Hilbert's Nullstellensatz

Alexander Maletzky*

March 17, 2025

Abstract

This entry formalizes Hilbert's Nullstellensatz, an important theorem in algebraic geometry that can be viewed as the generalization of the Fundamental Theorem of Algebra to multivariate polynomials: If a set of (multivariate) polynomials over an algebraically closed field has no common zero, then the ideal it generates is the entire polynomial ring. The formalization proves several equivalent versions of this celebrated theorem: the weak Nullstellensatz, the strong Nullstellensatz (connecting algebraic varieties and radical ideals), and the field-theoretic Nullstellensatz. The formalization follows Chapter 4.1. of *Ideals, Varieties, and Algorithms* by Cox, Little and O'Shea.

Contents

*Funded by the Austrian Science Fund (FWF): grant no. P 29498-N31

1 Algebraically Closed Fields

```
theory Algebraically-Closed-Fields
  imports HOL−Computational-Algebra.Fundamental-Theorem-Algebra
begin

lemma prod-eq-zeroE:
 assumes prod f I = (0::'a::{semiring-no-zero-divisors,comm-monoid-mult,zero-neq-one})
  obtains i where finite I and i ∈ I and f i = 0
proof −
  have finite I
  proof (rule ccontr)
    assume infinite I
    with assms show False by simp
  qed
  moreover from this assms obtain i where i ∈ I and f i = 0
  proof (induct I arbitrary: thesis)
    case empty
    from empty(2) show ?case by simp
  next
    case (insert j I)
    from insert.hyps(1, 2) have f j ∗ prod f I = prod f (insert j I) by simp
    also have . . . = 0 by fact
    finally have f j = 0 ∨ prod f I = 0 by simp
    thus ?case
    proof
      assume f j = 0
      with - show ?thesis by (rule insert.prems) simp
    next
      assume prod f I = 0
      then obtain i where i ∈ I and f i = 0 using insert.hyps(3) by blast
      from - this(2) show ?thesis by (rule insert.prems) (simp add: ‹i ∈ I›)
    qed
  qed
  ultimately show ?thesis ..
qed

lemma degree-prod-eq:
  assumes finite I and ⋀i. i ∈ I =⇒ f i ≠ 0
 shows Polynomial.degree (prod f I :: -::semiring-no-zero-divisors poly) = (∑ i∈I.
Polynomial.degree (f i))
  using assms
proof (induct I)
  case empty
  show ?case by simp
next
  case (insert j J)
  have 1: f i ≠ 0 if i ∈ J for i
  proof (rule insert.prems)
```

from *that* **show** $i \in$ *insert j J* **by** *simp*
qed
hence *eq*: *Polynomial.degree* (*prod f J*) = ($\sum$ *i*∈*J*. *Polynomial.degree* (*f i*)) **by** (*rule insert.hyps*)
from *insert.hyps*(*1*, *2*) **have** *Polynomial.degree* (*prod f* (*insert j J*)) = *Polynomial.degree* (*f j* ∗ *prod f J*)
by *simp*
also have ... = *Polynomial.degree* (*f j*) + *Polynomial.degree* (*prod f J*)
proof (*rule degree-mult-eq*)
show *f j* ≠ *0* **by** (*rule insert.prems*) *simp*
next
show *prod f J* ≠ *0*
proof
assume *prod f J* = *0*
then obtain *i* **where** *i* ∈ *J* **and** *f i* = *0* **by** (*rule prod-eq-zeroE*)
from *this*(*1*) **have** *f i* ≠ *0* **by** (*rule 1*)
thus *False* **using** ‹*f i* = *0*› **..**
qed
qed
also from *insert.hyps*(*1*, *2*) **have** ... = ($\sum$ *i*∈*insert j J*. *Polynomial.degree* (*f i*)) **by** (*simp add*: *eq*)
finally show *?case* **.**
qed

class *alg-closed-field* =
assumes *alg-closed-field-axiom*: ⋀*p*::′*a*::*field poly*. *0* < *Polynomial.degree p* ⟹ ∃ *z*. *poly p z* = *0*
begin

lemma *rootE*:
assumes *0* < *Polynomial.degree p*
obtains *z* **where** *poly p z* = (*0*::′*a*)
proof −
from *assms* **have** ∃ *z*. *poly p z* = *0* **by** (*rule alg-closed-field-axiom*)
then obtain *z* **where** *poly p z* = *0* **..**
thus *?thesis* **..**
qed

lemma *infinite-UNIV*: *infinite* (*UNIV*::′*a set*)
proof
assume *fin*: *finite* (*UNIV*::′*a set*)
define *p* **where** *p* = ($\prod$ *a*∈*UNIV*. [:− *a*, *1*::′*a*:]) + [:−*1*:]
have *Polynomial.degree* ($\prod$ *a*∈*UNIV*. [:− *a*, *1*::′*a*:]) = ($\sum$ *a*∈*UNIV*. *Polynomial.degree* [:− *a*, *1*::′*a*:])
using *fin* **by** (*rule degree-prod-eq*) *simp*
also have ... = ($\sum$ *a*∈(*UNIV*::′*a set*). *1*) **by** *simp*
also have ... = *card* (*UNIV*::′*a set*) **by** *simp*
also from *fin* **have** ... > *0* **by** (*rule finite-UNIV-card-ge-0*)
finally have *0* < *Polynomial.degree* ($\prod$ *a*∈*UNIV*. [:− *a*, *1*::′*a*:]) **.**

hence *Polynomial.degree* [:−*1*:] < *Polynomial.degree* (∏ *a*∈*UNIV*. [:− *a*, *1*::′*a*:])
by *simp*
hence *Polynomial.degree p* = *Polynomial.degree* (∏ *a*∈*UNIV*. [:− *a*, *1*::′*a*:]) **unfolding** *p-def*
by (*rule degree-add-eq-left*)
also have . . . > *0* **by** *fact*
finally have *0* < *Polynomial.degree p* **.**
then obtain *z* **where** *poly p z* = *0* **by** (*rule rootE*)
hence (∏ *a*∈*UNIV*. (*z* − *a*)) = *1* **by** (*simp add*: *p-def poly-prod*)
thus *False* **by** (*metis UNIV-I cancel-comm-monoid-add-class.diff-cancel fin one-neq-zero prod-zero-iff*)
qed

lemma *linear-factorsE*:
fixes *p* :: ′*a poly*
obtains *c A m* **where** *finite A* **and** *p* = *Polynomial.smult c* (∏ *a*∈*A*. [:− *a*, *1*:] ^ *m a*)
and ⋀*a*. *m a* = *0* ⟷ *a* ∉ *A* **and** *c* = *0* ⟷ *p* = *0* **and** ⋀*z*. *poly p z* = *0* ⟷ (*c* = *0* ∨ *z* ∈ *A*)
proof −
obtain *c A m* **where** *fin*: *finite A* **and** *p*: *p* = *Polynomial.smult c* (∏ *a*∈*A*. [:− *a*, *1*:] ^ *m a*)
and ∗: ⋀*x*. *m x* = *0* ⟷ *x* ∉ *A*
proof (*induct p arbitrary*: *thesis rule*: *poly-root-induct*[**where** *P*=λ-. *True*])
case *0*
show *?case*
proof (*rule 0*)
show *0* = *Polynomial.smult 0* (∏ *a*∈{}. [:− *a*, *1*:] ^ (λ-. *0*) *a*) **by** *simp*
qed *simp-all*
next
case (*no-roots p*)
have *Polynomial.degree p* = *0*
proof (*rule ccontr*)
assume *Polynomial.degree p* ≠ *0*
hence *0* < *Polynomial.degree p* **by** *simp*
then obtain *z* **where** *poly p z* = *0* **by** (*rule rootE*)
moreover have *poly p z* ≠ *0* **by** (*rule no-roots*) *blast*
ultimately show *False* **by** *simp*
qed
then obtain *c* **where** *p*: *p* = [:*c*:] **by** (*rule degree-eq-zeroE*)
show *?case*
proof (*rule no-roots*)
show *p* = *Polynomial.smult c* (∏ *a*∈{}. [:− *a*, *1*:] ^ (λ-. *0*) *a*) **by** (*simp add*: *p*)
qed *simp-all*
next
case (*root a p*)
obtain *A c m* **where** *1*: *finite A* **and** *p*: *p* = *Polynomial.smult c* (∏ *a*∈*A*. [:− *a*, *1*:] ^ *m a*)

and *2*: $\bigwedge x.\ m\ x = 0 \longleftrightarrow x \notin A$ **by** (*rule root.hyps*) *blast*
define m' **where** $m' = (\lambda x.\ \mathit{if}\ x = a\ \mathit{then}\ \mathit{Suc}\ (m\ x)\ \mathit{else}\ m\ x)$
show *?case*
proof (*rule root.prems*)
from *1* **show** *finite* (*insert a A*) **by** *simp*
next
have $[:a, -\ 1:] * p = [:-\ a, 1:] * (-\ p)$ **by** *simp*
also have $\ldots = [:-\ a, 1:] * (\mathit{Polynomial.smult}\ (-\ c)\ (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a))$
by (*simp add*: *p*)
also have $\ldots = \mathit{Polynomial.smult}\ (-\ c)\ ([:-\ a, 1:] * (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a))$
by (*simp only*: *mult-smult-right ac-simps*)
also have $[:-\ a, 1:] * (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a) = (\prod a{\in}\mathit{insert}\ a\ A.\ [:-\ a, 1:]\ \hat{}\ m'\ a)$
proof (*cases* $a \in A$)
case *True*
with *1* **have** $(\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a) = [:-\ a, 1:]\ \hat{}\ m\ a * (\prod a{\in}A-\{a\}.\ [:-\ a, 1:]\ \hat{}\ m\ a)$
by (*simp add*: *prod.remove*)
also from *refl* **have** $(\prod a{\in}A-\{a\}.\ [:-\ a, 1:]\ \hat{}\ m\ a) = (\prod a{\in}A-\{a\}.\ [:-\ a, 1:]\ \hat{}\ m'\ a)$
by (*rule prod.cong*) (*simp add*: *m'-def*)
finally have $[:-\ a, 1:] * (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a) =$
$([:-\ a, 1:] * [:-\ a, 1:]\ \hat{}\ m\ a) * (\prod a{\in}A - \{a\}.\ [:-\ a, 1:]\ \hat{}\ m'\ a)$
by (*simp only*: *mult.assoc*)
also have $[:-\ a, 1:] * [:-\ a, 1:]\ \hat{}\ m\ a = [:-\ a, 1:]\ \hat{}\ m'\ a$ **by** (*simp add*: *m'-def*)
finally show *?thesis* **using** *1* **by** (*simp add*: *prod.insert-remove*)
next
case *False*
with *1* **have** $(\prod a{\in}\mathit{insert}\ a\ A.\ [:-\ a, 1:]\ \hat{}\ m'\ a) = [:-\ a, 1:]\ \hat{}\ m'\ a * (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m'\ a)$
by *simp*
also from *refl* **have** $(\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m'\ a) = (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a)$
proof (*rule prod.cong*)
fix *x*
assume $x \in A$
with *False* **have** $x \neq a$ **by** *blast*
thus $[:-\ x, 1:]\ \hat{}\ m'\ x = [:-\ x, 1:]\ \hat{}\ m\ x$ **by** (*simp add*: *m'-def*)
qed
finally have $(\prod a{\in}\mathit{insert}\ a\ A.\ [:-\ a, 1:]\ \hat{}\ m'\ a) = [:-\ a, 1:]\ \hat{}\ m'\ a * (\prod a{\in}A.\ [:-\ a, 1:]\ \hat{}\ m\ a)$ **.**
also from *False* **have** $m'\ a = 1$ **by** (*simp add*: *m'-def 2*)
finally show *?thesis* **by** *simp*
qed
finally show $[:a, -\ 1:] * p = \mathit{Polynomial.smult}\ (-\ c)\ (\prod a{\in}\mathit{insert}\ a\ A.\ [:-$

```
a, 1:] ^ m' a) .
    next
      fix x
      show m' x = 0 ⟷ x ∉ insert a A by (simp add: m'-def 2)
    qed
  qed
  moreover have c = 0 ⟷ p = 0
  proof
    assume p = 0
    hence [:c:] = 0 ∨ (∏ a∈A. [:− a, 1:] ^ m a) = 0 by (simp add: p)
    thus c = 0
    proof
      assume [:c:] = 0
      thus ?thesis by simp
    next
      assume (∏ a∈A. [:− a, 1:] ^ m a) = 0
      then obtain a where [:− a, 1:] ^ m a = 0 by (rule prod-eq-zeroE)
      thus ?thesis by simp
    qed
  qed (simp add: p)
  moreover {
    fix z
    have 0 < m z if z ∈ A by (rule ccontr) (simp add: ∗ that)
     hence poly p z = 0 ⟷ (c = 0 ∨ z ∈ A) by (auto simp: p poly-prod ∗ fin
elim: prod-eq-zeroE)
  }
  ultimately show ?thesis ..
qed

end

instance complex :: alg-closed-field
  by standard (rule fundamental-theorem-of-algebra, simp add: constant-degree)

end
```

2 Properties of the Lexicographic Order on Power-Products

```
theory Lex-Order-PP
  imports Polynomials.Power-Products
begin
```

We prove some useful properties of the purely lexicographic order relation on power-products.

```
lemma lex-pm-keys-leE:
  assumes lex-pm s t and x ∈ keys (s::'x::linorder ⇒₀ 'a::add-linorder-min)
  obtains y where y ∈ keys t and y ≤ x
```

```
  using assms(1) unfolding lex-pm-alt
proof (elim disjE exE conjE)
  assume s = t
  show ?thesis
  proof
    from assms(2) show x ∈ keys t by (simp only: ‹s = t›)
  qed (fact order.refl)
next
  fix y
  assume 1: lookup s y < lookup t y and 2: ∀ y'<y. lookup s y' = lookup t y'
  show ?thesis
  proof (cases y ≤ x)
    case True
    from zero-min 1 have 0 < lookup t y by (rule le-less-trans)
    hence y ∈ keys t by (simp add: dual-order.strict-implies-not-eq in-keys-iff)
    thus ?thesis using True ..
  next
    case False
    hence x < y by simp
    with 2 have lookup s x = lookup t x by simp
    with assms(2) have x ∈ keys t by (simp only: in-keys-iff not-False-eq-True)
    thus ?thesis using order.refl ..
  qed
qed

lemma lex-pm-except-max:
  assumes lex-pm s t and keys s ∪ keys t ⊆ {..x}
  shows lex-pm (except s {x}) (except t {x})
proof −
  from assms(1) have s = t ∨ (∃ x. lookup s x < lookup t x ∧ (∀ y<x. lookup s y
= lookup t y))
    by (simp only: lex-pm-alt)
  thus ?thesis
  proof (elim disjE exE conjE)
    assume s = t
    thus ?thesis by (simp only: lex-pm-refl)
  next
    fix y
    assume ∀ z<y. lookup s z = lookup t z
    hence eq: lookup s z = lookup t z if z < y for z using that by simp
    assume ∗: lookup s y < lookup t y
    hence y ∈ keys s ∪ keys t by (auto simp flip: lookup-not-eq-zero-eq-in-keys)
    with assms(2) have y ∈ {..x} ..
    hence y = x ∨ y < x by auto
    thus ?thesis
    proof
      assume y: y = x
      have except s {x} = except t {x}
      proof (rule poly-mapping-eqI)
```

```
      fix z
      show lookup (except s {x}) z = lookup (except t {x}) z
      proof (rule linorder-cases)
        assume z < y
        thus ?thesis by (simp add: lookup-except eq)
      next
        assume y < z
        hence z ∉ {..x} by (simp add: y)
        with assms(2) have z ∉ keys s and z ∉ keys t by blast+
        with ‹y < z› show ?thesis by (simp add: y lookup-except in-keys-iff)
      next
        assume z = y
        thus ?thesis by (simp add: y lookup-except)
      qed
    qed
    thus ?thesis by (simp only: lex-pm-refl)
  next
    assume y < x
    show ?thesis unfolding lex-pm-alt
    proof (intro disjI2 exI conjI allI impI)
      from ‹y < x› * show lookup (except s {x}) y < lookup (except t {x}) y
        by (simp add: lookup-except)
    next
      fix z
      assume z < y
      hence z < x using ‹y < x› by (rule less-trans)
      with ‹z < y› show lookup (except s {x}) z = lookup (except t {x}) z
        by (simp add: lookup-except eq)
    qed
  qed
 qed
qed

lemma lex-pm-strict-plus-left:
 assumes lex-pm-strict s t and ⋀x y. x ∈ keys t ⟹ y ∈ keys u ⟹ x < y
 shows lex-pm-strict (u + s) (t::- ⇒₀ 'a::add-linorder-min)
proof −
 from assms(1) obtain x where 1: lookup s x < lookup t x and 2: ⋀y. y < x
⟹ lookup s y = lookup t y
   by (auto simp: lex-pm-strict-def less-poly-mapping-def less-fun-def)
 from 1 have x ∈ keys t by (auto simp flip: lookup-not-eq-zero-eq-in-keys)
 have lookup-u: lookup u z = 0 if z ≤ x for z
 proof (rule ccontr)
   assume lookup u z ≠ 0
   hence z ∈ keys u by (simp add: in-keys-iff)
   with ‹x ∈ keys t› have x < z by (rule assms(2))
   with that show False by simp
 qed
 from 1 have lookup (u + s) x < lookup t x by (simp add: lookup-add lookup-u)
```

moreover have *lookup* (*u* + *s*) *y* = *lookup t y* **if** *y* < *x* **for** *y* **using** *that*
by (*simp add*: *lookup-add 2 lookup-u*)
ultimately show *?thesis* **by** (*auto simp*: *lex-pm-strict-def less-poly-mapping-def less-fun-def*)
qed

end

3 Polynomial Mappings and Univariate Polynomials

theory *Univariate-PM*
imports *HOL−Computational-Algebra.Polynomial Polynomials.MPoly-PM*
begin

3.1 Morphisms *pm-of-poly* and *poly-of-pm*

Many things in this section are copied from theory *Polynomials.MPoly-Type-Univariate*.

lemma *pm-of-poly-aux*:
{*t*. (*poly.coeff p* (*lookup t x*) *when t* ∈ .[{*x*}]) ≠ *0*} =
Poly-Mapping.single x ‘ {*d*. *poly.coeff p d* ≠ *0*} (**is** *?M* = -)
proof (*intro subset-antisym subsetI*)
fix *t*
assume *t* ∈ *?M*
hence ⋀*y*. *y* ≠ *x* ⟹ *Poly-Mapping.lookup t y* = *0* **by** (*fastforce simp*: *PPs-def in-keys-iff*)
hence *t* = *Poly-Mapping.single x* (*lookup t x*)
using *poly-mapping-eqI* **by** (*metis* (*full-types*) *lookup-single-eq lookup-single-not-eq*)
then show *t* ∈ (*Poly-Mapping.single x*) ‘ {*d*. *poly.coeff p d* ≠ *0*} **using** ‹*t* ∈ *?M*› **by** *auto*
qed (*auto split*: *if-splits simp*: *PPs-def*)

lift-definition *pm-of-poly* :: ′*x* ⇒ ′*a poly* ⇒ (′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*comm-monoid-add*
is λ*x p t*. (*poly.coeff p* (*lookup t x*)) *when t* ∈ .[{*x*}]
proof −
fix *x*::′*x* **and** *p*::′*a poly*
show *finite* {*t*. (*poly.coeff p* (*lookup t x*) *when t* ∈ .[{*x*}]) ≠ *0*} **unfolding** *pm-of-poly-aux*
using *finite-surj*[*OF MOST-coeff-eq-0*[*unfolded eventually-cofinite*]] **by** *blast*
qed

definition *poly-of-pm* :: ′*x* ⇒ ((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*) ⇒ ′*a*::*comm-monoid-add poly*
where *poly-of-pm x p* = *Abs-poly* (λ*d*. *lookup p* (*Poly-Mapping.single x d*))

lemma *lookup-pm-of-poly-single* [*simp*]:
lookup (*pm-of-poly x p*) (*Poly-Mapping.single x d*) = *poly.coeff p d*
by (*simp add*: *pm-of-poly.rep-eq PPs-closed-single*)

lemma *keys-pm-of-poly*: *keys* (*pm-of-poly x p*) = *Poly-Mapping.single x* ‘ {*d. poly.coeff p d* ≠ *0*}
proof −
have *keys* (*pm-of-poly x p*) = {*t.* (*poly.coeff p* (*lookup t x*) *when t* ∈ .[{*x*}]) ≠ *0*}
by (*rule set-eqI*) (*simp add*: *pm-of-poly.rep-eq flip*: *lookup-not-eq-zero-eq-in-keys*)
also have ... = *Poly-Mapping.single x* ‘ {*d. poly.coeff p d* ≠ *0*} **by** (*fact pm-of-poly-aux*)
finally show *?thesis* **.**
qed

lemma *coeff-poly-of-pm* [*simp*]: *poly.coeff* (*poly-of-pm x p*) *k* = *lookup p* (*Poly-Mapping.single x k*)
proof −
have *0*:*Poly-Mapping.single x* ‘ {*d. lookup p* (*Poly-Mapping.single x d*) ≠ *0*} ⊆ {*d. lookup p d* ≠ *0*}
by *auto*
have $\forall_\infty$ *k. lookup p* (*Poly-Mapping.single x k*) = *0* **unfolding** *coeff-def eventually-cofinite*
using *finite-imageD*[*OF finite-subset*[*OF 0 Poly-Mapping.finite-lookup*]] *inj-single*
by (*metis inj-eq inj-onI*)
then show *?thesis* **by** (*simp add*: *poly-of-pm-def Abs-poly-inverse*)
qed

lemma *pm-of-poly-of-pm*:
assumes *p* ∈ *P*[{*x*}]
shows *pm-of-poly x* (*poly-of-pm x p*) = *p*
proof (*rule poly-mapping-eqI*)
fix *t*
from *assms* **have** *keys p* ⊆ .[{*x*}] **by** (*rule PolysD*)
show *lookup* (*pm-of-poly x* (*poly-of-pm x p*)) *t* = *lookup p t*
proof (*simp add*: *pm-of-poly.rep-eq when-def*, *intro conjI impI*)
assume *t* ∈ .[{*x*}]
hence *Poly-Mapping.single x* (*lookup t x*) = *t*
by (*simp add*: *PPsD keys-subset-singleton-imp-monomial*)
thus *lookup p* (*Poly-Mapping.single x* (*lookup t x*)) = *lookup p t* **by** *simp*
next
assume *t* ∉ .[{*x*}]
with *assms PolysD* **have** *t* ∉ *keys p* **by** *blast*
thus *lookup p t* = *0* **by** (*simp add*: *in-keys-iff*)
qed
qed

lemma *poly-of-pm-of-poly* [*simp*]: *poly-of-pm x* (*pm-of-poly x p*) = *p*
by (*simp add*: *poly-of-pm-def coeff-inverse*)

lemma *pm-of-poly-in-Polys*: *pm-of-poly x p* ∈ *P*[{*x*}]
by (*auto simp*: *keys-pm-of-poly PPs-closed-single intro*!: *PolysI*)

lemma *pm-of-poly-zero* [*simp*]: *pm-of-poly x 0* = *0*

by (*rule poly-mapping-eqI*) (*simp add*: *pm-of-poly.rep-eq*)

lemma *pm-of-poly-eq-zero-iff* [*iff*]: *pm-of-poly x p = 0* ⟷ *p = 0*
by (*metis poly-of-pm-of-poly pm-of-poly-zero*)

lemma *pm-of-poly-monom*: *pm-of-poly x* (*Polynomial.monom c d*) = *monomial c* (*Poly-Mapping.single x d*)
proof (*rule poly-mapping-eqI*)
fix *t*
show *lookup* (*pm-of-poly x* (*Polynomial.monom c d*)) *t* = *lookup* (*monomial c* (*monomial d x*)) *t*
proof (*cases t* ∈ .[{*x*}])
case *True*
thus *?thesis*
by (*auto simp*: *pm-of-poly.rep-eq lookup-single PPs-singleton when-def dest*: *monomial-inj*)
next
case *False*
thus *?thesis* **by** (*auto simp add*: *pm-of-poly.rep-eq lookup-single PPs-singleton*)
qed
qed

lemma *pm-of-poly-plus*: *pm-of-poly x* (*p* + *q*) = *pm-of-poly x p* + *pm-of-poly x q*
by (*rule poly-mapping-eqI*) (*simp add*: *pm-of-poly.rep-eq lookup-add when-add-distrib*)

lemma *pm-of-poly-uminus* [*simp*]: *pm-of-poly x* (− *p*) = − *pm-of-poly x p*
by (*rule poly-mapping-eqI*) (*simp add*: *pm-of-poly.rep-eq when-distrib*)

lemma *pm-of-poly-minus*: *pm-of-poly x* (*p* − *q*) = *pm-of-poly x p* − *pm-of-poly x q*
by (*rule poly-mapping-eqI*) (*simp add*: *pm-of-poly.rep-eq lookup-minus when-diff-distrib*)

