

Alon’s Combinatorial Nullstellensatz

Arthur F. Ramos David Barros Hulak Ruy J. G. B. de Queiroz

May 10, 2026

Abstract

This entry formalizes Alon’s Combinatorial Nullstellensatz for sparse multivariate polynomials over fields. The proof derives the coefficient formula from univariate Lagrange interpolation and uses it to obtain the standard nonvanishing conclusion over finite grids. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

1 Overview

Alon’s Combinatorial Nullstellensatz is a polynomial-method theorem about a distinguished coefficient of a multivariate polynomial. In the form formalized here, if all monomials have total degree bounded by $d_1 + \dots + d_n$ and the coefficient of $X_1^{d_1} \dots X_n^{d_n}$ is nonzero, then the polynomial has a nonzero value on every product grid whose i -th side has more than d_i elements.

The Isabelle/HOL development represents multivariate polynomials sparsely, as finite-support coefficient functions from exponent lists to field elements. It first proves the necessary univariate Lagrange interpolation identities, then sums polynomial evaluations over a finite grid with the corresponding Lagrange weights. All monomials except the distinguished exponent vector cancel, yielding the coefficient formula and the nonvanishing theorem.

2 Sources

The entry follows Alon’s original paper [1]. The formal proof uses only Isabelle/HOL’s standard computational algebra library as its mathematical base.

Contents

1 Overview	1
2 Sources	1
3 Alon’s Combinatorial Nullstellensatz	1
3.1 Univariate interpolation	2
3.2 Sparse multivariate polynomials	4
theory <i>Combinatorial-Nullstellensatz</i>	
imports <i>HOL-Computational-Algebra.Polynomial</i>	
begin	

3 Alon’s Combinatorial Nullstellensatz

Alon’s Combinatorial Nullstellensatz [1] is a polynomial method theorem: if the coefficient of a distinguished monomial of total degree $d_1 + \dots + d_n$ is nonzero, then the polynomial cannot vanish on every point of any grid whose i -th side has more than d_i elements.

The development below proves this statement for sparse multivariate polynomials over arbitrary fields. A polynomial is represented by a finite-support coefficient function from exponent lists to coefficients. This keeps the formalization independent of a particular multivariate polynomial library while still matching the usual mathematical statement.

3.1 Univariate interpolation

The proof starts with a small amount of univariate interpolation. For a finite set S and a point $x \in S$, the following denominator and basis polynomial are the standard Lagrange factors.

definition *lagrange-denom* :: 'a::field set \Rightarrow 'a \Rightarrow 'a **where**
lagrange-denom S $x = (\prod_{y \in S - \{x\}} x - y)$

definition *lagrange-basis* :: 'a::field set \Rightarrow 'a \Rightarrow 'a **poly where**
lagrange-basis S $x =$
 $smult$ (*inverse* (*lagrange-denom* S x))
 $(\prod_{y \in S - \{x\}} [: - y, 1:])$

lemma *lagrange-denom-nonzero*:
assumes *finite* S $x \in S$
shows *lagrange-denom* S $x \neq 0$
using *assms* **by** (*auto simp: lagrange-denom-def*)

lemma *poly-lagrange-basis*:
assumes *finite* S $x \in S$ $z \in S$
shows *poly* (*lagrange-basis* S x) $z =$ (*if* $z = x$ *then* 1 *else* 0)
proof (*cases* $z = x$)
case *True*
have *poly* $(\prod_{y \in S - \{x\}} [: - y, 1:])$ $x =$ *lagrange-denom* S x
by (*simp add: lagrange-denom-def poly-prod*)
with *assms True lagrange-denom-nonzero*[*of* S x] **show** *?thesis*
by (*simp add: lagrange-basis-def*)

next
case *False*
with *assms* **have** $z \in S - \{x\}$
by *simp*
then have *poly* $(\prod_{y \in S - \{x\}} [: - y, 1:])$ $z = 0$
using *assms* **by** (*simp add: poly-prod*)
with *False* **show** *?thesis*
by (*simp add: lagrange-basis-def*)

