

Hilbert's Nullstellensatz

Alexander Maletzky*

May 26, 2024

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

1	Algebraically Closed Fields	2
2	Properties of the Lexicographic Order on Power-Products	6
3	Polynomial Mappings and Univariate Polynomials	9
3.1	Morphisms <i>pm-of-poly</i> and <i>poly-of-pm</i>	9
3.2	Evaluating Polynomials	16
3.3	Morphisms <i>flat-pm-of-poly</i> and <i>poly-of-focus</i>	17
4	Hilbert's Nullstellensatz	20
4.1	Preliminaries	20
4.2	Ideals and Varieties	23
4.3	Radical Ideals	27
4.4	Geometric Version of the Nullstellensatz	31
5	Field-Theoretic Version of Hilbert's Nullstellensatz	52
5.1	Getting Rid of Sort Constraints in Geometric Version	52
5.2	Field-Theoretic Version of the Nullstellensatz	57

*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.prem) 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.prem) (simp add: ⟨i ∈ I⟩)
    qed
  qed
  ultimately show ?thesis ..
qed

lemma degree-prod-eq:
  assumes finite I and  $\bigwedge i. i \in I \implies f i \neq 0$ 
  shows Polynomial.degree (prod f I :: ::semiring-no-zero-divisors poly) =  $(\sum i \in I. \text{Polynomial.degree } (f i))$ 
  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.prem)

```

```

    from that show  $i \in \text{insert } j \ J$  by simp
  qed
  hence eq:  $\text{Polynomial.degree } (\text{prod } f \ J) = (\sum i \in J. \text{Polynomial.degree } (f \ i))$  by
  (rule insert.hyps)
  from insert.hyps(1, 2) have  $\text{Polynomial.degree } (\text{prod } f \ (\text{insert } j \ J)) = \text{Polynomial.degree } (f \ j * \text{prod } f \ J)$ 
  by simp
  also have  $\dots = \text{Polynomial.degree } (f \ j) + \text{Polynomial.degree } (\text{prod } f \ J)$ 
  proof (rule degree-mult-eq)
    show  $f \ j \neq 0$  by (rule insert.prem) simp
  next
    show  $\text{prod } f \ J \neq 0$ 
  proof
    assume  $\text{prod } f \ J = 0$ 
    then obtain  $i$  where  $i \in J$  and  $f \ i = 0$  by (rule prod-eq-zeroE)
    from this(1) have  $f \ i \neq 0$  by (rule 1)
    thus False using  $\langle f \ i = 0 \rangle$  ..
  qed
  qed
  also from insert.hyps(1, 2) have  $\dots = (\sum i \in \text{insert } j \ J. \text{Polynomial.degree } (f \ i))$  by (simp add: eq)
  finally show ?case .
  qed

```

```

class alg-closed-field =
  assumes alg-closed-field-axiom:  $\bigwedge p :: 'a :: \text{field poly}. \ 0 < \text{Polynomial.degree } p \implies \exists z. \text{poly } p \ z = 0$ 
begin

```

```

lemma rootE:
  assumes  $0 < \text{Polynomial.degree } p$ 
  obtains  $z$  where  $\text{poly } p \ z = (0 :: 'a)$ 
proof -
  from assms have  $\exists z. \text{poly } p \ z = 0$  by (rule alg-closed-field-axiom)
  then obtain  $z$  where  $\text{poly } p \ z = 0$  ..
  thus ?thesis ..
qed

```

```

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

```

hence $\text{Polynomial.degree } [-1:] < \text{Polynomial.degree } (\prod a \in \text{UNIV}. [- a, 1::'a:])$
 by *simp*
 hence $\text{Polynomial.degree } p = \text{Polynomial.degree } (\prod a \in \text{UNIV}. [- a, 1::'a:])$ **un-**
folding *p-def*
 by (*rule degree-add-eq-left*)
 also have $\dots > 0$ by *fact*
 finally have $0 < \text{Polynomial.degree } p$.
 then obtain z where $\text{poly } p z = 0$ by (*rule rootE*)
 hence $(\prod a \in \text{UNIV}. (z - a)) = 1$ 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 $p :: 'a \text{ poly}$
 obtains $c A m$ where *finite* A and $p = \text{Polynomial.smult } c (\prod a \in A. [- a, 1:]$
 $\wedge m a)$
 and $\bigwedge a. m a = 0 \longleftrightarrow a \notin A$ and $c = 0 \longleftrightarrow p = 0$ and $\bigwedge z. \text{poly } p z = 0$
 $\longleftrightarrow (c = 0 \vee z \in A)$
proof –
 obtain $c A m$ where *fin*: *finite* A and $p: p = \text{Polynomial.smult } c (\prod a \in A. [- a, 1:]$
 $\wedge m a)$
 and $*$: $\bigwedge x. m x = 0 \longleftrightarrow x \notin A$
proof (*induct* p *arbitrary*: *thesis* *rule*: *poly-root-induct*[**where** $P = \lambda-. \text{True}$])
 case 0
 show *?case*
proof (*rule* 0)
 show $0 = \text{Polynomial.smult } 0 (\prod a \in \{\}. [- a, 1:] \wedge (\lambda-. 0) a)$ by *simp*
qed *simp-all*
next
 case (*no-roots* p)
 have $\text{Polynomial.degree } p = 0$
proof (*rule* *ccontr*)
 assume $\text{Polynomial.degree } p \neq 0$
 hence $0 < \text{Polynomial.degree } p$ by *simp*
 then obtain z where $\text{poly } p z = 0$ by (*rule* *rootE*)
 moreover have $\text{poly } p z \neq 0$ 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 $p = \text{Polynomial.smult } c (\prod a \in \{\}. [- a, 1:] \wedge (\lambda-. 0) a)$ by (*simp* *add*:
 p)
qed *simp-all*
next
 case (*root* a p)
 obtain $A c m$ where 1 : *finite* A and $p: p = \text{Polynomial.smult } c (\prod a \in A. [- a, 1:]$
 $\wedge m a)$

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

```

a, 1:] ^ m' a) .
  next
  fix x
  show m' x = 0  $\longleftrightarrow$  x  $\notin$  insert a A by (simp add: m'-def 2)
  qed
qed
moreover have c = 0  $\longleftrightarrow$  p = 0
proof
  assume p = 0
  hence [:c:] = 0  $\vee$  ( $\prod$  a $\in$ A. [: - a, 1:] ^ m a) = 0 by (simp add: p)
  thus c = 0
  proof
    assume [:c:] = 0
    thus ?thesis by simp
  next
    assume ( $\prod$  a $\in$ 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  $\in$  A by (rule ccontr) (simp add: * that)
  hence poly p z = 0  $\longleftrightarrow$  (c = 0  $\vee$  z  $\in$  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  $\in$  keys (s::'x::linorder  $\Rightarrow_0$  'a::add-linorder-min)
  obtains y where y  $\in$  keys t and y  $\leq$  x

```

```

using assms(1) unfolding lex-pm-alt
proof (elim disjE exE conjE)
  assume  $s = t$ 
  show ?thesis
  proof
    from assms(2) show  $x \in \text{keys } t$  by (simp only: ‹s = t›)
  qed (fact order.refl)
next
  fix  $y$ 
  assume  $1: \text{lookup } s \ y < \text{lookup } t \ y$  and  $2: \forall y' < y. \text{lookup } s \ y' = \text{lookup } t \ y'$ 
  show ?thesis
  proof (cases y ≤ x)
    case True
      from zero-min 1 have  $0 < \text{lookup } t \ y$  by (rule le-less-trans)
      hence  $y \in \text{keys } t$  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  $\text{lookup } s \ x = \text{lookup } t \ x$  by simp
      with assms(2) have  $x \in \text{keys } t$  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  $s = t \vee (\exists x. \text{lookup } s \ x < \text{lookup } t \ x \wedge (\forall y < x. \text{lookup } s \ y = \text{lookup } t \ y))$ 
  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  $\forall z < y. \text{lookup } s \ z = \text{lookup } t \ z$ 
    hence eq: lookup s z = lookup t z if z < y for z using that by simp
    assume  $*$ :  $\text{lookup } s \ y < \text{lookup } t \ y$ 
    hence  $y \in \text{keys } s \cup \text{keys } t$  by (auto simp flip: lookup-not-eq-zero-eq-in-keys)
    with assms(2) have  $y \in \{..x\}$  ..
    hence  $y = x \vee y < x$  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  $\bigwedge x y. x \in \text{keys } t \implies y \in \text{keys } u \implies x < y$ 
shows lex-pm-strict (u + s) (t::-  $\Rightarrow_0$  'a::add-linorder-min)
proof -
  from assms(1) obtain x where 1: lookup s x < lookup t x and 2:  $\bigwedge y. y < x \implies \text{lookup } s y = \text{lookup } t y$ 
  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 $\text{lookup } (u + s) \ y = \text{lookup } t \ y$ **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*:

$\{t. (\text{poly.coeff } p (\text{lookup } t \ x) \ \text{when } t \in \cdot.\{x\}) \neq 0\} =$
 $\text{Poly-Mapping.single } x \ \{d. \text{poly.coeff } p \ d \neq 0\}$ (**is** *?M = -*)

proof (*intro subset-antisym subsetI*)

fix t

assume $t \in ?M$

hence $\bigwedge y. y \neq x \implies \text{Poly-Mapping.lookup } t \ y = 0$ **by** (*fastforce simp: PPs-def in-keys-iff*)

hence $t = \text{Poly-Mapping.single } x (\text{lookup } t \ x)$

using *poly-mapping-eqI* **by** (*metis (full-types) lookup-single-eq lookup-single-not-eq*)

then show $t \in (\text{Poly-Mapping.single } x) \ \{d. \text{poly.coeff } p \ d \neq 0\}$ **using** $\langle t \in ?M \rangle$ **by** *auto*

qed (*auto split: if-splits simp: PPs-def*)

lift-definition *pm-of-poly* $:: 'x \Rightarrow 'a \ \text{poly} \Rightarrow ('x \Rightarrow_0 \ \text{nat}) \Rightarrow_0 'a::\text{comm-monoid-add}$
is $\lambda x \ p \ t. (\text{poly.coeff } p (\text{lookup } t \ x)) \ \text{when } t \in \cdot.\{x\}$

proof –

fix $x::'x$ **and** $p::'a \ \text{poly}$

show $\text{finite } \{t. (\text{poly.coeff } p (\text{lookup } t \ x) \ \text{when } t \in \cdot.\{x\}) \neq 0\}$ **unfolding**
pm-of-poly-aux

using *finite-surj[OF MOST-coeff-eq-0[unfolded eventually-cofinite]]* **by** *blast*

qed

definition *poly-of-pm* $:: 'x \Rightarrow (('x \Rightarrow_0 \ \text{nat}) \Rightarrow_0 'a) \Rightarrow 'a::\text{comm-monoid-add} \ \text{poly}$
where *poly-of-pm* $x \ p = \text{Abs-poly } (\lambda d. \text{lookup } p (\text{Poly-Mapping.single } x \ d))$

lemma *lookup-pm-of-poly-single* [*simp*]:

$\text{lookup } (\text{pm-of-poly } x \ p) (\text{Poly-Mapping.single } x \ d) = \text{poly.coeff } p \ d$

by (*simp add: pm-of-poly.rep-eq PPs-closed-single*)

lemma *keys-pm-of-poly*: $keys (pm\text{-of-poly } x\ p) = Poly\text{-Mapping.single } x \text{ ' } \{d. poly.coeff\ p\ d \neq 0\}$

proof –

have $keys (pm\text{-of-poly } x\ p) = \{t. (poly.coeff\ p\ (lookup\ t\ x)\ \text{when } t \in \cdot[\{x\}]) \neq 0\}$
by (*rule set-eqI*) (*simp add: pm-of-poly.rep-eq flip: lookup-not-eq-zero-eq-in-keys*)
also have $\dots = Poly\text{-Mapping.single } x \text{ ' } \{d. poly.coeff\ p\ d \neq 0\}$ **by** (*fact pm-of-poly-aux*)

finally show *?thesis* .

qed

lemma *coeff-poly-of-pm [simp]*: $poly.coeff (poly\text{-of-pm } x\ p)\ k = lookup\ p (Poly\text{-Mapping.single } x\ k)$

proof –

have $0 : Poly\text{-Mapping.single } x \text{ ' } \{d. lookup\ p (Poly\text{-Mapping.single } x\ d) \neq 0\} \subseteq \{d. lookup\ p\ d \neq 0\}$

by *auto*

have $\forall_{\infty} k. lookup\ p (Poly\text{-Mapping.single } x\ k) = 0$ **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 $p \in P[\{x\}]$

shows $pm\text{-of-poly } x (poly\text{-of-pm } x\ p) = p$

proof (*rule poly-mapping-eqI*)

fix t

from *assms* **have** $keys\ p \subseteq \cdot[\{x\}]$ **by** (*rule PolysD*)

show $lookup (pm\text{-of-poly } x (poly\text{-of-pm } x\ p))\ t = lookup\ p\ t$

proof (*simp add: pm-of-poly.rep-eq when-def, intro conjI impI*)

assume $t \in \cdot[\{x\}]$

hence $Poly\text{-Mapping.single } x (lookup\ t\ x) = t$

by (*simp add: PPsD keys-subset-singleton-imp-monomial*)

thus $lookup\ p (Poly\text{-Mapping.single } x (lookup\ t\ x)) = lookup\ p\ t$ **by** *simp*

next

assume $t \notin \cdot[\{x\}]$

with *assms PolysD* **have** $t \notin keys\ p$ **by** *blast*

thus $lookup\ p\ t = 0$ **by** (*simp add: in-keys-iff*)

qed

qed

lemma *poly-of-pm-of-poly [simp]*: $poly\text{-of-pm } x (pm\text{-of-poly } x\ p) = p$

by (*simp add: poly-of-pm-def coeff-inverse*)

lemma *pm-of-poly-in-Polys*: $pm\text{-of-poly } x\ p \in P[\{x\}]$

by (*auto simp: keys-pm-of-poly PPs-closed-single intro!: PolysI*)

lemma *pm-of-poly-zero [simp]*: $pm\text{-of-poly } x\ 0 = 0$

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

lemma *pm-of-poly-eq-zero-iff* [iff]: $pm\text{-of-poly } x \ p = 0 \longleftrightarrow p = 0$
 by (metis poly-of-pm-of-poly pm-of-poly-zero)

lemma *pm-of-poly-monom*: $pm\text{-of-poly } x \ (Polynomial.monom \ c \ d) = monomial \ c \ (Poly-Mapping.single \ x \ d)$
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*: $pm\text{-of-poly } x \ (p + q) = pm\text{-of-poly } x \ p + pm\text{-of-poly } x \ q$
 by (rule poly-mapping-eqI) (simp add: pm-of-poly.rep-eq lookup-add when-add-distrib)

