

The Hermite–Lindemann–Weierstraß Transcendence Theorem

Manuel Eberl

March 12, 2021

Abstract

This article provides a formalisation of the Hermite–Lindemann–Weierstraß Theorem (also known as simply Hermite–Lindemann or Lindemann–Weierstraß). This theorem is one of the crowning achievements of 19th century number theory.

The theorem states that if $\alpha_1, \dots, \alpha_n \in \mathbb{C}$ are algebraic numbers that are linearly independent over \mathbb{Z} , then $e^{\alpha_1}, \dots, e^{\alpha_n}$ are algebraically independent over \mathbb{Q} .

Like the previous formalisation in Coq by Bernard [2], I proceeded by formalising Baker’s alternative formulation of the theorem [1] and then deriving the original one from that. Baker’s version states that for any algebraic numbers $\beta_1, \dots, \beta_n \in \mathbb{C}$ and distinct algebraic numbers $\alpha_1, \dots, \alpha_n \in \mathbb{C}$, we have:

$$\beta_1 e^{\alpha_1} + \dots + \beta_n e^{\alpha_n} = 0 \quad \text{iff} \quad \forall i. \beta_i = 0$$

This has a number of immediate corollaries, e.g.:

- e and π are transcendental
- e^z , $\sin z$, $\tan z$, etc. are transcendental for algebraic $z \in \mathbb{C} \setminus \{0\}$
- $\ln z$ is transcendental for algebraic $z \in \mathbb{C} \setminus \{0, 1\}$

Contents

1	Divisibility of algebraic integers	3
2	Auxiliary facts about univariate polynomials	7
3	The minimal polynomial of an algebraic number	18
4	The lexicographic ordering on complex numbers	22
5	Additional facts about multivariate polynomials	23
5.1	Miscellaneous	24
5.2	Converting a univariate polynomial into a multivariate one	24
6	More facts about algebraic numbers	28
6.1	Miscellaneous	28
6.2	Turning an algebraic number into an algebraic integer	31
6.3	Multiplying an algebraic number with a suitable integer turns it into an algebraic integer.	31
7	Miscellaneous facts	33
8	The Hermite–Lindemann–Weierstraß Transcendence Theorem	38
8.1	Main proof	38
8.2	Removing the restriction of full sets of conjugates	61
8.3	Removing the restriction to integer coefficients	73
8.4	The final theorem	83
8.5	The traditional formulation of the theorem	85
8.6	Simple corollaries	87
8.7	Transcendence of the trigonometric and hyperbolic functions	88

1 Divisibility of algebraic integers

```
theory Algebraic-Integer-Divisibility
imports Algebraic-Numbers.Algebraic-Numbers
begin
```

In this section, we define a notion of divisibility of algebraic integers: y is divisible by x if y / x is an algebraic integer (or if x and y are both zero).

Technically, the definition does not require x and y to be algebraic integers themselves, but we will always use it that way (in fact, in our case x will always be a rational integer).

```
definition alg-dvd :: 'a :: field  $\Rightarrow$  'a  $\Rightarrow$  bool (infix alg'-dvd 50) where
  x alg-dvd y  $\longleftrightarrow$  (x = 0  $\longrightarrow$  y = 0)  $\wedge$  algebraic-int (y / x)
```

```
lemma alg-dvd-imp-algebraic-int:
  fixes x y :: 'a :: field-char-0
  shows x alg-dvd y  $\implies$  algebraic-int x  $\implies$  algebraic-int y
  using algebraic-int-times[of y / x x] by (auto simp: alg-dvd-def)
```

```
lemma alg-dvd-0-left-iff [simp]: 0 alg-dvd x  $\longleftrightarrow$  x = 0
  by (auto simp: alg-dvd-def)
```

```
lemma alg-dvd-0-right [iff]: x alg-dvd 0
  by (auto simp: alg-dvd-def)
```

```
lemma one-alg-dvd-iff [simp]: 1 alg-dvd x  $\longleftrightarrow$  algebraic-int x
  by (auto simp: alg-dvd-def)
```

```
lemma alg-dvd-of-int [intro]:
  assumes x dvd y
  shows of-int x alg-dvd of-int y
proof (cases of-int x = (0 :: 'a))
  case False
  from assms obtain z where z: y = x * z
  by (elim dvdE)
  have algebraic-int (of-int z)
  by auto
  also have of-int z = of-int y / (of-int x :: 'a)
  using False by (simp add: z field-simps)
  finally show ?thesis
  using False by (simp add: alg-dvd-def)
qed (use assms in (auto simp: alg-dvd-def))
```

```
lemma alg-dvd-of-nat [intro]:
  assumes x dvd y
  shows of-nat x alg-dvd of-nat y
  using alg-dvd-of-int[of int x int y] assms by simp
```

```
lemma alg-dvd-of-int-iff [simp]:
```

$(\text{of-int } x :: 'a :: \text{field-char-0}) \text{ alg-dvd of-int } y \longleftrightarrow x \text{ dvd } y$
proof
assume $(\text{of-int } x :: 'a) \text{ alg-dvd of-int } y$
hence $\text{of-int } y / (\text{of-int } x :: 'a) \in \mathbf{Z}$ **and** $\text{nz}: \text{of-int } x = (0::'a) \longrightarrow \text{of-int } y = (0::'a)$
by $(\text{auto simp: alg-dvd-def dest!: rational-algebraic-int-is-int})$
then obtain n **where** $\text{of-int } y / \text{of-int } x = (\text{of-int } n :: 'a)$
by (elim Ints-cases)
hence $\text{of-int } y = (\text{of-int } (x * n) :: 'a)$
unfolding of-int-mult **using** nz **by** $(\text{auto simp: field-simps})$
hence $y = x * n$
by $(\text{subst (asm) of-int-eq-iff})$
thus $x \text{ dvd } y$
by auto
qed blast

lemma $\text{alg-dvd-of-nat-iff}$ $[\text{simp}]$:
 $(\text{of-nat } x :: 'a :: \text{field-char-0}) \text{ alg-dvd of-nat } y \longleftrightarrow x \text{ dvd } y$
proof $-$
have $(\text{of-int } (\text{int } x) :: 'a) \text{ alg-dvd of-int } (\text{int } y) \longleftrightarrow x \text{ dvd } y$
by $(\text{subst alg-dvd-of-int-iff}) \text{ auto}$
thus $?thesis$ **unfolding** of-int-of-nat-eq .
qed

lemma alg-dvd-add $[\text{intro}]$:
fixes $x y z :: 'a :: \text{field-char-0}$
shows $x \text{ alg-dvd } y \Longrightarrow x \text{ alg-dvd } z \Longrightarrow x \text{ alg-dvd } (y + z)$
unfolding alg-dvd-def **by** $(\text{auto simp: add-divide-distrib})$

lemma $\text{alg-dvd-uminus-right}$ $[\text{intro}]$: $x \text{ alg-dvd } y \Longrightarrow x \text{ alg-dvd } -y$
by $(\text{auto simp: alg-dvd-def})$

lemma $\text{alg-dvd-uminus-right-iff}$ $[\text{simp}]$: $x \text{ alg-dvd } -y \longleftrightarrow x \text{ alg-dvd } y$
using $\text{alg-dvd-uminus-right}[of x y]$ $\text{alg-dvd-uminus-right}[of x -y]$ **by** auto

lemma alg-dvd-diff $[\text{intro}]$:
fixes $x y z :: 'a :: \text{field-char-0}$
shows $x \text{ alg-dvd } y \Longrightarrow x \text{ alg-dvd } z \Longrightarrow x \text{ alg-dvd } (y - z)$
unfolding alg-dvd-def **by** $(\text{auto simp: diff-divide-distrib})$

lemma alg-dvd-triv-left $[\text{intro}]$: $\text{algebraic-int } y \Longrightarrow x \text{ alg-dvd } x * y$
by $(\text{auto simp: alg-dvd-def})$

lemma $\text{alg-dvd-triv-right}$ $[\text{intro}]$: $\text{algebraic-int } x \Longrightarrow y \text{ alg-dvd } x * y$
by $(\text{auto simp: alg-dvd-def})$

lemma $\text{alg-dvd-triv-left-iff}$: $x \text{ alg-dvd } x * y \longleftrightarrow x = 0 \vee \text{algebraic-int } y$
by $(\text{auto simp: alg-dvd-def})$

lemma *alg-dvd-triv-right-iff*: $y \text{ alg-dvd } x * y \iff y = 0 \vee \text{algebraic-int } x$
by (*auto simp: alg-dvd-def*)

lemma *alg-dvd-triv-left-iff'* [*simp*]: $x \neq 0 \implies x \text{ alg-dvd } x * y \iff \text{algebraic-int } y$
by (*simp add: alg-dvd-triv-left-iff*)

lemma *alg-dvd-triv-right-iff'* [*simp*]: $y \neq 0 \implies y \text{ alg-dvd } x * y \iff \text{algebraic-int } x$
by (*simp add: alg-dvd-triv-right-iff*)

lemma *alg-dvd-trans* [*trans*]:
fixes $x \ y \ z :: 'a :: \text{field-char-0}$
shows $x \text{ alg-dvd } y \implies y \text{ alg-dvd } z \implies x \text{ alg-dvd } z$
using *algebraic-int-times*[*of y / x z / y*] **by** (*auto simp: alg-dvd-def*)

lemma *alg-dvd-mono* [*simp*]:
fixes $a \ b \ c \ d :: 'a :: \text{field-char-0}$
shows $a \text{ alg-dvd } c \implies b \text{ alg-dvd } d \implies (a * b) \text{ alg-dvd } (c * d)$
using *algebraic-int-times*[*of c / a d / b*] **by** (*auto simp: alg-dvd-def*)

lemma *alg-dvd-mult* [*simp*]:
fixes $a \ b \ c :: 'a :: \text{field-char-0}$
shows $a \text{ alg-dvd } c \implies \text{algebraic-int } b \implies a \text{ alg-dvd } (b * c)$
using *alg-dvd-mono*[*of a c 1 b*] **by** (*auto simp: mult.commute*)

lemma *alg-dvd-mult2* [*simp*]:
fixes $a \ b \ c :: 'a :: \text{field-char-0}$
shows $a \text{ alg-dvd } b \implies \text{algebraic-int } c \implies a \text{ alg-dvd } (b * c)$
using *alg-dvd-mult*[*of a b c*] **by** (*simp add: mult.commute*)

A crucial theorem: if an integer x divides a rational number y , then y is in fact also an integer, and that integer is a multiple of x .

lemma *alg-dvd-int-rat*:
fixes $y :: 'a :: \text{field-char-0}$
assumes *of-int x alg-dvd y* **and** $y \in \mathbb{Q}$
shows $\exists n. y = \text{of-int } n \wedge x \text{ dvd } n$
proof (*cases x = 0*)
case *False*
have $y / \text{of-int } x \in \mathbb{Z}$
by (*intro rational-algebraic-int-is-int*) (*use assms in <auto simp: alg-dvd-def>*)
then obtain n **where** $n: \text{of-int } n = y / (\text{of-int } x :: 'a)$
by (*elim Ints-cases*) *auto*
hence $y = \text{of-int } (n * x)$
using *False* **by** (*simp add: field-simps*)
thus *?thesis* **by** (*intro exI[of - x * n]*) *auto*
qed (*use assms in auto*)

lemma *prod-alg-dvd-prod*:
fixes $f :: 'a \Rightarrow 'b :: \text{field-char-0}$
assumes $\bigwedge x. x \in A \implies f \ x \text{ alg-dvd } g \ x$

shows $\text{prod } f \ A \ \text{alg-dvd } \text{prod } g \ A$
using *assms* **by** (*induction A rule: infinite-finite-induct*) *auto*

lemma *alg-dvd-sum*:

fixes $f :: 'a \Rightarrow 'b :: \text{field-char-0}$
assumes $\bigwedge x. x \in A \Longrightarrow y \ \text{alg-dvd } f \ x$
shows $y \ \text{alg-dvd } \text{sum } f \ A$
using *assms* **by** (*induction A rule: infinite-finite-induct*) *auto*

lemma *not-alg-dvd-sum*:

fixes $f :: 'a \Rightarrow 'b :: \text{field-char-0}$
assumes $\bigwedge x. x \in A - \{x'\} \Longrightarrow y \ \text{alg-dvd } f \ x$
assumes $\neg y \ \text{alg-dvd } f \ x'$
assumes $x' \in A \ \text{finite } A$
shows $\neg y \ \text{alg-dvd } \text{sum } f \ A$

proof

assume $*$: $y \ \text{alg-dvd } \text{sum } f \ A$
have $y \ \text{alg-dvd } \text{sum } f \ A - \text{sum } f \ (A - \{x'\})$
using $\langle x' \in A \rangle$ **by** (*intro alg-dvd-diff[OF * alg-dvd-sum]*) *assms*) *auto*
also have $\dots = \text{sum } f \ (A - (A - \{x'\}))$
using *assms* **by** (*subst sum-diff*) *auto*
also have $A - (A - \{x'\}) = \{x'\}$
using *assms* **by** *auto*
finally show *False* **using** *assms* **by** *simp*

qed

lemma *fact-dvd-pochhammer*:

assumes $m \leq n + 1$
shows $\text{fact } m \ \text{dvd } \text{pochhammer } (\text{int } n - \text{int } m + 1) \ m$

proof –

have $(\text{real } n \ \text{gchoose } m) * \text{fact } m = \text{of-int } (\text{pochhammer } (\text{int } n - \text{int } m + 1) \ m)$
by (*simp add: gbinomial-pochhammer' pochhammer-of-int [symmetric]*)
also have $(\text{real } n \ \text{gchoose } m) * \text{fact } m = \text{of-int } (\text{int } (n \ \text{choose } m) * \text{fact } m)$
by (*simp add: binomial-gbinomial*)
finally have $\text{int } (n \ \text{choose } m) * \text{fact } m = \text{pochhammer } (\text{int } n - \text{int } m + 1) \ m$
by (*subst (asm) of-int-eq-iff*)
from this [*symmetric*] **show** *?thesis* **by** *simp*

qed

lemma *coeff-higher-pderiv*:

$\text{coeff } ((\text{pderiv } \widetilde{m}) \ f) \ n = \text{pochhammer } (\text{of-nat } (\text{Suc } n)) \ m * \text{coeff } f \ (n + m)$
by (*induction m arbitrary: n*) (*simp-all add: coeff-pderiv pochhammer-rec algebra-simps*)

lemma *fact-alg-dvd-poly-higher-pderiv*:

fixes $p :: 'a :: \text{field-char-0}$ *poly*
assumes $\bigwedge i. \text{algebraic-int } (\text{poly.coeff } p \ i) \ \text{algebraic-int } x \ m \leq k$
shows $\text{fact } m \ \text{alg-dvd } \text{poly } ((\text{pderiv } \widetilde{k}) \ p) \ x$
unfolding *poly-altdef*

```

proof (intro alg-dvd-sum, goal-cases)
  case (1 i)
  have (of-int (fact m) :: 'a) alg-dvd (of-int (fact k))
    by (intro alg-dvd-of-int fact-dvd assms)
  also have (of-int (fact k) :: 'a) alg-dvd of-int (pochhammer (int i + 1) k)
    using fact-dvd-pochhammer[of k i + k]
    by (intro alg-dvd-of-int fact-dvd-pochhammer) (auto simp: algebra-simps)
  finally have fact m alg-dvd (pochhammer (of-nat i + 1) k :: 'a)
    by (simp flip: pochhammer-of-int)
  also have ... alg-dvd pochhammer (of-nat i + 1) k * poly.coeff p (i + k)
    by (rule alg-dvd-triv-left) (rule assms)
  also have ... = poly.coeff ((pderiv  $\widehat{k}$ ) p) i
    unfolding coeff-higher-pderiv by (simp add: add-ac flip: pochhammer-of-int)
  also have ... alg-dvd poly.coeff ((pderiv  $\widehat{k}$ ) p) i * x  $\widehat{i}$ 
    by (intro alg-dvd-triv-left algebraic-int-power assms)
  finally show ?case .
qed

end

```

2 Auxiliary facts about univariate polynomials

theory More-Polynomial-HLW

imports

HOL-Computational-Algebra.Computational-Algebra
 Polynomial-Factorization.Gauss-Lemma
 Power-Sum-Polynomials.Power-Sum-Polynomials-Library
 Algebraic-Numbers.Algebraic-Numbers

begin

instance poly :: ({idom-divide, normalization-semidom-multiplicative, factorial-ring-gcd, semiring-gcd-mult-normalize}) factorial-semiring-multiplicative ..

lemma lead-coeff-prod-mset:

fixes A :: 'a :: {comm-semiring-1, semiring-no-zero-divisors} poly multiset
shows Polynomial.lead-coeff (prod-mset A) = prod-mset (image-mset Polynomial.lead-coeff A)
by (induction A) (auto simp: Polynomial.lead-coeff-mult)

lemma content-normalize [simp]:

fixes p :: 'a :: {factorial-semiring, idom-divide, semiring-gcd, normalization-semidom-multiplicative} poly

shows content (normalize p) = content p

proof (cases p = 0)

case [simp]: False

have content p = content (unit-factor p * normalize p)

by simp

also have ... = content (unit-factor p) * content (normalize p)

by (rule content-mult)

also have $\text{content } (\text{unit-factor } p) = 1$
by *(auto simp: unit-factor-poly-def)*
finally show *?thesis* **by** *simp*
qed *auto*

lemma *rat-to-normalized-int-poly-exists*:
fixes $p :: \text{rat poly}$
assumes $p \neq 0$
obtains $q \text{ lc}$ **where** $p = \text{Polynomial.smult } lc \text{ (of-int-poly } q) \text{ lc} > 0 \text{ content } q = 1$
proof –
define lc **where** $lc = \text{fst } (\text{rat-to-normalized-int-poly } p)$
define q **where** $q = \text{snd } (\text{rat-to-normalized-int-poly } p)$
have $eq: \text{rat-to-normalized-int-poly } p = (lc, q)$
by *(simp add: lc-def q-def)*
show *?thesis*
using *rat-to-normalized-int-poly[OF eq] assms*
by *(intro that[of lc q]) auto*
qed

lemma *irreducible-imp-squarefree*:
assumes *irreducible* p
shows *squarefree* p
proof *(rule squarefreeI)*
fix q **assume** $q \wedge^2 \text{ dvd } p$
then obtain r **where** $qr: p = q \wedge^2 * r$
by *(elim dvdE)*
have $q \text{ dvd } 1 \vee q * r \text{ dvd } 1$
by *(intro irreducibleD[OF assms]) (use qr in ‹simp-all add: power2-eq-square-mult-ac›)*
thus $q \text{ dvd } 1$
by *(meson dvd-mult-left)*
qed

lemma *squarefree-imp-rsquarefree*:
fixes $p :: 'a :: \text{idom poly}$
assumes *squarefree* p
shows *rsquarefree* p
unfolding *rsquarefree-def*
proof *(intro conjI allI)*
fix $x :: 'a$
have $\text{Polynomial.order } x \text{ } p < 2$
proof *(rule ccontr)*
assume $\neg(\text{Polynomial.order } x \text{ } p < 2)$
hence $[-x, 1:] \wedge^2 \text{ dvd } p$
by *(subst order-divides) auto*
from *assms* **and** *this* **have** $[-x, 1:] \text{ dvd } 1$
by *(rule squarefreeD)*
hence $\text{Polynomial.degree } [-x, 1:] \leq \text{Polynomial.degree } (1 :: 'a \text{ poly})$
by *(rule dvd-imp-degree-le) auto*


```

    thus False by simp
  qed
  thus Polynomial.order x p = 0  $\vee$  Polynomial.order x p = 1
    by linarith
  qed (use assms in auto)

lemma squarefree-imp-coprime-pderiv:
  fixes p :: 'a :: {factorial-ring-gcd, semiring-gcd-mult-normalize, semiring-char-0}
  poly
  assumes squarefree p and content p = 1
  shows Rings.coprime p (pderiv p)
  proof (rule coprimeI-primes)
    fix d assume d: prime d d dvd p d dvd pderiv p
    show False
  proof (cases Polynomial.degree d = 0)
    case deg: False
    obtain q where dq: p = d * q
      using d by (elim dvdE)
    have ⟨d dvd q * pderiv d⟩
      using d by (simp add: dq pderiv-mult dvd-add-right-iff)
    moreover have  $\neg$ d dvd pderiv d
  proof
    assume d dvd pderiv d
    hence Polynomial.degree d  $\leq$  Polynomial.degree (pderiv d)
      using d deg by (intro dvd-imp-degree-le) (auto simp: pderiv-eq-0-iff)
    hence Polynomial.degree d = 0
      by (subst (asm) degree-pderiv) auto
    thus False using deg by contradiction
  qed
  ultimately have d dvd q
    using d(1) by (simp add: prime-dvd-mult-iff)
  hence d  $\wedge$  2 dvd p
    by (auto simp: dq power2-eq-square)
  from assms(1) and this have is-unit d
    by (rule squarefreeD)
  thus False using ⟨prime d⟩ by auto
next
  case True
  then obtain d' where [simp]: d = [:d':]
    by (elim degree-eq-zeroE)
  from d have d' dvd content p
    by (simp add: const-poly-dvd-iff-dvd-content)
  with assms and prime-imp-prime-elem[OF ⟨prime d⟩] show False
    by (auto simp: prime-elem-const-poly-iff)
  qed
  qed (use assms in auto)

lemma irreducible-imp-coprime-pderiv:
  fixes p :: 'a :: {idom-divide, semiring-char-0} poly

```

```

assumes irreducible p Polynomial.degree p ≠ 0
shows Rings.coprime p (pderiv p)
proof (rule Rings.coprimeI)
  fix d assume d: d dvd p d dvd pderiv p
  obtain q where dq: p = d * q
    using d by (elim dvdE)
  have is-unit d ∨ is-unit q
    using assms dq by (auto simp: irreducible-def)
  thus is-unit d
proof
  assume unit: is-unit q
  with d have p dvd pderiv p
    using algebraic-semidom-class.mult-unit-dvd-iff dq by blast
  hence Polynomial.degree p = 0
    by (meson not-dvd-pderiv)
  with assms(2) show ?thesis by contradiction
qed
qed

```

```

lemma poly-gcd-eq-0I:
  assumes poly p x = 0 poly q x = 0
  shows poly (gcd p q) x = 0
  using assms by (simp add: poly-eq-0-iff-dvd)

```

```

lemma poly-eq-0-coprime:
  assumes Rings.coprime p q p ≠ 0 q ≠ 0
  shows poly p x ≠ 0 ∨ poly q x ≠ 0
proof –
  have False if poly p x = 0 poly q x = 0
proof –
  have [: -x, 1:] dvd p [: -x, 1:] dvd q
    using that by (simp-all add: poly-eq-0-iff-dvd)
  hence [: -x, 1:] dvd 1
    using ⟨Rings.coprime p q⟩ by (meson not-coprimeI)
  thus False
    by (simp add: is-unit-poly-iff)
qed
thus ?thesis
  by blast
qed

```

```

lemma coprime-of-int-polyI:
  assumes Rings.coprime p q
  shows Rings.coprime (of-int-poly p) (of-int-poly q :: 'a :: {field-char-0, field-gcd} poly)
  using assms gcd-of-int-poly[of p q, where ?'a = 'a] unfolding coprime-iff-gcd-eq-1
  by simp

```

```

lemma irreducible-imp-rsquarefree-of-int-poly:

```

```

fixes p :: int poly
assumes irreducible p and Polynomial.degree p > 0
shows rsquarefree (of-int-poly p :: 'a :: {field-gcd, field-char-0} poly)
proof -
{
  fix x :: 'a
  assume x: poly (of-int-poly p) x = 0 poly (pderiv (of-int-poly p)) x = 0
  define d where d = gcd (of-int-poly p) (pderiv (of-int-poly p) :: 'a poly)
  have poly d x = 0
    using x unfolding d-def by (intro poly-gcd-eq-0I) auto
  moreover have d ≠ 0
    using assms by (auto simp: d-def)
  ultimately have 0 < Polynomial.degree d
    by (intro Nat.gr0I) (auto elim!: degree-eq-zeroE)
  also have Polynomial.degree d = Polynomial.degree (gcd p (pderiv p))
    unfolding d-def of-int-hom.map-poly-pderiv[symmetric] gcd-of-int-poly by
simp
  finally have deg: ... > 0 .

  have gcd p (pderiv p) dvd p
    by auto
  from irreducibleD'[OF assms(1) this] and deg have p dvd gcd p (pderiv p)
    by auto
  also have ... dvd pderiv p
    by auto
  finally have Polynomial.degree p = 0
    by auto
  with assms have False by simp
}
thus ?thesis by (auto simp: rsquarefree-roots)
qed

```

```

lemma squarefree-of-int-polyI:
  assumes squarefree p content p = 1
  shows squarefree (of-int-poly p :: 'a :: {field-char-0,field-gcd} poly)
proof -
  have Rings.coprime p (pderiv p)
    by (rule squarefree-imp-coprime-pderiv) fact+
  hence Rings.coprime (of-int-poly p :: 'a poly) (of-int-poly (pderiv p))
    by (rule coprime-of-int-polyI)
  also have of-int-poly (pderiv p) = pderiv (of-int-poly p :: 'a poly)
    by (simp add: of-int-hom.map-poly-pderiv)
  finally show ?thesis
    using coprime-pderiv-imp-squarefree by blast
qed

```

```

lemma higher-pderiv-pcompose-linear:
  (pderiv  $\hat{\hat{}}$  n) (pcompose p [:0, c:]) =
  Polynomial.smult (c  $\hat{\hat{}}$  n) (pcompose ((pderiv  $\hat{\hat{}}$  n) p) [:0, c:])

```

by (*induction n*) (*simp-all add: pderiv-pcompose pderiv-smult pderiv-pCons pcompose-smult mult-ac*)

lemma *poly-poly-eq*:

poly (*poly p* [:x:]) *y* = *poly* (*eval-poly* ($\lambda p. [:poly p y:]$) *p* [:0, 1:] *x*)
by (*induction p*) (*auto simp: eval-poly-def*)

lemma *poly-poly-poly-y-x* [*simp*]:

fixes *p* :: 'a :: *idom poly poly*

shows *poly* (*poly* (*poly-y-x p*) [:y:]) *x* = *poly* (*poly p* [:x:]) *y*

proof (*induction p*)

case (*pCons a p*)

have *poly* (*poly* (*poly-y-x* (*pCons a p*)) [:y:]) *x* =

poly a y + *poly* (*poly* (*map-poly* (*pCons 0*) (*poly-y-x p*)) [:y:]) *x*

by (*simp add: poly-y-x-pCons eval-poly-def*)

also have *pCons 0* = ($\lambda p::'a \text{ poly. Polynomial.monom } 1 \ 1 * p$)

by (*simp add: Polynomial.monom-altdef*)

also have *map-poly ...* (*poly-y-x p*) = *Polynomial.smult* (*Polynomial.monom 1* 1) (*poly-y-x p*)

by (*simp add: smult-conv-map-poly*)

also have *poly ...* [:y:] = *Polynomial.monom 1 1 * poly* (*poly-y-x p*) [:y:]

by *simp*

also have *poly a y* + *poly ... x* = *poly* (*poly* (*pCons a p*) [:x:]) *y*

by (*simp add: pCons poly-monom*)

finally show ?*case* .

qed *auto*

lemma (*in idom-hom*) *map-poly-higher-pderiv* [*hom-distrib*]:

map-poly hom ((*pderiv* $\widehat{\widehat{n}}$) *p*) = (*pderiv* $\widehat{\widehat{n}}$) (*map-poly hom p*)

by (*induction n*) (*simp-all add: map-poly-pderiv*)

lemma *coeff-prod-linear-factors*:

fixes *f* :: 'a \Rightarrow 'b :: *comm-ring-1*

assumes [*intro*]: *finite A*

shows *Polynomial.coeff* ($\prod x \in A. [:-f x, 1:] \wedge e x$) *i* =

$(\sum X \mid X \in \text{Pow } (\text{SIGMA } x:A. \{..\leq e x\}) \wedge i = \text{sum } e A - \text{card } X.$

$(-1) \wedge \text{card } X * (\prod x \in X. f (fst x)))$)

proof –

define *poly-X* **where** *poly-X* = (*Polynomial.monom 1 1* :: 'b *poly*)

have [*simp*]: $(-1) \wedge n = [:(-1) \wedge n :: 'b:]$ **for** *n* :: *nat*

by (*simp flip: pCons-one add: poly-const-pow*)

have ($\prod x \in A. [:-f x, 1:] \wedge e x$) = ($\prod (x,-) \in \text{Sigma } A (\lambda x. \{..\leq e x\}). [:-f x, 1:]$)

by (*subst prod.Sigma [symmetric]*) *auto*

also have ... = ($\prod (x,-) \in \text{Sigma } A (\lambda x. \{..\leq e x\}). \text{poly-X} - [:-f x:]$)

by (*intro prod.cong*) (*auto simp: poly-X-def monom-altdef*)

also have ... = ($\sum X \in \text{Pow } (\text{SIGMA } x:A. \{..\leq e x\}).$

Polynomial.smult ($(-1) \wedge \text{card } X * (\prod x \in X. f (fst x))$)

$(\text{poly-X} \wedge \text{card } ((\text{SIGMA } x:A. \{..\leq e x\}) - X))$)

unfolding *case-prod-unfold*

by (*subst prod-diff1*) (*auto simp: mult-ac simp flip: coeff-lift-hom.hom-prod*)
 also have ... = $(\sum_{X \in \text{Pow}} (\text{SIGMA } x:A. \{..<e x\}).$
 Polynomial.monom $((- 1) ^ \text{card } X * (\prod_{x \in X}. f (fst x)))$ (*card* $((\text{SIGMA } x:A. \{..<e x\}) - X))$)
 unfolding *poly-X-def monom-power Polynomial.smult-monom* by *simp*
 also have *Polynomial.coeff* ... *i* = $(\sum_{X \in \{X \in \text{Pow} (\text{SIGMA } x:A. \{..<e x\}). i$
 =
 $\text{sum } e A - \text{card } X\}. (- 1) ^ \text{card } X * (\prod_{x \in X}. f (fst x)))$
 unfolding *Polynomial.coeff-sum*
 proof (*intro sum.mono-neutral-cong-right ballI, goal-cases*)
 case (*3 X*)
 hence *X*: $X \subseteq (\text{SIGMA } x:A. \{..<e x\})$
 by *auto*
 have *card-le*: $\text{card } X \leq \text{card} (\text{SIGMA } x:A. \{..<e x\})$
 using *X* by (*intro card-mono*) *auto*
 have *finite X*
 by (*rule finite-subset[OF X]*) *auto*
 hence $\text{card} ((\text{SIGMA } x:A. \{..<e x\}) - X) = \text{card} (\text{SIGMA } x:A. \{..<e x\}) -$
card X
 using *3* by (*intro card-Diff-subset*) *auto*
 also have *card-eq*: $\text{card} (\text{SIGMA } x:A. \{..<e x\}) = \text{sum } e A$
 by (*subst card-SigmaI*) *auto*
 finally show ?*case*
 using *3 card-le card-eq* by (*auto simp: algebra-simps*)
 next
 case (*4 X*)
 hence *X*: $X \subseteq (\text{SIGMA } x:A. \{..<e x\})$
 by *auto*
 have *finite X*
 by (*rule finite-subset[OF X]*) *auto*
 hence $\text{card} ((\text{SIGMA } x:A. \{..<e x\}) - X) = \text{card} (\text{SIGMA } x:A. \{..<e x\}) -$
card X
 using *4* by (*intro card-Diff-subset*) *auto*
 also have *card-eq*: $\text{card} (\text{SIGMA } x:A. \{..<e x\}) = \text{sum } e A$
 by (*subst card-SigmaI*) *auto*
 finally show ?*case*
 using *4 card-eq* by (*auto simp: algebra-simps*)
 qed *auto*
 finally show ?*thesis* .
 qed

lemma (*in comm-ring-hom*) *synthetic-div-hom*:
synthetic-div (*map-poly hom p*) (*hom x*) = *map-poly hom* (*synthetic-div p x*)
 by (*induction p*) (*auto simp: map-poly-pCons-hom*)

lemma *synthetic-div-altdef*:
 fixes *p* :: 'a :: field *poly*
 shows *synthetic-div p c* = *p div* $[-c, 1]$
 proof -

```

define q where  $q = p \text{ div } [-c, 1:]$ 
have Polynomial.degree ( $p \text{ mod } [-c, 1:]$ ) = 0
proof (cases  $p \text{ mod } [-c, 1:] = 0$ )
  case False
    hence Polynomial.degree ( $p \text{ mod } [-c, 1:]$ ) < Polynomial.degree  $[-c, 1:]$ 
    by (intro degree-mod-less') auto
    thus ?thesis by simp
qed auto
then obtain d where  $d: p \text{ mod } [-c, 1:] = [:d:]$ 
  by (elim degree-eq-zeroE)

have p-eq:  $p = q * [-c, 1:] + [:d:]$ 
  unfolding q-def d [symmetric] by presburger
have [simp]: poly  $p \ c = d$ 
  by (simp add: p-eq)
have  $p + \text{Polynomial.smult } c \ q = p\text{Cons } (\text{poly } p \ c) \ q$ 
  by (subst p-eq) auto
from synthetic-div-unique[OF this] show ?thesis
  by (auto simp: q-def)
qed

lemma (in ring-closed) poly-closed [intro]:
  assumes  $\bigwedge i. \text{poly.coeff } p \ i \in A \ x \in A$ 
  shows poly  $p \ x \in A$ 
  unfolding poly-altdef by (intro sum-closed mult-closed power-closed assms)

lemma (in ring-closed) coeff-pCons-closed [intro]:
  assumes  $\bigwedge i. \text{poly.coeff } p \ i \in A \ x \in A$ 
  shows poly.coeff ( $p\text{Cons } x \ p$ )  $i \in A$ 
  unfolding poly-altdef using assms by (auto simp: coeff-pCons split: nat.splits)

lemma (in ring-closed) coeff-poly-mult-closed [intro]:
  assumes  $\bigwedge i. \text{poly.coeff } p \ i \in A \ \bigwedge i. \text{poly.coeff } q \ i \in A$ 
  shows poly.coeff ( $p * q$ )  $i \in A$ 
  unfolding coeff-mult using assms by auto

lemma (in ring-closed) coeff-poly-prod-closed [intro]:
  assumes  $\bigwedge x \ i. x \in X \implies \text{poly.coeff } (f \ x) \ i \in A$ 
  shows poly.coeff ( $\text{prod } f \ X$ )  $i \in A$ 
  using assms by (induction X arbitrary: i rule: infinite-finite-induct) auto

lemma (in ring-closed) coeff-poly-power-closed [intro]:
  assumes  $\bigwedge i. \text{poly.coeff } p \ i \in A$ 
  shows poly.coeff ( $p \wedge^n$ )  $i \in A$ 
  using coeff-poly-prod-closed[of  $\{..<n\} \ \lambda-. \ p \ i$ ] assms by simp

lemma (in ring-closed) synthetic-div-closed:
  assumes  $\bigwedge i. \text{poly.coeff } p \ i \in A \ x \in A$ 
  shows poly.coeff (synthetic-div  $p \ x$ )  $i \in A$ 