qed

lemma *degree-lagrange-basis-le*:
assumes *finite* S $x \in S$
shows *degree* (*lagrange-basis* S x) \leq *card* $S - 1$
proof -
have *degree* $(\prod_{y \in S - \{x\}} [: - y, 1:] :: 'a$ *poly*)
 $=$ $(\sum_{y \in S - \{x\}} \text{degree } ([: - y, 1:] :: 'a::field$ *poly*))
using *assms* **by** (*intro degree-prod-sum-eq*) *auto*
also have $\dots \leq$ *card* $S - 1$
using *assms* **by** *auto*
finally show *?thesis*
by (*simp add: lagrange-basis-def degree-smult-le dual-order.trans*)

qed

lemma *lagrange-interpolation*:
fixes $P :: 'a::\{field,ring-no-zero-divisors\}$ *poly*
assumes *fin: finite* S **and** *deg: degree* $P <$ *card* S

shows $P = (\sum x \in S. \text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x))$
proof (*rule poly-eqI-degree*[**where** $A = S$])
fix z
assume $z: z \in S$
have $\text{poly } (\sum x \in S. \text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x)) \ z =$
 $(\sum x \in S. \text{poly } P \ x * \text{poly } (\text{lagrange-basis } S \ x) \ z)$
by (*simp add: poly-sum*)
also have $\dots = \text{poly } P \ z$
using *fin z*
by (*subst sum.remove*[*of S z*]) (*auto simp: poly-lagrange-basis eq-commute*)
finally show $\text{poly } P \ z = \text{poly } (\sum x \in S. \text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x)) \ z$
by *simp*
next
show $\text{card } S > \text{degree } P$
using *deg* **by** *simp*
next
have $\text{degree } (\sum x \in S. \text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x)) \leq \text{card } S - 1$
proof (*intro degree-sum-le fin*)
fix x
assume $x \in S$
then show $\text{degree } (\text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x)) \leq \text{card } S - 1$
by (*meson degree-lagrange-basis-le degree-smult-le fin order-trans*)
qed
moreover from *deg* **have** $\text{card } S > 0$
by *simp*
ultimately show $\text{card } S > \text{degree } (\sum x \in S. \text{smult } (\text{poly } P \ x) \ (\text{lagrange-basis } S \ x))$
by *linarith*
qed

lemma *coeff-lagrange-basis-top:*

assumes *fin: finite S and x: x ∈ S*
shows $\text{poly.coeff } (\text{lagrange-basis } S \ x) \ (\text{card } S - 1) =$
 $\text{inverse } (\text{lagrange-denom } S \ x)$
proof –
have *deg-prod: degree* $(\prod y \in S - \{x\}. [:- y, 1:] :: 'a::field \text{poly}) = \text{card } S - 1$
proof –
have $\text{degree } (\prod y \in S - \{x\}. [:- y, 1:] :: 'a \text{poly}) = (\sum y \in S - \{x\}. \text{degree } ([:- y, 1:] :: 'a \text{poly}))$
using *fin* **by** (*intro degree-prod-sum-eq auto*)
also have $\dots = \text{card } S - 1$
using *fin x* **by** *simp*
finally show *?thesis .*
qed
have $\text{poly.coeff } (\text{lagrange-basis } S \ x) \ (\text{card } S - 1) =$
 $\text{inverse } (\text{lagrange-denom } S \ x) * \text{poly.coeff } (\prod y \in S - \{x\}. [:- y, 1:] :: 'a \text{poly}) \ (\text{card } S - 1)$
by (*simp add: lagrange-basis-def*)
also have $\dots = \text{inverse } (\text{lagrange-denom } S \ x) * \text{lead-coeff } (\prod y \in S - \{x\}. [:- y, 1:] :: 'a \text{poly})$
by (*simp add: deg-prod*)
also have $\dots = \text{inverse } (\text{lagrange-denom } S \ x)$
using *fin* **by** (*simp add: lead-coeff-prod*)
finally show *?thesis .*
qed

lemma *lagrange-power-sum:*

fixes $S :: 'a::field \text{ set}$
assumes *fin: finite S and card: card S = Suc d*
assumes $k: k \leq d$
shows $(\sum x \in S. x \wedge k / \text{lagrange-denom } S \ x) = (\text{if } k = d \text{ then } 1 \text{ else } 0)$

