

# The Hermite–Lindemann–WeierstraSS Transcendence Theorem

Manuel Eberl

September 13, 2023

## 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  $\pi$  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

<b>1</b>	<b>Divisibility of algebraic integers</b>	<b>3</b>
<b>2</b>	<b>Auxiliary facts about univariate polynomials</b>	<b>7</b>
<b>3</b>	<b>The lexicographic ordering on complex numbers</b>	<b>19</b>
<b>4</b>	<b>Additional facts about multivariate polynomials</b>	<b>21</b>
4.1	Miscellaneous . . . . .	21
4.2	Converting a univariate polynomial into a multivariate one . . . . .	22
<b>5</b>	<b>More facts about algebraic numbers</b>	<b>25</b>
5.1	Miscellaneous . . . . .	25
5.2	Turning an algebraic number into an algebraic integer . . . . .	28
5.3	Multiplying an algebraic number with a suitable integer turns it into an algebraic integer. . . . .	28
<b>6</b>	<b>Miscellaneous facts</b>	<b>30</b>
<b>7</b>	<b>The Hermite–Lindemann–WeierstraSS Transcendence Theorem</b>	<b>35</b>
7.1	Main proof . . . . .	36
7.2	Removing the restriction of full sets of conjugates . . . . .	58
7.3	Removing the restriction to integer coefficients . . . . .	70
7.4	The final theorem . . . . .	80
7.5	The traditional formulation of the theorem . . . . .	82
7.6	Simple corollaries . . . . .	84
7.7	Transcendence of the trigonometric and hyperbolic functions	86

# 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  $y \iff (x = 0 \longrightarrow y = 0) \wedge$  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  $x \iff x = 0$ 
  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  $x \iff$  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**  $(\text{elim Ints-cases})$   
**hence**  $\text{of-int } y = (\text{of-int } (x * n) :: 'a)$   
**unfolding**  $\text{of-int-mult}$  **using**  $\text{nz}$  **by**  $(\text{auto simp: field-simps})$   
**hence**  $y = x * n$   
**by**  $(\text{subst (asm) of-int-eq-iff})$   
**thus**  $x \text{ dvd } y$   
**by**  $\text{auto}$   
**qed**  $\text{blast}$

**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**  $\text{of-int-of-nat-eq}$  .  
**qed**

**lemma**  $\text{alg-dvd-add}$   $[\text{intro}]$ :  
**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**  $\text{alg-dvd-def}$  **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**  $\text{alg-dvd-uminus-right}$   $[\text{of } x y]$   $\text{alg-dvd-uminus-right}$   $[\text{of } x -y]$  **by**  $\text{auto}$

**lemma**  $\text{alg-dvd-diff}$   $[\text{intro}]$ :  
**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**  $\text{alg-dvd-def}$  **by**  $(\text{auto simp: diff-divide-distrib})$

**lemma**  $\text{alg-dvd-triv-left}$   $[\text{intro}]$ :  $\text{algebraic-int } y \Longrightarrow x \text{ alg-dvd } x * y$   
**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**  $\bigwedge x. x \in A \implies y \text{ alg-dvd } f x$   
**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**  $\bigwedge x. x \in A - \{x'\} \implies y \text{ alg-dvd } f x$   
**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*:

$\text{coeff } ((\text{pderiv } \sim m) f) n = \text{pochhammer } (\text{of-nat } (\text{Suc } n)) m * \text{coeff } f (n + m)$   
**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**  $\text{fact } m \text{ alg-dvd } \text{poly } ((\text{pderiv } \sim k) p) x$

```

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 content (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 :: rat poly
  assumes p ≠ 0
  obtains q lc where p = Polynomial.smult lc (of-int-poly q) lc > 0 content q =
  1
proof -
  define lc where lc = fst (rat-to-normalized-int-poly p)
  define q where q = snd (rat-to-normalized-int-poly p)
  have eq: 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 ^ 2 dvd p
  then obtain r where qr: p = q ^ 2 * r
    by (elim dvdE)
  have q dvd 1 ∨ q * r dvd 1
    by (intro irreducibleD[OF assms]) (use qr in ⟨simp-all add: power2-eq-square
mult-ac⟩)
  thus q dvd 1
    by (meson dvd-mult-left)
qed