```

```

proof –
  from assms(1) have  $\forall i. \text{poly.coeff } p \ i \in A$ 
    by blast
  from this and assms(2) show ?thesis
    by (induction p arbitrary: i) (auto simp: coeff-pCons split: nat.splits)
qed

lemma pcompose-monom: pcompose (Polynomial.monom c n) p = Polynomial.smult
c (p ^ n)
  by (simp add: monom-altdef pcompose-hom.hom-power pcompose-smult)

lemma poly-roots-uminus [simp]: poly-roots (-p) = poly-roots p
  using poly-roots-smult[of -1 p] by (simp del: poly-roots-smult)

lemma poly-roots-normalize [simp]:
  fixes p :: 'a :: {normalization-semidom, idom-divide} poly
  shows poly-roots (normalize p) = poly-roots p
proof (cases p = 0)
  case [simp]: False
  have poly-roots p = poly-roots (unit-factor p * normalize p)
    by simp
  also have  $\dots = \text{poly-roots (normalize p)}$ 
    unfolding unit-factor-poly-def by simp
  finally show ?thesis ..
qed auto

lemma poly-roots-of-int-normalize [simp]:
  poly-roots (of-int-poly (normalize p) :: 'a :: {idom, ring-char-0} poly) =
poly-roots (of-int-poly p)
proof (cases p = 0)
  case [simp]: False
  have poly-roots (of-int-poly p :: 'a poly) = poly-roots (of-int-poly (unit-factor p * normalize p))
    by simp
  also have  $\dots = \text{poly-roots (Polynomial.smult (of-int (sgn (Polynomial.lead-coeff p)))$ 
    (of-int-poly (normalize p)))
    by (simp add: unit-factor-poly-def of-int-hom.map-poly-hom-smult)
  also have  $\dots = \text{poly-roots (Ring-Hom-Poly.of-int-poly (normalize p) :: 'a poly)}$ 
    by (intro poly-roots-smult) (auto simp: sgn-if)
  finally show ?thesis ..
qed auto

lemma poly-roots-power [simp]: poly-roots (p ^ n) = repeat-mset n (poly-roots p)
proof (cases p = 0)
  case True
  thus ?thesis by (cases n) auto
next

```

case *False*
thus *?thesis* **by** (*induction n*) (*auto simp: poly-roots-mult*)
qed

lemma *poly-roots-conv-sum-prime-factors:*

poly-roots q = (∑ p ∈ #prime-factorization q. poly-roots p)

proof (*cases q = 0*)

case [*simp*]: *False*

have $(\sum p \in \#prime\text{-factorization } q. poly\text{-roots } p) =$
 $poly\text{-roots } (prod\text{-mset } (prime\text{-factorization } q))$

by (*rule poly-roots-prod-mset [symmetric]*) *auto*

also have $\dots = poly\text{-roots } (normalize (prod\text{-mset } (prime\text{-factorization } q)))$

by *simp*

also have $normalize (prod\text{-mset } (prime\text{-factorization } q)) = normalize q$

by (*rule prod-mset-prime-factorization-weak*) *auto*

also have $poly\text{-roots } \dots = poly\text{-roots } q$

by *simp*

finally show *?thesis ..*

qed *auto*

lemma *poly-roots-of-int-conv-sum-prime-factors:*

poly-roots (of-int-poly q :: 'a :: {idom, ring-char-0} poly) =

$(\sum p \in \#prime\text{-factorization } q. poly\text{-roots } (of\text{-int-poly } p))$

proof (*cases q = 0*)

case [*simp*]: *False*

have $(\sum p \in \#prime\text{-factorization } q. poly\text{-roots } (of\text{-int-poly } p :: 'a \text{ poly})) =$
 $poly\text{-roots } (\prod p \in \#prime\text{-factorization } q. of\text{-int-poly } p)$

by (*subst poly-roots-prod-mset*) (*auto simp: multiset.map-comp o-def*)

also have $(\prod p \in \#prime\text{-factorization } q. of\text{-int-poly } p :: 'a \text{ poly}) =$

$of\text{-int-poly } (prod\text{-mset } (prime\text{-factorization } q))$

by *simp*

also have $poly\text{-roots } \dots = poly\text{-roots } (of\text{-int-poly } (normalize (prod\text{-mset } (prime\text{-factorization } q))))$

by (*rule poly-roots-of-int-normalize [symmetric]*)

also have $normalize (prod\text{-mset } (prime\text{-factorization } q)) = normalize q$

by (*rule prod-mset-prime-factorization-weak*) *auto*

also have $poly\text{-roots } (of\text{-int-poly } \dots :: 'a \text{ poly}) = poly\text{-roots } (of\text{-int-poly } q)$

by *simp*

finally show *?thesis ..*

qed *auto*

lemma *dvd-imp-poly-roots-subset:*

assumes $q \neq 0$ $p \text{ dvd } q$

shows $poly\text{-roots } p \subseteq \# poly\text{-roots } q$

proof –

from *assms* **have** $p \neq 0$

by *auto*

thus *?thesis*
using *assms* **by** (*intro mset-subset-eqI*) (*auto intro: dvd-imp-order-le*)
qed

lemma *abs-prod-mset*: $|prod-mset (A :: 'a :: idom-abs-sgn multiset)| = prod-mset$
(image-mset abs A)
by (*induction A*) (*auto simp: abs-mult*)

lemma *content-1-imp-nonconstant-prime-factors*:
assumes *content* ($p :: int poly$) = 1 **and** $q \in prime-factors p$
shows *Polynomial.degree* $q > 0$
proof –

let $?d = Polynomial.degree :: int poly \Rightarrow nat$
let $?lc = Polynomial.lead-coeff :: int poly \Rightarrow int$
define P **where** $P = prime-factorization p$
define $P1$ **where** $P1 = filter-mset (\lambda p. ?d p = 0) P$
define $P2$ **where** $P2 = filter-mset (\lambda p. ?d p > 0) P$
have [*simp*]: $p \neq 0$
using *assms* **by** *auto*
have $1 = content (normalize p)$
using *assms* **by** *simp*
also **have** $normalize p = prod-mset P$
unfolding $P-def$ **by** (*rule prod-mset-prime-factorization [symmetric]*) *auto*
also **have** $P = filter-mset (\lambda p. ?d p = 0) P + filter-mset (\lambda p. ?d p > 0) P$
by (*induction P*) *auto*
also **have** $prod-mset \dots = prod-mset P1 * prod-mset P2$
unfolding $P1-def P2-def$ **by** (*subst prod-mset.union*) *auto*
also **have** $content \dots = content (prod-mset P1) * content (prod-mset P2)$
unfolding *content-mult ..*
also **have** $image-mset id P1 = image-mset (\lambda q. [?:lc q:]) P1$
by (*intro image-mset-cong*) (*auto simp: P1-def elim!: degree-eq-zeroE*)
hence $P1 = image-mset (\lambda q. [?:lc q:]) P1$
by *simp*
also **have** $content (prod-mset \dots) = |(\prod q \in \#P1. ?lc q)|$
by (*simp add: content-prod-mset multiset.map-comp o-def abs-prod-mset*)
finally **have** $|(\prod q \in \#P1. ?lc q)| * content (prod-mset P2) = 1 ..$
hence $|(\prod q \in \#P1. ?lc q)| dvd 1$
unfolding *dvd-def* **by** *metis*

have *set-mset* $P1 = \{\}$

proof (*rule ccontr*)

assume *set-mset* $P1 \neq \{\}$

then obtain q **where** $q \in \# P1$

by *blast*

have $|?lc q| dvd (\prod q \in \#P1. |?lc q|)$

by (*rule dvd-prod-mset*) (*use q in auto*)

also **have** $\dots = |(\prod q \in \#P1. ?lc q)|$

by (*simp add: abs-prod-mset multiset.map-comp o-def*)

also **have** $\dots dvd 1$

```

    by fact
  finally have is-unit (?lc q)
    by simp
  hence is-unit q
    using q unfolding P1-def by (auto elim!: degree-eq-zeroE)
  moreover have prime q
    using q unfolding P1-def P-def by auto
  ultimately show False by auto
qed
with assms show ?thesis
  by (auto simp: P1-def P-def)
qed

end

```

3 The minimal polynomial of an algebraic number

theory *Min-Int-Poly*

imports

Algebraic-Numbers.Algebraic-Numbers

HOL-Computational-Algebra.Computational-Algebra

More-Polynomial-HLW

begin

Given an algebraic number x in a field, the minimal polynomial is the unique irreducible integer polynomial with positive leading coefficient that has x as a root.

Note that we assume characteristic 0 since the material upon which all of this builds also assumes it.

definition *min-int-poly* :: 'a :: field-char-0 \Rightarrow int poly **where**

```

  min-int-poly x =
    (if algebraic x then THE p. p represents x  $\wedge$  irreducible p  $\wedge$  Polynomial.lead-coeff
     p > 0
     else [:0, 1:])

```

lemma

fixes $x :: 'a :: \{field-char-0, field-gcd\}$

shows *min-int-poly-represents* [intro]: algebraic $x \implies$ *min-int-poly* x represents x

and *min-int-poly-irreducible* [intro]: irreducible (*min-int-poly* x)

and *lead-coeff-min-int-poly-pos*: Polynomial.lead-coeff (*min-int-poly* x) > 0

proof –

note * = theI'[OF algebraic-imp-represents-unique, of x]

show *min-int-poly* x represents x **if** algebraic x

using *[OF that] **by** (simp add: that min-int-poly-def)

have irreducible [:0, 1::int:]

by (rule irreducible-linear-poly) auto

thus irreducible (*min-int-poly* x)

using * **by** (auto simp: min-int-poly-def)

show *Polynomial.lead-coeff (min-int-poly x) > 0*
using * by (*auto simp: min-int-poly-def*)
qed

lemma
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *degree-min-int-poly-pos [intro]: Polynomial.degree (min-int-poly x) > 0*
and *degree-min-int-poly-nonzero [simp]: Polynomial.degree (min-int-poly x) \neq 0*
proof –
show *Polynomial.degree (min-int-poly x) > 0*
proof (*cases algebraic x*)
case *True*
hence *min-int-poly x represents x*
by *auto*
thus *?thesis by blast*
qed (*auto simp: min-int-poly-def*)
thus *Polynomial.degree (min-int-poly x) \neq 0*
by *blast*
qed

lemma *min-int-poly-squarefree [intro]:*
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *squarefree (min-int-poly x)*
by (*rule irreducible-imp-squarefree*) *auto*

lemma *min-int-poly-primitive [intro]:*
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *primitive (min-int-poly x)*
by (*rule irreducible-imp-primitive*) *auto*

lemma *min-int-poly-content [simp]:*
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *content (min-int-poly x) = 1*
using *min-int-poly-primitive[of x] by (simp add: primitive-def)*

lemma *ipoly-min-int-poly [simp]:*
algebraic x \implies ipoly (min-int-poly x) (x :: 'a :: {field-gcd, field-char-0}) = 0
using *min-int-poly-represents[of x] by (auto simp: represents-def)*

lemma *min-int-poly-nonzero [simp]:*
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *min-int-poly x \neq 0*
using *lead-coeff-min-int-poly-pos[of x] by auto*

lemma *min-int-poly-normalize [simp]:*
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
shows *normalize (min-int-poly x) = min-int-poly x*
unfolding *normalize-poly-def using lead-coeff-min-int-poly-pos[of x] by simp*

```

lemma min-int-poly-prime-elem [intro]:
  fixes  $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ 
  shows prime-elem (min-int-poly  $x$ )
  using min-int-poly-irreducible[of  $x$ ] by blast

lemma min-int-poly-prime [intro]:
  fixes  $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ 
  shows prime (min-int-poly  $x$ )
  using min-int-poly-prime-elem[of  $x$ ]
  by (simp only: prime-normalize-iff [symmetric] min-int-poly-normalize)

lemma min-int-poly-unique:
  fixes  $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ 
  assumes  $p$  represents  $x$  irreducible  $p$  Polynomial.lead-coeff  $p > 0$ 
  shows min-int-poly  $x = p$ 
proof –
  from assms(1) have  $x$ : algebraic  $x$ 
    using algebraic-iff-represents by blast
  thus ?thesis
    using the1-equality[OF algebraic-imp-represents-unique[OF  $x$ ], of  $p$ ] assms
    unfolding min-int-poly-def by auto
qed

lemma min-int-poly-of-int [simp]:
  min-int-poly (of-int  $n :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ ) = [ $:-$ of-int  $n$ , 1:]
  by (intro min-int-poly-unique irreducible-linear-poly) auto

lemma min-int-poly-of-nat [simp]:
  min-int-poly (of-nat  $n :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ ) = [ $:-$ of-nat  $n$ , 1:]
  using min-int-poly-of-int[of int  $n$ ] by (simp del: min-int-poly-of-int)

lemma min-int-poly-0 [simp]: min-int-poly ( $0 :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ ) =
[ $0$ , 1:]
  using min-int-poly-of-int[of 0] unfolding of-int-0 by simp

lemma min-int-poly-1 [simp]: min-int-poly ( $1 :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ ) =
[ $-1$ , 1:]
  using min-int-poly-of-int[of 1] unfolding of-int-1 by simp

lemma poly-min-int-poly-0-eq-0-iff [simp]:
  fixes  $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$ 
  assumes algebraic  $x$ 
  shows poly (min-int-poly  $x$ )  $0 = 0 \iff x = 0$ 
proof
  assume *: poly (min-int-poly  $x$ )  $0 = 0$ 
  show  $x = 0$ 
  proof (rule ccontr)
    assume  $x \neq 0$ 

```

```

    hence poly (min-int-poly x) 0 ≠ 0
      using assms by (intro represents-irr-non-0) auto
    with * show False by contradiction
  qed
qed auto

lemma min-int-poly-conv-Gcd:
  fixes x :: 'a :: {field-char-0, field-gcd}
  assumes algebraic x
  shows min-int-poly x = Gcd {p. p ≠ 0 ∧ p represents x}
proof (rule sym, rule Gcd-eqI, (safe)?)
  fix p assume p: ∧q. q ∈ {p. p ≠ 0 ∧ p represents x} ⇒ p dvd q
  show p dvd min-int-poly x
    using assms by (intro p) auto
next
  fix p assume p: p ≠ 0 p represents x
  have min-int-poly x represents x
    using assms by auto
  hence poly (gcd (of-int-poly (min-int-poly x)) (of-int-poly p)) x = 0
    using p by (intro poly-gcd-eq-0I) auto
  hence ipoly (gcd (min-int-poly x) p) x = 0
    by (subst (asm) gcd-of-int-poly) auto
  hence gcd (min-int-poly x) p represents x
    using p unfolding represents-def by auto

  have min-int-poly x dvd gcd (min-int-poly x) p ∨ is-unit (gcd (min-int-poly x) p)
    by (intro irreducibleD') auto
  moreover from (gcd (min-int-poly x) p represents x) have ¬is-unit (gcd (min-int-poly
x) p)
    by (auto simp: represents-def)
  ultimately have min-int-poly x dvd gcd (min-int-poly x) p
    by blast
  also have ... dvd p
    by blast
  finally show min-int-poly x dvd p .
qed auto

lemma min-int-poly-eqI:
  fixes x :: 'a :: {field-char-0, field-gcd}
  assumes p represents x irreducible p Polynomial.lead-coeff p ≥ 0
  shows min-int-poly x = p
proof -
  from assms have [simp]: p ≠ 0
    by auto
  have Polynomial.lead-coeff p ≠ 0
    by auto
  with assms(3) have Polynomial.lead-coeff p > 0
    by linarith
  moreover have algebraic x

```

using $\langle p \text{ represents } x \rangle$ **by** $(\text{meson algebraic-iff-represents})$
ultimately show $?thesis$
unfolding $min-int-poly-def$
using $the1-equality[OF \text{ algebraic-imp-represents-unique}[OF \langle algebraic \ x \rangle], \text{ of } p]$
assms **by** *auto*
qed
end

4 The lexicographic ordering on complex numbers

theory *Complex-Lexorder*
imports *Complex-Main HOL-Library.Multiset*
begin

We define a lexicographic order on the complex numbers, comparing first the real parts and, if they are equal, the imaginary parts. This ordering is of course not compatible with multiplication, but it is compatible with addition.

definition *less-eq-complex-lex* (**infix** $\leq_{\mathbb{C}}$ 50) **where**
 $less-eq-complex-lex \ x \ y \iff Re \ x < Re \ y \vee Re \ x = Re \ y \wedge Im \ x \leq Im \ y$

definition *less-complex-lex* (**infix** $<_{\mathbb{C}}$ 50) **where**
 $less-complex-lex \ x \ y \iff Re \ x < Re \ y \vee Re \ x = Re \ y \wedge Im \ x < Im \ y$

interpretation *complex-lex*:
 $linordered-ab-group-add \ (+) \ 0 \ (-) \ uminus \ less-eq-complex-lex \ less-complex-lex$
by *standard* (*auto simp: less-eq-complex-lex-def less-complex-lex-def complex-eq-iff*)

lemmas [*trans*] =
 $complex-lex.order.trans \ complex-lex.less-le-trans$
 $complex-lex.less-trans \ complex-lex.le-less-trans$

lemma (**in** *ordered-comm-monoid-add*) *sum-mono-complex-lex*:
 $(\bigwedge i. i \in K \implies f \ i \leq_{\mathbb{C}} \ g \ i) \implies (\sum i \in K. f \ i) \leq_{\mathbb{C}} (\sum i \in K. g \ i)$
by (*induct K rule: infinite-finite-induct*) (*use complex-lex.add-mono in auto*)

lemma *sum-strict-mono-ex1-complex-lex*:
fixes $f \ g :: 'i \Rightarrow complex$
assumes *finite A*
and $\forall x \in A. f \ x \leq_{\mathbb{C}} \ g \ x$
and $\exists a \in A. f \ a <_{\mathbb{C}} \ g \ a$
shows $sum \ f \ A <_{\mathbb{C}} \ sum \ g \ A$
proof–
from *assms*(β) **obtain** a **where** $a : a \in A \ f \ a <_{\mathbb{C}} \ g \ a$ **by** *blast*
have $sum \ f \ A = sum \ f \ ((A - \{a\}) \cup \{a\})$
by (*simp add: insert-absorb[OF $\langle a \in A \rangle$]*)
also have $\dots = sum \ f \ (A - \{a\}) + sum \ f \ \{a\}$

using $\langle \text{finite } A \rangle$ **by** $(\text{subst sum.union-disjoint})$ *auto*
also have $\dots \leq_{\mathbb{C}} \text{sum } g (A - \{a\}) + \text{sum } f \{a\}$
by $(\text{intro complex-lex.add-mono sum-mono-complex-lex})$ $(\text{simp-all add: assms})$
also have $\dots <_{\mathbb{C}} \text{sum } g (A - \{a\}) + \text{sum } g \{a\}$
using a **by** $(\text{intro complex-lex.add-strict-left-mono})$ *auto*
also have $\dots = \text{sum } g ((A - \{a\}) \cup \{a\})$
using $\langle \text{finite } A \rangle$ **by** $(\text{subst sum.union-disjoint[symmetric]})$ *auto*
also have $\dots = \text{sum } g A$ **by** $(\text{simp add: insert-absorb[OF } \langle a \in A \rangle])$
finally show *?thesis*
by *simp*
qed

lemma *sum-list-mono-complex-lex*:
assumes *list-all2* $(\leq_{\mathbb{C}})$ $xs\ ys$
shows *sum-list* $xs \leq_{\mathbb{C}}$ *sum-list* ys
using *assms* **by** *induction* $(\text{auto intro: complex-lex.add-mono})$

lemma *sum-mset-mono-complex-lex*:
assumes *rel-mset* $(\leq_{\mathbb{C}})$ $A\ B$
shows *sum-mset* $A \leq_{\mathbb{C}}$ *sum-mset* B
using *assms* **by** $(\text{auto simp: rel-mset-def sum-mset-sum-list intro: sum-list-mono-complex-lex})$

lemma *rel-msetI*:
assumes *list-all2* $R\ xs\ ys$ *mset* $xs = A$ *mset* $ys = B$
shows *rel-mset* $R\ A\ B$
using *assms* **by** $(\text{auto simp: rel-mset-def})$

lemma *mset-replicate* [*simp*]: *mset* $(\text{replicate } n\ x) = \text{replicate-mset } n\ x$
by $(\text{induction } n)$ *auto*

lemma *rel-mset-replicate-mset-right*:
assumes $\bigwedge x. x \in \# A \implies R\ x\ y$ *size* $A = n$
shows *rel-mset* $R\ A (\text{replicate-mset } n\ y)$
proof –
obtain xs **where** [*simp*]: $A = \text{mset } xs$
by (metis ex-mset)
from *assms* **have** $\forall x \in \text{set } xs. R\ x\ y$
by *auto*
hence *list-all2* $R\ xs (\text{replicate } (\text{length } xs)\ y)$
by $(\text{induction } xs)$ *auto*
with *assms*(2) **show** *?thesis*
by $(\text{intro rel-msetI[of } R\ xs\ \text{replicate } n\ y])$ *auto*
qed

end

5 Additional facts about multivariate polynomials

theory *More-Multivariate-Polynomial-HLW*

imports *Power-Sum-Polynomials.Power-Sum-Polynomials-Library*
begin

5.1 Miscellaneous

lemma *Var-altdef*: $\text{Var } i = \text{monom } (\text{Poly-Mapping.single } i \ 1) \ 1$
by *transfer'* (*simp add: Var₀-def*)

lemma *Const-conv-monom*: $\text{Const } c = \text{monom } 0 \ c$
by *transfer'* (*auto simp: Const₀-def*)

lemma *smult-conv-mult-Const*: $\text{smult } c \ p = \text{Const } c * p$
by (*simp add: smult-conv-mult Const-conv-monom*)

lemma *mpoly-map-vars-Var [simp]*: $\text{bij } f \implies \text{mpoly-map-vars } f \ (\text{Var } i) = \text{Var } (f \ i)$
unfolding *Var-altdef*
by (*subst mpoly-map-vars-monom*) (*auto simp: permutep-single bij-imp-bij-inv inv-inv-eq*)

lemma *symmetric-mpoly-symmetric-prod'*:
assumes $\bigwedge \pi. \pi \text{ permutes } A \implies g \ \pi \text{ permutes } X$
assumes $\bigwedge x \ \pi. x \in X \implies \pi \text{ permutes } A \implies \text{mpoly-map-vars } \pi \ (f \ x) = f \ (g \ \pi \ x)$
shows $\text{symmetric-mpoly } A \ (\prod_{x \in X}. f \ x)$
unfolding *symmetric-mpoly-def*
proof safe
fix π **assume** $\pi: \pi \text{ permutes } A$
have $\text{mpoly-map-vars } \pi \ (\text{prod } f \ X) = (\prod_{x \in X}. \text{mpoly-map-vars } \pi \ (f \ x))$
by *simp*
also have $\dots = (\prod_{x \in X}. f \ (g \ \pi \ x))$
by (*intro prod.cong assms π refl*)
also have $\dots = (\prod_{x \in g \ \pi \ X}. f \ x)$
using *assms(1)[OF π]* **by** (*subst prod.reindex*) (*auto simp: permutes-inj-on*)
also have $g \ \pi \ X = X$
using *assms(1)[OF π]* **by** (*simp add: permutes-image*)
finally show $\text{mpoly-map-vars } \pi \ (\text{prod } f \ X) = \text{prod } f \ X$.
qed

5.2 Converting a univariate polynomial into a multivariate one

lift-definition *mpoly-of-poly-aux* :: $\text{nat} \Rightarrow 'a \ :: \text{zero poly} \Rightarrow (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a$
is

$\lambda i \ c \ m. \text{if } \text{Poly-Mapping.keys } m \subseteq \{i\} \text{ then } c \ (\text{Poly-Mapping.lookup } m \ i) \text{ else } 0$
proof goal-cases
case (*1 i c*)
hence *fin*: $\text{finite } \{n. c \ n \neq 0\}$
by (*metis eventually-cofinite*)


```

show finite {x. (if keys x ⊆ {i} then c (lookup x i) else 0) ≠ 0}
proof (rule finite-subset)
  show finite (Poly-Mapping.single i ' {n. c n ≠ 0})
  by (intro finite-imageI fin)
next
show {x. (if keys x ⊆ {i} then c (lookup x i) else 0) ≠ 0} ⊆
  Poly-Mapping.single i ' {n. c n ≠ 0}
proof (safe, split if-splits)
  fix x :: (nat ⇒0 nat)
  assume x: keys x ⊆ {i} c (lookup x i) ≠ 0
  hence x = Poly-Mapping.single i (lookup x i)
  by (metis Diff-eq-empty-iff keys-empty-iff lookup-single-eq
    remove-key-keys remove-key-single remove-key-sum)
  thus x ∈ Poly-Mapping.single i ' {n. c n ≠ 0}
  using x by blast
qed auto
qed
qed

lift-definition mpoly-of-poly :: nat ⇒ 'a :: zero poly ⇒ 'a mpoly is
  mpoly-of-poly-aux .

lemma mpoly-of-poly-0 [simp]: mpoly-of-poly i 0 = 0
  by (transfer', transfer) auto

lemma coeff-mpoly-of-poly1 [simp]:
  coeff (mpoly-of-poly i p) (Poly-Mapping.single i n) = poly.coeff p n
  by (transfer', transfer) auto

lemma coeff-mpoly-of-poly2 [simp]:
  assumes ¬keys x ⊆ {i}
  shows coeff (mpoly-of-poly i p) x = 0
  using assms by (transfer', transfer) auto

lemma coeff-mpoly-of-poly:
  coeff (mpoly-of-poly i p) m =
  (poly.coeff p (Poly-Mapping.lookup m i) when keys m ⊆ {i})
  by (transfer', transfer) auto

lemma poly-mapping-single-eq-0-iff [simp]: Poly-Mapping.single i n = 0 ⟷ n =
  0
  by (metis lookup-single-eq single-zero)

lemma mpoly-of-poly-pCons [simp]:
  fixes p :: 'a :: semiring-1 poly
  shows mpoly-of-poly i (pCons c p) = Const c + Var i * mpoly-of-poly i p
proof (rule mpoly-eqI)
  fix mon :: nat ⇒0 nat
  define moni :: nat ⇒0 nat where moni = Poly-Mapping.single i 1

```

```

have coeff (Var i * mpoly-of-poly i p) mon =
  (∑ l. (1 when l = moni) * (∑ q. coeff (mpoly-of-poly i p) q when mon =
moni + q))
  unfolding coeff-mpoly-times prod-fun-def coeff-Var moni-def
  by (rule Sum-any.cong) (auto simp: when-def)
also have ... = (∑ a. coeff (mpoly-of-poly i p) a when mon = moni + a)
  by (subst Sum-any-left-distrib [symmetric]) simp-all
finally have eq: coeff (Var i * mpoly-of-poly i p) mon = ... .

  show coeff (mpoly-of-poly i (pCons c p)) mon = coeff (Const c + Var i *
mpoly-of-poly i p) mon
  proof (cases keys mon ⊆ {i})
    case False
      hence [simp]: mon ≠ 0
      by auto
      obtain j where j: j ∈ keys mon j ≠ i
      using False by auto
      have coeff (mpoly-of-poly i p) mon' = 0 if mon-eq: mon = moni + mon' for
mon'
      proof –
        have Poly-Mapping.lookup mon j ≠ 0
          using j by (meson lookup-eq-zero-in-keys-contradict)
        also have Poly-Mapping.lookup mon j = Poly-Mapping.lookup mon' j
          unfolding mon-eq moni-def using j by (simp add: lookup-add lookup-single)
        finally have j ∈ keys mon'
          by (meson lookup-not-eq-zero-eq-in-keys)
        with j have ¬keys mon' ⊆ {i}
          by blast
        thus ?thesis by simp
      qed
      hence coeff (Var i * mpoly-of-poly i p) mon = 0
        unfolding eq by (intro Sum-any-zeroI) (auto simp: when-def)
      thus ?thesis using False
        by (simp add: mpoly-coeff-Const)
    next
      case True
      define n where n = Poly-Mapping.lookup mon i
      have mon-eq: mon = Poly-Mapping.single i n
        using True unfolding n-def
        by (metis Diff-eq-empty-iff add-cancel-right-left keys-empty-iff remove-key-keys
remove-key-sum)
      have eq': mon = moni + mon' ↔ n > 0 ∧ mon' = Poly-Mapping.single i (n
– 1) for mon'
      proof safe
        assume eq: mon = moni + mon'
        thus n > 0 mon' = Poly-Mapping.single i (n – 1)
          unfolding moni-def mon-eq using grOI by (force simp: single-diff)+
      next
        assume n > 0 mon' = Poly-Mapping.single i (n – 1)