lemma *pm-of-poly-one* [*simp*]: *pm-of-poly x 1* = *1*
by (*simp add*: *pm-of-poly-monom flip*: *single-one monom-eq-1*)

lemma *pm-of-poly-pCons*:
pm-of-poly x (*pCons c p*) =
monomial c 0 + *punit.monom-mult* (*1*::-::*monoid-mult*) (*Poly-Mapping.single x 1*) (*pm-of-poly x p*)
(**is** *?l* = *?r*)
proof (*rule poly-mapping-eqI*)
fix *t*
let *?x* = *Poly-Mapping.single x* (*Suc 0*)
show *lookup ?l t* = *lookup ?r t*
proof (*cases ?x adds t*)
case *True*
have *1*: *t* − *?x* ∈ .[{*x*}] ⟷ *t* ∈ .[{*x*}]
proof
assume *t* − *?x* ∈ .[{*x*}]

moreover have *?x* ∈ .[{*x*}] **by** (*rule PPs-closed-single*) *simp*
ultimately have (*t* − *?x*) + *?x* ∈ .[{*x*}] **by** (*rule PPs-closed-plus*)
with *True* **show** *t* ∈ .[{*x*}] **by** (*simp add*: *adds-minus*)
qed (*rule PPs-closed-minus*)
from *True* **have** *0* < *lookup t x*
by (*metis adds-minus lookup-add lookup-single-eq n-not-Suc-n neq0-conv plus-eq-zero-2*)
moreover from *this* **have** *t* ≠ *0* **by** *auto*
ultimately show *?thesis* **using** *True*
by (*simp add*: *pm-of-poly.rep-eq lookup-add lookup-single punit.lookup-monom-mult 1 coeff-pCons*
lookup-minus split: *nat.split*)
next
case *False*
moreover have *t* ∈ .[{*x*}] ⟷ *t* = *0*
proof
assume *t* ∈ .[{*x*}]
hence *keys t* ⊆ {*x*} **by** (*rule PPsD*)
show *t* = *0*
proof (*rule ccontr*)
assume *t* ≠ *0*
hence *keys t* ≠ {} **by** *simp*
then obtain *y* **where** *y* ∈ *keys t* **by** *blast*
with ‹*keys t* ⊆ {*x*}› **have** *y* ∈ {*x*} **..**
hence *y* = *x* **by** *simp*
with ‹*y* ∈ *keys t*› **have** *Suc 0* ≤ *lookup t x* **by** (*simp add*: *in-keys-iff*)
hence *?x adds t*
by (*metis adds-poly-mappingI le0 le-funI lookup-single-eq lookup-single-not-eq*)
with *False* **show** *False* **..**
qed
qed (*simp only*: *zero-in-PPs*)
ultimately show *?thesis*
by (*simp add*: *pm-of-poly.rep-eq lookup-add lookup-single punit.lookup-monom-mult when-def*)
qed
qed

lemma *pm-of-poly-smult* [*simp*]: *pm-of-poly x* (*Polynomial.smult c p*) = *c* · *pm-of-poly x p*
by (*rule poly-mapping-eqI*) (*simp add*: *pm-of-poly.rep-eq when-distrib*)

lemma *pm-of-poly-times*: *pm-of-poly x* (*p* ∗ *q*) = *pm-of-poly x p* ∗ *pm-of-poly x* (*q*::-::*ring-1 poly*)
proof (*induct p*)
case *0*
show *?case* **by** *simp*
next
case (*pCons a p*)
show *?case*

by (*simp add*: *pm-of-poly-plus pm-of-poly-pCons map-scale-eq-times pCons*(*2*) *algebra-simps*
flip: *times-monomial-left*)
qed

lemma *pm-of-poly-sum*: *pm-of-poly x* (*sum f I*) = ($\sum$ *i*∈*I*. *pm-of-poly x* (*f i*))
by (*induct I rule*: *infinite-finite-induct*) (*simp-all add*: *pm-of-poly-plus*)

lemma *pm-of-poly-prod*: *pm-of-poly x* (*prod f I*) = ($\prod$ *i*∈*I*. *pm-of-poly x* (*f i* :: -::*ring-1 poly*))
by (*induct I rule*: *infinite-finite-induct*) (*simp-all add*: *pm-of-poly-times*)

lemma *pm-of-poly-power* [*simp*]: *pm-of-poly x* (*p* ^ *m*) = *pm-of-poly x* (*p*::-::*ring-1 poly*) ^ *m*
by (*induct m*) (*simp-all add*: *pm-of-poly-times*)

lemma *poly-of-pm-zero* [*simp*]: *poly-of-pm x 0* = *0*
by (*metis poly-of-pm-of-poly pm-of-poly-zero*)

lemma *poly-of-pm-eq-zero-iff*: *poly-of-pm x p* = *0* ⟷ *keys p* ∩ .[{*x*}] = {}
proof
assume *eq*: *poly-of-pm x p* = *0*
{
fix *t*
assume *t* ∈ .[{*x*}]
then obtain *d* **where** *t* = *Poly-Mapping.single x d* **unfolding** *PPs-singleton* **..**
moreover assume *t* ∈ *keys p*
ultimately have *0* ≠ *lookup p* (*Poly-Mapping.single x d*) **by** (*simp add*: *in-keys-iff*)
also have *lookup p* (*Poly-Mapping.single x d*) = *Polynomial.coeff* (*poly-of-pm x p*) *d*
by *simp*
also have ... = *0* **by** (*simp add*: *eq*)
finally have *False* **by** *blast*
}
thus *keys p* ∩ .[{*x*}] = {} **by** *blast*
next
assume ∗: *keys p* ∩ .[{*x*}] = {}
{
fix *d*
have *Poly-Mapping.single x d* ∈ .[{*x*}] (**is** *?x* ∈ -) **by** (*rule PPs-closed-single*) *simp*
with ∗ **have** *?x* ∉ *keys p* **by** *blast*
hence *Polynomial.coeff* (*poly-of-pm x p*) *d* = *0* **by** (*simp add*: *in-keys-iff*)
}
thus *poly-of-pm x p* = *0* **using** *leading-coeff-0-iff* **by** *blast*
qed

lemma *poly-of-pm-monomial*:
poly-of-pm x (*monomial c t*) = (*Polynomial.monom c* (*lookup t x*) *when* $t \in$.[{x}])
proof (*cases* $t \in$.[{x}])
case *True*
moreover from *this* **obtain** *d* **where** *t* = *Poly-Mapping.single x d*
by (*metis PPsD keys-subset-singleton-imp-monomial*)
ultimately show *?thesis* **unfolding** *Polynomial.monom.abs-eq coeff-poly-of-pm*
by (*auto simp*: *poly-of-pm-def lookup-single when-def*
dest!: *monomial-inj intro*!: *arg-cong*[**where** *f*=*Abs-poly*])
next
case *False*
moreover from *this* **have** $t \neq$ *monomial d x* **for** *d* **by** (*auto simp*: *PPs-closed-single*)
ultimately show *?thesis* **unfolding** *Polynomial.monom.abs-eq coeff-poly-of-pm*
by (*auto simp*: *poly-of-pm-def lookup-single when-def zero-poly.abs-eq*)
qed

lemma *poly-of-pm-plus*: *poly-of-pm x* (*p* + *q*) = *poly-of-pm x p* + *poly-of-pm x q*
unfolding *Polynomial.plus-poly.abs-eq coeff-poly-of-pm* **by** (*simp add*: *poly-of-pm-def lookup-add*)

lemma *poly-of-pm-uminus* [*simp*]: *poly-of-pm x* (− *p*) = − *poly-of-pm x p*
unfolding *Polynomial.uminus-poly.abs-eq coeff-poly-of-pm* **by** (*simp add*: *poly-of-pm-def*)

lemma *poly-of-pm-minus*: *poly-of-pm x* (*p* − *q*) = *poly-of-pm x p* − *poly-of-pm x q*
unfolding *Polynomial.minus-poly.abs-eq coeff-poly-of-pm* **by** (*simp add*: *poly-of-pm-def lookup-minus*)

lemma *poly-of-pm-one* [*simp*]: *poly-of-pm x 1* = *1*
by (*simp add*: *poly-of-pm-monomial zero-in-PPs flip*: *single-one monom-eq-1*)

lemma *poly-of-pm-times*:
poly-of-pm x (*p* ∗ *q*) = *poly-of-pm x p* ∗ *poly-of-pm x* (*q*::- $\Rightarrow_0$ *'a*::*comm-semiring-1*)
proof −
have *eq*: *poly-of-pm x* (*monomial c t* ∗ *q*) = *poly-of-pm x* (*monomial c t*) ∗ *poly-of-pm x q*
if $c \neq 0$ **for** *c t*
proof (*cases* $t \in$.[{x}])
case *True*
then obtain *d* **where** *t*: *t* = *Poly-Mapping.single x d* **unfolding** *PPs-singleton* ..
have *poly-of-pm x* (*monomial c t*) ∗ *poly-of-pm x q* = *Polynomial.monom c* (*lookup t x*) ∗ *poly-of-pm x q*
by (*simp add*: *True poly-of-pm-monomial*)
also have . . . = *poly-of-pm x* (*monomial c t* ∗ *q*) **unfolding** *t*
proof (*induct d*)
case *0*
have *Polynomial.smult c* (*poly-of-pm x q*) = *poly-of-pm x* ($c \cdot q$)

unfolding *Polynomial.smult.abs-eq coeff-poly-of-pm* **by** (*simp add*: *poly-of-pm-def*)
with *that* **show** *?case* **by** (*simp add*: *Polynomial.times-poly-def flip*: *map-scale-eq-times*)
next
case (*Suc d*)
have *1*: *Poly-Mapping.single x a adds Poly-Mapping.single x b* $\longleftrightarrow$ $a \leq b$ **for** *a b* :: *nat*
by (*metis adds-def deg-pm-mono deg-pm-single le-Suc-ex single-add*)
have *2*: *poly-of-pm x* (*punit.monom-mult 1* (*Poly-Mapping.single x 1*) *r*) = *pCons 0* (*poly-of-pm x r*)
for *r* :: - $\Rightarrow_0$ *'a* **unfolding** *poly.coeff-inject*[*symmetric*]
by (*rule ext*) (*simp add*: *coeff-pCons punit.lookup-monom-mult adds-zero monomial-0-iff 1*
flip: *single-diff split*: *nat.split*)
from *Suc that* **have** *Polynomial.monom c* (*lookup* (*monomial* (*Suc d*) *x*) *x*) ∗ *poly-of-pm x q* =
poly-of-pm x (*punit.monom-mult 1* (*Poly-Mapping.single x 1*)
((*monomial c* (*monomial d x*)) ∗ *q*))
by (*simp add*: *Polynomial.times-poly-def 2 del*: *One-nat-def*)
also have ... = *poly-of-pm x* (*monomial c* (*Poly-Mapping.single x* (*Suc d*)) ∗ *q*)
by (*simp add*: *ac-simps times-monomial-monomial flip*: *single-add times-monomial-left*)
finally show *?case* **.**
qed
finally show *?thesis* **by** (*rule sym*)
next
case *False*
{
fix *s*
assume $s \in$ *keys* (*monomial c t* ∗ *q*)
also have ... $\subseteq$ (+) *t* ‘ *keys q* **unfolding** *times-monomial-left*
by (*fact punit.keys-monom-mult-subset*[*simplified*])
finally obtain *u* **where** *s*: *s* = *t* + *u* **..**
assume $s \in .[\{x\}]$
hence $s - u \in .[\{x\}]$ **by** (*rule PPs-closed-minus*)
hence $t \in .[\{x\}]$ **by** (*simp add*: *s*)
with *False* **have** *False* **..**
}
hence *poly-of-pm x* (*monomial c t* ∗ *q*) = *0* **by** (*auto simp*: *poly-of-pm-eq-zero-iff*)
with *False* **show** *?thesis* **by** (*simp add*: *poly-of-pm-monomial*)
qed
show *?thesis*
by (*induct p rule*: *poly-mapping-plus-induct*) (*simp-all add*: *poly-of-pm-plus eq distrib-right*)
qed

lemma *poly-of-pm-sum*: *poly-of-pm x* (*sum f I*) = ($\sum$ *i*∈*I*. *poly-of-pm x* (*f i*))
by (*induct I rule*: *infinite-finite-induct*) (*simp-all add*: *poly-of-pm-plus*)

lemma *poly-of-pm-prod*: *poly-of-pm x* (*prod f I*) = ($\prod$ *i*∈*I*. *poly-of-pm x* (*f i*))

by (*induct I rule*: *infinite-finite-induct*) (*simp-all add*: *poly-of-pm-times*)

lemma *poly-of-pm-power* [*simp*]: *poly-of-pm x* (*p* ^ *m*) = *poly-of-pm x p* ^ *m*
by (*induct m*) (*simp-all add*: *poly-of-pm-times*)

3.2 Evaluating Polynomials

lemma *poly-eq-poly-eval*: *poly* (*poly-of-pm x p*) *a* = *poly-eval* (λ*y*. *a when y* = *x*) *p*
proof (*induction p rule*: *poly-mapping-plus-induct*)
case *1*
show *?case* **by** *simp*
next
case (*2 p c t*)
show *?case*
proof (*cases t* ∈ .[{*x*}])
case *True*
have *poly-eval* (λ*y*. *a when y* = *x*) (*monomial c t*) = *c* ∗ (∏ *y*∈*keys t*. (*a when y* = *x*) ^ *lookup t y*)
by (*simp only*: *poly-eval-monomial*)
also from *True* **have** (∏ *y*∈*keys t*. (*a when y* = *x*) ^ *lookup t y*) = (∏ *y*∈{*x*}. (*a when y* = *x*) ^ *lookup t y*)
by (*intro prod.mono-neutral-left ballI*) (*auto simp*: *in-keys-iff dest*: *PPsD*)
also have ... = *a* ^ *lookup t x* **by** *simp*
finally show *?thesis*
by (*simp add*: *poly-of-pm-plus poly-of-pm-monomial poly-monom poly-eval-plus True 2*(*3*))
next
case *False*
have *poly-eval* (λ*y*. *a when y* = *x*) (*monomial c t*) = *c* ∗ (∏ *y*∈*keys t*. (*a when y* = *x*) ^ *lookup t y*)
by (*simp only*: *poly-eval-monomial*)
also from *finite-keys* **have** (∏ *y*∈*keys t*. (*a when y* = *x*) ^ *lookup t y*) = *0*
proof (*rule prod-zero*)
from *False* **obtain** *y* **where** *y* ∈ *keys t* **and** *y* ≠ *x* **by** (*auto simp*: *PPs-def*)
from *this*(*1*) **show** ∃ *y*∈*keys t*. (*a when y* = *x*) ^ *lookup t y* = *0*
proof
from ‹*y* ∈ *keys t*› **have** *0* < *lookup t y* **by** (*simp add*: *in-keys-iff*)
with ‹*y* ≠ *x*› **show** (*a when y* = *x*) ^ *lookup t y* = *0* **by** (*simp add*: *zero-power*)
qed
qed
finally show *?thesis*
by (*simp add*: *poly-of-pm-plus poly-of-pm-monomial poly-monom poly-eval-plus False 2*(*3*))
qed
qed

corollary *poly-eq-poly-eval′*:

assumes $p \in P[\{x\}]$
shows *poly* (*poly-of-pm x p*) *a* = *poly-eval* (λ-. *a*) *p*
unfolding *poly-eq-poly-eval* **using** *refl*
proof (*rule poly-eval-cong*)
fix *y*
assume $y \in$ *indets p*
also from *assms* **have** $\ldots \subseteq \{x\}$ **by** (*rule PolysD*)
finally show (*a when y = x*) = *a* **by** *simp*
qed

lemma *poly-eval-eq-poly*: *poly-eval a* (*pm-of-poly x p*) = *poly p* (*a x*)
by (*induct p*)
(*simp-all add*: *pm-of-poly-pCons poly-eval-plus poly-eval-times poly-eval-monomial*
flip: *times-monomial-left*)

3.3 Morphisms *flat-pm-of-poly* and *poly-of-focus*

definition *flat-pm-of-poly* :: $'x \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a)$ *poly* $\Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0$ $'a$::*semiring-1*)
where *flat-pm-of-poly x* = *flatten* $\circ$ *pm-of-poly x*

definition *poly-of-focus* :: $'x \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a) \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a$::*comm-monoid-add*) *poly*
where *poly-of-focus x* = *poly-of-pm x* $\circ$ *focus* $\{x\}$

lemma *flat-pm-of-poly-in-Polys*:
assumes *range* (*poly.coeff p*) $\subseteq P[Y]$
shows *flat-pm-of-poly x p* $\in P[$*insert x Y*$]$
proof −
let *?p* = *pm-of-poly x p*
from *assms* **have** *lookup ?p* ‘ *keys ?p* $\subseteq P[Y]$ **by** (*simp add*: *keys-pm-of-poly image-image*) *blast*
with *pm-of-poly-in-Polys* **have** *flatten ?p* $\in P[\{x\} \cup Y]$ **by** (*rule flatten-in-Polys*)
thus *?thesis* **by** (*simp add*: *flat-pm-of-poly-def*)
qed

corollary *indets-flat-pm-of-poly-subset*:
indets (*flat-pm-of-poly x p*) $\subseteq$ *insert x* ($\bigcup$ (*indets* ‘ *range* (*poly.coeff p*)))
proof −
let *?p* = *pm-of-poly x p*
let *?Y* = $\bigcup$ (*indets* ‘ *range* (*poly.coeff p*))
have *range* (*poly.coeff p*) $\subseteq P[?Y]$ **by** (*auto intro*: *PolysI-alt*)
hence *flat-pm-of-poly x p* $\in P[$*insert x ?Y*$]$ **by** (*rule flat-pm-of-poly-in-Polys*)
thus *?thesis* **by** (*rule PolysD*)
qed

lemma
shows *flat-pm-of-poly-zero* [*simp*]: *flat-pm-of-poly x 0* = *0*
and *flat-pm-of-poly-monom*: *flat-pm-of-poly x* (*Polynomial.monom c d*) =

punit.monom-mult 1 (*Poly-Mapping.single x d*) *c*
and *flat-pm-of-poly-plus*: *flat-pm-of-poly x* (*p* + *q*) =
flat-pm-of-poly x p + *flat-pm-of-poly x q*
and *flat-pm-of-poly-one* [*simp*]: *flat-pm-of-poly x 1* = *1*
and *flat-pm-of-poly-sum*: *flat-pm-of-poly x* (*sum f I*) = ($\sum$ *i*∈*I*. *flat-pm-of-poly x* (*f i*))
by (*simp-all add*: *flat-pm-of-poly-def pm-of-poly-monom flatten-monomial pm-of-poly-plus flatten-plus pm-of-poly-sum flatten-sum*)

lemma
shows *flat-pm-of-poly-uminus* [*simp*]: *flat-pm-of-poly x* (− *p*) = − *flat-pm-of-poly x p*
and *flat-pm-of-poly-minus*: *flat-pm-of-poly x* (*p* − *q*) =
flat-pm-of-poly x p − *flat-pm-of-poly x* (*q*::-::*ring poly*)
by (*simp-all add*: *flat-pm-of-poly-def pm-of-poly-minus flatten-minus*)

lemma *flat-pm-of-poly-pCons*:
flat-pm-of-poly x (*pCons c p*) =
c + *punit.monom-mult 1* (*Poly-Mapping.single x 1*) (*flat-pm-of-poly x* (*p*::-::*comm-semiring-1 poly*))
by (*simp add*: *flat-pm-of-poly-def pm-of-poly-pCons flatten-plus flatten-monomial flatten-times*
flip: *times-monomial-left*)

lemma *flat-pm-of-poly-smult* [*simp*]:
flat-pm-of-poly x (*Polynomial.smult c p*) = *c* ∗ *flat-pm-of-poly x* (*p*::-::*comm-semiring-1 poly*)
by (*simp add*: *flat-pm-of-poly-def map-scale-eq-times flatten-times flatten-monomial pm-of-poly-times*)

lemma
shows *flat-pm-of-poly-times*: *flat-pm-of-poly x* (*p* ∗ *q*) = *flat-pm-of-poly x p* ∗ *flat-pm-of-poly x q*
and *flat-pm-of-poly-prod*: *flat-pm-of-poly x* (*prod f I*) =
($\prod$ *i*∈*I*. *flat-pm-of-poly x* (*f i* :: -::*comm-ring-1 poly*))
and *flat-pm-of-poly-power*: *flat-pm-of-poly x* (*p* ^ *m*) = *flat-pm-of-poly x* (*p*::-::*comm-ring-1 poly*) ^ *m*
by (*simp-all add*: *flat-pm-of-poly-def flatten-times pm-of-poly-times flatten-prod pm-of-poly-prod*)

lemma *coeff-poly-of-focus-subset-Polys*:
assumes *p* ∈ *P*[*X*]
shows *range* (*poly.coeff* (*poly-of-focus x p*)) ⊆ *P*[*X* − {*x*}]
proof −
have *range* (*poly.coeff* (*poly-of-focus x p*)) ⊆ *range* (*lookup* (*focus* {*x*} *p*))
by (*auto simp*: *poly-of-focus-def*)
also from *assms* **have** . . . ⊆ *P*[*X* − {*x*}] **by** (*rule focus-coeffs-subset-Polys′*)
finally show *?thesis* **.**
qed

lemma
shows *poly-of-focus-zero* [*simp*]: *poly-of-focus x 0* = *0*
and *poly-of-focus-uminus* [*simp*]: *poly-of-focus x* (− *p*) = − *poly-of-focus x p*
and *poly-of-focus-plus*: *poly-of-focus x* (*p* + *q*) = *poly-of-focus x p* + *poly-of-focus x q*
and *poly-of-focus-minus*: *poly-of-focus x* (*p* − *q*) = *poly-of-focus x p* − *poly-of-focus x q*
and *poly-of-focus-one* [*simp*]: *poly-of-focus x 1* = *1*
and *poly-of-focus-sum*: *poly-of-focus x* (*sum f I*) = ($\sum$ *i*∈*I*. *poly-of-focus x* (*f i*))
by (*simp-all add*: *poly-of-focus-def keys-focus poly-of-pm-plus focus-plus poly-of-pm-minus focus-minus*
poly-of-pm-sum focus-sum)

lemma *poly-of-focus-eq-zero-iff* [*iff*]: *poly-of-focus x p* = *0* ⟷ *p* = *0*
using *focus-in-Polys*[*of* {*x*} *p*]
by (*auto simp*: *poly-of-focus-def poly-of-pm-eq-zero-iff Int-absorb2 dest*: *PolysD*)

lemma *poly-of-focus-monomial*:
poly-of-focus x (*monomial c t*) = *Polynomial.monom* (*monomial c* (*except t* {*x*})) (*lookup t x*)
by (*simp add*: *poly-of-focus-def focus-monomial poly-of-pm-monomial PPs-def keys-except lookup-except*)

lemma
shows *poly-of-focus-times*: *poly-of-focus x* (*p* ∗ *q*) = *poly-of-focus x p* ∗ *poly-of-focus x q*
and *poly-of-focus-prod*: *poly-of-focus x* (*prod f I*) =
($\prod$ *i*∈*I*. *poly-of-focus x* (*f i* :: - ⇒$_0$ -::*comm-semiring-1*))
and *poly-of-focus-power*: *poly-of-focus x* (*p* ^ *m*) = *poly-of-focus x* (*p*::- ⇒$_0$ -::*comm-semiring-1*) ^ *m*
by (*simp-all add*: *poly-of-focus-def poly-of-pm-times focus-times poly-of-pm-prod focus-prod*)

lemma *flat-pm-of-poly-of-focus* [*simp*]: *flat-pm-of-poly x* (*poly-of-focus x p*) = *p*
by (*simp add*: *flat-pm-of-poly-def poly-of-focus-def pm-of-poly-of-pm focus-in-Polys*)

lemma *poly-of-focus-flat-pm-of-poly*:
assumes *range* (*poly.coeff p*) ⊆ *P*[− {*x*}]
shows *poly-of-focus x* (*flat-pm-of-poly x p*) = *p*
proof −
from *assms* **have** *lookup* (*pm-of-poly x p*) ‘ *keys* (*pm-of-poly x p*) ⊆ *P*[− {*x*}]
by (*simp add*: *keys-pm-of-poly image-image*) *blast*
thus *?thesis* **by** (*simp add*: *flat-pm-of-poly-def poly-of-focus-def focus-flatten pm-of-poly-in-Polys*)
qed

lemma *flat-pm-of-poly-eq-zeroD*:
assumes *flat-pm-of-poly x p* = *0* **and** *range* (*poly.coeff p*) ⊆ *P*[− {*x*}]

shows *p* = *0*
proof −
from *assms*(*2*) **have** *p* = *poly-of-focus x* (*flat-pm-of-poly x p*)
by (*simp only*: *poly-of-focus-flat-pm-of-poly*)
also have ... = *0* **by** (*simp add*: *assms*(*1*))
finally show *?thesis* **.**
qed

lemma *poly-poly-of-focus*: *poly* (*poly-of-focus x p*) *a* = *poly-eval* (λ-. *a*) (*focus* {*x*} *p*)
by (*simp add*: *poly-of-focus-def poly-eq-poly-eval′ focus-in-Polys*)

corollary *poly-poly-of-focus-monomial*:
poly (*poly-of-focus x p*) (*monomial 1* (*Poly-Mapping.single x 1*)) = (*p*::- $\Rightarrow_0$ -::*comm-semiring-1*)
unfolding *poly-poly-of-focus poly-eval-focus* **by** (*rule poly-subst-id*) *simp*

end

4 Hilbert's Nullstellensatz

theory *Nullstellensatz*
imports *Algebraically-Closed-Fields*
HOL−Computational-Algebra.Fraction-Field
Lex-Order-PP
Univariate-PM
Groebner-Bases.Groebner-PM
begin

We prove the geometric version of Hilbert's Nullstellensatz, i.e. the precise correspondence between algebraic varieties and radical ideals. The field-theoretic version of the Nullstellensatz is proved in theory *Nullstellensatz-Field*.