proof –
let $?X = \text{monom } (1::'a) \ k$
have $\text{interp: } ?X = (\sum x \in S. \text{smult } (\text{poly } ?X \ x) \ (\text{lagrange-basis } S \ x))$
using $\text{card } k$ **by** $(\text{intro } \text{lagrange-interpolation}[OF \ \text{fin}]) \ (\text{simp } \text{add: } \text{degree-monom-eq})$
have coeff-basis:
 $\text{poly.coeff } (\text{lagrange-basis } S \ x) \ d = \text{inverse } (\text{lagrange-denom } S \ x)$ **if** $x \in S$ **for** x
using $\text{coeff-lagrange-basis-top}[OF \ \text{fin } \text{that}] \ \text{card}$ **by** simp
have $\text{poly.coeff } ?X \ d =$
 $\text{poly.coeff } (\sum x \in S. \text{smult } (\text{poly } ?X \ x) \ (\text{lagrange-basis } S \ x)) \ d$
using interp **by** simp
also have $\dots = (\sum x \in S. x \wedge k / \text{lagrange-denom } S \ x)$
using fin **by** $(\text{simp } \text{add: } \text{coeff-sum } \text{poly-monom } \text{coeff-basis } \text{divide-inverse})$
finally show $?thesis$
using k **by** simp
qed

lemma $\text{lagrange-power-sum-list:}$
fixes $xs :: 'a::\text{field } \text{list}$
assumes $\text{dist: } \text{distinct } xs$ **and** $\text{len: } \text{length } xs = \text{Suc } d$ **and** $k: k \leq d$
shows $\text{sum-list } (\text{map } (\lambda x. x \wedge k / \text{lagrange-denom } (\text{set } xs) \ x) \ xs) = (\text{if } k = d \ \text{then } 1 \ \text{else } 0)$
proof –
have $\text{sum-list } (\text{map } (\lambda x. x \wedge k / \text{lagrange-denom } (\text{set } xs) \ x) \ xs) =$
 $(\sum x \in \text{set } xs. x \wedge k / \text{lagrange-denom } (\text{set } xs) \ x)$
using dist **by** $(\text{rule } \text{sum-list-distinct-conv-sum-set})$
also have $\dots = (\text{if } k = d \ \text{then } 1 \ \text{else } 0)$
using $\text{dist } \text{len } k$ **by** $(\text{intro } \text{lagrange-power-sum}) \ (\text{auto } \text{simp: } \text{distinct-card})$
finally show $?thesis .$
qed

3.2 Sparse multivariate polynomials

Monomials are indexed by exponent lists. The value of $[e_1, \dots, e_n]$ at $[x_1, \dots, x_n]$ is $x_1 \wedge e_1 * \dots * x_n \wedge e_n$; mismatched lengths evaluate to zero. The predicate sparse-poly records finite support and a fixed arity.

fun $\text{monomial-value} :: \text{nat } \text{list} \Rightarrow 'a::\text{comm-semiring-1 } \text{list} \Rightarrow 'a$ **where**
 $\text{monomial-value } [] \ [] = 1$
 $|\ \text{monomial-value } (e \# es) \ (x \# xs) = x \wedge e * \text{monomial-value } es \ xs$
 $|\ \text{monomial-value } _ _ = 0$

fun $\text{grid-weight} :: 'a::\text{field } \text{list } \text{list} \Rightarrow 'a \ \text{list} \Rightarrow 'a$ **where**
 $\text{grid-weight } [] \ [] = 1$
 $|\ \text{grid-weight } (S \# Ss) \ (x \# xs) = \text{lagrange-denom } (\text{set } S) \ x * \text{grid-weight } Ss \ xs$
 $|\ \text{grid-weight } _ _ = 1$

definition $\text{support} :: (\text{nat } \text{list} \Rightarrow 'a::\text{zero}) \Rightarrow \text{nat } \text{list } \text{set}$ **where**
 $\text{support } p = \{m. p \ m \neq 0\}$

definition $\text{sparse-poly} :: \text{nat} \Rightarrow (\text{nat } \text{list} \Rightarrow 'a::\text{zero}) \Rightarrow \text{bool}$ **where**
 $\text{sparse-poly } n \ p \longleftrightarrow \text{finite } (\text{support } p) \wedge (\forall m \in \text{support } p. \text{length } m = n)$