```

thus $mon = moni + Poly\text{-}Mapping.single\ i\ (n - 1)$
unfolding $mon\text{-}eq\ moni\text{-}def$ **by** $(subst\ single\text{-}add\ [symmetric])\ auto$
qed
have $coeff\ (Var\ i * mpoly\text{-}of\text{-}poly\ i\ p)\ mon = (poly.coeff\ p\ (n - 1)\ when\ (n > 0))$
unfolding $eq\ eq'$ **by** $(auto\ simp:\ when\text{-}def)$
thus *?thesis*
by $(auto\ simp:\ mon\text{-}eq\ when\text{-}def\ mpoly\text{-}coeff\text{-}Const\ coeff\text{-}pCons\ split:\ nat.splits)$
qed
qed

lemma $mpoly\text{-}of\text{-}poly\text{-}1\ [simp]: mpoly\text{-}of\text{-}poly\ i\ 1 = 1$
unfolding $one\text{-}pCons\ mpoly\text{-}of\text{-}poly\text{-}pCons\ mpoly\text{-}of\text{-}poly\text{-}0$ **by** $simp$

lemma $mpoly\text{-}of\text{-}poly\text{-}uminus\ [simp]: mpoly\text{-}of\text{-}poly\ i\ (-p) = -mpoly\text{-}of\text{-}poly\ i\ p$
by $(rule\ mpoly\text{-}eqI)\ (auto\ simp:\ coeff\text{-}mpoly\text{-}of\text{-}poly\ when\text{-}def)$

lemma $mpoly\text{-}of\text{-}poly\text{-}add\ [simp]: mpoly\text{-}of\text{-}poly\ i\ (p + q) = mpoly\text{-}of\text{-}poly\ i\ p + mpoly\text{-}of\text{-}poly\ i\ q$
by $(rule\ mpoly\text{-}eqI)\ (auto\ simp:\ coeff\text{-}mpoly\text{-}of\text{-}poly\ when\text{-}def)$

lemma $mpoly\text{-}of\text{-}poly\text{-}diff\ [simp]: mpoly\text{-}of\text{-}poly\ i\ (p - q) = mpoly\text{-}of\text{-}poly\ i\ p - mpoly\text{-}of\text{-}poly\ i\ q$
by $(rule\ mpoly\text{-}eqI)\ (auto\ simp:\ coeff\text{-}mpoly\text{-}of\text{-}poly\ when\text{-}def)$

lemma $mpoly\text{-}of\text{-}poly\text{-}smult\ [simp]:$
 $mpoly\text{-}of\text{-}poly\ i\ (Polynomial.smult\ c\ p) = smult\ c\ (mpoly\text{-}of\text{-}poly\ i\ p)$
by $(rule\ mpoly\text{-}eqI)\ (auto\ simp:\ coeff\text{-}mpoly\text{-}of\text{-}poly\ when\text{-}def)$

lemma $mpoly\text{-}of\text{-}poly\text{-}mult\ [simp]:$
fixes $p\ q :: 'a :: comm\text{-}semiring\text{-}1\ poly$
shows $mpoly\text{-}of\text{-}poly\ i\ (p * q) = mpoly\text{-}of\text{-}poly\ i\ p * mpoly\text{-}of\text{-}poly\ i\ q$
by $(induction\ p)\ (auto\ simp:\ algebra\text{-}simps\ smult\text{-}conv\text{-}mult\text{-}Const)$

lemma $insertion\text{-}mpoly\text{-}of\text{-}poly\ [simp]: insertion\ f\ (mpoly\text{-}of\text{-}poly\ i\ p) = poly\ p\ (f\ i)$
by $(induction\ p)\ (auto\ simp:\ insertion\text{-}add\ insertion\text{-}mult)$

lemma $mapping\text{-}of\text{-}mpoly\text{-}of\text{-}poly\ [simp]: mapping\text{-}of\ (mpoly\text{-}of\text{-}poly\ i\ p) = mpoly\text{-}of\text{-}poly\text{-}aux\ i\ p$
by $transfer'\ simp$

lemma $vars\text{-}mpoly\text{-}of\text{-}poly: vars\ (mpoly\text{-}of\text{-}poly\ i\ p) \subseteq \{i\}$

proof –

have $x = i$ **if** $xa \in keys\ (mpoly\text{-}of\text{-}poly\text{-}aux\ i\ p)$ $x \in keys\ xa$ **for** $x\ xa$
using $that$
by $(meson\ in\text{-}mono\ lookup\text{-}eq\ zero\text{-}in\text{-}keys\ contradict\ mpoly\text{-}of\text{-}poly\text{-}aux.rep\text{-}eq\ singletonD)$
thus *?thesis*

by (auto simp: vars-def)
qed

lemma *mpoly-map-vars-mpoly-of-poly* [simp]:
 assumes *bij f*
 shows $mpoly\text{-map-}vars\ f\ (mpoly\text{-of-poly}\ i\ p) = mpoly\text{-of-poly}\ (f\ i)\ p$
proof (rule *mpoly-eqI*, goal-cases)
 case (1 *mon*)
 have $f\ \text{keys}\ mon \subseteq \{i\} \iff keys\ mon \subseteq \{f\ i\}$
 using *assms* by (simp add: *vimage-subset-eq*)
 thus ?case using *assms*
 by (simp add: *coeff-mpoly-map-vars coeff-mpoly-of-poly lookup-permutep keys-permutep when-def*)
 qed

end

6 More facts about algebraic numbers

theory *More-Algebraic-Numbers-HLW*
 imports *Algebraic-Numbers.Algebraic-Numbers*
 begin

6.1 Miscellaneous

lemma *in-Ints-imp-algebraic* [simp, intro]: $x \in \mathbb{Z} \implies algebraic\ x$
 by (intro *algebraic-int-imp-algebraic int-imp-algebraic-int*)

lemma *in-Rats-imp-algebraic* [simp, intro]: $x \in \mathbb{Q} \implies algebraic\ x$
 by (auto elim!: *Rats-cases'* intro: *algebraic-div*)

lemma *algebraic-uminus-iff* [simp]: $algebraic\ (-x) \iff algebraic\ x$
 using *algebraic-uminus*[of *x*] *algebraic-uminus*[of $-x$] by auto

lemma *algebraic-0* [simp]: $algebraic\ (0 :: 'a :: field-char-0)$
 and *algebraic-1* [simp]: $algebraic\ (1 :: 'a :: field-char-0)$
 by auto

lemma *algebraic-sum* [intro]:
 $(\bigwedge x. x \in A \implies algebraic\ (f\ x)) \implies algebraic\ (sum\ f\ A)$
 by (induction *A* rule: *infinite-finite-induct*) (auto intro!: *algebraic-plus*)

lemma *algebraic-prod* [intro]:
 $(\bigwedge x. x \in A \implies algebraic\ (f\ x)) \implies algebraic\ (prod\ f\ A)$
 by (induction *A* rule: *infinite-finite-induct*) (auto intro!: *algebraic-times*)

lemma *algebraic-sum-list* [intro]:
 $(\bigwedge x. x \in set\ xs \implies algebraic\ x) \implies algebraic\ (sum-list\ xs)$
 by (induction *xs*) (auto intro!: *algebraic-plus*)

lemma algebraic-prod-list [intro]:
 $(\bigwedge x. x \in \text{set } xs \implies \text{algebraic } x) \implies \text{algebraic } (\text{prod-list } xs)$
by (induction xs) (auto intro!: algebraic-times)

lemma algebraic-sum-mset [intro]:
 $(\bigwedge x. x \in \# A \implies \text{algebraic } x) \implies \text{algebraic } (\text{sum-mset } A)$
by (induction A) (auto intro!: algebraic-plus)

lemma algebraic-prod-mset [intro]:
 $(\bigwedge x. x \in \# A \implies \text{algebraic } x) \implies \text{algebraic } (\text{prod-mset } A)$
by (induction A) (auto intro!: algebraic-times)

lemma algebraic-power [intro]: $\text{algebraic } x \implies \text{algebraic } (x \wedge n)$
by (induction n) (auto intro: algebraic-times)

lemma algebraic-csqr [intro]: $\text{algebraic } x \implies \text{algebraic } (\text{csqr } x)$
by (rule algebraic-nth-root[of 2 x]) auto

lemma algebraic-csqr-iff [simp]: $\text{algebraic } (\text{csqr } x) \longleftrightarrow \text{algebraic } x$
proof
assume $\text{algebraic } (\text{csqr } x)$
hence $\text{algebraic } (\text{csqr } x \wedge 2)$
by (rule algebraic-power)
also have $\text{csqr } x \wedge 2 = x$
by simp
finally show $\text{algebraic } x$.
qed auto

lemmas [intro] = algebraic-plus algebraic-times algebraic-uminus algebraic-div

lemma algebraic-power-iff [simp]:
assumes $n > 0$
shows $\text{algebraic } (x \wedge n) \longleftrightarrow \text{algebraic } x$
using algebraic-nth-root[of n x \wedge n x] **assms** **by** auto

lemma algebraic-ii [simp]: $\text{algebraic } i$
by (intro algebraic-int-imp-algebraic) auto

lemma algebraic-int-fact [simp, intro]: $\text{algebraic-int } (\text{fact } n)$
by (intro int-imp-algebraic-int fact-in-Ints)

lemma algebraic-minus [intro]: $\text{algebraic } x \implies \text{algebraic } y \implies \text{algebraic } (x - y)$
using algebraic-plus[of x -y] **by** simp

lemma algebraic-add-cancel-left [simp]:
assumes $\text{algebraic } x$
shows $\text{algebraic } (x + y) \longleftrightarrow \text{algebraic } y$
proof

assume *algebraic* $(x + y)$
hence *algebraic* $(x + y - x)$
using *assms* **by** (*intro algebraic-minus*) *auto*
thus *algebraic* y **by** *simp*
qed (*auto intro: algebraic-plus assms*)

lemma *algebraic-add-cancel-right* [*simp*]:
assumes *algebraic* y
shows *algebraic* $(x + y) \longleftrightarrow$ *algebraic* x
using *algebraic-add-cancel-left*[*of y x*] *assms*
by (*simp add: add.commute del: algebraic-add-cancel-left*)

lemma *algebraic-diff-cancel-left* [*simp*]:
assumes *algebraic* x
shows *algebraic* $(x - y) \longleftrightarrow$ *algebraic* y
using *algebraic-add-cancel-left*[*of x -y*] *assms* **by** (*simp del: algebraic-add-cancel-left*)

lemma *algebraic-diff-cancel-right* [*simp*]:
assumes *algebraic* y
shows *algebraic* $(x - y) \longleftrightarrow$ *algebraic* x
using *algebraic-add-cancel-right*[*of -y x*] *assms* **by** (*simp del: algebraic-add-cancel-right*)

lemma *algebraic-mult-cancel-left* [*simp*]:
assumes *algebraic* x $x \neq 0$
shows *algebraic* $(x * y) \longleftrightarrow$ *algebraic* y
proof
assume *algebraic* $(x * y)$
hence *algebraic* $(x * y / x)$
using *assms* **by** (*intro algebraic-div*) *auto*
also have $x * y / x = y$
using *assms* **by** *auto*
finally show *algebraic* y .
qed (*auto intro: algebraic-times assms*)

lemma *algebraic-mult-cancel-right* [*simp*]:
assumes *algebraic* y $y \neq 0$
shows *algebraic* $(x * y) \longleftrightarrow$ *algebraic* x
using *algebraic-mult-cancel-left*[*of y x*] *assms*
by (*simp add: mult.commute del: algebraic-mult-cancel-left*)

lemma *algebraic-inverse-iff* [*simp*]: *algebraic* $(inverse\ y) \longleftrightarrow$ *algebraic* y
proof
assume *algebraic* $(inverse\ y)$
hence *algebraic* $(inverse\ (inverse\ y))$
by (*rule algebraic-inverse*)
thus *algebraic* y **by** *simp*
qed (*auto intro: algebraic-inverse*)

lemma *algebraic-divide-cancel-left* [*simp*]:

```

assumes algebraic x x ≠ 0
shows algebraic (x / y) ↔ algebraic y
proof -
  have algebraic (x * inverse y) ↔ algebraic (inverse y)
    by (intro algebraic-mult-cancel-left assms)
  also have ... ↔ algebraic y
    by (intro algebraic-inverse-iff)
  finally show ?thesis by (simp add: field-simps)
qed

```

```

lemma algebraic-divide-cancel-right [simp]:
  assumes algebraic y y ≠ 0
  shows algebraic (x / y) ↔ algebraic x
proof -
  have algebraic (x * inverse y) ↔ algebraic x
    using assms by (intro algebraic-mult-cancel-right) auto
  thus ?thesis by (simp add: field-simps)
qed

```

6.2 Turning an algebraic number into an algebraic integer

6.3 Multiplying an algebraic number with a suitable integer turns it into an algebraic integer.

```

lemma algebraic-imp-algebraic-int:
  fixes x :: 'a :: field-char-0
  assumes ipoly p x = 0 p ≠ 0
  defines c ≡ Polynomial.lead-coeff p
  shows algebraic-int (of-int c * x)
proof -
  define n where n = Polynomial.degree p
  define p' where p' = Abs-poly (λi. if i = n then 1 else c ^ (n - i - 1) *
poly.coeff p i)
  have n > 0
    using assms unfolding n-def by (intro Nat.gr0I) (auto elim!: degree-eq-zeroE)

  have coeff-p': poly.coeff p' i =
    (if i = n then 1 else c ^ (n - i - 1) * poly.coeff p i)
    (is - = ?f i) for i unfolding p'-def
proof (subst poly.Abs-poly-inverse)
  have eventually (λi. poly.coeff p i = 0) cofinite
    using MOST-coeff-eq-0 by blast
  hence eventually (λi. ?f i = 0) cofinite
    by eventually-elim (use assms in (auto simp: n-def))
  thus ?f ∈ {f. eventually (λi. f i = 0) cofinite} by simp
qed auto

have deg-p': Polynomial.degree p' = n
proof -
  from assms have (λn. ∀ i > n. poly.coeff p' i = 0) = (λn. ∀ i > n. poly.coeff p i

```

```

= 0)
  by (auto simp: coeff-p' fun-eq-iff n-def)
  thus ?thesis
  by (simp add: Polynomial.degree-def n-def)
qed

have lead-coeff-p': Polynomial.lead-coeff p' = 1
  by (simp add: coeff-p' deg-p')

have 0 = of-int (c ^ (n - 1)) * (∑ i ≤ n. of-int (poly.coeff p i) * x ^ i)
  using assms unfolding n-def poly-altdef by simp
also have ... = (∑ i ≤ n. of-int (c ^ (n - 1) * poly.coeff p i) * x ^ i)
  by (simp add: sum-distrib-left sum-distrib-right mult-ac)
also have ... = (∑ i ≤ n. of-int (poly.coeff p' i) * (of-int c * x) ^ i)
proof (intro sum.cong, goal-cases)
  case (2 i)
  have of-int (poly.coeff p' i) * (of-int c * x) ^ i =
    of-int (c ^ i * poly.coeff p' i) * x ^ i
  by (simp add: algebra-simps)
  also have c ^ i * poly.coeff p' i = c ^ (n - 1) * poly.coeff p i
  proof (cases i = n)
    case True
    hence c ^ i * poly.coeff p' i = c ^ n
      by (auto simp: coeff-p' simp flip: power-Suc)
    also have n = Suc (n - 1)
      using ⟨n > 0⟩ by simp
    also have c ^ ... = c * c ^ (n - 1)
      by simp
    finally show ?thesis
      using True by (simp add: c-def n-def)
  next
  case False
  thus ?thesis using 2
    by (auto simp: coeff-p' simp flip: power-add)
  qed
  finally show ?case ..
qed auto
also have ... = ipoly p' (of-int c * x)
  by (simp add: poly-altdef n-def deg-p')
finally have ipoly p' (of-int c * x) = 0 ..

with lead-coeff-p' show ?thesis
  unfolding algebraic-int-altdef-ipoly by blast
qed

lemma algebraic-imp-algebraic-int':
  fixes x :: 'a :: field-char-0
  assumes ipoly p x = 0 p ≠ 0 Polynomial.lead-coeff p dvd c
  shows algebraic-int (of-int c * x)

```



```

proof –
  from assms( $\beta$ ) obtain  $c'$  where c-eq:  $c = \text{Polynomial.lead-coeff } p * c'$ 
    by auto
  have algebraic-int (of-int  $c' * (\text{of-int } (\text{Polynomial.lead-coeff } p) * x)$ )
    by (rule algebraic-int-times[OF - algebraic-imp-algebraic-int]) (use assms in auto)
  also have of-int  $c' * (\text{of-int } (\text{Polynomial.lead-coeff } p) * x) = \text{of-int } c * x$ 
    by (simp add: c-eq mult-ac)
  finally show ?thesis .
qed

end

```

7 Miscellaneous facts

theory *Misc-HLW*

imports

Complex-Main

HOL-Library.Multiset

HOL-Library.Permutations

HOL-Library.FuncSet

HOL-Library.Groups-Big-Fun

HOL-Library.Poly-Mapping

HOL-Library.Landau-Symbols

HOL-Computational-Algebra.Computational-Algebra

begin

lemma *set-mset-subset-singletonD*:

assumes *set-mset* $A \subseteq \{x\}$

shows $A = \text{replicate-mset } (\text{size } A) x$

using *assms* **by** (*induction A*) *auto*

lemma *image-mset-eq-replicate-msetD*:

assumes *image-mset* $f A = \text{replicate-mset } n y$

shows $\forall x \in \#A. f x = y$

proof –

have $f ` \text{set-mset } A = \text{set-mset } (\text{image-mset } f A)$

by *simp*

also note *assms*

finally show *?thesis* **by** (*auto split: if-splits*)

qed

lemma *bij-betw-permutes-compose-left*:

assumes π *permutes* A

shows *bij-betw* $(\lambda \sigma. \pi \circ \sigma) \{\sigma. \sigma \text{ permutes } A\} \{\sigma. \sigma \text{ permutes } A\}$

proof (*rule bij-betwI*)

show $(\circ) \pi \in \{\sigma. \sigma \text{ permutes } A\} \rightarrow \{\sigma. \sigma \text{ permutes } A\}$

by (*auto intro: permutes-compose assms*)

show $(\circ) (\text{inv-into UNIV } \pi) \in \{\sigma. \sigma \text{ permutes } A\} \rightarrow \{\sigma. \sigma \text{ permutes } A\}$

by (auto intro: permutes-compose assms permutes-inv)
qed (use permutes-inverses[OF assms] in auto)

lemma *bij-betw-compose-left-perm-Pi*:

assumes π permutes B
shows $\text{bij-betw } (\lambda f. (\pi \circ f)) (A \rightarrow B) (A \rightarrow B)$
proof (rule *bij-betwI*)
have $*$: $(\lambda f. (\pi \circ f)) \in (A \rightarrow B) \rightarrow A \rightarrow B$ **if** π : π permutes B **for** π
by (auto simp: permutes-in-image[OF π])
show $(\lambda f. (\pi \circ f)) \in (A \rightarrow B) \rightarrow A \rightarrow B$
by (rule $*$) fact
show $(\lambda f. (\text{inv-into UNIV } \pi \circ f)) \in (A \rightarrow B) \rightarrow A \rightarrow B$
by (intro $*$ permutes-inv) fact
qed (auto simp: permutes-inverses[OF assms] fun-eq-iff)

lemma *bij-betw-compose-left-perm-PiE*:

assumes π permutes B
shows $\text{bij-betw } (\lambda f. \text{restrict } (\pi \circ f) A) (A \rightarrow_E B) (A \rightarrow_E B)$
proof (rule *bij-betwI*)
have $*$: $(\lambda f. \text{restrict } (\pi \circ f) A) \in (A \rightarrow_E B) \rightarrow A \rightarrow_E B$ **if** π : π permutes B **for** π
by (auto simp: permutes-in-image[OF π])
show $(\lambda f. \text{restrict } (\pi \circ f) A) \in (A \rightarrow_E B) \rightarrow A \rightarrow_E B$
by (rule $*$) fact
show $(\lambda f. \text{restrict } (\text{inv-into UNIV } \pi \circ f) A) \in (A \rightarrow_E B) \rightarrow A \rightarrow_E B$
by (intro $*$ permutes-inv) fact
qed (auto simp: permutes-inverses[OF assms] fun-eq-iff)

lemma *bij-betw-image-mset-set*:

assumes *bij-betw* $f A B$
shows $\text{image-mset } f (\text{mset-set } A) = \text{mset-set } B$
using *assms* **by** (simp add: *bij-betw-def image-mset-mset-set*)

lemma *finite-multisets-of-size*:

assumes *finite* A
shows $\text{finite } \{X. \text{set-mset } X \subseteq A \wedge \text{size } X = n\}$
proof (rule *finite-subset*)
show $\{X. \text{set-mset } X \subseteq A \wedge \text{size } X = n\} \subseteq \text{mset } \{xs. \text{set } xs \subseteq A \wedge \text{length } xs = n\}$
proof
fix X **assume** X : $X \in \{X. \text{set-mset } X \subseteq A \wedge \text{size } X = n\}$
obtain xs **where** [simp]: $X = \text{mset } xs$
by (metis *ex-mset*)
thus $X \in \text{mset } \{xs. \text{set } xs \subseteq A \wedge \text{length } xs = n\}$
using X **by** *auto*
qed
next
show *finite* $(\text{mset } \{xs. \text{set } xs \subseteq A \wedge \text{length } xs = n\})$
by (intro *finite-imageI finite-lists-length-eq assms*)

qed

lemma *sum-mset-image-mset-sum-mset-image-mset:*

$sum\text{-}mset\ (image\text{-}mset\ g\ (sum\text{-}mset\ (image\text{-}mset\ f\ A))) =$
 $sum\text{-}mset\ (image\text{-}mset\ (\lambda x. sum\text{-}mset\ (image\text{-}mset\ g\ (f\ x)))\ A)$
by (*induction A*) *auto*

lemma *sum-mset-image-mset-singleton:* $sum\text{-}mset\ (image\text{-}mset\ (\lambda x. \{\#\ x\#\})\ A)$
 $= image\text{-}mset\ f\ A$

by (*induction A*) *auto*

lemma *sum-mset-conv-sum:*

$sum\text{-}mset\ (image\text{-}mset\ f\ A) = (\sum_{x \in set\text{-}mset\ A} of\text{-}nat\ (count\ A\ x) * f\ x)$

proof (*induction A rule: full-multiset-induct*)

case (*less A*)

show *?case*

proof (*cases A = \{\#\}*)

case *False*

then obtain *x where x: x ∈ \#\ A*

by *auto*

define *n where n = count A x*

define *A' where A' = filter-mset (\lambda y. y ≠ x) A*

have *A-eq: A = replicate-mset n x + A'*

by (*intro multiset-eqI*) (*auto simp: A'-def n-def*)

have [*simp*]: $x \notin \#\ A'$ *count A' x = 0*

by (*auto simp: A'-def*)

have $n \neq 0$

using *x by (auto simp: n-def)*

have $sum\text{-}mset\ (image\text{-}mset\ f\ A) = of\text{-}nat\ n * f\ x + sum\text{-}mset\ (image\text{-}mset\ f\ A')$

by (*simp add: A-eq*)

also have $A' \subset \#\ A$

unfolding *A'-def using x by (simp add: filter-mset-eq-conv subset-mset-def)*

with *less.IH* **have** $sum\text{-}mset\ (image\text{-}mset\ f\ A') = (\sum_{x \in set\text{-}mset\ A'} of\text{-}nat\ (count\ A'\ x) * f\ x)$

by *simp*

also have $\dots = (\sum_{x \in set\text{-}mset\ A'} of\text{-}nat\ (count\ A\ x) * f\ x)$

by (*intro sum.cong*) (*auto simp: A-eq*)

also have $of\text{-}nat\ n * f\ x + \dots = (\sum_{x \in insert\ x\ (set\text{-}mset\ A')} of\text{-}nat\ (count\ A\ x) * f\ x)$

by (*subst sum.insert*) (*auto simp: A-eq*)

also from $\langle n \neq 0 \rangle$ **have** $insert\ x\ (set\text{-}mset\ A') = set\text{-}mset\ A$

by (*auto simp: A-eq*)

finally show *?thesis .*

qed *auto*

qed

lemma *sum-mset-conv-Sum-any:*

$sum\text{-}mset\ (image\text{-}mset\ f\ A) = Sum\text{-}any\ (\lambda x. of\text{-}nat\ (count\ A\ x) * f\ x)$
proof –
have $sum\text{-}mset\ (image\text{-}mset\ f\ A) = (\sum x \in set\text{-}mset\ A. of\text{-}nat\ (count\ A\ x) * f\ x)$
by $(rule\ sum\text{-}mset\text{-}conv\text{-}sum)$
also have $\dots = Sum\text{-}any\ (\lambda x. of\text{-}nat\ (count\ A\ x) * f\ x)$
proof $(rule\ Sum\text{-}any.\text{expand}\text{-}superset\ [symmetric])$
show $\{x. of\text{-}nat\ (count\ A\ x) * f\ x \neq 0\} \subseteq set\text{-}mset\ A$
proof
fix x **assume** $x \in \{x. of\text{-}nat\ (count\ A\ x) * f\ x \neq 0\}$
hence $count\ A\ x \neq 0$
by $(intro\ notI)\ auto$
thus $x \in \# A$
by $auto$
qed
qed $auto$
finally show $?thesis$.
qed

lemma $Sum\text{-}any\text{-}sum\text{-}swap$:
assumes $finite\ A \wedge y. finite\ \{x. f\ x\ y \neq 0\}$
shows $Sum\text{-}any\ (\lambda x. sum\ (f\ x)\ A) = (\sum y \in A. Sum\text{-}any\ (\lambda x. f\ x\ y))$
proof –
have $Sum\text{-}any\ (\lambda x. sum\ (f\ x)\ A) = Sum\text{-}any\ (\lambda x. Sum\text{-}any\ (\lambda y. f\ x\ y\ when\ y \in A))$
unfolding $when\text{-}def$ **by** $(subst\ Sum\text{-}any.\text{conditionalize})$ $(use\ assms\ in\ simp\text{-}all)$
also have $\dots = Sum\text{-}any\ (\lambda y. Sum\text{-}any\ (\lambda x. f\ x\ y\ when\ y \in A))$
by $(intro\ Sum\text{-}any.\text{swap}[of\ (\bigcup y \in A. \{x. f\ x\ y \neq 0\}) \times A]\ finite\text{-}SigmaI\ finite\text{-}UN\text{-}I\ assms)\ auto$
also have $(\lambda y. Sum\text{-}any\ (\lambda x. f\ x\ y\ when\ y \in A)) = (\lambda y. Sum\text{-}any\ (\lambda x. f\ x\ y\ when\ y \in A))$
by $(auto\ simp:\ when\text{-}def)$
also have $Sum\text{-}any\ \dots = (\sum y \in A. Sum\text{-}any\ (\lambda x. f\ x\ y))$
unfolding $when\text{-}def$ **by** $(subst\ Sum\text{-}any.\text{conditionalize})$ $(use\ assms\ in\ simp\text{-}all)$
finally show $?thesis$.
qed

lemma $(in\ landau\text{-}pair)\ big\text{-}power$:
assumes $f \in L\ F\ g$
shows $(\lambda x. f\ x \wedge n) \in L\ F\ (\lambda x. g\ x \wedge n)$
using $big\text{-}prod[of\ \{..<n\}\ \lambda\cdot. f\ F\ \lambda\cdot. g]\ assms$ **by** $simp$

lemma $(in\ landau\text{-}pair)\ small\text{-}power$:
assumes $f \in l\ F\ g\ n > 0$
shows $(\lambda x. f\ x \wedge n) \in l\ F\ (\lambda x. g\ x \wedge n)$
using $assms(2,1)$
by $(induction\ rule:\ nat\text{-}induct\text{-}non\text{-}zero)\ (auto\ intro!:\ small.\text{mult})$

lemma $pairwise\text{-}imp\text{-}disjoint\text{-}family\text{-}on$:
assumes $pairwise\ R\ A$

assumes $\bigwedge m n. m \in A \implies n \in A \implies R m n \implies f m \cap f n = \{\}$
shows *disjoint-family-on f A*
using *assms*
unfolding *disjoint-family-on-def pairwise-def* **by** *blast*

lemma (in *comm-monoid-set*) *If-eq*:

assumes $y \in A$ *finite A*
shows $F (\lambda x. g x (if x = y then h1 x else h2 x)) A = f (g y (h1 y)) (F (\lambda x. g x (h2 x)) (A - \{y\}))$
proof –
have $F (\lambda x. g x (if x = y then h1 x else h2 x)) A = f (g y (h1 y)) (F (\lambda x. g x (if x = y then h1 x else h2 x)) (A - \{y\}))$
using *assms* **by** (*subst remove[of - y]*) *auto*
also have $F (\lambda x. g x (if x = y then h1 x else h2 x)) (A - \{y\}) = F (\lambda x. g x (h2 x)) (A - \{y\})$
by (*intro cong*) *auto*
finally show *?thesis* **by** *simp*
qed

lemma *prod-nonzeroI*:

fixes $f :: 'a \Rightarrow 'b :: \{semiring-no-zero-divisors, comm-semiring-1\}$
assumes $\bigwedge x. x \in A \implies f x \neq 0$
shows $prod f A \neq 0$
using *assms* **by** (*induction rule: infinite-finite-induct*) *auto*

lemma *frequently-prime-cofinite*: *frequently (prime :: nat \Rightarrow bool) cofinite*
unfolding *INFM-nat-le* **by** (*meson bigger-prime less-imp-le*)

lemma *frequently-eventually-mono*:

assumes *frequently Q F eventually P F* $\bigwedge x. P x \implies Q x \implies R x$
shows *frequently R F*
proof (*rule frequently-mp[OF - assms(1)]*)
show *eventually* ($\lambda x. Q x \longrightarrow R x$) *F*
using *assms(2)* **by** *eventually-elim (use assms(3) in blast)*
qed

lemma *bij-betw-Diff*:

assumes *bij-betw f A B* *bij-betw f A' B'* $A' \subseteq A$ $B' \subseteq B$
shows *bij-betw f (A - A') (B - B')*
unfolding *bij-betw-def*
proof
have *inj-on f A*
using *assms(1)* **by** (*auto simp: bij-betw-def*)
thus *inj-on f (A - A')*
by (*rule inj-on-subset*) *auto*
have $f \text{ ` } (A - A') = f \text{ ` } A - f \text{ ` } A'$
by (*intro inj-on-image-set-diff[OF <inj-on f A>]*) (*use <A' \subseteq A> in auto*)
also have $\dots = B - B'$
using *assms(1,2)* **by** (*auto simp: bij-betw-def*)

finally show $f'(A - A') = B - B'$.
qed

lemma *bij-betw-singleton*: $\text{bij-betw } f \{x\} \{y\} \iff f x = y$
by (*auto simp: bij-betw-def*)

end

8 The Hermite–Lindemann–Weierstraß Transcendence Theorem

theory *Hermite-Lindemann*

imports

Pi-Transcendental.Pi-Transcendental
Algebraic-Numbers.Algebraic-Numbers
Algebraic-Integer-Divisibility
Min-Int-Poly
Complex-Lexorder
More-Polynomial-HLW
More-Multivariate-Polynomial-HLW
More-Algebraic-Numbers-HLW
Misc-HLW

begin

The Hermite–Lindemann–Weierstraß theorem answers questions about the transcendence of the exponential function and other related complex functions. It proves that a large number of combinations of exponentials is always transcendental.

A first (much weaker) version of the theorem was proven by Hermite. Lindemann and Weierstraß then successively generalised it shortly afterwards, and finally Baker gave another, arguably more elegant formulation (which is the one that we will prove, and then derive the traditional version from it).

To honour the contributions of all three of these 19th-century mathematicians, I refer to the theorem as the Hermite–Lindemann–Weierstraß theorem, even though in other literature it is often called Hermite–Lindemann or Lindemann–Weierstraß. To keep things short, the Isabelle name of the theorem, however, will omit Weierstraß’s name.

8.1 Main proof

Following Baker, We first prove the following special form of the theorem: Let $m > 0$ and $q_1, \dots, q_m \in \mathbb{Z}[X]$ be irreducible, non-constant, and pairwise

coprime polynomials. Let β_1, \dots, β_m be non-zero integers. Then

$$\sum_{i=1}^m \beta_i \sum_{q_i(\alpha)=0} e^\alpha \neq 0$$

The difference to the final theorem is that

1. The coefficients β_i are non-zero integers (as opposed to arbitrary algebraic numbers)
2. The exponents α_i occur in full sets of conjugates, and each set has the same coefficient.