4.1 Preliminaries

lemma *finite-linorder-induct* [*consumes 1*, *case-names empty insert*]:
assumes *finite* (*A*::′*a*::*linorder set*) **and** *P* {}
and ⋀*a A*. *finite A* ⟹ *A* ⊆ {..<*a*} ⟹ *P A* ⟹ *P* (*insert a A*)
shows *P A*
proof −
define *k* **where** *k* = *card A*
thus *?thesis* **using** *assms*(*1*)
proof (*induct k arbitrary*: *A*)
case *0*
with *assms*(*2*) **show** *?case* **by** *simp*
next
case (*Suc k*)

define *a* **where** $a = Max\ A$
from *Suc.prems*(*1*) **have** $A \neq \{\}$ **by** *auto*
with *Suc.prems*(*2*) **have** $a \in A$ **unfolding** *a-def* **by** (*rule Max-in*)
with *Suc.prems* **have** $k = card\ (A - \{a\})$ **by** *simp*
moreover from *Suc.prems*(*2*) **have** *finite* $(A - \{a\})$ **by** *simp*
ultimately have $P\ (A - \{a\})$ **by** (*rule Suc.hyps*)
with ‹*finite* $(A - \{a\})$› - **have** P (*insert* a $(A - \{a\})$)
proof (*rule assms*(*3*))
show $A - \{a\} \subseteq \{..<a\}$
proof
fix *b*
assume $b \in A - \{a\}$
hence $b \in A$ **and** $b \neq a$ **by** *simp-all*
moreover from *Suc.prems*(*2*) *this*(*1*) **have** $b \leq a$ **unfolding** *a-def* **by** (*rule Max-ge*)
ultimately show $b \in \{..<a\}$ **by** *simp*
qed
qed
with ‹$a \in A$› **show** *?case* **by** (*simp add*: *insert-absorb*)
qed
qed

lemma *Fract-same*: $Fract\ a\ a = (1\ when\ a \neq 0)$
by (*simp add*: *One-fract-def Zero-fract-def eq-fract when-def*)

lemma *Fract-eq-zero-iff*: $Fract\ a\ b = 0 \longleftrightarrow a = 0 \vee b = 0$
by (*metis* (*no-types*, *lifting*) *Zero-fract-def eq-fract*(*1*) *eq-fract*(*2*) *mult-eq-0-iff one-neq-zero*)

lemma *poly-plus-rightE*:
obtains *c* **where** $poly\ p\ (x + y) = poly\ p\ x + c * y$
proof (*induct p arbitrary*: *thesis*)
case *0*
have $poly\ 0\ (x + y) = poly\ 0\ x + 0 * y$ **by** *simp*
thus *?case* **by** (*rule 0*)
next
case (*pCons a p*)
obtain *c* **where** $poly\ p\ (x + y) = poly\ p\ x + c * y$ **by** (*rule pCons.hyps*)
hence $poly\ (pCons\ a\ p)\ (x + y) = a + (x + y) * (poly\ p\ x + c * y)$ **by** *simp*
also have $\ldots = poly\ (pCons\ a\ p)\ x + (x * c + (poly\ p\ x + c * y)) * y$ **by** (*simp add*: *algebra-simps*)
finally show *?case* **by** (*rule pCons.prems*)
qed

lemma *poly-minus-rightE*:
obtains *c* **where** $poly\ p\ (x - y) = poly\ p\ x - c * (y{::}\text{-}{::}comm\text{-}ring)$
by (*metis add-diff-cancel-right′ diff-add-cancel poly-plus-rightE*)

lemma *map-poly-plus*:

assumes *f 0* = *0* **and** $\bigwedge$*a b*. *f* (*a* + *b*) = *f a* + *f b*
shows *map-poly f* (*p* + *q*) = *map-poly f p* + *map-poly f q*
by (*rule Polynomial.poly-eqI*) (*simp add*: *coeff-map-poly assms*)

lemma *map-poly-minus*:
assumes *f 0* = *0* **and** $\bigwedge$*a b*. *f* (*a* − *b*) = *f a* − *f b*
shows *map-poly f* (*p* − *q*) = *map-poly f p* − *map-poly f q*
by (*rule Polynomial.poly-eqI*) (*simp add*: *coeff-map-poly assms*)

lemma *map-poly-sum*:
assumes *f 0* = *0* **and** $\bigwedge$*a b*. *f* (*a* + *b*) = *f a* + *f b*
shows *map-poly f* (*sum g A*) = ($\sum$ *a*∈*A*. *map-poly f* (*g a*))
by (*induct A rule*: *infinite-finite-induct*) (*simp-all add*: *map-poly-plus assms*)

lemma *map-poly-times*:
assumes *f 0* = *0* **and** $\bigwedge$*a b*. *f* (*a* + *b*) = *f a* + *f b* **and** $\bigwedge$*a b*. *f* (*a* ∗ *b*) = *f a* ∗ *f b*
shows *map-poly f* (*p* ∗ *q*) = *map-poly f p* ∗ *map-poly f q*
proof (*induct p*)
case *0*
show *?case* **by** *simp*
next
case (*pCons c p*)
show *?case* **by** (*simp add*: *assms map-poly-plus map-poly-smult map-poly-pCons pCons*)
qed

lemma *poly-Fract*:
assumes *set* (*Polynomial.coeffs p*) ⊆ *range* (λ*x*. *Fract x 1*)
obtains *q m* **where** *poly p* (*Fract a b*) = *Fract q* (*b* ^ *m*)
using *assms*
proof (*induct p arbitrary*: *thesis*)
case *0*
have *poly 0* (*Fract a b*) = *Fract 0* (*b* ^ *1*) **by** (*simp add*: *fract-collapse*)
thus *?case* **by** (*rule 0*)
next
case (*pCons c p*)
from *pCons.hyps*(*1*) **have** *insert c* (*set* (*Polynomial.coeffs p*)) = *set* (*Polynomial.coeffs* (*pCons c p*))
by *auto*
with *pCons.prems*(*2*) **have** *c* ∈ *range* (λ*x*. *Fract x 1*) **and** *set* (*Polynomial.coeffs p*) ⊆ *range* (λ*x*. *Fract x 1*)
by *blast+*
from *this*(*2*) **obtain** *q0 m0* **where** *poly-p*: *poly p* (*Fract a b*) = *Fract q0* (*b* ^ *m0*)
using *pCons.hyps*(*2*) **by** *blast*
from ‹*c* ∈ -› **obtain** *c0* **where** *c*: *c* = *Fract c0 1* **..**
show *?case*
proof (*cases b* = *0*)

case *True*
hence *poly* (*pCons c p*) (*Fract a b*) = *Fract c0* (*b* ^ *0*) **by** (*simp add*: *c fract-collapse*)
thus *?thesis* **by** (*rule pCons.prems*)
next
case *False*
hence *poly* (*pCons c p*) (*Fract a b*) = *Fract* (*c0* ∗ *b* ^ *Suc m0* + *a* ∗ *q0*) (*b* ^ *Suc m0*)
by (*simp add*: *poly-p c*)
thus *?thesis* **by** (*rule pCons.prems*)
qed
qed

lemma (**in** *ordered-term*) *lt-sum-le-Max*: *lt* (*sum f A*) $\preceq_t$ *ord-term-lin.Max* {*lt* (*f a*) | *a*. *a* ∈ *A*}
proof (*induct A rule*: *infinite-finite-induct*)
case (*infinite A*)
thus *?case* **by** (*simp add*: *min-term-min*)
next
case *empty*
thus *?case* **by** (*simp add*: *min-term-min*)
next
case (*insert a A*)
show *?case*
proof (*cases A* = {})
case *True*
thus *?thesis* **by** *simp*
next
case *False*
from *insert.hyps*(*1*, *2*) **have** *lt* (*sum f* (*insert a A*)) = *lt* (*f a* + *sum f A*) **by** *simp*
also have ... $\preceq_t$ *ord-term-lin.max* (*lt* (*f a*)) (*lt* (*sum f A*)) **by** (*rule lt-plus-le-max*)
also have ... $\preceq_t$ *ord-term-lin.max* (*lt* (*f a*)) (*ord-term-lin.Max* {*lt* (*f a*) |*a*. *a* ∈ *A*})
using *insert.hyps*(*3*) *ord-term-lin.max.mono* **by** *blast*
also from *insert.hyps*(*1*) *False* **have** ... = *ord-term-lin.Max* (*insert* (*lt* (*f a*)) {*lt* (*f x*) |*x*. *x* ∈ *A*})
by *simp*
also have ... = *ord-term-lin.Max* {*lt* (*f x*) |*x*. *x* ∈ *insert a A*}
by (*rule arg-cong*[**where** *f*=*ord-term-lin.Max*]) *blast*
finally show *?thesis* **.**
qed
qed

4.2 Ideals and Varieties

definition *variety-of* :: ((*'x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*) *set* ⇒ (*'x* ⇒ *'a*::*comm-semiring-1*) *set*
where *variety-of F* = {*a*. ∀ *f*∈*F*. *poly-eval a f* = *0*}

definition *ideal-of* :: (′*x* ⇒ ′*a*::*comm-semiring-1*) *set* ⇒ ((′*x* ⇒$_0$ *nat*) ⇒$_0$ ′*a*) *set*
where *ideal-of A* = {*f*. ∀ *a*∈*A*. *poly-eval a f* = *0*}

abbreviation $\mathcal{V}$ ≡ *variety-of*
abbreviation $\mathcal{I}$ ≡ *ideal-of*

lemma *variety-ofI*: (⋀*f*. *f* ∈ *F* ⟹ *poly-eval a f* = *0*) ⟹ *a* ∈ $\mathcal{V}$ *F*
by (*simp add*: *variety-of-def*)

lemma *variety-ofI-alt*: *poly-eval a* ‘ *F* ⊆ {*0*} ⟹ *a* ∈ $\mathcal{V}$ *F*
by (*auto intro*: *variety-ofI*)

lemma *variety-ofD*: *a* ∈ $\mathcal{V}$ *F* ⟹ *f* ∈ *F* ⟹ *poly-eval a f* = *0*
by (*simp add*: *variety-of-def*)

lemma *variety-of-empty* [*simp*]: $\mathcal{V}$ {} = *UNIV*
by (*simp add*: *variety-of-def*)

lemma *variety-of-UNIV* [*simp*]: $\mathcal{V}$ *UNIV* = {}
by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq UNIV-I one-neq-zero poly-eval-one variety-of-def*)

lemma *variety-of-antimono*: *F* ⊆ *G* ⟹ $\mathcal{V}$ *G* ⊆ $\mathcal{V}$ *F*
by (*auto simp*: *variety-of-def*)

lemma *variety-of-ideal* [*simp*]: $\mathcal{V}$ (*ideal F*) = $\mathcal{V}$ *F*
proof
show $\mathcal{V}$ (*ideal F*) ⊆ $\mathcal{V}$ *F* **by** (*intro variety-of-antimono ideal.span-superset*)
next
show $\mathcal{V}$ *F* ⊆ $\mathcal{V}$ (*ideal F*)
proof (*intro subsetI variety-ofI*)
fix *a f*
assume *a* ∈ $\mathcal{V}$ *F* **and** *f* ∈ *ideal F*
from *this*(*2*) **show** *poly-eval a f* = *0*
proof (*induct f rule*: *ideal.span-induct-alt*)
case *base*
show *?case* **by** *simp*
next
case (*step c f g*)
with ‹*a* ∈ $\mathcal{V}$ *F*› **show** *?case* **by** (*auto simp*: *poly-eval-plus poly-eval-times dest*: *variety-ofD*)
qed
qed
qed

lemma *ideal-ofI*: (⋀*a*. *a* ∈ *A* ⟹ *poly-eval a f* = *0*) ⟹ *f* ∈ $\mathcal{I}$ *A*
by (*simp add*: *ideal-of-def*)

lemma *ideal-ofD*: *f* ∈ ℐ *A* ⟹ *a* ∈ *A* ⟹ *poly-eval a f* = *0*
by (*simp add*: *ideal-of-def*)

lemma *ideal-of-empty* [*simp*]: ℐ {} = *UNIV*
by (*simp add*: *ideal-of-def*)

lemma *ideal-of-antimono*: *A* ⊆ *B* ⟹ ℐ *B* ⊆ ℐ *A*
by (*auto simp*: *ideal-of-def*)

lemma *ideal-ideal-of* [*simp*]: *ideal* (ℐ *A*) = ℐ *A*
unfolding *ideal.span-eq-iff*
proof (*rule ideal.subspaceI*)
show *0* ∈ ℐ *A* **by** (*rule ideal-ofI*) *simp*
next
fix *f g*
assume *f* ∈ ℐ *A*
hence *f*: *poly-eval a f* = *0* **if** *a* ∈ *A* **for** *a* **using** *that* **by** (*rule ideal-ofD*)
assume *g* ∈ ℐ *A*
hence *g*: *poly-eval a g* = *0* **if** *a* ∈ *A* **for** *a* **using** *that* **by** (*rule ideal-ofD*)
show *f* + *g* ∈ ℐ *A* **by** (*rule ideal-ofI*) (*simp add*: *poly-eval-plus f g*)
next
fix *c f*
assume *f* ∈ ℐ *A*
hence *f*: *poly-eval a f* = *0* **if** *a* ∈ *A* **for** *a* **using** *that* **by** (*rule ideal-ofD*)
show *c* ∗ *f* ∈ ℐ *A* **by** (*rule ideal-ofI*) (*simp add*: *poly-eval-times f*)
qed

lemma *ideal-of-UN*: ℐ (⋃ (*A* ‘ *J*)) = (⋂*j*∈*J*. ℐ (*A j*))
proof (*intro set-eqI iffI ideal-ofI INT-I*)
fix *p j a*
assume *p* ∈ ℐ (⋃ (*A* ‘ *J*))
assume *j* ∈ *J* **and** *a* ∈ *A j*
hence *a* ∈ ⋃ (*A* ‘ *J*) **..**
with ‹*p* ∈ -› **show** *poly-eval a p* = *0* **by** (*rule ideal-ofD*)
next
fix *p a*
assume *a* ∈ ⋃ (*A* ‘ *J*)
then obtain *j* **where** *j* ∈ *J* **and** *a* ∈ *A j* **..**
assume *p* ∈ (⋂*j*∈*J*. ℐ (*A j*))
hence *p* ∈ ℐ (*A j*) **using** ‹*j* ∈ *J*› **..**
thus *poly-eval a p* = *0* **using** ‹*a* ∈ *A j*› **by** (*rule ideal-ofD*)
qed

corollary *ideal-of-Un*: ℐ (*A* ∪ *B*) = ℐ *A* ∩ ℐ *B*
using *ideal-of-UN*[*of id* {*A*, *B*}] **by** *simp*

lemma *variety-of-ideal-of-variety* [*simp*]: 𝒱 (ℐ (𝒱 *F*)) = 𝒱 *F* (**is** - = *?V*)
proof
have *F* ⊆ ℐ (𝒱 *F*) **by** (*auto intro*!: *ideal-ofI dest*: *variety-ofD*)

thus $\mathcal{V}$ ($\mathcal{I}$ *?V*) ⊆ *?V* **by** (*rule variety-of-antimono*)
next
show *?V* ⊆ $\mathcal{V}$ ($\mathcal{I}$ *?V*) **by** (*auto intro*!: *variety-ofI dest*: *ideal-ofD*)
qed

lemma *ideal-of-inj-on*: *inj-on* $\mathcal{I}$ (*range* ($\mathcal{V}$::(('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*comm-semiring-1*) *set* ⇒ -))
proof (*rule inj-onI*)
fix *A B* :: ('*x* ⇒ '*a*) *set*
assume *A* ∈ *range* $\mathcal{V}$
then obtain *F* **where** *A*: *A* = $\mathcal{V}$ *F* **..**
assume *B* ∈ *range* $\mathcal{V}$
then obtain *G* **where** *B*: *B* = $\mathcal{V}$ *G* **..**
assume $\mathcal{I}$ *A* = $\mathcal{I}$ *B*
hence $\mathcal{V}$ ($\mathcal{I}$ *A*) = $\mathcal{V}$ ($\mathcal{I}$ *B*) **by** *simp*
thus *A* = *B* **by** (*simp add*: *A B*)
qed

lemma *ideal-of-variety-of-ideal* [*simp*]: $\mathcal{I}$ ($\mathcal{V}$ ($\mathcal{I}$ *A*)) = $\mathcal{I}$ *A* (**is** - = *?I*)
proof
have *A* ⊆ $\mathcal{V}$ ($\mathcal{I}$ *A*) **by** (*auto intro*!: *variety-ofI dest*: *ideal-ofD*)
thus $\mathcal{I}$ ($\mathcal{V}$ *?I*) ⊆ *?I* **by** (*rule ideal-of-antimono*)
next
show *?I* ⊆ $\mathcal{I}$ ($\mathcal{V}$ *?I*) **by** (*auto intro*!: *ideal-ofI dest*: *variety-ofD*)
qed

lemma *variety-of-inj-on*: *inj-on* $\mathcal{V}$ (*range* ($\mathcal{I}$::('*x* ⇒ '*a*::*comm-semiring-1*) *set* ⇒ -))
proof (*rule inj-onI*)
fix *F G* :: (('*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*) *set*
assume *F* ∈ *range* $\mathcal{I}$
then obtain *A* **where** *F*: *F* = $\mathcal{I}$ *A* **..**
assume *G* ∈ *range* $\mathcal{I}$
then obtain *B* **where** *G*: *G* = $\mathcal{I}$ *B* **..**
assume $\mathcal{V}$ *F* = $\mathcal{V}$ *G*
hence $\mathcal{I}$ ($\mathcal{V}$ *F*) = $\mathcal{I}$ ($\mathcal{V}$ *G*) **by** *simp*
thus *F* = *G* **by** (*simp add*: *F G*)
qed

lemma *image-map-indets-ideal-of*:
assumes *inj f*
shows *map-indets f* ' $\mathcal{I}$ *A* = $\mathcal{I}$ ((λ*a*. *a* ∘ *f*) −' (*A*::('*x* ⇒ '*a*::*comm-semiring-1*) *set*)) ∩ *P*[*range f*]
proof −
{
fix *p* **and** *a*::'*x* ⇒ '*a*
assume ∀ *a*∈(λ*a*. *a* ∘ *f*) −' *A*. *poly-eval* (*a* ∘ *f*) *p* = *0*
hence *eq*: *poly-eval* (*a* ∘ *f*) *p* = *0* **if** *a* ∘ *f* ∈ *A* **for** *a* **using** *that* **by** *simp*
have *the-inv f* ∘ *f* = *id* **by** (*rule ext*) (*simp add*: *assms the-inv-f-f*)

hence *a*: *a* = *a* ∘ *the-inv f* ∘ *f* **by** (*simp add*: *comp-assoc*)
moreover assume *a* ∈ *A*
ultimately have (*a* ∘ *the-inv f*) ∘ *f* ∈ *A* **by** *simp*
hence *poly-eval* ((*a* ∘ *the-inv f*) ∘ *f*) *p* = *0* **by** (*rule eq*)
hence *poly-eval a p* = *0* **by** (*simp flip*: *a*)
}
thus *?thesis*
by (*auto simp*: *ideal-of-def poly-eval-map-indets simp flip*: *range-map-indets intro*!: *imageI*)
qed

lemma *variety-of-map-indets*: 𝒱 (*map-indets f* ‘ *F*) = (λ*a*. *a* ∘ *f*) −‘ 𝒱 *F*
by (*auto simp*: *variety-of-def poly-eval-map-indets*)

4.3 Radical Ideals

definition *radical* :: ′*a*::*monoid-mult set* ⇒ ′*a set* (‹√(-)› [*999*] *999*)
where *radical F* = {*f*. ∃ *m*. *f* ^ *m* ∈ *F*}

lemma *radicalI*: *f* ^ *m* ∈ *F* ⟹ *f* ∈ √*F*
by (*auto simp*: *radical-def*)

lemma *radicalE*:
assumes *f* ∈ √*F*
obtains *m* **where** *f* ^ *m* ∈ *F*
using *assms* **by** (*auto simp*: *radical-def*)

lemma *radical-empty* [*simp*]: √{} = {}
by (*simp add*: *radical-def*)

lemma *radical-UNIV* [*simp*]: √*UNIV* = *UNIV*
by (*simp add*: *radical-def*)

lemma *radical-ideal-eq-UNIV-iff*: √*ideal F* = *UNIV* ⟷ *ideal F* = *UNIV*
proof
assume √*ideal F* = *UNIV*
hence *1* ∈ √*ideal F* **by** *simp*
then obtain *m* **where** *1* ^ *m* ∈ *ideal F* **by** (*rule radicalE*)
thus *ideal F* = *UNIV* **by** (*simp add*: *ideal-eq-UNIV-iff-contains-one*)
qed *simp*

lemma *zero-in-radical-ideal* [*simp*]: *0* ∈ √*ideal F*
proof (*rule radicalI*)
show *0* ^ *1* ∈ *ideal F* **by** (*simp add*: *ideal.span-zero*)
qed

lemma *radical-mono*: *F* ⊆ *G* ⟹ √*F* ⊆ √*G*
by (*auto elim*!: *radicalE intro*: *radicalI*)

lemma *radical-superset*: $F \subseteq \sqrt{}F$
proof
  fix *f*
  assume $f \in F$
  hence f ^ $1 \in F$ **by** *simp*
  thus $f \in \sqrt{}F$ **by** (*rule radicalI*)
qed

lemma *radical-idem* [*simp*]: $\sqrt{}\sqrt{}F = \sqrt{}F$
proof
  show $\sqrt{}\sqrt{}F \subseteq \sqrt{}F$ **by** (*auto elim*!: *radicalE intro*: *radicalI simp flip*: *power-mult*)
qed (*fact radical-superset*)

lemma *radical-Int-subset*: $\sqrt{}(A \cap B) \subseteq \sqrt{}A \cap \sqrt{}B$
  by (*auto intro*: *radicalI elim*: *radicalE*)

lemma *radical-ideal-Int*: $\sqrt{}$(*ideal F* ∩ *ideal G*) = $\sqrt{}$*ideal F* ∩ $\sqrt{}$*ideal G*
  using *radical-Int-subset*
proof (*rule subset-antisym*)
  show $\sqrt{}$*ideal F* ∩ $\sqrt{}$*ideal G* ⊆ $\sqrt{}$(*ideal F* ∩ *ideal G*)
  proof
    fix *p*
    assume *p* ∈ $\sqrt{}$*ideal F* ∩ $\sqrt{}$*ideal G*
    hence *p* ∈ $\sqrt{}$*ideal F* **and** *p* ∈ $\sqrt{}$*ideal G* **by** *simp-all*
    from *this*(*1*) **obtain** *m1* **where** *p1*: *p* ^ *m1* ∈ *ideal F* **by** (*rule radicalE*)
    from ‹*p* ∈ $\sqrt{}$*ideal G*› **obtain** *m2* **where** *p* ^ *m2* ∈ *ideal G* **by** (*rule radicalE*)
    hence *p* ^ *m1* ∗ *p* ^ *m2* ∈ *ideal G* **by** (*rule ideal.span-scale*)
   moreover from *p1* **have** *p* ^ *m2* ∗ *p* ^ *m1* ∈ *ideal F* **by** (*rule ideal.span-scale*)
   ultimately have *p* ^ (*m1* + *m2*) ∈ *ideal F* ∩ *ideal G* **by** (*simp add*: *power-add*
mult.commute)
    thus *p* ∈ $\sqrt{}$(*ideal F* ∩ *ideal G*) **by** (*rule radicalI*)
  qed
qed