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

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

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

lemma *pm-of-poly-pCons*:
 $pm\text{-of-poly } x \ (pCons \ c \ p) = monomial \ c \ 0 + punit.monom-mult \ (1:::monoid-mult) \ (Poly-Mapping.single \ x \ 1) \ (pm\text{-of-poly } x \ p)$
 (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: $t - ?x \in .[{x}] \longleftrightarrow t \in .[{x}]$
proof
 assume $t - ?x \in .[{x}]$

```

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) = ($\sum i \in I$. 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) = ($\prod i \in I$. 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 \longleftrightarrow keys p \cap .[x] = {}

proof

assume eq: poly-of-pm x p = 0

{

fix t

assume t \in .[x]

then obtain d where t = Poly-Mapping.single x d unfolding PPs-singleton

..

moreover assume t \in keys p

ultimately have 0 \neq 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 \cap .[x] = {} by blast

next

assume *: keys p \cap .[x] = {}

{

fix d

have Poly-Mapping.single x d \in .[x] (is ?x \in -) by (rule PPs-closed-single)

simp

with * have ?x \notin 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 ∈ .[{x}])
proof (*cases t ∈ .[{x}]*)
 case *True*
 moreover from this obtain d where $t = \text{Poly-Mapping.single } x \ d$
 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 $t \neq \text{monomial } d \ x$ **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::- =>_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 ∈ .[{x}]*)
 case *True*
 then obtain d where $t: t = \text{Poly-Mapping.single } x \ d$ **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 $\dots = \text{poly-of-pm } x \ (\text{monomial } c \ t * q)$ **unfolding** t
 proof (*induct d*)
 case 0
 have *Polynomial.smult c (poly-of-pm x q) = poly-of-pm x (c · 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) = ($\sum i \in I. poly-of-pm x (f i)$)*
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) = ($\prod i \in I. poly-of-pm x (f i)$)*

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

lemma *poly-of-pm-power* [*simp*]: $\text{poly-of-pm } x (p \wedge m) = \text{poly-of-pm } x p \wedge m$
by (*induct m*) (*simp-all add: poly-of-pm-times*)

3.2 Evaluating Polynomials

lemma *poly-eq-poly-eval*: $\text{poly} (\text{poly-of-pm } x p) a = \text{poly-eval} (\lambda y. a \text{ when } y = x)$
 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* \in $\cdot.\{x\}$)

case True

have $\text{poly-eval} (\lambda y. a \text{ when } y = x) (\text{monomial } c t) = c * (\prod_{y \in \text{keys } t} (a \text{ when } y = x) \wedge \text{lookup } t y)$

by (*simp only: poly-eval-monomial*)

also from True **have** $(\prod_{y \in \text{keys } t} (a \text{ when } y = x) \wedge \text{lookup } t y) = (\prod_{y \in \{x\}} (a \text{ when } y = x) \wedge \text{lookup } t y)$

by (*intro prod.mono-neutral-left ballI*) (*auto simp: in-keys-iff dest: PPsD*)

also have $\dots = a \wedge \text{lookup } t x$ **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 $\text{poly-eval} (\lambda y. a \text{ when } y = x) (\text{monomial } c t) = c * (\prod_{y \in \text{keys } t} (a \text{ when } y = x) \wedge \text{lookup } t y)$

by (*simp only: poly-eval-monomial*)

also from finite-keys **have** $(\prod_{y \in \text{keys } t} (a \text{ when } y = x) \wedge \text{lookup } t y) = 0$

proof (*rule prod-zero*)

from False **obtain** y **where** $y \in \text{keys } t$ **and** $y \neq x$ **by** (*auto simp: PPs-def*)

from this(1) **show** $\exists y \in \text{keys } t. (a \text{ when } y = x) \wedge \text{lookup } t y = 0$

proof

from $\langle y \in \text{keys } t \rangle$ **have** $0 < \text{lookup } t y$ **by** (*simp add: in-keys-iff*)

with $\langle y \neq x \rangle$ **show** $(a \text{ when } y = x) \wedge \text{lookup } t y = 0$ **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 $\text{poly} (\text{poly-of-pm } x \ p) \ a = \text{poly-eval} (\lambda-. \ a) \ p$
unfolding poly-eq-poly-eval **using** refl
proof $(\text{rule } \text{poly-eval-cong})$
fix y
assume $y \in \text{indets } p$
also from assms **have** $\dots \subseteq \{x\}$ **by** $(\text{rule } \text{PolysD})$
finally show $(a \ \text{when } y = x) = a$ **by** simp
qed

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

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

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

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

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

corollary $\text{indets-flat-pm-of-poly-subset}$:
 $\text{indets} (\text{flat-pm-of-poly } x \ p) \subseteq \text{insert } x \ (\bigcup (\text{indets } ' \ \text{range} (\text{poly.coeff } p)))$

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

lemma
shows $\text{flat-pm-of-poly-zero}$ $[\text{simp}]$: $\text{flat-pm-of-poly } x \ 0 = 0$
and $\text{flat-pm-of-poly-monom}$: $\text{flat-pm-of-poly } x \ (\text{Polynomial.monom } c \ d) =$

$punit.monom-mult\ 1\ (Poly-Mapping.single\ x\ d)\ c$
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*: $flat-pm-of-poly\ x\ (sum\ f\ I) = (\sum\ i \in I. flat-pm-of-poly\ x\ (f\ i))$
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*: $flat-pm-of-poly\ x\ (prod\ f\ I) =$
 $(\prod\ i \in I. flat-pm-of-poly\ x\ (f\ i :: -:comm-ring-1\ poly))$
and *flat-pm-of-poly-power*: $flat-pm-of-poly\ x\ (p \hat{\ } m) = flat-pm-of-poly\ x\ (p:::comm-ring-1\ poly) \hat{\ } m$
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 $p \in P[X]$
shows $range\ (poly.coeff\ (poly-of-focus\ x\ p)) \subseteq P[X - \{x\}]$
proof –
have $range\ (poly.coeff\ (poly-of-focus\ x\ p)) \subseteq range\ (lookup\ (focus\ \{x\}\ p))$
by (*auto simp: poly-of-focus-def*)
also from *assms* **have** $\dots \subseteq P[X - \{x\}]$ **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 $(- p) = - \text{poly-of-focus } x p$
and *poly-of-focus-plus*: *poly-of-focus* x $(p + q) = \text{poly-of-focus } x p + \text{poly-of-focus } x q$
and *poly-of-focus-minus*: *poly-of-focus* x $(p - q) = \text{poly-of-focus } x p - \text{poly-of-focus } x q$
and *poly-of-focus-one* [*simp*]: *poly-of-focus* x $1 = 1$
and *poly-of-focus-sum*: *poly-of-focus* x $(\text{sum } f I) = (\sum_{i \in I}. \text{poly-of-focus } x (f i))$
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 $p = 0 \longleftrightarrow p = 0$

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 $(p * q) = \text{poly-of-focus } x p * \text{poly-of-focus } x q$

and *poly-of-focus-prod*: *poly-of-focus* x $(\text{prod } f I) =$

$(\prod_{i \in I}. \text{poly-of-focus } x (f i :: - \Rightarrow_0 \text{::comm-semiring-1}))$

and *poly-of-focus-power*: *poly-of-focus* x $(p \wedge m) = \text{poly-of-focus } x (p \text{::} - \Rightarrow_0 \text{::comm-semiring-1}) \wedge m$

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 $\text{range } (\text{poly.coeff } p) \subseteq P[- \{x\}]$

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) $\subseteq P[- \{x\}]$

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** $\text{range } (\text{poly.coeff } p) \subseteq P[- \{x\}]$

```

shows  $p = 0$ 
proof –
  from assms(2) have  $p = \text{poly-of-focus } x \text{ (flat-pm-of-poly } x \text{ } p)$ 
    by (simp only: poly-of-focus-flat-pm-of-poly)
  also have  $\dots = 0$  by (simp add: assms(1))
  finally show ?thesis .
qed

lemma poly-poly-of-focus:  $\text{poly } (\text{poly-of-focus } x \text{ } p) \ a = \text{poly-eval } (\lambda\cdot. \ a) \ (\text{focus } \{x\} \ p)$ 
by (simp add: poly-of-focus-def poly-eq-poly-eval' focus-in-Polys)

corollary poly-poly-of-focus-monomial:
   $\text{poly } (\text{poly-of-focus } x \text{ } p) \ (\text{monomial } 1 \ (\text{Poly-Mapping.single } x \ 1)) = (p::- \Rightarrow_0 \ -::\text{comm-semiring-1})$ 
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 ( $A::'a::\text{linorder set}$ ) and  $P \ \{\}$ 
    and  $\bigwedge a \ A. \ \text{finite } A \implies A \subseteq \{..<a\} \implies P \ A \implies P \ (\text{insert } a \ A)$ 
  shows  $P \ A$ 
proof –
  define  $k$  where  $k = \text{card } A$ 
  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** $a = \text{Max } A$
from $\text{Suc.prem}(1)$ **have** $A \neq \{\}$ **by** *auto*
with $\text{Suc.prem}(2)$ **have** $a \in A$ **unfolding** $a\text{-def}$ **by** (*rule Max-in*)
with Suc.prem **have** $k = \text{card } (A - \{a\})$ **by** *simp*
moreover from $\text{Suc.prem}(2)$ **have** $\text{finite } (A - \{a\})$ **by** *simp*
ultimately have $P (A - \{a\})$ **by** (*rule Suc.hyps*)
with $\langle \text{finite } (A - \{a\}) \rangle$ **- have** $P (\text{insert } a (A - \{a\}))$
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 $\text{Suc.prem}(2)$ $\text{this}(1)$ **have** $b \leq a$ **unfolding** $a\text{-def}$ **by**
 (*rule Max-ge*)
 ultimately show $b \in \{..<a\}$ **by** *simp*
 qed
 qed
 with $\langle a \in A \rangle$ **show** $?case$ **by** (*simp add: insert-absorb*)
qed
qed

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

lemma *Fract-eq-zero-iff*: $\text{Fract } a \ b = 0 \longleftrightarrow a = 0 \vee b = 0$
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** $\text{poly } p (x + y) = \text{poly } p \ x + c * y$
proof (*induct p arbitrary: thesis*)
 case 0
 have $\text{poly } 0 (x + y) = \text{poly } 0 \ x + 0 * y$ **by** *simp*
 thus $?case$ **by** (*rule 0*)
next
 case ($p\text{Cons } a \ p$)
 obtain c **where** $\text{poly } p (x + y) = \text{poly } p \ x + c * y$ **by** (*rule pCons.hyps*)
 hence $\text{poly } (p\text{Cons } a \ p) (x + y) = a + (x + y) * (\text{poly } p \ x + c * y)$ **by** *simp*
 also have $\dots = \text{poly } (p\text{Cons } a \ p) \ x + (x * c + (\text{poly } p \ x + c * y)) * y$ **by** (*simp add: algebra-simps*)
 finally show $?case$ **by** (*rule pCons.prem*)
qed

lemma *poly-minus-rightE*:
obtains c **where** $\text{poly } p (x - y) = \text{poly } p \ x - c * (y:::\text{comm-ring})$
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 $\text{map-poly}\ f\ (p + q) = \text{map-poly}\ f\ p + \text{map-poly}\ f\ q$
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 $\text{map-poly}\ f\ (p - q) = \text{map-poly}\ f\ p - \text{map-poly}\ f\ q$
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 $\text{map-poly}\ f\ (\text{sum}\ g\ A) = (\sum a \in A. \text{map-poly}\ f\ (g\ a))$
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 $\text{map-poly}\ f\ (p * q) = \text{map-poly}\ f\ p * \text{map-poly}\ f\ q$
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 $\text{set}\ (\text{Polynomial.coeffs}\ p) \subseteq \text{range}\ (\lambda x. \text{Fract}\ x\ 1)$
obtains $q\ m$ **where** $\text{poly}\ p\ (\text{Fract}\ a\ b) = \text{Fract}\ q\ (b \wedge m)$
using *assms*
proof (induct *p* arbitrary: *thesis*)
 case 0
 have $\text{poly}\ 0\ (\text{Fract}\ a\ b) = \text{Fract}\ 0\ (b \wedge 1)$ **by** (simp add: *fract-collapse*)
 thus ?case **by** (rule 0)
next
 case (pCons *c p*)
 from *pCons.hyps*(1) **have** $\text{insert}\ c\ (\text{set}\ (\text{Polynomial.coeffs}\ p)) = \text{set}\ (\text{Polynomial.coeffs}\ (p\ \text{Cons}\ c\ p))$
 by auto
 with *pCons.prem*s(2) **have** $c \in \text{range}\ (\lambda x. \text{Fract}\ x\ 1)$ **and** $\text{set}\ (\text{Polynomial.coeffs}\ p) \subseteq \text{range}\ (\lambda x. \text{Fract}\ x\ 1)$
 by blast+
 from *this*(2) **obtain** $q0\ m0$ **where** $\text{poly}\ p\ (\text{Fract}\ a\ b) = \text{Fract}\ q0\ (b \wedge m0)$
 using *pCons.hyps*(2) **by** blast
 from $\langle c \in \cdot \rangle$ **obtain** $c0$ **where** $c = \text{Fract}\ c0\ 1 \dots$
 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.prem)
    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.prem)
    qed
  qed

```

```

lemma (in ordered-term) lt-sum-le-Max: lt (sum f A) ≤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 ... ≤t ord-term-lin.max (lt (f a)) (lt (sum f A)) by (rule lt-plus-le-max)
        also have ... ≤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 ⇒0 nat) ⇒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 \Rightarrow 'a::comm-semiring-1) set \Rightarrow (('x \Rightarrow_0 nat) \Rightarrow_0 'a) set
where *ideal-of* A = {f. $\forall a \in A. \text{poly-eval } a \ f = 0$ }

abbreviation $\mathcal{V} \equiv \text{variety-of}$

abbreviation $\mathcal{I} \equiv \text{ideal-of}$

lemma *variety-ofI*: $(\bigwedge f. f \in F \Longrightarrow \text{poly-eval } a \ f = 0) \Longrightarrow a \in \mathcal{V} \ F$
by (*simp add: variety-of-def*)

lemma *variety-ofI-alt*: $\text{poly-eval } a \ ' F \subseteq \{0\} \Longrightarrow a \in \mathcal{V} \ F$
by (*auto intro: variety-ofI*)

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

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

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

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

lemma *variety-of-ideal* [*simp*]: $\mathcal{V} \ (\text{ideal } F) = \mathcal{V} \ F$

proof

show $\mathcal{V} \ (\text{ideal } F) \subseteq \mathcal{V} \ F$ **by** (*intro variety-of-antimono ideal.span-superset*)

next

show $\mathcal{V} \ F \subseteq \mathcal{V} \ (\text{ideal } F)$

proof (*intro subsetI variety-ofI*)

fix a f

assume $a \in \mathcal{V} \ F$ **and** $f \in \text{ideal } F$

from *this*(2) **show** $\text{poly-eval } a \ f = 0$

proof (*induct f rule: ideal.span-induct-alt*)

case base

show ?case **by** *simp*

next

case (*step c f g*)

with $\langle a \in \mathcal{V} \ F \rangle$ **show** ?case **by** (*auto simp: poly-eval-plus poly-eval-times*

dest: variety-ofD)

qed

qed

qed

lemma *ideal-ofI*: $(\bigwedge a. a \in A \Longrightarrow \text{poly-eval } a \ f = 0) \Longrightarrow f \in \mathcal{I} \ A$
by (*simp add: ideal-of-def*)

lemma *ideal-ofD*: $f \in \mathcal{I} A \implies a \in A \implies \text{poly-eval } a f = 0$
by (*simp add: ideal-of-def*)