In a similar fashion to the proofs of the transcendence of e and π , we define some number J depending on the α_i and β_i and an arbitrary sufficiently large prime p . We then show that, on one hand, J is an integer multiple of $(p-1)!$, but on the other hand it is bounded from above by a term of the form $C_1 \cdot C_2^p$. This is then clearly a contradiction if p is chosen large enough.

lemma *Hermite-Lindemann-aux1:*

fixes $P :: \text{int poly set}$ **and** $\beta :: \text{int poly} \Rightarrow \text{int}$

assumes *finite P and* $P \neq \{\}$

assumes *distinct: pairwise Rings.coprime P*

assumes *irred:* $\bigwedge p. p \in P \Rightarrow \text{irreducible } p$

assumes *nonconstant:* $\bigwedge p. p \in P \Rightarrow \text{Polynomial.degree } p > 0$

assumes $\beta\text{-nz:}$ $\bigwedge p. p \in P \Rightarrow \beta p \neq 0$

defines $\text{Roots} \equiv (\lambda p. \{\alpha :: \text{complex. poly (of-int-poly } p) \alpha = 0\})$

shows $(\sum p \in P. \text{of-int } (\beta p) * (\sum \alpha \in \text{Roots } p. \text{exp } \alpha)) \neq 0$

proof

note $[\text{intro}] = \langle \text{finite } P \rangle$

assume sum-eq-0: $(\sum p \in P. \text{of-int } (\beta p) * (\sum \alpha \in \text{Roots } p. \text{exp } \alpha)) = 0$

define Roots' **where** $\text{Roots}' = (\bigcup p \in P. \text{Roots } p)$

have $\text{finite-Roots } [\text{intro}]:$ $\text{finite } (\text{Roots } p)$ **if** $p \in P$ **for** p

using $\text{nonconstant}[of p]$ **that** **by** $(\text{auto intro: poly-roots-finite simp: Roots-def})$

have $[\text{intro}]:$ $\text{finite } \text{Roots}'$

by $(\text{auto simp: Roots'-def})$

have $[\text{simp}]:$ $0 \notin P$

using $\text{nonconstant}[of 0]$ **by** auto

have $[\text{simp}]:$ $p \neq 0$ **if** $p \in P$ **for** p

using that **by** auto

The polynomials in P do not have multiple roots:

have rsquarefree: $\text{rsquarefree (of-int-poly } q :: \text{complex poly)}$ **if** $q \in P$ **for** q

by $(\text{rule irreducible-imp-rsquarefree-of-int-poly})$ $(\text{use that in } \langle \text{auto intro: irred nonconstant} \rangle)$

No two different polynomials in P have roots in common:

have *disjoint*: *disjoint-family-on* *Roots P*
using *distinct*
proof (*rule pairwise-imp-disjoint-family-on*)
fix *p q* **assume** *P*: *p* ∈ *P* *q* ∈ *P* **and** *Rings.coprime p q*
hence *Rings.coprime (of-int-poly p :: complex poly) (of-int-poly q)*
by (*intro coprime-of-int-polyI*)
thus *Roots p* ∩ *Roots q* = {}
using *poly-eq-0-coprime[of of-int-poly p of-int-poly q :: complex poly] P*
by (*auto simp: Roots-def*)
qed

define *n-roots* :: *int poly* ⇒ *nat* (#-)
where *n-roots* = (λ*p*. *card (Roots p)*)
define *n* **where** *n* = (∑ *p* ∈ *P*. #*p*)
have *n-altdef*: *n* = *card Roots'*
unfolding *n-def Roots'-def n-roots-def* **using** *disjoint*
by (*subst card-UN-disjoint*) (*auto simp: disjoint-family-on-def*)
have *Roots-nonempty*: *Roots p* ≠ {} **if** *p* ∈ *P* **for** *p*
using *nonconstant[OF that]* **by** (*auto simp: Roots-def fundamental-theorem-of-algebra*
constant-degree)
have *Roots' ≠ {}*
using *Roots-nonempty* (*P* ≠ {}) **by** (*auto simp: Roots'-def*)
have *n > 0*
using (*Roots' ≠ {}*) (*finite Roots'*) **by** (*auto simp: n-altdef*)

We can split each polynomial in P into a product of linear factors:

have *of-int-poly-P*:
of-int-poly q = *Polynomial.smult (Polynomial.lead-coeff q) (∏ *x* ∈ *Roots* *q*. [:-*x*, 1:])*
if *q* ∈ *P* **for** *q*
using *complex-poly-decompose-rsquarefree[OF rsquarefree[OF that]]* **by** (*simp*
add: Roots-def)

We let l be an integer such that $l\alpha$ is an algebraic integer for all our roots α :

define *l* **where** *l* = (*LCM q* ∈ *P*. *Polynomial.lead-coeff q*)
have *alg-int*: *algebraic-int (of-int l * x)* **if** *x* ∈ *Roots'* **for** *x*
proof –
from *that* **obtain** *q* **where** *q*: *q* ∈ *P* *ipoly q x* = 0
by (*auto simp: Roots'-def Roots-def*)
show *?thesis*
by (*rule algebraic-imp-algebraic-int'[of q]*) (*use q in* (*auto simp: l-def*))
qed
have *l* ≠ 0
using (*finite P*) **by** (*auto simp: l-def Lcm-0-iff*)
moreover **have** *l* ≥ 0
unfolding *l-def* **by** (*rule Lcm-int-greater-eq-0*)
ultimately **have** *l* > 0 **by** *linarith*

We can split the product of all the polynomials in P into linear factors:

```

define lc-factor where lc-factor = ( $\prod_{q \in P}. l \wedge \text{Polynomial.degree } q \text{ div } \text{Polynomial.lead-coeff } q$ )
have lc-factor:  $\text{Polynomial.smult } (\text{of-int } l \wedge n) (\prod_{\alpha' \in \text{Roots}'}. [:-\alpha', 1:]) =$ 
   $\text{of-int-poly } (\text{Polynomial.smult } \text{lc-factor } (\prod P))$ 
proof –
  define lc where lc = ( $\lambda q. \text{Polynomial.lead-coeff } q :: \text{int}$ )
  define d where d = ( $\text{Polynomial.degree} :: \text{int poly} \Rightarrow \text{nat}$ )
  have ( $\prod_{q \in P}. \text{of-int-poly } q$ ) =
    ( $\prod_{q \in P}. \text{Polynomial.smult } (\text{lc } q) (\prod_{x \in \text{Roots } q}. [:-x, 1:]) :: \text{complex poly}$ )
    unfolding lc-def by (intro prod.cong of-int-poly-P refl)
  also have  $\dots = \text{Polynomial.smult } (\prod_{q \in P}. \text{lc } q) (\prod_{q \in P}. (\prod_{x \in \text{Roots } q}. [:-x, 1:]))$ 
  by (simp add: prod-smult)
  also have ( $\prod_{q \in P}. (\prod_{x \in \text{Roots } q}. [:-x, 1:])$ ) = ( $\prod_{x \in \text{Roots}'}. [:-x, 1:]$ )
    unfolding Roots'-def using disjoint
  by (intro prod.UNION-disjoint [symmetric]) (auto simp: disjoint-family-on-def)
  also have  $\text{Polynomial.smult } (\text{of-int } \text{lc-factor}) (\text{Polynomial.smult } (\prod_{q \in P}. \text{lc } q) \dots) =$ 
     $\text{Polynomial.smult } (\prod_{q \in P}. \text{of-int } (l \wedge d \text{ } q \text{ div } \text{lc } q * \text{lc } q)) (\prod_{x \in \text{Roots}'}. \text{pCons } (- \ x) \ 1)$ 
    by (simp add: lc-factor-def prod.distrib lc-def d-def)
  also have ( $\prod_{q \in P}. \text{of-int } (l \wedge d \text{ } q \text{ div } \text{lc } q * \text{lc } q)$ ) = ( $\prod_{q \in P}. \text{of-int } l \wedge d \text{ } q :: \text{complex}$ )
  proof (intro prod.cong, goal-cases)
    case ( $2 \text{ } q$ )
    have lc q dvd l
      unfolding l-def lc-def using  $2$  by auto
    also have  $\dots \text{ } \text{dvd } l \wedge d \text{ } q$ 
      using  $2$  nonconstant[of q] by (intro dvd-power) (auto simp: d-def)
    finally show ?case by simp
  qed auto
  also have  $\dots = l \wedge (\sum_{q \in P}. d \text{ } q)$ 
    by (simp add: power-sum)
  also have ( $\sum_{q \in P}. d \text{ } q$ ) = ( $\sum_{q \in P}. \text{n-roots } q$ )
  proof (intro sum.cong, goal-cases)
    case ( $2 \text{ } q$ )
    thus ?case using rsquarefree[OF 2]
      by (subst (asm) rsquarefree-card-degree) (auto simp: d-def n-roots-def Roots-def)
  qed auto
  also have  $\dots = n$ 
    by (simp add: n-def)
  finally show ?thesis
    by (simp add: of-int-hom.map-poly-hom-smult of-int-poly-hom.hom-prod)
  qed

```

We define R to be the radius of the smallest circle around the origin in which all our roots lie:

```

define  $R :: \text{real}$  where  $R = \text{Max} (\text{norm} \text{ ' } \text{Roots}' )$ 
have  $R\text{-ge}: R \geq \text{norm } \alpha$  if  $\alpha \in \text{Roots}'$  for  $\alpha$ 
  unfolding  $R\text{-def}$  using that by (intro Max-ge) auto
have  $R \geq 0$ 
proof –
  from  $\langle \text{Roots}' \neq \{\} \rangle$  obtain  $\alpha$  where  $\alpha \in \text{Roots}'$ 
  by auto
  have  $0 \leq \text{norm } \alpha$ 
  by simp
  also have  $\dots \leq R$ 
  by (intro R-ge) fact
  finally show  $R \geq 0$ 
  by simp
qed

```

Now the main part of the proof: for any sufficiently large prime p , our assumptions imply $(p-1)!^n \leq C' l^{np} (2R)^{np-1}$ for some constant C' :

```

define  $C :: \text{nat} \Rightarrow \text{real}$  where  $C = (\lambda p. l \wedge (n * p) * (2 * R) \wedge (n * p - 1))$ 
define  $C'$  where
   $C' = (\prod_{x \in \text{Roots}'}. \sum_{q \in P}. \text{real-of-int } |\beta \ q| * (\sum_{\alpha \in \text{Roots}} q. \text{cmod } \alpha * \exp (\text{cmod } \alpha)))$ 

```

We commence with the proof of the main inequality.

```

have  $\text{ineq}: \text{fact } (p - 1) \wedge n \leq C' * C \ p \wedge n$ 
  if  $p$ : prime  $p$ 
  and  $p\text{-ineqs}: \forall q \in P. p > |\beta \ q|$ 
     $\text{real } p > \text{norm} (\prod_{\alpha \in \text{Roots}'}. \text{of-int } (l \wedge n) * (\prod_{x \in \text{Roots}' - \{\alpha\}}. \alpha - x))$ 
  for  $p :: \text{nat}$ 
proof –
  have  $p > 1$ 
  using prime-gt-1-nat[OF p] .

```

We define the polynomial function

$$f_i(X) = l^{np} \frac{\prod_{\alpha} (X - \alpha)^p}{X - \alpha_i}$$

where the product runs over all roots α .

```

define  $f\text{-poly} :: \text{complex} \Rightarrow \text{complex poly}$  where
   $f\text{-poly} = (\lambda \alpha. \text{Polynomial.smult } (l \wedge (n * p)) ((\prod_{\alpha' \in \text{Roots}'}. [:-\alpha', 1:] \wedge p) \text{ div } [:-\alpha, 1:]))$ 
have  $f\text{-poly-altdef}: f\text{-poly } \alpha = \text{Polynomial.smult } (l \wedge (n * p))$ 
   $((\prod_{\alpha' \in \text{Roots}'}. [:-\alpha', 1:] \wedge (\text{if } \alpha' = \alpha \text{ then } p - 1 \text{ else } p)))$ 
  if  $\alpha \in \text{Roots}'$  for  $\alpha$ 
proof –
  have  $(\prod_{\alpha' \in \text{Roots}'}. [:-\alpha', 1:] \wedge (\text{if } \alpha' = \alpha \text{ then } p - 1 \text{ else } p)) * [:-\alpha, 1:] =$ 
   $[:-\alpha, 1:] \wedge (p - 1) * (\prod_{x \in \text{Roots}' - \{\alpha\}}. [:-x, 1:] \wedge p) * [:-\alpha, 1:]$ 
  using that by (subst prod.If-eq) (auto simp: algebra-simps)
  also have  $\dots = (\prod_{x \in \text{Roots}' - \{\alpha\}}. [:-x, 1:] \wedge p) * [:-\alpha, 1:] \wedge \text{Suc } (p - 1)$ 

```

```

    by (simp only: power-Suc mult-ac)
    also have Suc (p - 1) = p
    using ⟨p > 1⟩ by auto
    also have (∏ x∈Roots' - {α}. [- x, 1:] ^ p) * [- α, 1:] ^ p = (∏ x∈Roots'.
[- x, 1:] ^ p)
    using that by (subst prod.remove[of - α]) auto
    finally have eq: (∏ α'∈Roots'. [-α', 1:] ^ (if α'=α then p-1 else p)) * [-α,
1:] =
        (∏ x∈Roots'. [- x, 1:] ^ p) .
    show ?thesis
    unfolding f-poly-def eq[symmetric] by (subst nonzero-mult-div-cancel-right)
auto
qed

```

```

define f :: complex ⇒ complex ⇒ complex
  where f = (λα x. l^(n*p) * (∏ α'∈Roots'. (x - α') ^ (if α' = α then p - 1
else p)))
  have eval-f: poly (f-poly α) x = f α x if α ∈ Roots' for α x
  using that by (simp add: f-poly-altdef poly-prod f-def)
  have deg-f: Polynomial.degree (f-poly α) = n * p - 1 if α ∈ Roots' for α
  proof -
    have Polynomial.degree (f-poly α) = p - 1 + (n - 1) * p
    unfolding f-poly-altdef[OF that] using that ⟨l > 0⟩ ⟨finite Roots'⟩
    by (subst prod.If-eq) (auto simp: degree-prod-eq degree-power-eq degree-mult-eq
n-altdef)
    also have p - 1 + (n - 1) * p = n * p - 1
    using ⟨n > 0⟩ ⟨p > 1⟩ by (cases n) auto
    finally show ?thesis .
  qed

```

Next, we define the function $I_i(z) = \int_0^z e^{z-t} f_i(t) dt$, and, based on that, the numbers $J_i = \sum_{i=1}^m \beta_i \sum_{q_i(x)=0} I_i(x)$, and the number J , which is the product of all the J_i :

```

define I :: complex ⇒ complex ⇒ complex
  where I = (λα x. lindemann-weierstrass-aux.I (f-poly α) x)
define J :: complex ⇒ complex
  where J = (λα. ∑ q∈P. β q * (∑ x∈Roots q. I α x))

define J' :: complex
  where J' = (∏ α∈Roots'. J α)

```

Reusing some of the machinery from the proof that e is transcendental, we find the following equality for J_i :

```

  have J-eq: J α = -(∑ q∈P. of-int (β q) * (∑ x∈Roots q. ∑ j<n*p. poly
((pderiv ^ j) (f-poly α)) x))
  if α ∈ Roots' for α
  proof -
    have n * p ≥ 1 * 2
    using ⟨n > 0⟩ ⟨p > 1⟩ by (intro mult-mono) auto

```

hence $[simp]: \{..n*p - Suc\ 0\} = \{..<n*p\}$
by *auto*
have $J\ \alpha = (\sum q \in P. \beta\ q * (\sum x \in Roots\ q. I\ \alpha\ x))$
unfolding *J-def ..*
also have $\dots = (\sum q \in P. of-int\ (\beta\ q) * (\sum x \in Roots\ q. exp\ x * (\sum j < n*p. poly\ ((pderiv\ \hat{\sim} j)\ (f-poly\ \alpha))\ 0))) -$
 $(\sum q \in P. of-int\ (\beta\ q) * (\sum x \in Roots\ q. \sum j < n*p. poly\ ((pderiv\ \hat{\sim} j)\ (f-poly\ \alpha))\ x))$
unfolding *I-def lindemann-weierstrass-aux.I-def*
by *(simp add: deg-f that ring-distrib sum-subtractf sum-distrib-left sum-distrib-right mult-ac)*
also have $\dots = -(\sum q \in P. of-int\ (\beta\ q) * (\sum x \in Roots\ q. \sum j < n*p. poly\ ((pderiv\ \hat{\sim} j)\ (f-poly\ \alpha))\ x))$
unfolding *sum-distrib-right [symmetric] mult.assoc [symmetric] sum-eq-0*
by *simp*
finally show *?thesis .*
qed

The next big step is to show that $(p-1)! \mid J_i$ as an algebraic integer (i.e. $J_i/(p-1)!$ is an algebraic integer), but $p \nmid J_i$. This is done by brute force: We show that every summand in the above sum has $p!$ as a factor, except for the one corresponding to $x = \alpha_i, j = p-1$, which has $(p-1)!$ as a factor but not p .

have *J: fact (p - 1) alg-dvd J alpha -of-nat p alg-dvd J alpha if alpha: alpha in Roots' for alpha*
proof -
define *h where h = (lambda' j. poly ((pderiv ~ j) (f-poly alpha)) alpha')*
from *alpha obtain q where q: q in P alpha in Roots q*
by *(auto simp: Roots'-def)*

have $J\ \alpha = -(\sum (q, \alpha') \in Sigma\ P\ Roots. \sum j < n*p. of-int\ (\beta\ q) * h\ \alpha'\ j)$
unfolding *J-eq[OF alpha] h-def sum-distrib-left* **by** *(subst (2) sum.Sigma) auto*
also have $\dots = -(\sum ((q, \alpha'), i) \in Sigma\ P\ Roots \times \{..<n*p\}. of-int\ (\beta\ q) * h\ \alpha'\ i)$
by *(subst (2) sum.Sigma [symmetric]) (auto simp: case-prod-unfold)*
finally have *J-eq': J alpha = - (sum ((q, alpha'), i) in Sigma P Roots x {..<n * p}. of-int (beta q) * h alpha' i) .*

have *h-alpha-pm1-eq: h alpha (p-1) = of-int (l^(n*p)) * fact (p-1) * (prod alpha' in Roots' - {alpha}. (alpha - alpha') ^ p)*
proof -
have $h\ \alpha\ (p-1) = of-int\ (l\ \hat{\wedge}\ (n * p)) * poly\ ((pderiv\ \hat{\sim}\ (p-1))\ (prod\ \alpha' \in Roots'. [:-alpha', 1:] \hat{\wedge}\ (if\ \alpha' = \alpha\ then\ p - 1\ else\ p)))\ \alpha$
unfolding *h-def f-poly-altdef[OF alpha] higher-pderiv-smult poly-smult ..*
also have $(prod\ \alpha' \in Roots'. [:-alpha', 1:] \hat{\wedge}\ (if\ \alpha' = \alpha\ then\ p - 1\ else\ p)) = [:-alpha, 1:] \hat{\wedge}\ (p-1) * (prod\ \alpha' \in Roots' - \{alpha\}. [:-alpha', 1:] \hat{\wedge}\ p)$
using *alpha* **by** *(subst prod.If-eq) auto*
also have $poly\ ((pderiv\ \hat{\sim}\ (p-1))\ \dots)\ \alpha = fact\ (p-1) * (prod\ \alpha' \in Roots' - \{alpha\}. (\alpha - \alpha') \hat{\wedge}\ p)$

by (subst poly-higher-pderiv-ax2) (simp-all add: poly-prod)
 finally show ?thesis by (simp only: mult.assoc)
 qed

have fact (p-1) alg-dvd h α (p-1)
 proof -
 have fact (p-1) alg-dvd fact (p-1) * (of-int (l \hat{p}) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$.
 (l* $\alpha - l*\alpha')$ \hat{p}))
 by (intro alg-dvd-triv-left algebraic-int-times[of of-int (l \hat{p})]
 algebraic-int-prod algebraic-int-power algebraic-int-diff
 alg-int α algebraic-int-of-int) auto
 also have ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. (l* $\alpha - l*\alpha')$ \hat{p}) = ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. of-int
 l \hat{p} * ($\alpha - \alpha')$ \hat{p})
 by (subst power-mult-distrib [symmetric]) (simp-all add: algebra-simps)
 also have ... = of-int (l \hat{p} * (n-1)) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. ($\alpha - \alpha')$ \hat{p})
 using α by (subst prod.distrib) (auto simp: card-Diff-subset n-altdef simp
 flip: power-mult)
 also have of-int (l \hat{p}) * ... = of-int (l \hat{p} * (n-1)) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$.
 ($\alpha - \alpha')$ \hat{p})
 unfolding mult.assoc [symmetric] power-add [symmetric] of-int-power ..
 also have p + p * (n - 1) = n * p
 using (n > 0) by (cases n) (auto simp: mult-ac)
 also have fact (p - 1) * (of-int (l \hat{p} * (n-1))) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. ($\alpha - \alpha')$ \hat{p})
 = h α (p-1)
 unfolding h- α -pm1-eq by (simp add: mult-ac)
 finally show ?thesis .
 qed

have \neg of-nat p alg-dvd of-int (β q) * h α (p-1)
 unfolding h- α -pm1-eq mult.assoc [symmetric] of-int-mult [symmetric]
 proof
 define r where r = ($\lambda \alpha$. of-int (l \hat{n}) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. $\alpha - \alpha'$))
 have alg-int-r: algebraic-int (r α) if $\alpha \in \text{Roots}'$ for α
 proof -
 have algebraic-int (of-int l * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. of-int l * $\alpha -$ of-int l *
 α'))
 by (intro algebraic-int-times[OF algebraic-int-of-int] algebraic-int-prod
 algebraic-int-power algebraic-int-diff alg-int that) auto
 also have ... = of-int l * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. of-int l * ($\alpha - \alpha'$))
 by (simp add: algebra-simps flip: power-mult-distrib)
 also have ... = of-int (l \hat{p} * (n-1)) * ($\prod \alpha' \in \text{Roots}' - \{\alpha\}$. $\alpha - \alpha'$)
 using that by (simp add: r-def prod.distrib card-Diff-subset
 n-altdef power-add mult-ac flip: power-mult)
 also have 1 + (n - 1) = n
 using (n > 0) by auto
 finally show algebraic-int (r α)
 unfolding r-def .
 qed

```

have ( $\prod \alpha' \in \text{Roots}' . r \alpha' \in \mathbb{Q}$ )
proof -
  obtain Root where Root-bij: bij-betw Root  $\{..<n\}$  Roots'
    using ex-bij-betw-nat-finite[OF  $\langle \text{finite } \text{Roots}' \rangle$ ] unfolding n-altdef
atLeast0LessThan by metis
  have Root-in-Roots': Root i  $\in \text{Roots}'$  if i  $< n$  for i
    using Root-bij that by (auto simp: bij-betw-def)

define R :: complex mpoly where
  R = ( $\prod i < n . \text{Const } (\text{of-int } (l \wedge n)) * (\prod j \in \{..<n\} - \{i\} . \text{Var } i - \text{Var } j)$ )
have insertion Root R  $\in \mathbb{Q}$ 
proof (rule symmetric-poly-of-roots-in-subring)
  show symmetric-mpoly  $\{..<n\}$  R
    unfolding R-def
  proof (rule symmetric-mpoly-symmetric-prod'[of -  $\lambda \pi . \pi$ ], goal-cases)
    case ( $2 i \pi$ )
    from  $\langle \pi \text{ permutes } \{..<n\} \rangle$  have [simp]: bij  $\pi$ 
      by (rule permutes-bij)
    have mpoly-map-vars  $\pi$  (Const (of-int ( $l \wedge n$ )) *
      ( $\prod j \in \{..<n\} - \{i\} . \text{Var } i - \text{Var } j$ )) :: complex mpoly =
      Const (of-int ( $l \wedge n$ )) * ( $\prod j \in \{..<n\} - \{i\} . \text{Var } (\pi i) - \text{Var } (\pi j)$ )
      by simp
    also have ( $\prod j \in \{..<n\} - \{i\} . \text{Var } (\pi i) - \text{Var } (\pi j)$ ) =
      ( $\prod j \in \{..<n\} - \{\pi i\} . \text{Var } (\pi i) - \text{Var } j$ )
      using  $2 \text{ permutes-in-image}$ [OF  $2(2)$ , of i]
    by (intro prod.reindex-bij-betw bij-betw-Diff permutes-imp-bij[OF  $2(2)$ ])
      (auto simp: bij-betw-singleton)
    finally show ?case by simp
  qed
next
  show vars R  $\subseteq \{..<n\}$  unfolding R-def
    by (intro order.trans[OF vars-prod] UN-least order.trans[OF vars-mult]
      UN-least order.trans[OF vars-power] order.trans[OF vars-diff])
      (auto simp: vars-Var)
next
  show ring-closed ( $\mathbb{Q}$  :: complex set)
    by unfold-locales auto
  then interpret ring-closed  $\mathbb{Q}$  :: complex set .
  show  $\forall m . \text{MPoly-Type.coeff } R m \in \mathbb{Q}$ 
    unfolding R-def
    by (intro allI coeff-prod-closed coeff-mult-closed coeff-power-closed)
      (auto simp: mpoly-coeff-Const coeff-Var when-def)
next
  let ?lc = of-int ( $\prod p \in P . \text{Polynomial.lead-coeff } p$ ) :: complex
  have ( $\prod q \in P . \text{of-int-poly } q$ ) = ( $\prod q \in P . \text{Polynomial.smult}$ 
    (of-int (Polynomial.lead-coeff q)) ( $\prod x \in \text{Roots } q . [:-x, 1:]$ ))
    by (intro prod.cong of-int-poly-P refl)
  also have  $\dots = \text{Polynomial.smult } ?lc$  ( $\prod q \in P . \prod x \in \text{Roots } q . [:-x, 1:]$ )
    by (simp add: prod-smult)

```

also have $(\prod q \in P. \prod x \in \text{Roots } q. [-x, 1:]) = (\prod x \in \text{Roots}'. [-x, 1:])$
unfolding *Roots'-def* **using** *disjoint*
by (*intro prod.UNION-disjoint [symmetric]*) (*auto simp: disjoint-family-on-def*)
also have $\dots = (\prod i < n. [- \text{Root } i, 1:])$
by (*intro prod.reindex-bij-betw [symmetric] Root-bij*)
finally show *of-int-poly* $(\prod P) = \text{Polynomial.smult } ?lc (\prod i < n. [- \text{Root } i, 1:])$
by (*simp add: of-int-poly-hom.hom-prod*)
have *prod Polynomial.lead-coeff* $P \neq 0$
by (*intro prod-nonzeroI*) *auto*
thus *inverse ?lc * ?lc = 1* *inverse ?lc* $\in \mathbb{Q}$
by (*auto simp: field-simps simp flip: of-int-prod*)
qed auto
also have *insertion* $\text{Root } R = (\prod i < n. \text{of-int } (l \wedge n) * (\prod j \in \{..<n\} - \{i\}. \text{Root } i - \text{Root } j))$
by (*simp add: R-def insertion-prod insertion-mult insertion-power insertion-diff*)
also have $\dots = (\prod i < n. \text{of-int } (l \wedge n) * (\prod \alpha' \in \text{Roots}' - \{\text{Root } i\}. \text{Root } i - \alpha'))$
proof (*intro prod.cong, goal-cases*)
case (2 i)
hence $(\prod j \in \{..<n\} - \{i\}. \text{Root } i - \text{Root } j) = (\prod \alpha' \in \text{Roots}' - \{\text{Root } i\}. \text{Root } i - \alpha')$
*Root } i - \alpha')
by (*intro prod.reindex-bij-betw bij-betw-Diff Root-bij*)
(auto intro: Root-in-Roots' simp: bij-betw-singleton)
thus *?case by simp*
qed auto
also have $\dots = (\prod \alpha' \in \text{Roots}'. r \alpha')$
unfolding *r-def* **by** (*intro prod.reindex-bij-betw Root-bij*)
finally show $(\prod \alpha' \in \text{Roots}'. r \alpha') \in \mathbb{Q}$.
qed
moreover have *algebraic-int* $(\prod \alpha' \in \text{Roots}'. r \alpha')$
by (*intro algebraic-int-prod alg-int-r*)
ultimately have *is-int*: $(\prod \alpha' \in \text{Roots}'. r \alpha') \in \mathbb{Z}$
using *rational-algebraic-int-is-int* **by** *blast*
then obtain R' **where** $R': (\prod \alpha' \in \text{Roots}'. r \alpha') = \text{of-int } R'$
by (*elim Ints-cases*)
have $(\prod \alpha' \in \text{Roots}'. r \alpha') \neq 0$
using $(l > 0)$ **by** (*intro prod-nonzeroI*) (*auto simp: r-def (finite Roots')*)
with R' **have** *[simp]:* $R' \neq 0$
by *auto*
assume *of-nat p alg-dvd of-int* $(\beta q * l \wedge (n * p)) * \text{fact } (p-1) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. (\alpha - \alpha') \wedge p)$
also have $\dots = \text{of-int } (\beta q) * \text{fact } (p-1) * r \alpha \wedge p$
by (*simp add: r-def mult-ac power-mult-distrib power-mult prod-power-distrib*)
also have $\dots \text{alg-dvd of-int } (\beta q) * \text{fact } (p-1) * r \alpha \wedge p * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. r \alpha') \wedge p$*

```

      by (intro alg-dvd-triv-left algebraic-int-prod alg-int-r algebraic-int-power)
auto
  also have ... = of-int (β q) * fact (p-1) * (∏ α'∈Roots'. r α') ^ p
    using α by (subst (2) prod.remove[of - α]) (auto simp: mult-ac
power-mult-distrib)
  also have ... = of-int (β q * fact (p - 1) * R' ^ p)
    by (simp add: R')
  also have of-nat p = of-int (int p)
    by simp
  finally have int p dvd β q * fact (p - 1) * R' ^ p
    by (subst (asm) alg-dvd-of-int-iff)
  moreover have prime (int p)
    using ⟨prime p⟩ by auto
  ultimately have int p dvd β q ∨ int p dvd fact (p - 1) ∨ int p dvd R' ^ p
    by (simp add: prime-dvd-mult-iff)
  moreover have ¬int p dvd β q
  proof
    assume int p dvd β q
    hence int p ≤ |β q|
      using β-nz[of q] dvd-imp-le-int[of β q int p] q by auto
    with p-ineqs(1) q show False by auto
  qed
  moreover have ¬int p dvd fact (p - 1)
  proof -
    have ¬p dvd fact (p - 1)
      using ⟨p > 1⟩ p by (subst prime-dvd-fact-iff) auto
    hence ¬int p dvd int (fact (p - 1))
      by (subst int-dvd-int-iff)
    thus ?thesis unfolding of-nat-fact .
  qed
  moreover have ¬int p dvd R' ^ p
  proof
    assume int p dvd R' ^ p
    hence int p dvd R'
      using ⟨prime (int p)⟩ prime-dvd-power by metis
    hence int p ≤ |R'|
      using β-nz[of q] dvd-imp-le-int[of R' int p] q by auto
    hence real p ≤ real-of-int |R'|
      by linarith
    also have ... = norm (∏ α∈Roots'. r α)
      unfolding R' by simp
    finally show False unfolding r-def using p-ineqs(2)
      by linarith
  qed
  ultimately show False
    by blast
  qed

```

have fact-p-dvd: fact p alg-dvd h α' j if α' ∈ Roots' α' ≠ α ∨ j ≠ p - 1 for


```

 $\alpha' j$ 
proof (cases  $j \geq p$ )
  case False
    with that have  $j: j < (\text{if } \alpha' = \alpha \text{ then } p - 1 \text{ else } p)$ 
      by auto
    have  $h \alpha' j = 0$ 
      unfolding h-def f-poly-altdef[OF  $\alpha$ ]
      by (intro poly-higher-pderiv-aux1'[OF  $j$ ] dvd-smult dvd-prodI that) auto
    thus ?thesis by simp
  next
    case True
    define  $e$  where  $e = (\lambda x. \text{if } x = \alpha \text{ then } p - 1 \text{ else } p)$ 
    define  $Q$  where  $Q = (\prod_{x \in \text{Roots}'}. [-x, 1:] \wedge e x)$ 
    define  $Q'$  where  $Q' = \text{Polynomial.smult } (of\text{-int } (l \wedge^{(n * p + j)})) (pcompose Q$ 
 $[:0, 1 / of\text{-int } l:])$ 
    have  $poly ((pderiv \wedge^j) Q) \alpha' / l \wedge^j =$ 
 $poly ((pderiv \wedge^j) (pcompose Q [:0, 1 / of\text{-int } l:])) (l * \alpha')$ 
      using  $\langle l > 0 \rangle$  by (simp add: higher-pderiv-pcompose-linear poly-pcompose
field-simps)

    have  $sum e \text{Roots}' = (n - 1) * p + (p - 1)$ 
      unfolding e-def using  $\alpha$ 
      by (subst sum.If-eq) (auto simp: card-Diff-subset n-altdef algebra-simps)
    also have  $\dots = n * p - 1$ 
      using  $\langle n > 0 \rangle \langle p > 1 \rangle$  by (cases n) auto
    finally have [simp]:  $sum e \text{Roots}' = n * p - 1$  .

    have  $h \alpha' j = of\text{-int } (l \wedge^{(n * p)}) * poly ((pderiv \wedge^j) Q) \alpha'$ 
      unfolding h-def f-poly-altdef[OF  $\alpha$ ] higher-pderiv-smult poly-smult e-def
Q-def ..
    also have  $poly ((pderiv \wedge^j) Q) \alpha' =$ 
 $of\text{-int } l \wedge^j * poly ((pderiv \wedge^j) (pcompose Q [:0, 1 / of\text{-int } l:])) (l$ 
 $* \alpha')$ 
      using  $\langle l > 0 \rangle$  by (simp add: higher-pderiv-pcompose-linear poly-pcompose
field-simps)
    also have  $of\text{-int } (l \wedge^{(n * p)}) * \dots = poly ((pderiv \wedge^j) Q') (l * \alpha')$ 
      by (simp add: Q'-def higher-pderiv-smult power-add)
    also have fact p alg-dvd ...
    proof (rule fact-alg-dvd-poly-higher-pderiv)
      show  $j \geq p$  by fact
      show algebraic-int (of-int  $l * \alpha'$ )
        by (rule alg-int) fact
      interpret alg-int: ring-closed  $\{x :: \text{complex. algebraic-int } x\}$ 
        by standard auto
      show algebraic-int (poly.coeff  $Q' i$ ) for  $i$ 
      proof (cases  $i \leq \text{Polynomial.degree } Q'$ )
        case False
          thus ?thesis
          by (simp add: coeff-eq-0)

```

```

next
case True
hence  $i \leq n * p - 1$  using  $\langle l > 0 \rangle$ 
  by (simp add: Q'-def degree-prod-eq Q-def degree-power-eq)
also have  $n * p > 0$ 
  using  $\langle n > 0 \rangle \langle p > 1 \rangle$  by auto
hence  $n * p - 1 < n * p$ 
  by simp
finally have  $i: i < n * p$  .

have  $\text{poly.coeff } Q' i = \text{of-int } l^{(n * p + j)} / \text{of-int } l^i * \text{poly.coeff } Q i$ 
  by (simp add: Q'-def coeff-pcompose-linear field-simps)
also have  $\text{of-int } l^{(n * p + j)} = (\text{of-int } l^{(n * p + j - i)} :: \text{complex})$ 
*  $\text{of-int } l^i$ 
  unfolding power-add [symmetric] using  $i$  by simp
hence  $\text{of-int } l^{(n * p + j)} / \text{of-int } l^i = (\text{of-int } l^{(n * p + j - i)} ::$ 
complex)
  using  $\langle l > 0 \rangle$  by (simp add: field-simps)
also have  $\dots * \text{poly.coeff } Q i =$ 
 $(\sum X \in \{X. X \subseteq (\text{SIGMA } x:\text{Roots}'. \{..<e x\}) \wedge i = n * p - \text{Suc } (\text{card}$ 
X)\}.
 $\text{of-int } l^{(n * p + j - (n * p - \text{Suc } (\text{card } X)))} * ((-1)^{\text{card } X} *$ 
prod fst X))
  unfolding Q-def by (subst coeff-prod-linear-factors) (auto simp:
sum-distrib-left)
also have algebraic-int ...
proof (intro algebraic-int-sum, goal-cases)
case (1 X)
hence  $X: X \subseteq (\text{SIGMA } x:\text{Roots}'. \{..<e x\})$ 
  by auto
have card-eq:  $\text{card } (\text{SIGMA } x:\text{Roots}'. \{..<e x\}) = n * p - 1$ 
  by (subst card-SigmaI) auto
from X have  $\text{card } X \leq \text{card } (\text{SIGMA } x:\text{Roots}'. \{..<e x\})$ 
  by (intro card-mono) auto
hence  $\text{card } X \leq n * p - 1$ 
  using card-eq by auto
also have  $\dots < n * p$ 
  using  $\langle n * p > 0 \rangle$  by simp
finally have card-less:  $\text{card } X < n * p$  .
have algebraic-int  $((-1)^{\text{card } X} * \text{of-int } l^{(j + 1)} * (\prod x \in X. \text{of-int}$ 
 $l * \text{fst } x))$ 
  using X by (intro algebraic-int-times algebraic-int-prod alg-int) auto
thus ?case
  using card-less by (simp add: power-add prod.distrib mult-ac)
qed
finally show ?thesis .
qed
qed
finally show ?thesis .