lemma *ideal-radical-ideal* [*simp*]: *ideal* ($\sqrt{}$*ideal F*) = $\sqrt{}$*ideal F* (**is** - = *?R*)
  unfolding *ideal.span-eq-iff*
proof (*rule ideal.subspaceI*)
  have *0* ^ *1* ∈ *ideal F* **by** (*simp add*: *ideal.span-zero*)
  thus *0* ∈ *?R* **by** (*rule radicalI*)
next
  fix *a b*
  assume *a* ∈ *?R*
  then obtain *m* **where** *a* ^ *m* ∈ *ideal F* **by** (*rule radicalE*)
  have *a*: *a* ^ *k* ∈ *ideal F* **if** *m* ≤ *k* **for** *k*
  proof −
    from ‹*a* ^ *m* ∈ -› **have** *a* ^ (*k* − *m* + *m*) ∈ *ideal F* **by** (*simp only*: *power-add*
ideal.span-scale)
    with *that* **show** *?thesis* **by** *simp*
  qed

assume $b \in ?R$
then obtain n **where** b ^ $n \in$ *ideal F* **by** (*rule radicalE*)
have b: b ^ $k \in$ *ideal F* **if** $n \le k$ **for** k
proof −
from ‹b ^ $n \in$ -› **have** b ^ $(k - n + n) \in$ *ideal F* **by** (*simp only*: *power-add ideal.span-scale*)
with *that* **show** *?thesis* **by** *simp*
qed
have $(a + b)$ ^ $(m + n) \in$ *ideal F* **unfolding** *binomial-ring*
proof (*rule ideal.span-sum*)
fix k
show *of-nat* $(m + n$ *choose* $k) * a$ ^ $k * b$ ^ $(m + n - k) \in$ *ideal F*
proof (*cases* $k \le m$)
case *True*
hence $n \le m + n - k$ **by** *simp*
hence b ^ $(m + n - k) \in$ *ideal F* **by** (*rule b*)
thus *?thesis* **by** (*rule ideal.span-scale*)
next
case *False*
hence $m \le k$ **by** *simp*
hence a ^ $k \in$ *ideal F* **by** (*rule a*)
hence *of-nat* $(m + n$ *choose* $k) * b$ ^ $(m + n - k) * a$ ^ $k \in$ *ideal F* **by** (*rule ideal.span-scale*)
thus *?thesis* **by** (*simp only*: *ac-simps*)
qed
qed
thus $a + b \in ?R$ **by** (*rule radicalI*)
next
fix c a
assume $a \in ?R$
then obtain m **where** a ^ $m \in$ *ideal F* **by** (*rule radicalE*)
hence $(c * a)$ ^ $m \in$ *ideal F* **by** (*simp only*: *power-mult-distrib ideal.span-scale*)
thus $c * a \in ?R$ **by** (*rule radicalI*)
qed

lemma *radical-ideal-of* [*simp*]: $\sqrt{}\mathcal{I}$ $A = \mathcal{I}$ (A::(- ⇒ -::*semiring-1-no-zero-divisors*) *set*)
proof
show $\sqrt{}\mathcal{I}$ $A \subseteq \mathcal{I}$ A **by** (*auto elim*!: *radicalE dest*!: *ideal-ofD intro*!: *ideal-ofI simp*: *poly-eval-power*)
qed (*fact radical-superset*)

lemma *variety-of-radical-ideal* [*simp*]: $\mathcal{V}$ ($\sqrt{}$*ideal F*) $= \mathcal{V}$ (F::(- $\Rightarrow_0$ -::*semiring-1-no-zero-divisors*) *set*)
proof
have $F \subseteq$ *ideal F* **by** (*rule ideal.span-superset*)
also have $\ldots \subseteq \sqrt{}$*ideal F* **by** (*rule radical-superset*)
finally show $\mathcal{V}$ ($\sqrt{}$*ideal F*) $\subseteq \mathcal{V}$ F **by** (*rule variety-of-antimono*)
next

show $\mathcal{V}$ *F* ⊆ $\mathcal{V}$ (√*ideal F*)
proof (*intro subsetI variety-ofI*)
fix *a f*
assume *a* ∈ $\mathcal{V}$ *F*
hence *a* ∈ $\mathcal{V}$ (*ideal F*) **by** *simp*
assume *f* ∈ √*ideal F*
then obtain *m* **where** *f* ⌢ *m* ∈ *ideal F* **by** (*rule radicalE*)
with ‹*a* ∈ $\mathcal{V}$ (*ideal F*)› **have** *poly-eval a* (*f* ⌢ *m*) = *0* **by** (*rule variety-ofD*)
thus *poly-eval a f* = *0* **by** (*simp add*: *poly-eval-power*)
qed
qed

lemma *image-map-indets-radical*:
assumes *inj f*
shows *map-indets f* ‘ √*F* = √(*map-indets f* ‘ (*F*::(- ⇒$_0$ ′*a*::*comm-ring-1*) *set*)) ∩ *P*[*range f*]
proof
show *map-indets f* ‘ √*F* ⊆ √(*map-indets f* ‘ *F*) ∩ *P*[*range f*]
by (*auto simp*: *radical-def simp flip*: *map-indets-power range-map-indets intro*!: *imageI*)
next
show √(*map-indets f* ‘ *F*) ∩ *P*[*range f*] ⊆ *map-indets f* ‘ √*F*
proof
fix *p*
assume *p* ∈ √(*map-indets f* ‘ *F*) ∩ *P*[*range f*]
hence *p* ∈ √(*map-indets f* ‘ *F*) **and** *p* ∈ *range* (*map-indets f*)
by (*simp-all add*: *range-map-indets*)
from *this*(*1*) **obtain** *m* **where** *p* ⌢ *m* ∈ *map-indets f* ‘ *F* **by** (*rule radicalE*)
then obtain *q* **where** *q* ∈ *F* **and** *p-m*: *p* ⌢ *m* = *map-indets f q* **..**
from *assms* **obtain** *g* **where** *g* ∘ *f* = *id* **and** *map-indets g* ∘ *map-indets f* = (*id*::- ⇒ - ⇒$_0$ ′*a*)
by (*rule map-indets-inverseE*)
hence *eq*: *map-indets g* (*map-indets f p*′) = *p*′ **for** *p*′::- ⇒$_0$ ′*a*
by (*simp add*: *pointfree-idE*)
from *p-m* **have** *map-indets g* (*p* ⌢ *m*) = *map-indets g* (*map-indets f q*) **by** (*rule arg-cong*)
hence (*map-indets g p*) ⌢ *m* = *q* **by** (*simp add*: *eq*)
from ‹*p* ∈ *range* -› **obtain** *p*′ **where** *p* = *map-indets f p*′ **..**
hence *p* = *map-indets f* (*map-indets g p*) **by** (*simp add*: *eq*)
moreover have *map-indets g p* ∈ √*F*
proof (*rule radicalI*)
from ‹*q* ∈ *F*› **show** *map-indets g p* ⌢ *m* ∈ *F* **by** (*simp add*: *p-m eq flip*: *map-indets-power*)
qed
ultimately show *p* ∈ *map-indets f* ‘ √*F* **by** (*rule image-eqI*)
qed
qed

4.4 Geometric Version of the Nullstellensatz

lemma *weak-Nullstellensatz-aux-1*:
  assumes ⋀*i*. *i* ∈ *I* ⟹ *g i* ∈ *ideal B*
  obtains *c* **where** *c* ∈ *ideal B* **and** (∏ *i*∈*I*. (*f i* + *g i*) ^ *m i*) = (∏ *i*∈*I*. *f i* ^ *m i*) + *c*
  using *assms*
proof (*induct I arbitrary*: *thesis rule*: *infinite-finite-induct*)
  case (*infinite I*)
  from *ideal.span-zero* **show** *?case* **by** (*rule infinite*) (*simp add*: *infinite*(*1*))
next
  case *empty*
  from *ideal.span-zero* **show** *?case* **by** (*rule empty*) *simp*
next
  case (*insert j I*)
  have *g i* ∈ *ideal B* **if** *i* ∈ *I* **for** *i* **by** (*rule insert.prems*) (*simp add*: *that*)
  with *insert.hyps*(*3*) **obtain** *c* **where** *c*: *c* ∈ *ideal B*
    and *1*: (∏ *i*∈*I*. (*f i* + *g i*) ^ *m i*) = (∏ *i*∈*I*. *f i* ^ *m i*) + *c* **by** *blast*
  define *k* **where** *k* = *m j*
  obtain *d* **where** *2*: (*f j* + *g j*) ^ *m j* = *f j* ^ *m j* + *d* ∗ *g j* **unfolding** *k-def*[*symmetric*]
  proof (*induct k arbitrary*: *thesis*)
    case *0*
    have (*f j* + *g j*) ^ *0* = *f j* ^ *0* + *0* ∗ *g j* **by** *simp*
    thus *?case* **by** (*rule 0*)
  next
    case (*Suc k*)
    obtain *d* **where** (*f j* + *g j*) ^ *k* = *f j* ^ *k* + *d* ∗ *g j* **by** (*rule Suc.hyps*)
    hence (*f j* + *g j*) ^ *Suc k* = (*f j* ^ *k* + *d* ∗ *g j*) ∗ (*f j* + *g j*) **by** *simp*
    also have . . . = *f j* ^ *Suc k* + (*f j* ^ *k* + *d* ∗ (*f j* + *g j*)) ∗ *g j* **by** (*simp add*: *algebra-simps*)
    finally show *?case* **by** (*rule Suc.prems*)
  qed
  from *c* **have** ∗: *f j* ^ *m j* ∗ *c* + (((∏ *i*∈*I*. *f i* ^ *m i*) + *c*) ∗ *d*) ∗ *g j* ∈ *ideal B* (**is** *?c* ∈ -)
    by (*intro ideal.span-add ideal.span-scale insert.prems insertI1*)
  from *insert.hyps*(*1*, *2*) **have** (∏ *i*∈*insert j I*. (*f i* + *g i*) ^ *m i*) =
                    (*f j* ^ *m j* + *d* ∗ *g j*) ∗ ((∏ *i*∈*I*. *f i* ^ *m i*) + *c*)
    by (*simp add*: *1 2*)
  also from *insert.hyps*(*1*, *2*) **have** . . . = (∏ *i*∈*insert j I*. *f i* ^ *m i*) + *?c* **by** (*simp add*: *algebra-simps*)
  finally have (∏ *i*∈*insert j I*. (*f i* + *g i*) ^ *m i*) = (∏ *i*∈*insert j I*. *f i* ^ *m i*) + *?c* **.**
  with ∗ **show** *?case* **by** (*rule insert.prems*)
qed

lemma *weak-Nullstellensatz-aux-2*:
  assumes *finite X* **and** *F* ⊆ *P*[*insert x X*] **and** *X* ⊆ {..<*x*::′*x*::{*countable*,*linorder*}}
    and *1* ∉ *ideal F* **and** *ideal F* ∩ *P*[{*x*}] ⊆ {*0*}
  obtains *a*::′*a*::*alg-closed-field* **where** *1* ∉ *ideal* (*poly-eval* (λ-. *monomial a 0*) ‘

focus {*x*} ‘ *F*)
proof −
  let *?x* = *monomial 1* (*Poly-Mapping.single x 1*)
  from *assms*(*3*) **have** *x* ∉ *X* **by** *blast*
  hence *eq1*: *insert x X* − {*x*} = *X* **and** *eq2*: *insert x X* − *X* = {*x*} **by** *blast+*

  interpret *i*: *pm-powerprod lex-pm lex-pm-strict*::(′*x* $\Rightarrow_0$ *nat*) ⇒ -
    unfolding *lex-pm-def lex-pm-strict-def*
   by *standard* (*simp-all add*: *lex-pm-zero-min lex-pm-plus-monotone flip*: *lex-pm-def*)
  have *lpp-focus*: *i.lpp* (*focus X g*) = *except* (*i.lpp g*) {*x*} **if** *g* ∈ *P*[*insert x X*] **for** *g*::- $\Rightarrow_0$ ′*a*
  proof (*cases g* = *0*)
    case *True*
    thus *?thesis* **by** *simp*
  next
    case *False*
    have *keys-focus-g*: *keys* (*focus X g*) = (λ*t*. *except t* {*x*}) ‘ *keys g*
      unfolding *keys-focus* **using** *refl*
    proof (*rule image-cong*)
      fix *t*
      assume *t* ∈ *keys g*
      also from *that* **have** . . . ⊆ .[*insert x X*] **by** (*rule PolysD*)
      finally have *keys t* ⊆ *insert x X* **by** (*rule PPsD*)
      hence *except t* (− *X*) = *except t* (*insert x X* ∩ − *X*)
      by (*metis* (*no-types*, *lifting*) *Int-commute except-keys-Int inf.orderE inf-left-commute*)
      also from ‹*x* ∉ *X*› **have** *insert x X* ∩ − *X* = {*x*} **by** *simp*
      finally show *except t* (− *X*) = *except t* {*x*} **.**
    qed
    show *?thesis*
    proof (*rule i.punit.lt-eqI-keys*)
      from *False* **have** *i.lpp g* ∈ *keys g* **by** (*rule i.punit.lt-in-keys*)
      thus *except* (*i.lpp g*) {*x*} ∈ *keys* (*focus X g*) **unfolding** *keys-focus-g* **by** (*rule imageI*)

      fix *t*
      assume *t* ∈ *keys* (*focus X g*)
      then obtain *s* **where** *s* ∈ *keys g* **and** *t*: *t* = *except s* {*x*} **unfolding** *keys-focus-g* **..**
      from *this*(*1*) **have** *lex-pm s* (*i.lpp g*) **by** (*rule i.punit.lt-max-keys*)
      moreover have *keys s* ∪ *keys* (*i.lpp g*) ⊆ {..*x*}
      proof (*rule Un-least*)
        from ‹*g* ∈ *P*[-]› **have** *keys g* ⊆ .[*insert x X*] **by** (*rule PolysD*)
        with ‹*s* ∈ *keys g*› **have** *s* ∈ .[*insert x X*] **..**
        hence *keys s* ⊆ *insert x X* **by** (*rule PPsD*)
        thus *keys s* ⊆ {..*x*} **using** *assms*(*3*) **by** *auto*

        from ‹*i.lpp g* ∈ *keys g*› ‹*keys g* ⊆ -› **have** *i.lpp g* ∈ .[*insert x X*] **..**
        hence *keys* (*i.lpp g*) ⊆ *insert x X* **by** (*rule PPsD*)
        thus *keys* (*i.lpp g*) ⊆ {..*x*} **using** *assms*(*3*) **by** *auto*

qed
ultimately show *lex-pm t* (*except* (*i.lpp g*) {*x*}) **unfolding** *t* **by** (*rule lex-pm-except-max*)
qed
qed

define *G* **where** *G* = *i.punit.reduced-GB F*
from *assms*(*1*) **have** *finite* (*insert x X*) **by** *simp*
hence *fin-G*: *finite G* **and** *G-sub*: *G* ⊆ *P*[*insert x X*] **and** *ideal-G*: *ideal G* = *ideal F*
and *0* ∉ *G* **and** *G-isGB*: *i.punit.is-Groebner-basis G* **unfolding** *G-def* **using** *assms*(*2*)
by (*rule i.finite-reduced-GB-Polys*, *rule i.reduced-GB-Polys*, *rule i.reduced-GB-ideal-Polys*, *rule i.reduced-GB-nonzero-Polys*, *rule i.reduced-GB-is-GB-Polys*)
define *G′* **where** *G′* = *focus X* ‘ *G*
from *fin-G* ‹*0* ∉ *G*› **have** *fin-G′*: *finite G′* **and** *0* ∉ *G′* **by** (*auto simp*: *G′-def*)
have *G′-sub*: *G′* ⊆ *P*[*X*] **by** (*auto simp*: *G′-def intro*: *focus-in-Polys*)
define *G″* **where** *G″* = *i.lcf* ‘ *G′*
from ‹*0* ∉ *G′*› **have** *0* ∉ *G″* **by** (*auto simp*: *G″-def i.punit.lc-eq-zero-iff*)
have *lookup-focus-in*: *lookup* (*focus X g*) *t* ∈ *P*[{*x*}] **if** *g* ∈ *G* **for** *g t*
proof −
have *lookup* (*focus X g*) *t* ∈ *range* (*lookup* (*focus X g*)) **by** (*rule rangeI*)
from *that G-sub* **have** *g* ∈ *P*[*insert x X*] **..**
hence *range* (*lookup* (*focus X g*)) ⊆ *P*[*insert x X* − *X*] **by** (*rule focus-coeffs-subset-Polys′*)
with ‹- ∈ *range* -› **have** *lookup* (*focus X g*) *t* ∈ *P*[*insert x X* − *X*] **..**
also have *insert x X* − *X* = {*x*} **by** (*simp only*: *eq2*)
finally show *?thesis* **.**
qed
hence *lcf-in*: *i.lcf* (*focus X g*) ∈ *P*[{*x*}] **if** *g* ∈ *G* **for** *g*
unfolding *i.punit.lc-def* **using** *that* **by** *blast*
have *G″-sub*: *G″* ⊆ *P*[{*x*}]
proof
fix *c*
assume *c* ∈ *G″*
then obtain *g′* **where** *g′* ∈ *G′* **and** *c*: *c* = *i.lcf g′* **unfolding** *G″-def* **..**
from ‹*g′* ∈ *G′*› **obtain** *g* **where** *g* ∈ *G* **and** *g′*: *g′* = *focus X g* **unfolding** *G′-def* **..**
from *this*(*1*) **show** *c* ∈ *P*[{*x*}] **unfolding** *c g′* **by** (*rule lcf-in*)
qed
define *P* **where** *P* = *poly-of-pm x* ‘ *G″*
from *fin-G′* **have** *fin-P*: *finite P* **by** (*simp add*: *P-def G″-def*)
have *0* ∉ *P*
proof
assume *0* ∈ *P*
then obtain *g″* **where** *g″* ∈ *G″* **and** *0* = *poly-of-pm x g″* **unfolding** *P-def* **..**
from *this*(*2*) **have** ∗: *keys g″* ∩ .[{*x*}] = {} **by** (*simp add*: *poly-of-pm-eq-zero-iff*)
from ‹*g″* ∈ *G″*› *G″-sub* **have** *g″* ∈ *P*[{*x*}] **..**
hence *keys g″* ⊆ .[{*x*}] **by** (*rule PolysD*)
with ∗ **have** *keys g″* = {} **by** *blast*

with ‹$g'' \in G''$› ‹$0 \notin G''$› **show** *False* **by** *simp*
qed
define Z **where** $Z = (\bigcup p\in P. \{z. \ poly\ p\ z = 0\})$
have *finite Z* **unfolding** *Z-def* **using** *fin-P*
proof (*rule finite-UN-I*)
fix *p*
assume $p \in P$
with ‹$0 \notin P$› **have** $p \neq 0$ **by** *blast*
thus *finite* $\{z. \ poly\ p\ z = 0\}$ **by** (*rule poly-roots-finite*)
qed
with *infinite-UNIV*[**where** $'a='a$] **have** $- Z \neq \{\}$ **using** *finite-compl* **by** *fastforce*
then obtain *a* **where** $a \notin Z$ **by** *blast*

have *a-nz*: *poly-eval* (λ-. *a*) (*i.lcf* (*focus X g*)) $\neq 0$ **if** $g \in G$ **for** *g*
proof −
from *that G-sub* **have** $g \in P[insert\ x\ X]$ **..**
have *poly-eval* (λ-. *a*) (*i.lcf* (*focus X g*)) = *poly* (*poly-of-pm x* (*i.lcf* (*focus X g*))) *a*
by (*rule sym*, *intro poly-eq-poly-eval′ lcf-in that*)
moreover have *poly-of-pm x* (*i.lcf* (*focus X g*)) $\in P$
by (*auto simp*: *P-def G′′-def G′-def that intro*!: *imageI*)
ultimately show *?thesis* **using** ‹$a \notin Z$› **by** (*simp add*: *Z-def*)
qed

let *?e* = *poly-eval* (λ-. *monomial a 0*)
have *lookup-e-focus*: *lookup* (*?e* (*focus* $\{x\}$ *g*)) *t* = *poly-eval* (λ-. *a*) (*lookup* (*focus X g*) *t*)
if $g \in P[insert\ x\ X]$ **for** *g t*
proof −
have *focus* $(- \{x\})$ *g* = *focus* $(- \{x\} \cap insert\ x\ X)$ *g* **by** (*rule sym*) (*rule focus-Int*, *fact*)
also have ... = *focus X g* **by** (*simp add*: *Int-commute eq1 flip*: *Diff-eq*)
finally show *?thesis* **by** (*simp add*: *lookup-poly-eval-focus*)
qed
have *lpp-e-focus*: *i.lpp* (*?e* (*focus* $\{x\}$ *g*)) = *except* (*i.lpp g*) $\{x\}$ **if** $g \in G$ **for** *g*
proof (*rule i.punit.lt-eqI-keys*)
from *that G-sub* **have** $g \in P[insert\ x\ X]$ **..**
hence *lookup* (*?e* (*focus* $\{x\}$ *g*)) (*except* (*i.lpp g*) $\{x\}$) = *poly-eval* (λ-. *a*) (*i.lcf* (*focus X g*))
by (*simp only*: *lookup-e-focus lpp-focus i.punit.lc-def*)
also from *that* **have** ... $\neq 0$ **by** (*rule a-nz*)
finally show *except* (*i.lpp g*) $\{x\} \in$ *keys* (*?e* (*focus* $\{x\}$ *g*)) **by** (*simp add*: *in-keys-iff*)

fix *t*
assume $t \in$ *keys* (*?e* (*focus* $\{x\}$ *g*))
hence $0 \neq$ *lookup* (*?e* (*focus* $\{x\}$ *g*)) *t* **by** (*simp add*: *in-keys-iff*)
also from ‹$g \in P[\text{-}]$› **have** *lookup* (*?e* (*focus* $\{x\}$ *g*)) *t* = *poly-eval* (λ-. *a*) (*lookup* (*focus X g*) *t*)

by (*rule lookup-e-focus*)
finally have $t \in$ *keys* (*focus X g*) by (*auto simp flip*: *lookup-not-eq-zero-eq-in-keys*)
hence *lex-pm t* (*i.lpp* (*focus X g*)) by (*rule i.punit.lt-max-keys*)
with ‹$g \in P$[-]› show *lex-pm t* (*except* (*i.lpp g*) $\{x\}$) by (*simp only*: *lpp-focus*)
qed

show *?thesis*
proof
define *G3* where *G3* = *?e* ‘ *focus* $\{x\}$ ‘ *G*
have *G3* $\subseteq$ *P*[*X*]
proof
fix *h*
assume $h \in$ *G3*
then obtain *h0* where *h0* $\in$ *G* and *h*: *h* = *?e* (*focus* $\{x\}$ *h0*) by (*auto simp*: *G3-def*)
from *this*(*1*) *G-sub* have *h0* $\in$ *P*[*insert x X*] ..
hence $h \in$ *P*[*insert x X* $-$ $\{x\}$] unfolding *h* by (*rule poly-eval-focus-in-Polys*)
thus $h \in$ *P*[*X*] by (*simp only*: *eq1*)
qed
from *fin-G* have *finite G3* by (*simp add*: *G3-def*)

have *ideal G3* $\cap$ *P*[$-$ $\{x\}$] = *?e* ‘ *focus* $\{x\}$ ‘ *ideal G*
by (*simp only*: *G3-def image-poly-eval-focus-ideal*)
also have ... = *ideal* (*?e* ‘ *focus* $\{x\}$ ‘ *F*) $\cap$ *P*[$-$ $\{x\}$]
by (*simp only*: *ideal-G image-poly-eval-focus-ideal*)
finally have *eq3*: *ideal G3* $\cap$ *P*[$-$ $\{x\}$] = *ideal* (*?e* ‘ *focus* $\{x\}$ ‘ *F*) $\cap$ *P*[$-$ $\{x\}$]
.
from *assms*(*1*) ‹*G3* $\subseteq$ *P*[*X*]› ‹*finite G3*› have *G3-isGB*: *i.punit.is-Groebner-basis G3*
proof (*rule i.punit.isGB-I-spoly-rep*[*simplified*, *OF dickson-grading-varnum*,
where *m=0*, *simplified i.dgrad-p-set-varnum*])
fix *g1 g2*
assume *g1* $\in$ *G3*
then obtain *g1′* where *g1′* $\in$ *G* and *g1*: *g1* = *?e* (*focus* $\{x\}$ *g1′*)
unfolding *G3-def* by *blast*
from *this*(*1*) have *lpp1*: *i.lpp g1* = *except* (*i.lpp g1′*) $\{x\}$ unfolding *g1* by (*rule lpp-e-focus*)
from ‹*g1′* $\in$ *G*› *G-sub* have *g1′* $\in$ *P*[*insert x X*] ..
assume *g2* $\in$ *G3*
then obtain *g2′* where *g2′* $\in$ *G* and *g2*: *g2* = *?e* (*focus* $\{x\}$ *g2′*)
unfolding *G3-def* by *blast*
from *this*(*1*) have *lpp2*: *i.lpp g2* = *except* (*i.lpp g2′*) $\{x\}$ unfolding *g2* by (*rule lpp-e-focus*)
from ‹*g2′* $\in$ *G*› *G-sub* have *g2′* $\in$ *P*[*insert x X*] ..