```

```

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

```



```

    hence Polynomial.degree  $[-x, 1:] \leq \textit{Polynomial.degree}$   $(1 :: 'a \textit{ poly})$ 
      by  $(\textit{rule dvd-imp-degree-le}) \textit{ auto}$ 
    thus False by simp
  qed
  thus Polynomial.order  $x \ p = 0 \vee \textit{Polynomial.order}$   $x \ p = 1$ 
    by linarith
qed  $(\textit{use assms in auto})$ 

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

```

```

lemma irreducible-imp-coprime-pderiv:
  fixes  $p :: 'a :: \{idom-divide, semiring-char-0\}$  poly
  assumes irreducible  $p$   $Polynomial.degree\ p \neq 0$ 
  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  $is-unit\ d \vee is-unit\ q$ 
    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  $Rings.coprime\ p\ q$   $p \neq 0$   $q \neq 0$ 
  shows  $poly\ p\ x \neq 0 \vee poly\ q\ x \neq 0$ 
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  $\langle Rings.coprime\ p\ q \rangle$  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  $assms\ gcd-of-int-poly[of\ p\ q, \text{where } ?'a = 'a]$  unfolding coprime-iff-gcd-eq-1
  by simp

```

```

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

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

```

```

lemma squarefree-of-int-polyI:
  assumes squarefree  $p$   $\text{content } p = 1$ 
  shows  $\text{squarefree (of-int-poly } p :: 'a :: \{\text{field-char-0, field-gcd}\} \text{ poly)}$ 
proof –
  have  $\text{Rings.coprime } p (\text{pderiv } p)$ 
    by (rule squarefree-imp-coprime-pderiv) fact+
  hence  $\text{Rings.coprime (of-int-poly } p :: 'a \text{ poly}) (\text{of-int-poly (pderiv } p))$ 
    by (rule coprime-of-int-polyI)
  also have  $\text{of-int-poly (pderiv } p) = \text{pderiv (of-int-poly } p :: 'a \text{ poly)}$ 
    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:

```

$(pderiv \widehat{\sim} n) (pcompose p [:0, c:]) =$   
 $Polynomial.smult (c \widehat{\sim} n) (pcompose ((pderiv \widehat{\sim} n) p) [:0, c:])$   
**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**  $pCons 0 = (\lambda p. 'a\ poly.\ Polynomial.monom\ 1\ 1 * p)$   
**by** (*simp add: Polynomial.monom-altdef*)  
**also have**  $map-poly \dots (poly-y-x p) = Polynomial.smult (Polynomial.monom\ 1\ 1) (poly-y-x p)$   
**by** (*simp add: smult-conv-map-poly*)  
**also have**  $poly \dots [:y:] = Polynomial.monom\ 1\ 1 * poly (poly-y-x p) [:y:]$   
**by** *simp*  
**also have**  $poly a y + poly \dots x = poly (poly (pCons a p) [:x:]) y$   
**by** (*simp add: pCons poly-monom*)  
**finally show** *?case .*  
**qed** *auto*

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

$map-poly\ hom ((pderiv \widehat{\sim} n) p) = (pderiv \widehat{\sim} n) (map-poly\ hom\ p)$   
**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**  $Polynomial.coeff (\prod x \in A. [-f\ x, 1:] \widehat{\sim} e\ x) i =$   
 $(\sum X \mid X \in Pow (SIGMA\ x:A. \{..\leq e\ x\}) \wedge i = sum\ e\ A - card\ X.$   
 $(-1) \widehat{\sim} card\ X * (\prod x \in X. f (fst\ x)))$