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

lemma *ideal-of-antimono*: $A \subseteq B \implies \mathcal{I} B \subseteq \mathcal{I} A$
by (*auto simp: ideal-of-def*)

lemma *ideal-ideal-of* [*simp*]: $\text{ideal } (\mathcal{I} A) = \mathcal{I} A$

unfolding *ideal.span-eq-iff*

proof (*rule ideal.subspaceI*)

show $0 \in \mathcal{I} A$ **by** (*rule ideal-ofI*) *simp*

next

fix $f g$

assume $f \in \mathcal{I} A$

hence f : *poly-eval* $a f = 0$ **if** $a \in A$ **for** a **using** *that* **by** (*rule ideal-ofD*)

assume $g \in \mathcal{I} A$

hence g : *poly-eval* $a g = 0$ **if** $a \in A$ **for** a **using** *that* **by** (*rule ideal-ofD*)

show $f + g \in \mathcal{I} A$ **by** (*rule ideal-ofI*) (*simp add: poly-eval-plus f g*)

next

fix $c f$

assume $f \in \mathcal{I} A$

hence f : *poly-eval* $a f = 0$ **if** $a \in A$ **for** a **using** *that* **by** (*rule ideal-ofD*)

show $c * f \in \mathcal{I} A$ **by** (*rule ideal-ofI*) (*simp add: poly-eval-times f*)

qed

lemma *ideal-of-UN*: $\mathcal{I} (\bigcup (A \text{ ' } J)) = (\bigcap_{j \in J} \mathcal{I} (A j))$

proof (*intro set-eqI iffI ideal-ofI INT-I*)

fix $p j a$

assume $p \in \mathcal{I} (\bigcup (A \text{ ' } J))$

assume $j \in J$ **and** $a \in A j$

hence $a \in \bigcup (A \text{ ' } J)$..

with $\langle p \in \cdot \rangle$ **show** *poly-eval* $a p = 0$ **by** (*rule ideal-ofD*)

next

fix $p a$

assume $a \in \bigcup (A \text{ ' } J)$

then obtain j **where** $j \in J$ **and** $a \in A j$..

assume $p \in (\bigcap_{j \in J} \mathcal{I} (A j))$

hence $p \in \mathcal{I} (A j)$ **using** $\langle j \in J \rangle$..

thus *poly-eval* $a p = 0$ **using** $\langle a \in A j \rangle$ **by** (*rule ideal-ofD*)

qed

corollary *ideal-of-Un*: $\mathcal{I} (A \cup B) = \mathcal{I} A \cap \mathcal{I} B$

using *ideal-of-UN*[*of id* $\{A, B\}$] **by** *simp*

lemma *variety-of-ideal-of-variety* [*simp*]: $\mathcal{V} (\mathcal{I} (\mathcal{V} F)) = \mathcal{V} F$ (**is** $- = ?V$)

proof

have $F \subseteq \mathcal{I} (\mathcal{V} F)$ **by** (*auto intro!: ideal-ofI dest: variety-ofD*)

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

lemma *ideal-of-inj-on: inj-on \mathcal{I} (range ($\mathcal{V}::('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{comm-semiring-1}$) set \Rightarrow -))*
proof (*rule inj-onI*)
fix $A B :: ('x \Rightarrow 'a)$ *set*
assume $A \in \text{range } \mathcal{V}$
then obtain F **where** $A: A = \mathcal{V} F$..
assume $B \in \text{range } \mathcal{V}$
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 $A \subseteq \mathcal{V} (\mathcal{I} A)$ **by** (*auto intro!: variety-ofI dest: ideal-ofD*)
thus $\mathcal{I} (\mathcal{V} \ ?I) \subseteq \ ?I$ **by** (*rule ideal-of-antimono*)
next
show $\ ?I \subseteq \mathcal{I} (\mathcal{V} \ ?I)$ **by** (*auto intro!: ideal-ofI dest: variety-ofD*)
qed

lemma *variety-of-inj-on: inj-on \mathcal{V} (range ($\mathcal{I}::('x \Rightarrow 'a::\text{comm-semiring-1})$ set \Rightarrow -))*
proof (*rule inj-onI*)
fix $F G :: (('x \Rightarrow_0 \text{nat}) \Rightarrow_0 'a)$ *set*
assume $F \in \text{range } \mathcal{I}$
then obtain A **where** $F: F = \mathcal{I} A$..
assume $G \in \text{range } \mathcal{I}$
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 ' $\mathcal{I} A = \mathcal{I} ((\lambda a. a \circ f) - ' (A::('x \Rightarrow 'a::\text{comm-semiring-1})$ set)) $\cap P[\text{range } f]$*
proof -
{
fix p **and** $a::'x \Rightarrow 'a$
assume $\forall a \in (\lambda a. a \circ f) - ' A. \text{poly-eval } (a \circ f) p = 0$
hence *eq: poly-eval (a \circ f) p = 0 if a \circ f \in A for a using that by simp*
have *the-inv f \circ f = id* **by** (*rule ext*) (*simp add: assms the-inv-f-f*)

hence $a: a = a \circ \text{the-inv } f \circ f$ **by** (*simp add: comp-assoc*)
moreover assume $a \in A$
ultimately have $(a \circ \text{the-inv } f) \circ f \in A$ **by** *simp*
hence $\text{poly-eval } ((a \circ \text{the-inv } f) \circ f) p = 0$ **by** (*rule eq*)
hence $\text{poly-eval } a p = 0$ **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*: $\mathcal{V} (\text{map-indets } f \text{ ' } F) = (\lambda a. a \circ f) \text{ ' } \mathcal{V} F$
by (*auto simp: variety-of-def poly-eval-map-indets*)

4.3 Radical Ideals

definition *radical* :: 'a::monoid-mult set \Rightarrow 'a set ($\sqrt{(-)}$ [999] 999)
where $\text{radical } F = \{f. \exists m. f \wedge m \in F\}$

lemma *radicalI*: $f \wedge m \in F \implies f \in \sqrt{F}$
by (*auto simp: radical-def*)

lemma *radicalE*:
assumes $f \in \sqrt{F}$
obtains m **where** $f \wedge m \in F$
using *assms* **by** (*auto simp: radical-def*)

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

lemma *radical-UNIV* [*simp*]: $\sqrt{UNIV} = UNIV$
by (*simp add: radical-def*)

lemma *radical-ideal-eq-UNIV-iff*: $\sqrt{\text{ideal } F} = UNIV \iff \text{ideal } F = UNIV$

proof

assume $\sqrt{\text{ideal } F} = UNIV$
hence $1 \in \sqrt{\text{ideal } F}$ **by** *simp*
then obtain m **where** $1 \wedge m \in \text{ideal } F$ **by** (*rule radicalE*)
thus $\text{ideal } F = UNIV$ **by** (*simp add: ideal-eq-UNIV-iff-contains-one*)
qed *simp*

lemma *zero-in-radical-ideal* [*simp*]: $0 \in \sqrt{\text{ideal } F}$

proof (*rule radicalI*)

show $0 \wedge 1 \in \text{ideal } F$ **by** (*simp add: ideal.span-zero*)

qed

lemma *radical-mono*: $F \subseteq G \implies \sqrt{F} \subseteq \sqrt{G}$
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*: $\sqrt{(\text{ideal } F \cap \text{ideal } G)} = \sqrt{\text{ideal } F} \cap \sqrt{\text{ideal } G}$

using *radical-Int-subset*

proof (*rule subset-antisym*)

show $\sqrt{\text{ideal } F} \cap \sqrt{\text{ideal } G} \subseteq \sqrt{(\text{ideal } F \cap \text{ideal } G)}$

proof

fix p

assume $p \in \sqrt{\text{ideal } F} \cap \sqrt{\text{ideal } G}$

hence $p \in \sqrt{\text{ideal } F}$ **and** $p \in \sqrt{\text{ideal } G}$ **by** *simp-all*

from *this*(1) **obtain** $m1$ **where** $p1$: $p^{m1} \in \text{ideal } F$ **by** (*rule radicalE*)

from $\langle p \in \sqrt{\text{ideal } G} \rangle$ **obtain** $m2$ **where** $p^{m2} \in \text{ideal } G$ **by** (*rule radicalE*)

hence $p^{m1} * p^{m2} \in \text{ideal } G$ **by** (*rule ideal.span-scale*)

moreover from $p1$ **have** $p^{m2} * p^{m1} \in \text{ideal } F$ **by** (*rule ideal.span-scale*)

ultimately have $p^{(m1 + m2)} \in \text{ideal } F \cap \text{ideal } G$ **by** (*simp add*: *power-add* *mult.commute*)

thus $p \in \sqrt{(\text{ideal } F \cap \text{ideal } G)}$ **by** (*rule radicalI*)

qed

qed

lemma *ideal-radical-ideal* [*simp*]: $\text{ideal } (\sqrt{\text{ideal } F}) = \sqrt{\text{ideal } F}$ (**is** $=$ $?R$)

unfolding *ideal.span-eq-iff*

proof (*rule ideal.subspaceI*)

have $0^1 \in \text{ideal } F$ **by** (*simp add*: *ideal.span-zero*)

thus $0 \in ?R$ **by** (*rule radicalI*)

next

fix a b

assume $a \in ?R$

then obtain m **where** $a^m \in \text{ideal } F$ **by** (*rule radicalE*)

have a : $a^k \in \text{ideal } F$ **if** $m \leq k$ **for** k

proof $-$

from $\langle a^m \in \text{ideal } F \rangle$ **have** $a^{(k - m + m)} \in \text{ideal } F$ **by** (*simp only*: *power-add* *ideal.span-scale*)

with that show *?thesis* **by** *simp*

qed

assume $b \in ?R$
then obtain n **where** $b \wedge n \in \text{ideal } F$ **by** (rule radicalE)
have $b \wedge k \in \text{ideal } F$ **if** $n \leq k$ **for** k
proof –
from $\langle b \wedge n \in \cdot \rangle$ **have** $b \wedge (k - n + n) \in \text{ideal } F$ **by** (simp only: power-add ideal.span-scale)
with that show ?thesis **by** simp
qed
have $(a + b) \wedge (m + n) \in \text{ideal } F$ **unfolding** binomial-ring
proof (rule ideal.span-sum)
fix k
show of-nat $(m + n \text{ choose } k) * a \wedge k * b \wedge (m + n - k) \in \text{ideal } F$
proof (cases $k \leq m$)
case True
hence $n \leq m + n - k$ **by** simp
hence $b \wedge (m + n - k) \in \text{ideal } F$ **by** (rule b)
thus ?thesis **by** (rule ideal.span-scale)
next
case False
hence $m \leq k$ **by** simp
hence $a \wedge k \in \text{ideal } F$ **by** (rule a)
hence of-nat $(m + n \text{ choose } k) * b \wedge (m + n - k) * a \wedge k \in \text{ideal } F$ **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** $a \wedge m \in \text{ideal } F$ **by** (rule radicalE)
hence $(c * a) \wedge m \in \text{ideal } F$ **by** (simp only: power-mult-distrib ideal.span-scale)
thus $c * a \in ?R$ **by** (rule radicalI)
qed

lemma radical-ideal-of [simp]: $\sqrt{\mathcal{I} \ A} = \mathcal{I} \ (A::(- \Rightarrow -::\text{semiring-1-no-zero-divisors}) \text{ set})$

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]: $\mathcal{V}(\sqrt{\text{ideal } F}) = \mathcal{V}(F::(- \Rightarrow_0 -::\text{semiring-1-no-zero-divisors}) \text{ set})$

proof

have $F \subseteq \text{ideal } F$ **by** (rule ideal.span-superset)

also have $\dots \subseteq \sqrt{\text{ideal } F}$ **by** (rule radical-superset)

finally show $\mathcal{V}(\sqrt{\text{ideal } F}) \subseteq \mathcal{V} \ F$ **by** (rule variety-of-antimono)

next

```

show  $\mathcal{V} F \subseteq \mathcal{V} (\sqrt{\text{ideal } F})$ 
proof (intro subsetI variety-ofI)
  fix a f
  assume  $a \in \mathcal{V} F$ 
  hence  $a \in \mathcal{V} (\text{ideal } F)$  by simp
  assume  $f \in \sqrt{\text{ideal } F}$ 
  then obtain m where  $f \wedge m \in \text{ideal } F$  by (rule radicalE)
  with  $\langle a \in \mathcal{V} (\text{ideal } F) \rangle$  have  $\text{poly-eval } a (f \wedge m) = 0$  by (rule variety-ofD)
  thus  $\text{poly-eval } a f = 0$  by (simp add: poly-eval-power)
qed
qed

lemma image-map-indets-radical:
  assumes inj f
  shows  $\text{map-indets } f \text{ ' } \sqrt{F} = \sqrt{(\text{map-indets } f \text{ ' } (F::(- \Rightarrow_0 'a::\text{comm-ring-1}) \text{ set})) \cap P[\text{range } f]}$ 
proof
  show  $\text{map-indets } f \text{ ' } \sqrt{F} \subseteq \sqrt{(\text{map-indets } f \text{ ' } F) \cap P[\text{range } f]}$ 
  by (auto simp: radical-def simp flip: map-indets-power range-map-indets intro!: imageI)
next
  show  $\sqrt{(\text{map-indets } f \text{ ' } F) \cap P[\text{range } f]} \subseteq \text{map-indets } f \text{ ' } \sqrt{F}$ 
proof
  fix p
  assume  $p \in \sqrt{(\text{map-indets } f \text{ ' } F) \cap P[\text{range } f]}$ 
  hence  $p \in \sqrt{(\text{map-indets } f \text{ ' } F)}$  and  $p \in \text{range } (\text{map-indets } f)$ 
  by (simp-all add: range-map-indets)
  from this(1) obtain m where  $p \wedge m \in \text{map-indets } f \text{ ' } F$  by (rule radicalE)
  then obtain q where  $q \in F$  and  $p \cdot m = \text{map-indets } f q$  ..
  from assms obtain g where  $g \circ f = \text{id}$  and  $\text{map-indets } g \circ \text{map-indets } f =$ 
  ( $\text{id}::(- \Rightarrow_0 'a)$ )
  by (rule map-indets-inverseE)
  hence  $\text{eq: map-indets } g (\text{map-indets } f p) = p$  for  $p::(- \Rightarrow_0 'a)$ 
  by (simp add: pointfree-idE)
  from p-m have  $\text{map-indets } g (p \wedge m) = \text{map-indets } g (\text{map-indets } f q)$  by (rule
  arg-cong)
  hence  $(\text{map-indets } g p) \wedge m = q$  by (simp add: eq)
  from  $\langle p \in \text{range } \rightarrow \rangle$  obtain p' where  $p = \text{map-indets } f p'$  ..
  hence  $p = \text{map-indets } f (\text{map-indets } g p)$  by (simp add: eq)
  moreover have  $\text{map-indets } g p \in \sqrt{F}$ 
  proof (rule radicalI)
    from  $\langle q \in F \rangle$  show  $\text{map-indets } g p \wedge m \in F$  by (simp add: p-m eq flip:
    map-indets-power)
  qed
  ultimately show  $p \in \text{map-indets } f \text{ ' } \sqrt{F}$  by (rule image-eqI)
qed
qed

```

4.4 Geometric Version of the Nullstellensatz

lemma *weak-Nullstellensatz-aux-1:*

assumes $\bigwedge i. i \in I \implies g i \in \text{ideal } B$

obtains c **where** $c \in \text{ideal } B$ **and** $(\prod_{i \in I}. (f i + g i) \wedge m i) = (\prod_{i \in I}. f i \wedge m i) + c$