define *l* where *l* = *lcs* (*except* (*i.lpp g1′*) $\{x\}$) (*except* (*i.lpp g2′*) $\{x\}$)
define *c1* where *c1* = *i.lcf* (*focus X g1′*)
define *c2* where *c2* = *i.lcf* (*focus X g2′*)
define *c* where *c* = *poly-eval* (λ-. *a*) *c1* $*$ *poly-eval* (λ-. *a*) *c2*

define *s* **where** *s* = *c2* ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g1′*) {*x*}) *g1′* −
c1 ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g2′*) {*x*}) *g2′*
have *c1* ∈ *P*[{*x*}] **unfolding** *c1-def* **using** ‹*g1′* ∈ *G*› **by** (*rule lcf-in*)
hence *eval-c1*: *poly-eval* (λ-. *monomial a 0*) (*focus* {*x*} *c1*) = *monomial* (*poly-eval* (λ-. *a*) *c1*) *0*
by (*simp add*: *focus-Polys poly-eval-sum poly-eval-monomial monomial-power-map-scale times-monomial-monomial flip*: *punit.monomial-prod-sum monomial-sum*)
(*simp add*: *poly-eval-alt*)
have *c2* ∈ *P*[{*x*}] **unfolding** *c2-def* **using** ‹*g2′* ∈ *G*› **by** (*rule lcf-in*)
hence *eval-c2*: *poly-eval* (λ-. *monomial a 0*) (*focus* {*x*} *c2*) = *monomial* (*poly-eval* (λ-. *a*) *c2*) *0*
by (*simp add*: *focus-Polys poly-eval-sum poly-eval-monomial monomial-power-map-scale times-monomial-monomial flip*: *punit.monomial-prod-sum monomial-sum*)
(*simp add*: *poly-eval-alt*)

assume *spoly-nz*: *i.punit.spoly g1 g2* ≠ *0*
assume *g1* ≠ *0* **and** *g2* ≠ *0*
hence *g1′* ≠ *0* **and** *g2′* ≠ *0* **by** (*auto simp*: *g1 g2*)
have *c1-nz*: *poly-eval* (λ-. *a*) *c1* ≠ *0* **unfolding** *c1-def* **using** ‹*g1′* ∈ *G*› **by** (*rule a-nz*)
moreover have *c2-nz*: *poly-eval* (λ-. *a*) *c2* ≠ *0* **unfolding** *c2-def* **using** ‹*g2′* ∈ *G*› **by** (*rule a-nz*)
ultimately have *c* ≠ *0* **by** (*simp add*: *c-def*)
hence *inverse c* ≠ *0* **by** *simp*
from ‹*g1′* ∈ *P*[-]› **have** *except* (*i.lpp g1′*) {*x*} ∈ .[*insert x X* − {*x*}]
by (*intro PPs-closed-except′ i.PPs-closed-lpp*)
moreover from ‹*g2′* ∈ *P*[-]› **have** *except* (*i.lpp g2′*) {*x*} ∈ .[*insert x X* − {*x*}]
by (*intro PPs-closed-except′ i.PPs-closed-lpp*)
ultimately have *l* ∈ .[*insert x X* − {*x*}] **unfolding** *l-def* **by** (*rule PPs-closed-lcs*)
hence *l* ∈ .[*X*] **by** (*simp only*: *eq1*)
hence *l* ∈ .[*insert x X*] **by** *rule* (*rule PPs-mono*, *blast*)
moreover from ‹*c1* ∈ *P*[{*x*}]› **have** *c1* ∈ *P*[*insert x X*] **by** *rule* (*intro Polys-mono*, *simp*)
moreover from ‹*c2* ∈ *P*[{*x*}]› **have** *c2* ∈ *P*[*insert x X*] **by** *rule* (*intro Polys-mono*, *simp*)
ultimately have *s* ∈ *P*[*insert x X*] **using** ‹*g1′* ∈ *P*[-]› ‹*g2′* ∈ *P*[-]› **unfolding** *s-def*
by (*intro Polys-closed-minus Polys-closed-times Polys-closed-monom-mult PPs-closed-minus*)
have *s* ∈ *ideal G* **unfolding** *s-def times-monomial-left*[*symmetric*]
by (*intro ideal.span-diff ideal.span-scale ideal.span-base* ‹*g1′* ∈ *G*› ‹*g2′* ∈ *G*›)
with *G-isGB* **have** (*i.punit.red G*)** *s 0* **by** (*rule i.punit.GB-imp-zero-reducibility*[*simplified*])
with ‹*finite* (*insert x X*)› *G-sub fin-G* ‹*s* ∈ *P*[-]›
obtain *q0* **where** *1*: *s* = *0* + (∑ *g*∈*G*. *q0 g* ∗ *g*) **and** *2*: ⋀*g*. *q0 g* ∈ *P*[*insert*

x X]
and *3*: $\bigwedge$*g*. *lex-pm* (*i.lpp* (*q0 g* ∗ *g*)) (*i.lpp s*)
by (*rule i.punit.red-rtrancl-repE*[*simplified*, *OF dickson-grading-varnum*, **where** *m*=*0*,
simplified i.dgrad-p-set-varnum]) *blast*

define *q* **where** *q* = (λ*g*. *inverse c* · ($\sum$ *h*∈{*y*∈*G*. *?e* (*focus* {*x*} *y*) = *g*}. *?e* (*focus* {*x*} (*q0 h*))))

have *eq4*: *?e* (*focus* {*x*} (*monomial 1* (*l* − *t*))) = *monomial 1* (*l* − *t*) **for** *t*
proof −
have *focus* {*x*} (*monomial* (*1*::′*a*) (*l* − *t*)) = *monomial* (*monomial 1* (*l* − *t*)) *0*
proof (*intro focus-Polys-Compl Polys-closed-monomial PPs-closed-minus*)
from ‹*x* ∉ *X*› **have** *X* ⊆ − {*x*} **by** *simp*
hence .[*X*] ⊆ .[− {*x*}] **by** (*rule PPs-mono*)
with ‹*l* ∈ .[*X*]› **show** *l* ∈ .[− {*x*}] **..**
qed
thus *?thesis* **by** (*simp add*: *poly-eval-monomial*)
qed
from *c2-nz* **have** *eq5*: *inverse c* ∗ *poly-eval* (λ-. *a*) *c2* = *1* / *lookup g1* (*i.lpp g1*)
unfolding *lpp1* **using** ‹*g1*′ ∈ *P*[-]›
by (*simp add*: *c-def mult.assoc divide-inverse-commute g1 lookup-e-focus*
flip: *lpp-focus i.punit.lc-def c1-def*)
from *c1-nz* **have** *eq6*: *inverse c* ∗ *poly-eval* (λ-. *a*) *c1* = *1* / *lookup g2* (*i.lpp g2*)
unfolding *lpp2* **using** ‹*g2*′ ∈ *P*[-]›
by (*simp add*: *c-def mult.assoc mult.left-commute*[*of inverse* (*poly-eval* (λ-. *a*) *c1*)]
divide-inverse-commute g2 lookup-e-focus flip: *lpp-focus i.punit.lc-def c2-def*)
have *l-alt*: *l* = *lcs* (*i.lpp g1*) (*i.lpp g2*) **by** (*simp only*: *l-def lpp1 lpp2*)
have *spoly-eq*: *i.punit.spoly g1 g2* = (*inverse c*) · *?e* (*focus* {*x*} *s*)
by (*simp add*: *s-def focus-minus focus-times poly-eval-minus poly-eval-times eval-c1 eval-c2*
eq4 eq5 eq6 map-scale-eq-times times-monomial-monomial right-diff-distrib
i.punit.spoly-def Let-def
flip: *mult.assoc times-monomial-left g1 g2 lpp1 lpp2 l-alt*)
also have . . . = ($\sum$ *g*∈*G*. *inverse c* · (*?e* (*focus* {*x*} (*q0 g*)) ∗ *?e* (*focus* {*x*} *g*)))
by (*simp add*: *1 focus-sum poly-eval-sum focus-times poly-eval-times map-scale-sum-distrib-left*)
also have . . . = ($\sum$ *g*∈*G3*. $\sum$ *h*∈{*y*∈*G*. *?e* (*focus*{*x*} *y*) = *g*}.
inverse c · (*?e* (*focus* {*x*} (*q0 h*)) ∗ *?e* (*focus* {*x*} *h*)))
unfolding *G3-def image-image* **using** *fin-G* **by** (*rule sum.image-gen*)
also have . . . = ($\sum$ *g*∈*G3*. *inverse c* · ($\sum$ *h*∈{*y*∈*G*. *?e* (*focus*{*x*} *y*) = *g*}. *?e* (*focus* {*x*} (*q0 h*))) ∗ *g*)
by (*intro sum.cong refl*) (*simp add*: *map-scale-eq-times sum-distrib-left*

sum-distrib-right mult.assoc)
  also from *refl* **have** ... = ($\sum$ *g*∈*G3*. *q g* ∗ *g*) **by** (*rule sum.cong*) (*simp add*: *q-def sum-distrib-right*)
  finally have *i.punit.spoly g1 g2* = ($\sum$ *g*∈*G3*. *q g* ∗ *g*) **.**
  thus *i.punit.spoly-rep* (*varnum X*) *0 G3 g1 g2*
  proof (*rule i.punit.spoly-repI*[*simplified*, **where** *m*=*0* **and** *d*=*varnum X*,
      simplified i.dgrad-p-set-varnum])
    fix *g*
    show *q g* ∈ *P*[*X*] **unfolding** *q-def*
    proof (*intro Polys-closed-map-scale Polys-closed-sum*)
      fix *g0*
      from ‹*q0 g0* ∈ *P*[*insert x X*]› **have** *?e* (*focus* {*x*} (*q0 g0*)) ∈ *P*[*insert x X* − {*x*}]
        by (*rule poly-eval-focus-in-Polys*)
      thus *?e* (*focus* {*x*} (*q0 g0*)) ∈ *P*[*X*] **by** (*simp only*: *eq1*)
    qed

    assume *q g* ≠ *0* ∧ *g* ≠ *0*
    hence *q g* ≠ *0* **..**
    have *i.lpp* (*q g* ∗ *g*) = *i.lpp* ($\sum$ *h*∈{*y*∈*G*. *?e* (*focus* {*x*} *y*) = *g*}. *inverse c* · *?e* (*focus* {*x*} (*q0 h*)) ∗ *g*)
      by (*simp add*: *q-def map-scale-sum-distrib-left sum-distrib-right*)
    also have *lex-pm* ... (*i.ordered-powerprod-lin.Max*
        {*i.lpp* (*inverse c* · *?e* (*focus* {*x*} (*q0 h*)) ∗ *g*) | *h*. *h* ∈ {*y*∈*G*. *?e* (*focus* {*x*} *y*) = *g*}})
    (**is** *lex-pm* - (*i.ordered-powerprod-lin.Max ?A*)) **by** (*fact i.punit.lt-sum-le-Max*)
    also have *lex-pm* ... (*i.lpp s*)
    proof (*rule i.ordered-powerprod-lin.Max.boundedI*)
      from *fin-G* **show** *finite ?A* **by** *simp*
    next
      show *?A* ≠ {}
      proof
        assume *?A* = {}
        hence {*h* ∈ *G*. *?e* (*focus* {*x*} *h*) = *g*} = {} **by** *simp*
        hence *q g* = *0* **by** (*simp only*: *q-def sum.empty map-scale-zero-right*)
        with ‹*q g* ≠ *0*› **show** *False* **..**
      qed
    next
      fix *t*
      assume *t* ∈ *?A*
      then obtain *h* **where** *h* ∈ *G* **and** *g*[*symmetric*]: *?e* (*focus* {*x*} *h*) = *g*
        and *t* = *i.lpp* (*inverse c* · *?e* (*focus* {*x*} (*q0 h*)) ∗ *g*) **by** *blast*
      note *this*(*3*)
      also have *i.lpp* (*inverse c* · *?e* (*focus* {*x*} (*q0 h*)) ∗ *g*) =
          i.lpp (*inverse c* · (*?e* (*focus* {*x*} (*q0 h* ∗ *h*))))
      by (*simp only*: *map-scale-eq-times mult.assoc g poly-eval-times focus-times*)
      also from ‹*inverse c* ≠ *0*› **have** ... = *i.lpp* (*?e* (*focus* {*x*} (*q0 h* ∗ *h*)))
        by (*rule i.punit.lt-map-scale*)
      also have *lex-pm* ... (*i.lpp* (*q0 h* ∗ *h*))

proof (*rule i.punit.lt-le*, *rule ccontr*)
fix *u*
assume *lookup* (*?e* (*focus* {*x*} (*q0 h* ∗ *h*))) *u* ≠ *0*
hence *u* ∈ *keys* (*?e* (*focus* {*x*} (*q0 h* ∗ *h*))) **by** (*simp add*: *in-keys-iff*)
with *keys-poly-eval-focus-subset* **have** *u* ∈ (λ*v*. *except v* {*x*}) ' *keys* (*q0 h* ∗ *h*) **..**
then obtain *v* **where** *v* ∈ *keys* (*q0 h* ∗ *h*) **and** *u*: *u* = *except v* {*x*} **..**
have *lex-pm u* (*Poly-Mapping.single x* (*lookup v x*) + *u*)
by (*metis add.commute add.right-neutral i.plus-monotone-left lex-pm-zero-min*)
also have ... = *v* **by** (*simp only*: *u flip*: *plus-except*)
also from ‹*v* ∈ -› **have** *lex-pm v* (*i.lpp* (*q0 h* ∗ *h*)) **by** (*rule i.punit.lt-max-keys*)
finally have *lex-pm u* (*i.lpp* (*q0 h* ∗ *h*)) **.**
moreover assume *lex-pm-strict* (*i.lpp* (*q0 h* ∗ *h*)) *u*
ultimately show *False* **by** *simp*
qed
also have *lex-pm* ... (*i.lpp s*) **by** *fact*
finally show *lex-pm t* (*i.lpp s*) **.**
qed
also have *lex-pm-strict* ... *l*
proof (*rule i.punit.lt-less*)
from *spoly-nz* **show** *s* ≠ *0* **by** (*auto simp*: *spoly-eq*)
next
fix *t*
assume *lex-pm l t*

have *g1'* = *flatten* (*focus X g1'*) **by** *simp*
also have ... = *flatten* (*monomial c1* (*i.lpp* (*focus X g1'*)) + *i.punit.tail* (*focus X g1'*))
by (*simp only*: *c1-def flip*: *i.punit.leading-monomial-tail*)
also from ‹*g1'* ∈ *P*[-]› **have** ... = *punit.monom-mult 1* (*except* (*i.lpp g1'*) {*x*}) *c1* +
flatten (*i.punit.tail* (*focus X g1'*))
by (*simp only*: *flatten-plus flatten-monomial lpp-focus*)
finally have *punit.monom-mult 1* (*except* (*i.lpp g1'*) {*x*}) *c1* +
flatten (*i.punit.tail* (*focus X g1'*)) = *g1'* (**is** *?l* = -) **by** (*rule sym*)
moreover have *c2* ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g1'*) {*x*}) *?l* =
punit.monom-mult 1 l (*c1* ∗ *c2*) +
c2 ∗ *punit.monom-mult 1* (*l* − *i.lpp* (*focus X g1'*))
(*flatten* (*i.punit.tail* (*focus X g1'*)))
(**is** - = *punit.monom-mult 1 l* (*c1* ∗ *c2*) + *?a*)
by (*simp add*: *punit.monom-mult-dist-right punit.monom-mult-assoc l-def minus-plus adds-lcs*)
(*simp add*: *distrib-left lpp-focus* ‹*g1'* ∈ *P*[-]› *flip*: *times-monomial-left*)
ultimately have *a*: *c2* ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g1'*) {*x*}) *g1'* =
punit.monom-mult 1 l (*c1* ∗ *c2*) + *?a* **by** *simp*

have *g2′* = *flatten* (*focus X g2′*) **by** *simp*
also have ... = *flatten* (*monomial c2* (*i.lpp* (*focus X g2′*)) + *i.punit.tail* (*focus X g2′*))
by (*simp only*: *c2-def flip*: *i.punit.leading-monomial-tail*)
also from ‹*g2′* ∈ *P*[-]› **have** ... = *punit.monom-mult 1* (*except* (*i.lpp g2′*) {*x*}) *c2* +
flatten (*i.punit.tail* (*focus X g2′*))
by (*simp only*: *flatten-plus flatten-monomial lpp-focus*)
finally have *punit.monom-mult 1* (*except* (*i.lpp g2′*) {*x*}) *c2* +
flatten (*i.punit.tail* (*focus X g2′*)) = *g2′* (**is** *?l* = -) **by** (*rule sym*)
moreover have *c1* ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g2′*) {*x*}) *?l* =
punit.monom-mult 1 l (*c1* ∗ *c2*) +
c1 ∗ *punit.monom-mult 1* (*l* − *i.lpp* (*focus X g2′*))
(*flatten* (*i.punit.tail* (*focus X g2′*)))
(**is** - = *punit.monom-mult 1 l* (*c1* ∗ *c2*) + *?b*)
by (*simp add*: *punit.monom-mult-dist-right punit.monom-mult-assoc l-def minus-plus adds-lcs-2*)
(*simp add*: *distrib-left lpp-focus* ‹*g2′* ∈ *P*[-]› *flip*: *times-monomial-left*)
ultimately have *b*: *c1* ∗ *punit.monom-mult 1* (*l* − *except* (*i.lpp g2′*) {*x*}) *g2′* =
punit.monom-mult 1 l (*c1* ∗ *c2*) + *?b* **by** *simp*

have *lex-pm-strict-t*: *lex-pm-strict t* (*l* − *i.lpp* (*focus X h*) + *i.lpp* (*focus X h*))
if *t* ∈ *keys* (*d* ∗ *punit.monom-mult 1* (*l* − *i.lpp* (*focus X h*))
(*flatten* (*i.punit.tail* (*focus X h*))))
and *h* ∈ *G* **and** *d* ∈ *P*[{*x*}] **for** *d h*
proof −
have *0*: *lex-pm-strict* (*u* + *v*) *w* **if** *lex-pm-strict v w* **and** *w* ∈ .[*X*] **and** *u* ∈ .[{*x*}]
for *u v w* **using** *that*(*1*)
proof (*rule lex-pm-strict-plus-left*)
fix *y z*
assume *y* ∈ *keys w*
also from *that*(*2*) **have** ... ⊆ *X* **by** (*rule PPsD*)
also have ... ⊆ {..<*x*} **by** *fact*
finally have *y* < *x* **by** *simp*
assume *z* ∈ *keys u*
also from *that*(*3*) **have** ... ⊆ {*x*} **by** (*rule PPsD*)
finally show *y* < *z* **using** ‹*y* < *x*› **by** *simp*
qed
let *?h* = *focus X h*
from *that*(*2*) **have** *?h* ∈ *G′* **by** (*simp add*: *G′-def*)
with ‹*G′* ⊆ *P*[*X*]› **have** *?h* ∈ *P*[*X*] **..**
hence *i.lpp ?h* ∈ .[*X*] **by** (*rule i.PPs-closed-lpp*)
from *that*(*1*) **obtain** *t1 t2* **where** *t1* ∈ *keys d*
and *t2* ∈ *keys* (*punit.monom-mult 1* (*l* − *i.lpp ?h*) (*flatten* (*i.punit.tail*

?h)))
and t: t = t1 + t2 **by** (rule in-keys-timesE)
from this(2) **obtain** t3 **where** t3 ∈ keys (flatten (i.punit.tail ?h))
and t2: t2 = l − i.lpp ?h + t3 **by** (auto simp: punit.keys-monom-mult)
from this(1) **obtain** t4 t5 **where** t4 ∈ keys (i.punit.tail ?h)
and t5-in: t5 ∈ keys (lookup (i.punit.tail ?h) t4) **and** t3: t3 = t4 + t5
using keys-flatten-subset **by** blast
from this(1) **have** 1: lex-pm-strict t4 (i.lpp ?h) **by** (rule i.punit.keys-tail-less-lt)
from that(2) **have** lookup ?h t4 ∈ P[{x}] **by** (rule lookup-focus-in)
hence keys (lookup ?h t4) ⊆ .[{x}] **by** (rule PolysD)
moreover from t5-in **have** t5-in: t5 ∈ keys (lookup ?h t4)
by (simp add: i.punit.lookup-tail split: if-split-asm)
ultimately have t5 ∈ .[{x}] **..**
with 1 ‹i.lpp ?h ∈ -› **have** lex-pm-strict (t5 + t4) (i.lpp ?h) **by** (rule 0)
hence lex-pm-strict t3 (i.lpp ?h) **by** (simp only: t3 add.commute)
hence lex-pm-strict t2 (l − i.lpp ?h + i.lpp ?h) **unfolding** t2
by (rule i.plus-monotone-strict-left)
moreover from ‹l ∈ .[X]› ‹i.lpp ?h ∈ .[X]› **have** l − i.lpp ?h + i.lpp
?h ∈ .[X]
by (intro PPs-closed-plus PPs-closed-minus)
moreover from ‹t1 ∈ keys d› that(3) **have** t1 ∈ .[{x}] **by** (auto dest:
PolysD)
ultimately show ?thesis **unfolding** t **by** (rule 0)
qed
show lookup s t = 0
proof (rule ccontr)
assume lookup s t ≠ 0
hence t ∈ keys s **by** (simp add: in-keys-iff)
also have . . . = keys (?a − ?b) **by** (simp add: s-def a b)
also have . . . ⊆ keys ?a ∪ keys ?b **by** (fact keys-minus)
finally show False
proof
assume t ∈ keys ?a
hence lex-pm-strict t (l − i.lpp (focus X g1′) + i.lpp (focus X g1′))
using ‹g1′ ∈ G› ‹c2 ∈ P[{x}]› **by** (rule lex-pm-strict-t)
with ‹g1′ ∈ P[-]› **have** lex-pm-strict t l
by (simp add: lpp-focus l-def minus-plus adds-lcs)
with ‹lex-pm l t› **show** ?thesis **by** simp
next
assume t ∈ keys ?b
hence lex-pm-strict t (l − i.lpp (focus X g2′) + i.lpp (focus X g2′))
using ‹g2′ ∈ G› ‹c1 ∈ P[{x}]› **by** (rule lex-pm-strict-t)
with ‹g2′ ∈ P[-]› **have** lex-pm-strict t l
by (simp add: lpp-focus l-def minus-plus adds-lcs-2)
with ‹lex-pm l t› **show** ?thesis **by** simp
qed
qed
qed
also have . . . = lcs (i.lpp g1) (i.lpp g2) **by** (simp only: l-def lpp1 lpp2)

finally show *lex-pm-strict* (*i.lpp* (*q g* ∗ *g*)) (*lcs* (*i.lpp g1*) (*i.lpp g2*)) **.**
qed
qed
have *1* ∈ *ideal* (*?e* ' *focus* {*x*} ' *F*) ⟷ *1* ∈ *ideal* (*?e* ' *focus* {*x*} ' *F*) ∩ *P*[− {*x*}]
by (*simp add*: *one-in-Polys*)
also have . . . ⟷ *1* ∈ *ideal G3* **by** (*simp add*: *one-in-Polys flip*: *eq3*)
also have ¬ . . .
proof
note *G3-isGB*
moreover assume *1* ∈ *ideal G3*
moreover have *1* ≠ (*0*::- ⇒$_0$ *'a*) **by** *simp*
ultimately obtain *g* **where** *g* ∈ *G3* **and** *g* ≠ *0* **and** *i.lpp g adds i.lpp* (*1*::- ⇒$_0$ *'a*)
by (*rule i.punit.GB-adds-lt*[*simplified*])
from *this*(*3*) **have** *i.lpp g* = *0* **by** (*simp add*: *i.punit.lt-monomial adds-zero flip*: *single-one*)
hence *monomial* (*i.lcf g*) *0* = *g* **by** (*rule i.punit.lt-eq-min-term-monomial*[*simplified*])
from ‹*g* ∈ *G3*› **obtain** *g'* **where** *g'* ∈ *G* **and** *g*: *g* = *?e* (*focus* {*x*} *g'*) **by** (*auto simp*: *G3-def*)
from *this*(*1*) **have** *i.lpp g* = *except* (*i.lpp g'*) {*x*} **unfolding** *g* **by** (*rule lpp-e-focus*)
hence *keys* (*i.lpp g'*) ⊆ {*x*} **by** (*simp add*: ‹*i.lpp g* = *0*› *except-eq-zero-iff*)
have *g'* ∈ *P*[{*x*}]
proof (*intro PolysI subsetI PPsI*)
fix *t y*
assume *t* ∈ *keys g'*
hence *lex-pm t* (*i.lpp g'*) **by** (*rule i.punit.lt-max-keys*)
moreover assume *y* ∈ *keys t*
ultimately obtain *z* **where** *z* ∈ *keys* (*i.lpp g'*) **and** *z* ≤ *y* **by** (*rule lex-pm-keys-leE*)
with ‹*keys* (*i.lpp g'*) ⊆ {*x*}› **have** *x* ≤ *y* **by** *blast*
from ‹*g'* ∈ *G*› *G-sub* **have** *g'* ∈ *P*[*insert x X*] **..**
hence *indets g'* ⊆ *insert x X* **by** (*rule PolysD*)
moreover from ‹*y* ∈ -› ‹*t* ∈ -› **have** *y* ∈ *indets g'* **by** (*rule in-indetsI*)
ultimately have *y* ∈ *insert x X* **..**
thus *y* ∈ {*x*}
proof
assume *y* ∈ *X*
with *assms*(*3*) **have** *y* ∈ {..<*x*} **..**
with ‹*x* ≤ *y*› **show** *?thesis* **by** *simp*
qed *simp*
qed
moreover from ‹*g'* ∈ *G*› **have** *g'* ∈ *ideal G* **by** (*rule ideal.span-base*)
ultimately have *g'* ∈ *ideal F* ∩ *P*[{*x*}] **by** (*simp add*: *ideal-G*)
with *assms*(*5*) **have** *g'* = *0* **by** *blast*
hence *g* = *0* **by** (*simp add*: *g*)
with ‹*g* ≠ *0*› **show** *False* **..**
qed