**proof** –

**define** *poly-X* **where**  $poly-X = (Polynomial.monom\ 1\ 1 :: 'b\ poly)$   
**have** [*simp*]:  $(-1) \widehat{\sim} n = [:(-1) \widehat{\sim} n :: 'b:]$  **for**  $n :: nat$   
**by** (*simp flip: pCons-one add: poly-const-pow*)  
**have**  $(\prod x \in A. [-f\ x, 1:] \widehat{\sim} e\ x) = (\prod (x,-) \in Sigma\ A (\lambda x. \{..\leq e\ x\}). [-f\ x, 1:])$   
**by** (*subst prod.Sigma [symmetric]*) *auto*  
**also have**  $\dots = (\prod (x,-) \in Sigma\ A (\lambda x. \{..\leq e\ x\}). poly-X - [:f\ x:])$   
**by** (*intro prod.cong*) (*auto simp: poly-X-def monom-altdef*)  
**also have**  $\dots = (\sum X \in Pow (SIGMA\ x:A. \{..\leq e\ x\}).$   
 $Polynomial.smult ((-1) \widehat{\sim} card\ X * (\prod x \in X. f (fst\ x)))$

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

**unfolding** *case-prod-unfold*  
**by** (*subst prod-diff1*) (*auto simp: mult-ac simp flip: coeff-lift-hom.hom-prod*)  
**also have**  $\dots = (\sum X \in \text{Pow } (\text{SIGMA } x:A. \{..\leq e x\}).$   
 $\text{Polynomial.monom } ((- 1) \wedge \text{card } X * (\prod x \in X. f (fst x))) (\text{card } ((\text{SIGMA } x:A. \{..\leq e x\}) - X))$   
**unfolding** *poly-X-def monom-power Polynomial.smult-monom* **by** *simp*  
**also have** *Polynomial.coeff*  $\dots i = (\sum X \in \{X \in \text{Pow } (\text{SIGMA } x:A. \{..\leq e x\}). i$   
 $=$   
 $\text{sum } e A - \text{card } X\}. (- 1) \wedge \text{card } X * (\prod x \in X. f (fst x)))$   
**unfolding** *Polynomial.coeff-sum*  
**proof** (*intro sum.mono-neutral-cong-right ballI, goal-cases*)  
**case** ( $\exists X$ )  
**hence**  $X: X \subseteq (\text{SIGMA } x:A. \{..\leq e x\})$   
**by** *auto*  
**have** *card-le*:  $\text{card } X \leq \text{card } (\text{SIGMA } x:A. \{..\leq e x\})$   
**using**  $X$  **by** (*intro card-mono*) *auto*  
**have** *finite X*  
**by** (*rule finite-subset[OF X]*) *auto*  
**hence**  $\text{card } ((\text{SIGMA } x:A. \{..\leq e x\}) - X) = \text{card } (\text{SIGMA } x:A. \{..\leq e x\}) -$   
 $\text{card } X$   
**using**  $\exists$  **by** (*intro card-Diff-subset*) *auto*  
**also have** *card-eq*:  $\text{card } (\text{SIGMA } x:A. \{..\leq e x\}) = \text{sum } e A$   
**by** (*subst card-SigmaI*) *auto*  
**finally show** *?case*  
**using**  $\exists$  *card-le card-eq* **by** (*auto simp: algebra-simps*)  
**next**  
**case** ( $\not\exists X$ )  
**hence**  $X: X \subseteq (\text{SIGMA } x:A. \{..\leq e x\})$   
**by** *auto*  
**have** *finite X*  
**by** (*rule finite-subset[OF X]*) *auto*  
**hence**  $\text{card } ((\text{SIGMA } x:A. \{..\leq e x\}) - X) = \text{card } (\text{SIGMA } x:A. \{..\leq e x\}) -$   
 $\text{card } X$   
**using**  $\not\exists$  **by** (*intro card-Diff-subset*) *auto*  
**also have** *card-eq*:  $\text{card } (\text{SIGMA } x:A. \{..\leq e x\}) = \text{sum } e A$   
**by** (*subst card-SigmaI*) *auto*  
**finally show** *?case*  
**using**  $\not\exists$  *card-eq* **by** (*auto simp: algebra-simps*)  
**qed** *auto*  
**finally show** *?thesis* .  
**qed**