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 $g i \in \text{ideal } B$ **if** $i \in I$ **for** i **by** (*rule insert.prem*) (*simp add: that*)

with *insert.hyps(3)* **obtain** c **where** $c: c \in \text{ideal } B$

and $1: (\prod_{i \in I}. (f i + g i) \wedge m i) = (\prod_{i \in I}. f i \wedge m i) + c$ **by** *blast*

define k **where** $k = m j$

obtain d **where** $2: (f j + g j) \wedge m j = f j \wedge m j + d * g j$ **unfolding**
k-def[symmetric]

proof (*induct k arbitrary: thesis*)

case 0

have $(f j + g j) \wedge 0 = f j \wedge 0 + 0 * g j$ **by** *simp*

thus *?case* **by** (*rule 0*)

next

case (*Suc k*)

obtain d **where** $(f j + g j) \wedge k = f j \wedge k + d * g j$ **by** (*rule Suc.hyps*)

hence $(f j + g j) \wedge \text{Suc } k = (f j \wedge k + d * g j) * (f j + g j)$ **by** *simp*

also have $\dots = f j \wedge \text{Suc } k + (f j \wedge k + d * (f j + g j)) * g j$ **by** (*simp add: algebra-simps*)

finally show *?case* **by** (*rule Suc.prem*)

qed

from c **have** $*$: $f j \wedge m j * c + (((\prod_{i \in I}. f i \wedge m i) + c) * d) * g j \in \text{ideal } B$ (**is**
?c ∈ -)

by (*intro ideal.span-add ideal.span-scale insert.prem insertI1*)

from *insert.hyps(1, 2)* **have** $(\prod_{i \in \text{insert } j I}. (f i + g i) \wedge m i) =$
 $(f j \wedge m j + d * g j) * ((\prod_{i \in I}. f i \wedge m i) + c)$

by (*simp add: 1 2*)

also from *insert.hyps(1, 2)* **have** $\dots = (\prod_{i \in \text{insert } j I}. f i \wedge m i) + ?c$ **by** (*simp add: algebra-simps*)

finally have $(\prod_{i \in \text{insert } j I}. (f i + g i) \wedge m i) = (\prod_{i \in \text{insert } j I}. f i \wedge m i) + ?c$.

with $*$ **show** *?case* **by** (*rule insert.prem*)

qed

lemma *weak-Nullstellensatz-aux-2:*

assumes *finite X* **and** $F \subseteq P[\text{insert } x X]$ **and** $X \subseteq \{..<x::'x::\{\text{countable, linorder}\}\}$

and $1 \notin \text{ideal } F$ **and** $\text{ideal } F \cap P[\{x\}] \subseteq \{0\}$

obtains $a::'a::\text{alg-closed-field}$ **where** $1 \notin \text{ideal } (\text{poly-eval } (\lambda-. \text{monomial } a 0))$ ‘

```

focus {x} ‘ F)
proof -
  let ?x = monomial 1 (Poly-Mapping.single x 1)
  from assms( $\beta$ ) have  $x \notin X$  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)  $\Rightarrow$  -
  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  $g \in P[\text{insert } x \ X]$  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) = ( $\lambda t.$  except t {x}) ‘ keys g
          unfolding keys-focus using refl
        proof (rule image-cong)
          fix t
            assume  $t \in \text{keys } g$ 
            also from that have  $\dots \subseteq .[\text{insert } x \ X]$  by (rule PolysD)
            finally have  $\text{keys } t \subseteq \text{insert } x \ X$  by (rule PPsD)
            hence except t (- X) = except t (insert x X  $\cap$  - X)
            by (metis (no-types, lifting) Int-commute except-keys-Int inf.orderE inf-left-commute)
            also from  $\langle x \notin X \rangle$  have insert x X  $\cap$  - 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  $i.lpp \ g \in \text{keys } g$  by (rule i.punit.lt-in-keys)
          thus except (i.lpp g) {x}  $\in \text{keys}$  (focus X g) unfolding keys-focus-g by (rule imageI)

          fix t
            assume  $t \in \text{keys}$  (focus X g)
            then obtain s where  $s \in \text{keys } g$  and  $t: t = \text{except } s \ \{x\}$  unfolding
keys-focus-g ..
            from this(1) have lex-pm s (i.lpp g) by (rule i.punit.lt-max-keys)
            moreover have  $\text{keys } s \cup \text{keys} \ (i.lpp \ g) \subseteq \{..x\}$ 
            proof (rule Un-least)
              from  $\langle g \in P[-] \rangle$  have  $\text{keys } g \subseteq .[\text{insert } x \ X]$  by (rule PolysD)
              with  $\langle s \in \text{keys } g \rangle$  have  $s \in .[\text{insert } x \ X]$  ..
              hence  $\text{keys } s \subseteq \text{insert } x \ X$  by (rule PPsD)
              thus  $\text{keys } s \subseteq \{..x\}$  using assms( $\beta$ ) by auto

            from  $\langle i.lpp \ g \in \text{keys } g \rangle$   $\langle \text{keys } g \subseteq - \rangle$  have  $i.lpp \ g \in .[\text{insert } x \ X]$  ..
            hence  $\text{keys} \ (i.lpp \ g) \subseteq \text{insert } x \ X$  by (rule PPsD)
            thus  $\text{keys} \ (i.lpp \ g) \subseteq \{..x\}$  using assms( $\beta$ ) by auto

```


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

define G **where** $G = i.\text{punit.reduced-GB } F$
from $\text{assms}(1)$ **have** $\text{finite (insert } x \ X)$ **by** simp
hence $\text{fin-G: finite } G$ **and** $\text{G-sub: } G \subseteq P[\text{insert } x \ X]$ **and** $\text{ideal-G: ideal } G = \text{ideal } F$
and $0 \notin G$ **and** $\text{G-isGB: i.punit.is-Groebner-basis } G$ **unfolding** G-def **using** $\text{assms}(2)$
by $(\text{rule } i.\text{finite-reduced-GB-Polys}, \text{rule } i.\text{reduced-GB-Polys}, \text{rule } i.\text{reduced-GB-ideal-Polys}, \text{rule } i.\text{reduced-GB-nonzero-Polys}, \text{rule } i.\text{reduced-GB-is-GB-Polys})$
define G' **where** $G' = \text{focus } X \ ' \ G$
from $\text{fin-G } \langle 0 \notin G \rangle$ **have** $\text{fin-G': finite } G'$ **and** $0 \notin G'$ **by** $(\text{auto simp: } G'\text{-def})$
have $\text{G'-sub: } G' \subseteq P[X]$ **by** $(\text{auto simp: } G'\text{-def intro: focus-in-Polys})$
define G'' **where** $G'' = i.\text{lef } ' \ G'$
from $\langle 0 \notin G' \rangle$ **have** $0 \notin G''$ **by** $(\text{auto simp: } G''\text{-def } i.\text{punit.lc-eq-zero-iff})$
have $\text{lookup-focus-in: lookup (focus } X \ g) \ t \in P[\{x\}] \text{ if } g \in G \text{ for } g \ t$
proof –
have $\text{lookup (focus } X \ g) \ t \in \text{range (lookup (focus } X \ g))}$ **by** (rule rangeI)
from $\text{that } G\text{-sub}$ **have** $g \in P[\text{insert } x \ X]$ **..**
hence $\text{range (lookup (focus } X \ g)) \subseteq P[\text{insert } x \ X - X]$ **by** $(\text{rule focus-coeffs-subset-Polys'})$
with $\langle - \in \text{range } - \rangle$ **have** $\text{lookup (focus } X \ g) \ t \in P[\text{insert } x \ X - X]$ **..**
also have $\text{insert } x \ X - X = \{x\}$ **by** (simp only: eq2)
finally show $?thesis$.
qed
hence $\text{lef-in: i.lef (focus } X \ g) \in P[\{x\}] \text{ if } g \in G \text{ for } g$
unfolding $i.\text{punit.lc-def}$ **using** that by blast
have $\text{G''-sub: } G'' \subseteq P[\{x\}]$
proof
fix c
assume $c \in G''$
then obtain g' **where** $g' \in G'$ **and** $c = i.\text{lef } g'$ **unfolding** $G''\text{-def ..}$
from $\langle g' \in G' \rangle$ **obtain** g **where** $g \in G$ **and** $g': g' = \text{focus } X \ g$ **unfolding** $G'\text{-def ..}$
from $\text{this}(1)$ **show** $c \in P[\{x\}]$ **unfolding** $c \ g'$ **by** (rule lef-in)
qed
define P **where** $P = \text{poly-of-pm } x \ ' \ G''$
from fin-G' **have** $\text{fin-P: finite } P$ **by** $(\text{simp add: } P\text{-def } G''\text{-def})$
have $0 \notin P$
proof
assume $0 \in P$
then obtain g'' **where** $g'' \in G''$ **and** $0 = \text{poly-of-pm } x \ g''$ **unfolding** $P\text{-def ..}$
from $\text{this}(2)$ **have** $*$: $\text{keys } g'' \cap .[\{x\}] = \{\}$ **by** $(\text{simp add: poly-of-pm-eq-zero-iff})$
from $\langle g'' \in G'' \rangle$ G''-sub **have** $g'' \in P[\{x\}]$ **..**
hence $\text{keys } g'' \subseteq .[\{x\}]$ **by** (rule PolysD)
with $*$ **have** $\text{keys } g'' = \{\}$ **by** blast

with $\langle g'' \in G'' \rangle \langle 0 \notin G'' \rangle$ **show** *False* **by** *simp*
qed
define Z **where** $Z = (\bigcup p \in P. \{z. \text{poly } p \ z = 0\})$
have *finite* Z **unfolding** $Z\text{-def}$ **using** $\text{fin-}P$
proof (*rule finite-UN-I*)
fix p
assume $p \in P$
with $\langle 0 \notin P \rangle$ **have** $p \neq 0$ **by** *blast*
thus *finite* $\{z. \text{poly } p \ z = 0\}$ **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 $a\text{-nz: poly-eval } (\lambda-. a) (i.\text{lc}f \text{ (focus } X \ g)) \neq 0$ **if** $g \in G$ **for** g
proof –
from *that G-sub* **have** $g \in P[\text{insert } x \ X]$ **..**
have $\text{poly-eval } (\lambda-. a) (i.\text{lc}f \text{ (focus } X \ g)) = \text{poly } (\text{poly-of-pm } x \ (i.\text{lc}f \text{ (focus } X \ g)))$ a
by (*rule sym, intro poly-eq-poly-eval' lc-f-in that*)
moreover have $\text{poly-of-pm } x \ (i.\text{lc}f \text{ (focus } X \ g)) \in P$
by (*auto simp: P-def G''-def G'-def that intro!: imageI*)
ultimately show *?thesis* **using** $\langle a \notin Z \rangle$ **by** (*simp add: Z-def*)
qed

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

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

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

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

have $\text{ideal } G3 \cap P[- \ \{x\}] = ?e \ \text{‘ focus } \{x\} \ \text{‘ ideal } G$
 by (simp only: $G3\text{-def image-poly-eval-focus-ideal}$)
 also have $\dots = \text{ideal } (?e \ \text{‘ focus } \{x\} \ \text{‘ } F) \cap P[- \ \{x\}]$
 by (simp only: $\text{ideal-}G \ \text{image-poly-eval-focus-ideal}$)
 finally have $\text{eq3: ideal } G3 \cap P[- \ \{x\}] = \text{ideal } (?e \ \text{‘ focus } \{x\} \ \text{‘ } F) \cap P[- \ \{x\}]$

•
 from $\text{assms}(1) \ \langle G3 \subseteq P[X] \rangle \ \langle \text{finite } G3 \rangle$ have $G3\text{-isGB: } i.\text{punit.is-Groebner-basis } G3$
 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 $g1: g1 = ?e (\text{focus } \{x\} \ g1')$
 unfolding $G3\text{-def}$ by blast
 from $\text{this}(1)$ have $\text{lpp1: } i.\text{lpp } g1 = \text{except } (i.\text{lpp } g1') \ \{x\}$ unfolding $g1$ by
 (rule lpp-e-focus)
 from $\langle g1' \in G \rangle \ G\text{-sub}$ have $g1' \in P[\text{insert } x \ X] \ ..$
 assume $g2 \in G3$
 then obtain $g2'$ where $g2' \in G$ and $g2: g2 = ?e (\text{focus } \{x\} \ g2')$
 unfolding $G3\text{-def}$ by blast
 from $\text{this}(1)$ have $\text{lpp2: } i.\text{lpp } g2 = \text{except } (i.\text{lpp } g2') \ \{x\}$ unfolding $g2$ by
 (rule lpp-e-focus)
 from $\langle g2' \in G \rangle \ G\text{-sub}$ have $g2' \in P[\text{insert } x \ X] \ ..$

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

define s **where** $s = c2 * \text{punit.monom-mult } 1 (l - \text{except } (i.\text{lpp } g1') \{x\})$
 $g1' -$
 $c1 * \text{punit.monom-mult } 1 (l - \text{except } (i.\text{lpp } g2') \{x\}) g2'$
have $c1 \in P[\{x\}]$ **unfolding** $c1\text{-def}$ **using** $\langle g1' \in G \rangle$ **by** (*rule lcf-in*)
hence $\text{eval-c1}: \text{poly-eval } (\lambda-. \text{monomial } a \ 0) (\text{focus } \{x\} \ c1) = \text{monomial}$
 $(\text{poly-eval } (\lambda-. a) \ c1) \ 0$
by (*simp add: focus-Polys poly-eval-sum poly-eval-monomial monomial-power-map-scale*
times-monomial-monomial flip: punit.monomial-prod-sum mono-
mial-sum)
(simp add: poly-eval-alt)
have $c2 \in P[\{x\}]$ **unfolding** $c2\text{-def}$ **using** $\langle g2' \in G \rangle$ **by** (*rule lcf-in*)
hence $\text{eval-c2}: \text{poly-eval } (\lambda-. \text{monomial } a \ 0) (\text{focus } \{x\} \ c2) = \text{monomial}$
 $(\text{poly-eval } (\lambda-. a) \ c2) \ 0$
by (*simp add: focus-Polys poly-eval-sum poly-eval-monomial monomial-power-map-scale*
times-monomial-monomial flip: punit.monomial-prod-sum mono-
mial-sum)
(simp add: poly-eval-alt)

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

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

define q **where** $q = (\lambda g. \text{inverse } c \cdot (\sum h \in \{y \in G. ?e \ (\text{focus } \{x\} \ y) = g\}. ?e$
 $(\text{focus } \{x\} \ (q0 \ h))))$

have $eq4: ?e \ (\text{focus } \{x\} \ (\text{monomial } 1 \ (l - t))) = \text{monomial } 1 \ (l - t)$ **for** t
proof –
have $\text{focus } \{x\} \ (\text{monomial } (1::'a) \ (l - t)) = \text{monomial } (\text{monomial } 1 \ (l -$
 $t)) \ 0$
proof $(\text{intro focus-Polys-Compl Polys-closed-monomial PPs-closed-minus})$
from $\langle x \notin X \rangle$ **have** $X \subseteq - \{x\}$ **by** simp
hence $.[X] \subseteq .[- \{x\}]$ **by** (rule PPs-mono)
with $\langle l \in .[X] \rangle$ **show** $l \in .[- \{x\}] \ \dots$
qed
thus $?thesis$ **by** $(\text{simp add: poly-eval-monomial})$
qed
from $c2\text{-nz}$ **have** $eq5: \text{inverse } c * \text{poly-eval } (\lambda-. \ a) \ c2 = 1 / \text{lookup } g1 \ (i.\text{lpp}$
 $g1)$
unfolding lpp1 **using** $\langle g1' \in P[-] \rangle$
by $(\text{simp add: c-def mult.assoc divide-inverse-commute } g1 \ \text{lookup-e-focus}$
 $\text{flip: lpp-focus } i.\text{punit.lc-def } c1\text{-def})$
from $c1\text{-nz}$ **have** $eq6: \text{inverse } c * \text{poly-eval } (\lambda-. \ a) \ c1 = 1 / \text{lookup } g2 \ (i.\text{lpp}$
 $g2)$
unfolding lpp2 **using** $\langle g2' \in P[-] \rangle$
by $(\text{simp add: c-def mult.assoc mult.left-commute[of inverse (poly-eval } (\lambda-. \ a) \ c1])]$
 $\text{divide-inverse-commute } g2 \ \text{lookup-e-focus flip: lpp-focus } i.\text{punit.lc-def}$
 $c2\text{-def})$
have $l\text{-alt}: l = \text{lcs } (i.\text{lpp } g1) \ (i.\text{lpp } g2)$ **by** $(\text{simp only: l-def lpp1 lpp2})$
have $\text{spoly-eq}: i.\text{punit.spoly } g1 \ g2 = (\text{inverse } c) \cdot ?e \ (\text{focus } \{x\} \ s)$
by $(\text{simp add: s-def focus-minus focus-times poly-eval-minus poly-eval-times}$
 eval-c1 eval-c2
 $eq4 \ eq5 \ eq6 \ \text{map-scale-eq-times times-monomial-monomial}$
 $\text{right-diff-distrib}$
 $i.\text{punit.spoly-def Let-def}$
 $\text{flip: mult.assoc times-monomial-left } g1 \ g2 \ \text{lpp1 lpp2 l-alt})$
also **have** $\dots = (\sum g \in G. \text{inverse } c \cdot (?e \ (\text{focus } \{x\} \ (q0 \ g))) * ?e \ (\text{focus } \{x\} \ g))$
 $g))$
by $(\text{simp add: 1 focus-sum poly-eval-sum focus-times poly-eval-times map-scale-sum-distrib-left})$
also **have** $\dots = (\sum g \in G \exists. \sum h \in \{y \in G. ?e \ (\text{focus } \{x\} \ y) = g\}. \text{inverse } c \cdot (?e \ (\text{focus } \{x\} \ (q0 \ h))) * ?e \ (\text{focus } \{x\} \ h))$
 $\text{inverse } c \cdot (?e \ (\text{focus } \{x\} \ (q0 \ h))) * ?e \ (\text{focus } \{x\} \ h))$
unfolding $G3\text{-def image-image}$ **using** fin-G **by** $(\text{rule sum.image-gen})$
also **have** $\dots = (\sum g \in G \exists. \text{inverse } c \cdot (\sum h \in \{y \in G. ?e \ (\text{focus } \{x\} \ y) = g\}. ?e$
 $(\text{focus } \{x\} \ (q0 \ h))) * g)$
by $(\text{intro sum.cong refl}) \ (\text{simp add: map-scale-eq-times sum-distrib-left})$

