

The Chevalley–Warning Theorem

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

July 16, 2026

Abstract

This development formalizes the Chevalley–Warning theorem for finite fields: if finitely many multivariate polynomials over a finite field have total degree sum strictly smaller than the number of variables, then the number of their common zeros is divisible by the characteristic [1, 3]. The formalization reuses the concrete multivariate-polynomial representation and operations from the AFP *Polynomials* entry [2], and the proof follows the standard finite-field power-sum argument. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

Contents

1	The Chevalley–Warning theorem	1
1.1	Polynomial operations	3
1.2	Finite-field power sums	10
1.3	The theorem	17

```
theory Chevalley-Warning
  imports
    HOL-Number-Theory.Residues
    HOL-Computational-Algebra.Polynomial
    HOL-Library.FuncSet
    Polynomials.Polynomials
begin
```

1 The Chevalley–Warning theorem

We reuse the concrete multivariate-polynomial representation from the AFP entry *Polynomials.Polynomials*. Thus a polynomial has type $(\text{'v}, \text{'a}) \text{Polynomials.poly}$ and consists of AFP monomials paired with coefficients. The theorem takes an explicit finite set of variables, so it applies directly to the usual natural-number-indexed AFP polynomials.

definition *variables-poly* ::

$'v::\text{linorder set} \Rightarrow ('v, 'a) \text{ poly} \Rightarrow \text{bool}$ **where**
 $\text{variables-poly } V p \longleftrightarrow \text{poly-vars } p \subseteq V$

definition $\text{total-degree-poly-le} :: ('v::\text{linorder}, 'a) \text{ poly} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
 $\text{total-degree-poly-le } p d \longleftrightarrow \text{poly-degree } p \leq d$

definition $\text{total-degree-poly-less} :: ('v::\text{linorder}, 'a) \text{ poly} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
 $\text{total-degree-poly-less } p d \longleftrightarrow \text{poly-degree } p < d$

definition $\text{common-zeros} ::$
 $'v::\text{linorder set} \Rightarrow$
 $('v, 'a::\text{comm-semiring-1}) \text{ poly list} \Rightarrow ('v \Rightarrow 'a) \text{ set}$ **where**
 $\text{common-zeros } V ps =$
 $\{\alpha \in \text{PiE } V (\lambda\cdot. \text{UNIV}). \text{list-all } (\lambda p. \text{eval-poly } \alpha p = 0) ps\}$

lemma $\text{variables-poly-UNIV [simp]: variables-poly UNIV } p$
by ($\text{simp add: variables-poly-def}$)

lemma $\text{monom-vars-subset-poly-vars:}$
assumes $(m, c) \in \text{set } p$
shows $\text{monom-vars } m \subseteq \text{poly-vars } p$
using assms
by ($\text{auto simp: monom-vars-def poly-vars-def}$)

lemma poly-degree-leI:
assumes $\bigwedge m c. (m, c) \in \text{set } p \implies \text{monom-degree } m \leq d$
shows $\text{poly-degree } p \leq d$
using assms
by ($\text{induction } p$) ($\text{auto simp: poly-degree-def split: prod.splits}$)

lemma $\text{total-degree-poly-leD:}$
assumes $\text{total-degree-poly-le } p d (m, c) \in \text{set } p$
shows $\text{monom-degree } m \leq d$
proof –
have $\text{monom-degree } m \in \text{set } (\text{map } (\lambda(m, c). \text{monom-degree } m) p)$
using $\text{assms}(2)$ **by** force
then have $\text{monom-degree } m \leq \text{max-list } (\text{map } (\lambda(m, c). \text{monom-degree } m) p)$
by (rule max-list)
with $\text{assms}(1)$ **show** $?thesis$
by ($\text{simp add: total-degree-poly-le-def poly-degree-def}$)
qed

lemma $\text{total-degree-poly-lessD:}$
assumes $\text{total-degree-poly-less } p d (m, c) \in \text{set } p$
shows $\text{monom-degree } m < d$
using $\text{total-degree-poly-leD[of } p \text{ poly-degree } p m c] \text{ assms}$
by ($\text{simp add: total-degree-poly-le-def total-degree-poly-less-def}$)

lemma $\text{total-degree-poly-le-mono:}$

assumes *total-degree-poly-le* $p \ d \ d \leq e$
shows *total-degree-poly-le* $p \ e$
using *assms* **by** (*simp* *add: total-degree-poly-le-def*)

lemma *total-degree-poly-less-of-le*:
assumes *total-degree-poly-le* $p \ d \ d < e$
shows *total-degree-poly-less* $p \ e$
using *assms*
by (*simp* *add: total-degree-poly-le-def total-degree-poly-less-def*)

1.1 Polynomial operations

fun *poly-prod* :: ($'v::\text{linorder}$, $'a::\text{comm-semiring-1}$) *poly list* $\Rightarrow$ ($'v$, $'a$) *poly* **where**
poly-prod [] = *one-poly*
| *poly-prod* ($p \# \text{ps}$) = *poly-mult* p (*poly-prod* *ps*)

definition *indicator-factor* ::
 $\text{nat} \Rightarrow$ ($'v::\text{linorder}$, $'a::\text{comm-ring-1}$) *poly* $\Rightarrow$ ($'v$, $'a$) *poly* **where**
indicator-factor $q \ p$ = *poly-minus* (*poly-const* 1) (*poly-power* p ($q - 1$))

definition *indicator-poly* ::
 $\text{nat} \Rightarrow$ ($'v::\text{linorder}$, $'a::\text{comm-ring-1}$) *poly list* $\Rightarrow$ ($'v$, $'a$) *poly* **where**
indicator-poly $q \ \text{ps}$ = *poly-prod* (*map* (*indicator-factor* q) *ps*)

lemma *monom-list-degree-mult*:
monom-list-degree (*monom-mult-list* $m \ n$) =
monom-list-degree m + *monom-list-degree* n
by (*induction* $m \ n$ *rule: monom-mult-list.induct*)
(*auto simp: monom-list-degree-def monom-mult-list.simps*
split: list.splits prod.splits if-splits)

