show $C n (\text{max-pointwise } r Y) \subseteq \bigcap (C n \text{ ' } Y)$
 apply (*clarsimp simp: C-def*)
 by (*metis* $\langle \text{finite } X \rangle$ *finite-subset len-X-r max-pointwise-ge pointwise-le-trans*
subsetD that(1))
qed
then have $SUM2 = SUM3$
 by (*simp add: SUM2-def SUM3-def card-C-max*)
have $U n = C n \text{ ?z0} - (\text{length-sum-set } r n \cap \{x. \text{useless } x\})$
 by (*auto simp: U-def C-def length-sum-set-def*)
then have $\text{card } (U n) = \text{card } (C n \text{ ?z0}) - \text{card}(\text{length-sum-set } r n \cap \{x. \text{useless}$
 $x\})$
 using *finite-length-sum-set*
 by (*simp add: C-def Collect-mono-iff inf.coboundedI1 length-sum-set-def flip:*
card-Diff-subset)

```

then have card-U-eq-diff:  $\text{card } (U \ n) = \text{card } (C \ n \ ?z0) - \text{nat } SUM1$ 
using SUM1-card by presburger
have  $SUM3 \geq 0$ 
using  $\langle 0 \leq SUM1 \rangle \langle SUM1 = SUM2 \rangle \langle SUM2 = SUM3 \rangle$  by blast
have **:  $\bigwedge Y. \llbracket Y \subseteq X; Y \neq \{\} \rrbracket \implies \text{Suc } (\sigma \ (\text{max-pointwise } r \ Y)) \leq n + r$ 
by (metis  $\langle 1 < r \rangle$   $\sigma$ -max-pointwise-le add.commute add-le-mono less-or-eq-imp-le
plus-1-eq-Suc)
have  $\text{real } (f \ n) = \text{card } (U \ n)$ 
unfolding f-def r-def U-def length-sum-set-def
using card-sumset-iterated-length-sum-set-useful length-sum-set-def by pres-
burger
also have  $\dots = \text{card } (C \ n \ ?z0) - \text{nat } SUM3$ 
using card-U-eq-diff  $\langle SUM1 = SUM2 \rangle \langle SUM2 = SUM3 \rangle$  by presburger
also have  $\dots = \text{real } (\text{card } (C \ n \ (\text{replicate } r \ 0))) - \text{real } (\text{nat } SUM3)$ 
using SUM2-le  $\langle SUM2 = SUM3 \rangle$  of-nat-diff by blast
also have  $\dots = p \ (\text{real } n)$ 
using  $\langle 1 < r \rangle \langle n > 0 \rangle$ 
apply (simp add: p-def p0 q  $\langle SUM3 \geq 0 \rangle$ )
apply (simp add: SUM3-def binomial-gbinomial of-nat-diff  $\sigma$ -max-pointwise-le
algebra-simps **)
done
finally show  $\text{real } (f \ n) = p \ (\text{real } n) .$ 
qed
qed

end

end

```

References

- [1] W. T. Gowers. Introduction to additive combinatorics. Lecture notes, University of Cambridge, 2022.
- [2] A. G. Khovanskii. Newton polyhedron, Hilbert polynomial, and sums of finite sets. *Functional Analysis and Its Applications*, 26(4):276–281, 1992.
- [3] A. G. Khovanskii. Sums of finite sets, orbits of commutative semigroups, and Hilbert functions. *Functional Analysis and Its Applications*, 29(2):102–112, 1995.
- [4] M. B. Nathanson and I. Z. Ruzsa. Polynomial growth of sumsets in abelian semigroups. *Journal de Théorie des Nombres de Bordeaux*, 14(2):553–560, 2002.
- [5] I. Z. Ruzsa. Sumsets and structure. Lecture notes, Institute of Mathematics, Budapest.