sum-distrib-right mult.assoc
also from *reft* **have** $\dots = (\sum_{g \in G^3}. q\ g * g)$ **by** (*rule sum.cong*) (*simp add: q-def sum-distrib-right*)
finally have $i.punit.spoly\ g1\ g2 = (\sum_{g \in G^3}. q\ g * g)$.
thus $i.punit.spoly\text{-rep}\ (varnum\ X)\ 0\ G^3\ g1\ g2$
proof (*rule i.punit.spoly-repI[simplified, where m=0 and d=varnum X, simplified i.dgrad-p-set-varnum]*)
fix g
show $q\ g \in P[X]$ **unfolding** *q-def*
proof (*intro Polys-closed-map-scale Polys-closed-sum*)
fix $g0$
from $\langle q0\ g0 \in P[\text{insert } x\ X] \rangle$ **have** $?e\ (focus\ \{x\}\ (q0\ g0)) \in P[\text{insert } x\ X - \{x\}]$
by (*rule poly-eval-focus-in-Polys*)
thus $?e\ (focus\ \{x\}\ (q0\ g0)) \in P[X]$ **by** (*simp only: eq1*)
qed

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

```

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 ∈ (λv. 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-exception)
  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 $g2' = \text{flatten } (\text{focus } X \ g2')$ **by** *simp*
also have $\dots = \text{flatten } (\text{monomial } c2 \ (i.\text{lpp } (\text{focus } X \ g2')) + i.\text{punit.tail } (\text{focus } X \ g2'))$
by (*simp only: c2-def flip: i.punit.leading-monomial-tail*)
also from $\langle g2' \in P[-] \rangle$ **have** $\dots = \text{punit.monom-mult } 1 \ (\text{except } (i.\text{lpp } g2') \ {x}) \ c2 +$
 $\text{flatten } (i.\text{punit.tail } (\text{focus } X \ g2'))$
by (*simp only: flatten-plus flatten-monomial lpp-focus*)
finally have $\text{punit.monom-mult } 1 \ (\text{except } (i.\text{lpp } g2') \ {x}) \ c2 +$
 $\text{flatten } (i.\text{punit.tail } (\text{focus } X \ g2')) = g2' \ (\text{is } ?l = -) \ \text{by}$
(rule sym)
moreover have $c1 * \text{punit.monom-mult } 1 \ (l - \text{except } (i.\text{lpp } g2') \ {x}) \ ?l$
 $=$
 $\text{punit.monom-mult } 1 \ l \ (c1 * c2) +$
 $c1 * \text{punit.monom-mult } 1 \ (l - i.\text{lpp } (\text{focus } X \ g2'))$
 $(\text{flatten } (i.\text{punit.tail } (\text{focus } X \ g2')))$
 $(\text{is } - = \text{punit.monom-mult } 1 \ l \ (c1 * c2) + ?b)$
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 $\langle g2' \in P[-] \rangle$ flip: times-monomial-left)
ultimately have $b: c1 * \text{punit.monom-mult } 1 \ (l - \text{except } (i.\text{lpp } g2') \ {x})$
 $g2' =$
 $\text{punit.monom-mult } 1 \ l \ (c1 * c2) + ?b \ \text{by } \text{simp}$

have *lex-pm-strict-t: lex-pm-strict* $t \ (l - i.\text{lpp } (\text{focus } X \ h) + i.\text{lpp } (\text{focus } X \ h))$
if $t \in \text{keys } (d * \text{punit.monom-mult } 1 \ (l - i.\text{lpp } (\text{focus } X \ h))$
 $(\text{flatten } (i.\text{punit.tail } (\text{focus } X \ h))))$
and $h \in G$ **and** $d \in P[\{x\}]$ **for** $d \ h$
proof –
have $0: \text{lex-pm-strict } (u + v) \ w$ **if** *lex-pm-strict* $v \ w$ **and** $w \in .[X]$ **and**
 $u \in .[\{x\}]$
for $u \ v \ w$ **using** *that(1)*
proof (*rule lex-pm-strict-plus-left*)
fix $y \ z$
assume $y \in \text{keys } w$
also from *that(2)* **have** $\dots \subseteq X$ **by** (*rule PPsD*)
also have $\dots \subseteq \{.\langle x \rangle\}$ **by** *fact*
finally have $y < x$ **by** *simp*
assume $z \in \text{keys } u$
also from *that(3)* **have** $\dots \subseteq \{x\}$ **by** (*rule PPsD*)
finally show $y < z$ **using** $\langle y < x \rangle$ **by** *simp*
qed
let $?h = \text{focus } X \ h$
from *that(2)* **have** $?h \in G'$ **by** (*simp add: G'-def*)
with $\langle G' \subseteq P[X] \rangle$ **have** $?h \in P[X]$ **..**
hence $i.\text{lpp } ?h \in .[X]$ **by** (*rule i.PPs-closed-lpp*)
from *that(1)* **obtain** $t1 \ t2$ **where** $t1 \in \text{keys } d$
and $t2 \in \text{keys } (\text{punit.monom-mult } 1 \ (l - i.\text{lpp } ?h) (\text{flatten } (i.\text{punit.tail } (\text{focus } X \ h))))$

?h)))

and $t: t = t1 + t2$ by (rule *in-keys-timesE*)

from *this(2)* obtain $t3$ where $t3 \in \text{keys}$ (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 $t4 \in \text{keys}$ (*i.punit.tail ?h*)

and $t5\text{-in}: t5 \in \text{keys}$ (lookup (*i.punit.tail ?h*) $t4$) and $t3: t3 = t4 + t5$

using *keys-flatten-subset* by blast

from *this(1)* have $1: \text{lex-pm-strict } t4$ (*i.lpp ?h*) by (rule *i.punit.keys-tail-less-lt*)

from *that(2)* have lookup $?h t4 \in P[\{x\}]$ by (rule *lookup-focus-in*)

hence keys (lookup $?h t4$) $\subseteq .[\{x\}]$ by (rule *PolysD*)

moreover from $t5\text{-in}$ have $t5\text{-in}: t5 \in \text{keys}$ (lookup $?h t4$)

by (simp add: *i.punit.lookup-tail split: if-split-asm*)

ultimately have $t5 \in .[\{x\}]$..

with $1 \langle i.lpp ?h \in \cdot \rangle$ 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 $\langle l \in .[X] \rangle \langle i.lpp ?h \in .[X] \rangle$ have $l - i.lpp ?h + i.lpp$

$?h \in .[X]$

by (intro *PPs-closed-plus PPs-closed-minus*)

moreover from $\langle t1 \in \text{keys } d \rangle$ *that(3)* have $t1 \in .[\{x\}]$ by (auto dest: *PolysD*)

ultimately show *?thesis* unfolding t by (rule 0)

qed

show lookup $s t = 0$

proof (rule *ccontr*)

assume lookup $s t \neq 0$

hence $t \in \text{keys } s$ by (simp add: *in-keys-iff*)

also have $\dots = \text{keys } (?a - ?b)$ by (simp add: *s-def a b*)

also have $\dots \subseteq \text{keys } ?a \cup \text{keys } ?b$ by (fact *keys-minus*)

finally show *False*

proof

assume $t \in \text{keys } ?a$

hence *lex-pm-strict* t ($l - i.lpp$ (focus $X g1'$) + *i.lpp* (focus $X g1'$))

using $\langle g1' \in G \rangle \langle c2 \in P[\{x\}] \rangle$ by (rule *lex-pm-strict-t*)

with $\langle g1' \in P[-] \rangle$ have *lex-pm-strict* $t l$

by (simp add: *lpp-focus l-def minus-plus adds-lcs*)

with $\langle \text{lex-pm } l t \rangle$ show *?thesis* by simp

next

assume $t \in \text{keys } ?b$

hence *lex-pm-strict* t ($l - i.lpp$ (focus $X g2'$) + *i.lpp* (focus $X g2'$))

using $\langle g2' \in G \rangle \langle c1 \in P[\{x\}] \rangle$ by (rule *lex-pm-strict-t*)

with $\langle g2' \in P[-] \rangle$ have *lex-pm-strict* $t l$

by (simp add: *lpp-focus l-def minus-plus adds-lcs-2*)

with $\langle \text{lex-pm } l t \rangle$ show *?thesis* by simp

qed

qed

qed

also have $\dots = \text{lcs}$ (*i.lpp g1*) (*i.lpp g2*) by (simp only: *l-def lpp1 lpp2*)

finally show $\text{lex-pm-strict } (i.\text{lpp } (q \ g \ * \ g)) \ (lcs \ (i.\text{lpp } g1) \ (i.\text{lpp } g2)) \ .$
qed
qed
have $1 \in \text{ideal } (?e \ \text{focus } \{x\} \ 'F) \longleftrightarrow 1 \in \text{ideal } (?e \ \text{focus } \{x\} \ 'F) \cap P[-\{x\}]$
by (*simp add: one-in-Polys*)
also have $\dots \longleftrightarrow 1 \in \text{ideal } G3$ **by** (*simp add: one-in-Polys flip: eq3*)
also have $\neg \dots$
proof
note *G3-isGB*
moreover assume $1 \in \text{ideal } G3$
moreover have $1 \neq (0::-\Rightarrow_0 \ 'a)$ **by** *simp*
ultimately obtain g **where** $g \in G3$ **and** $g \neq 0$ **and** $i.\text{lpp } g$ **adds** $i.\text{lpp } (1::-\Rightarrow_0 \ 'a)$
by (*rule i.punit.GB-adds-lt[simplified]*)
from *this(3)* **have** $i.\text{lpp } g = 0$ **by** (*simp add: i.punit.lt-monomial adds-zero flip: single-one*)
hence $\text{monomial } (i.\text{lcf } g) \ 0 = g$ **by** (*rule i.punit.lt-eq-min-term-monomial[simplified]*)
from $\langle g \in G3 \rangle$ **obtain** g' **where** $g' \in G$ **and** $g: g = ?e \ (\text{focus } \{x\} \ g')$ **by** (*auto simp: G3-def*)
from *this(1)* **have** $i.\text{lpp } g = \text{except } (i.\text{lpp } g') \ \{x\}$ **unfolding** g **by** (*rule lpp-e-focus*)
hence $\text{keys } (i.\text{lpp } g') \subseteq \{x\}$ **by** (*simp add: $\langle i.\text{lpp } g = 0 \rangle$ except-eq-zero-iff*)
have $g' \in P[\{x\}]$
proof (*intro PolysI subsetI PPsI*)
fix $t \ y$
assume $t \in \text{keys } g'$
hence $\text{lex-pm } t \ (i.\text{lpp } g')$ **by** (*rule i.punit.lt-max-keys*)
moreover assume $y \in \text{keys } t$
ultimately obtain z **where** $z \in \text{keys } (i.\text{lpp } g')$ **and** $z \leq y$ **by** (*rule lex-pm-keys-leE*)
with $\langle \text{keys } (i.\text{lpp } g') \subseteq \{x\} \rangle$ **have** $x \leq y$ **by** *blast*
from $\langle g' \in G \rangle$ *G-sub* **have** $g' \in P[\text{insert } x \ X]$ **..**
hence $\text{indets } g' \subseteq \text{insert } x \ X$ **by** (*rule PolysD*)
moreover from $\langle y \in - \rangle \ \langle t \in - \rangle$ **have** $y \in \text{indets } g'$ **by** (*rule in-indetsI*)
ultimately have $y \in \text{insert } x \ X$ **..**
thus $y \in \{x\}$
proof
assume $y \in X$
with *assms(3)* **have** $y \in \{..<x\}$ **..**
with $\langle x \leq y \rangle$ **show** *?thesis* **by** *simp*
qed *simp*
qed
moreover from $\langle g' \in G \rangle$ **have** $g' \in \text{ideal } G$ **by** (*rule ideal.span-base*)
ultimately have $g' \in \text{ideal } F \cap P[\{x\}]$ **by** (*simp add: ideal-G*)
with *assms(5)* **have** $g' = 0$ **by** *blast*
hence $g = 0$ **by** (*simp add: g*)
with $\langle g \neq 0 \rangle$ **show** *False* **..**
qed

finally show $1 \notin \text{ideal } (?e \text{ 'focus } \{x\} \text{ ' } F)$.
qed
qed

lemma *weak-Nullstellensatz-aux-3*:
assumes $F \subseteq P[\text{insert } x \ X]$ **and** $x \notin X$ **and** $1 \notin \text{ideal } F$ **and** $\neg \text{ideal } F \cap P[\{x\}] \subseteq \{0\}$
obtains $a::\text{'a::alg-closed-field}$ **where** $1 \notin \text{ideal } (\lambda\cdot. \text{monomial } a \ 0)$ **'focus** $\{x\}$ **'** F)
proof –
let $?x = \text{monomial } 1 \ (\text{Poly-Mapping.single } x \ 1)$
from *assms(4)* **obtain** f **where** $f \in \text{ideal } F$ **and** $f \in P[\{x\}]$ **and** $f \neq 0$ **by** *blast*
define p **where** $p = \text{poly-of-pm } x \ f$
from $\langle f \in P[\{x\}] \rangle \langle f \neq 0 \rangle$ **have** $p \neq 0$
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 : $p = \text{Polynomial.smult } c \ (\prod_{a \in A}. [-a, 1:] \wedge m \ a)$
and $\bigwedge x. m \ x = 0 \longleftrightarrow x \notin A$ **and** $c = 0 \longleftrightarrow p = 0$ **and** $\bigwedge z. \text{poly } p \ z = 0 \longleftrightarrow (c = 0 \vee z \in A)$
by (*rule linear-factorsE*) *blast*
from *this(4, 5)* **have** $c \neq 0$ **and** $\bigwedge z. \text{poly } p \ z = 0 \longleftrightarrow z \in A$ **by** (*simp-all add: \langle p \neq 0 \rangle*)
have $\exists a \in A. 1 \notin \text{ideal } (\text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0)) \text{ 'focus } \{x\} \text{ ' } F)$
proof (*rule ccontr*)
assume *asm*: $\neg (\exists a \in A. 1 \notin \text{ideal } (\text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0)) \text{ 'focus } \{x\} \text{ ' } F)$
obtain $g \ h$ **where** $g \ a \in \text{ideal } F$ **and** $1: h \ a * (?x - \text{monomial } a \ 0) + g \ a = 1$
if $a \in A$ **for** a
proof –
define P **where** $P = (\lambda gh \ a. \text{fst } gh \in \text{ideal } F \wedge \text{fst } gh + \text{snd } gh * (?x - \text{monomial } a \ 0) = 1)$
define gh **where** $gh = (\lambda a. \text{SOME } gh. P \ gh \ a)$
show *?thesis*
proof
fix a
assume $a \in A$
with *asm* **have** $1 \in \text{ideal } (\text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0)) \text{ 'focus } \{x\} \text{ ' } F)$ **by** *blast*
hence $1 \in \text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0) \text{ 'focus } \{x\} \text{ ' ideal } F$
by (*simp add: image-poly-eval-focus-ideal one-in-Polys*)
then obtain g **where** $g \in \text{ideal } F$ **and** $1 = \text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0)$ (*focus* $\{x\}$ g)
unfolding *image-image ..*
note *this(2)*
also have $\text{poly-eval } (\lambda\cdot. \text{monomial } a \ 0) (\text{focus } \{x\} \ g) = \text{poly } (\text{poly-of-focus } x \ g)$ (*monomial* $a \ 0$)
by (*simp only: poly-poly-of-focus*)
also have $\dots = \text{poly } (\text{poly-of-focus } x \ g) (?x - (?x - \text{monomial } a \ 0))$ **by** *simp*

also obtain h **where** $\dots = \text{poly}(\text{poly-of-focus } x \ g) \ ?x - h * (?x - \text{monomial } a \ 0)$
by (*rule poly-minus-rightE*)
also have $\dots = g - h * (?x - \text{monomial } a \ 0)$ **by** (*simp only: poly-poly-of-focus-monomial*)
finally have $g - h * (?x - \text{monomial } a \ 0) = 1$ **by** (*rule sym*)
with $\langle g \in \text{ideal } F \rangle$ **have** $P(g, -h) \ a$ **by** (*simp add: P-def*)
hence $P(gh \ a) \ a$ **unfolding** *gh-def* **by** (*rule someI*)
thus $\text{fst}(gh \ a) \in \text{ideal } F$ **and** $\text{snd}(gh \ a) * (?x - \text{monomial } a \ 0) + \text{fst}(gh \ a) = 1$
by (*simp-all only: P-def add.commute*)
qed
qed
from *this(1)* **obtain** g' **where** $g' \in \text{ideal } F$
and $2: (\prod a \in A. (h \ a * (?x - \text{monomial } a \ 0) + g \ a) \ ^m \ a) = (\prod a \in A. (h \ a * (?x - \text{monomial } a \ 0)) \ ^m \ a) + g'$
by (*rule weak-Nullstellensatz-aux-1*)
have $1 = (\prod a \in A. (h \ a * (?x - \text{monomial } a \ 0) + g \ a) \ ^m \ a)$
by (*rule sym*) (*intro prod.neutral ballI, simp only: 1 power-one*)
also have $\dots = (\prod a \in A. h \ a \ ^m \ a) * (\prod a \in A. (?x - \text{monomial } a \ 0) \ ^m \ a) + g'$
by (*simp only: 2 power-mult-distrib prod.distrib*)
also have $(\prod a \in A. (?x - \text{monomial } a \ 0) \ ^m \ a) = \text{pm-of-poly } x \ (\prod a \in A. [:- a, 1:] \ ^m \ a)$
by (*simp add: pm-of-poly-prod pm-of-poly-pCons single-uminus punit.monom-mult-monomial flip: single-one*)
also from $\langle c \neq 0 \rangle$ **have** $\dots = \text{monomial}(\text{inverse } c) \ 0 * \text{pm-of-poly } x \ p$
by (*simp add: p map-scale-assoc flip: map-scale-eq-times*)
also from $\langle f \in P[\{x\}] \rangle$ **have** $\dots = \text{monomial}(\text{inverse } c) \ 0 * f$
by (*simp only: <p = poly-of-pm x f> pm-of-poly-of-pm*)
finally have $1 = ((\prod a \in A. h \ a \ ^m \ a) * \text{monomial}(\text{inverse } c) \ 0) * f + g'$
by (*simp only: mult.assoc*)
also from $\langle f \in \text{ideal } F \rangle \ \langle g' \in \text{ideal } F \rangle$ **have** $\dots \in \text{ideal } F$ **by** (*intro ideal.span-add ideal.span-scale*)
finally have $1 \in \text{ideal } F$.
with *assms(3)* **show** *False ..*
qed
then obtain a **where** $1 \notin \text{ideal}(\text{poly-eval}(\lambda-. \text{monomial } a \ 0) \ \text{'focus } \{x\} \ \text{' } F)$
..
thus *?thesis ..*
qed

theorem *weak-Nullstellensatz:*

assumes *finite* X **and** $F \subseteq P[X]$ **and** $\mathcal{V} \ F = (\{\}::(\text{'x}::\{\text{countable}, \text{linorder}\}) \Rightarrow \text{'a}::\text{alg-closed-field}) \ \text{set}$

shows $\text{ideal } F = \text{UNIV}$

unfolding *ideal-eq-UNIV-iff-contains-one*

proof (*rule ccontr*)

assume $1 \notin \text{ideal } F$

with *assms(1, 2)* **obtain** a **where** $1 \notin \text{ideal}(\text{poly-eval } a \ \text{' } F)$