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

**lemma** *synthetic-div-altdef*:  
**fixes**  $p :: 'a :: \text{field poly}$

**shows** *synthetic-div*  $p \ c = p \ \text{div} \ [-c, 1:]$   
**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*  $(p \ \text{mod} \ [-c, 1:]) < \text{Polynomial.degree} \ [-c, 1:]$   
  **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**  $\text{poly.coeff } (\text{synthetic-div } p \ x) \ i \in A$   
**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*:  $\text{pcompose } (\text{Polynomial.monom } c \ n) \ p = \text{Polynomial.smult } c \ (p \wedge n)$   
**by** (*simp add: monom-altdef pcompose-hom.hom-power pcompose-smult*)

**lemma** *poly-roots-uminus* [*simp*]:  $\text{poly-roots } (-p) = \text{poly-roots } p$   
**using** *poly-roots-smult*[*of -1 p*] **by** (*simp del: poly-roots-smult*)

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

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

**lemma** *poly-roots-power* [*simp*]:  $\text{poly-roots } (p \wedge n) = \text{repeat-mset } n \ (\text{poly-roots } p)$   
**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 (∑ p∈#prime-factorization q. poly-roots p) =
    poly-roots (prod-mset (prime-factorization q))
    by (rule poly-roots-prod-mset [symmetric]) auto
  also have ... = poly-roots (normalize (prod-mset (prime-factorization q)))
    by simp
  also have normalize (prod-mset (prime-factorization q)) = normalize q
    by (rule prod-mset-prime-factorization-weak) auto
  also have poly-roots ... = poly-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) =
    (∑ p∈#prime-factorization q. poly-roots (of-int-poly p))
proof (cases q = 0)
  case [simp]: False

```

```

  have (∑ p∈#prime-factorization q. poly-roots (of-int-poly p :: 'a poly)) =
    poly-roots (∏ p∈#prime-factorization q. of-int-poly p)
    by (subst poly-roots-prod-mset) (auto simp: multiset.map-comp o-def)
  also have (∏ p∈#prime-factorization q. of-int-poly p :: 'a poly) =
    of-int-poly (prod-mset (prime-factorization q))
    by simp
  also have poly-roots ... = poly-roots (of-int-poly (normalize (prod-mset (prime-factorization
q))))
    by (rule poly-roots-of-int-normalize [symmetric])
  also have normalize (prod-mset (prime-factorization q)) = normalize q
    by (rule prod-mset-prime-factorization-weak) auto
  also have poly-roots (of-int-poly ... :: 'a poly) = poly-roots (of-int-poly q)
    by simp
  finally show ?thesis ..
qed auto

```

```

lemma dvd-imp-poly-roots-subset:
  assumes q ≠ 0 p dvd q
  shows poly-roots p ⊆# poly-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:  $|\text{prod-mset } (A :: 'a :: \text{idom-abs-sgn multiset})| = \text{prod-mset}$ 
(image-mset abs A)
  by (induction A) (auto simp: abs-mult)

```

```

lemma content-1-imp-nonconstant-prime-factors:

```

```

  assumes content ( $p :: \text{int poly}$ ) = 1 and  $q \in \text{prime-factors } p$ 
  shows  $\text{Polynomial.degree } q > 0$ 

```

```

proof -

```

```

  let  $?d = \text{Polynomial.degree} :: \text{int poly} \Rightarrow \text{nat}$ 

```

```

  let  $?lc = \text{Polynomial.lead-coeff} :: \text{int poly} \Rightarrow \text{int}$ 

```