definition $\text{total-degree-le} :: (\text{nat } \text{list} \Rightarrow 'a::\text{zero}) \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
 $\text{total-degree-le } p \ d \longleftrightarrow (\forall m \in \text{support } p. \text{sum-list } m \leq d)$

definition $\text{eval-sparse-poly} :: (\text{nat } \text{list} \Rightarrow 'a::\text{comm-semiring-1}) \Rightarrow 'a \ \text{list} \Rightarrow 'a$ **where**
 $\text{eval-sparse-poly } p \ xs = (\sum m \in \text{support } p. p \ m * \text{monomial-value } m \ xs)$

lemma $\text{product-lists-set-Cons:}$
 $\text{set } (\text{product-lists } (xs \# xss)) = (\lambda(x, ys). x \# ys) \ ` (\text{set } xs \times \text{set } (\text{product-lists } xss))$

by auto

lemma *sum-list-concat*:

$sum\text{-list } (concat\ xss) = sum\text{-list } (map\ sum\text{-list } xss)$
by (*induction xss*) *simp-all*

lemma *sum-list-map-zero*:

fixes $f :: 'a \Rightarrow 'b::monoid\text{-add}$
assumes $\bigwedge x. x \in set\ xs \implies f\ x = 0$
shows $sum\text{-list } (map\ f\ xs) = 0$
using *assms* by (*induction xs*) *auto*

lemma *sum-list-product-lists-Cons*:

$sum\text{-list } (map\ f\ (product\text{-lists } (xs\ \# \ xss))) =$
 $sum\text{-list } (map\ (\lambda x. sum\text{-list } (map\ (\lambda ys. f\ (x\ \# \ ys))\ (product\text{-lists } xss)))\ xs)$
by (*simp add: sum-list-concat map-concat comp-def*)

lemma *grid-weight-Cons*:

$grid\text{-weight } (S\ \# \ Ss)\ (x\ \# \ xs) = lagrange\text{-denom } (set\ S)\ x * grid\text{-weight } Ss\ xs$
by *simp*

lemma *monomial-value-Cons*:

$monomial\text{-value } (e\ \# \ es)\ (x\ \# \ xs) = x\ ^\ e * monomial\text{-value } es\ xs$
by *simp*

definition *grid-monom-sum* :: $'a::field\ list\ list \Rightarrow nat\ list \Rightarrow 'a$ **where**

$grid\text{-monom-sum } Xss\ es =$
 $sum\text{-list } (map\ (\lambda xs. monomial\text{-value } es\ xs / grid\text{-weight } Xss\ xs)\ (product\text{-lists } Xss))$

lemma *grid-monom-sum-Cons*:

assumes *dist: distinct Xs*
shows $grid\text{-monom-sum } (Xs\ \# \ Xss)\ (e\ \# \ es) =$
 $sum\text{-list } (map\ (\lambda x. x\ ^\ e / lagrange\text{-denom } (set\ Xs)\ x)\ Xs) * grid\text{-monom-sum } Xss\ es$

proof –

have *denom-nz*: $lagrange\text{-denom } (set\ Xs)\ x \neq 0$ **if** $x \in set\ Xs$ **for** x
using *that* by (*intro lagrange-denom-nonzero*) *auto*

have *inner*:

$sum\text{-list } (map\ (\lambda xs. monomial\text{-value } (e\ \# \ es)\ (x\ \# \ xs) /$
 $grid\text{-weight } (Xs\ \# \ Xss)\ (x\ \# \ xs))\ (product\text{-lists } Xss)) =$
 $(x\ ^\ e / lagrange\text{-denom } (set\ Xs)\ x) * sum\text{-list } (map\ (\lambda xs. monomial\text{-value } es\ xs / grid\text{-weight } Xss\ xs)$
 $(product\text{-lists } Xss))$

if $x \in set\ Xs$ **for** x

proof –

have *term-eq*:

$\bigwedge xs. monomial\text{-value } (e\ \# \ es)\ (x\ \# \ xs) / grid\text{-weight } (Xs\ \# \ Xss)\ (x\ \# \ xs) =$
 $(x\ ^\ e / lagrange\text{-denom } (set\ Xs)\ x) * (monomial\text{-value } es\ xs / grid\text{-weight } Xss\ xs)$

using *denom-nz[OF that]*

by (*simp add: divide-inverse ac-simps*)

have $sum\text{-list } (map\ (\lambda xs. monomial\text{-value } (e\ \# \ es)\ (x\ \# \ xs) /$
 $grid\text{-weight } (Xs\ \# \ Xss)\ (x\ \# \ xs))\ (product\text{-lists } Xss)) =$
 $sum\text{-list } (map\ (\lambda xs. (x\ ^\ e / lagrange\text{-denom } (set\ Xs)\ x) * (monomial\text{-value } es\ xs / grid\text{-weight } Xss\ xs))\ (product\text{-lists } Xss))$

by (*simp add: term-eq*)

also have $\dots =$

$(x\ ^\ e / lagrange\text{-denom } (set\ Xs)\ x) * sum\text{-list } (map\ (\lambda xs. monomial\text{-value } es\ xs / grid\text{-weight } Xss\ xs)\ (product\text{-lists } Xss))$

by (rule sum-list-const-mult)
 finally show ?thesis .
 qed
 have mapped:
 map ($\lambda x. \text{sum-list (map (\lambda xs. \text{monomial-value (e \# es) (x \# xs) / \text{grid-weight (Xs \# Xss) (x \# xs) (product-lists Xss)) Xs = \text{map (\lambda x. (x \wedge e / \text{lagrange-denom (set Xs) x) * \text{sum-list (map (\lambda xs. \text{monomial-value es xs / \text{grid-weight Xss xs) (product-lists Xss)) Xs}$)))))
 using inner by auto
 have grid-monom-sum (Xs # Xss) (e # es) =
 sum-list (map ($\lambda x. \text{sum-list (map (\lambda xs. \text{monomial-value (e \# es) (x \# xs) / \text{grid-weight (Xs \# Xss) (x \# xs) (product-lists Xss)) Xs}$)))
 unfolding grid-monom-sum-def by (rule sum-list-product-lists-Cons)
 also have ... =
 sum-list (map ($\lambda x. x \wedge e / \text{lagrange-denom (set Xs) x) Xs$) *
 grid-monom-sum Xss es
 using grid-monom-sum-def[of Xss es] mapped
 sum-list-mult-const[of - grid-monom-sum Xss es Xs]
 by presburger
 finally show ?thesis .
 qed

 lemma grid-monom-sum-Nil [simp]:
 grid-monom-sum [] [] = 1
 by (simp add: grid-monom-sum-def)

 lemma monomial-value-wrong-length:
 length es \neq length xs \implies monomial-value es xs = 0
 proof (induction es arbitrary: xs)
 case Nil
 then show ?case
 by (cases xs) simp-all
 next
 case (Cons e es xs)
 then show ?case
 by (cases xs) auto
 qed

 lemma grid-monom-sum-wrong-length:
 assumes length es \neq length Xss
 shows grid-monom-sum Xss es = 0
 proof -
 have zero: monomial-value es xs = 0 if xs \in set (product-lists Xss) for xs
 by (metis in-set-product-lists-length assms monomial-value-wrong-length that)
 show ?thesis
 unfolding grid-monom-sum-def
 by (rule sum-list-map-zero) (metis divide-eq-0-iff zero)
 qed