```

qed

have $p\text{-dvd}$: $\text{of-nat } p \text{ alg-dvd } h \alpha' j$ if $\alpha' \in \text{Roots}' \alpha' \neq \alpha \vee j \neq p - 1$ for $\alpha' j$
proof –
 have $\text{of-nat } p \text{ alg-dvd}$ (of-nat ($\text{fact } p$) :: complex)
 by ($\text{intro alg-dvd-of-nat dvd-fact}$) ($\text{use } \langle p > 1 \rangle$ in auto)
 hence $\text{of-nat } p \text{ alg-dvd}$ ($\text{fact } p$:: complex)
 by simp
 also have ... $\text{alg-dvd } h \alpha' j$
 using that by (intro fact-p-dvd)
 finally show $?thesis$.
qed

show $\text{fact } (p - 1) \text{ alg-dvd } J \alpha$
 unfolding $J\text{-eq}'$
 proof ($\text{intro alg-dvd-uminus-right alg-dvd-sum}$, safe intro! : $\text{alg-dvd-mult alge-}$
 braic-int-of-int)
 fix $q \alpha' j$
 assume $q \in P \alpha' \in \text{Roots } q j < n * p$
 hence $\alpha' \in \text{Roots}'$
 by ($\text{auto simp: Roots'-def}$)
 show $\text{fact } (p - 1) \text{ alg-dvd } h \alpha' j$
 proof ($\text{cases } \alpha' = \alpha \wedge j = p - 1$)
 case True
 thus $?thesis$ using $\langle \text{fact } (p - 1) \text{ alg-dvd } h \alpha (p - 1) \rangle$
 by simp
 next
 case False
 have of-int ($\text{fact } (p - 1)$) alg-dvd (of-int ($\text{fact } p$) :: complex)
 by ($\text{intro alg-dvd-of-int fact-dvd}$) auto
 hence $\text{fact } (p - 1) \text{ alg-dvd}$ ($\text{fact } p$:: complex)
 by simp
 also have ... $\text{alg-dvd } h \alpha' j$
 using $\text{False } \langle \alpha' \in \text{Roots}' \rangle$ by (intro fact-p-dvd) auto
 finally show $?thesis$.
 qed
qed

show $\neg \text{of-nat } p \text{ alg-dvd } J \alpha$
 unfolding $J\text{-eq}' \text{ alg-dvd-uminus-right-iff}$
 proof ($\text{rule not-alg-dvd-sum}$)
 have $p - 1 < 1 * p$
 using $\langle p > 1 \rangle$ by simp
 also have $1 * p \leq n * p$
 using $\langle n > 0 \rangle$ by ($\text{intro mult-right-mono}$) auto
 finally show $((q, \alpha), p - 1) \in \text{Sigma } P \text{ Roots} \times \{..<n*p\}$
 using $q \langle n > 0 \rangle$ by auto
 next
 fix z assume z : $z \in \text{Sigma } P \text{ Roots} \times \{..<n*p\} - \{((q, \alpha), p - 1)\}$

```

from  $z$  have  $\text{snd } (fst\ z) \in \text{Roots}'$ 
  by (auto simp: Roots'-def)
moreover have  $\text{fst } (fst\ z) = q$  if  $\alpha \in \text{Roots } (fst\ (fst\ z))$ 
proof –
  have  $\alpha \in \text{Roots } (fst\ (fst\ z)) \cap \text{Roots } q$   $q \in P$   $\text{fst } (fst\ z) \in P$ 
  using that  $q\ z$  by auto
  with disjoint show ?thesis
  unfolding disjoint-family-on-def by blast
qed
ultimately have of-nat  $p$  alg-dvd  $h$  ( $\text{snd } (fst\ z)$ ) ( $\text{snd } z$ )
  using  $z$  by (intro p-dvd) auto
  thus of-nat  $p$  alg-dvd (case  $z$  of  $(x, xa) \Rightarrow (\text{case } x \text{ of } (q, \alpha') \Rightarrow \lambda i. \text{of-int } (\beta\ q) * h\ \alpha' i) xa$ )
  using  $z$  by auto
qed (use  $\langle$ of-nat  $p$  alg-dvd of-int  $(\beta\ q) * h\ \alpha\ (p-1)\rangle$  in auto)
qed

```

Our next goal is to show that J is rational. This is done by repeated applications of the fundamental theorem of symmetric polynomials, exploiting the fact that J is symmetric in all the α_i for each set of conjugates.

```

define  $g :: \text{int poly poly}$ 
  where  $g = \text{synthetic-div } (\text{map-poly } (\lambda x. [x:]))$ 
     $((\text{Polynomial.smult lc-factor } (\prod P)) \hat{\ } p) [0, 1:]$ 
have  $g: \text{map-poly } (\lambda p. \text{ipoly } p\ \alpha)$   $g = \text{f-poly } \alpha$  if  $\alpha: \alpha \in \text{Roots}'$  for  $\alpha$ 
proof –
  interpret  $\alpha: \text{comm-ring-hom } \lambda p. \text{ipoly } p\ \alpha$ 
  by standard (auto simp: of-int-hom.poly-map-poly-eval-poly of-int-poly-hom.hom-mult)
  define  $Q :: \text{int poly}$  where  $Q = (\text{Polynomial.smult lc-factor } (\prod P)) \hat{\ } p$ 
  have  $\text{f-poly } \alpha = \text{Polynomial.smult } (\text{of-int } (l \hat{\ } (n * p)))$   $((\prod \alpha' \in \text{Roots}'. [:-\alpha', 1:]) \hat{\ } p)$ 
div  $[:-\alpha, 1:]$ 
  unfolding f-poly-def div-smult-left [symmetric] prod-power-distrib[symmetric]
  ..
  also have  $\text{of-int } (l \hat{\ } (n * p)) = (\text{of-int } l \hat{\ } n) \hat{\ } p$ 
  by (simp add: power-mult)
  also have  $\text{Polynomial.smult } \dots ((\prod \alpha' \in \text{Roots}'. [:-\alpha', 1:]) \hat{\ } p) =$ 
     $(\text{Polynomial.smult } (\text{of-int } l \hat{\ } n) (\prod \alpha' \in \text{Roots}'. [:-\alpha', 1:])) \hat{\ } p$ 
  by (simp only: smult-power)
  also have  $\dots = \text{of-int-poly } Q$ 
  by (subst lc-factor) (simp-all add: Q-def of-int-poly-hom.hom-power)
  also have  $\dots \text{ div } [:-\alpha, 1:] = \text{synthetic-div } (\text{of-int-poly } Q)\ \alpha$ 
  unfolding synthetic-div-altdef ..
  also have  $\dots = \text{synthetic-div } (\text{map-poly } (\lambda p. \text{ipoly } p\ \alpha))$   $(\text{map-poly } (\lambda x. [x:]))$ 
   $Q))$  (ipoly  $[0, 1:]\ \alpha$ )
  by (simp add: map-poly-map-poly o-def)
  also have  $\dots = \text{map-poly } (\lambda p. \text{ipoly } p\ \alpha)$   $g$ 
  unfolding g-def Q-def by (rule  $\alpha.\text{synthetic-div-hom}$ )
  finally show ?thesis ..
qed

```

```

obtain  $Q$  where  $Q: J \alpha = -(\sum q \in P. \text{of-int } (\beta q) * \text{eval-poly of-rat } (Q q) \alpha)$ 
if  $\alpha \in \text{Roots}'$  for  $\alpha$ 
proof –
  define  $g' :: \text{nat} \Rightarrow \text{complex poly poly}$ 
    where  $g' = (\lambda j. (\text{map-poly of-int-poly } ((\text{pderiv } \sim j) g)))$ 
  obtain  $\text{root} :: \text{int poly} \Rightarrow \text{nat} \Rightarrow \text{complex}$ 
    where  $\text{root}: \bigwedge q. q \in P \implies \text{bij-betw } (\text{root } q) \{..<\#q\} (\text{Roots } q)$ 
  using  $\text{ex-bij-betw-nat-finite}[OF \text{finite-Roots}]$  unfolding  $n\text{-roots-def atLeast0LessThan}$ 
    by  $\text{metis}$ 
  have  $\exists Q'. \text{map-poly of-rat } Q' = (\sum x \in \text{Roots } q. \text{poly } (g' j) [x:])$  if  $q: q \in P$ 
for  $q j$ 
  proof –
    define  $Q :: \text{nat} \Rightarrow \text{complex poly mpoly}$ 
      where  $Q = (\lambda j. (\sum i < \#q. \text{mpoly-of-poly } i (g' j)))$ 
    define  $\text{ratpolys} :: \text{complex poly set}$  where  $\text{ratpolys} = \{p. \forall i. \text{poly.coeff } p i \in \mathbb{Q}\}$ 
    have  $\text{insertion } ((\lambda x. [x:]) \circ \text{root } q) (Q j) \in \text{ratpolys}$ 
    proof ( $\text{rule symmetric-poly-of-roots-in-subring}$ )
      show  $\text{ring-closed ratpolys}$ 
        by  $\text{standard } (\text{auto simp: ratpolys-def intro!: coeff-mult-semiring-closed})$ 
      show  $\forall m. \text{MPoly-Type.coeff } (Q j) m \in \text{ratpolys}$ 
        by ( $\text{auto simp: Q-def ratpolys-def Polynomial.coeff-sum coeff-mpoly-of-poly}$ 
           $\text{when-def } g'\text{-def}$ 
           $\text{intro!: sum-in-Rats}$ )
      show  $\text{vars } (Q j) \subseteq \{..<\#q\}$  unfolding  $Q\text{-def}$ 
    by ( $\text{intro order.trans}[OF \text{vars-sum}] \text{UN-least order.trans}[OF \text{vars-mpoly-of-poly}]$ )
    auto
    show  $\text{symmetric-mpoly } \{..<\#q\} (Q j)$  unfolding  $Q\text{-def}$ 
    by ( $\text{rule symmetric-mpoly-symmetric-sum}[of - id]$ ) ( $\text{auto simp: permutes-bij}$ )
    interpret  $\text{coeff-lift-hom: map-poly-idom-hom } \lambda x. [x:]$ 
      by  $\text{standard}$ 
    define  $lc :: \text{complex}$  where  $lc = \text{of-int } (\text{Polynomial.lead-coeff } q)$ 
    have  $\text{of-int-poly } q = \text{Polynomial.smult } (\text{Polynomial.lead-coeff } q) (\prod x \in \text{Roots}$ 
       $q. [:-x, 1:])$ 
      by ( $\text{rule of-int-poly-P}$ )  $\text{fact}$ 
    also have  $\text{poly-lift } \dots = \text{Polynomial.smult } [lc:] (\prod a \in \text{Roots } q. [:-[a:],$ 
       $1:])$ 
      by ( $\text{simp add: poly-lift-def map-poly-smult coeff-lift-hom.hom-prod lc-def}$ )
    also have  $(\prod a \in \text{Roots } q. [:-[a:], 1:]) = (\prod i < \#q. [:-[\text{root } q i:], 1:])$ 
      by ( $\text{intro prod.reindex-bij-betw } [\text{symmetric}] \text{root } q$ )
    also have  $\dots = (\prod i < \#q. [:-((\lambda x. [x:]) \circ \text{root } q) i, 1:])$ 
      by  $\text{simp}$ 
    finally show  $\text{poly-lift } (\text{Ring-Hom-Poly.of-int-poly } q) = \text{Polynomial.smult}$ 
       $[lc:] \dots$ 
    have  $lc \neq 0$ 
      using  $q$  by ( $\text{auto simp: lc-def}$ )
    thus  $[\text{inverse } lc:] * [lc:] = 1$ 
      by ( $\text{simp add: field-simps}$ )
    qed ( $\text{auto simp: ratpolys-def coeff-pCons split: nat.splits}$ )

```

also have $insertion ((\lambda x. [:x:]) \circ root\ q) (Q\ j) = (\sum i < \#q. poly\ (g'\ j)\ [:root\ q\ i:])$
by (*simp add: Q-def insertion-sum poly-sum*)
also have $\dots = (\sum x \in Roots\ q. poly\ (g'\ j)\ [:x:])$
by (*intro sum.reindex-bij-betw root q*)
finally have $\forall i. poly.coeff\ (\sum x \in Roots\ q. poly\ (g'\ j)\ [:x:])\ i \in \mathbb{Q}$
by (*auto simp: ratpolys-def*)
thus *?thesis*
using *ratpolyE* **by** *metis*
qed
then obtain Q **where** $Q: \bigwedge q\ j. q \in P \implies map\ poly\ of\ rat\ (Q\ q\ j) =$
 $(\sum x \in Roots\ q. poly\ (g'\ j)\ [:x:])$
by *metis*
define Q' **where** $Q' = (\lambda q. \sum j < n * p. Q\ q\ j)$

have $J\ \alpha = - (\sum q \in P. of\ int\ (\beta\ q) * eval\ poly\ of\ rat\ (Q'\ q)\ \alpha)$ **if** $\alpha: \alpha \in$
 $Roots'$ **for** α
proof –
have $J\ \alpha = - (\sum q \in P. of\ int\ (\beta\ q) * (\sum x \in Roots\ q. \sum j < n * p. poly\ ((pderiv$
 $\overset{\sim}{\sim} j)\ (f\ poly\ \alpha))\ x))$
(is $- = - ?S$) **unfolding** $J\ eq[OF\ \alpha]$ **..**
also have $?S = (\sum q \in P. of\ int\ (\beta\ q) * eval\ poly\ of\ rat\ (Q'\ q)\ \alpha)$
proof (*rule sum.cong, goal-cases*)
case $q: (2\ q)$
interpret $\alpha: idom\ hom\ \lambda p. ipoly\ p\ \alpha$
by *standard* (*auto simp: of-int-hom.poly-map-poly-eval-poly of-int-poly-hom.hom-mult*)

have $(\sum x \in Roots\ q. \sum j < n * p. poly\ ((pderiv\ \overset{\sim}{\sim} j)\ (f\ poly\ \alpha))\ x) =$
 $(\sum j < n * p. \sum x \in Roots\ q. poly\ ((pderiv\ \overset{\sim}{\sim} j)\ (f\ poly\ \alpha))\ x)$
by (*rule sum.swap*)
also have $\dots = (\sum j < n * p. eval\ poly\ of\ rat\ (Q\ q\ j)\ \alpha)$
proof (*rule sum.cong, goal-cases*)
case $j: (2\ j)$
have $(\sum x \in Roots\ q. poly\ ((pderiv\ \overset{\sim}{\sim} j)\ (f\ poly\ \alpha))\ x) =$
 $(\sum x \in Roots\ q. poly\ (poly\ (g'\ j)\ [:x:])\ \alpha)$
proof (*rule sum.cong, goal-cases*)
case $(2\ x)$
have $poly\ ((pderiv\ \overset{\sim}{\sim} j)\ (f\ poly\ \alpha))\ x =$
 $poly\ ((pderiv\ \overset{\sim}{\sim} j)\ (map\ poly\ (\lambda p. ipoly\ p\ \alpha)\ g))\ x$
by (*subst g[OF\ \alpha, symmetric]*) (*rule refl*)
also have $\dots = poly\ (eval\ poly\ ((\lambda p. [:poly\ p\ \alpha:]) \circ of\ int\ poly)\ ((pderiv$
 $\overset{\sim}{\sim} j)\ g)\ [:0, 1:]\ x)$
unfolding *o-def\ \alpha.map-poly-higher-pderiv* [*symmetric*]
by (*simp only: \alpha.map-poly-eval-poly*)
also have $\dots = poly\ (eval\ poly\ (\lambda p. [:poly\ p\ \alpha:]$
 $(map\ poly\ of\ int\ poly\ ((pderiv\ \overset{\sim}{\sim} j)\ g))\ [:0, 1:]\ x)$
unfolding *eval-poly-def* **by** (*subst map-poly-map-poly*) *auto*
also have $\dots = poly\ (poly\ (map\ poly\ of\ int\ poly\ ((pderiv\ \overset{\sim}{\sim} j)\ g))\ [:x:])$

α

```

    by (rule poly-poly-eq [symmetric])
  also have ... = poly (poly (g' j) [:x:])  $\alpha$ 
    by (simp add: g'-def)
  finally show ?case .
qed auto
also have ... = poly ( $\sum_{x \in \text{Roots } q} \text{poly } (g' j) [:x:]$ )  $\alpha$ 
  by (simp add: poly-sum)
also have ... = eval-poly of-rat (Q q j)  $\alpha$ 
  using q by (simp add: Q eval-poly-def)
  finally show ?case .
qed auto
also have ... = eval-poly of-rat (Q' q)  $\alpha$ 
  by (simp add: Q'-def of-rat-hom.eval-poly-sum)
  finally show ?case by simp
qed auto
finally show J  $\alpha$  = - ( $\sum_{q \in P} \text{of-int } (\beta q) * \text{eval-poly of-rat } (Q' q) \alpha$ ) .
qed
thus ?thesis using that[of Q'] by metis
qed

have J'  $\in \mathbb{Q}$ 
proof -
  have ( $\prod_{\alpha \in \text{Roots } q} J \alpha$ )  $\in \mathbb{Q}$  if q: q  $\in P$  for q
  proof -
    obtain root where root: bij-betw root {.. $\#q$ } (Roots q)
      using ex-bij-betw-nat-finite[OF finite-Roots[OF q]]
    unfolding atLeast0LessThan n-roots-def by metis
    define Q' :: complex poly
      where Q' = -( $\sum_{q \in P} \text{Polynomial.smult } (\text{of-int } (\beta q)) (\text{map-poly of-rat } (Q q))$ )
  have ( $\prod_{\alpha \in \text{Roots } q} J \alpha$ ) = ( $\prod_{\alpha \in \text{Roots } q} -(\sum_{q \in P} \text{of-int } (\beta q) * \text{eval-poly of-rat } (Q q) \alpha)$ )
    by (intro prod.cong refl Q) (auto simp: Roots'-def q)
  also have ... = ( $\prod_{\alpha \in \text{Roots } q} \text{poly } Q' \alpha$ )
    by (simp add: Q'-def poly-sum eval-poly-def)
  also have ... = ( $\prod_{i < \#q} \text{poly } Q' (\text{root } i)$ )
    by (intro prod.reindex-bij-betw [symmetric] root)
  also have ... = insertion root ( $\prod_{i < \#q} \text{mpoly-of-poly } i Q'$ )
    by (simp add: insertion-prod)
  also have ...  $\in \mathbb{Q}$ 
  proof (rule symmetric-poly-of-roots-in-subring)
    show ring-closed ( $\mathbb{Q} :: \text{complex set}$ )
      by standard auto
    then interpret Q: ring-closed  $\mathbb{Q} :: \text{complex set}$  .
    show  $\forall m. \text{MPoly-Type.coeff } (\prod_{i < \#q} \text{mpoly-of-poly } i Q') m \in \mathbb{Q}$ 
      by (auto intro!: Q.coeff-prod-closed sum-in-Rats
        simp: coeff-mpoly-of-poly when-def Q'-def Polynomial.coeff-sum)
  qed

```

```

show symmetric-mpoly {.. $\#q$ } ( $\prod i < \#q$ . mpoly-of-poly  $i$   $Q'$ )
  by (intro symmetric-mpoly-symmetric-prod'[of - id]) (auto simp: per-
mutes-bij)
show vars ( $\prod i < \#q$ . mpoly-of-poly  $i$   $Q'$ )  $\subseteq$  {.. $\#q$ }
  by (intro order.trans[OF vars-prod] order.trans[OF vars-mpoly-of-poly]
UN-least) auto
define lc where lc = (of-int (Polynomial.lead-coeff  $q$ ) :: complex)
have of-int-poly  $q$  = Polynomial.smult lc ( $\prod x \in \text{Roots } q$ . [:-  $x$ , 1:])
  unfolding lc-def by (rule of-int-poly-P) fact
also have ( $\prod x \in \text{Roots } q$ . [:-  $x$ , 1:]) = ( $\prod i < \#q$ . [:- root  $i$ , 1:])
  by (intro prod.reindex-bij-betw [symmetric] root)
finally show of-int-poly  $q$  = Polynomial.smult lc ( $\prod i < \#q$ . [:- root  $i$ , 1:]) .
  have lc  $\neq$  0
    using  $q$  by (auto simp: lc-def)
  thus inverse lc * lc = 1 inverse lc  $\in \mathbb{Q}$ 
    by (auto simp: lc-def)
qed auto
finally show ?thesis .
qed
hence ( $\prod q \in P$ .  $\prod \alpha \in \text{Roots } q$ .  $J \alpha$ )  $\in \mathbb{Q}$ 
  by (rule prod-in-Rats)
also have ( $\prod q \in P$ .  $\prod \alpha \in \text{Roots } q$ .  $J \alpha$ ) =  $J'$ 
  unfolding Roots'-def  $J'$ -def using disjoint
by (intro prod.UNION-disjoint [symmetric]) (auto simp: disjoint-family-on-def)
finally show  $J' \in \mathbb{Q}$  .
qed

```

Since J' is clearly an algebraic integer, we now know that it is in fact an integer.

```

moreover have algebraic-int  $J'$ 
  unfolding  $J'$ -def
proof (intro algebraic-int-prod)
  fix  $x$  assume  $x \in \text{Roots}'$ 
  hence fact ( $p - 1$ ) alg-dvd  $J x$ 
    by (intro  $J$ )
  thus algebraic-int ( $J x$ )
    by (rule alg-dvd-imp-algebraic-int) auto
qed
ultimately have  $J' \in \mathbb{Z}$ 
  using rational-algebraic-int-is-int by blast

```

It is also non-zero, as none of the J_i have p as a factor and such cannot be zero.

```

have  $J' \neq 0$ 
  unfolding  $J'$ -def
proof (intro prod-nonzeroI)
  fix  $\alpha$  assume  $\alpha \in \text{Roots}'$ 
  hence  $\neg$ of-nat  $p$  alg-dvd  $J \alpha$ 
    using  $J(2)$ [of  $\alpha$ ] by auto

```


thus $J \alpha \neq 0$
by *auto*
qed

It then clearly follows that $(p - 1)!^n \leq J$:

have $\text{fact } (p - 1) \wedge n \text{ alg-dvd } J'$
proof –
have $\text{fact } (p - 1) \wedge n = (\prod \alpha \in \text{Roots}'. \text{fact } (p - 1))$
by (*simp add: n-altdef*)
also have ... $\text{alg-dvd } J'$
unfolding J' -*def* **by** (*intro prod-alg-dvd-prod J(1)*)
finally show *?thesis* .
qed

have $\text{fact } (p - 1) \wedge n \leq \text{norm } J'$
proof –
from $\langle J' \in \mathbb{Z} \rangle$ **obtain** J'' **where** [*simp*]: $J' = \text{of-int } J''$
by (*elim Ints-cases*)
have $\text{of-int } (\text{fact } (p - 1) \wedge n) = (\text{fact } (p - 1) \wedge n :: \text{complex})$
by *simp*
also have ... $\text{alg-dvd } J'$
by *fact*
also have $J' = \text{of-int } J''$
by *fact*
finally have $\text{fact } (p - 1) \wedge n \text{ dvd } J''$
by (*subst (asm) alg-dvd-of-int-iff*)
moreover from $\langle J' \neq 0 \rangle$ **have** $J'' \neq 0$
by *auto*
ultimately have $|J''| \geq |\text{fact } (p - 1) \wedge n|$
by (*intro dvd-imp-le-int*)
hence $\text{real-of-int } |J''| \geq \text{real-of-int } |\text{fact } (p - 1) \wedge n|$
by *linarith*
also have $\text{real-of-int } |J''| = \text{norm } J'$
by *simp*
finally show *?thesis*
by *simp*
qed

The standard M-L bound for $I_i(x)$ shows the following inequality:

also have $\text{norm } J' \leq C' * C p \wedge n$
proof –
have $\text{norm } J' = (\prod x \in \text{Roots}'. \text{norm } (J x))$
unfolding J' -*def prod-norm [symmetric]* ..
also have ... $\leq (\prod x \in \text{Roots}'. \sum q \in P. \text{real-of-int } |\beta q| * (\sum \alpha \in \text{Roots } q. \text{cmod } \alpha * \exp (\text{cmod } \alpha) * C p))$
proof (*intro prod-mono conjI*)
fix x **assume** $x: x \in \text{Roots}'$
show $\text{norm } (J x) \leq (\sum q \in P. \text{real-of-int } |\beta q| * (\sum \alpha \in \text{Roots } q. \text{norm } \alpha * \exp (\text{norm } \alpha) * C p))$

```

unfolding J-def
proof (intro sum-norm-le)
  fix q assume  $q \in P$ 
  show  $\text{norm } (\text{of-int } (\beta \ q) * \text{sum } (I \ x) \ (\text{Roots } \ q)) \leq$ 
     $\text{real-of-int } |\beta \ q| * (\sum \alpha \in \text{Roots } \ q. \text{norm } \alpha * \text{exp } (\text{norm } \alpha) * C \ p)$ 
  unfolding norm-mult norm-of-int of-int-abs
proof (intro mult-left-mono sum-norm-le)
  fix  $\alpha$  assume  $\alpha \in \text{Roots } \ q$ 
  hence  $\alpha: \alpha \in \text{Roots}'$ 
  using  $\langle q \in P \rangle$  by (auto simp: Roots'-def)
  show  $\text{norm } (I \ x \ \alpha) \leq \text{norm } \alpha * \text{exp } (\text{norm } \alpha) * C \ p$ 
  unfolding I-def
proof (intro lindemann-weierstrass-aux.lindemann-weierstrass-integral-bound)
  fix t assume  $t \in \text{closed-segment } 0 \ \alpha$ 
  also have closed-segment  $0 \ \alpha \subseteq \text{cball } 0 \ R$ 
  using  $\langle R \geq 0 \rangle$  R-ge[OF  $\alpha$ ] by (intro closed-segment-subset) auto
  finally have  $\text{norm } t \leq R$  by simp

  have norm-diff-le:  $\text{norm } (t - y) \leq 2 * R$  if  $y \in \text{Roots}'$  for y
  proof -
    have  $\text{norm } (t - y) \leq \text{norm } t + \text{norm } y$ 
    by (meson norm-triangle-ineq4)
    also have  $\dots \leq R + R$ 
    by (intro add-mono[OF  $\langle \text{norm } t \leq R \rangle$  R-ge]) that
    finally show ?thesis by simp
  qed

  have  $\text{norm } (\text{poly } (f\text{-poly } \ x) \ t) =$ 
     $|\text{real-of-int } l| \wedge (n * p) * (\prod y \in \text{Roots}'. \text{cmod } (t - y) \wedge (\text{if } y = x$ 
  then  $p - 1$  else  $p))$ 
  by (simp add: eval-f x f-def norm-mult norm-power flip: prod-norm)
  also have  $\dots \leq |\text{real-of-int } l| \wedge (n * p) * (\prod y \in \text{Roots}'. (2 * R) \wedge (\text{if } y$ 
   $= x$  then  $p - 1$  else  $p))$ 
  by (intro mult-left-mono prod-mono conjI power-mono norm-diff-le)
  auto
  also have  $\dots = |\text{real-of-int } l| \wedge (n * p) * (2 \wedge (p - 1) * R \wedge (p - 1) * (2 \wedge p * R \wedge p) \wedge (n - 1))$ 
  using x by (subst prod.If-eq) (auto simp: card-Diff-subset n-altdef)
  also have  $2 \wedge (p - 1) * R \wedge (p - 1) * (2 \wedge p * R \wedge p) \wedge (n - 1) = (2 \wedge ((p - 1) + p * (n - 1)))$ 
   $* (R \wedge ((p - 1) + p * (n - 1)))$ 
  unfolding power-mult power-mult-distrib power-add by (simp add: mult-ac)
  also have  $(p - 1) + p * (n - 1) = p * n - 1$ 
  using  $\langle n > 0 \rangle \langle p > 1 \rangle$  by (cases n) (auto simp: algebra-simps)
  also have  $2 \wedge (p * n - 1) * R \wedge (p * n - 1) = (2 * R) \wedge (n * p - 1)$ 
  unfolding power-mult-distrib by (simp add: mult-ac)
  finally show  $\text{norm } (\text{poly } (f\text{-poly } \ x) \ t) \leq C \ p$ 
  unfolding C-def using  $\langle l > 0 \rangle$  by simp
qed (use  $\langle R \geq 0 \rangle \langle l > 0 \rangle$  in  $\langle \text{auto simp: C-def} \rangle$ )

```

```

    qed auto
  qed
  qed auto
  also have ... = C' * C p ^ n
    by (simp add: C'-def power-mult-distrib n-altdef flip: sum-distrib-right
mult.assoc)
  finally show ?thesis .
  qed

```

And with that, we have our inequality:

```

  finally show fact (p - 1) ^ n ≤ C' * C p ^ n .
  qed

```

Some simple asymptotic estimates show that this is clearly a contradiction, since the left-hand side grows much faster than the right-hand side and there are infinitely many sufficiently large primes:

```

  have freq: frequently prime sequentially
    using frequently-prime-cofinite unfolding cofinite-eq-sequentially .
  have ev: eventually (λp. (∀ q∈P. int p > |β q|) ∧
    real p > norm (∏ α∈Roots'. of-int (l ^ n) * (∏ α'∈Roots' - {α}. (α - α'))))
    sequentially
  by (intro eventually-ball-finite (finite P) ballI eventually-conj filterlim-real-sequentially
    eventually-compose-filterlim[OF eventually-gt-at-top] filterlim-int-sequentially)

  have frequently (λp. fact (p - 1) ^ n ≤ C' * C p ^ n) sequentially
  by (rule frequently-eventually-mono[OF freq ev]) (use ineq in blast)
  moreover have eventually (λp. fact (p - 1) ^ n > C' * C p ^ n) sequentially
  proof (cases R = 0)
  case True
  have eventually (λp. p * n > 1) at-top using ⟨n > 0⟩
  by (intro eventually-compose-filterlim[OF eventually-gt-at-top] mult-nat-right-at-top)
  thus ?thesis
  by eventually-elim (use ⟨n > 0⟩ True in ⟨auto simp: C-def power-0-left mult-ac⟩)
  next
  case False
  hence R > 0
  using ⟨R ≥ 0⟩ by auto
  define D :: real where D = (2 * R * |real-of-int l|) ^ n
  have D > 0
  using ⟨R > 0⟩ ⟨l > 0⟩ unfolding D-def by (intro zero-less-power) auto

  have (λp. C' * C p ^ n) ∈ O(λp. C p ^ n)
  by simp
  also have (λp. C p ^ n) ∈ O(λp. ((2 * R * l) ^ (n * p)) ^ n)
  proof (rule landau-o.big-power[OF bigthetaD1])
  have np: eventually (λp. p * n > 0) at-top using ⟨n > 0⟩
  by (intro eventually-compose-filterlim[OF eventually-gt-at-top] mult-nat-right-at-top)
  have eventually (λp. (2 * R) * C p = (2 * R * l) ^ (n * p)) at-top
  using np