finally show *1* ∉ *ideal* (*?e* ' *focus* {*x*} ' *F*) **.**
qed
qed

lemma *weak-Nullstellensatz-aux-3*:
assumes *F* ⊆ *P*[*insert x X*] **and** *x* ∉ *X* **and** *1* ∉ *ideal F* **and** ¬ *ideal F* ∩ *P*[{*x*}] ⊆ {*0*}
obtains *a*::'*a*::*alg-closed-field* **where** *1* ∉ *ideal* (*poly-eval* (λ-. *monomial a 0*) ' *focus* {*x*} ' *F*)
proof −
let *?x* = *monomial 1* (*Poly-Mapping.single x 1*)
from *assms*(*4*) **obtain** *f* **where** *f* ∈ *ideal F* **and** *f* ∈ *P*[{*x*}] **and** *f* ≠ *0* **by** *blast*
define *p* **where** *p* = *poly-of-pm x f*
from ‹*f* ∈ *P*[{*x*}]› ‹*f* ≠ *0*› **have** *p* ≠ *0*
by (*auto simp*: *p-def poly-of-pm-eq-zero-iff simp flip*: *keys-eq-empty dest*!: *PolysD*(*1*))
obtain *c A m* **where** *A*: *finite A* **and** *p*: *p* = *Polynomial.smult c* (∏ *a*∈*A*. [:− *a*, *1*:] ^ *m a*)
and ⋀*x*. *m x* = *0* ⟷ *x* ∉ *A* **and** *c* = *0* ⟷ *p* = *0* **and** ⋀*z*. *poly p z* = *0* ⟷ (*c* = *0* ∨ *z* ∈ *A*)
by (*rule linear-factorsE*) *blast*
from *this*(*4*, *5*) **have** *c* ≠ *0* **and** ⋀*z*. *poly p z* = *0* ⟷ *z* ∈ *A* **by** (*simp-all add*: ‹*p* ≠ *0*›)
have ∃ *a*∈*A*. *1* ∉ *ideal* (*poly-eval* (λ-. *monomial a 0*) ' *focus* {*x*} ' *F*)
proof (*rule ccontr*)
assume *asm*: ¬ (∃ *a*∈*A*. *1* ∉ *ideal* (*poly-eval* (λ-. *monomial a 0*) ' *focus* {*x*} ' *F*))
obtain *g h* **where** *g a* ∈ *ideal F* **and** *1*: *h a* ∗ (*?x* − *monomial a 0*) + *g a* = *1*
if *a* ∈ *A* **for** *a*
proof −
define *P* **where** *P* = (λ*gh a*. *fst gh* ∈ *ideal F* ∧ *fst gh* + *snd gh* ∗ (*?x* − *monomial a 0*) = *1*)
define *gh* **where** *gh* = (λ*a*. *SOME gh*. *P gh a*)
show *?thesis*
proof
fix *a*
assume *a* ∈ *A*
with *asm* **have** *1* ∈ *ideal* (*poly-eval* (λ-. *monomial a 0*) ' *focus* {*x*} ' *F*) **by** *blast*
hence *1* ∈ *poly-eval* (λ-. *monomial a 0*) ' *focus* {*x*} ' *ideal F*
by (*simp add*: *image-poly-eval-focus-ideal one-in-Polys*)
then obtain *g* **where** *g* ∈ *ideal F* **and** *1* = *poly-eval* (λ-. *monomial a 0*) (*focus* {*x*} *g*)
unfolding *image-image* **..**
note *this*(*2*)
also have *poly-eval* (λ-. *monomial a 0*) (*focus* {*x*} *g*) = *poly* (*poly-of-focus x g*) (*monomial a 0*)
by (*simp only*: *poly-poly-of-focus*)
also have ... = *poly* (*poly-of-focus x g*) (*?x* − (*?x* − *monomial a 0*)) **by** *simp*

also obtain *h* where ... = *poly* (*poly-of-focus x g*) *?x* − *h* ∗ (*?x* − *monomial a 0*)
by (*rule poly-minus-rightE*)
also have ... = *g* − *h* ∗ (*?x* − *monomial a 0*) by (*simp only*: *poly-poly-of-focus-monomial*)
finally have *g* − *h* ∗ (*?x* − *monomial a 0*) = *1* by (*rule sym*)
with ‹*g* ∈ *ideal F*› have *P* (*g*, − *h*) *a* by (*simp add*: *P-def*)
hence *P* (*gh a*) *a* unfolding *gh-def* by (*rule someI*)
thus *fst* (*gh a*) ∈ *ideal F* and *snd* (*gh a*) ∗ (*?x* − *monomial a 0*) + *fst* (*gh a*) = *1*
by (*simp-all only*: *P-def add.commute*)
qed
qed
from *this*(*1*) obtain *g′* where *g′* ∈ *ideal F*
and *2*: (∏ *a*∈*A*. (*h a* ∗ (*?x* − *monomial a 0*) + *g a*) ^ *m a*) =
(∏ *a*∈*A*. (*h a* ∗ (*?x* − *monomial a 0*)) ^ *m a*) + *g′*
by (*rule weak-Nullstellensatz-aux-1*)
have *1* = (∏ *a*∈*A*. (*h a* ∗ (*?x* − *monomial a 0*) + *g a*) ^ *m a*)
by (*rule sym*) (*intro prod.neutral ballI*, *simp only*: *1 power-one*)
also have ... = (∏ *a*∈*A*. *h a* ^ *m a*) ∗ (∏ *a*∈*A*. (*?x* − *monomial a 0*) ^ *m a*) + *g′*
by (*simp only*: *2 power-mult-distrib prod.distrib*)
also have (∏ *a*∈*A*. (*?x* − *monomial a 0*) ^ *m a*) = *pm-of-poly x* (∏ *a*∈*A*. [:− *a*, *1*:] ^ *m a*)
by (*simp add*: *pm-of-poly-prod pm-of-poly-pCons single-uminus punit.monom-mult-monomial flip*: *single-one*)
also from ‹*c* ≠ *0*› have ... = *monomial* (*inverse c*) *0* ∗ *pm-of-poly x p*
by (*simp add*: *p map-scale-assoc flip*: *map-scale-eq-times*)
also from ‹*f* ∈ *P*[{*x*}]› have ... = *monomial* (*inverse c*) *0* ∗ *f*
by (*simp only*: ‹*p* = *poly-of-pm x f*› *pm-of-poly-of-pm*)
finally have *1* = ((∏ *a*∈*A*. *h a* ^ *m a*) ∗ *monomial* (*inverse c*) *0*) ∗ *f* + *g′*
by (*simp only*: *mult.assoc*)
also from ‹*f* ∈ *ideal F*› ‹*g′* ∈ *ideal F*› have ... ∈ *ideal F* by (*intro ideal.span-add ideal.span-scale*)
finally have *1* ∈ *ideal F* .
with *assms*(*3*) show *False* ..
qed
then obtain *a* where *1* ∉ *ideal* (*poly-eval* (λ-. *monomial a 0*) ‘ *focus* {*x*} ‘ *F*) ..
thus *?thesis* ..
qed

theorem *weak-Nullstellensatz*:
assumes *finite X* and *F* ⊆ *P*[*X*] and 𝒱 *F* = ({}::(′*x*::{*countable*,*linorder*} ⇒ ′*a*::*alg-closed-field*) *set*)
shows *ideal F* = *UNIV*
unfolding *ideal-eq-UNIV-iff-contains-one*
proof (*rule ccontr*)
assume *1* ∉ *ideal F*
with *assms*(*1*, *2*) obtain *a* where *1* ∉ *ideal* (*poly-eval a* ‘ *F*)

```
  proof (induct X arbitrary: F thesis rule: finite-linorder-induct)
    case empty
    have F ⊆ {0}
    proof
      fix f
      assume f ∈ F
      with empty.prems(2) have f ∈ P[{}] ..
      then obtain c where f: f = monomial c 0 unfolding Polys-empty ..
      also have c = 0
      proof (rule ccontr)
        assume c ≠ 0
        from ‹f ∈ F› have f ∈ ideal F by (rule ideal.span-base)
        hence monomial (inverse c) 0 ∗ f ∈ ideal F by (rule ideal.span-scale)
        with ‹c ≠ 0› have 1 ∈ ideal F by (simp add: f times-monomial-monomial)
        with empty.prems(3) show False ..
      qed
      finally show f ∈ {0} by simp
    qed
    hence poly-eval 0 ‘ F ⊆ {0} by auto
    hence ideal (poly-eval 0 ‘ F) = {0} by simp
    hence 1 ∉ ideal (poly-eval 0 ‘ F) by (simp del: ideal-eq-zero-iff)
    thus ?case by (rule empty.prems)
  next
    case (insert x X)
    obtain a0 where 1 ∉ ideal (poly-eval (λ-. monomial a0 0) ‘ focus {x} ‘ F)
(is - ∉ ideal ?F)
    proof (cases ideal F ∩ P[{x}] ⊆ {0})
      case True
      with insert.hyps(1) insert.prems(2) insert.hyps(2) insert.prems(3) obtain
a0
        where 1 ∉ ideal (poly-eval (λ-. monomial a0 0) ‘ focus {x} ‘ F)
        by (rule weak-Nullstellensatz-aux-2)
      thus ?thesis ..
    next
      case False
      from insert.hyps(2) have x ∉ X by blast
      with insert.prems(2) obtain a0 where 1 ∉ ideal (poly-eval (λ-. monomial
a0 0) ‘ focus {x} ‘ F)
        using insert.prems(3) False by (rule weak-Nullstellensatz-aux-3)
      thus ?thesis ..
    qed
    moreover have ?F ⊆ P[X]
    proof −
      {
        fix f
        assume f ∈ F
        with insert.prems(2) have f ∈ P[insert x X] ..
        hence poly-eval (λ-. monomial a0 0) (focus {x} f) ∈ P[insert x X − {x}]
          by (rule poly-eval-focus-in-Polys)
```

also have ... ⊆ $P[X]$ **by** (*rule Polys-mono*) *simp*
finally have *poly-eval* (λ-. *monomial a0 0*) (*focus* {*x*} *f*) ∈ $P[X]$ **.**
}
thus *?thesis* **by** *blast*
qed
ultimately obtain *a1* **where** *1* ∉ *ideal* (*poly-eval a1* ‘ *?F*) **using** *insert.hyps*(*3*) **by** *blast*
also have *poly-eval a1* ‘ *?F* = *poly-eval* (*a1*(*x* := *poly-eval a1* (*monomial a0 0*))) ‘ *F*
by (*simp add*: *image-image poly-eval-poly-eval-focus fun-upd-def*)
finally show *?case* **by** (*rule insert.prems*)
qed
hence *ideal* (*poly-eval a* ‘ *F*) ≠ *UNIV* **by** (*simp add*: *ideal-eq-UNIV-iff-contains-one*)
hence *ideal* (*poly-eval a* ‘ *F*) = {*0*} **using** *ideal-field-disj*[*of poly-eval a* ‘ *F*] **by** *blast*
hence *poly-eval a* ‘ *F* ⊆ {*0*} **by** *simp*
hence *a* ∈ $\mathcal{V}$ *F* **by** (*rule variety-ofI-alt*)
thus *False* **by** (*simp add*: *assms*(*3*))
qed

lemma *radical-idealI*:
assumes *finite X* **and** *F* ⊆ $P[X]$ **and** *f* ∈ $P[X]$ **and** *x* ∉ *X*
and $\mathcal{V}$ (*insert* (*1* − *punit.monom-mult 1* (*Poly-Mapping.single x 1*) *f*) *F*) = {}
shows (*f*::(′*x*::{*countable*,*linorder*} $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*alg-closed-field*) ∈ √*ideal F*
proof (*cases f* = *0*)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
from *assms*(*4*) **have** $P[X]$ ⊆ $P[-\ \{x\}]$ **by** (*auto simp*: *Polys-alt*)
with *assms*(*3*) **have** *f* ∈ $P[-\ \{x\}]$ **..**
let *?x* = *Poly-Mapping.single x 1*
let *?f* = *punit.monom-mult 1 ?x f*
from *assms*(*1*) **have** *finite* (*insert x X*) **by** *simp*
moreover have *insert* (*1* − *?f*) *F* ⊆ *P*[*insert x X*] **unfolding** *insert-subset*
proof (*intro conjI Polys-closed-minus one-in-Polys Polys-closed-monom-mult PPs-closed-single*)
have $P[X]$ ⊆ *P*[*insert x X*] **by** (*rule Polys-mono*) *blast*
with *assms*(*2*, *3*) **show** *f* ∈ *P*[*insert x X*] **and** *F* ⊆ *P*[*insert x X*] **by** *blast*+
qed *simp*
ultimately have *ideal* (*insert* (*1* − *?f*) *F*) = *UNIV*
using *assms*(*5*) **by** (*rule weak-Nullstellensatz*)
hence *1* ∈ *ideal* (*insert* (*1* − *?f*) *F*) **by** *simp*
then obtain *F*′ *q* **where** *fin*′: *finite F*′ **and** *F*′*-sub*: *F*′ ⊆ *insert* (*1* − *?f*) *F*
and *eq*: *1* = ($\sum$*f*′∈*F*′. *q f*′ ∗ *f*′) **by** (*rule ideal.spanE*)
show *f* ∈ √*ideal F*
proof (*cases 1* − *?f* ∈ *F*′)
case *True*
define *g* **where** *g* = (λ*x*::(′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*. *Fract x 1*)

define F'' **where** $F'' = F' - \{1 - ?f\}$
define *q0* **where** *q0* = *q* (*1* − *?f*)
have *g-0*: *g 0* = *0* **by** (*simp add*: *g-def fract-collapse*)
have *g-1*: *g 1* = *1* **by** (*simp add*: *g-def fract-collapse*)
have *g-plus*: *g* (*a* + *b*) = *g a* + *g b* **for** *a b* **by** (*simp add*: *g-def*)
have *g-minus*: *g* (*a* − *b*) = *g a* − *g b* **for** *a b* **by** (*simp add*: *g-def*)
have *g-times*: *g* (*a* ∗ *b*) = *g a* ∗ *g b* **for** *a b* **by** (*simp add*: *g-def*)
from *fin′* **have** *fin″*: *finite* F'' **by** (*simp add*: F''*-def*)
from F'*-sub* **have** F''*-sub*: $F'' \subseteq F$ **by** (*auto simp*: F''*-def*)

have *focus* {*x*} *?f* = *monomial 1 ?x* ∗ *focus* {*x*} *f*
by (*simp add*: *focus-times focus-monomial except-single flip*: *times-monomial-left*)
also from ‹*f* ∈ *P*[− {*x*}]› **have** *focus* {*x*} *f* = *monomial f 0* **by** (*rule focus-Polys-Compl*)
finally have *focus* {*x*} *?f* = *monomial f ?x* **by** (*simp add*: *times-monomial-monomial*)
hence *eq1*: *poly* (*map-poly g* (*poly-of-focus x* (*1* − *?f*))) (*Fract 1 f*) = *0*
by (*simp add*: *poly-of-focus-def focus-minus poly-of-pm-minus poly-of-pm-monomial PPs-closed-single map-poly-minus g-0 g-1 g-minus map-poly-monom poly-monom*)
(*simp add*: *g-def Fract-same* ‹*f* ≠ *0*›)
have *eq2*: *poly* (*map-poly g* (*poly-of-focus x f′*)) (*Fract 1 f*) = *Fract f′ 1* **if** *f′* ∈ F'' **for** *f′*
proof −
from *that* F''*-sub* **have** *f′* ∈ *F* **..**
with *assms*(*2*) **have** *f′* ∈ *P*[*X*] **..**
with ‹*P*[*X*] ⊆ -› **have** *f′* ∈ *P*[− {*x*}] **..**
hence *focus* {*x*} *f′* = *monomial f′ 0* **by** (*rule focus-Polys-Compl*)
thus *?thesis*
by (*simp add*: *poly-of-focus-def focus-minus poly-of-pm-minus poly-of-pm-monomial zero-in-PPs map-poly-minus g-0 g-1 g-minus map-poly-monom poly-monom*)
(*simp only*: *g-def*)
qed

define *p0m0* **where** *p0m0* = (λ*f′*. *SOME z*. *poly* (*map-poly g* (*poly-of-focus x* (*q f′*))) (*Fract 1 f*) = *Fract* (*fst z*) (*f* ^ *snd z*))
define *p0* **where** *p0* = *fst* ∘ *p0m0*
define *m0* **where** *m0* = *snd* ∘ *p0m0*
define *m* **where** *m* = *Max* (*m0* ‘ F'')
have *eq3*: *poly* (*map-poly g* (*poly-of-focus x* (*q f′*))) (*Fract 1 f*) = *Fract* (*p0 f′*) (*f* ^ *m0 f′*)
for *f′*
proof −
have *g a* = *0* ⟷ *a* = *0* **for** *a* **by** (*simp add*: *g-def Fract-eq-zero-iff*)
hence *set* (*Polynomial.coeffs* (*map-poly g* (*poly-of-focus x* (*q f′*)))) ⊆ *range* (λ*x*. *Fract x 1*)
by (*auto simp*: *set-coeffs-map-poly g-def*)
then obtain *p m′* **where** *poly* (*map-poly g* (*poly-of-focus x* (*q f′*))) (*Fract 1*

$f) = Fract\ p\ (f \,\widehat{}\, m')$
  by (*rule poly-Fract*)
  hence *poly* (*map-poly g* (*poly-of-focus x* (*q f'*))) (*Fract 1 f*) = *Fract* (*fst* (*p*, *m'*)) (*f* ^ *snd* (*p*, *m'*))
  by *simp*
  thus *?thesis* **unfolding** *p0-def m0-def p0m0-def o-def* **by** (*rule someI*)
qed

note *eq*
also from *True fin'* **have** $(\sum f'{\in}F'.\ q\ f' * f') = q0 * (1 - ?f) + (\sum f'{\in}F''.\ q\ f' * f')$
  by (*simp add*: *q0-def F''-def sum.remove*)
finally have $poly\text{-}of\text{-}focus\ x\ 1 = poly\text{-}of\text{-}focus\ x\ (q0 * (1 - ?f) + (\sum f'{\in}F''.\ q\ f' * f'))$
  by (*rule arg-cong*)
hence $1 = poly\ (map\text{-}poly\ g\ (poly\text{-}of\text{-}focus\ x\ (q0 * (1 - ?f) + (\sum f'{\in}F''.\ q\ f' * f')))) (Fract\ 1\ f)$
  by (*simp add*: *g-1*)
also have $\ldots = poly\ (map\text{-}poly\ g\ (poly\text{-}of\text{-}focus\ x\ (\sum f'{\in}F''.\ q\ f' * f')))\ (Fract\ 1\ f)$
  by (*simp only*: *poly-of-focus-plus map-poly-plus g-0 g-plus g-times poly-add poly-of-focus-times map-poly-times poly-mult eq1 mult-zero-right add-0-left*)
also have $\ldots = (\sum f'{\in}F''.\ Fract\ (p0\ f')\ (f \,\widehat{}\, m0\ f') * Fract\ f'\ 1)$
  by (*simp only*: *poly-of-focus-sum poly-of-focus-times map-poly-sum map-poly-times g-0 g-plus g-times poly-sum poly-mult eq2 eq3 cong*: *sum.cong*)
finally have $Fract\ (f \,\widehat{}\, m)\ 1 = Fract\ (f \,\widehat{}\, m)\ 1 * (\sum f'{\in}F''.\ Fract\ (p0\ f' * f')\ (f \,\widehat{}\, m0\ f'))$
  by *simp*
also have $\ldots = (\sum f'{\in}F''.\ Fract\ (f \,\widehat{}\, m * (p0\ f' * f'))\ (f \,\widehat{}\, m0\ f'))$
  by (*simp add*: *sum-distrib-left*)
also from *refl* **have** $\ldots = (\sum f'{\in}F''.\ Fract\ ((f \,\widehat{}\, (m - m0\ f') * p0\ f') * f')\ 1)$
proof (*rule sum.cong*)
  fix f'
  assume $f' \in F''$
  hence $m0\ f' \in m0\ `\ F''$ **by** (*rule imageI*)
  with - **have** $m0\ f' \le m$ **unfolding** *m-def* **by** (*rule Max-ge*) (*simp add*: *fin''*)
  hence $f \,\widehat{}\, m = f \,\widehat{}\, (m0\ f') * f \,\widehat{}\, (m - m0\ f')$ **by** (*simp flip*: *power-add*)
  hence $Fract\ (f \,\widehat{}\, m * (p0\ f' * f'))\ (f \,\widehat{}\, m0\ f') = Fract\ (f \,\widehat{}\, m0\ f')\ (f \,\widehat{}\, m0\ f') * Fract\ (f \,\widehat{}\, (m - m0\ f') * (p0\ f' * f'))\ 1$
  by (*simp add*: *ac-simps*)
  also from ‹$f \ne 0$› **have** $Fract\ (f \,\widehat{}\, m0\ f')\ (f \,\widehat{}\, m0\ f') = 1$ **by** (*simp add*: *Fract-same*)
  finally show $Fract\ (f \,\widehat{}\, m * (p0\ f' * f'))\ (f \,\widehat{}\, m0\ f') = Fract\ (f \,\widehat{}\, (m - m0\ f') * p0\ f' * f')\ 1$
  by (*simp add*: *ac-simps*)
qed
also from *fin''* **have** $\ldots = Fract\ (\sum f'{\in}F''.\ (f \,\widehat{}\, (m - m0\ f') * p0\ f') * f')\ 1$

```
      by (induct F′′) (simp-all add: fract-collapse)
    finally have f ^ m = (∑f′∈F′′. (f ^ (m − m0 f′) ∗ p0 f′) ∗ f′)
      by (simp add: eq-fract)
    also have ... ∈ ideal F′′ by (rule ideal.sum-in-spanI)
    also from ‹F′′ ⊆ F› have ... ⊆ ideal F by (rule ideal.span-mono)
    finally show f ∈ √ideal F by (rule radicalI)
  next
    case False
    with F′-sub have F′ ⊆ F by blast
    have 1 ∈ ideal F′ unfolding eq by (rule ideal.sum-in-spanI)
    also from ‹F′ ⊆ F› have ... ⊆ ideal F by (rule ideal.span-mono)
    finally have ideal F = UNIV by (simp only: ideal-eq-UNIV-iff-contains-one)
    thus ?thesis by simp
  qed
qed

corollary radical-idealI-extend-indets:
  assumes finite X and F ⊆ P[X]
   and 𝒱 (insert (1 − punit.monom-mult 1 (Poly-Mapping.single None 1) (extend-indets
f))
                           (extend-indets ‘ F)) = {}
  shows (f::(-::{countable,linorder} ⇒0 nat) ⇒0 -::alg-closed-field) ∈ √ideal F
proof −
  define Y where Y = X ∪ indets f
  from assms(1) have fin-Y: finite Y by (simp add: Y-def finite-indets)
  have P[X] ⊆ P[Y] by (rule Polys-mono) (simp add: Y-def)
  with assms(2) have F-sub: F ⊆ P[Y] by (rule subset-trans)
  have f-in: f ∈ P[Y] by (simp add: Y-def Polys-alt)

  let ?F = extend-indets ‘ F
  let ?f = extend-indets f
  let ?X = Some ‘ Y
  from fin-Y have finite ?X by (rule finite-imageI)
  moreover from F-sub have ?F ⊆ P[?X]
     by (auto simp: indets-extend-indets intro!: PolysI-alt imageI dest!: PolysD(2)
subsetD[of F])
  moreover from f-in have ?f ∈ P[?X]
    by (auto simp: indets-extend-indets intro!: PolysI-alt imageI dest!: PolysD(2))
  moreover have None ∉ ?X by simp
  ultimately have ?f ∈ √ideal ?F using assms(3) by (rule radical-idealI)
  also have ?f ∈ √ideal ?F ←→ f ∈ √ideal F
  proof
    assume f ∈ √ideal F
    then obtain m where f ^ m ∈ ideal F by (rule radicalE)
    hence extend-indets (f ^ m) ∈ extend-indets ‘ ideal F by (rule imageI)
   with extend-indets-ideal-subset have ?f ^ m ∈ ideal ?F unfolding extend-indets-power
..
    thus ?f ∈ √ideal ?F by (rule radicalI)
  next
```

```
    assume ?f ∈ √ideal ?F
    then obtain m where ?f ^ m ∈ ideal ?F by (rule radicalE)
    moreover have ?f ^ m ∈ P[− {None}]
     by (rule Polys-closed-power) (auto intro!: PolysI-alt simp: indets-extend-indets)
    ultimately have extend-indets (f ^ m) ∈ extend-indets ‘ ideal F
      by (simp add: extend-indets-ideal extend-indets-power)
   hence f ^ m ∈ ideal F by (simp only: inj-image-mem-iff[OF inj-extend-indets])
    thus f ∈ √ideal F by (rule radicalI)
  qed
  finally show ?thesis .
qed

theorem Nullstellensatz:
  assumes finite X and F ⊆ P[X]
    and (f::(-::{countable,linorder} ⇒₀ nat) ⇒₀ -::alg-closed-field) ∈ ℐ (𝒱 F)
  shows f ∈ √ideal F
  using assms(1, 2)
proof (rule radical-idealI-extend-indets)
  let ?f = punit.monom-mult 1 (monomial 1 None) (extend-indets f)
  show 𝒱 (insert (1 − ?f) (extend-indets ‘ F)) = {}
  proof (intro subset-antisym subsetI)
    fix a
    assume a ∈ 𝒱 (insert (1 − ?f) (extend-indets ‘ F))
    moreover have 1 − ?f ∈ insert (1 − ?f) (extend-indets ‘ F) by simp
    ultimately have poly-eval a (1 − ?f) = 0 by (rule variety-ofD)
    hence poly-eval a (extend-indets f) ≠ 0
     by (auto simp: poly-eval-minus poly-eval-times simp flip: times-monomial-left)
    hence poly-eval (a ∘ Some) f ≠ 0 by (simp add: poly-eval-extend-indets)
    have a ∘ Some ∈ 𝒱 F
    proof (rule variety-ofI)
      fix f'
      assume f' ∈ F
      hence extend-indets f' ∈ insert (1 − ?f) (extend-indets ‘ F) by simp
      with ‹a ∈ -› have poly-eval a (extend-indets f') = 0 by (rule variety-ofD)
      thus poly-eval (a ∘ Some) f' = 0 by (simp only: poly-eval-extend-indets)
    qed
    with assms(3) have poly-eval (a ∘ Some) f = 0 by (rule ideal-ofD)
    with ‹poly-eval (a ∘ Some) f ≠ 0› show a ∈ {} ..
  qed simp
qed

theorem strong-Nullstellensatz:
  assumes finite X and F ⊆ P[X]
 shows ℐ (𝒱 F) = √ideal (F::((-::{countable,linorder} ⇒₀ nat) ⇒₀ -::alg-closed-field)
set)
proof (intro subset-antisym subsetI)
  fix f
  assume f ∈ ℐ (𝒱 F)
  with assms show f ∈ √ideal F by (rule Nullstellensatz)
```

qed (*metis ideal-ofI variety-ofD variety-of-radical-ideal*)