```

proof (induct X arbitrary: F thesis rule: finite-linorder-induct)
  case empty
  have  $F \subseteq \{0\}$ 
  proof
    fix f
    assume  $f \in F$ 
    with empty.premis(2) have  $f \in P[\{\}]$  ..
    then obtain c where  $f = \text{monomial } c \ 0$  unfolding Polys-empty ..
    also have  $c = 0$ 
    proof (rule ccontr)
      assume  $c \neq 0$ 
      from  $\langle f \in F \rangle$  have  $f \in \text{ideal } F$  by (rule ideal.span-base)
      hence  $\text{monomial } (\text{inverse } c) \ 0 * f \in \text{ideal } F$  by (rule ideal.span-scale)
      with  $\langle c \neq 0 \rangle$  have  $1 \in \text{ideal } F$  by (simp add: f times-monomial-monomial)
      with empty.premis(3) show False ..
    qed
    finally show  $f \in \{0\}$  by simp
  qed
  hence  $\text{poly-eval } 0 \ ' F \subseteq \{0\}$  by auto
  hence  $\text{ideal } (\text{poly-eval } 0 \ ' F) = \{0\}$  by simp
  hence  $1 \notin \text{ideal } (\text{poly-eval } 0 \ ' F)$  by (simp del: ideal-eq-zero-iff)
  thus ?case by (rule empty.premis)
next
  case (insert x X)
  obtain a0 where  $1 \notin \text{ideal } (\text{poly-eval } (\lambda-. \text{monomial } a0 \ 0) \ ' \text{focus } \{x\} \ ' F)$ 
(is -  $\notin \text{ideal } ?F$ )
  proof (cases  $\text{ideal } F \cap P[\{x\}] \subseteq \{0\}$ )
    case True
      with insert.hyps(1) insert.premis(2) insert.hyps(2) insert.premis(3) obtain
a0
      where  $1 \notin \text{ideal } (\text{poly-eval } (\lambda-. \text{monomial } a0 \ 0) \ ' \text{focus } \{x\} \ ' F)$ 
by (rule weak-Nullstellensatz-aux-2)
      thus ?thesis ..
    next
    case False
      from insert.hyps(2) have  $x \notin X$  by blast
      with insert.premis(2) obtain a0 where  $1 \notin \text{ideal } (\text{poly-eval } (\lambda-. \text{monomial } a0 \ 0) \ ' \text{focus } \{x\} \ ' F)$ 
using insert.premis(3) False by (rule weak-Nullstellensatz-aux-3)
      thus ?thesis ..
  qed
moreover have  $?F \subseteq P[X]$ 
proof -
  {
    fix f
    assume  $f \in F$ 
    with insert.premis(2) have  $f \in P[\text{insert } x \ X]$  ..
    hence  $\text{poly-eval } (\lambda-. \text{monomial } a0 \ 0) (\text{focus } \{x\} \ f) \in P[\text{insert } x \ X - \{x\}]$ 
by (rule poly-eval-focus-in-Polys)
  }

```

also have $\dots \subseteq P[X]$ by (rule *Polys-mono*) simp
 finally have *poly-eval* ($\lambda\cdot$. monomial $a0$ 0) (focus $\{x\}$ f) $\in P[X]$.
 }
 thus ?thesis by blast
 qed
 ultimately obtain $a1$ where $1 \notin \text{ideal } (\text{poly-eval } a1 \text{ ' } ?F)$ using *insert.hyps*(3)
 by blast
 also have *poly-eval* $a1 \text{ ' } ?F = \text{poly-eval } (a1(x := \text{poly-eval } a1 (\text{monomial } a0 \text{ } 0))) \text{ ' } F$
 by (simp add: *image-image poly-eval-poly-eval-focus fun-upd-def*)
 finally show ?case by (rule *insert.prem*s)
 qed
 hence *ideal* (*poly-eval* $a \text{ ' } F$) $\neq UNIV$ by (simp add: *ideal-eq-UNIV-iff-contains-one*)
 hence *ideal* (*poly-eval* $a \text{ ' } F$) = $\{0\}$ using *ideal-field-disj*[of *poly-eval* $a \text{ ' } F$] by
 blast
 hence *poly-eval* $a \text{ ' } F \subseteq \{0\}$ by simp
 hence $a \in \mathcal{V} F$ by (rule *variety-ofI-alt*)
 thus False by (simp add: *assms*(3))
 qed

lemma *radical-idealI*:

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

```

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  $\langle f \in P[- \{x\}] \rangle$  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  $\langle f \neq 0 \rangle$ )
have  $eq2$ : poly (map-poly  $g$  (poly-of-focus  $x f'$ )) (Fract 1  $f$ ) = Fract  $f' 1$  if  $f' \in F''$  for  $f'$ 
proof -
from that  $F''$ -sub have  $f' \in F$  ..
with assms(2) have  $f' \in P[X]$  ..
with  $\langle P[X] \subseteq - \rangle$  have  $f' \in P[- \{x\}]$  ..
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  $p0m0 = (\lambda f'. SOME z. poly (map-poly g (poly-of-focus x (q f')))) (Fract 1 f) =$ 
 $Fract (fst z) (f \wedge snd z))$ 
define  $p0$  where  $p0 = fst \circ p0m0$ 
define  $m0$  where  $m0 = snd \circ p0m0$ 
define  $m$  where  $m = Max (m0 ' F'')$ 
have  $eq3$ : poly (map-poly  $g$  (poly-of-focus  $x (q f')$ )) (Fract 1  $f$ ) = Fract ( $p0 f'$ )
( $f \wedge m0 f'$ )
for  $f'$ 
proof -
have  $g a = 0 \iff a = 0$  for  $a$  by (simp add: g-def Fract-eq-zero-iff)
hence set (Polynomial.coeffs (map-poly  $g$  (poly-of-focus  $x (q f')$ )))  $\subseteq$  range
( $\lambda x. Fract x 1$ )
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

```

$f) = \text{Fract } p (f \wedge m')$
by (*rule poly-Fract*)
hence $\text{poly} (\text{map-poly } g (\text{poly-of-focus } x (q f')) (\text{Fract } 1 f) = \text{Fract} (\text{fst } (p, m')) (f \wedge \text{snd } (p, m'))$
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 $\text{poly-of-focus } x 1 = \text{poly-of-focus } x (q0 * (1 - ?f) + (\sum f' \in F''. q f' * f'))$
by (*rule arg-cong*)
hence $1 = \text{poly} (\text{map-poly } g (\text{poly-of-focus } x (q0 * (1 - ?f) + (\sum f' \in F''. q f' * f')))) (\text{Fract } 1 f)$
by (*simp add: g-1*)
also have $\dots = \text{poly} (\text{map-poly } g (\text{poly-of-focus } x (\sum f' \in F''. q f' * f')) (\text{Fract } 1 f)$
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 $\dots = (\sum f' \in F''. \text{Fract } (p0 f') (f \wedge m0 f') * \text{Fract } f' 1)$
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 $\text{Fract } (f \wedge m) 1 = \text{Fract } (f \wedge m) 1 * (\sum f' \in F''. \text{Fract } (p0 f' * f') (f \wedge m0 f'))$
by (*simp*)
also have $\dots = (\sum f' \in F''. \text{Fract } (f \wedge m * (p0 f' * f')) (f \wedge m0 f'))$
by (*simp add: sum-distrib-left*)
also from *refl* **have** $\dots = (\sum f' \in F''. \text{Fract } ((f \wedge (m - m0 f') * p0 f') * f') 1)$
proof (*rule sum.cong*)
fix f'
assume $f' \in F''$
hence $m0 f' \in m0 ' F''$ **by** (*rule imageI*)
with - have $m0 f' \leq m$ **unfolding** *m-def* **by** (*rule Max-ge*) (*simp add: fin''*)
hence $f \wedge m = f \wedge (m0 f') * f \wedge (m - m0 f')$ **by** (*simp flip: power-add*)
hence $\text{Fract } (f \wedge m * (p0 f' * f')) (f \wedge m0 f') = \text{Fract } (f \wedge m0 f') (f \wedge m0 f')$
*
$$\text{Fract } (f \wedge (m - m0 f') * (p0 f' * f')) 1$$
by (*simp add: ac-simps*)
also from $\langle f \neq 0 \rangle$ **have** $\text{Fract } (f \wedge m0 f') (f \wedge m0 f') = 1$ **by** (*simp add: Fract-same*)
finally show $\text{Fract } (f \wedge m * (p0 f' * f')) (f \wedge m0 f') = \text{Fract } (f \wedge (m - m0 f') * p0 f' * f') 1$
by (*simp add: ac-simps*)
qed
also from *fin''* **have** $\dots = \text{Fract } (\sum f' \in F''. (f \wedge (m - m0 f') * p0 f') * f') 1$