```

proof *eventually-elim*
case (*elim p*)
have $2 * R * C p = l \wedge (n * p) * (2 * R) \wedge (Suc (n * p - 1))$
by (*simp add: C-def algebra-simps*)
also have $Suc (n * p - 1) = n * p$
using *elim by auto*
finally show *?case*
by (*simp add: algebra-simps*)
qed
hence $(\lambda p. (2 * R) * C p) \in \Theta(\lambda p. (2 * R * l) \wedge (n * p))$
by (*intro bigthetaI-cong*)
thus $C \in \Theta(\lambda p. (2 * R * l) \wedge (n * p))$
using $\langle R > 0 \rangle$ **by** *simp*
qed
also have $\dots = O(\lambda p. (D \wedge p) \wedge n)$
using $\langle l > 0 \rangle$ **by** (*simp flip: power-mult add: power2-eq-square mult-ac D-def*)
also have $(\lambda p. (D \wedge p) \wedge n) \in o(\lambda p. fact (p - 1) \wedge n)$
proof (*intro landau-o.small-power*)
have *eventually* $(\lambda p. D \wedge p = D * D \wedge (p - 1))$ *at-top*
using *eventually-gt-at-top[of 0]*
by *eventually-elim* (*use* $\langle D > 0 \rangle$ **in** *auto simp flip: power-Suc*)
hence $(\lambda p. D \wedge p) \in \Theta(\lambda p. D * D \wedge (p - 1))$
by (*intro bigthetaI-cong*)
hence $(\lambda p. D \wedge p) \in \Theta(\lambda p. D \wedge (p - 1))$
using $\langle D > 0 \rangle$ **by** *simp*
also have $(\lambda p. D \wedge (p - 1)) \in o(\lambda p. fact (p - 1))$
by (*intro smalloI-tendsto[OF filterlim-compose[OF power-over-fact-tendsto-0]]*
filterlim-minus-nat-at-top) *auto*
finally show $(\lambda p. D \wedge p) \in o(\lambda x. fact (x - 1))$.
qed *fact+*
finally have *smallo*: $(\lambda p. C' * C p \wedge n) \in o(\lambda p. fact (p - 1) \wedge n)$.
have *eventually* $(\lambda p. |C' * C p \wedge n| \leq 1/2 * fact (p - 1) \wedge n)$ *at-top*
using *landau-o.smallD[OF smallo, of 1/2]* **by** *simp*
thus *eventually* $(\lambda p. C' * C p \wedge n < fact (p - 1) \wedge n)$ *at-top*
proof *eventually-elim*
case (*elim p*)
have $C' * C p \wedge n \leq |C' * C p \wedge n|$
by *simp*
also have $\dots \leq 1/2 * fact (p - 1) \wedge n$
by *fact*
also have $\dots < fact (p - 1) \wedge n$
by *simp*
finally show *?case* .
qed
qed
ultimately have *frequently* $(\lambda p::nat. False)$ *sequentially*
by (*rule frequently-eventually-mono*) *auto*
thus *False*
by *simp*

qed

8.2 Removing the restriction of full sets of conjugates

We will now remove the restriction that the α_i must occur in full sets of conjugates by multiplying the equality with all permutations of roots.

lemma *Hermite-Lindemann-aux2*:

```

fixes  $X :: \text{complex set}$  and  $\beta :: \text{complex} \Rightarrow \text{int}$ 
assumes  $\text{finite } X$ 
assumes  $\text{nz: } \bigwedge x. x \in X \implies \beta x \neq 0$ 
assumes  $\text{alg: } \bigwedge x. x \in X \implies \text{algebraic } x$ 
assumes  $\text{sum0: } (\sum x \in X. \text{of-int } (\beta x) * \text{exp } x) = 0$ 
shows  $X = \{\}$ 
proof (rule ccontr)
assume  $X \neq \{\}$ 
note [intro] =  $\langle \text{finite } X \rangle$ 

```

Let P be the smallest integer polynomial whose roots are a superset of X :

```

define  $P :: \text{int poly}$  where  $P = \prod (\text{min-int-poly } ` X)$ 
define  $\text{Roots} :: \text{complex set}$  where  $\text{Roots} = \{x. \text{ipoly } P x = 0\}$ 
have [simp]:  $P \neq 0$ 
using  $\langle \text{finite } X \rangle$  by (auto simp: P-def)
have [intro]:  $\text{finite } \text{Roots}$ 
unfolding  $\text{Roots-def}$  by (intro poly-roots-finite) auto

```

```

have  $X \subseteq \text{Roots}$ 

```

```

proof safe

```

```

fix  $x$  assume  $x \in X$ 

```

```

hence  $\text{ipoly } (\text{min-int-poly } x) x = 0$ 

```

```

by (intro ipoly-min-int-poly alg)

```

```

thus  $x \in \text{Roots}$ 

```

```

using  $\langle \text{finite } X \rangle \langle x \in X \rangle$ 

```

```

by (auto simp: Roots-def P-def of-int-poly-hom.hom-prod poly-prod)

```

qed

```

have  $\text{squarefree } (\text{of-int-poly } P :: \text{complex poly})$ 

```

```

unfolding  $P\text{-def}$  of-int-poly-hom.hom-prod

```

```

proof (rule squarefree-prod-coprime; safe)

```

```

fix  $x$  assume  $x \in X$ 

```

```

thus  $\text{squarefree } (\text{of-int-poly } (\text{min-int-poly } x) :: \text{complex poly})$ 

```

```

by (intro squarefree-of-int-polyI) auto

```

```

next

```

```

fix  $x y$  assume  $xy: x \in X y \in X \text{min-int-poly } x \neq \text{min-int-poly } y$ 

```

```

thus  $\text{Rings.coprime } (\text{of-int-poly } (\text{min-int-poly } x)) (\text{of-int-poly } (\text{min-int-poly } y))$ 

```

```

::  $\text{complex poly}$ 

```

```

by (intro coprime-of-int-polyI[OF primes-coprime]) auto

```

qed

Since we will need a numbering of these roots, we obtain one:

```

define  $n$  where  $n = \text{card } \text{Roots}$ 
obtain  $\text{Root}$  where  $\text{Root}: \text{bij-betw } \text{Root } \{..<n\} \text{Roots}$ 
using  $\text{ex-bij-betw-nat-finite}[OF \langle \text{finite } \text{Roots} \rangle]$  unfolding  $n\text{-def atLeast0LessThan}$ 
by  $\text{metis}$ 
define  $\text{unRoot} :: \text{complex} \Rightarrow \text{nat}$  where  $\text{unRoot} = \text{inv-into } \{..<n\} \text{Root}$ 
have  $\text{unRoot}: \text{bij-betw } \text{unRoot } \text{Roots } \{..<n\}$ 
unfolding  $\text{unRoot-def}$  by ( $\text{intro } \text{bij-betw-inv-into } \text{Root}$ )
have  $\text{unRoot-Root } [simp]: \text{unRoot } (\text{Root } i) = i$  if  $i < n$  for  $i$ 
unfolding  $\text{unRoot-def}$  using  $\text{Root}$  that by ( $\text{subst inv-into-f-f}$ ) ( $\text{auto simp:}$ 
 $\text{bij-betw-def}$ )
have  $\text{Root-unRoot } [simp]: \text{Root } (\text{unRoot } x) = x$  if  $x \in \text{Roots}$  for  $x$ 
unfolding  $\text{unRoot-def}$  using  $\text{Root}$  that by ( $\text{subst f-inv-into-f}$ ) ( $\text{auto simp:}$ 
 $\text{bij-betw-def}$ )
have  $[simp, \text{intro}]: \text{Root } i \in \text{Roots}$  if  $i < n$  for  $i$ 
using  $\text{Root}$  that by ( $\text{auto simp: } \text{bij-betw-def}$ )
have  $[simp, \text{intro}]: \text{unRoot } x < n$  if  $x \in \text{Roots}$  for  $x$ 
using  $\text{unRoot}$  that by ( $\text{auto simp: } \text{bij-betw-def}$ )

```

We will also need to convert between permutations of natural numbers less than n and permutations of the roots:

```

define  $\text{convert-perm} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow (\text{complex} \Rightarrow \text{complex})$  where
 $\text{convert-perm} = (\lambda \sigma x. \text{if } x \in \text{Roots} \text{ then } \text{Root } (\sigma (\text{unRoot } x)) \text{ else } x)$ 
have  $\text{bij-convert}: \text{bij-betw } \text{convert-perm } \{\sigma. \sigma \text{ permutes } \{..<n\}\} \{\sigma. \sigma \text{ permutes } \text{Roots}\}$ 
using  $\text{bij-betw-permutations}[OF \text{Root}]$  unfolding  $\text{convert-perm-def unRoot-def}$ 
.
have  $\text{permutes-convert-perm } [intro]: \text{convert-perm } \sigma \text{ permutes } \text{Roots}$  if  $\sigma \text{ permutes } \{..<n\}$  for  $\sigma$ 
using  $\text{that } \text{bij-convert}$  unfolding  $\text{bij-betw-def}$  by  $\text{blast}$ 
have  $\text{convert-perm-compose}: \text{convert-perm } (\pi \circ \sigma) = \text{convert-perm } \pi \circ \text{convert-perm } \sigma$ 
if  $\pi \text{ permutes } \{..<n\}$   $\sigma \text{ permutes } \{..<n\}$  for  $\sigma \pi$ 
proof ( $\text{intro ext}$ )
fix  $x$  show  $\text{convert-perm } (\pi \circ \sigma) x = (\text{convert-perm } \pi \circ \text{convert-perm } \sigma) x$ 
proof ( $\text{cases } x \in \text{Roots}$ )
case  $\text{True}$ 
thus  $?thesis$ 
using  $\text{permutes-in-image}[OF \text{that}(2), \text{of } \text{unRoot } x]$ 
by ( $\text{auto simp: } \text{convert-perm-def } \text{bij-betw-def}$ )
qed ( $\text{auto simp: } \text{convert-perm-def}$ )
qed

```

We extend the coefficient vector to the new roots by setting their coefficients to 0:

```

define  $\beta'$  where  $\beta' = (\lambda x. \text{if } x \in X \text{ then } \beta x \text{ else } 0)$ 

```

We now define the set of all permutations of our roots:

```

define  $\text{perms}$  where  $\text{perms} = \{\pi. \pi \text{ permutes } \text{Roots}\}$ 

```

```

have [intro]: finite perms
  unfolding perms-def by (rule finite-permutations) auto
have [simp]: card perms = fact n
  unfolding perms-def n-def by (intro card-permutations) auto

```

The following is the set of all $n!$ -tuples of roots, disregarding permutation of components. In other words: all multisets of roots with size $n!$.

```

define Roots-ms :: complex multiset set where
  Roots-ms = {X. set-mset X  $\subseteq$  Roots  $\wedge$  size X = fact n}
have [intro]: finite Roots-ms
  unfolding Roots-ms-def by (rule finite-multisets-of-size) auto

```

Next, the following is the set of $n!$ -tuples whose entries are precisely the multiset X :

```

define tuples :: complex multiset  $\Rightarrow$  (complex  $\Rightarrow$  complex)  $\Rightarrow$  complex set where
  tuples = ( $\lambda X$ . {f  $\in$  perms  $\rightarrow_E$  Roots. image-mset f (mset-set perms) = X})
have fin-tuples [intro]: finite (tuples X) for X
  unfolding tuples-def by (rule finite-subset[of - perms  $\rightarrow_E$  Roots, OF - finite-PiE]) auto
define tuples' :: (complex multiset  $\times$  (complex  $\Rightarrow$  complex)  $\Rightarrow$  complex) set where
  tuples' = (SIGMA X:Roots-ms. tuples X)

```

The following shows that our *tuples* definition is stable under permutation of the roots.

```

have bij-convert': bij-betw ( $\lambda f$ . f  $\circ$  ( $\lambda g$ .  $\sigma \circ g$ )) (tuples X) (tuples X)
  if  $\sigma$ :  $\sigma$  permutes Roots for  $\sigma$  X
proof (rule bij-betwI)
  have *: ( $\lambda f$ . f  $\circ$  ( $\circ$ )  $\sigma$ )  $\in$  tuples X  $\rightarrow$  tuples X if  $\sigma$ :  $\sigma$  permutes Roots for  $\sigma$ 
  proof
    fix f assume f: f  $\in$  tuples X
    show f  $\circ$  ( $\circ$ )  $\sigma$   $\in$  tuples X
      unfolding tuples-def
    proof safe
      fix  $\sigma'$ 
      assume  $\sigma'$ :  $\sigma' \in$  perms
      show (f  $\circ$  ( $\circ$ )  $\sigma$ )  $\sigma' \in$  Roots
        using permutes-compose[OF -  $\sigma$ , of  $\sigma'$ ]  $\sigma$   $\sigma'$  f by (auto simp: perms-def tuples-def)
    next
      fix  $\sigma'$ 
      assume  $\sigma'$ :  $\sigma' \notin$  perms
      have  $\neg(\sigma \circ \sigma')$  permutes Roots
      proof
        assume  $(\sigma \circ \sigma')$  permutes Roots
        hence inv-into UNIV  $\sigma \circ (\sigma \circ \sigma')$  permutes Roots
          by (rule permutes-compose) (use permutes-inv[OF  $\sigma$ ] in simp-all)
        also have inv-into UNIV  $\sigma \circ (\sigma \circ \sigma') = \sigma'$ 

```

```

      by (auto simp: fun-eq-iff permutes-inverses[OF  $\sigma$ ])
      finally show False using  $\sigma'$  by (simp add: perms-def)
    qed
  thus (f  $\circ$  ( $\circ$ )  $\sigma$ )  $\sigma' = \text{undefined}$ 
    using f by (auto simp: perms-def tuples-def)
next
have image-mset (f  $\circ$  ( $\circ$ )  $\sigma$ ) (mset-set perms) =
  image-mset f (image-mset (( $\circ$ )  $\sigma$ ) (mset-set perms))
  by (rule multiset.map-comp [symmetric])
also have image-mset (( $\circ$ )  $\sigma$ ) (mset-set perms) = mset-set perms
  using bij-betw-permutes-compose-left[OF  $\sigma$ ]
  by (subst image-mset-mset-set) (auto simp: bij-betw-def perms-def)
also have image-mset f ... = X
  using f by (auto simp: tuples-def)
finally show image-mset (f  $\circ$  ( $\circ$ )  $\sigma$ ) (mset-set perms) = X .
qed
qed

show ( $\lambda f. f \circ (\circ) \sigma$ )  $\in \text{tuples } X \rightarrow \text{tuples } X$ 
  by (rule *) fact
show ( $\lambda f. f \circ (\circ) (\text{inv-into UNIV } \sigma)$ )  $\in \text{tuples } X \rightarrow \text{tuples } X$ 
  by (intro * permutes-inv) fact
show f  $\circ$  ( $\circ$ )  $\sigma \circ$  ( $\circ$ ) ( $\text{inv-into UNIV } \sigma$ ) = f if  $f \in \text{tuples } X$  for f
  by (auto simp: fun-eq-iff o-def permutes-inverses[OF  $\sigma$ ])
show f  $\circ$  ( $\circ$ ) ( $\text{inv-into UNIV } \sigma$ )  $\circ$  ( $\circ$ )  $\sigma = f$  if  $f \in \text{tuples } X$  for f
  by (auto simp: fun-eq-iff o-def permutes-inverses[OF  $\sigma$ ])
qed

```

Next, we define the multiset of of possible exponents that we can get for a given $n!$ -multiset of roots,

define $R :: \text{complex multiset} \Rightarrow \text{complex multiset}$ **where**
 $R = (\lambda X. \text{image-mset } (\lambda f. \sum_{\sigma \in \text{perms. } \sigma} (f \sigma)) (\text{mset-set } (\text{tuples } X)))$

We show that, for each such multiset, there is a content-free integer polynomial that has exactly these exponents as roots. This shows that they form a full set of conjugates (but note this polynomial is not necessarily squarefree). The proof is yet another application of the fundamental theorem of symmetric polynomials.

obtain $Q :: \text{complex multiset} \Rightarrow \text{int poly}$
where $Q: \bigwedge X. X \in \text{Roots-ms} \implies \text{poly-roots } (\text{of-int-poly } (Q X)) = R X$
 $\bigwedge X. X \in \text{Roots-ms} \implies \text{content } (Q X) = 1$
proof –
{
fix $X :: \text{complex multiset}$
assume $X: X \in \text{Roots-ms}$
define $Q :: \text{complex poly mpoly}$ **where**
 $Q = (\prod_{f \in \text{tuples } X. \text{Const } [:0, 1]} -$
 $(\sum_{\sigma \mid \sigma \text{ permutes } \{..<n\}. \text{Var } (\sigma (\text{unRoot } (f (\text{convert-perm } \sigma))))))$)


```

define  $Q1$  where  $Q1 = (\prod f \in \text{tuples } X. [:- (\sum \sigma \mid \sigma \text{ permutes } \text{Roots}. \sigma (f \sigma)), 1:])$ 
define  $\text{ratpolys} :: \text{complex poly set}$  where  $\text{ratpolys} = \{p. \forall i. \text{poly.coeff } p \ i \in \mathbb{Q}\}$ 

have  $\text{insertion } (\lambda x. [:\text{Root } x:]) \ Q \in \text{ratpolys}$ 
proof ( $\text{rule symmetric-poly-of-roots-in-subring}[\text{where } l = \lambda x. [:\text{Root } x:]]$ )
  show  $\text{ring-closed ratpolys}$ 
  unfolding  $\text{ratpolys-def}$  by  $\text{standard } (\text{auto intro: coeff-mult-semiring-closed})$ 
  then interpret  $\text{ratpolys: ring-closed ratpolys}$  .
  have  $pCons \ 0 \ 1 \in \text{ratpolys}$ 
    by ( $\text{auto simp: ratpolys-def coeff-pCons split: nat.splits}$ )
  thus  $\forall m. \text{MPoly-Type.coeff } Q \ m \in \text{ratpolys}$ 
    unfolding  $Q\text{-def}$ 
    by ( $\text{intro allI ratpolys.coeff-prod-closed}$ )
    ( $\text{auto intro!: ratpolys.minus-closed ratpolys.sum-closed ratpolys.uminus-closed}$ 
 $\text{simp: coeff-Var mpoly-coeff-Const when-def}$ )
  next
  show  $\text{ring-homomorphism } (\lambda x :: \text{complex}. [:\text{Root } x:]) \ ..$ 
  next
  have  $\sigma (\text{unRoot } (f (\text{convert-perm } \sigma))) < n$  if  $f \in \text{tuples } X \ \sigma \text{ permutes } \{..<n\}$  for  $f \ \sigma$ 
  proof –
    have  $\text{convert-perm } \sigma \in \text{perms}$ 
    using  $\text{bij-convert that(2)}$  by ( $\text{auto simp: bij-betw-def perms-def}$ )
    hence  $f (\text{convert-perm } \sigma) \in \text{Roots}$ 
    using  $\text{that}$  by ( $\text{auto simp: tuples-def}$ )
    thus  $?thesis$ 
    using  $\text{permutes-in-image[OF that(2)]}$  by  $\text{simp}$ 
  qed
  thus  $\text{vars } Q \subseteq \{..<n\}$ 
  unfolding  $Q\text{-def}$ 
  by ( $\text{intro order.trans[OF vars-prod] UN-least order.trans[OF vars-sum]}$ 
 $\text{order.trans[OF vars-diff] Un-least}$ ) ( $\text{auto simp: vars-Var}$ )
next
define  $lc :: \text{complex}$  where  $lc = \text{of-int } (\text{Polynomial.lead-coeff } P)$ 
show  $[:\text{inverse } lc:] \in \text{ratpolys}$ 
  by ( $\text{auto simp: ratpolys-def coeff-pCons lc-def split: nat.splits}$ )
show  $\forall i. [:\text{poly.coeff } (\text{of-int-poly } P) \ i:] \in \text{ratpolys}$ 
  by ( $\text{auto simp: ratpolys-def coeff-pCons split: nat.splits}$ )
have  $lc \neq 0$ 
  by ( $\text{auto simp: lc-def}$ )
thus  $[:\text{inverse } lc:] * [:\text{lc}] = 1$ 
  by  $\text{auto}$ 
have  $\text{rsquarefree } (\text{of-int-poly } P :: \text{complex poly})$ 
  using  $\text{squarefree } (\text{of-int-poly } P :: \text{complex poly})$  by ( $\text{intro square-free-imp-rsquarefree}$ )
hence  $\text{of-int-poly } P = \text{Polynomial.smult } lc \ (\prod x \in \text{Roots}. [:-x, 1:])$ 
  unfolding  $\text{lc-def Roots-def of-int-hom.hom-lead-coeff[symmetric]}$ 

```

```

    by (rule complex-poly-decompose-rsquarefree [symmetric])
  also have ( $\prod_{x \in \text{Roots}}. [-x, 1:] = (\prod_{i < n}. [-\text{Root } i, 1:])$ )
    by (rule prod.reindex-bij-betw[OF Root, symmetric])
  finally show of-int-poly  $P = \text{Polynomial.smult } lc (\prod_{i < n}. [-\text{Root } i, 1:])$  .
next
show symmetric-mpoly  $\{..<n\} Q$ 
  unfolding symmetric-mpoly-def
proof safe
  fix  $\pi$  assume  $\pi: \pi$  permutes  $\{..<n\}$ 
  have mpoly-map-vars  $\pi Q = (\prod_{f \in \text{tuples } X}. \text{Const } (p\text{Cons } 0 1) - (\sum \sigma$ 
|  $\sigma$  permutes  $\{..<n\}$ .
    Var (( $\pi \circ \sigma$ ) (unRoot (f (convert-perm  $\sigma$ ))))))
  by (simp add: Q-def permutes-bij[OF  $\pi$ ])
  also have ... = ( $\prod_{f \in \text{tuples } X}. \text{Const } (p\text{Cons } 0 1) - (\sum \sigma$  |  $\sigma$  permutes
 $\{..<n\}$ .
    Var (( $\pi \circ \sigma$ ) (unRoot ((f  $\circ$  ( $\lambda \sigma$ . convert-perm  $\pi \circ \sigma$ )) (convert-perm
 $\sigma$ ))))))
  using  $\pi$  by (intro prod.reindex-bij-betw [OF bij-convert', symmetric])
auto
  also have ... = Q
  unfolding Q-def
proof (rule prod.cong, goal-cases)
  case (2 f)
  have ( $\sum \sigma$  |  $\sigma$  permutes  $\{..<n\}$ . Var (( $\pi \circ \sigma$ ) (unRoot ((f  $\circ$  ( $\lambda \sigma$ .
convert-perm  $\pi \circ \sigma$ )) (convert-perm  $\sigma$ )))))) =
    ( $\sum \sigma$  |  $\sigma$  permutes  $\{..<n\}$ . Var (( $\pi \circ \sigma$ ) (unRoot (f (convert-perm
( $\pi \circ \sigma$ ))))))
  using  $\pi$  by (intro sum.cong refl, subst convert-perm-compose) simp-all
  also have ... = ( $\sum \sigma$  |  $\sigma$  permutes  $\{..<n\}$ . Var ( $\sigma$  (unRoot (f
(convert-perm  $\sigma$ ))))))
  using  $\pi$  by (rule setum-permutations-compose-left [symmetric])
  finally show ?case by simp
qed auto
  finally show mpoly-map-vars  $\pi Q = Q$  .
qed
qed auto
also have insertion ( $\lambda x. [-\text{Root } x:]$ )  $Q = Q1$ 
  unfolding Q-def Q1-def insertion-prod insertion-sum insertion-diff inser-
tion-Const insertion-Var
proof (intro prod.cong, goal-cases)
  case f: (2 f)
  have ( $\sum \sigma$  |  $\sigma$  permutes  $\{..<n\}$ .  $[-\text{Root } (\sigma$  (unRoot (f (convert-perm  $\sigma$ )))):])
=
    ( $\sum \sigma$  |  $\sigma$  permutes  $\{..<n\}$ .  $[-\text{convert-perm } \sigma$  (f (convert-perm  $\sigma$ )):])
proof (rule sum.cong, goal-cases)
  case (2  $\sigma$ )
  have convert-perm  $\sigma$  permutes Roots
  using bij-convert 2 by (auto simp: bij-betw-def)
  hence f (convert-perm  $\sigma$ )  $\in$  Roots

```

```

    using f by (auto simp: tuples-def perms-def)
    thus ?case by (simp add: convert-perm-def)
qed simp-all
also have ... = ( $\sum \sigma \mid \sigma \text{ permutes } \text{Roots. } [:\sigma (f \sigma):]$ )
  by (rule sum.reindex-bij-betw[OF bij-convert])
finally show ?case
  by (simp flip: pCons-one coeff-lift-hom.hom-sum)
qed simp-all
finally have Q1  $\in$  ratpolys
  by auto
then obtain Q2 :: rat poly where Q2: Q1 = map-poly of-rat Q2
  unfolding ratpolys-def using ratpolyE[of Q1] by blast

have Q1  $\neq$  0
  unfolding Q1-def using fin-tuples[of X] by auto
with Q2 have Q2  $\neq$  0
  by auto
obtain Q3 :: int poly and lc :: rat
  where Q3: Q2 = Polynomial.smult lc (of-int-poly Q3) and lc > 0 and
content Q3 = 1
  using rat-to-normalized-int-poly-exists[OF  $\langle$ Q2  $\neq$  0 $\rangle$ ] by metis

have poly-roots (of-int-poly Q3) = poly-roots (map-poly (of-rat  $\circ$  of-int) Q3)
  by simp
also have map-poly (of-rat  $\circ$  of-int) Q3 = map-poly of-rat (map-poly of-int
Q3)
  by (subst map-poly-map-poly) auto
also have poly-roots ... = poly-roots (Polynomial.smult (of-rat lc) ...)
  using  $\langle$ lc > 0 $\rangle$  by simp
also have Polynomial.smult (of-rat lc) (map-poly of-rat (map-poly of-int Q3))
=
  map-poly of-rat (Polynomial.smult lc (map-poly of-int Q3))
  by (simp add: of-rat-hom.map-poly-hom-smult)
also have ... = Q1
  by (simp only: Q3 [symmetric] Q2 [symmetric])
also have poly-roots Q1 = R X
  unfolding Q1-def
  by (subst poly-roots-prod, force, subst poly-roots-linear)
  (auto simp: R-def perms-def sum-mset-image-mset-singleton sum-unfold-sum-mset)
finally have  $\exists Q. \text{poly-roots (of-int-poly } Q) = R X \wedge \text{content } Q = 1$ 
  using  $\langle$ content Q3 = 1 $\rangle$  by metis
}
hence  $\exists Q. \forall X \in \text{Roots-ms. poly-roots (of-int-poly (Q X)) = R X} \wedge \text{content (Q
X)} = 1$ 
  by metis
thus ?thesis using that by metis
qed

```

We can now collect all the $e^{\sum \alpha_i}$ that happen to be equal and let the following

be their coefficients:

```

define  $\beta'' :: \text{int poly} \Rightarrow \text{int}$ 
  where  $\beta'' = (\lambda q. \sum X \in \text{Roots-ms. int (count (prime-factorization (Q X)) q) *}$ 
 $(\prod_{x \in \#X. \beta' x}))$ 
  have  $\text{supp-}\beta'' : \{q. \beta'' q \neq 0\} \subseteq (\bigcup X \in \text{Roots-ms. prime-factors (Q X)})$ 
  unfolding  $\beta''\text{-def}$  using sum.not-neutral-contains-not-neutral by fastforce

```

We have to prove that β'' is not zero everywhere. We do this by selecting the nonzero term with the maximal exponent (w.r.t. the lexicographic ordering on the complex numbers) in every factor of the product and show that there is no other summand corresponding to these, so that their non-zero coefficient cannot get cancelled.

```

have  $\{q. \beta'' q \neq 0\} \neq \{\}$ 
proof -
  define  $f$  where  $f = \text{restrict } (\lambda \sigma. \text{inv-into UNIV } \sigma \text{ (complex-lex.Max } (\sigma ' X)))$ 
  perms
  have  $f : f \in \text{perms} \rightarrow X$ 
  proof
    fix  $\sigma$  assume  $\sigma : \sigma \in \text{perms}$ 
    have  $\text{complex-lex.Max } (\sigma ' X) \in \sigma ' X$ 
    using  $\langle X \neq \{\} \rangle$  by (intro complex-lex.Max-in finite-imageI) auto
    thus  $f \sigma \in X$ 
    using  $\sigma$  by (auto simp: f-def permutes-inverses[of  $\sigma$  Roots] perms-def)
  qed
hence  $f' : f \in \text{perms} \rightarrow_E \text{Roots}$ 
  using  $\langle X \subseteq \text{Roots} \rangle$  by (auto simp: f-def PiE-def)

define  $Y$  where  $Y = \text{image-mset } f \text{ (mset-set perms)}$ 
have  $Y \in \text{Roots-ms}$  using  $f'$   $\langle \text{finite perms} \rangle$ 
  by (auto simp: Roots-ms-def Y-def)

have  $(\sum \sigma \in \text{perms. } \sigma (f \sigma)) \in \# R Y$ 
proof -
  from  $f'$  have  $f \in \text{tuples } Y$ 
  unfolding tuples-def Y-def by simp
  thus ?thesis
  unfolding R-def using fin-tuples[of Y] by auto
qed
also have  $R Y = \text{poly-roots (of-int-poly (Q Y))}$ 
  by (rule Q(1) [symmetric]) fact
also have  $\dots = (\sum p \in \#\text{prime-factorization (Q Y). poly-roots (of-int-poly p)})$ 
  by (rule poly-roots-of-int-conv-sum-prime-factors)
finally obtain  $q$  where  $q : q \in \text{prime-factors (Q Y)} \ (\sum \sigma \in \text{perms. } \sigma (f \sigma)) \in \#$ 
 $\text{poly-roots (of-int-poly } q)$ 
  by auto

have  $\beta'' q = (\sum X \in \{Y\}. \text{int (count (prime-factorization (Q X)) q) * prod-mset}$ 
 $(\text{image-mset } \beta' X))$ 

```

```

unfolding  $\beta''$ -def
proof (intro sum.mono-neutral-right ballI)
  fix  $Y'$  assume  $Y'$ :  $Y' \in \text{Roots-ms} - \{Y\}$ 
  show  $\text{int} (\text{count} (\text{prime-factorization} (Q Y')) q) * \prod_{\#} (\text{image-mset } \beta' Y')$ 
= 0
proof (cases set-mset  $Y' \subseteq X$ )
  case  $Y'$ -subset: True
  have  $q \notin \text{prime-factors} (Q Y')$ 
  proof
    assume  $q'$ :  $q \in \text{prime-factors} (Q Y')$ 
    have  $\text{poly-roots} (\text{of-int-poly } q :: \text{complex poly}) \subseteq_{\#}$ 
       $\text{poly-roots} (\text{of-int-poly} (Q Y'))$ 
    using  $q'$  by (intro dvd-imp-poly-roots-subset of-int-poly-hom.hom-dvd)
  auto
  with  $q(2)$  have  $(\sum \sigma \in \text{perms}. \sigma (f \sigma)) \in_{\#} \text{poly-roots} (\text{of-int-poly} (Q Y'))$ 
  by (meson mset-subset-eqD)
  also have  $\text{poly-roots} (\text{of-int-poly} (Q Y')) = R Y'$ 
  using  $Q(1)[\text{of } Y'] Y'$  by auto
  finally obtain  $g$  where  $g$ :  $g \in \text{tuples } Y' (\sum \sigma \in \text{perms}. \sigma (f \sigma)) =$ 
 $(\sum \sigma \in \text{perms}. \sigma (g \sigma))$ 
  unfolding  $R$ -def using  $\text{fin-tuples}[\text{of } Y']$  by auto

moreover have  $(\sum \sigma \in \text{perms}. \sigma (g \sigma)) <_{\mathbb{C}} (\sum \sigma \in \text{perms}. \sigma (f \sigma))$ 
proof (rule sum-strict-mono-ex1-complex-lex)
  show  $le: \forall \sigma \in \text{perms}. \sigma (g \sigma) \leq_{\mathbb{C}} \sigma (f \sigma)$ 
  proof
    fix  $\sigma$  assume  $\sigma$ :  $\sigma \in \text{perms}$ 
    hence  $\sigma'$ :  $\sigma$  permutes Roots
    by (auto simp: perms-def)
    have  $\text{image-mset } g (\text{mset-set perms}) = Y'$ 
    using  $g$  by (auto simp: tuples-def)
    also have  $\text{set-mset} \dots \subseteq X$ 
    by fact
    finally have  $g \text{ ' perms} \subseteq X$ 
    using (finite perms) by auto
    hence  $\sigma (g \sigma) \leq_{\mathbb{C}} \text{complex-lex.Max} (\sigma \text{ ' } X)$ 
    using (finite perms)  $\sigma$ 
    by (intro complex-lex.Max.coboundedI finite-imageI imageI)
    (auto simp: tuples-def)
    also have  $\dots = \sigma (f \sigma)$ 
    using  $\sigma$  by (simp add: f-def permutes-inverses[OF  $\sigma'$ ])
    finally show  $\sigma (g \sigma) \leq_{\mathbb{C}} \sigma (f \sigma)$  .
  qed

have  $\text{image-mset } g (\text{mset-set perms}) \neq \text{image-mset } f (\text{mset-set perms})$ 
using  $Y' g$  by (auto simp: tuples-def Y-def)
then obtain  $\sigma$  where  $\sigma$ :  $\sigma \in_{\#} \text{mset-set perms } g \sigma \neq f \sigma$ 
by (meson multiset.map-cong)
have  $\sigma$  permutes Roots