lemma *monom-degree-mult* [*simp*]:
monom-degree ($m * n$) = *monom-degree* m + *monom-degree* n
by *transfer* (*rule monom-list-degree-mult*)

lemma *monom-degree-one* [*simp*]:
monom-degree ($1::'v::\text{linorder monom}$) = 0
by *transfer* (*simp add: monom-list-degree-def*)

lemma *monom-vars-mult* [*simp*]:
monom-vars ($m * n$) = *monom-vars* $m \cup$ *monom-vars* n
unfolding *monom-vars-def*
by *transfer* (*metis monom-list-mult-list-vars set-map*)

lemma *monom-mult-poly-monoms*:
poly-monoms (*monom-mult-poly* (m , c) p) $\subseteq$ $(*) \ m \ \text{'poly-monoms } p$
proof (*induction* p)
case *Nil*
then show ?*case*

```

    by (simp add: monom-mult-poly.simps)
next
  case (Cons t p)
  obtain n d where t: t = (n, d)
  by force
  show ?case
  using Cons.IH
  by (cases c * d = 0)
    (auto simp: monom-mult-poly.simps t)
qed

lemma variables-poly-const [simp]:
  variables-poly V (poly-const c :: ('v::linorder, 'a::zero) poly)
  by (auto simp: variables-poly-def poly-vars-def poly-const-def monom-vars-def)

lemma total-degree-poly-le-const [simp]:
  total-degree-poly-le (poly-const c :: ('v::linorder, 'a::zero) poly) 0
  by (auto simp: total-degree-poly-le-def poly-degree-def poly-const-def)

lemma variables-poly-add:
  assumes p: variables-poly V p and q: variables-poly V q
  shows variables-poly V (poly-add p q)
proof (unfold variables-poly-def, rule subsetI)
  fix x
  assume x ∈ poly-vars (poly-add p q)
  then obtain m c where
    mc: (m, c) ∈ set (poly-add p q) and x: x ∈ monom-vars m
  by (auto simp: poly-vars-def monom-vars-def)
  have m ∈ poly-monoms p ∪ poly-monoms q
    using poly-add-monoms[of p q] mc by force
  then have monom-vars m ⊆ V
  proof
    assume m ∈ poly-monoms p
    then obtain c' where (m, c') ∈ set p
    by auto
    then show ?thesis
    using monom-vars-subset-poly-vars[of m c' p] p
    by (auto simp: variables-poly-def)
  next
    assume m ∈ poly-monoms q
    then obtain c' where (m, c') ∈ set q
    by auto
    then show ?thesis
    using monom-vars-subset-poly-vars[of m c' q] q
    by (auto simp: variables-poly-def)
  qed
  with x show x ∈ V
  by blast
qed

```

lemma *total-degree-poly-le-add*:
assumes *p*: *total-degree-poly-le* *p* *d* **and** *q*: *total-degree-poly-le* *q* *d*
shows *total-degree-poly-le* (*poly-add* *p* *q*) *d*
proof (*unfold total-degree-poly-le-def*, *rule poly-degree-leI*)
fix *m c*
assume (*m, c*) $\in$ *set* (*poly-add* *p* *q*)
then have *m* $\in$ *poly-monoms* (*poly-add* *p* *q*)
by force
then have *m* $\in$ *poly-monoms* *p* $\cup$ *poly-monoms* *q*
using *poly-add-monoms*[*of* *p* *q*] **by blast**
then show *monom-degree* *m* $\leq$ *d*
using *total-degree-poly-leD*[*OF* *p*] *total-degree-poly-leD*[*OF* *q*]
by auto
qed

lemma *total-degree-monom-mult-poly*:
assumes *m*: *monom-degree* *m* $\leq$ *d* **and** *p*: *total-degree-poly-le* *p* *e*
shows *total-degree-poly-le* (*monom-mult-poly* (*m, c*) *p*) (*d* + *e*)
proof (*unfold total-degree-poly-le-def*, *rule poly-degree-leI*)
fix *r a*
assume (*r, a*) $\in$ *set* (*monom-mult-poly* (*m, c*) *p*)
then have *r* $\in$ *poly-monoms* (*monom-mult-poly* (*m, c*) *p*)
by force
then obtain *r'* **where** *r'*: *r* = *m* * *r'* **and** *r'* $\in$ *poly-monoms* *p*
using *monom-mult-poly-monoms*[*of* *m* *c* *p*] **by blast**
then obtain *b* **where** (*r', b*) $\in$ *set* *p*
by auto
then have *monom-degree* *r'* $\leq$ *e*
by (*rule total-degree-poly-leD*[*OF* *p*])
with *m* **show** *monom-degree* *r* $\leq$ *d* + *e*
by (*simp add: r'*)
qed