 lemma grid-monom-sum-delta:
 fixes Xss :: 'a::field list list
 assumes grids: list-all2 ($\lambda Xs d. \text{distinct Xs} \wedge \text{length Xs} = \text{Suc } d$) Xss ds
 assumes len: length es = length ds
 assumes deg: sum-list es \leq sum-list ds
 shows grid-monom-sum Xss es = (if es = ds then 1 else 0)
 using grids len deg
 proof (induction Xss arbitrary: ds es)
 case Nil

```

then show ?case
  by (cases ds; cases es) simp-all
next
case (Cons X Xss ds es)
then obtain d ds' where ds: ds = d # ds'
  by (cases ds) auto
then obtain e es' where es: es = e # es'
  using Cons.prem1 by (cases es) auto
from Cons.prem2 ds es have X: distinct X length X = Suc d
  by auto
from Cons.prem3 ds es have grids-tail:
  list-all2 ( $\lambda Xs d. \text{distinct } Xs \wedge \text{length } Xs = \text{Suc } d$ ) Xss ds'
  and len-tail: length es' = length ds'
  and deg-all:  $e + \text{sum-list } es' \leq d + \text{sum-list } ds'$ 
  by auto
show ?case
proof (cases e d rule: linorder-cases)
  case less
  have head-delta: sum-list (map ( $\lambda x. x \wedge e / \text{lagrange-denom } (\text{set } X) x$ ) X) =
    (if e = d then 1 else 0)
  using X less by (intro lagrange-power-sum-list) auto
  then have head: sum-list (map ( $\lambda x. x \wedge e / \text{lagrange-denom } (\text{set } X) x$ ) X) = 0
  using less by simp
  show ?thesis
  using ds es less head X by (simp add: grid-monom-sum-Cons)
next
  case equal
  have head-delta: sum-list (map ( $\lambda x. x \wedge e / \text{lagrange-denom } (\text{set } X) x$ ) X) =
    (if e = d then 1 else 0)
  using X equal by (intro lagrange-power-sum-list) auto
  then have head: sum-list (map ( $\lambda x. x \wedge e / \text{lagrange-denom } (\text{set } X) x$ ) X) = 1
  using equal by simp
  have tail-deg: sum-list es'  $\leq$  sum-list ds'
  using deg-all equal by simp
  have tail: grid-monom-sum Xss es' = (if es' = ds' then 1 else 0)
  by (rule Cons.IH) (use grids-tail len-tail tail-deg in auto)
  show ?thesis
  using ds es equal head tail X by (simp add: grid-monom-sum-Cons)
next
  case greater
  have tail-deg: sum-list es'  $\leq$  sum-list ds'
  using deg-all greater by simp
  have tail-ne: es'  $\neq$  ds'
  proof
    assume es' = ds'
    with deg-all greater show False
    by simp
  qed
  have tail: grid-monom-sum Xss es' = 0
  using Cons.IH[OF grids-tail len-tail tail-deg] tail-ne by auto
  show ?thesis
  using ds es greater tail X by (simp add: grid-monom-sum-Cons)
qed
qed

```

The next lemma is the coefficient formula specialized to the sparse representation: the weighted sum of all grid evaluations extracts exactly the coefficient of the target exponent list.

```

lemma sum-list-sum:
  fixes f :: 'b  $\Rightarrow$  'c  $\Rightarrow$  'a::comm-monoid-add

```

assumes *finite A*
shows *sum-list* (*map* ($\lambda x. \sum a \in A. f x a$) *xs*) =
 $(\sum a \in A. \text{sum-list } (\text{map } (\lambda x. f x a) \text{ xs}))$
using *assms* **by** (*induction xs*) (*simp-all add: sum.distrib*)