```

  define  $P$  where  $P = \text{prime-factorization } p$ 

```

```

  define  $P1$  where  $P1 = \text{filter-mset } (\lambda p. ?d p = 0) P$ 

```

```

  define  $P2$  where  $P2 = \text{filter-mset } (\lambda p. ?d p > 0) P$ 

```

```

  have [simp]:  $p \neq 0$ 

```

```

    using assms by auto

```

```

  have  $1 = \text{content } (\text{normalize } p)$ 

```

```

    using assms by simp

```

```

  also have  $\text{normalize } p = \text{prod-mset } P$ 

```

```

    unfolding  $P\text{-def}$  by (rule prod-mset-prime-factorization [symmetric]) auto

```

```

  also have  $P = \text{filter-mset } (\lambda p. ?d p = 0) P + \text{filter-mset } (\lambda p. ?d p > 0) P$ 

```

```

    by (induction P) auto

```

```

  also have  $\text{prod-mset } \dots = \text{prod-mset } P1 * \text{prod-mset } P2$ 

```

```

    unfolding  $P1\text{-def } P2\text{-def}$  by (subst prod-mset.union) auto

```

```

  also have  $\text{content } \dots = \text{content } (\text{prod-mset } P1) * \text{content } (\text{prod-mset } P2)$ 

```

```

    unfolding content-mult ..

```

```

  also have  $\text{image-mset id } P1 = \text{image-mset } (\lambda q. [?:?lc q:]) P1$ 

```

```

    by (intro image-mset-cong) (auto simp: P1-def elim!: degree-eq-zeroE)

```

```

  hence  $P1 = \text{image-mset } (\lambda q. [?:?lc q:]) P1$ 

```

```

    by simp

```

```

  also have  $\text{content } (\text{prod-mset } \dots) = |\prod_{q \in \#P1}. ?lc q|$ 

```

```

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

```

```

  finally have  $|\prod_{q \in \#P1}. ?lc q| * \text{content } (\text{prod-mset } P2) = 1 ..$ 

```

```

  hence  $|\prod_{q \in \#P1}. ?lc q| \text{ dvd } 1$ 

```

```

    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  $|\text{?lc } q| \text{ dvd } (\prod_{q \in \#P1}. |\text{?lc } q|)$ 

```

```

    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 ... 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

This theory imports both univariate and multivariate polynomials and thereby causes several overlaps in notation of polynomials.

```

theory More-Min-Int-Poly
  imports
    Algebraic-Numbers.Min-Int-Poly
    HOL-Computational-Algebra.Computational-Algebra
    More-Polynomial-HLW
begin

```

```

lemma min-int-poly-squarefree [intro]:
  fixes x :: 'a :: {field-char-0, field-gcd}
  shows squarefree (min-int-poly x)
  by (rule irreducible-imp-squarefree) 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  $\langle \text{gcd (min-int-poly x) p represents x} \rangle$  have  $\neg \text{is-unit (gcd (min-int-poly x) p)}$ 
  by (auto simp: represents-def)
ultimately have min-int-poly x dvd gcd (min-int-poly x) p
  by blast
also have  $\dots \text{ dvd p}$ 
  by blast
finally show min-int-poly x dvd p .
qed auto

end

```

### 3 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  $\longleftrightarrow \text{Re } x < \text{Re } y \vee \text{Re } x = \text{Re } y \wedge \text{Im } x \leq \text{Im } y$ 

```

```

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

```

```

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

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

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

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

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

**lemma**  $\text{rel-mset-replicate-mset-right}$ :  
**assumes**  $\bigwedge x. x \in \# A \implies R \text{ } x \text{ } y \text{ } \text{size } A = n$   
**shows**  $\text{rel-mset } R A (\text{replicate-mset } n \text{ } y)$

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

end

## 4 Additional facts about multivariate polynomials

**theory** *More-Multivariate-Polynomial-HLW*  
**imports** *Power-Sum-Polynomials.Power-Sum-Polynomials-Library*  
**begin**