lemma *variables-monom-mult-poly*:
assumes *m*: *monom-vars* *m* $\subseteq$ *V* **and** *p*: *variables-poly* *V* *p*
shows *variables-poly* *V* (*monom-mult-poly* (*m, c*) *p*)
proof (*unfold variables-poly-def*, *rule subsetI*)
fix *x*
assume *x* $\in$ *poly-vars* (*monom-mult-poly* (*m, c*) *p*)
then obtain *r a* **where**
ra: (*r, a*) $\in$ *set* (*monom-mult-poly* (*m, c*) *p*)
and *x*: *x* $\in$ *monom-vars* *r*
by (*auto simp: poly-vars-def monom-vars-def*)
then obtain *r'* **where** *r'*: *r* = *m* * *r'* **and** *r'* $\in$ *poly-monoms* *p*
using *monom-mult-poly-monoms*[*of* *m* *c* *p*] **by force**
then obtain *b* **where** *r'p*: (*r', b*) $\in$ *set* *p*
by auto
have *monom-vars* *r'* $\subseteq$ *V*

```

    using monom-vars-subset-poly-vars[OF r'p] p
    by (auto simp: variables-poly-def)
  with m x show  $x \in V$ 
    by (auto simp: r')
qed

lemma variables-poly-mult:
  assumes p: variables-poly V p and q: variables-poly V q
  shows variables-poly V (poly-mult p q)
  using p
proof (induction p)
  case Nil
  then show ?case
    by (simp add: poly-mult.simps variables-poly-def poly-vars-def)
next
  case (Cons t p)
  obtain m c where t:  $t = (m, c)$ 
    by force
  have m-vars: monom-vars  $m \subseteq V$ 
    using monom-vars-subset-poly-vars[of m c t # p] Cons.prem1 t
    by (auto simp: variables-poly-def)
  have p-vars: variables-poly V p
    using Cons.prem2
    by (auto simp: variables-poly-def poly-vars-def)
  have head: variables-poly V (monom-mult-poly (m, c) q)
    by (rule variables-monom-mult-poly[OF m-vars q])
  have tail: variables-poly V (poly-mult p q)
    by (rule Cons.IH[OF p-vars])
  show ?case
    by (simp add: poly-mult.simps t variables-poly-add[OF head tail])
qed

lemma total-degree-poly-le-mult:
  assumes p: total-degree-poly-le p d and q: total-degree-poly-le q e
  shows total-degree-poly-le (poly-mult p q) (d + e)
  using p
proof (induction p)
  case Nil
  then show ?case
    by (simp add: poly-mult.simps total-degree-poly-le-def poly-degree-def)
next
  case (Cons t p)
  obtain m c where t:  $t = (m, c)$ 
    by force
  have mc:  $(m, c) \in \text{set } (t \# p)$ 
    by (simp add: t)
  have m-degree: monom-degree  $m \leq d$ 
    by (rule total-degree-poly-leD[OF Cons.prem1 mc])
  have p-degree: total-degree-poly-le p d

```

```

    using Cons.premis t
    by (auto simp: total-degree-poly-le-def poly-degree-def)
  have head:
    total-degree-poly-le (monom-mult-poly (m, c) q) (d + e)
    by (rule total-degree-monom-mult-poly[OF m-degree q])
  have tail: total-degree-poly-le (poly-mult p q) (d + e)
    by (rule Cons.IH[OF p-degree])
  have total-degree-poly-le
    (poly-add (monom-mult-poly (m, c) q) (poly-mult p q)) (d + e)
    by (rule total-degree-poly-le-add[OF head tail])
  then show ?case
    by (simp add: poly-mult.simps t)
qed

```

```

lemma variables-poly-minus:
  assumes p: variables-poly V p and q: variables-poly V q
  shows variables-poly V (poly-minus p q)
proof -
  have neg: variables-poly V (monom-mult-poly (1, -1) q)
    by (rule variables-monom-mult-poly[OF - q])
    (simp add: monom-vars-def)
  show ?thesis
    unfolding poly-minus-def
    by (rule variables-poly-add[OF p neg])
qed

```

```

lemma total-degree-poly-le-minus:
  assumes p: total-degree-poly-le p d and q: total-degree-poly-le q d
  shows total-degree-poly-le (poly-minus p q) d
proof -
  have neg': total-degree-poly-le (monom-mult-poly (1, -1) q) (0 + d)
    by (rule total-degree-monom-mult-poly[OF - q]) simp
  have neg: total-degree-poly-le (monom-mult-poly (1, -1) q) d
    using neg' by simp
  show ?thesis
    unfolding poly-minus-def
    by (rule total-degree-poly-le-add[OF p neg])
qed

```

```

lemma variables-poly-power:
  assumes p: variables-poly V p
  shows variables-poly V (poly-power p k)
proof (induction k)
  case 0
  then show ?case
    by (simp add: poly-power.simps one-poly-def variables-poly-def poly-vars-def)
next
  case (Suc k)
  show ?case

```

unfolding *poly-power.simps*
 by (rule *variables-poly-mult[OF p Suc.IH]*)
 qed

lemma *total-degree-poly-le-power*:
 assumes *p*: *total-degree-poly-le p d*
 shows *total-degree-poly-le (poly-power p k) (k * d)*
proof (*induction k*)
 case 0
 then show ?case
 by (simp add: *poly-power.simps one-poly-def*
 total-degree-poly-le-def poly-degree-def)
 next
 case (*Suc k*)
 have *total-degree-poly-le (poly-mult p (poly-power p k)) (d + k * d)*
 by (rule *total-degree-poly-le-mult[OF p Suc.IH]*)
 then show ?case
 by (simp add: *poly-power.simps algebra-simps*)
 qed

lemma *eval-poly-prod*:
eval-poly α (poly-prod ps) = prod-list (map (eval-poly α) ps)
 by (*induction ps*) *simp-all*

lemma *variables-poly-prod*:
 assumes *list-all (variables-poly V) ps*
 shows *variables-poly V (poly-prod ps)*
 using *assms*
proof (*induction ps*)
 case Nil
 then show ?case
 by (simp add: *one-poly-def variables-poly-def poly-vars-def*)
 next
 case (*Cons p ps*)
 have *p*: *variables-poly V p* and *ps*: *list-all (variables-poly V) ps*
 using *Cons.prem*s by *simp-all*
 show ?case
 by (simp add: *variables-poly-mult[OF p Cons.IH[OF ps]]*)
 qed

lemma *total-degree-poly-prod*:
 assumes *list-all2 total-degree-poly-le ps ds*
 shows *total-degree-poly-le (poly-prod ps) (sum-list ds)*
 using *assms*
proof (*induction ps arbitrary: ds*)
 case Nil
 then show ?case
 by (simp add: *one-poly-def total-degree-poly-le-def poly-degree-def*)
 next

```

case (Cons p ps)
then obtain d ds' where ds: ds = d # ds'
  by (cases ds) auto
have p: total-degree-poly-le p d
  using Cons.premis ds by simp
have rest: list-all2 total-degree-poly-le ps ds'
  using Cons.premis ds by simp
have total-degree-poly-le (poly-mult p (poly-prod ps)) (d + sum-list ds')
  by (rule total-degree-poly-le-mult[OF p Cons.IH[OF rest]])
then show ?case
  by (simp add: ds)
qed