The following lemma can be used for actually *deciding* whether a polynomial is contained in the radical of an ideal or not.

lemma *radical-ideal-iff*:
  assumes *finite X* **and** $F \subseteq P[X]$ **and** $f \in P[X]$ **and** $x \notin X$
  shows (*f*::(-::{*countable*,*linorder*} $\Rightarrow_0$ *nat*) $\Rightarrow_0$ -::*alg-closed-field*) $\in \sqrt{ideal\ F} \longleftrightarrow$
      1 $\in$ *ideal* (*insert* (*1* $-$ *punit.monom-mult 1* (*Poly-Mapping.single x 1*) *f*) *F*)
proof $-$
  let *?f* = *punit.monom-mult 1* (*Poly-Mapping.single x 1*) *f*
  show *?thesis*
  proof
    assume $f \in \sqrt{ideal\ F}$
    then obtain *m* **where** *f* ⌢ *m* $\in$ *ideal F* **by** (*rule radicalE*)
    from *assms*(*1*) **have** *finite* (*insert x X*) **by** *simp*
    moreover have *insert* (*1* $-$ *?f*) *F* $\subseteq$ *P*[*insert x X*] **unfolding** *insert-subset*
     proof (*intro conjI Polys-closed-minus one-in-Polys Polys-closed-monom-mult PPs-closed-single*)
      have *P*[*X*] $\subseteq$ *P*[*insert x X*] **by** (*rule Polys-mono*) *blast*
      with *assms*(*2*, *3*) **show** *f* $\in$ *P*[*insert x X*] **and** *F* $\subseteq$ *P*[*insert x X*] **by** *blast+*
    qed *simp*
    moreover have $\mathcal{V}$ (*insert* (*1* $-$ *?f*) *F*) = {}
    proof (*intro subset-antisym subsetI*)
      fix *a*
      assume *a* $\in \mathcal{V}$ (*insert* (*1* $-$ *?f*) *F*)
      moreover have *1* $-$ *?f* $\in$ *insert* (*1* $-$ *?f*) *F* **by** *simp*
      ultimately have *poly-eval a* (*1* $-$ *?f*) = *0* **by** (*rule variety-ofD*)
      hence *poly-eval a* (*f* ⌢ *m*) $\neq$ *0*
        by (*auto simp*: *poly-eval-minus poly-eval-times poly-eval-power simp flip*: *times-monomial-left*)
       from ‹*a* $\in$ -› **have** *a* $\in \mathcal{V}$ (*ideal* (*insert* (*1* $-$ *?f*) *F*)) **by** (*simp only*: *variety-of-ideal*)
     moreover from ‹*f* ⌢ *m* $\in$ *ideal F*› *ideal.span-mono* **have** *f* ⌢ *m* $\in$ *ideal* (*insert* (*1* $-$ *?f*) *F*)
       by (*rule rev-subsetD*) *blast*
      ultimately have *poly-eval a* (*f* ⌢ *m*) = *0* **by** (*rule variety-ofD*)
      with ‹*poly-eval a* (*f* ⌢ *m*) $\neq$ *0*› **show** *a* $\in$ {} **..**
    qed *simp*
   ultimately have *ideal* (*insert* (*1* $-$ *?f*) *F*) = *UNIV* **by** (*rule weak-Nullstellensatz*)
    thus *1* $\in$ *ideal* (*insert* (*1* $-$ *?f*) *F*) **by** *simp*
  next
    assume *1* $\in$ *ideal* (*insert* (*1* $-$ *?f*) *F*)
    have $\mathcal{V}$ (*insert* (*1* $-$ *?f*) *F*) = {}
    proof (*intro subset-antisym subsetI*)
      fix *a*
      assume *a* $\in \mathcal{V}$ (*insert* (*1* $-$ *?f*) *F*)
      hence *a* $\in \mathcal{V}$ (*ideal* (*insert* (*1* $-$ *?f*) *F*)) **by** (*simp only*: *variety-of-ideal*)
      hence *poly-eval a 1* = *0* **using** ‹*1* $\in$ -› **by** (*rule variety-ofD*)

thus $a \in \{\}$ **by** *simp*
qed *simp*
with *assms* **show** $f \in \sqrt{}$*ideal F* **by** (*rule radical-idealI*)
qed
qed

end

5 Field-Theoretic Version of Hilbert's Nullstellensatz

theory *Nullstellensatz-Field*
imports *Nullstellensatz HOL−Types-To-Sets.Types-To-Sets*
begin

Building upon the geometric version of Hilbert's Nullstellensatz in *Nullstellensatz.Nullstellensatz*, we prove its field-theoretic version here. To that end we employ the 'types to sets' methodology.

5.1 Getting Rid of Sort Constraints in Geometric Version

We can use the 'types to sets' approach to get rid of the *countable* and *linorder* sort constraints on the type of indeterminates in the geometric version of the Nullstellensatz. Once the 'types to sets' methodology is integrated as a standard component into the main library of Isabelle, the theorems in *Nullstellensatz.Nullstellensatz* could be replaced by their counterparts in this section.

lemmas *radical-idealI-internalized* = *radical-idealI*[*unoverload-type* ′*x*]

lemma *radical-idealI*:
assumes *finite X* **and** $F \subseteq P[X]$ **and** $f \in P[X]$ **and** $x \notin X$
and $\mathcal{V}$ (*insert* (*1* − *punit.monom-mult 1* (*Poly-Mapping.single x 1*) *f*) *F*) = {}
shows (f::(′$x \Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*alg-closed-field*) $\in \sqrt{}$*ideal F*
proof −
define *Y* **where** *Y* = *insert x X*
from *assms*(*1*) **have** *fin-Y*: *finite Y* **by** (*simp add*: *Y-def*)
have $X \subseteq Y$ **by** (*auto simp*: *Y-def*)
hence $P[X] \subseteq P[Y]$ **by** (*rule Polys-mono*)
with *assms*(*2*, *3*) **have** *F-sub*: $F \subseteq P[Y]$ **and** $f \in P[Y]$ **by** *auto*
{

We define the type ′*y* to be isomorphic to *Y*.

assume ∃(*Rep* :: ′*y* ⇒ ′*x*) *Abs. type-definition Rep Abs Y*
then obtain *rep* :: ′*y* ⇒ ′*x* **and** *abs* :: ′*x* ⇒ ′*y* **where** *t*: *type-definition rep abs Y*
by *blast*

then interpret *y*: *type-definition rep abs Y* **.**

from *well-ordering* **obtain** $le\text{-}y'::('y \times 'y)\ set$ **where** *fld*: $Field\ le\text{-}y' = UNIV$
and *wo*: $Well\text{-}order\ le\text{-}y'$ **by** *meson*
define *le-y* **where** $le\text{-}y = (\lambda a\ b::'y.\ (a,\ b) \in le\text{-}y')$

from $\langle f \in P[Y] \rangle$ **have** *0*: $map\text{-}indets\ rep\ (map\text{-}indets\ abs\ f) = f$ **unfolding** *map-indets-map-indets*
by (*intro map-indets-id*) (*auto intro*!: *y.Abs-inverse dest*: *PolysD*)
have *1*: $map\text{-}indets\ (rep \circ abs)$ ‘ $F = F$
proof
from *F-sub* **show** $map\text{-}indets\ (rep \circ abs)$ ‘ $F \subseteq F$
by (*smt* (*verit*) *PolysD*(*2*) *comp-apply image-subset-iff map-indets-id subsetD y.Abs-inverse*)
next
from *F-sub* **show** $F \subseteq map\text{-}indets\ (rep \circ abs)$ ‘ F
by (*smt* (*verit*) *PolysD*(*2*) *comp-apply image-eqI map-indets-id subsetD subsetI y.Abs-inverse*)
qed
have *2*: *inj rep* **by** (*meson inj-onI y.Rep-inject*)
hence *3*: *inj* (*map-indets rep*) **by** (*rule map-indets-injI*)
from *fin-Y* **have** *4*: *finite* (*abs* ‘ *Y*) **by** (*rule finite-imageI*)
from *wo* **have** *le-y-refl*: *le-y x x* **for** *x*
by (*simp add*: *le-y-def well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def refl-on-def fld*)
have *le-y-total*: $le\text{-}y\ x\ y \lor le\text{-}y\ y\ x$ **for** *x y*
proof (*cases* $x = y$)
case *True*
thus *?thesis* **by** (*simp add*: *le-y-refl*)
next
case *False*
with *wo* **show** *?thesis*
by (*simp add*: *le-y-def well-order-on-def linear-order-on-def total-on-def Relation.total-on-def fld*)
qed

from *4 finite-imp-inj-to-nat-seg y.Abs-image* **have** *class.countable TYPE*(*'y*)
by *unfold-locales fastforce*
moreover have *class.linorder le-y* (*strict le-y*)
apply *standard*
subgoal by (*fact refl*)
subgoal by (*fact le-y-refl*)
subgoal using *wo*
by (*auto simp*: *le-y-def well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def fld dest*: *transD*)
subgoal using *wo*
by (*simp add*: *le-y-def well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def antisym-def fld*)
subgoal by (*fact le-y-total*)

done
moreover from *assms(1)* **have** *finite (abs ‘ X)* **by** *(rule finite-imageI)*
moreover have *map-indets abs ‘ F ⊆ P[abs ‘ X]*
proof *(rule subset-trans)*
from *assms(2)* **show** *map-indets abs ‘ F ⊆ map-indets abs ‘ P[X]* **by** *(rule image-mono)*
qed *(simp only: image-map-indets-Polys)*
moreover have *map-indets abs f ∈ P[abs ‘ X]*
proof
from *assms(3)* **show** *map-indets abs f ∈ map-indets abs ‘ P[X]* **by** *(rule imageI)*
qed *(simp only: image-map-indets-Polys)*
moreover from *assms(4) y.Abs-inject* **have** *abs x ∉ abs ‘ X* **unfolding** *Y-def* **by** *blast*
moreover have *𝒱 (insert (1 − punit.monom-mult 1 (Poly-Mapping.single (abs x) (Suc 0))*
(map-indets abs f)) (map-indets abs ‘ F)) = {}
proof *(intro set-eqI iffI)*
fix *a*
assume *a ∈ 𝒱 (insert (1 − punit.monom-mult 1 (Poly-Mapping.single (abs x) (Suc 0))*
(map-indets abs f)) (map-indets abs ‘ F))
also have *... = (λb. b ∘ abs) −‘ 𝒱 (insert (1 − punit.monom-mult 1 (Poly-Mapping.single x 1) f) F)*
by *(simp add: map-indets-minus map-indets-times map-indets-monomial*
flip: variety-of-map-indets times-monomial-left)
finally show *a ∈ {}* **by** *(simp only: assms(5) vimage-empty)*
qed *simp*
ultimately have *map-indets abs f ∈ √ideal (map-indets abs ‘ F)*
by *(rule radical-idealI-internalized[***where** *'x='y, untransferred, simplified])*
hence *map-indets rep (map-indets abs f) ∈ map-indets rep ‘ √ideal (map-indets abs ‘ F)*
by *(rule imageI)*
also from *2* **have** *... = √(ideal F ∩ P[Y]) ∩ P[Y]*
by *(simp add: image-map-indets-ideal image-map-indets-radical image-image map-indets-map-indets*
1 y.Rep-range)
also have *... ⊆ √ideal F* **using** *radical-mono* **by** *blast*
finally have *?thesis* **by** *(simp only: 0)*
}
note *rl = this[cancel-type-definition]*
have *Y ≠ {}* **by** *(simp add: Y-def)*
thus *?thesis* **by** *(rule rl)*
qed

corollary *radical-idealI-extend-indets*:
assumes *finite X* **and** *F ⊆ P[X]*
and *𝒱 (insert (1 − punit.monom-mult 1 (Poly-Mapping.single None 1) (extend-indets f))*

(*extend-indets* ‘ *F*)) = {}
shows (*f*::- ⇒$_0$ -::*alg-closed-field*) ∈ √*ideal F*
proof −
define *Y* **where** *Y* = *X* ∪ *indets f*
from *assms*(*1*) **have** *fin-Y*: *finite Y* **by** (*simp add*: *Y-def finite-indets*)
have *P*[*X*] ⊆ *P*[*Y*] **by** (*rule Polys-mono*) (*simp add*: *Y-def*)
with *assms*(*2*) **have** *F-sub*: *F* ⊆ *P*[*Y*] **by** (*rule subset-trans*)
have *f-in*: *f* ∈ *P*[*Y*] **by** (*simp add*: *Y-def Polys-alt*)

let *?F* = *extend-indets* ‘ *F*
let *?f* = *extend-indets f*
let *?X* = *Some* ‘ *Y*
from *fin-Y* **have** *finite ?X* **by** (*rule finite-imageI*)
moreover from *F-sub* **have** *?F* ⊆ *P*[*?X*]
by (*auto simp*: *indets-extend-indets intro*!: *PolysI-alt imageI dest*!: *PolysD*(*2*) *subsetD*[*of F*])
moreover from *f-in* **have** *?f* ∈ *P*[*?X*]
by (*auto simp*: *indets-extend-indets intro*!: *PolysI-alt imageI dest*!: *PolysD*(*2*))
moreover have *None* ∉ *?X* **by** *simp*
ultimately have *?f* ∈ √*ideal ?F* **using** *assms*(*3*) **by** (*rule radical-idealI*)
also have *?f* ∈ √*ideal ?F* ⟷ *f* ∈ √*ideal F*
proof
assume *f* ∈ √*ideal F*
then obtain *m* **where** *f* ⌢ *m* ∈ *ideal F* **by** (*rule radicalE*)
hence *extend-indets* (*f* ⌢ *m*) ∈ *extend-indets* ‘ *ideal F* **by** (*rule imageI*)
with *extend-indets-ideal-subset* **have** *?f* ⌢ *m* ∈ *ideal ?F* **unfolding** *extend-indets-power* **..**
thus *?f* ∈ √*ideal ?F* **by** (*rule radicalI*)
next
assume *?f* ∈ √*ideal ?F*
then obtain *m* **where** *?f* ⌢ *m* ∈ *ideal ?F* **by** (*rule radicalE*)
moreover have *?f* ⌢ *m* ∈ *P*[− {*None*}]
by (*rule Polys-closed-power*) (*auto intro*!: *PolysI-alt simp*: *indets-extend-indets*)
ultimately have *extend-indets* (*f* ⌢ *m*) ∈ *extend-indets* ‘ *ideal F*
by (*simp add*: *extend-indets-ideal extend-indets-power*)
hence *f* ⌢ *m* ∈ *ideal F* **by** (*simp only*: *inj-image-mem-iff*[*OF inj-extend-indets*])
thus *f* ∈ √*ideal F* **by** (*rule radicalI*)
qed
finally show *?thesis* **.**
qed

theorem *Nullstellensatz*:
assumes *finite X* **and** *F* ⊆ *P*[*X*]
and (*f*::- ⇒$_0$ -::*alg-closed-field*) ∈ $\mathcal{I}$ ($\mathcal{V}$ *F*)
shows *f* ∈ √*ideal F*
using *assms*(*1*, *2*)
proof (*rule radical-idealI-extend-indets*)
let *?f* = *punit.monom-mult 1* (*monomial 1 None*) (*extend-indets f*)
show $\mathcal{V}$ (*insert* (*1* − *?f*) (*extend-indets* ‘ *F*)) = {}

```
  proof (intro subset-antisym subsetI)
    fix a
    assume a ∈ 𝒱 (insert (1 − ?f) (extend-indets ‘ F))
    moreover have 1 − ?f ∈ insert (1 − ?f) (extend-indets ‘ F) by simp
    ultimately have poly-eval a (1 − ?f) = 0 by (rule variety-ofD)
    hence poly-eval a (extend-indets f) ≠ 0
     by (auto simp: poly-eval-minus poly-eval-times simp flip: times-monomial-left)
    hence poly-eval (a ∘ Some) f ≠ 0 by (simp add: poly-eval-extend-indets)
    have a ∘ Some ∈ 𝒱 F
    proof (rule variety-ofI)
      fix f′
      assume f′ ∈ F
      hence extend-indets f′ ∈ insert (1 − ?f) (extend-indets ‘ F) by simp
      with ‹a ∈ -› have poly-eval a (extend-indets f′) = 0 by (rule variety-ofD)
      thus poly-eval (a ∘ Some) f′ = 0 by (simp only: poly-eval-extend-indets)
    qed
    with assms(3) have poly-eval (a ∘ Some) f = 0 by (rule ideal-ofD)
    with ‹poly-eval (a ∘ Some) f ≠ 0› show a ∈ {} ..
  qed simp
qed

theorem strong-Nullstellensatz:
  assumes finite X and F ⊆ P[X]
  shows ℐ (𝒱 F) = √ideal (F::(- ⇒₀ -::alg-closed-field) set)
proof (intro subset-antisym subsetI)
  fix f
  assume f ∈ ℐ (𝒱 F)
  with assms show f ∈ √ideal F by (rule Nullstellensatz)
qed (metis ideal-ofI variety-ofD variety-of-radical-ideal)

theorem weak-Nullstellensatz:
  assumes finite X and F ⊆ P[X] and 𝒱 F = ({}::(- ⇒ -::alg-closed-field) set)
  shows ideal F = UNIV
proof −
  from assms(1, 2) have ℐ (𝒱 F) = √ideal F by (rule strong-Nullstellensatz)
  thus ?thesis by (simp add: assms(3) flip: radical-ideal-eq-UNIV-iff)
qed

lemma radical-ideal-iff:
  assumes finite X and F ⊆ P[X] and f ∈ P[X] and x ∉ X
  shows (f::- ⇒₀ -::alg-closed-field) ∈ √ideal F ⟷
           1 ∈ ideal (insert (1 − punit.monom-mult 1 (Poly-Mapping.single x 1)
f) F)
proof −
  let ?f = punit.monom-mult 1 (Poly-Mapping.single x 1) f
  show ?thesis
  proof
    assume f ∈ √ideal F
    then obtain m where f ^ m ∈ ideal F by (rule radicalE)
```

from *assms(1)* **have** *finite* (*insert x X*) **by** *simp*
moreover have *insert* (*1* − *?f*) *F* ⊆ *P*[*insert x X*] **unfolding** *insert-subset*
proof (*intro conjI Polys-closed-minus one-in-Polys Polys-closed-monom-mult PPs-closed-single*)
have *P*[*X*] ⊆ *P*[*insert x X*] **by** (*rule Polys-mono*) *blast*
with *assms(2, 3)* **show** *f* ∈ *P*[*insert x X*] **and** *F* ⊆ *P*[*insert x X*] **by** *blast+*
qed *simp*
moreover have $\mathcal{V}$ (*insert* (*1* − *?f*) *F*) = {}
proof (*intro subset-antisym subsetI*)
fix *a*
assume *a* ∈ $\mathcal{V}$ (*insert* (*1* − *?f*) *F*)
moreover have *1* − *?f* ∈ *insert* (*1* − *?f*) *F* **by** *simp*
ultimately have *poly-eval a* (*1* − *?f*) = *0* **by** (*rule variety-ofD*)
hence *poly-eval a* (*f* ⌢ *m*) ≠ *0*
by (*auto simp*: *poly-eval-minus poly-eval-times poly-eval-power simp flip*: *times-monomial-left*)
from ‹*a* ∈ -› **have** *a* ∈ $\mathcal{V}$ (*ideal* (*insert* (*1* − *?f*) *F*)) **by** (*simp only*: *variety-of-ideal*)
moreover from ‹*f* ⌢ *m* ∈ *ideal F*› *ideal.span-mono* **have** *f* ⌢ *m* ∈ *ideal* (*insert* (*1* − *?f*) *F*)
by (*rule rev-subsetD*) *blast*
ultimately have *poly-eval a* (*f* ⌢ *m*) = *0* **by** (*rule variety-ofD*)
with ‹*poly-eval a* (*f* ⌢ *m*) ≠ *0*› **show** *a* ∈ {} **..**
qed *simp*
ultimately have *ideal* (*insert* (*1* − *?f*) *F*) = *UNIV* **by** (*rule weak-Nullstellensatz*)
thus *1* ∈ *ideal* (*insert* (*1* − *?f*) *F*) **by** *simp*
next
assume *1* ∈ *ideal* (*insert* (*1* − *?f*) *F*)
have $\mathcal{V}$ (*insert* (*1* − *?f*) *F*) = {}
proof (*intro subset-antisym subsetI*)
fix *a*
assume *a* ∈ $\mathcal{V}$ (*insert* (*1* − *?f*) *F*)
hence *a* ∈ $\mathcal{V}$ (*ideal* (*insert* (*1* − *?f*) *F*)) **by** (*simp only*: *variety-of-ideal*)
hence *poly-eval a 1* = *0* **using** ‹*1* ∈ -› **by** (*rule variety-ofD*)
thus *a* ∈ {} **by** *simp*
qed *simp*
with *assms* **show** *f* ∈ √*ideal F* **by** (*rule radical-idealI*)
qed
qed

5.2 Field-Theoretic Version of the Nullstellensatz

Due to the possibility of infinite indeterminate-types, we have to explicitly add the set of indeterminates under consideration to the definition of maximal ideals.

definition *generates-max-ideal* :: *'x set* ⇒ ((*'x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a::comm-ring-1*) *set* ⇒ *bool*
where *generates-max-ideal X F* ⟷ (*ideal F* ≠ *UNIV* ∧