lemma *eval-sparse-poly-grid-sum:*

fixes *p :: nat list* \Rightarrow *'a::field*
assumes *sp: sparse-poly* (*length ds*) *p*
assumes *grids: list-all2* ($\lambda Xs d. \text{distinct } Xs \wedge \text{length } Xs = \text{Suc } d$) *Xss ds*
assumes *deg: total-degree-le p* (*sum-list ds*)
shows *sum-list* (*map* ($\lambda xs. \text{eval-sparse-poly } p \text{ xs} / \text{grid-weight } Xss \text{ xs}$) (*product-lists Xss*)) =
 $p \text{ ds}$
proof –
have *fin: finite* (*support p*)
using *sp* **by** (*simp add: sparse-poly-def*)
have *len-Xss: length Xss = length ds*
using *grids* **by** (*simp add: list-all2-lengthD*)
have *sum-list* (*map* ($\lambda xs. \text{eval-sparse-poly } p \text{ xs} / \text{grid-weight } Xss \text{ xs}$) (*product-lists Xss*)) =
 $\text{sum-list } (\text{map } (\lambda xs. \sum m \in \text{support } p. p \text{ m} * (\text{monomial-value } m \text{ xs} / \text{grid-weight } Xss \text{ xs})) (\text{product-lists } Xss))$
by (*simp add: eval-sparse-poly-def sum-divide-distrib sum-distrib-left mult.assoc*)
also have $\dots =$
 $(\sum m \in \text{support } p. \text{sum-list } (\text{map } (\lambda xs. p \text{ m} * (\text{monomial-value } m \text{ xs} / \text{grid-weight } Xss \text{ xs})) (\text{product-lists } Xss)))$
using *fin* **by** (*simp add: sum-list-sum*)
also have $\dots = (\sum m \in \text{support } p. p \text{ m} * \text{grid-monom-sum } Xss \text{ m})$
by (*intro sum.cong refl*)
(simp add: grid-monom-sum-def sum-list-const-mult divide-inverse ac-simps)
also have $\dots = p \text{ ds}$
proof –
have *delta: grid-monom-sum Xss m = (if m = ds then 1 else 0)* **if** $m \in \text{support } p$ **for** *m*
proof –
have *length m = length ds*
using *sp that* **by** (*simp add: sparse-poly-def*)
moreover have *sum-list m ≤ sum-list ds*
using *deg that* **by** (*simp add: total-degree-le-def*)
ultimately show *?thesis*
by (*rule grid-monom-sum-delta[OF grids]*)
qed
show *?thesis*
proof (*cases ds ∈ support p*)
case *True*
have $(\sum m \in \text{support } p. p \text{ m} * \text{grid-monom-sum } Xss \text{ m}) =$
 $p \text{ ds} * \text{grid-monom-sum } Xss \text{ ds}$
using *fin True delta* **by** (*subst sum.remove[of support p ds]*) *auto*
also have $\dots = p \text{ ds}$
using *delta True* **by** *simp*
finally show *?thesis* .
next
case *False*
then have *p0: p ds = 0*
by (*simp add: support-def*)
have $(\sum m \in \text{support } p. p \text{ m} * \text{grid-monom-sum } Xss \text{ m}) = 0$
using *fin False delta* **by** (*intro sum.neutral*) *auto*
with *p0* **show** *?thesis*
by *simp*
qed
qed

finally show *?thesis* .
qed

theorem *combinatorial-nullstellensatz-exact-lists*:

fixes $p :: \text{nat list} \Rightarrow 'a::\text{field}$
assumes $sp: \text{sparse-poly } (\text{length } ds) \ p$
assumes $deg: \text{total-degree-le } p \ (\text{sum-list } ds)$
assumes $coeff: p \ ds \neq 0$
assumes $grids: \text{list-all2 } (\lambda Xs \ d. \text{distinct } Xs \wedge \text{length } Xs = \text{Suc } d) \ Xss \ ds$
shows $\exists xs \in \text{set } (\text{product-lists } Xss). \text{eval-sparse-poly } p \ xs \neq 0$
proof (*rule ccontr*)
assume $\neg ?thesis$
then have $zero: \bigwedge xs. xs \in \text{set } (\text{product-lists } Xss) \implies \text{eval-sparse-poly } p \ xs = 0$
by *auto*
have $\text{sum-list } (\text{map } (\lambda xs. \text{eval-sparse-poly } p \ xs / \text{grid-weight } Xss \ xs) \ (\text{product-lists } Xss)) = 0$
by (*rule sum-list-map-zero*) (*metis divide-eq-0-iff zero*)
moreover have $\text{sum-list } (\text{map } (\lambda xs. \text{eval-sparse-poly } p \ xs / \text{grid-weight } Xss \ xs) \ (\text{product-lists } Xss)) =$
 $p \ ds$
by (*rule eval-sparse-poly-grid-sum[OF sp grids deg]*)
ultimately show *False*
using *coeff* **by** *simp*
qed

Finally, the standard “more than d_i points” formulation follows by selecting $d_i + 1$ points from each side of the grid.