```

```

lemma eval-indicator-factor:
  eval-poly  $\alpha$  (indicator-factor q p) = 1 - eval-poly  $\alpha$  p  $^{\wedge}(q - 1)$ 
  by (simp add: indicator-factor-def)

```

```

lemma eval-indicator-poly:
  eval-poly  $\alpha$  (indicator-poly q ps) =
    prod-list (map ( $\lambda p. 1 - \text{eval-poly } \alpha p ^{\wedge}(q - 1)$ ) ps)
  unfolding indicator-poly-def
  by (simp add: eval-poly-prod eval-indicator-factor comp-def)

```

```

lemma variables-indicator-factor:
  assumes variables-poly V p
  shows variables-poly V (indicator-factor q p)
  using assms
  by (simp add: indicator-factor-def variables-poly-minus variables-poly-power)

```

```

lemma variables-indicator-poly:
  assumes list-all (variables-poly V) ps
  shows variables-poly V (indicator-poly q ps)
proof -
  have list-all (variables-poly V) (map (indicator-factor q) ps)
    using assms
  by (auto simp: list-all-iff intro: variables-indicator-factor)
then show ?thesis
  unfolding indicator-poly-def
  by (rule variables-poly-prod)
qed

```

```

lemma total-degree-indicator-factor:
  fixes p :: ('v::linorder, 'a::comm-ring-1) poly
  assumes p: total-degree-poly-le p d
  shows total-degree-poly-le (indicator-factor q p) ((q - 1) * d)
proof -
  have pow: total-degree-poly-le (poly-power p (q - 1)) ((q - 1) * d)
    by (rule total-degree-poly-le-power[OF p])
  have const:

```

```

    total-degree-poly-le (poly-const 1 :: ('v, 'a) poly)
      ((q - 1) * d)
  by (rule total-degree-poly-le-mono[OF total-degree-poly-le-const]) simp
show ?thesis
  unfolding indicator-factor-def
  by (rule total-degree-poly-le-minus[OF const pow])
qed

```

```

lemma total-degree-indicator-poly:
  assumes list-all2 total-degree-poly-le ps ds
  shows total-degree-poly-le (indicator-poly q ps) ((q - 1) * sum-list ds)
  using assms
proof (induction ps arbitrary: ds)
  case Nil
  then show ?case
    by (simp add: indicator-poly-def one-poly-def
      total-degree-poly-le-def poly-degree-def)
next
  case (Cons p ps)
  then obtain d ds' where ds: ds = d # ds'
    by (cases ds) auto
  have p: total-degree-poly-le p d
    using Cons.prem1 ds by simp
  have rest: list-all2 total-degree-poly-le ps ds'
    using Cons.prem2 ds by simp
  have factor:
    total-degree-poly-le (indicator-factor q p) ((q - 1) * d)
  by (rule total-degree-indicator-factor[OF p])
  have tail:
    total-degree-poly-le (indicator-poly q ps) ((q - 1) * sum-list ds')
  by (rule Cons.IH[OF rest])
  have total-degree-poly-le
    (poly-mult (indicator-factor q p) (indicator-poly q ps))
    ((q - 1) * d + (q - 1) * sum-list ds')
  by (rule total-degree-poly-le-mult[OF factor tail])
  then show ?case
    by (simp add: indicator-poly-def ds algebra-simps)
qed

```

1.2 Finite-field power sums

```

lemma finite-field-card-gt-one:
  1 < card (UNIV :: 'a::finite-field set)
proof -
  have card ({0, 1} :: 'a set) ≤ card (UNIV :: 'a set)
    by (rule card-mono) auto
  then show ?thesis
    by simp
qed

```

```

lemma finite-field-power-card-minus-one:
  fixes  $x :: 'a::\text{finite-field}$ 
  assumes  $x \neq 0$ 
  shows  $x^{\text{card } (\text{UNIV} :: 'a \text{ set}) - 1} = 1$ 
proof -
  let  $?q = \text{card } (\text{UNIV} :: 'a \text{ set})$ 
  have  $q\text{-gt}: 1 < ?q$ 
    by (rule finite-field-card-gt-one)
  have  $x^?q = x$ 
    by (rule finite-field-power-card-eq-same)
  moreover have  $x^?q = x^{(?q - 1)} * x$ 
proof -
    have  $q\text{-suc}: ?q = \text{Suc } (?q - 1)$ 
      using  $q\text{-gt}$  by simp
    have  $x^?q = x^{\text{Suc } (?q - 1)}$ 
      using  $q\text{-suc}$  by simp
    also have  $\dots = x^{(?q - 1)} * x$ 
      by (rule power-Suc2)
    finally show ?thesis .
qed
  ultimately have  $x^{(?q - 1)} * x = 1 * x$ 
    by simp
  then show ?thesis
    using assms by simp
qed

```

```

lemma finite-field-exists-power-ne-one:
  fixes  $k :: \text{nat}$ 
  assumes  $k\text{-pos}: 0 < k$ 
  assumes  $k\text{-lt}: k < \text{card } (\text{UNIV} :: 'a::\text{finite-field} \text{ set}) - 1$ 
  shows  $\exists a::'a. a \neq 0 \wedge a^k \neq 1$ 
proof -
  let  $?P = \text{monom } (1::'a) \ k - 1$ 
  have  $P\text{-nonzero}: ?P \neq 0$ 
proof
    assume  $?P = 0$ 
    then have  $\text{poly.coeff } ?P \ k = 0$ 
      by simp
    moreover have  $\text{poly.coeff } ?P \ k = 1$ 
      using  $k\text{-pos}$  by simp
    ultimately show False
      by simp
qed
  have  $\text{deg-}P: \text{degree } ?P \leq k$ 
proof -
    have  $\text{deg-monom}: \text{degree } (\text{monom } (1::'a) \ k) \leq k$ 
      by (rule degree-monom-le)
    have  $\text{deg-one}: \text{degree } (1::'a \text{ Polynomial.poly}) \leq k$ 