($\forall F'. F' \subseteq P[X] \longrightarrow$ *ideal* $F \subset$ *ideal* $F' \longrightarrow$ *ideal* $F' = UNIV$))

lemma *generates-max-idealI*:
assumes *ideal* $F \neq UNIV$ **and** $\bigwedge F'. F' \subseteq P[X] \Longrightarrow$ *ideal* $F \subset$ *ideal* $F' \Longrightarrow$ *ideal* $F' = UNIV$
shows *generates-max-ideal X F*
using *assms* **by** (*simp add*: *generates-max-ideal-def*)

lemma *generates-max-idealI-alt*:
assumes *ideal* $F \neq UNIV$ **and** $\bigwedge p. p \in P[X] \Longrightarrow p \notin$ *ideal* $F \Longrightarrow 1 \in$ *ideal* (*insert p F*)
shows *generates-max-ideal X F*
using *assms*(*1*)
proof (*rule generates-max-idealI*)
fix F'
assume $F' \subseteq P[X]$ **and** *sub*: *ideal* $F \subset$ *ideal* F'
from *this*(*2*) *ideal.span-subset-spanI* **have** $\neg F' \subseteq$ *ideal* F **by** *blast*
then obtain p **where** $p \in F'$ **and** $p \notin$ *ideal* F **by** *blast*
from *this*(*1*) ‹$F' \subseteq P[X]$› **have** $p \in P[X]$ **..**
hence $1 \in$ *ideal* (*insert p F*) **using** ‹$p \notin$ -› **by** (*rule assms*(*2*))
also have $\ldots \subseteq$ *ideal* $(F' \cup F)$ **by** (*rule ideal.span-mono*) (*simp add*: ‹$p \in F'$›)
also have $\ldots =$ *ideal* (*ideal* $F' \cup$ *ideal* F) **by** (*simp add*: *ideal.span-Un ideal.span-span*)
also from *sub* **have** *ideal* $F' \cup$ *ideal* $F =$ *ideal* F' **by** *blast*
finally show *ideal* $F' = UNIV$ **by** (*simp only*: *ideal-eq-UNIV-iff-contains-one ideal.span-span*)
qed

lemma *generates-max-idealD*:
assumes *generates-max-ideal X F*
shows *ideal* $F \neq UNIV$ **and** $F' \subseteq P[X] \Longrightarrow$ *ideal* $F \subset$ *ideal* $F' \Longrightarrow$ *ideal* $F' = UNIV$
using *assms* **by** (*simp-all add*: *generates-max-ideal-def*)

lemma *generates-max-ideal-cases*:
assumes *generates-max-ideal X F* **and** $F' \subseteq P[X]$ **and** *ideal* $F \subseteq$ *ideal* F'
obtains *ideal* $F =$ *ideal* F' | *ideal* $F' = UNIV$
using *assms* **by** (*auto simp*: *generates-max-ideal-def*)

lemma *max-ideal-UNIV-radical*:
assumes *generates-max-ideal UNIV F*
shows $\surd$*ideal* $F =$ *ideal* F
proof (*rule ccontr*)
assume $\surd$*ideal* $F \neq$ *ideal* F
with *radical-superset* **have** *ideal* $F \subset \surd$*ideal* F **by** *blast*
also have $\ldots =$ *ideal* ($\surd$*ideal* F) **by** *simp*
finally have *ideal* $F \subset$ *ideal* ($\surd$*ideal* F) **.**
with *assms* - **have** *ideal* ($\surd$*ideal* F) $= UNIV$ **by** (*rule generates-max-idealD*) *simp*

hence $\sqrt{}$*ideal F* = *UNIV* **by** *simp*
hence *1* ∈ $\sqrt{}$*ideal F* **by** *simp*
hence *1* ∈ *ideal F* **by** (*auto elim*: *radicalE*)
hence *ideal F* = *UNIV* **by** (*simp only*: *ideal-eq-UNIV-iff-contains-one*)
moreover from *assms* **have** *ideal F* ≠ *UNIV* **by** (*rule generates-max-idealD*)
ultimately show *False* **by** *simp*
qed

lemma *max-ideal-shape-aux*:
(λ*x*. *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*) ' *X* ⊆ *P*[*X*]
by (*auto intro*!: *Polys-closed-minus Polys-closed-monomial PPs-closed-single zero-in-PPs*)

lemma *max-ideal-shapeI*:
generates-max-ideal X ((λ*x*. *monomial* (*1*::'*a*::*field*) (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*) ' *X*)
(**is** *generates-max-ideal X ?F*)
proof (*rule generates-max-idealI-alt*)

show *ideal ?F* ≠ *UNIV*
proof
assume *ideal ?F* = *UNIV*
hence $\mathcal{V}$ (*ideal ?F*) = $\mathcal{V}$ *UNIV* **by** (*rule arg-cong*)
hence $\mathcal{V}$ *?F* = {} **by** *simp*
moreover have *a* ∈ $\mathcal{V}$ *?F* **by** (*rule variety-ofI*) (*auto simp*: *poly-eval-minus poly-eval-monomial*)
ultimately show *False* **by** *simp*
qed
next
fix *p*
assume *p* ∈ *P*[*X*] **and** *p* ∉ *ideal ?F*
have *p* ∈ *ideal* (*insert p ?F*) **by** (*rule ideal.span-base*) *simp*
let *?f* = λ*x*. *monomial* (*1*::'*a*) (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*
let *?g* = λ*x*. *monomial* (*1*::'*a*) (*Poly-Mapping.single x 1*) + *monomial* (*a x*) *0*
define *q* **where** *q* = *poly-subst ?g p*
have *p* = *poly-subst ?f q* **unfolding** *q-def poly-subst-poly-subst*
by (*rule sym*, *rule poly-subst-id*)
(*simp add*: *poly-subst-plus poly-subst-monomial subst-pp-single flip*: *times-monomial-left*)
also have ... = (∑ *t*∈*keys q*. *punit.monom-mult* (*lookup q t*) *0* (*subst-pp ?f t*)) **by** (*fact poly-subst-def*)
also have ... = *punit.monom-mult* (*lookup q 0*) *0* (*subst-pp ?f 0*) +
(∑ *t*∈*keys q* − {*0*}. *monomial* (*lookup q t*) *0* ∗ *subst-pp ?f t*)
(**is** - = - + *?r*)
by (*cases 0* ∈ *keys q*) (*simp-all add*: *sum.remove in-keys-iff flip*: *times-monomial-left*)
also have ... = *monomial* (*lookup q 0*) *0* + *?r* **by** (*simp flip*: *times-monomial-left*)
finally have *eq*: *p* − *?r* = *monomial* (*lookup q 0*) *0* **by** *simp*
have *?r* ∈ *ideal ?F*
proof (*intro ideal.span-sum ideal.span-scale*)
fix *t*

assume $t \in$ *keys q* $- \{0\}$
hence $t \in$ *keys q* **and** *keys t* $\neq \{\}$ **by** *simp-all*
from *this*(2) **obtain** x **where** $x \in$ *keys t* **by** *blast*
hence $x \in$ *indets q* **using** ‹$t \in$ *keys q*› **by** (*rule in-indetsI*)
then obtain y **where** $y \in$ *indets p* **and** $x \in$ *indets* (*?g y*) **unfolding** *q-def*
by (*rule in-indets-poly-substE*)
from *this*(2) *indets-plus-subset* **have** $x \in$ *indets* (*monomial* (*1*::'*a*) (*Poly-Mapping.single y 1*)) $\cup$
indets (*monomial* (*a y*) *0*) **..**
with ‹$y \in$ *indets p*› **have** $x \in$ *indets p* **by** (*simp add*: *indets-monomial*)
also from ‹$p \in P[X]$› **have** $\ldots \subseteq X$ **by** (*rule PolysD*)
finally have $x \in X$ **.**
from ‹$x \in$ *keys t*› **have** *lookup t x* $\neq 0$ **by** (*simp add*: *in-keys-iff*)
hence *eq*: *b* ^ *lookup t x* = *b* ^ *Suc* (*lookup t x* − *1*) **for** *b* **by** *simp*

have *subst-pp ?f t* = ($\prod$ *y*∈*keys t*. *?f y* ^ *lookup t y*) **by** (*fact subst-pp-def*)
also from ‹$x \in$ *keys t*› **have** $\ldots$ = (($\prod$ *y*∈*keys t* − $\{x\}$. *?f y* ^ *lookup t y*) ∗ *?f x* ^ (*lookup t x* − *1*)) ∗ *?f x*
by (*simp add*: *prod.remove mult.commute eq*)
also from ‹$x \in X$› **have** $\ldots \in$ *ideal ?F* **by** (*intro ideal.span-scale ideal.span-base imageI*)
finally show *subst-pp ?f t* $\in$ *ideal ?F* **.**
qed
also have $\ldots \subseteq$ *ideal* (*insert p ?F*) **by** (*rule ideal.span-mono*) *blast*
finally have *?r* $\in$ *ideal* (*insert p ?F*) **.**
with ‹$p \in$ *ideal* -› **have** *p* − *?r* $\in$ *ideal* (*insert p ?F*) **by** (*rule ideal.span-diff*)
hence *monomial* (*lookup q 0*) *0* $\in$ *ideal* (*insert p ?F*) **by** (*simp only*: *eq*)
hence *monomial* (*inverse* (*lookup q 0*)) *0* ∗ *monomial* (*lookup q 0*) *0* $\in$ *ideal* (*insert p ?F*)
by (*rule ideal.span-scale*)
hence *monomial* (*inverse* (*lookup q 0*) ∗ *lookup q 0*) *0* $\in$ *ideal* (*insert p ?F*)
by (*simp add*: *times-monomial-monomial*)
moreover have *lookup q 0* $\neq 0$
proof
assume *lookup q 0* = *0*
with *eq* ‹*?r* $\in$ *ideal ?F*› **have** $p \in$ *ideal ?F* **by** *simp*
with ‹$p \notin$ *ideal ?F*› **show** *False* **..**
qed
ultimately show *1* $\in$ *ideal* (*insert p ?F*) **by** *simp*
qed

We first prove the following lemma assuming that the type of indeterminates is finite, and then transfer the result to arbitrary types of indeterminates by using the 'types to sets' methodology. This approach facilitates the proof considerably.

lemma *max-ideal-shapeD-finite*:
assumes *generates-max-ideal UNIV* (*F*::(('*x*::*finite* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ '*a*::*alg-closed-field*) *set*)
obtains *a* **where** *ideal F* = *ideal* (*range* (λ*x*. *monomial 1* (*Poly-Mapping.single*

x 1) − *monomial* (*a x*) *0*))
proof −
  have *fin*: *finite* (*UNIV*::*'x set*) **by** *simp*
  have (⋂*a*∈𝒱 *F*. *ideal* (*range* (λ*x*. *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*))) = ℐ (𝒱 *F*)
    (**is** *?A* = -)
  proof (*intro set-eqI iffI ideal-ofI INT-I*)
    fix *p a*
    assume *p* ∈ *?A* **and** *a* ∈ 𝒱 *F*
    hence *p* ∈ *ideal* (*range* (λ*x*. *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*))
      (**is** - ∈ *ideal ?B*) **..**
    have *a* ∈ 𝒱 *?B*
    proof (*rule variety-ofI*)
      fix *f*
      assume *f* ∈ *?B*
      then obtain *x* **where** *f* = *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0* **..**
      thus *poly-eval a f* = *0* **by** (*simp add*: *poly-eval-minus poly-eval-monomial*)
    qed
    hence *a* ∈ 𝒱 (*ideal ?B*) **by** (*simp only*: *variety-of-ideal*)
    thus *poly-eval a p* = *0* **using** ‹*p* ∈ *ideal* -› **by** (*rule variety-ofD*)
  next
    fix *p a*
    assume *p* ∈ ℐ (𝒱 *F*) **and** *a* ∈ 𝒱 *F*
    hence *eq*: *poly-eval a p* = *0* **by** (*rule ideal-ofD*)
    have *p* ∈ √*ideal* (*range* (λ*x*. *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*)) (**is** - ∈ √*ideal ?B*)
      using *fin max-ideal-shape-aux*
    proof (*rule Nullstellensatz*)
      show *p* ∈ ℐ (𝒱 *?B*)
      proof (*rule ideal-ofI*)
        fix *a0*
        assume *a0* ∈ 𝒱 *?B*
        have *a0* = *a*
        proof
          fix *x*
          have *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0* ∈ *?B* **by** (*rule rangeI*)
          with ‹*a0* ∈ -› **have** *poly-eval a0* (*monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*) = *0*
            by (*rule variety-ofD*)
          thus *a0 x* = *a x* **by** (*simp add*: *poly-eval-minus poly-eval-monomial*)
        qed
        thus *poly-eval a0 p* = *0* **by** (*simp only*: *eq*)
      qed
    qed
    also have ... = *ideal* (*range* (λ*x*. *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*))

using *max-ideal-shapeI* **by** (*rule max-ideal-UNIV-radical*)
finally show *p* ∈ *ideal* (*range* (λ*x*. *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*)) **.**
qed
also from *fin* **have** ... = √*ideal F* **by** (*rule strong-Nullstellensatz*) *simp*
also from *assms* **have** ... = *ideal F* **by** (*rule max-ideal-UNIV-radical*)
finally have *eq*: *?A* = *ideal F* **.**
also from *assms* **have** ... ≠ *UNIV* **by** (*rule generates-max-idealD*)
finally obtain *a* **where** *a* ∈ 𝒱 *F*
and *ideal* (*range* (λ*x*. *monomial 1* (*Poly-Mapping.single x* (*1*::*nat*)) − *monomial* (*a x*) *0*)) ≠ *UNIV*
(**is** *?B* ≠ -) **by** *auto*
from ‹*a* ∈ 𝒱 *F*› **have** *ideal F* ⊆ *?B* **by** (*auto simp flip*: *eq*)
with *assms max-ideal-shape-aux* **show** *?thesis*
proof (*rule generates-max-ideal-cases*)
assume *ideal F* = *?B*
thus *?thesis* **..**
next
assume *?B* = *UNIV*
with ‹*?B* ≠ *UNIV*› **show** *?thesis* **..**
qed
qed

lemmas *max-ideal-shapeD-internalized* = *max-ideal-shapeD-finite*[*unoverload-type* ′*x*]

lemma *max-ideal-shapeD*:
assumes *finite X* **and** *F* ⊆ *P*[*X*]
and *generates-max-ideal X* (*F*::((′*x* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*::*alg-closed-field*) *set*)
obtains *a* **where** *ideal F* = *ideal* ((λ*x*. *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*a x*) *0*) ‘ *X*)
proof (*cases X* = {})
case *True*
from *assms*(*3*) **have** *ideal F* ≠ *UNIV* **by** (*rule generates-max-idealD*)
hence *1* ∉ *ideal F* **by** (*simp add*: *ideal-eq-UNIV-iff-contains-one*)
have *F* ⊆ {*0*}
proof
fix *f*
assume *f* ∈ *F*
with *assms*(*2*) **have** *f* ∈ *P*[*X*] **..**
then obtain *c* **where** *f*: *f* = *monomial c 0* **by** (*auto simp*: *True Polys-empty*)
with ‹*f* ∈ *F*› **have** *monomial c 0* ∈ *ideal F* **by** (*simp only*: *ideal.span-base*)
hence *monomial* (*inverse c*) *0* ∗ *monomial c 0* ∈ *ideal F* **by** (*rule ideal.span-scale*)
hence *monomial* (*inverse c* ∗ *c*) *0* ∈ *ideal F* **by** (*simp add*: *times-monomial-monomial*)
with ‹*1* ∉ *ideal F*› *left-inverse* **have** *c* = *0* **by** *fastforce*
thus *f* ∈ {*0*} **by** (*simp add*: *f*)
qed
hence *ideal F* = *ideal* ((λ*x*. *monomial 1* (*Poly-Mapping.single x 1*) − *monomial* (*undefined x*) *0*) ‘ *X*)

by (*simp add*: *True*)
thus *?thesis* **..**
next
case *False*
{

We define the type $'y$ to be isomorphic to X.

assume ∃(*Rep* :: $'y \Rightarrow 'x$) *Abs. type-definition Rep Abs X*
then obtain *rep* :: $'y \Rightarrow 'x$ **and** *abs* :: $'x \Rightarrow 'y$ **where** *t*: *type-definition rep abs X*
by *blast*
then interpret *y*: *type-definition rep abs X* **.**

have *1*: *map-indets* (*rep* ∘ *abs*) ‘ *A* = *A* **if** *A* ⊆ *P*[*X*] **for** *A*::(- $\Rightarrow_0$ $'a$) *set*
proof
from *that* **show** *map-indets* (*rep* ∘ *abs*) ‘ *A* ⊆ *A*
by (*smt* (*verit*) *PolysD*(*2*) *comp-apply image-subset-iff map-indets-id subsetD y.Abs-inverse*)
next
from *that* **show** *A* ⊆ *map-indets* (*rep* ∘ *abs*) ‘ *A*
by (*smt* (*verit*) *PolysD*(*2*) *comp-apply image-eqI map-indets-id subsetD subsetI y.Abs-inverse*)
qed
have *2*: *inj rep* **by** (*meson inj-onI y.Rep-inject*)
hence *3*: *inj* (*map-indets rep*) **by** (*rule map-indets-injI*)

have *class.finite TYPE*($'y$)
proof
from *assms*(*1*) **have** *finite* (*abs* ‘ *X*) **by** (*rule finite-imageI*)
thus *finite* (*UNIV*::$'y$ *set*) **by** (*simp only*: *y.Abs-image*)
qed
moreover have *generates-max-ideal UNIV* (*map-indets abs* ‘ *F*)
proof (*intro generates-max-idealI notI*)
assume *ideal* (*map-indets abs* ‘ *F*) = *UNIV*
hence *1* ∈ *ideal* (*map-indets abs* ‘ *F*) **by** *simp*
hence *map-indets rep 1* ∈ *map-indets rep* ‘ *ideal* (*map-indets abs* ‘ *F*) **by** (*rule imageI*)
also from *map-indets-plus map-indets-times* **have** ... ⊆ *ideal* (*map-indets rep* ‘ *map-indets abs* ‘ *F*)
by (*rule image-ideal-subset*)
also from *assms*(*2*) **have** *map-indets rep* ‘ *map-indets abs* ‘ *F* = *F*
by (*simp only*: *image-image map-indets-map-indets 1*)
finally have *1* ∈ *ideal F* **by** *simp*
moreover from *assms*(*3*) **have** *ideal F* ≠ *UNIV* **by** (*rule generates-max-idealD*)
ultimately show *False* **by** (*simp add*: *ideal-eq-UNIV-iff-contains-one*)
next
fix F'
assume *ideal* (*map-indets abs* ‘ *F*) ⊂ *ideal* F'
with *inj-on-subset* **have** *map-indets rep* ‘ *ideal* (*map-indets abs* ‘ *F*) ⊂

map-indets rep ‘ ideal F′
by (*rule image-strict-mono*) (*fact 3*, *fact subset-UNIV*)
hence *sub*: *ideal F* ∩ *P*[*X*] ⊂ *ideal* (*map-indets rep ‘ F′*) ∩ *P*[*X*] **using** *2 assms*(*2*)
by (*simp add*: *image-map-indets-ideal image-image map-indets-map-indets 1 y.Rep-range*)
have *ideal F* ⊂ *ideal* (*map-indets rep ‘ F′*)
proof (*intro psubsetI notI ideal.span-subset-spanI subsetI*)
fix *p*
assume *p* ∈ *F*
with *assms*(*2*) *ideal.span-base sub* **show** *p* ∈ *ideal* (*map-indets rep ‘ F′*) **by** *blast*
next
assume *ideal F* = *ideal* (*map-indets rep ‘ F′*)
with *sub* **show** *False* **by** *simp*
qed
with *assms*(*3*) - **have** *ideal* (*map-indets rep ‘ F′*) = *UNIV*
proof (*rule generates-max-idealD*)
from *subset-UNIV* **have** *map-indets rep ‘ F′* ⊆ *range* (*map-indets rep*) **by** (*rule image-mono*)
also have . . . = *P*[*X*] **by** (*simp only*: *range-map-indets y.Rep-range*)
finally show *map-indets rep ‘ F′* ⊆ *P*[*X*] **.**
qed
hence *P*[*range rep*] = *ideal* (*map-indets rep ‘ F′*) ∩ *P*[*range rep*] **by** *simp*
also from *2* **have** . . . = *map-indets rep ‘ ideal F′* **by** (*simp only*: *image-map-indets-ideal*)
finally have *map-indets rep ‘ ideal F′* = *range* (*map-indets rep*)
by (*simp only*: *range-map-indets*)
with *3* **show** *ideal F′* = *UNIV* **by** (*metis inj-image-eq-iff*)
qed
ultimately obtain *a*
where ∗: *ideal* (*map-indets abs ‘ F*) =
ideal (*range* (*λx. monomial 1* (*Poly-Mapping.single x* (*Suc 0*)) − *monomial* (*a x*) *0*))
(**is** - = *?A*)
by (*rule max-ideal-shapeD-internalized*[**where** *′x*=*′y*, *untransferred*, *simplified*])
hence *map-indets rep ‘ ideal* (*map-indets abs ‘ F*) = *map-indets rep ‘ ?A* **by** *simp*
with *2 assms*(*2*) **have** *ideal F* ∩ *P*[*X*] =
ideal (*range* (*λx. monomial 1* (*Poly-Mapping.single* (*rep x*) *1*) − *monomial* (*a x*) *0*)) ∩ *P*[*X*]
(**is** - = *ideal ?B* ∩ -)
by (*simp add*: *image-map-indets-ideal y.Rep-range image-image map-indets-map-indets map-indets-minus map-indets-monomial 1*)
also have *?B* = (*λx. monomial 1* (*Poly-Mapping.single x 1*) − *monomial* ((*a* ∘ *abs*) *x*) *0*) *‘ X*
(**is** - = *?C*)
proof

show *?B* ⊆ *?C* **by** (*smt* (*verit*) *comp-apply image-iff image-subset-iff y.Abs-image y.Abs-inverse*)
next
from *y.Rep-inverse y.Rep-range* **show** *?C* ⊆ *?B* **by** *auto*
qed
finally have *eq*: *ideal F* ∩ *P*[*X*] = *ideal ?C* ∩ *P*[*X*] **.**
have *ideal F* = *ideal ?C*
proof (*intro subset-antisym ideal.span-subset-spanI subsetI*)
fix *p*
assume *p* ∈ *F*
with *assms*(*2*) *ideal.span-base* **have** *p* ∈ *ideal F* ∩ *P*[*X*] **by** *blast*
thus *p* ∈ *ideal ?C* **by** (*simp add*: *eq*)
next
fix *p*
assume *p* ∈ *?C*
then obtain *x* **where** *x* ∈ *X* **and** *p* = *monomial 1* (*monomial 1 x*) − *monomial* ((*a* ∘ *abs*) *x*) *0* **..**
note *this*(*2*)
also from ‹*x* ∈ *X*› **have** . . . ∈ *P*[*X*]
by (*intro Polys-closed-minus Polys-closed-monomial PPs-closed-single zero-in-PPs*)
finally have *p* ∈ *P*[*X*] **.**
with ‹*p* ∈ *?C*› **have** *p* ∈ *ideal ?C* ∩ *P*[*X*] **by** (*simp add*: *ideal.span-base*)
also have . . . = *ideal F* ∩ *P*[*X*] **by** (*simp only*: *eq*)
finally show *p* ∈ *ideal F* **by** *simp*
qed
hence *?thesis* **..**
}
note *rl* = *this*[*cancel-type-definition*]
from *False* **show** *?thesis* **by** (*rule rl*)
qed

theorem *Nullstellensatz-field*:
assumes *finite X* **and** *F* ⊆ *P*[*X*] **and** *generates-max-ideal X* (*F*::(- ⇒$_0$ -::*alg-closed-field*) *set*)
and *x* ∈ *X*
shows {*0*} ⊂ *ideal F* ∩ *P*[{*x*}]
unfolding *subset-not-subset-eq*
proof (*intro conjI notI*)
show {*0*} ⊆ *ideal F* ∩ *P*[{*x*}] **by** (*auto intro*: *ideal.span-zero zero-in-Polys*)
next
from *assms*(*1*, *2*, *3*) **obtain** *a*
where *eq*: *ideal F* = *ideal* ((λ*x*. *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*) ‘ *X*)
by (*rule max-ideal-shapeD*)
let *?p* = λ*x*. *monomial 1* (*monomial 1 x*) − *monomial* (*a x*) *0*
from *assms*(*4*) **have** *?p x* ∈ *?p* ‘ *X* **by** (*rule imageI*)
also have . . . ⊆ *ideal F* **unfolding** *eq* **by** (*rule ideal.span-superset*)
finally have *?p x* ∈ *ideal F* **.**
moreover have *?p x* ∈ *P*[{*x*}]

by (*auto intro*!: *Polys-closed-minus Polys-closed-monomial PPs-closed-single zero-in-PPs*)
ultimately have *?p x* ∈ *ideal F* ∩ *P*[{*x*}] **..**
also assume . . . ⊆ {*0*}
finally show *False*
by (*metis diff-eq-diff-eq diff-self monomial-0D monomial-inj one-neq-zero singletonD*)
qed

end

References

[1] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Undergraduate Texts in Mathematics. Springer, 2007.