```

    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 $?f \in \sqrt{\text{ideal } ?F}$
then obtain m **where** $?f \wedge m \in \text{ideal } ?F$ **by** (rule radicalE)
moreover have $?f \wedge m \in P[- \{None\}]$
by (rule Polys-closed-power) (auto intro!: PolysI-alt simp: indets-extend-indets)
ultimately have $\text{extend-indets } (f \wedge m) \in \text{extend-indets ' ideal } F$
by (simp add: extend-indets-ideal extend-indets-power)
hence $f \wedge m \in \text{ideal } F$ **by** (simp only: inj-image-mem-iff[OF inj-extend-indets])
thus $f \in \sqrt{\text{ideal } F}$ **by** (rule radicalI)
qed
finally show $?thesis$.
qed

theorem Nullstellensatz:

assumes *finite* X **and** $F \subseteq P[X]$
and $(f::(-::\{\text{countable}, \text{linorder}\} \Rightarrow_0 \text{nat}) \Rightarrow_0 -::\text{alg-closed-field}) \in \mathcal{I} (\mathcal{V} F)$
shows $f \in \sqrt{\text{ideal } F}$
using *assms*(1, 2)
proof (rule radical-idealI-extend-indets)
let $?f = \text{punit.monom-mult } 1$ (*monomial 1 None*) (*extend-indets f*)
show $\mathcal{V} (\text{insert } (1 - ?f) (\text{extend-indets ' } F)) = \{\}$
proof (*intro subset-antisym subsetI*)
fix a
assume $a \in \mathcal{V} (\text{insert } (1 - ?f) (\text{extend-indets ' } F))$
moreover have $1 - ?f \in \text{insert } (1 - ?f) (\text{extend-indets ' } F)$ **by** *simp*
ultimately have $\text{poly-eval } a (1 - ?f) = 0$ **by** (rule variety-ofD)
hence $\text{poly-eval } a (\text{extend-indets } f) \neq 0$
by (*auto simp: poly-eval-minus poly-eval-times simp flip: times-monomial-left*)
hence $\text{poly-eval } (a \circ \text{Some}) f \neq 0$ **by** (simp add: poly-eval-extend-indets)
have $a \circ \text{Some} \in \mathcal{V} F$
proof (rule variety-ofI)
fix f'
assume $f' \in F$
hence $\text{extend-indets } f' \in \text{insert } (1 - ?f) (\text{extend-indets ' } F)$ **by** *simp*
with $\langle a \in - \rangle$ **have** $\text{poly-eval } a (\text{extend-indets } f') = 0$ **by** (rule variety-ofD)
thus $\text{poly-eval } (a \circ \text{Some}) f' = 0$ **by** (simp only: poly-eval-extend-indets)
qed
with *assms*(3) **have** $\text{poly-eval } (a \circ \text{Some}) f = 0$ **by** (rule ideal-ofD)
with $\langle \text{poly-eval } (a \circ \text{Some}) f \neq 0 \rangle$ **show** $a \in \{\}$..
qed *simp*
qed

theorem strong-Nullstellensatz:

assumes *finite* X **and** $F \subseteq P[X]$
shows $\mathcal{I} (\mathcal{V} F) = \sqrt{\text{ideal } (F::(-::\{\text{countable}, \text{linorder}\} \Rightarrow_0 \text{nat}) \Rightarrow_0 -::\text{alg-closed-field}) \text{ set}}$
proof (*intro subset-antisym subsetI*)
fix f
assume $f \in \mathcal{I} (\mathcal{V} F)$
with *assms* **show** $f \in \sqrt{\text{ideal } F}$ **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 $\langle a \in \cdot \rangle$ **have** $a \in \mathcal{V} (ideal (insert (1 - ?f) F))$ **by** (*simp only: variety-of-ideal*)

moreover from $\langle f^m \in ideal F \rangle$ *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 $\langle poly-eval a (f^m) \neq 0 \rangle$ **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** $\langle 1 \in \cdot \rangle$ **by** (*rule variety-ofD*)

```

    thus a ∈ {} by simp
  qed simp
  with assms show f ∈ √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 ⊆ P[X] and f ∈ P[X] and x ∉ X
  and V (insert (1 - punit.monom-mult 1 (Poly-Mapping.single x 1) f) F) = {}
  shows (f :: ('x ⇒0 nat) ⇒0 'a :: alg-closed-field) ∈ √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 ⊆ Y by (auto simp: Y-def)
  hence P[X] ⊆ P[Y] by (rule Polys-mono)
  with assms(2, 3) have F-sub: F ⊆ P[Y] and f ∈ 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** $le-y'$::($'y \times 'y$) *set* **where** fld : *Field* $le-y' = UNIV$
and wo : *Well-order* $le-y'$ **by** *meson*
define $le-y$ **where** $le-y = (\lambda a b::'y. (a, b) \in le-y')$

from $\langle f \in P[Y] \rangle$ **have** 0 : *map-indets rep (map-indets abs f) = f* **unfolding**
map-indets-map-indets
by (*intro map-indets-id*) (*auto intro!*: $y.Abs-inverse$ *dest: PolysD*)
have 1 : *map-indets (rep \circ abs) ' F = F*
proof
from $F-sub$ **show** *map-indets (rep \circ abs) ' F \subseteq F*
by (*smt PolysD(2) comp-apply image-subset-iff map-indets-id subsetD*
 $y.Abs-inverse$)
next
from $F-sub$ **show** $F \subseteq$ *map-indets (rep \circ abs) ' F*
by (*smt 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 $le-y-total$: $le-y\ x\ y \vee le-y\ y\ x$ **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 $\mathcal{V} (\text{insert } (1 - \text{punit.monom-mult } 1) (\text{Poly-Mapping.single } (\text{abs } x) (\text{Suc } 0)))$
 $(\text{map-indets } \text{abs } f) (\text{map-indets } \text{abs ‘ } F) = \{\}$
proof (*intro set-eqI iffI*)
fix *a*
assume $a \in \mathcal{V} (\text{insert } (1 - \text{punit.monom-mult } 1) (\text{Poly-Mapping.single } (\text{abs } x) (\text{Suc } 0)))$
 $(\text{map-indets } \text{abs } f) (\text{map-indets } \text{abs ‘ } F)$
also have $\dots = (\lambda b. b \circ \text{abs}) - ‘ \mathcal{V} (\text{insert } (1 - \text{punit.monom-mult } 1) (\text{Poly-Mapping.single } x) 1) f) F$
by (*simp add: map-indets-minus map-indets-times map-indets-monomial flip: variety-of-map-indets times-monomial-left*)
finally show $a \in \{\}$ **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** $\dots = \sqrt{(\text{ideal } F \cap P[Y]) \cap P[Y]}$
by (*simp add: image-map-indets-ideal image-map-indets-radical image-image map-indets-map-indets 1 y.Rep-range*)
also have $\dots \subseteq \sqrt{\text{ideal } F}$ **using** *radical-mono* **by** *blast*
finally have *?thesis* **by** (*simp only: 0*)
}
note *rl = this* [*cancel-type-definition*]
have $Y \neq \{\}$ **by** (*simp add: Y-def*)
thus *?thesis* **by** (*rule rl*)
qed

corollary *radical-idealI-extend-indets:*
assumes *finite X* **and** $F \subseteq P[X]$
and $\mathcal{V} (\text{insert } (1 - \text{punit.monom-mult } 1) (\text{Poly-Mapping.single } \text{None } 1) (\text{extend-indets } f))$

$(\text{extend-indets } ' F)) = \{\}$

shows $(f::- \Rightarrow_0 -::\text{alg-closed-field}) \in \sqrt{\text{ideal } F}$

proof –

define Y **where** $Y = X \cup \text{indets } f$

from $\text{assms}(1)$ **have** $\text{fin-}Y$: $\text{finite } Y$ **by** $(\text{simp add: } Y\text{-def finite-indets})$

have $P[X] \subseteq P[Y]$ **by** $(\text{rule Polys-mono}) (\text{simp add: } Y\text{-def})$

with $\text{assms}(2)$ **have** $F\text{-sub}$: $F \subseteq P[Y]$ **by** $(\text{rule subset-trans})$

have $f\text{-in}$: $f \in P[Y]$ **by** $(\text{simp add: } Y\text{-def Polys-alt})$

let $?F = \text{extend-indets } ' F$

let $?f = \text{extend-indets } f$

let $?X = \text{Some } ' Y$

from $\text{fin-}Y$ **have** $\text{finite } ?X$ **by** $(\text{rule finite-imageI})$

moreover from $F\text{-sub}$ **have** $?F \subseteq P[?X]$

by $(\text{auto simp: indets-extend-indets intro!: PolysI-alt imageI dest!: PolysD}(2))$

$\text{subsetD}[\text{of } F]$

moreover from $f\text{-in}$ **have** $?f \in P[?X]$

by $(\text{auto simp: indets-extend-indets intro!: PolysI-alt imageI dest!: PolysD}(2))$

moreover have $\text{None} \notin ?X$ **by** simp

ultimately have $?f \in \sqrt{\text{ideal } ?F}$ **using** $\text{assms}(3)$ **by** $(\text{rule radical-idealI})$

also have $?f \in \sqrt{\text{ideal } ?F} \longleftrightarrow f \in \sqrt{\text{ideal } F}$

proof

assume $f \in \sqrt{\text{ideal } F}$

then obtain m **where** $f \wedge m \in \text{ideal } F$ **by** (rule radicalE)

hence $\text{extend-indets } (f \wedge m) \in \text{extend-indets } ' \text{ideal } F$ **by** (rule imageI)

with $\text{extend-indets-ideal-subset}$ **have** $?f \wedge m \in \text{ideal } ?F$ **unfolding** $\text{extend-indets-power}$

..

thus $?f \in \sqrt{\text{ideal } ?F}$ **by** (rule radicalI)

next

assume $?f \in \sqrt{\text{ideal } ?F}$

then obtain m **where** $?f \wedge m \in \text{ideal } ?F$ **by** (rule radicalE)

moreover have $?f \wedge m \in P[- \{ \text{None} \}]$

by $(\text{rule Polys-closed-power}) (\text{auto intro!: PolysI-alt simp: indets-extend-indets})$

ultimately have $\text{extend-indets } (f \wedge m) \in \text{extend-indets } ' \text{ideal } F$

by $(\text{simp add: extend-indets-ideal extend-indets-power})$

hence $f \wedge m \in \text{ideal } F$ **by** $(\text{simp only: inj-image-mem-iff}[OF \text{ inj-extend-indets}])$

thus $f \in \sqrt{\text{ideal } F}$ **by** (rule radicalI)

qed

finally show $?thesis$.

qed

theorem Nullstellensatz:

assumes $\text{finite } X$ **and** $F \subseteq P[X]$

and $(f::- \Rightarrow_0 -::\text{alg-closed-field}) \in \mathcal{I} (\mathcal{V } F)$

shows $f \in \sqrt{\text{ideal } F}$

using $\text{assms}(1, 2)$

proof $(\text{rule radical-idealI-extend-indets})$

let $?f = \text{punit.monom-mult } 1 (\text{monomial } 1 \text{ None}) (\text{extend-indets } f)$

show $\mathcal{V} (\text{insert } (1 - ?f) (\text{extend-indets } ' F)) = \{\}$

proof (*intro subset-antisym subsetI*)
fix a
assume $a \in \mathcal{V}$ (*insert (1 - ?f) (extend-indets ' F)*)
moreover have $1 - ?f \in \text{insert } (1 - ?f) \text{ (extend-indets ' F)}$ **by** *simp*
ultimately have $\text{poly-eval } a \text{ (1 - ?f) = 0}$ **by** (*rule variety-ofD*)
hence $\text{poly-eval } a \text{ (extend-indets f) } \neq 0$
by (*auto simp: poly-eval-minus poly-eval-times simp flip: times-monomial-left*)
hence $\text{poly-eval } (a \circ \text{Some}) f \neq 0$ **by** (*simp add: poly-eval-extend-indets*)
have $a \circ \text{Some} \in \mathcal{V} F$
proof (*rule variety-ofI*)
fix f'
assume $f' \in F$
hence $\text{extend-indets } f' \in \text{insert } (1 - ?f) \text{ (extend-indets ' F)}$ **by** *simp*
with $\langle a \in \cdot \rangle$ **have** $\text{poly-eval } a \text{ (extend-indets } f') = 0$ **by** (*rule variety-ofD*)
thus $\text{poly-eval } (a \circ \text{Some}) f' = 0$ **by** (*simp only: poly-eval-extend-indets*)
qed
with *assms(3)* **have** $\text{poly-eval } (a \circ \text{Some}) f = 0$ **by** (*rule ideal-ofD*)
with $\langle \text{poly-eval } (a \circ \text{Some}) f \neq 0 \rangle$ **show** $a \in \{\}$..
qed *simp*
qed

theorem *strong-Nullstellensatz*:
assumes *finite X and* $F \subseteq P[X]$
shows $\mathcal{I}(\mathcal{V} F) = \sqrt{\text{ideal } (F::(- \Rightarrow_0 \text{alg-closed-field}) \text{ set})}$
proof (*intro subset-antisym subsetI*)
fix f
assume $f \in \mathcal{I}(\mathcal{V} F)$
with *assms* **show** $f \in \sqrt{\text{ideal } F}$ **by** (*rule Nullstellensatz*)
qed (*metis ideal-ofI variety-ofD variety-of-radical-ideal*)

theorem *weak-Nullstellensatz*:
assumes *finite X and* $F \subseteq P[X]$ **and** $\mathcal{V} F = (\{\}::(- \Rightarrow \text{alg-closed-field}) \text{ set})$
shows *ideal F = UNIV*
proof –
from *assms(1, 2)* **have** $\mathcal{I}(\mathcal{V} F) = \sqrt{\text{ideal } F}$ **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* $F \subseteq P[X]$ **and** $f \in P[X]$ **and** $x \notin X$
shows $(f::(- \Rightarrow_0 \text{alg-closed-field}) \in \sqrt{\text{ideal } F} \iff$
 $1 \in \text{ideal } (\text{insert } (1 - \text{punit.monom-mult } 1 \text{ (Poly-Mapping.single } x \ 1))$
 $f) F)$
proof –
let $?f = \text{punit.monom-mult } 1 \text{ (Poly-Mapping.single } x \ 1) f$
show *?thesis*
proof
assume $f \in \sqrt{\text{ideal } F}$
then obtain m **where** $f \wedge m \in \text{ideal } F$ **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* $a\ (f \wedge m) \neq 0$
by (*auto simp: poly-eval-minus poly-eval-times poly-eval-power simp flip: times-monomial-left*)
from $\langle a \in \cdot \rangle$ **have** $a \in \mathcal{V}(ideal\ (insert\ (1 - ?f)\ F))$ **by** (*simp only: variety-of-ideal*)
moreover **from** $\langle f \wedge m \in ideal\ F \rangle$ *ideal.span-mono* **have** $f \wedge m \in ideal\ (insert\ (1 - ?f)\ F)$
by (*rule rev-subsetD*) *blast*
ultimately **have** *poly-eval* $a\ (f \wedge m) = 0$ **by** (*rule variety-ofD*)
with $\langle poly-eval\ a\ (f \wedge m) \neq 0 \rangle$ **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** $\langle 1 \in \cdot \rangle$ **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

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 \Rightarrow (*'x* \Rightarrow_0 nat) \Rightarrow_0 '*a*::comm-ring-1) set \Rightarrow bool

where *generates-max-ideal* X F \longleftrightarrow (*ideal* F \neq UNIV \wedge

$(\forall F'. F' \subseteq P[X] \longrightarrow \text{ideal } F \subseteq \text{ideal } F' \longrightarrow \text{ideal } F' = \text{UNIV}))$

lemma *generates-max-idealI*:

assumes *ideal* $F \neq \text{UNIV}$ **and** $\bigwedge F'. F' \subseteq P[X] \implies \text{ideal } F \subseteq \text{ideal } F' \implies \text{ideal } F' = \text{UNIV}$

shows *generates-max-ideal* $X F$

using *assms* **by** (*simp add: generates-max-ideal-def*)

lemma *generates-max-idealI-alt*:

assumes *ideal* $F \neq \text{UNIV}$ **and** $\bigwedge p. p \in P[X] \implies p \notin \text{ideal } F \implies 1 \in \text{ideal } (insert\ p\ F)$

shows *generates-max-ideal* $X F$

using *assms*(1)

proof (*rule generates-max-idealI*)

fix F'

assume $F' \subseteq P[X]$ **and** *sub*: *ideal* $F \subseteq \text{ideal } F'$

from *this*(2) *ideal.span-subset-spanI* **have** $\neg F' \subseteq \text{ideal } F$ **by** *blast*

then obtain p **where** $p \in F'$ **and** $p \notin \text{ideal } F$ **by** *blast*

from *this*(1) $\langle F' \subseteq P[X] \rangle$ **have** $p \in P[X]$ **..**

hence $1 \in \text{ideal } (insert\ p\ F)$ **using** $\langle p \notin \rightarrow \rangle$ **by** (*rule assms*(2))

also have $\dots \subseteq \text{ideal } (F' \cup F)$ **by** (*rule ideal.span-mono*) (*simp add: \langle p \in F' \rangle*)

also have $\dots = \text{ideal } (\text{ideal } F' \cup \text{ideal } F)$ **by** (*simp add: ideal.span-Un ideal.span-span*)

also from *sub* **have** *ideal* $F' \cup \text{ideal } F = \text{ideal } F'$ **by** *blast*

finally show *ideal* $F' = \text{UNIV}$ **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* $F \neq \text{UNIV}$ **and** $F' \subseteq P[X] \implies \text{ideal } F \subseteq \text{ideal } F' \implies \text{ideal } F' = \text{UNIV}$

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* $F \subseteq \text{ideal } F'$

obtains *ideal* $F = \text{ideal } F' \mid \text{ideal } F' = \text{UNIV}$

using *assms* **by** (*auto simp: generates-max-ideal-def*)

lemma *max-ideal-UNIV-radical*:

assumes *generates-max-ideal* $\text{UNIV } F$

shows $\sqrt{\text{ideal } F} = \text{ideal } F$

proof (*rule ccontr*)

assume $\sqrt{\text{ideal } F} \neq \text{ideal } F$

with *radical-superset* **have** *ideal* $F \subseteq \sqrt{\text{ideal } F}$ **by** *blast*

also have $\dots = \text{ideal } (\sqrt{\text{ideal } F})$ **by** *simp*

finally have *ideal* $F \subseteq \text{ideal } (\sqrt{\text{ideal } F})$.

with *assms* - **have** *ideal* $(\sqrt{\text{ideal } F}) = \text{UNIV}$ **by** (*rule generates-max-idealD*)
simp

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

lemma *max-ideal-shape-aux*:

$(\lambda x. \text{monomial } 1 (\text{Poly-Mapping.single } x \ 1) - \text{monomial } (a \ x) \ 0) \ ' X \subseteq P[X]$
by (*auto intro!: Polys-closed-minus Polys-closed-monomial PPs-closed-single zero-in-PPs*)

lemma *max-ideal-shapeI*:

$\text{generates-max-ideal } X ((\lambda x. \text{monomial } (1::'a::\text{field}) (\text{Poly-Mapping.single } x \ 1) - \text{monomial } (a \ x) \ 0) \ ' X)$

(**is** *generates-max-ideal* $X \ ?F$)

proof (*rule generates-max-idealI-alt*)

show $\text{ideal } ?F \neq \text{UNIV}$

proof

assume $\text{ideal } ?F = \text{UNIV}$

hence $\mathcal{V} (\text{ideal } ?F) = \mathcal{V} \text{UNIV}$ **by** (*rule arg-cong*)

hence $\mathcal{V} ?F = \{\}$ **by** *simp*

moreover have $a \in \mathcal{V} ?F$ **by** (*rule variety-ofI*) (*auto simp: poly-eval-minus poly-eval-monomial*)

ultimately show *False* **by** *simp*

qed

next

fix p

assume $p \in P[X]$ **and** $p \notin \text{ideal } ?F$

have $p \in \text{ideal } (\text{insert } p \ ?F)$ **by** (*rule ideal.span-base*) *simp*

let $?f = \lambda x. \text{monomial } (1::'a) (\text{Poly-Mapping.single } x \ 1) - \text{monomial } (a \ x) \ 0$

let $?g = \lambda x. \text{monomial } (1::'a) (\text{Poly-Mapping.single } x \ 1) + \text{monomial } (a \ x) \ 0$

define q **where** $q = \text{poly-subst } ?g \ p$

have $p = \text{poly-subst } ?f \ q$ **unfolding** $q\text{-def}$ *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 $\dots = (\sum_{t \in \text{keys } q} \text{punit.monom-mult } (\text{lookup } q \ t) \ 0 \ (\text{subst-pp } ?f \ t))$

by (*fact poly-subst-def*)

also have $\dots = \text{punit.monom-mult } (\text{lookup } q \ 0) \ 0 \ (\text{subst-pp } ?f \ 0) +$

$(\sum_{t \in \text{keys } q - \{0\}} \text{monomial } (\text{lookup } q \ t) \ 0 \ * \ \text{subst-pp } ?f \ t)$

(**is** $- = - + ?r$)

by (*cases* $0 \in \text{keys } q$) (*simp-all add: sum.remove in-keys-iff flip: times-monomial-left*)

also have $\dots = \text{monomial } (\text{lookup } q \ 0) \ 0 + ?r$ **by** (*simp flip: times-monomial-left*)

finally have $\text{eq: } p - ?r = \text{monomial } (\text{lookup } q \ 0) \ 0$ **by** *simp*

have $?r \in \text{ideal } ?F$

proof (*intro ideal.span-sum ideal.span-scale*)

fix t

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

have $\text{subst-pp } ?f \ t = (\prod_{y \in \text{keys } t} ?f \ y \wedge \text{lookup } t \ y)$ **by** *(fact subst-pp-def)*
also from $\langle x \in \text{keys } t \rangle$ **have** $\dots = ((\prod_{y \in \text{keys } t - \{x\}} ?f \ y \wedge \text{lookup } t \ y) * ?f$
 $x \wedge (\text{lookup } t \ x - 1)) * ?f \ x$
by *(simp add: prod.remove mult.commute eq)*
also from $\langle x \in X \rangle$ **have** $\dots \in \text{ideal } ?F$ **by** *(intro ideal.span-scale ideal.span-base imageI)*
finally show $\text{subst-pp } ?f \ t \in \text{ideal } ?F$.
qed
also have $\dots \subseteq \text{ideal } (\text{insert } p \ ?F)$ **by** *(rule ideal.span-mono) blast*
finally have $?r \in \text{ideal } (\text{insert } p \ ?F)$.
with $\langle p \in \text{ideal } \rightarrow \rangle$ **have** $p - ?r \in \text{ideal } (\text{insert } p \ ?F)$ **by** *(rule ideal.span-diff)*
hence $\text{monomial } (\text{lookup } q \ 0) \ 0 \in \text{ideal } (\text{insert } p \ ?F)$ **by** *(simp only: eq)*
hence $\text{monomial } (\text{inverse } (\text{lookup } q \ 0)) \ 0 * \text{monomial } (\text{lookup } q \ 0) \ 0 \in \text{ideal}$
 $(\text{insert } p \ ?F)$
by *(rule ideal.span-scale)*
hence $\text{monomial } (\text{inverse } (\text{lookup } q \ 0)) * \text{lookup } q \ 0) \ 0 \in \text{ideal } (\text{insert } p \ ?F)$
by *(simp add: times-monomial-monomial)*
moreover have $\text{lookup } q \ 0 \neq 0$
proof
assume $\text{lookup } q \ 0 = 0$
with $eq \ \langle ?r \in \text{ideal } ?F \rangle$ **have** $p \in \text{ideal } ?F$ **by** *simp*
with $\langle p \notin \text{ideal } ?F \rangle$ **show** *False ..*
qed
ultimately show $1 \in \text{ideal } (\text{insert } p \ ?F)$ **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* $(F::('x::\text{finite} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a::\text{alg-closed-field})$
set)

obtains a **where** $\text{ideal } F = \text{ideal } (\text{range } (\lambda x. \text{monomial } 1 \ (\text{Poly-Mapping.single$