```

```

    using k-pos by simp
  have degree ?P ≤
    max (degree (monom (1::'a) k)) (degree (1::'a Polynomial.poly))
  by (rule degree-diff-le-max)
  also have ... ≤ k
  using deg-monom deg-one by simp
  finally show ?thesis .
qed
have roots-bound: card {x::'a. poly ?P x = 0} ≤ k
proof -
  have card {x::'a. poly ?P x = 0} ≤ degree ?P
  by (rule poly-roots-degree[OF P-nonzero])
  also have ... ≤ k
  by (rule deg-P)
  finally show ?thesis .
qed
let ?NZ = {x::'a. x ≠ 0}
have card-NZ: card ?NZ = card (UNIV :: 'a set) - 1
proof -
  have ?NZ = (UNIV :: 'a set) - {0}
  by auto
  also have card ... = card (UNIV :: 'a set) - card ({0} :: 'a set)
  by (rule card-Diff-subset) auto
  finally show ?thesis
  by simp
qed
show ?thesis
proof (rule ccontr)
  assume ¬ ?thesis
  then have all-roots:  $\bigwedge x::'a. x \neq 0 \implies x^k = 1$ 
  by blast
  have NZ-roots: ?NZ ⊆ {x::'a. poly ?P x = 0}
  using all-roots by (auto simp: poly-monom)
  have card ?NZ ≤ card {x::'a. poly ?P x = 0}
  by (rule card-mono[OF poly-roots-finite[OF P-nonzero] NZ-roots])
  also have ... ≤ k
  by (rule roots-bound)
  finally show False
  using k-lt card-NZ by simp
qed
qed

lemma finite-field-power-sum-less:
  assumes k-lt:  $k < \text{card } (\text{UNIV} :: 'a::\text{finite-field set}) - 1$ 
  shows  $(\sum_{x \in \text{UNIV}} x^k :: 'a) = 0$ 
proof (cases k = 0)
  case True
  have (of-nat (card (UNIV :: 'a set)) :: 'a) = 0
  by (simp add: of-nat-eq-0-iff-char-dvd CHAR-dvd-CARD)

```

```

with True show ?thesis
  by simp
next
case False
then obtain a :: 'a where a0: a ≠ 0 and ak: a ^ k ≠ 1
  using finite-field-exists-power-ne-one[of k] k-lt by auto
let ?S = (∑ x ∈ UNIV. x ^ k :: 'a)
have ?S = (∑ x ∈ UNIV. (a * x) ^ k)
  by (rule sum.reindex-bij-witness[where i = λx. a * x and j = λx. x / a])
  (use a0 in auto)
also have ... = a ^ k * ?S
  by (simp add: power-mult-distrib sum-distrib-left)
finally have (a ^ k - 1) * ?S = 0
  by (simp add: algebra-simps)
with ak show ?thesis
  by simp
qed

lemma eval-monom-list-prod-sum-var:
  fixes α :: 'v::linorder ⇒ 'a::comm-semiring-1
  assumes finite: finite V and vars: fst ' set m ⊆ V
  shows eval-monom-list α m =
    (∏ x ∈ V. α x ^ sum-var-list m x)
  using assms
proof (induction m)
case Nil
then show ?case
  by (simp add: sum-var-list-def)
next
case (Cons ve m)
obtain v e where ve: ve = (v, e)
  by force
have vV: v ∈ V and mV: fst ' set m ⊆ V
  using Cons.premis by (auto simp: ve)
have delta-prod:
  (∏ x ∈ V. α x ^ (if x = v then e else 0)) = α v ^ e
proof -
  have (∏ x ∈ V. α x ^ (if x = v then e else 0)) =
    (∏ x ∈ V. if x = v then α x ^ e else 1)
  by (intro prod.cong) auto
  also have ... = α v ^ e
  using Cons.premis(1) vV
  by (simp add: prod.delta)
  finally show ?thesis .
qed
have (∏ x ∈ V. α x ^ sum-var-list (ve # m) x) =
  (∏ x ∈ V.
    α x ^ (if x = v then e else 0) * α x ^ sum-var-list m x)
  by (intro prod.cong)

```

```

    (simp-all add: ve sum-var-list-def power-add)
  also have ... =
    ( $\prod_{x \in V}. \alpha x \wedge (\text{if } x = v \text{ then } e \text{ else } 0)$ ) *
    ( $\prod_{x \in V}. \alpha x \wedge \text{sum-var-list } m x$ )
    by (rule prod.distrib)
  also have ... =  $\alpha v \wedge e * \text{eval-monom-list } \alpha m$ 
    by (simp add: delta-prod Cons.IH[OF Cons.prem1] mV)
  also have ... =  $\text{eval-monom-list } \alpha (ve \# m)$ 
    by (simp add: ve algebra-simps)
  finally show ?case
    by simp
qed

```

```

lemma eval-monom-prod-sum-var:
  fixes  $\alpha :: 'v::\text{linorder} \Rightarrow 'a::\text{comm-semiring-1}$ 
  assumes finite V monom-vars m  $\subseteq V$ 
  shows  $\text{eval-monom } \alpha m = (\prod_{x \in V}. \alpha x \wedge \text{sum-var } m x)$ 
  using assms
  unfolding monom-vars-def
  by transfer (auto intro: eval-monom-list-prod-sum-var)

