Alon's Combinatorial Nullstellensatz

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

June 25, 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 $d_1 + \cdots + d_n$ and the coefficient of $X_1^{d_1} \cdots X_n^{d_n}$ 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

theory *Combinatorial-Nullstellensatz*
imports *HOL−Computational-Algebra.Polynomial*
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 $d_1 + \ldots + d_n$ 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* ⇒ *'a* ⇒ *'a* **where**
lagrange-denom S x = (∏ *y*∈*S* − {*x*}. *x* − *y*)

definition *lagrange-basis* :: *'a*::*field set* ⇒ *'a* ⇒ *'a poly* **where**
lagrange-basis S x =
smult (*inverse* (*lagrange-denom S x*))
(∏ *y*∈*S* − {*x*}. [:− *y*, *1*:])

lemma *lagrange-denom-nonzero*:
assumes *finite S x* ∈ *S*
shows *lagrange-denom S x* ≠ *0*
⟨*proof*⟩

lemma *poly-lagrange-basis*:
assumes *finite S x* ∈ *S z* ∈ *S*
shows *poly* (*lagrange-basis S x*) *z* = (*if z* = *x then 1 else 0*)
⟨*proof*⟩

lemma *degree-lagrange-basis-le*:
assumes *finite S x* ∈ *S*
shows *degree* (*lagrange-basis S x*) ≤ *card S* − *1*
⟨*proof*⟩

lemma *lagrange-interpolation*:
fixes *P* :: *'a*::{*field*,*ring-no-zero-divisors*} *poly*
assumes *fin*: *finite S* **and** *deg*: *degree P* < *card S*
shows *P* = (∑ *x*∈*S*. *smult* (*poly P x*) (*lagrange-basis S x*))
⟨*proof*⟩

lemma *coeff-lagrange-basis-top*:
assumes *fin*: *finite S* **and** *x*: *x* ∈ *S*
shows *poly.coeff* (*lagrange-basis S x*) (*card S* − *1*) =
inverse (*lagrange-denom S x*)
⟨*proof*⟩

lemma *lagrange-power-sum*:
fixes *S* :: *'a*::*field set*
assumes *fin*: *finite S* **and** *card*: *card S* = *Suc d*
assumes *k*: *k* ≤ *d*
shows (∑ *x*∈*S*. *x* ^ *k* / *lagrange-denom S x*) = (*if k* = *d then 1 else 0*)
⟨*proof*⟩

lemma *lagrange-power-sum-list*:
fixes *xs* :: *'a*::*field list*
assumes *dist*: *distinct xs* **and** *len*: *length xs* = *Suc d* **and** *k*: *k* ≤ *d*
shows *sum-list* (*map* (λ*x*. *x* ^ *k* / *lagrange-denom* (*set xs*) *x*) *xs*) = (*if k* = *d then 1 else 0*)
⟨*proof*⟩

3.2 Sparse multivariate polynomials

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

fun *monomial-value* :: *nat list* ⇒ *'a::comm-semiring-1 list* ⇒ *'a* **where**
  monomial-value [] [] = *1*
| *monomial-value* (*e* # *es*) (*x* # *xs*) = *x* ^ *e* ∗ *monomial-value es xs*
| *monomial-value* - - = *0*

fun *grid-weight* :: *'a::field list list* ⇒ *'a list* ⇒ *'a* **where**
  grid-weight [] [] = *1*
| *grid-weight* (*S* # *Ss*) (*x* # *xs*) = *lagrange-denom* (*set S*) *x* ∗ *grid-weight Ss xs*
| *grid-weight* - - = *1*

definition *support* :: (*nat list* ⇒ *'a::zero*) ⇒ *nat list set* **where**
  support p = {*m*. *p m* ≠ *0*}

definition *sparse-poly* :: *nat* ⇒ (*nat list* ⇒ *'a::zero*) ⇒ *bool* **where**
  sparse-poly n p ⟷ *finite* (*support p*) ∧ (∀ *m*∈*support p*. *length m* = *n*)

definition *total-degree-le* :: (*nat list* ⇒ *'a::zero*) ⇒ *nat* ⇒ *bool* **where**
  total-degree-le p d ⟷ (∀ *m*∈*support p*. *sum-list m* ≤ *d*)

definition *eval-sparse-poly* :: (*nat list* ⇒ *'a::comm-semiring-1*) ⇒ *'a list* ⇒ *'a* **where**
  eval-sparse-poly p xs = (∑ *m*∈*support p*. *p m* ∗ *monomial-value m xs*)

lemma *product-lists-set-Cons*:
  set (*product-lists* (*xs* # *xss*)) = (λ(*x*, *ys*). *x* # *ys*) ‘ (*set xs* × *set* (*product-lists xss*))
  ⟨*proof*⟩

lemma *sum-list-concat*:
  sum-list (*concat xss*) = *sum-list* (*map sum-list xss*)
  ⟨*proof*⟩