```

```

    using  $\sigma$   $\langle$ finite perms $\rangle$  by (auto simp: perms-def)
  have  $\sigma$  ( $g$   $\sigma$ )  $\neq$   $\sigma$  ( $f$   $\sigma$ )
    using permutes-inj[OF  $\langle$  $\sigma$  permutes Roots $\rangle$ ]  $\sigma$  by (auto simp: inj-def)
  moreover have  $\sigma$  ( $g$   $\sigma$ )  $\leq_{\mathbb{C}}$   $\sigma$  ( $f$   $\sigma$ )
    using le  $\sigma$   $\langle$ finite perms $\rangle$  by auto
  ultimately have  $\sigma$  ( $g$   $\sigma$ )  $<_{\mathbb{C}}$   $\sigma$  ( $f$   $\sigma$ )
    by simp
  thus  $\exists \sigma \in \text{perms}. \sigma$  ( $g$   $\sigma$ )  $<_{\mathbb{C}}$   $\sigma$  ( $f$   $\sigma$ )
    using  $\sigma$   $\langle$ finite perms $\rangle$  by auto
  qed (use  $\langle$ finite perms $\rangle$  in simp-all)
  ultimately show False by simp
qed
thus ?thesis by auto
qed (auto simp:  $\beta'$ -def)
qed (use  $\langle$  $Y \in \text{Roots-ms}$  $\rangle$  in auto)
  also have ... = int (count (prime-factorization (Q Y)) q) * prod-mset
(image-mset  $\beta'$  Y)
  by simp
  also have ...  $\neq$  0
    using q nz  $\langle$ finite X $\rangle$   $\langle$ X  $\neq$   $\{\}$  $\rangle$   $\langle$ finite perms $\rangle$  f by (auto simp:  $\beta'$ -def Y-def)
  finally show {q.  $\beta''$  q  $\neq$  0}  $\neq$   $\{\}$ 
    by auto
qed

```

We are now ready for the final push: we start with the original sum that we know to be zero, multiply it with the other permutations, and then multiply out the sum.

```

  have 0 = ( $\sum x \in X. \beta$  x * exp x)
    using sum0 ..
  also have ... = ( $\sum x \in \text{Roots}. \beta'$  x * exp x)
    by (intro sum.mono-neutral-cong-left  $\langle$ X  $\subseteq$  Roots $\rangle$ ) (auto simp:  $\beta'$ -def)
  also have ... dvd ( $\prod \sigma \in \text{perms}. \sum x \in \text{Roots}. \beta'$  x * exp ( $\sigma$  x))
    by (rule dvd-prodI[OF  $\langle$ finite perms $\rangle$ ])
      (use permutes-id[of Roots] in (simp-all add: id-def perms-def))
  also have ... = ( $\sum f \in \text{perms} \rightarrow_E \text{Roots}. \prod \sigma \in \text{perms}. \beta'$  (f  $\sigma$ ) * exp ( $\sigma$  (f  $\sigma$ )))
    by (rule prod-sum-PiE) auto
  also have ... = ( $\sum f \in \text{perms} \rightarrow_E \text{Roots}. (\prod \sigma \in \text{perms}. \beta'$  (f  $\sigma$ )) * exp ( $\sum \sigma \in \text{perms}. \sigma$  (f  $\sigma$ )))
    using  $\langle$ finite perms $\rangle$  by (simp add: prod.distrib exp-sum)
  also have ... = ( $\sum (X,f) \in \text{tuples}'. (\prod \sigma \in \text{perms}. \beta'$  (f  $\sigma$ )) * exp ( $\sum \sigma \in \text{perms}. \sigma$  (f  $\sigma$ )))
    using  $\langle$ finite perms $\rangle$ 
    by (intro sum.reindex-bij-witness[of - snd  $\lambda f. (\text{image-mset } f (\text{mset-set perms}), f)$ ])
      (auto simp: tuples'-def tuples-def Roots-ms-def PiE-def Pi-def)
  also have ... = ( $\sum (X,f) \in \text{tuples}'. (\prod x \in \#X. \beta'$  x) * exp ( $\sum \sigma \in \text{perms}. \sigma$  (f  $\sigma$ )))
  proof (safe intro!: sum.cong)
    fix X :: complex multiset and f :: (complex  $\Rightarrow$  complex)  $\Rightarrow$  complex
    assume (X, f)  $\in$  tuples'
  
```

hence $X: X \in \text{Roots-ms } X = \text{image-mset } f \text{ (mset-set perms) and } f: f \in \text{perms}$
 $\rightarrow_E \text{Roots}$
by (*auto simp: tuples'-def tuples-def*)
have $(\prod \sigma \in \text{perms. } \beta' (f \sigma)) = (\prod \sigma \in \# \text{mset-set perms. } \beta' (f \sigma))$
by (*meson prod-unfold-prod-mset*)
also have $\dots = (\prod x \in \# X. \beta' x)$
unfolding $X(2)$ **by** (*simp add: multiset.map-comp o-def*)
finally show $(\prod \sigma \in \text{perms. } \beta' (f \sigma)) * \exp (\sum \sigma \in \text{perms. } \sigma (f \sigma)) =$
 $(\prod x \in \# X. \beta' x) * \exp (\sum \sigma \in \text{perms. } \sigma (f \sigma))$ **by** *simp*
qed
also have $\dots = (\sum X \in \text{Roots-ms. } \sum f \in \text{tuples } X. (\prod x \in \# X. \beta' x) * \exp (\sum \sigma \in \text{perms.}$
 $\sigma (f \sigma)))$
unfolding *tuples'-def* **by** (*intro sum.Sigma [symmetric] auto*)
also have $\dots = (\sum X \in \text{Roots-ms. of-int } (\prod x \in \# X. \beta' x) * (\sum f \in \text{tuples } X. \exp$
 $(\sum \sigma \in \text{perms. } \sigma (f \sigma))))$
by (*simp add: sum-distrib-left*)
also have $\dots = (\sum X \in \text{Roots-ms. of-int } (\prod x \in \# X. \beta' x) * (\sum x \in \# R X. \exp x))$
by (*simp only: R-def multiset.map-comp o-def sum-unfold-sum-mset*)
also have $\dots = (\sum X \in \text{Roots-ms. of-int } (\prod x \in \# X. \beta' x) * (\sum x \in \# \text{poly-roots}$
 $(\text{of-int-poly } (Q X)). \exp x))$
by (*intro sum.cong (simp-all flip: Q)*)

Our problem now is that the polynomials $Q X$ can still contain multiple roots and that their roots might not be disjoint. We therefore split them all into irreducible factors and collect equal terms.

also have $\dots = (\sum X \in \text{Roots-ms. } (\sum p. \text{of-int } (\text{int } (\text{count } (\text{prime-factorization}$
 $(Q X)) p) * (\prod x \in \# X. \beta' x)) * (\sum x \mid \text{ipoly } p x = 0. \exp x))$
proof (*rule sum.cong, goal-cases*)
case $(2 X)$
have $(\sum x \in \# \text{poly-roots } (\text{of-int-poly } (Q X) :: \text{complex poly}). \exp x) =$
 $(\sum x \in \# (\sum p \in \# \text{prime-factorization } (Q X). \text{poly-roots } (\text{of-int-poly } p)).$
 $\exp x)$
by (*subst poly-roots-of-int-conv-sum-prime-factors (rule refl)*)
also have $\dots = (\sum p \in \# \text{prime-factorization } (Q X). \sum x \in \# \text{poly-roots } (\text{of-int-poly}$
 $p). \exp x)$
by (*rule sum-mset-image-mset-sum-mset-image-mset*)
also have *rsquarefree* (*of-int-poly p :: complex poly*) **if** $p \in \text{prime-factors } (Q X)$
for p
proof (*rule irreducible-imp-rsquarefree-of-int-poly*)
have *prime p*
using *that by auto*
thus *irreducible p*
by *blast*
next
show *Polynomial.degree p > 0*
by (*intro content-1-imp-nonconstant-prime-factors[OF Q(2) that] 2*)
qed
hence $(\sum p \in \# \text{prime-factorization } (Q X). \sum x \in \# \text{poly-roots } (\text{of-int-poly } p). \exp$

```

x) =
  (∑ p ∈ #prime-factorization (Q X). ∑ x | ipoly p x = 0. exp (x :: complex))
  unfolding sum-unfold-sum-mset
  by (intro arg-cong[of - - sum-mset] image-mset-cong sum.cong refl,
      subst rsquarefree-poly-roots-eq) auto
  also have ... = (∑ p. count (prime-factorization (Q X)) p * (∑ x | ipoly p x
= 0. exp (x :: complex)))
  by (rule sum-mset-conv-Sum-any)
  also have of-int (∏ x ∈ #X. β' x) * ... =
    (∑ p. of-int (int (count (prime-factorization (Q X)) p) * (∏ x ∈ #X.
β' x)) * (∑ x | ipoly p x = 0. exp x))
  by (subst Sum-any-right-distrib) (auto simp: mult-ac)
  finally show ?case by simp
qed auto
also have ... = (∑ q. of-int (β'' q) * (∑ x | ipoly q x = 0. exp x))
  unfolding β''-def of-int-sum
  by (subst Sum-any-sum-swap [symmetric]) (auto simp: sum-distrib-right)
  also have ... = (∑ q | β'' q ≠ 0. of-int (β'' q) * (∑ x | ipoly q x = 0. exp x))
  by (intro Sum-any.expand-superset finite-subset[OF supp-β'']) auto
  finally have (∑ q | β'' q ≠ 0. of-int (β'' q) * (∑ x | ipoly q x = 0. exp (x ::
complex))) = 0
  by simp

```

We are now in the situation of our the specialised Hermite–Lindemann Theorem we proved earlier and can easily derive a contradiction.

```

moreover have (∑ q | β'' q ≠ 0. of-int (β'' q) * (∑ x | ipoly q x = 0. exp (x ::
complex))) ≠ 0
proof (rule Hermite-Lindemann-aux1)
  show finite {q. β'' q ≠ 0}
  by (rule finite-subset[OF supp-β'']) auto
next
  show pairwise Rings.coprime {q. β'' q ≠ 0}
proof (rule pairwiseI, clarify)
  fix p q assume pq: p ≠ q β'' p ≠ 0 β'' q ≠ 0
  hence prime p prime q
  using supp-β'' Q(2) by auto
  with pq show Rings.coprime p q
  by (simp add: primes-coprime)
qed
next
  fix q :: int poly
  assume q: q ∈ {q. β'' q ≠ 0}
  also note supp-β''
  finally obtain X where X: X ∈ Roots-ms q ∈ prime-factors (Q X)
  by blast
  show irreducible q
  using X by (intro prime-elem-imp-irreducible prime-imp-prime-elem) auto
  show Polynomial.degree q > 0 using X
  by (intro content-1-imp-nonconstant-prime-factors[OF Q(2)][of X])

```


qed (use $\langle \{x. \beta'' x \neq 0\} \neq \{\} \rangle$ in auto)

ultimately show *False* by contradiction
qed

8.3 Removing the restriction to integer coefficients

Next, we weaken the restriction that the β_i must be integers to the restriction that they must be rationals. This is done simply by multiplying with the least common multiple of the demoninators.

lemma *Hermite-Lindemann-aux3*:

fixes $X :: \text{complex set}$ **and** $\beta :: \text{complex} \Rightarrow \text{rat}$

assumes *finite X*

assumes *nz*: $\bigwedge x. x \in X \implies \beta x \neq 0$

assumes *alg*: $\bigwedge x. x \in X \implies \text{algebraic } x$

assumes *sum0*: $(\sum x \in X. \text{of-rat } (\beta x) * \text{exp } x) = 0$

shows $X = \{\}$

proof –

define $l :: \text{int}$ **where** $l = \text{Lcm } ((\text{snd} \circ \text{quotient-of} \circ \beta) ` X)$

have [*simp*]: $\text{snd } (\text{quotient-of } r) \neq 0$ **for** r

using *quotient-of-denom-pos'[of r]* **by** *simp*

have [*simp*]: $l \neq 0$

using $\langle \text{finite } X \rangle$ **by** (auto *simp*: *l-def Lcm-0-iff*)

have *of-int l * β x ∈ ℤ if x ∈ X for x*

proof –

define a b **where** $a = \text{fst } (\text{quotient-of } (\beta x))$ **and** $b = \text{snd } (\text{quotient-of } (\beta x))$

have $b > 0$

using *quotient-of-denom-pos'[of β x]* **by** (auto *simp*: *b-def*)

have $\beta x = \text{of-int } a / \text{of-int } b$

by (*intro quotient-of-div*) (auto *simp*: *a-def b-def*)

also have $\text{of-int } l * \dots = \text{of-int } (l * a) / \text{of-int } b$

using $\langle b > 0 \rangle$ **by** (*simp add*: *field-simps*)

also have $\dots \in \mathbb{Z}$ **using** *that*

by (*intro of-int-divide-in-Ints*) (auto *simp*: *l-def b-def*)

finally show *?thesis* .

qed

hence $\forall x \in X. \exists n. \text{of-int } n = \text{of-int } l * \beta x$

using *Ints-cases* **by** *metis*

then obtain β' **where** $\beta': \text{of-int } (\beta' x) = \text{of-int } l * \beta x$ **if** $x \in X$ **for** x

by *metis*

show *?thesis*

proof (*rule Hermite-Lindemann-aux2*)

have $0 = \text{of-int } l * (\sum x \in X. \text{of-rat } (\beta x) * \text{exp } x :: \text{complex})$

by (*simp add*: *sum0*)

also have $\dots = (\sum x \in X. \text{of-int } (\beta' x) * \text{exp } x)$

unfolding *sum-distrib-left*

proof (*rule sum.cong, goal-cases*)

```

case (2 x)
have of-int l * of-rat (β x) = of-rat (of-int l * β x)
  by (simp add: of-rat-mult)
also have of-int l * β x = of-int (β' x)
  using 2 by (rule β' [symmetric])
finally show ?case by (simp add: mult-ac)
qed simp-all
finally show ... = 0 ..
next
fix x assume x ∈ X
hence of-int (β' x) ≠ (0 :: rat) using nz
  by (subst β') auto
thus β' x ≠ 0
  by auto
qed (use alg ⟨finite X⟩ in auto)
qed

```

Next, we weaken the restriction that the β_i must be rational to them being algebraic. Similarly to before, this is done by multiplying over all possible permutations of the β_i (in some sense) to introduce more symmetry, from which it then follows by the fundamental theorem of symmetric polynomials that the resulting coefficients are rational.

lemma *Hermite-Lindemann-aux4*:

```

fixes β :: complex ⇒ complex
assumes [intro]: finite X
assumes alg1:  $\bigwedge x. x \in X \implies \text{algebraic } x$ 
assumes alg2:  $\bigwedge x. x \in X \implies \text{algebraic } (\beta x)$ 
assumes nz:  $\bigwedge x. x \in X \implies \beta x \neq 0$ 
assumes sum0:  $(\sum x \in X. \beta x * \text{exp } x) = 0$ 
shows X = {}
proof (rule ccontr)
assume X: X ≠ {}
note [intro!] = finite-PiE

```

We now take more or less the same approach as before, except that now we find a polynomial that has all of the conjugates of the coefficients β as roots. Note that this is a slight deviation from Baker's proof, who picks one polynomial for each β independently. I did it this way because, as Bernard [2] observed, it makes the proof a bit easier.

```

define P :: int poly where P =  $\prod ((\text{min-int-poly} \circ \beta) \text{ ` } X)$ 
define Roots :: complex set where Roots = {x. ipoly P x = 0}
have 0 ∉ Roots using ⟨finite X⟩ alg2 nz
  by (auto simp: Roots-def P-def poly-prod)
have [simp]: P ≠ 0
  using ⟨finite X⟩ by (auto simp: P-def)
have [intro]: finite Roots
  unfolding Roots-def by (intro poly-roots-finite) auto

```

```

have  $\beta \text{ ' } X \subseteq \text{Roots}$ 
proof safe
  fix  $x$  assume  $x \in X$ 
  hence  $\text{ipoly } (\text{min-int-poly } (\beta x)) (\beta x) = 0$ 
  by ( $\text{intro ipoly-min-int-poly alg2}$ )
  thus  $\beta x \in \text{Roots}$ 
  using  $\langle \text{finite } X \rangle \langle x \in X \rangle$ 
  by ( $\text{auto simp: Roots-def P-def of-int-poly-hom.hom-prod poly-prod}$ )
qed

```

```

have  $\text{squarefree } (\text{of-int-poly } P :: \text{complex poly})$ 
  unfolding  $P\text{-def of-int-poly-hom.hom-prod o-def}$ 
proof ( $\text{rule squarefree-prod-coprime; safe}$ )
  fix  $x$  assume  $x \in X$ 
  thus  $\text{squarefree } (\text{of-int-poly } (\text{min-int-poly } (\beta x)) :: \text{complex poly})$ 
  by ( $\text{intro squarefree-of-int-polyI}$ )  $\text{auto}$ 
next
  fix  $x y$  assume  $xy: x \in X y \in X \text{min-int-poly } (\beta x) \neq \text{min-int-poly } (\beta y)$ 
  thus  $\text{Rings.coprime } (\text{of-int-poly } (\text{min-int-poly } (\beta x)))$ 
   $(\text{of-int-poly } (\text{min-int-poly } (\beta y)) :: \text{complex poly})$ 
  by ( $\text{intro coprime-of-int-polyI[OF primes-coprime]}$ )  $\text{auto}$ 
qed

```

```

define  $n$  where  $n = \text{card Roots}$ 
define  $m$  where  $m = \text{card } X$ 
have  $\text{Roots} \neq \{\}$ 
  using  $\langle \beta \text{ ' } X \subseteq \text{Roots} \rangle \langle X \neq \{\} \rangle$  by  $\text{auto}$ 
hence  $n > 0 m > 0$ 
  using  $\langle \text{finite Roots} \rangle \langle \text{finite } X \rangle \langle X \neq \{\} \rangle$  by ( $\text{auto simp: n-def m-def}$ )
have  $\text{fin1 [simp]: finite } (X \rightarrow_E \text{Roots})$ 
  by  $\text{auto}$ 
have  $[\text{simp}]: \text{card } (X \rightarrow_E \text{Roots}) = n \wedge m$ 
  by ( $\text{subst card-PiE}$ ) ( $\text{auto simp: m-def n-def}$ )

```

We again find a bijection between the roots and the natural numbers less than n :

```

obtain  $\text{Root where Root: bij-betw Root } \{..<n\} \text{Roots}$ 
  using  $\text{ex-bij-betw-nat-finite[OF } \langle \text{finite Roots} \rangle]$  unfolding  $n\text{-def atLeast0LessThan}$ 
by  $\text{metis}$ 
define  $\text{unRoot :: complex} \Rightarrow \text{nat where unRoot} = \text{inv-into } \{..<n\} \text{Root}$ 
have  $\text{unRoot: bij-betw unRoot Roots } \{..<n\}$ 
  unfolding  $\text{unRoot-def by } (\text{intro bij-betw-inv-into Root})$ 
have  $\text{unRoot-Root [simp]: unRoot } (\text{Root } i) = i$  if  $i < n$  for  $i$ 
  unfolding  $\text{unRoot-def using Root that by } (\text{subst inv-into-f-f})$  ( $\text{auto simp:}$ 
 $\text{bij-betw-def}$ )
have  $\text{Root-unRoot [simp]: Root } (\text{unRoot } x) = x$  if  $x \in \text{Roots}$  for  $x$ 
  unfolding  $\text{unRoot-def using Root that by } (\text{subst f-inv-into-f})$  ( $\text{auto simp:}$ 
 $\text{bij-betw-def}$ )
have  $[\text{simp, intro}]: \text{Root } i \in \text{Roots}$  if  $i < n$  for  $i$ 

```

```

using Root that by (auto simp: bij-betw-def)
have [simp, intro]: unRoot  $x < n$  if  $x \in \text{Roots}$  for  $x$ 
using unRoot that by (auto simp: bij-betw-def)

```

And we again define the set of multisets and tuples that we will get in the expanded product.

```

define Roots-ms :: complex multiset set where
  Roots-ms = { $Y$ . set-mset  $Y \subseteq X \wedge \text{size } Y = n \wedge m$ }
have [intro]: finite Roots-ms
unfolding Roots-ms-def by (rule finite-multisets-of-size) auto
define tuples :: complex multiset  $\Rightarrow ((\text{complex} \Rightarrow \text{complex}) \Rightarrow \text{complex})$  set
where tuples = ( $\lambda Y$ . { $f \in (X \rightarrow_E \text{Roots}) \rightarrow_E X$ . image-mset  $f$  (mset-set ( $X \rightarrow_E \text{Roots}$ )) =  $Y$ )})
have [intro]: finite (tuples Y) for  $Y$ 
unfolding tuples-def by (rule finite-subset[of - (X →E Roots) →E X]) auto

```

We will also need to convert permutations over the natural and over the roots again.

```

define convert-perm :: ( $\text{nat} \Rightarrow \text{nat}$ )  $\Rightarrow$  ( $\text{complex} \Rightarrow \text{complex}$ ) where
  convert-perm = ( $\lambda \sigma$   $x$ . if  $x \in \text{Roots}$  then Root ( $\sigma$  (unRoot  $x$ )) else  $x$ )
have bij-convert: bij-betw convert-perm { $\sigma$ .  $\sigma$  permutes {.. $n$ }} { $\sigma$ .  $\sigma$  permutes Roots}
using bij-betw-permutations[OF Root] unfolding convert-perm-def unRoot-def
.
have permutes-convert-perm [intro]: convert-perm  $\sigma$  permutes Roots if  $\sigma$  permutes {.. $n$ } for  $\sigma$ 
using that bij-convert unfolding bij-betw-def by blast

```

We also need a small lemma showing that our tuples are stable under permutation of the roots.

```

have bij-betw-compose-perm:
  bij-betw ( $\lambda f$ . restrict ( $\lambda g$ .  $f$  (restrict ( $\pi \circ g$ )  $X$ )) ( $X \rightarrow_E \text{Roots}$ )) (tuples  $Y$ )
(tuples  $Y$ )
if  $\pi$ :  $\pi$  permutes Roots and  $Y \in \text{Roots-ms}$  for  $\pi$   $Y$ 
proof (rule bij-betwI)
have *: ( $\lambda f$ . restrict ( $\lambda g$ .  $f$  (restrict ( $\pi \circ g$ )  $X$ )) ( $X \rightarrow_E \text{Roots}$ ))  $\in$  tuples  $Y \rightarrow$ 
tuples  $Y$ 
if  $\pi$ :  $\pi$  permutes Roots for  $\pi$ 
proof
fix  $f$  assume  $f$ :  $f \in$  tuples  $Y$ 
hence  $f'$ :  $f \in (X \rightarrow_E \text{Roots}) \rightarrow_E X$ 
by (auto simp: tuples-def)
define  $f'$  where  $f' = (\lambda g$ .  $f$  (restrict ( $\pi \circ g$ )  $X$ ))
have  $f' \in (X \rightarrow_E \text{Roots}) \rightarrow X$  unfolding f'-def
using  $f'$  bij-betw-apply[OF bij-betw-compose-left-perm-PiE[OF  $\pi$ , of  $X$ ]] by
blast
hence restrict  $f'$  ( $X \rightarrow_E \text{Roots}$ )  $\in (X \rightarrow_E \text{Roots}) \rightarrow_E X$ 
by simp

```

```

moreover have image-mset (restrict f' (X →E Roots)) (mset-set (X →E
Roots)) = Y
proof –
  have image-mset (restrict f' (X →E Roots)) (mset-set (X →E Roots)) =
    image-mset f' (mset-set (X →E Roots))
  by (intro image-mset-cong) auto
  also have ... = image-mset f (image-mset (λg. restrict (π ∘ g) X) (mset-set
(X →E Roots)))
  unfolding f'-def o-def multiset.map-comp by (simp add: o-def)
  also have image-mset (λg. restrict (π ∘ g) X) (mset-set (X →E Roots)) =
    mset-set (X →E Roots)
  by (intro bij-betw-image-mset-set bij-betw-compose-left-perm-PiE π)
  also have image-mset f ... = Y
  using f by (simp add: tuples-def)
  finally show ?thesis .
qed
ultimately show restrict f' (X →E Roots) ∈ tuples Y
  by (auto simp: tuples-def)
qed
show (λf. restrict (λg. f (restrict (π ∘ g) X)) (X →E Roots)) ∈ tuples Y →
tuples Y
  by (intro * π)
  show (λf. restrict (λg. f (restrict (inv-into UNIV π ∘ g) X)) (X →E Roots))
∈ tuples Y → tuples Y
  by (intro * permutes-inv π)
next
have *: (λg∈X →E Roots. (λg∈X →E Roots. f (restrict (π ∘ g) X))
(restrict (inv-into UNIV π ∘ g) X)) = f (is ?lhs = -)
if f: f ∈ tuples Y and π: π permutes Roots for f π
proof
  fix g show ?lhs g = f g
  proof (cases g ∈ X →E Roots)
    case True
      have restrict (π ∘ restrict (inv-into UNIV π ∘ g) X) X = g
      using True
      by (intro ext) (auto simp: permutes-inverses[OF π])
      thus ?thesis using True
      by (auto simp: permutes-in-image[OF permutes-inv[OF π]])
    qed (use f in (auto simp: tuples-def))
  qed
show (λg∈X →E Roots. (λg∈X →E Roots. f (restrict (π ∘ g) X))
(restrict (inv-into UNIV π ∘ g) X)) = f if f ∈ tuples Y for f
  using *[OF that π] .
show (λg∈X →E Roots. (λg∈X →E Roots. f (restrict (inv-into UNIV π ∘ g)
X))
(restrict (π ∘ g) X)) = f if f ∈ tuples Y for f
  using *[OF that permutes-inv[OF π]] permutes-inv-inv[OF π] by simp
qed

```

We show that the coefficients in the expanded new sum are rational – again

using the fundamental theorem of symmetric polynomials.

```

define  $\beta'$  :: complex multiset  $\Rightarrow$  complex
  where  $\beta' = (\lambda Y. \sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots. } g (f g))$ 

have  $\beta' Y \in \mathbb{Q}$  if  $Y: Y \in \text{Roots-ms}$  for  $Y$ 
proof –
  define  $Q$  :: complex mpoly
    where  $Q = (\sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\text{unRoot } (g (f g))))$ 

  have insertion Root  $Q \in \mathbb{Q}$ 
proof (rule symmetric-poly-of-roots-in-subring)
  show ring-closed ( $\mathbb{Q}$  :: complex set)
    by standard auto
  then interpret ring-closed  $\mathbb{Q}$  :: complex set .
  show  $\forall m. \text{coeff } Q m \in \mathbb{Q}$ 
    by (auto simp: Q-def coeff-Var when-def intro!: sum-in-Rats coeff-prod-closed)
next
  show symmetric-mpoly  $\{..<n\}$   $Q$ 
    unfolding symmetric-mpoly-def
  proof safe
    fix  $\pi$  assume  $\pi: \pi$  permutes  $\{..<n\}$ 
    define  $\pi'$  where  $\pi' = \text{convert-perm } (\text{inv-into UNIV } \pi)$ 
    have  $\pi': \pi'$  permutes Roots
      unfolding  $\pi'$ -def by (intro permutes-convert-perm permutes-inv  $\pi$ )
    have mpoly-map-vars  $\pi Q = (\sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\pi$ 
(unRoot (g (f g))))))
      unfolding Q-def by (simp add: permutes-bij[OF  $\pi$ ])
      also have  $\dots = (\sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\text{unRoot } (g (f$ 
(restrict ( $\pi' \circ g$ ) X))))))
    proof (rule sum.cong, goal-cases)
      case (2 f)
      have  $f: f \in (X \rightarrow_E \text{Roots}) \rightarrow_E X$ 
        using 2 by (auto simp: tuples-def)
      have  $(\prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\pi (\text{unRoot } (g (f g)))))) =$ 
 $(\prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\pi (\text{unRoot } (\text{restrict } (\pi' \circ g) X (f (\text{restrict}$ 
( $\pi' \circ g$ ) X))))))
      using  $\pi'$  by (intro prod.reindex-bij-betw [symmetric] bij-betw-compose-left-perm-PiE)
      also have  $\dots = (\prod g \in X \rightarrow_E \text{Roots. } \text{Var } (\text{unRoot } (g (f (\text{restrict } (\pi' \circ g)$ 
X))))))
    proof (intro prod.cong refl arg-cong[of - - Var])
      fix  $g$  assume  $g: g \in X \rightarrow_E \text{Roots}$ 
      have restrict ( $\pi' \circ g$ )  $X \in X \rightarrow_E \text{Roots}$ 
        using bij-betw-compose-left-perm-PiE[OF  $\pi'$ , of X]  $g$  unfolding
bij-betw-def by blast
      hence  $*$ :  $f (\text{restrict } (\pi' \circ g) X) \in X$ 
        by (rule PiE-mem[OF f])
      hence  $**$ :  $g (f (\text{restrict } (\pi' \circ g) X)) \in \text{Roots}$ 
        by (rule PiE-mem[OF g])

```

```

      have unRoot (restrict (π' ∘ g) X (f (restrict (π' ∘ g) X))) =
        unRoot (Root (inv-into UNIV π (unRoot (g (f (restrict (π' ∘ g)
X))))))
      using ** by (subst π'-def) (auto simp: convert-perm-def)
      also have inv-into UNIV π (unRoot (g (f (restrict (π' ∘ g) X)))) ∈
{..<n}
      using ** by (subst permutes-in-image[OF permutes-inv[OF π]]) auto
      hence unRoot (Root (inv-into UNIV π (unRoot (g (f (restrict (π' ∘ g)
X)))))) =
        inv-into UNIV π (unRoot (g (f (restrict (π' ∘ g) X))))
      by (intro unRoot-Root) auto
      also have π ... = unRoot (g (f (restrict (π' ∘ g) X)))
      by (rule permutes-inverses[OF π])
      finally show π (unRoot (restrict (π' ∘ g) X (f (restrict (π' ∘ g) X)))) =
        unRoot (g (f (restrict (π' ∘ g) X))) .

    qed
    finally show ?case .
  qed simp-all
  also have ... = (∑ x∈tuples Y. ∏ g∈X →E Roots. Var (unRoot (g ((λg∈X
→E Roots. x (restrict (π' ∘ g) X)) g))))
  by (intro sum.cong prod.cong refl) auto
  also have ... = Q
  unfolding Q-def
  by (rule sum.reindex-bij-betw[OF bij-betw-compose-perm]) (use π' Y in
simp-all)
  finally show mpoly-map-vars π Q = Q .
  qed
next
show vars Q ⊆ {..<n}
  unfolding Q-def
  by (intro order.trans[OF vars-sum] UN-least order.trans[OF vars-prod])
(auto simp: vars-Var tuples-def)
next
define lc where lc = Polynomial.lead-coeff P
have lc ≠ 0
  unfolding lc-def by auto
thus inverse (of-int lc) * (of-int lc :: complex) = 1 and inverse (of-int lc) ∈
ℚ
  by auto
have rsquarefree (of-int-poly P :: complex poly)
using (squarefree (of-int-poly P :: complex poly)) by (intro squarefree-imp-rsquarefree)
hence of-int-poly P = Polynomial.smult (of-int lc) (∏ x∈Roots. [:-x, 1:])
  unfolding lc-def of-int-hom.hom-lead-coeff[symmetric] Roots-def
  by (rule complex-poly-decompose-rsquarefree [symmetric])
also have (∏ x∈Roots. [:-x, 1:]) = (∏ i<n. [:-Root i, 1:])
  by (rule prod.reindex-bij-betw[OF Root, symmetric])
finally show of-int-poly P = Polynomial.smult (of-int lc) (∏ i<n. [:- Root
i, 1:]) .
  qed auto

```

also have *insertion* $Root\ Q = (\sum f \in tuples\ Y. \prod g \in X \rightarrow_E Roots. Root\ (unRoot\ (g\ (f\ g))))$
by (*simp add: Q-def insertion-sum insertion-prod*)
also have $\dots = \beta' Y$
unfolding β' -def **by** (*intro sum.cong prod.cong refl Root-unRoot*) (*auto simp: tuples-def*)
finally show ?thesis .
qed
hence $\forall Y \in Roots\text{-}ms. \exists x. \beta' Y = of\text{-}rat\ x$
by (*auto elim!: Rats-cases*)
then obtain $\beta'' :: complex\ multiset \Rightarrow rat$
where $\beta'': \bigwedge Y. Y \in Roots\text{-}ms \implies \beta' Y = of\text{-}rat\ (\beta'' Y)$
by *metis*

We again collect all the terms that happen to have equal exponents and call their coefficients β'' :

define $\beta''' :: complex \Rightarrow rat$ **where** $\beta''' = (\lambda \alpha. \sum Y \in Roots\text{-}ms. (\beta'' Y\ when\ \sum \# Y = \alpha))$
have $supp\text{-}\beta''': \{x. \beta''' x \neq 0\} \subseteq sum\text{-}mset\ ' Roots\text{-}ms$
by (*auto simp: β''' -def when-def elim!: sum.not-neutral-contains-not-neutral split: if-splits*)

We again start with the sum that we now to be zero and multiply it with all the sums that can be obtained with different choices for the roots.