```

```

lemma monom-list-degree-sum-var:
  fixes m :: 'v::linorder monom-list
  assumes finite V fst 'set m  $\subseteq V$ 
  shows  $\text{monom-list-degree } m = (\sum_{x \in V}. \text{sum-var-list } m x)$ 
  using assms
proof (induction m)
  case Nil
  then show ?case
    by (simp add: monom-list-degree-def sum-var-list-def)
next
  case (Cons ve m)
  obtain v e where ve: ve = (v, e)
    by force
  have vV: v  $\in V$  and mV: fst 'set m  $\subseteq V$ 
    using Cons.prem1 by (auto simp: ve)
  have IH:
     $\text{monom-list-degree } m = (\sum_{x \in V}. \text{sum-var-list } m x)$ 
    by (rule Cons.IH[OF Cons.prem1] mV)
  have delta:  $(\sum_{x \in V}. \text{if } x = v \text{ then } e \text{ else } 0) = e$ 
    using Cons.prem1 vV
    by (simp add: sum.delta)
  have monom-list-degree (ve # m) = e +  $\text{monom-list-degree } m$ 
    by (simp add: ve monom-list-degree-def)
  also have ... = e +  $(\sum_{x \in V}. \text{sum-var-list } m x)$ 
    using IH by simp
  also have ... =
     $(\sum_{x \in V}. \text{if } x = v \text{ then } e \text{ else } 0) +$ 
     $(\sum_{x \in V}. \text{sum-var-list } m x)$ 

```

using *delta* by *simp*
 also have ... =
 $(\sum x \in V. (\text{if } x = v \text{ then } e \text{ else } 0) + \text{sum-var-list } m \ x)$
 by (*simp add: sum.distrib*)
 also have ... = $(\sum x \in V. \text{sum-var-list } (ve \ \# \ m) \ x)$
 by (*intro sum.cong*) (*simp-all add: ve sum-var-list-def*)
 finally show ?case .
 qed

lemma *monom-degree-sum-var*:
 fixes $m :: 'v::\text{linorder monom}$
 assumes *finite* V *monom-vars* $m \subseteq V$
 shows *monom-degree* $m = (\sum x \in V. \text{sum-var } m \ x)$
 using *assms*
 unfolding *monom-vars-def*
 by *transfer* (*auto intro: monom-list-degree-sum-var*)

lemma *sum-monom-assignments*:
 fixes $m :: 'v::\text{linorder monom}$
 assumes *finite*: *finite* V **and** *vars*: *monom-vars* $m \subseteq V$
 shows $(\sum \alpha \in \text{PiE } V \ (\lambda \cdot. \text{UNIV} :: 'a::\text{finite-field set}).$
 $\text{eval-monom } \alpha \ m) =$
 $(\prod x \in V. \sum y \in (\text{UNIV} :: 'a \text{ set}). y \wedge \text{sum-var } m \ x)$
proof –
 have $(\prod x \in V. \sum y \in (\text{UNIV} :: 'a \text{ set}). y \wedge \text{sum-var } m \ x) =$
 $(\sum \alpha \in \text{PiE } V \ (\lambda \cdot. \text{UNIV}).$
 $\prod x \in V. \alpha \ x \wedge \text{sum-var } m \ x)$
 by (*rule prod-sum-PiE[OF finite]*) *simp*
 then show ?thesis
 by (*simp add: eval-monom-prod-sum-var[OF finite vars]*)
 qed

lemma *exists-variable-degree-lt*:
 assumes *finite*: *finite* V
 assumes $(\sum x \in V. f \ x) < \text{card } V * k$
 shows $\exists x \in V. f \ x < k$
proof (*rule ccontr*)
 assume $\neg ?thesis$
 then have $ge: \bigwedge x. x \in V \implies k \leq f \ x$
 by *force*
 have $\text{card } V * k = (\sum x \in V. k)$
 using *finite* by *simp*
 also have $\dots \leq (\sum x \in V. f \ x)$
 by (*rule sum-mono*) (*use ge in auto*)
 finally show *False*
 using *assms* by *simp*
 qed

lemma *monomial-sum-zero-if-degree-lt*:

```

fixes  $m :: 'v::\text{linorder monom}$ 
assumes  $\text{finite: finite } V$  and  $\text{vars: monom-vars } m \subseteq V$ 
assumes  $\text{deg:}$ 
   $\text{monom-degree } m <$ 
   $\text{card } V * (\text{card } (UNIV :: 'a::\text{finite-field set}) - 1)$ 
shows  $(\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}). \text{eval-monom } \alpha \ m) = 0$ 
proof -
  have  $\exists x \in V. \text{sum-var } m \ x < \text{card } (UNIV :: 'a \text{ set}) - 1$ 
  proof ( $\text{rule exists-variable-degree-lt[OF finite]}$ )
    show  $(\sum x \in V. \text{sum-var } m \ x) <$ 
       $\text{card } V * (\text{card } (UNIV :: 'a \text{ set}) - 1)$ 
    using  $\text{deg monom-degree-sum-var[OF finite vars]}$  by  $\text{simp}$ 
  qed
  then obtain  $x$  where  $xV: x \in V$ 
  and  $x\text{-lt: sum-var } m \ x < \text{card } (UNIV :: 'a \text{ set}) - 1$ 
  by  $\text{blast}$ 
  have  $\text{factor-zero:}$ 
     $(\sum y \in UNIV. (y::'a) \wedge \text{sum-var } m \ x) = 0$ 
  by ( $\text{rule finite-field-power-sum-less[OF x-lt]}$ )
  have  $(\prod x \in V. \sum y \in (UNIV :: 'a \text{ set}). y \wedge \text{sum-var } m \ x) = 0$ 
  proof ( $\text{rule prod-zero[OF finite]}$ )
    show  $\exists x \in V. (\sum y \in (UNIV :: 'a \text{ set}). y \wedge \text{sum-var } m \ x) = 0$ 
    using  $xV \text{ factor-zero}$  by  $\text{blast}$ 
  qed
  then show  $?thesis$ 
  using  $\text{sum-monom-assignments[OF finite vars, where 'a='a]}$  by  $\text{simp}$ 
qed

lemma  $\text{finite-field-sum-poly-zero:}$ 
assumes  $\text{finite: finite } V$  and  $\text{vars: variables-poly } V \ p$ 
assumes  $\text{deg:}$ 
   $\text{total-degree-poly-less } p$ 
   $(\text{card } V * (\text{card } (UNIV :: 'a::\text{finite-field set}) - 1))$ 
shows  $(\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}). \text{eval-poly } \alpha \ p) = 0$ 
using  $\text{vars deg}$ 
proof ( $\text{induction } p$ )
  case  $Nil$ 
  then show  $?case$ 
  by  $\text{simp}$ 
next
  case  $(\text{Cons } t \ p)$ 
  obtain  $m \ c$  where  $t: t = (m, c)$ 
  by  $\text{force}$ 
  have  $m\text{-vars: monom-vars } m \subseteq V$ 
  using  $\text{monom-vars-subset-poly-vars[of m c t \# p]}$   $\text{Cons.prem1(1) } t$ 
  by ( $\text{auto simp: variables-poly-def}$ )
  have  $mc: (m, c) \in \text{set } (t \# p)$ 
  by ( $\text{simp add: } t$ )
  have  $m\text{-deg:}$ 