lemma *sum-list-map-zero*:
  fixes *f* :: *'a* ⇒ *'b::monoid-add*
  assumes ⋀*x*. *x* ∈ *set xs* ⟹ *f x* = *0*
  shows *sum-list* (*map f xs*) = *0*
  ⟨*proof*⟩

lemma *sum-list-product-lists-Cons*:
  sum-list (*map f* (*product-lists* (*xs* # *xss*))) =
    sum-list (*map* (λ*x*. *sum-list* (*map* (λ*ys*. *f* (*x* # *ys*)) (*product-lists xss*))) *xs*)
  ⟨*proof*⟩

lemma *grid-weight-Cons*:
  grid-weight (*S* # *Ss*) (*x* # *xs*) = *lagrange-denom* (*set S*) *x* ∗ *grid-weight Ss xs*
  ⟨*proof*⟩

lemma *monomial-value-Cons*:
  monomial-value (*e* # *es*) (*x* # *xs*) = *x* ^ *e* ∗ *monomial-value es xs*
  ⟨*proof*⟩

definition *grid-monom-sum* :: *'a::field list list* ⇒ *nat list* ⇒ *'a* **where**
  grid-monom-sum Xss es =
    sum-list (*map* (λ*xs*. *monomial-value es xs* / *grid-weight Xss xs*) (*product-lists Xss*))

lemma *grid-monom-sum-Cons*:
assumes *dist*: *distinct Xs*
shows *grid-monom-sum* (*Xs # Xss*) (*e # es*) =
sum-list (*map* (λ*x*. *x* ^ *e* / *lagrange-denom* (*set Xs*) *x*) *Xs*) ∗
grid-monom-sum Xss es
⟨*proof*⟩

lemma *grid-monom-sum-Nil* [*simp*]:
grid-monom-sum [] [] = *1*
⟨*proof*⟩

lemma *monomial-value-wrong-length*:
length es ≠ *length xs* ⟹ *monomial-value es xs* = *0*
⟨*proof*⟩

lemma *grid-monom-sum-wrong-length*:
assumes *length es* ≠ *length Xss*
shows *grid-monom-sum Xss es* = *0*
⟨*proof*⟩

lemma *grid-monom-sum-delta*:
fixes *Xss* :: *'a*::*field list list*
assumes *grids*: *list-all2* (λ*Xs d*. *distinct Xs* ∧ *length Xs* = *Suc d*) *Xss ds*
assumes *len*: *length es* = *length ds*
assumes *deg*: *sum-list es* ≤ *sum-list ds*
shows *grid-monom-sum Xss es* = (*if es* = *ds then 1 else 0*)
⟨*proof*⟩

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* ⇒ *'c* ⇒ *'a*::*comm-monoid-add*
assumes *finite A*
shows *sum-list* (*map* (λ*x*. ∑ *a*∈*A*. *f x a*) *xs*) =
(∑ *a*∈*A*. *sum-list* (*map* (λ*x*. *f x a*) *xs*))
⟨*proof*⟩

lemma *eval-sparse-poly-grid-sum*:
fixes *p* :: *nat list* ⇒ *'a*::*field*
assumes *sp*: *sparse-poly* (*length ds*) *p*
assumes *grids*: *list-all2* (λ*Xs d*. *distinct Xs* ∧ *length Xs* = *Suc d*) *Xss ds*
assumes *deg*: *total-degree-le p* (*sum-list ds*)
shows *sum-list* (*map* (λ*xs*. *eval-sparse-poly p xs* / *grid-weight Xss xs*) (*product-lists Xss*)) =
p ds
⟨*proof*⟩

theorem *combinatorial-nullstellensatz-exact-lists*:
fixes *p* :: *nat list* ⇒ *'a*::*field*
assumes *sp*: *sparse-poly* (*length ds*) *p*
assumes *deg*: *total-degree-le p* (*sum-list ds*)
assumes *coeff*: *p ds* ≠ *0*
assumes *grids*: *list-all2* (λ*Xs d*. *distinct Xs* ∧ *length Xs* = *Suc d*) *Xss ds*
shows ∃ *xs*∈*set* (*product-lists Xss*). *eval-sparse-poly p xs* ≠ *0*
⟨*proof*⟩

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 list list* ⇒ *nat list* ⇒ *'a list list* **where**
exact-grid-sublists Xss ds = *map* (λ(*Xs*, *d*). *take* (*Suc d*) *Xs*) (*zip Xss ds*)

lemma *exact-grid-sublists-all2*:
assumes *list-all2* ($\lambda Xs\ d.\ distinct\ Xs \wedge length\ Xs > d$) *Xss ds*
shows *list-all2* ($\lambda Xs\ d.\ distinct\ Xs \wedge length\ Xs = Suc\ d$)
(*exact-grid-sublists Xss ds*) *ds*
⟨*proof*⟩

lemma *exact-grid-sublists-subset*:
assumes *list-all2* ($\lambda Xs\ d.\ length\ Xs > d$) *Xss ds*
shows *set* (*product-lists* (*exact-grid-sublists Xss ds*)) $\subseteq$ *set* (*product-lists Xss*)
⟨*proof*⟩

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*⟩

end

References

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