```

x 1) - monomial (a x 0))
proof -
  have fin: finite (UNIV::'x set) by simp
  have ( $\bigcap a \in \mathcal{V} F$ . ideal (range ( $\lambda x$ . monomial 1 (Poly-Mapping.single x 1) -
monomial (a x 0)))) =  $\mathcal{I} (\mathcal{V} F)$ 
    (is ?A = -)
  proof (intro set-eqI iffI ideal-ofI INT-I)
    fix p a
    assume p  $\in$  ?A and a  $\in \mathcal{V} F$ 
    hence p  $\in$  ideal (range ( $\lambda x$ . monomial 1 (Poly-Mapping.single x 1) - monomial
(a x 0)))
      (is -  $\in$  ideal ?B) ..
    have a  $\in \mathcal{V} ?B$ 
    proof (rule variety-ofI)
      fix f
      assume f  $\in ?B$ 
      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  $\in \mathcal{V}$  (ideal ?B) by (simp only: variety-of-ideal)
    thus poly-eval a p = 0 using  $\langle p \in \text{ideal} \rightarrow \rangle$  by (rule variety-ofD)
  next
    fix p a
    assume p  $\in \mathcal{I} (\mathcal{V} F)$  and a  $\in \mathcal{V} F$ 
    hence eq: poly-eval a p = 0 by (rule ideal-ofD)
    have p  $\in \sqrt{\text{ideal}}$  (range ( $\lambda x$ . monomial 1 (monomial 1 x) - monomial (a x)
0)) (is -  $\in \sqrt{\text{ideal}}$  ?B)
      using fin max-ideal-shape-aux
    proof (rule Nullstellensatz)
      show p  $\in \mathcal{I} (\mathcal{V} ?B)$ 
      proof (rule ideal-ofI)
        fix a0
        assume a0  $\in \mathcal{V} ?B$ 
        have a0 = a
        proof
          fix x
          have monomial 1 (monomial 1 x) - monomial (a x) 0  $\in ?B$  by (rule
rangeI)
          with  $\langle a0 \in \rightarrow \rangle$  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 ( $\lambda x$ . monomial 1 (monomial 1 x) - monomial (a
x) 0))

```

```

    using max-ideal-shapeI by (rule max-ideal-UNIV-radical)
    finally show  $p \in \text{ideal } (\text{range } (\lambda x. \text{monomial } 1 (\text{monomial } 1 x) - \text{monomial } (a x) 0))$  .
  qed
  also from fin have  $\dots = \sqrt{\text{ideal } F}$  by (rule strong-Nullstellensatz) simp
  also from assms have  $\dots = \text{ideal } F$  by (rule max-ideal-UNIV-radical)
  finally have eq:  $?A = \text{ideal } F$  .
  also from assms have  $\dots \neq \text{UNIV}$  by (rule generates-max-idealD)
  finally obtain a where  $a \in \mathcal{V} F$ 
  and  $\text{ideal } (\text{range } (\lambda x. \text{monomial } 1 (\text{Poly-Mapping.single } x (1::\text{nat})) - \text{monomial } (a x) 0)) \neq \text{UNIV}$ 
  (is  $?B \neq -$ ) by auto
  from  $\langle a \in \mathcal{V} F \rangle$  have  $\text{ideal } F \subseteq ?B$  by (auto simp flip: eq)
  with assms max-ideal-shape-aux show ?thesis
  proof (rule generates-max-ideal-cases)
    assume  $\text{ideal } F = ?B$ 
    thus ?thesis ..
  next
    assume  $?B = \text{UNIV}$ 
    with  $\langle ?B \neq \text{UNIV} \rangle$  show ?thesis ..
  qed
qed

```

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

```

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

```

```

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

```

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

```

  assume  $\exists (Rep :: 'y \Rightarrow 'x)$  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  $\circ$  abs) 'A = A if  $A \subseteq P[X]$  for  $A :: (- \Rightarrow_0 'a)$  set
  proof
    from that show map-indets (rep  $\circ$  abs) 'A  $\subseteq$  A
      by (smt PolysD(2) comp-apply image-subset-iff map-indets-id subsetD
  y.Abs-inverse)
    next
      from that show  $A \subseteq$  map-indets (rep  $\circ$  abs) 'A
        by (smt 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  $1 \in$  ideal (map-indets abs 'F) by simp
    hence map-indets rep  $1 \in$  map-indets rep 'ideal (map-indets abs 'F) by
  (rule imageI)
    also from map-indets-plus map-indets-times have  $\dots \subseteq$  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  $1 \in$  ideal F by simp
  moreover from assms(3) have ideal F  $\neq$  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)  $\subset$  ideal F'
  with inj-on-subset have map-indets rep 'ideal (map-indets abs 'F)  $\subset$ 

```

map-ideal F'
by (*rule image-strict-mono*) (*fact 3*, *fact subset-UNIV*)
hence $sub: ideal\ F \cap P[X] \subseteq ideal\ (map-ideal\ rep\ 'F') \cap P[X]$ **using** 2
assms(2)
by (*simp add: image-map-ideal image-image map-ideal-map-ideal*
1 *y.Rep-range*)
have $ideal\ F \subseteq ideal\ (map-ideal\ rep\ 'F')$
proof (*intro psubsetI notI ideal.span-subset-spanI subsetI*)
fix p
assume $p \in F$
with *assms*(2) *ideal.span-base sub* **show** $p \in ideal\ (map-ideal\ rep\ 'F')$ **by**
blast
next
assume $ideal\ F = ideal\ (map-ideal\ rep\ 'F')$
with sub **show** *False* **by** *simp*
qed
with *assms*(3) - **have** $ideal\ (map-ideal\ rep\ 'F') = UNIV$
proof (*rule generates-max-idealD*)
from *subset-UNIV* **have** $map-ideal\ rep\ 'F' \subseteq range\ (map-ideal\ rep)$ **by**
(*rule image-mono*)
also **have** $\dots = P[X]$ **by** (*simp only: range-map-ideal y.Rep-range*)
finally **show** $map-ideal\ rep\ 'F' \subseteq P[X]$.
qed
hence $P[range\ rep] = ideal\ (map-ideal\ rep\ 'F') \cap P[range\ rep]$ **by** *simp*
also **from** 2 **have** $\dots = map-ideal\ rep\ 'ideal\ F'$ **by** (*simp only: im-*
age-map-ideal)
finally **have** $map-ideal\ rep\ 'ideal\ F' = range\ (map-ideal\ rep)$
by (*simp only: range-map-ideal*)
with 3 **show** $ideal\ F' = UNIV$ **by** (*metis inj-image-eq-iff*)
qed
ultimately obtain a
where $*$: $ideal\ (map-ideal\ abs\ 'F) =$
 $ideal\ (range\ (\lambda x. monomial\ 1\ (Poly-Mapping.single\ x\ (Suc\ 0))) -$
 $monomial\ (a\ x)\ 0))$
(*is - = ?A*)
by (*rule max-ideal-shapeD-internalized*[**where** $'x='y$, *untransferred*, *simpli-*
fied])
hence $map-ideal\ rep\ 'ideal\ (map-ideal\ abs\ 'F) = map-ideal\ rep\ ' ?A$ **by**
simp
with 2 *assms*(2) **have** $ideal\ F \cap P[X] =$
 $ideal\ (range\ (\lambda x. monomial\ 1\ (Poly-Mapping.single\ (rep\ x)\ 1) - monomial$
 $(a\ x)\ 0)) \cap P[X]$
(*is - = ideal ?B \cap -*)
by (*simp add: image-map-ideal y.Rep-range image-image map-ideal-map-ideal*
map-ideal-minus map-ideal-monomial 1)
also **have** $?B = (\lambda x. monomial\ 1\ (Poly-Mapping.single\ x\ 1) - monomial\ ((a$
 $\circ abs)\ x)\ 0)$ $'X$
(*is - = ?C*)
proof


```

    show ?B ⊆ ?C by (smt 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 ((λx. monomial 1 (monomial 1 x) - monomial (a
x) 0) ‘ X)
    by (rule max-ideal-shapeD)
    let ?p = λx. 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 $?p x \in \text{ideal } F \cap P[\{x\}] \dots$
also assume $\dots \subseteq \{0\}$
finally show *False*
by (*metis diff-eq-diff-eq diff-self monomial-0D monomial-inj one-neq-zero sin-
gletonD*)
qed
end

References

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