```

```

    monom-degree  $m <$ 
      card  $V * (\text{card } (UNIV :: 'a \text{ set}) - 1)$ 
  by (rule total-degree-poly-lessD[OF Cons.prem(2) mc])
have p-vars: variables-poly  $V \ p$ 
  using Cons.prem(1)
  by (auto simp: variables-poly-def poly-vars-def)
have p-deg:
  total-degree-poly-less  $p$ 
    (card  $V * (\text{card } (UNIV :: 'a \text{ set}) - 1)$ )
  using Cons.prem(2)  $t$ 
  by (auto simp: total-degree-poly-less-def poly-degree-def)
have head:
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}). \text{eval-monom } \alpha \ m * c$ ) = 0
proof -
  have ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}).$ 
    eval-monom  $\alpha \ m * c$ ) =
    ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}). \text{eval-monom } \alpha \ m$ ) *  $c$ 
  by (simp add: sum-distrib-right)
  also have ... = 0
  by (simp add: monomial-sum-zero-if-degree-lt[OF finite m-vars m-deg])
  finally show ?thesis .
qed
have ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}).$ 
  eval-poly  $\alpha \ (t \ \# \ p)$ ) =
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}).$ 
    eval-monom  $\alpha \ m * c + \text{eval-poly } \alpha \ p$ )
  by (intro sum.cong) (simp-all add: t)
also have ... =
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}).$ 
    eval-monom  $\alpha \ m * c$ ) +
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. UNIV :: 'a \text{ set}). \text{eval-poly } \alpha \ p$ )
  by (simp add: sum.distrib)
also have ... = 0
  using head Cons.IH[OF p-vars p-deg] by simp
  finally show ?case .
qed

```

1.3 The theorem

```

lemma finite-field-indicator-factor-value:
  fixes  $y :: 'a :: \text{finite-field}$ 
  shows  $1 - y \wedge (\text{card } (UNIV :: 'a \text{ set}) - 1) = \text{of-bool } (y = 0)$ 
proof (cases  $y = 0$ )
case True
  have  $0 < \text{card } (UNIV :: 'a \text{ set}) - 1$ 
  using finite-field-card-gt-one[where  $'a = 'a$ ] by simp
  with True show ?thesis
  by (simp add: of-bool-def)
next

```

```

case False
then have  $y \wedge (\text{card } (UNIV :: 'a \text{ set}) - 1) = 1$ 
  by (rule finite-field-power-card-minus-one)
with False show ?thesis
  by (simp add: of-bool-def)
qed

lemma prod-list-of-bool-list-all:
  prod-list (map ( $\lambda x. \text{of-bool } (P \ x) :: 'a::\text{comm-semiring-1}$ ) xs) =
    of-bool (list-all P xs)
  by (induction xs) (simp-all add: of-bool-def)

lemma eval-indicator-poly-value:
  fixes ps :: ('v::linorder, 'a::finite-field) poly list
  shows eval-poly  $\alpha$  (indicator-poly ( $\text{card } (UNIV :: 'a \text{ set})$ ) ps) =
    of-bool (list-all ( $\lambda p. \text{eval-poly } \alpha \ p = 0$ ) ps)
proof -
  have eval-poly  $\alpha$  (indicator-poly ( $\text{card } (UNIV :: 'a \text{ set})$ ) ps) =
    prod-list
      (map ( $\lambda p. 1 - \text{eval-poly } \alpha \ p \wedge (\text{card } (UNIV :: 'a \text{ set}) - 1)$ ) ps)
    by (rule eval-indicator-poly)
  also have ... =
    prod-list (map ( $\lambda p. \text{of-bool } (\text{eval-poly } \alpha \ p = 0) :: 'a$ ) ps)
  proof (rule arg-cong[where f=prod-list])
    show map ( $\lambda p. 1 - \text{eval-poly } \alpha \ p \wedge (\text{card } (UNIV :: 'a \text{ set}) - 1)$ ) ps =
      map ( $\lambda p. \text{of-bool } (\text{eval-poly } \alpha \ p = 0) :: 'a$ ) ps
    by (intro map-cong refl finite-field-indicator-factor-value)
  qed
  also have ... = of-bool (list-all ( $\lambda p. \text{eval-poly } \alpha \ p = 0$ ) ps)
    by (rule prod-list-of-bool-list-all)
  finally show ?thesis .
qed

lemma list-all2-variables:
  assumes
    list-all2 ( $\lambda p \ d. \text{variables-poly } V \ p \wedge \text{total-degree-poly-le } p \ d$ ) ps ds
  shows list-all (variables-poly V) ps
  using assms
proof (induction ps arbitrary: ds)
  case Nil
  then show ?case
    by simp
next
  case (Cons p ps)
  then obtain d ds' where ds: ds = d # ds'
    by (cases ds) auto
  then show ?case
    using Cons.IH[of ds] Cons.prems by auto
qed