have $0 = (\sum x \in X. \beta\ x * exp\ x)$
using *sum0 ..*
also have $\dots = (\sum x \in X. restrict\ \beta\ X\ x * exp\ x)$
by (*intro sum.cong auto*)
also have $\dots dvd\ (\prod f \in X \rightarrow_E Roots. \sum x \in X. f\ x * exp\ x)$
by (*rule dvd-prodI*) (*use $\langle \beta\ ' X \subseteq Roots \rangle$ in $\langle auto\ simp: id-def \rangle$*)
also have $\dots = (\sum f \in (X \rightarrow_E Roots) \rightarrow_E X. \prod g \in X \rightarrow_E Roots. g\ (f\ g) * exp\ (f\ g))$
by (*rule prod-sum-PiE*) *auto*
also have $\dots = (\sum f \in (X \rightarrow_E Roots) \rightarrow_E X. (\prod g \in X \rightarrow_E Roots. g\ (f\ g)) * exp\ (\sum g \in X \rightarrow_E Roots. f\ g))$
by (*simp add: prod.distrib exp-sum*)
also have $\dots = (\sum (Y, f) \in Sigma\ Roots\text{-}ms\ tuples. (\prod g \in X \rightarrow_E Roots. g\ (f\ g)) * exp\ (\sum g \in X \rightarrow_E Roots. f\ g))$
by (*intro sum.reindex-bij-witness*[*of - snd $\lambda f. (image\ mset\ f\ (mset\ set\ (X \rightarrow_E Roots)), f$)*])
(auto simp: Roots-ms-def tuples-def)
also have $\dots = (\sum (Y, f) \in Sigma\ Roots\text{-}ms\ tuples. (\prod g \in X \rightarrow_E Roots. g\ (f\ g)) * exp\ (\sum \# Y))$
by (*intro sum.cong*) (*auto simp: tuples-def sum-unfold-sum-mset*)
also have $\dots = (\sum Y \in Roots\text{-}ms. \beta' Y * exp\ (\sum \# Y))$
unfolding β' -def *sum-distrib-right* **by** (*rule sum.Sigma* [*symmetric*]) *auto*
also have $\dots = (\sum Y \in Roots\text{-}ms. of\text{-}rat\ (\beta'' Y) * exp\ (\sum \# Y))$
by (*intro sum.cong*) (*auto simp: β''*)


```

also have ... = ( $\sum_{Y \in \text{Roots-ms.}} \text{Sum-any } (\lambda \alpha. \text{of-rat } (\beta'' Y \text{ when } \sum_{\#} Y = \alpha) * \text{exp } \alpha)$ )
proof (rule sum.cong, goal-cases)
  case (2 Y)
    have  $\text{Sum-any } (\lambda \alpha. \text{of-rat } (\beta'' Y \text{ when } \sum_{\#} Y = \alpha) * \text{exp } \alpha) =$ 
      ( $\sum_{\alpha \in \{\sum_{\#} Y\}} \text{of-rat } (\beta'' Y \text{ when } \sum_{\#} Y = \alpha) * \text{exp } \alpha$ )
      by (intro Sum-any.expand-superset) auto
    thus ?case by simp
  qed auto
also have ... =  $\text{Sum-any } (\lambda \alpha. \text{of-rat } (\beta''' \alpha) * \text{exp } \alpha)$ 
  unfolding  $\beta'''$ -def of-rat-sum sum-distrib-right by (subst Sum-any-sum-swap)
auto
also have ... = ( $\sum_{\alpha \mid \beta''' \alpha \neq 0.} \text{of-rat } (\beta''' \alpha) * \text{exp } \alpha$ )
  by (intro Sum-any.expand-superset finite-subset[OF supp- $\beta'''$ ]) auto
finally have ( $\sum_{\alpha \mid \beta''' \alpha \neq 0.} \text{of-rat } (\beta''' \alpha) * \text{exp } \alpha = 0$ )
  by auto

```

We are now in the situation of our previous version of the theorem and can apply it to find that all the coefficients are zero.

```

have  $\{\alpha. \beta''' \alpha \neq 0\} = \{\}$ 
proof (rule Hermite-Lindemann-aux3)
  show finite  $\{\alpha. \beta''' \alpha \neq 0\}$ 
  by (rule finite-subset[OF supp- $\beta'''$ ]) auto
next
  show ( $\sum_{\alpha \mid \beta''' \alpha \neq 0.} \text{of-rat } (\beta''' \alpha) * \text{exp } \alpha = 0$ )
  by fact
next
  fix  $\alpha$  assume  $\alpha \in \{\alpha. \beta''' \alpha \neq 0\}$ 
  then obtain Y where  $Y: Y \in \text{Roots-ms } \alpha = \text{sum-mset } Y$ 
  using supp- $\beta'''$  by auto
  thus algebraic  $\alpha$  using alg1
  by (auto simp: Roots-ms-def)
qed auto

```

However, similarly to before, we can show that the coefficient corresponding to the term with the lexicographically greatest exponent (which is obtained by picking the term with the lexicographically greatest term in each of the factors of our big product) is non-zero.

```

moreover have  $\exists \alpha. \beta''' \alpha \neq 0$ 
proof –
  define  $\alpha$ -max where  $\alpha$ -max = complex-lex.Max X
  have [simp]:  $\alpha$ -max  $\in X$ 
  unfolding  $\alpha$ -max-def using  $\langle X \neq \{\} \rangle$  by (intro complex-lex.Max-in) auto
  define Y-max :: complex multiset where Y-max = replicate-mset (n ^ m)
 $\alpha$ -max
  define f-max where f-max = restrict ( $\lambda \cdot. \alpha$ -max) ( $X \rightarrow_E \text{Roots}$ )
  have [simp]: Y-max  $\in \text{Roots-ms}$ 
  by (auto simp: Y-max-def Roots-ms-def)
  have tuples Y-max =  $\{f\text{-max}\}$ 

```

proof safe
have $\text{image-mset } (\lambda \in X \rightarrow_E \text{Roots}. \alpha\text{-max}) (\text{mset-set } (X \rightarrow_E \text{Roots})) =$
 $\text{image-mset } (\lambda \cdot. \alpha\text{-max}) (\text{mset-set } (X \rightarrow_E \text{Roots}))$
by $(\text{intro image-mset-cong}) \text{ auto}$
thus $f\text{-max} \in \text{tuples } Y\text{-max}$
by $(\text{auto simp: f-max-def tuples-def Y-max-def image-mset-const-eq})$
next
fix f **assume** $f \in \text{tuples } Y\text{-max}$
hence $f: f \in (X \rightarrow_E \text{Roots}) \rightarrow_E X \text{ image-mset } f (\text{mset-set } (X \rightarrow_E \text{Roots}))$
 $= Y\text{-max}$
by $(\text{auto simp: tuples-def})$
hence $\forall g \in \# \text{mset-set } (X \rightarrow_E \text{Roots}). f g = \alpha\text{-max}$
by $(\text{intro image-mset-eq-replicate-msetD}[\text{where } n = n \wedge m]) (\text{auto simp:}$
 $Y\text{-max-def})$
thus $f = f\text{-max}$
using f **by** $(\text{auto simp: Y-max-def fun-eq-iff f-max-def})$
qed

have $\beta''' (\text{of-nat } (n \wedge m) * \alpha\text{-max}) = (\sum Y \in \text{Roots-ms}. \beta'' Y \text{ when } \sum \# Y =$
 $\text{of-nat } (n \wedge m) * \alpha\text{-max})$
unfolding $\beta''' \text{-def Roots-ms-def ..}$
also have $\sum \# Y \neq \text{of-nat } n \wedge m * \alpha\text{-max}$ **if** $Y \in \text{Roots-ms}$ $Y \neq Y\text{-max}$ **for** Y
proof -
have $\neg \text{set-mset } Y \subseteq \{\alpha\text{-max}\}$
using $\text{set-mset-subset-singletonD}[\text{of } Y \alpha\text{-max}] \text{ that}$
by $(\text{auto simp: Roots-ms-def Y-max-def split: if-splits})$
then obtain y **where** $y: y \in \# Y$ $y \neq \alpha\text{-max}$
by auto
have $y \in X \text{ set-mset } (Y - \{\#y\}) \subseteq X$
using y **that** **by** $(\text{auto simp: Roots-ms-def dest: in-diffD})$
hence $y \leq_{\mathbb{C}} \alpha\text{-max}$
using y **unfolding** $\alpha\text{-max-def}$ **by** $(\text{intro complex-lex.Max-ge}) \text{ auto}$
with y **have** $y <_{\mathbb{C}} \alpha\text{-max}$
by auto
have $*$: $Y = \{\#y\} + (Y - \{\#y\})$
using y **by** simp
have $\text{sum-mset } Y = y + \text{sum-mset } (Y - \{\#y\})$
by $(\text{subst } *) \text{ auto}$
also have $\dots <_{\mathbb{C}} \alpha\text{-max} + \text{sum-mset } (Y - \{\#y\})$
by $(\text{intro complex-lex.add-strict-right-mono}) \text{ fact}$
also have $\dots \leq_{\mathbb{C}} \alpha\text{-max} + \text{sum-mset } (\text{replicate-mset } (n \wedge m - 1) \alpha\text{-max})$
unfolding $\alpha\text{-max-def}$ **using** $\text{that } y \in \text{set-mset } (Y - \{\#y\}) \subseteq X$
by $(\text{intro complex-lex.add-left-mono sum-mset-mono-complex-lex}$
 $\text{rel-mset-replicate-mset-right complex-lex.Max-ge})$
 $(\text{auto simp: Roots-ms-def size-Diff-singleton})$
also have $\dots = \text{of-nat } (\text{Suc } (n \wedge m - 1)) * \alpha\text{-max}$
by $(\text{simp add: algebra-simps})$
also have $\text{Suc } (n \wedge m - 1) = n \wedge m$
using $\langle n > 0 \rangle$ **by** simp

finally show *?thesis by simp*
qed
hence $(\sum_{Y \in \text{Roots-ms.}} \beta'' Y \text{ when } \sum_{\#} Y = \text{of-nat } (n \wedge m) * \alpha\text{-max}) =$
 $(\sum_{Y \in \{Y\text{-max}\}} \beta'' Y \text{ when } \sum_{\#} Y = \text{of-nat } (n \wedge m) * \alpha\text{-max})$
by *(intro sum.mono-neutral-right ballI) auto*
also have $\dots = \beta'' Y\text{-max}$
by *(auto simp: when-def Y-max-def)*
also have $\text{of-rat } \dots = \beta' Y\text{-max}$
using $\beta''[\text{of } Y\text{-max}]$ **by** *auto*
also have $\dots = (\prod_{g \in X \rightarrow_E \text{Roots. } g} (f\text{-max } g))$
by *(auto simp: β' -def tuples Y-max = {f-max})*
also have $\dots = (\prod_{g \in X \rightarrow_E \text{Roots. } g} \alpha\text{-max})$
by *(intro prod.cong) (auto simp: f-max-def)*
also have $\dots \neq 0$
using $\langle 0 \notin \text{Roots} \rangle \langle \alpha\text{-max} \in X \rangle$ **by** *(intro prod-nonzeroI) (metis PiE-mem)*
finally show *?thesis by blast*
qed

ultimately show *False by blast*
qed

8.4 The final theorem

We now additionally allow some of the β_i to be zero:

lemma *Hermite-Lindemann'*:

fixes $\beta :: \text{complex} \Rightarrow \text{complex}$

assumes *finite X*

assumes $\bigwedge x. x \in X \implies \text{algebraic } x$

assumes $\bigwedge x. x \in X \implies \text{algebraic } (\beta x)$

assumes $(\sum_{x \in X. \beta x * \exp x) = 0$

shows $\forall x \in X. \beta x = 0$

proof –

have $\{x \in X. \beta x \neq 0\} = \{\}$

proof *(rule Hermite-Lindemann-aux4)*

have $(\sum x \mid x \in X \wedge \beta x \neq 0. \beta x * \exp x) = (\sum_{x \in X. \beta x * \exp x)$

by *(intro sum.mono-neutral-left assms(1)) auto*

also have $\dots = 0$

by *fact*

finally show $(\sum x \mid x \in X \wedge \beta x \neq 0. \beta x * \exp x) = 0$.

qed *(use assms in auto)*

thus *?thesis by blast*

qed

Lastly, we switch to indexed summation in order to obtain a version of the theorem that is somewhat nicer to use:

theorem *Hermite-Lindemann*:

fixes $\alpha \beta :: 'a \Rightarrow \text{complex}$

assumes *finite I*

```

assumes  $\bigwedge x. x \in I \implies \text{algebraic } (\alpha x)$ 
assumes  $\bigwedge x. x \in I \implies \text{algebraic } (\beta x)$ 
assumes inj-on  $\alpha I$ 
assumes  $(\sum_{x \in I}. \beta x * \text{exp } (\alpha x)) = 0$ 
shows  $\forall x \in I. \beta x = 0$ 
proof –
  define f where  $f = \text{inv-into } I \alpha$ 
  have [simp]:  $f (\alpha x) = x$  if  $x \in I$  for  $x$ 
    using that by (auto simp: f-def inv-into-f-f [OF assms(4)])
  have  $\forall x \in \alpha ' I. (\beta \circ f) x = 0$ 
  proof (rule Hermite-Lindemann')
    have  $0 = (\sum_{x \in I}. \beta x * \text{exp } (\alpha x))$ 
      using assms(5) ..
    also have  $\dots = (\sum_{x \in I}. (\beta \circ f) (\alpha x) * \text{exp } (\alpha x))$ 
      by (intro sum.cong) auto
    also have  $\dots = (\sum_{x \in \alpha ' I}. (\beta \circ f) x * \text{exp } x)$ 
      using assms(4) by (subst sum.reindex) auto
    finally show  $(\sum_{x \in \alpha ' I}. (\beta \circ f) x * \text{exp } x) = 0$  ..
  qed (use assms in auto)
  thus ?thesis by auto
qed

```

The following version using lists instead of sequences is even more convenient to use in practice:

```

corollary Hermite-Lindemann-list:
  fixes xs :: (complex  $\times$  complex) list
  assumes alg:  $\forall (x,y) \in \text{set } xs. \text{algebraic } x \wedge \text{algebraic } y$ 
  assumes distinct: distinct (map snd xs)
  assumes sum0:  $(\sum (c,\alpha) \leftarrow xs. c * \text{exp } \alpha) = 0$ 
  shows  $\forall c \in (\text{fst } ' \text{set } xs). c = 0$ 
proof –
  define n where  $n = \text{length } xs$ 
  have *:  $\forall i \in \{..<n\}. \text{fst } (xs ! i) = 0$ 
  proof (rule Hermite-Lindemann)
    from distinct have inj-on  $(\lambda i. \text{map snd } xs ! i) \{..<n\}$ 
      by (intro inj-on-nth) (auto simp: n-def)
    also have ?this  $\longleftrightarrow$  inj-on  $(\lambda i. \text{snd } (xs ! i)) \{..<n\}$ 
      by (intro inj-on-cong) (auto simp: n-def)
    finally show inj-on  $(\lambda i. \text{snd } (xs ! i)) \{..<n\}$  .
  next
  have  $0 = (\sum (c,\alpha) \leftarrow xs. c * \text{exp } \alpha)$ 
    using sum0 ..
  also have  $\dots = (\sum i < n. \text{fst } (xs ! i) * \text{exp } (\text{snd } (xs ! i)))$ 
    unfolding sum-list-sum-nth
    by (intro sum.cong) (auto simp: n-def case-prod-unfold)
  finally show  $\dots = 0$  ..
  next
  fix i assume  $i: i \in \{..<n\}$ 
  hence  $(\text{fst } (xs ! i), \text{snd } (xs ! i)) \in \text{set } xs$ 

```

```

    by (auto simp: n-def)
  with alg show algebraic (fst (xs ! i)) algebraic (snd (xs ! i))
    by blast+
qed auto

show ?thesis
proof (intro ballI, elim imageE)
  fix c x assume cx: c = fst x x ∈ set xs
  then obtain i where i ∈ {..

```

8.5 The traditional formulation of the theorem

What we proved above was actually Baker’s reformulation of the theorem. Thus, we now also derive the original one, which uses linear independence and algebraic independence.

It states that if $\alpha_1, \dots, \alpha_n$ are algebraic numbers that are linearly independent over \mathbf{Z} , then $e^{\alpha_1}, \dots, e^{\alpha_n}$ are algebraically independent over \mathbf{Q} .

Linear independence over the integers is just independence of a set of complex numbers when viewing the complex numbers as a \mathbf{Z} -module.

definition *linearly-independent-over-int* :: 'a :: field-char-0 set \Rightarrow bool **where**
linearly-independent-over-int = module.independent ($\lambda r x.$ of-int r * x)

Algebraic independence over the rationals means that the given set X of numbers fulfils no non-trivial polynomial equation with rational coefficients, i.e. there is no non-zero multivariate polynomial with rational coefficients that, when inserting the numbers from X , becomes zero.

Note that we could easily replace ‘rational coefficients’ with ‘algebraic coefficients’ here and the proof would still go through without any modifications.

definition *algebraically-independent-over-rat* :: nat \Rightarrow (nat \Rightarrow 'a :: field-char-0) \Rightarrow bool **where**
algebraically-independent-over-rat n a \longleftrightarrow
 $(\forall p.$ vars p \subseteq {..\wedge $(\forall m.$ coeff p m \in $\mathbf{Q}) \wedge$ insertion a p = 0 \longrightarrow p = 0)

corollary *Hermite-Lindemann-original*:

```

fixes n :: nat and  $\alpha$  :: nat  $\Rightarrow$  complex
assumes inj-on  $\alpha$  {..\bigwedge i. i < n \implies algebraic ( $\alpha$  i)
assumes linearly-independent-over-int ( $\alpha$  ' {..\lambda i. exp ( $\alpha$  i))
unfolding algebraically-independent-over-rat-def
proof safe

```

```

fix p assume p: vars p  $\subseteq$  {.. $n$ }  $\forall$  m. coeff p m  $\in$   $\mathbb{Q}$  insertion ( $\lambda$ i. exp ( $\alpha$  i)) p
= 0
define  $\alpha'$  where  $\alpha' = (\lambda$ m.  $\sum$  i<n. of-nat (lookup m i) *  $\alpha$  i)
define I where I = {m. coeff p m  $\neq$  0}

have lookup-eq-0: lookup m i = 0 if m  $\in$  I i  $\notin$  {.. $n$ } for i m
proof –
  have keys m  $\subseteq$  vars p
  using that coeff-notin-vars[of m p] by (auto simp: I-def)
  thus lookup m i = 0
  using in-keys-iff[of i m] that p(1) by blast
qed

have  $\forall$  x $\in$ I. coeff p x = 0
proof (rule Hermite-Lindemann)
  show finite I
  by (auto simp: I-def)
next
  show algebraic ( $\alpha'$  m) if m  $\in$  I for m
  unfolding  $\alpha'$ -def using assms(2) by fastforce
next
  show algebraic (coeff p m) if m  $\in$  I for m
  unfolding  $\alpha'$ -def using p(2) by blast
next
  show inj-on  $\alpha'$  I
  proof
    fix m1 m2 assume m12: m1  $\in$  I m2  $\in$  I  $\alpha'$  m1 =  $\alpha'$  m2
    define lu :: (nat  $\Rightarrow_0$  nat)  $\Rightarrow$  nat  $\Rightarrow$  int where lu = ( $\lambda$ m i. int (lookup m i))
    interpret int: Modules.module  $\lambda$ r x. of-int r * (x :: complex)
    by standard (auto simp: algebra-simps of-rat-mult of-rat-add)
    define idx where idx = inv-into {.. $n$ }  $\alpha$ 

    have lu m1 i = lu m2 i if i < n for i
    proof –
      have lu m1 (idx ( $\alpha$  i)) – lu m2 (idx ( $\alpha$  i)) = 0
      proof (rule int.independentD)
        show int.independent ( $\alpha$  ‘ {.. $n$ })
        using assms(3) by (simp add: linearly-independent-over-int-def)
      next
        have ( $\sum$  x $\in$  $\alpha$ {.. $n$ }. of-int (lu m1 (idx x) – lu m2 (idx x)) * x) =
          ( $\sum$  i<n. of-int (lu m1 (idx ( $\alpha$  i)) – lu m2 (idx ( $\alpha$  i))) *  $\alpha$  i)
        using assms(1) by (subst sum.reindex) auto
        also have ... = ( $\sum$  i<n. of-int (lu m1 i – lu m2 i) *  $\alpha$  i)
        by (intro sum.cong) (auto simp: idx-def inv-into-f-f[OF assms(1)])
        also have ... = 0
        using m12 by (simp add:  $\alpha'$ -def ring-distrib of-rat-diff sum-subtractf
lu-def)
      finally show ( $\sum$  x $\in$  $\alpha$ {.. $n$ }. of-int (lu m1 (idx x) – lu m2 (idx x)) * x)
= 0

```

```

    by (simp add:  $\alpha'$ -def ring-distrib of-rat-diff sum-subtractf lu-def)
  qed (use that in auto)
  thus ?thesis
    using that by (auto simp: idx-def inv-into-f-f[OF assms(1)])
  qed
  hence lookup m1 i = lookup m2 i for i
    using m12 by (cases i < n) (auto simp: lu-def lookup-eq-0)
  thus m1 = m2
    by (rule poly-mapping-eqI)
  qed
next
  have 0 = insertion ( $\lambda i. \exp (\alpha i)$ ) p
    using p(3) ..
  also have ... = ( $\sum m \in I. \text{coeff } p \ m * \text{Prod-any } (\lambda i. \exp (\alpha i) \wedge \text{lookup } m \ i)$ )
    unfolding insertion-altdef by (rule Sum-any.expand-superset) (auto simp:
I-def)
  also have ... = ( $\sum m \in I. \text{coeff } p \ m * \exp (\alpha' m)$ )
  proof (intro sum.cong, goal-cases)
    case (2 m)
    have Prod-any ( $\lambda i. \exp (\alpha i) \wedge \text{lookup } m \ i$ ) = ( $\prod i < n. \exp (\alpha i) \wedge \text{lookup } m$ 
i)
      using 2 lookup-eq-0[of m] by (intro Prod-any.expand-superset; force)
    also have ... =  $\exp (\alpha' m)$ 
      by (simp add: exp-sum exp-of-nat-mult  $\alpha'$ -def)
    finally show ?case by simp
  qed simp-all
  finally show ( $\sum m \in I. \text{coeff } p \ m * \exp (\alpha' m)$ ) = 0 ..
  qed
  thus p = 0
    by (intro mpoly-eqI) (auto simp: I-def)
  qed

```

8.6 Simple corollaries

Now, we derive all the usual obvious corollaries of the theorem in the obvious way.

First, the exponential of a non-zero algebraic number is transcendental.

corollary *algebraic-exp-complex-iff*:

assumes *algebraic* x

shows *algebraic* ($\exp x :: \text{complex}$) $\longleftrightarrow x = 0$

using *Hermite-Lindemann-list*[of [(1, x), ($-\exp x$, 0)]] *assms* **by** *auto*

More generally, any sum of exponentials with algebraic coefficients and exponents is transcendental if the exponents are all distinct and non-zero and at least one coefficient is non-zero.

corollary *sum-of-exp-transcendentalI*:

fixes $xs :: (\text{complex} \times \text{complex}) \text{ list}$

assumes $\forall (x,y) \in \text{set } xs. \text{algebraic } x \wedge \text{algebraic } y \wedge y \neq 0$

```

assumes  $\exists x \in \text{fst}'\text{set } xs. x \neq 0$ 
assumes distinct:  $\text{distinct } (\text{map } \text{snd } xs)$ 
shows  $\neg \text{algebraic } (\sum (c, \alpha) \leftarrow xs. c * \exp \alpha)$ 
proof
  define S where  $S = (\sum (c, \alpha) \leftarrow xs. c * \exp \alpha)$ 
  assume S: algebraic S
  have  $\forall c \in \text{fst}'\text{set } ((-S, 0) \# xs). c = 0$ 
  proof (rule Hermite-Lindemann-list)
    show  $(\sum (c, \alpha) \leftarrow (-S, 0) \# xs. c * \exp \alpha) = 0$ 
    by (auto simp: S-def)
  qed (use S assms in auto)
  with assms(2) show False
  by auto
qed

```

Any complex logarithm of an algebraic number other than 1 is transcendental (no matter which branch cut).

```

corollary transcendental-complex-logarithm:
  assumes algebraic x exp y = (x :: complex) x ≠ 1
  shows  $\neg \text{algebraic } y$ 
  using algebraic-exp-complex-iff[of y] assms by auto

```

In particular, this holds for the standard branch of the logarithm.

```

corollary transcendental-Ln:
  assumes algebraic x x ≠ 0 x ≠ 1
  shows  $\neg \text{algebraic } (\text{Ln } x)$ 
  by (rule transcendental-complex-logarithm) (use assms in auto)

```

The transcendence of e and π , which I have already formalised directly in other AFP entries, now follows as a simple corollary.

```

corollary exp-1-complex-transcendental:  $\neg \text{algebraic } (\exp 1 :: \text{complex})$ 
  by (subst algebraic-exp-complex-iff) auto

```

```

corollary pi-transcendental:  $\neg \text{algebraic } \pi$ 
proof –
  have  $\neg \text{algebraic } (i * \pi)$ 
  by (rule transcendental-complex-logarithm[of -1]) auto
  thus ?thesis by simp
qed

```

8.7 Transcendence of the trigonometric and hyperbolic functions

In a similar fashion, we can also prove the transcendence of all the trigonometric and hyperbolic functions such as \sin , \tan , \sinh , \arcsin , etc.

```

lemma transcendental-sinh:
  assumes algebraic z z ≠ 0

```


shows \neg algebraic ($\sinh z :: \text{complex}$)
proof –
have \neg algebraic ($\sum (a,b) \leftarrow [(1/2, z), (-1/2, -z)]. a * \exp b$)
using *assms* **by** (*intro sum-of-exp-transcendentalI*) *auto*
also have ($\sum (a,b) \leftarrow [(1/2, z), (-1/2, -z)]. a * \exp b$) = $\sinh z$
by (*simp add: sinh-def field-simps scaleR-conv-of-real*)
finally show ?thesis .
qed

lemma *transcendental-cosh*:
assumes algebraic $z z \neq 0$
shows \neg algebraic ($\cosh z :: \text{complex}$)
proof –
have \neg algebraic ($\sum (a,b) \leftarrow [(1/2, z), (1/2, -z)]. a * \exp b$)
using *assms* **by** (*intro sum-of-exp-transcendentalI*) *auto*
also have ($\sum (a,b) \leftarrow [(1/2, z), (1/2, -z)]. a * \exp b$) = $\cosh z$
by (*simp add: cosh-def field-simps scaleR-conv-of-real*)
finally show ?thesis .
qed

lemma *transcendental-sin*:
assumes algebraic $z z \neq 0$
shows \neg algebraic ($\sin z :: \text{complex}$)
unfolding *sin-conv-sinh* **using** *transcendental-sinh*[of $i * z$] *assms* **by** *simp*

lemma *transcendental-cos*:
assumes algebraic $z z \neq 0$
shows \neg algebraic ($\cos z :: \text{complex}$)
unfolding *cos-conv-cosh* **using** *transcendental-cosh*[of $i * z$] *assms* **by** *simp*

lemma *tan-square-neq-neg1*: $\tan (z :: \text{complex}) ^ 2 \neq -1$
proof
assume $\tan z ^ 2 = -1$
hence $\sin z ^ 2 = -(\cos z ^ 2)$
by (*auto simp: tan-def divide-simps split: if-splits*)
also have $\cos z ^ 2 = 1 - \sin z ^ 2$
by (*simp add: cos-squared-eq*)
finally show *False*
by *simp*
qed

lemma *transcendental-tan*:
assumes algebraic $z z \neq 0$
shows \neg algebraic ($\tan z :: \text{complex}$)
proof
assume algebraic ($\tan z$)

have *nz1: real-of-int $n + 1 / 2 \neq 0$ for n*

proof –
have $\text{real-of-int } (2 * n + 1) / \text{real-of-int } 2 \notin \mathbb{Z}$
by (*intro fraction-not-in-ints*) *auto*
also have $\text{real-of-int } (2 * n + 1) / \text{real-of-int } 2 = \text{real-of-int } n + 1 / 2$
by *simp*
finally show $\dots \neq 0$
by *auto*
qed

have $\text{nz2: } 1 + \tan z^2 \neq 0$
using *tan-square-neq-neg1[of z]* **by** (*subst add-eq-0-iff*)

have $\text{nz3: } \cos z \neq 0$
proof
assume $\cos z = 0$
then obtain n **where** $z = \text{complex-of-real } (\text{real-of-int } n * \pi) + \text{complex-of-real } \pi / 2$
by (*subst (asm) cos-eq-0*) *blast*
also have $\dots = \text{complex-of-real } ((\text{real-of-int } n + 1 / 2) * \pi)$
by (*simp add: ring-distrib*)
also have $\text{algebraic } \dots \iff \text{algebraic } ((\text{real-of-int } n + 1 / 2) * \pi)$
by (*rule algebraic-of-real-iff*)
also have $\neg \text{algebraic } ((\text{real-of-int } n + 1 / 2) * \pi)$
using *nz1[of n] transcendental-pi* **by** *simp*
finally show *False* **using** *assms(1)* **by** *contradiction*
qed

from nz3 **have** $*$: $\sin z^2 = \tan z^2 * \cos z^2$
by (*simp add: tan-def field-simps*)
also have $\cos z^2 = 1 - \sin z^2$
by (*simp add: cos-squared-eq*)
finally have $\sin z^2 * (1 + \tan z^2) = \tan z^2$
by (*simp add: algebra-simps*)
hence $\sin z^2 = \tan z^2 / (1 + \tan z^2)$
using nz2 **by** (*simp add: field-simps*)
also have $\text{algebraic } (\tan z^2 / (1 + \tan z^2))$
using $\langle \text{algebraic } (\tan z) \rangle$ **by** *auto*
finally have $\text{algebraic } (\sin z^2)$.
hence $\text{algebraic } (\sin z)$
by *simp*
thus *False*
using *transcendental-sin[OF (algebraic z) (z ≠ 0)]* **by** *contradiction*
qed

lemma *transcendental-cot*:
assumes $\text{algebraic } z \ z \neq 0$
shows $\neg \text{algebraic } (\cot z :: \text{complex})$
proof –
have $\neg \text{algebraic } (\tan z)$

by (rule transcendental-tan) fact+
 also have algebraic (tan z) \longleftrightarrow algebraic (inverse (tan z))
 by simp
 also have inverse (tan z) = cot z
 by (simp add: cot-def tan-def)
 finally show ?thesis .
 qed

lemma transcendental-tanh:
 assumes algebraic z z \neq 0 cosh z \neq 0
 shows \neg algebraic (tanh z :: complex)
 using transcendental-tan[of i * z] assms unfolding tanh-conv-tan by simp

lemma transcendental-Arcsin:
 assumes algebraic z z \neq 0
 shows \neg algebraic (Arcsin z)
proof –
 have i * z + csqrt (1 - z²) \neq 0
 using Arcsin-body-lemma by blast
 moreover have i * z + csqrt (1 - z²) \neq 1
proof
 assume i * z + csqrt (1 - z²) = 1
 hence Arcsin z = 0
 by (simp add: Arcsin-def)
 hence sin (Arcsin z) = 0
 by (simp only: sin-zero)
 also have sin (Arcsin z) = z
 by simp
 finally show False using (z \neq 0) by simp
qed
 ultimately have \neg algebraic (Ln (i * z + csqrt (1 - z²)))
 using assms by (intro transcendental-Ln) auto
 thus ?thesis
 by (simp add: Arcsin-def)
qed

lemma transcendental-Arccos:
 assumes algebraic z z \neq 1
 shows \neg algebraic (Arccos z)
proof –
 have z + i * csqrt (1 - z²) \neq 0
 using Arccos-body-lemma by blast
 moreover have z + i * csqrt (1 - z²) \neq 1
proof
 assume z + i * csqrt (1 - z²) = 1
 hence Arccos z = 0
 by (simp add: Arccos-def)
 hence cos (Arccos z) = 1
 by (simp only: cos-zero)
qed

```

also have  $\cos (\operatorname{Arccos} z) = z$ 
  by simp
finally show False using  $\langle z \neq 1 \rangle$  by simp
qed
ultimately have  $\neg \text{algebraic} (\operatorname{Ln} (z + i * \operatorname{csqrt} (1 - z^2)))$ 
  using assms by (intro transcendental-Ln) auto
thus ?thesis
  by (simp add: Arccos-def)
qed

```

```

lemma transcendental-Arctan:
  assumes algebraic  $z \ z \notin \{0, i, -i\}$ 
  shows  $\neg \text{algebraic} (\operatorname{Arctan} z)$ 
proof -
  have  $i * z \neq 1 \ 1 + i * z \neq 0$ 
    using assms(2) by (auto simp: complex-eq-iff)
  hence  $\neg \text{algebraic} (\operatorname{Ln} ((1 - i * z) / (1 + i * z)))$ 
    using assms by (intro transcendental-Ln) auto
  thus ?thesis
    by (simp add: Arctan-def)
qed

```

end

References

- [1] A. Baker. *Transcendental Number Theory*. Cambridge Mathematical Library. Cambridge University Press, 1975.
- [2] S. Bernard. Formalization of the Lindemann-Weierstrass theorem. In M. Ayala-Rincón and C. A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2017.
- [3] R. Steinberg and R. M. Redheffer. Analytic proof of the Lindemann theorem. *Pacific Journal of Mathematics*, 2(2):231–242, 1952.