definition *exact-grid-sublists* $:: 'a \text{ list list} \Rightarrow \text{nat list} \Rightarrow 'a \text{ list list}$ **where**
 $\text{exact-grid-sublists } Xss \ ds = \text{map } (\lambda(Xs, d). \text{take } (\text{Suc } d) \ Xs) \ (\text{zip } Xss \ ds)$

lemma *exact-grid-sublists-all2*:

assumes $\text{list-all2 } (\lambda Xs \ d. \text{distinct } Xs \wedge \text{length } Xs > d) \ Xss \ ds$
shows $\text{list-all2 } (\lambda Xs \ d. \text{distinct } Xs \wedge \text{length } Xs = \text{Suc } d)$
 $(\text{exact-grid-sublists } Xss \ ds) \ ds$
using *assms*
proof (*induction Xss arbitrary: ds*)
case *Nil*
then show *?case*
by (*cases ds*) (*simp-all add: exact-grid-sublists-def*)
next
case (*Cons X Xss ds*)
then obtain $d \ ds'$ **where** $ds: ds = d \ \#\ ds'$
by (*cases ds*) *auto*
with *Cons.prem* **have** $\text{distinct } (\text{take } (\text{Suc } d) \ X) \ \text{length } (\text{take } (\text{Suc } d) \ X) = \text{Suc } d$
by *auto*
moreover have $\text{list-all2 } (\lambda Xs \ d. \text{distinct } Xs \wedge \text{length } Xs = \text{Suc } d)$
 $(\text{exact-grid-sublists } Xss \ ds') \ ds'$
using *Cons.IH Cons.prem ds* **by** *auto*
ultimately show *?case*
by (*simp add: exact-grid-sublists-def ds*)
qed

lemma *exact-grid-sublists-subset*:

assumes $\text{list-all2 } (\lambda Xs \ d. \text{length } Xs > d) \ Xss \ ds$
shows $\text{set } (\text{product-lists } (\text{exact-grid-sublists } Xss \ ds)) \subseteq \text{set } (\text{product-lists } Xss)$
using *assms*
proof (*induction Xss arbitrary: ds*)
case *Nil*
then show *?case*
by (*cases ds*) (*simp-all add: exact-grid-sublists-def*)
next

```

case (Cons X Xss ds)
then obtain d ds' where ds: ds = d # ds'
  by (cases ds) auto
have tail: set (product-lists (exact-grid-sublists Xss ds'))  $\subseteq$  set (product-lists Xss)
  using Cons.IH Cons.premis ds by auto
show ?case
proof
  fix xs
  assume xs  $\in$  set (product-lists (exact-grid-sublists (X # Xss) ds))
  then obtain x ys where xs: xs = x # ys
    and x: x  $\in$  set (take (Suc d) X)
    and ys: ys  $\in$  set (product-lists (exact-grid-sublists Xss ds'))
    by (auto simp: exact-grid-sublists-def ds)
  from x have x  $\in$  set X
    by (rule in-set-takeD)
  moreover from tail ys have ys  $\in$  set (product-lists Xss)
    by auto
  ultimately show xs  $\in$  set (product-lists (X # Xss))
    using xs by auto
qed
qed

```

theorem *combinatorial-nullstellensatz-lists:*

```

fixes p :: nat list  $\Rightarrow$  'a::field
assumes sp: sparse-poly (length ds) p
assumes deg: total-degree-le p (sum-list ds)
assumes coeff: p ds  $\neq$  0
assumes grids: list-all2 ( $\lambda Xs d. distinct Xs \wedge length Xs > d$ ) Xss ds
shows  $\exists xs \in set (product-lists Xss). eval-sparse-poly p xs \neq 0$ 
proof –
  have exact: list-all2 ( $\lambda Xs d. distinct Xs \wedge length Xs = Suc d$ )
    (exact-grid-sublists Xss ds) ds
    by (rule exact-grid-sublists-all2[OF grids])
  obtain xs where xs: xs  $\in$  set (product-lists (exact-grid-sublists Xss ds))
    and nz: eval-sparse-poly p xs  $\neq$  0
    using combinatorial-nullstellensatz-exact-lists[OF sp deg coeff exact] by blast
  have lengths: list-all2 ( $\lambda Xs d. length Xs > d$ ) Xss ds
    by (rule list-all2-mono[OF grids]) auto
  have xs  $\in$  set (product-lists Xss)
    using exact-grid-sublists-subset[OF lengths] xs by auto
  with nz show ?thesis
    by blast
qed

```

end

References

- [1] N. Alon. Combinatorial nullstellensatz. *Combinatorics, Probability and Computing*, 8(1–2):7–29, 1999. DOI: <https://doi.org/10.1017/S0963548398003411>.