```

```

lemma list-all2-degrees:
  assumes
    list-all2 ( $\lambda p d. \text{variables-poly } V p \wedge \text{total-degree-poly-le } p d$ ) ps ds
  shows list-all2 total-degree-poly-le ps ds
  using assms
proof (induction ps arbitrary: ds)
  case Nil
  then show ?case
    by (cases ds) auto
next
  case (Cons p ps)
  then obtain d ds' where
    ds:  $ds = d \# ds'$ 
    and p: total-degree-poly-le p d
    and rest:
      list-all2
        ( $\lambda p d. \text{variables-poly } V p \wedge \text{total-degree-poly-le } p d$ ) ps ds'
    by (cases ds) auto
  have list-all2 total-degree-poly-le ps ds'
    by (rule Cons.IH[OF rest])
  with p show ?case
    by (simp add: ds)
qed

```

```

theorem chevalley-warning:
  fixes V :: 'v::linorder set
    and ps :: ('v, 'a::finite-field) poly list
  assumes finite: finite V
  assumes polys:
    list-all2 ( $\lambda p d. \text{variables-poly } V p \wedge \text{total-degree-poly-le } p d$ ) ps ds
  assumes degree-sum: sum-list ds < card V
  shows CHAR('a) dvd card (common-zeros V ps)
proof -
  let ?q = card (UNIV :: 'a set)
  let ?I = indicator-poly ?q ps
  have q-minus-pos:  $0 < ?q - 1$ 
    using finite-field-card-gt-one[where 'a='a] by simp
  have variables: list-all (variables-poly V) ps
    by (rule list-all2-variables[OF polys])
  have I-variables: variables-poly V ?I
    by (rule variables-indicator-poly[OF variables])
  have degrees: list-all2 total-degree-poly-le ps ds
    by (rule list-all2-degrees[OF polys])
  have I-degree-le:
    total-degree-poly-le ?I ((?q - 1) * sum-list ds)
    by (rule total-degree-indicator-poly[OF degrees])
  have degree-bound:
     $(?q - 1) * \text{sum-list } ds < \text{card } V * (?q - 1)$ 

```

```

using degree-sum q-minus-pos
by (simp add: mult-less-cancel1)
have I-degree-less:
  total-degree-poly-less ?I (card V * (?q - 1))
by (rule total-degree-poly-less-of-le[OF I-degree-le degree-bound])
have sum-zero:
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}). \text{eval-poly } \alpha \text{ ?I} = 0$ )
by (rule finite-field-sum-poly-zero[OF finite I-variables I-degree-less])
have finite-assignments: finite (PiE V ( $\lambda \cdot. \text{UNIV} :: 'a \text{ set}$ ))
by (rule finite-PiE[OF finite]) simp
have sum-card:
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}). \text{eval-poly } \alpha \text{ ?I} =$ 
    of-nat (card (common-zeros V ps)))
proof -
have ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}). \text{eval-poly } \alpha \text{ ?I} =$ 
  ( $\sum \alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}).$ 
    of-bool (list-all ( $\lambda p. \text{eval-poly } \alpha \text{ } p = 0$ ) ps) :: 'a))
by (intro sum.cong)
  (simp-all add: eval-indicator-poly-value)
also have ... =
  of-nat (card { $\alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}).$ 
    list-all ( $\lambda p. \text{eval-poly } \alpha \text{ } p = 0$ ) ps})
proof -
have (PiE V ( $\lambda \cdot. \text{UNIV} :: 'a \text{ set}$ )  $\cap$ 
  { $\alpha. \text{list-all } (\lambda p. \text{eval-poly } \alpha \text{ } p = 0) \text{ ps}$ }) =
  { $\alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}).$ 
    list-all ( $\lambda p. \text{eval-poly } \alpha \text{ } p = 0$ ) ps}
by auto
with finite-assignments show ?thesis
by simp
qed
also have { $\alpha \in \text{PiE } V (\lambda \cdot. \text{UNIV} :: 'a \text{ set}).$ 
  list-all ( $\lambda p. \text{eval-poly } \alpha \text{ } p = 0$ ) ps} = common-zeros V ps
by (auto simp: common-zeros-def)
finally show ?thesis .
qed
have (of-nat (card (common-zeros V ps)) :: 'a) = 0
using sum-zero sum-card by simp
then show ?thesis
by (simp add: of-nat-eq-0-iff-char-dvd)
qed

corollary chevalley-warning-finite-type:
fixes ps :: ('v::{finite, linorder}, 'a::finite-field) poly list
assumes polys: list-all2 total-degree-poly-le ps ds
assumes degree-sum: sum-list ds < card (UNIV :: 'v set)
shows CHAR('a) dvd card (common-zeros UNIV ps)
proof (rule chevalley-warning[where ds=ds])
show finite (UNIV :: 'v set)

```

```

    by simp
show list-all2
  ( $\lambda p d. \text{variables-poly } UNIV p \wedge \text{total-degree-poly-le } p d$ ) ps ds
  using polys
  by (rule list-all2-mono) simp
show sum-list ds < card (UNIV :: 'v set)
  using degree-sum by simp
qed

end

```

References

- [1] C. Chevalley. Démonstration d’une hypothèse de M. Artin. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 11(1):73–75, 1935. DOI: <https://doi.org/10.1007/BF02940714>.
- [2] C. Sternagel, R. Thiemann, A. Maletzky, F. Immler, F. Haftmann, A. Lochbihler, and A. Bentkamp. Executable multivariate polynomials. *Archive of Formal Proofs*, August 2010. <https://isa-afp.org/entries/Polynomials.html>, Formal proof development.
- [3] E. Warning. Bemerkung zur vorstehenden Arbeit von Herrn Chevalley. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 11(1):76–83, 1935. DOI: <https://doi.org/10.1007/BF02940715>.