### 4.1 Miscellaneous

**lemma** *Var-altdef*:  $\text{Var } i = \text{monom } (\text{Poly-Mapping.single } i \ 1) \ 1$   
**by** *transfer'* (*simp add: Var<sub>0</sub>-def*)

**lemma** *Const-conv-monom*:  $\text{Const } c = \text{monom } 0 \ c$   
**by** *transfer'* (*auto simp: Const<sub>0</sub>-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** *symmetric-mpoly*  $A \ (\prod_{x \in X}. f \ x)$   
**unfolding** *symmetric-mpoly-def*  
**proof safe**  
**fix**  $\pi$  **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  $\pi$  refl*)  
**also have**  $\dots = (\prod_{x \in g \ \pi \ 'X}. f \ x)$   
**using** *assms(1)[OF  $\pi$ ]* **by** (*subst prod.reindex*) (*auto simp: permutes-inj-on*)  
**also have**  $g \ \pi \ 'X = X$   
**using** *assms(1)[OF  $\pi$ ]* **by** (*simp add: permutes-image*)  
**finally show**  $\text{mpoly-map-vars } \pi \ (\text{prod } f \ X) = \text{prod } f \ X$  .  
**qed**

## 4.2 Converting a univariate polynomial into a multivariate one

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

$\lambda i\ c\ m.$  if  $Poly\text{-Mapping.keys}\ m \subseteq \{i\}$  then  $c\ (Poly\text{-Mapping.lookup}\ m\ i)$  else  $0$

**proof** *goal-cases*

**case**  $(1\ i\ c)$

**hence**  $fin: finite\ \{n.\ c\ n \neq 0\}$

**by**  $(metis\ eventually\text{-cofinite})$

**show**  $finite\ \{x.\ (if\ keys\ x \subseteq \{i\}\ then\ c\ (lookup\ x\ i)\ else\ 0) \neq 0\}$

**proof**  $(rule\ finite\text{-subset})$

**show**  $finite\ (Poly\text{-Mapping.single}\ i\ ' \{n.\ c\ n \neq 0\})$

**by**  $(intro\ finite\text{-imageI}\ fin)$

**next**

**show**  $\{x.\ (if\ keys\ x \subseteq \{i\}\ then\ c\ (lookup\ x\ i)\ else\ 0) \neq 0\} \subseteq$   
 $Poly\text{-Mapping.single}\ i\ ' \{n.\ c\ n \neq 0\}$

**proof**  $(safe,\ split\ if\text{-splits})$

**fix**  $x :: (nat \Rightarrow_0 nat)$

**assume**  $x: keys\ x \subseteq \{i\}\ c\ (lookup\ x\ i) \neq 0$

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

**by**  $(metis\ Diff\text{-eq}\text{-empty}\text{-iff}\ keys\text{-empty}\text{-iff}\ lookup\text{-single}\text{-eq}$   
 $remove\text{-key}\text{-keys}\ remove\text{-key}\text{-single}\ remove\text{-key}\text{-sum})$

**thus**  $x \in Poly\text{-Mapping.single}\ i\ ' \{n.\ c\ n \neq 0\}$

**using**  $x$  **by**  $blast$

**qed** *auto*

**qed**

**qed**

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

**lemma**  $mpoly\text{-of-poly}\ 0$   $[simp]: mpoly\text{-of-poly}\ i\ 0 = 0$

**by**  $(transfer'\ ,\ transfer)\ auto$

**lemma**  $coeff\text{-mpoly}\text{-of-poly}1$   $[simp]:$

$coeff\ (mpoly\text{-of-poly}\ i\ p)\ (Poly\text{-Mapping.single}\ i\ n) = poly.\text{coeff}\ p\ n$

**by**  $(transfer'\ ,\ transfer')\ auto$

**lemma**  $coeff\text{-mpoly}\text{-of-poly}2$   $[simp]:$

**assumes**  $\neg keys\ x \subseteq \{i\}$

**shows**  $coeff\ (mpoly\text{-of-poly}\ i\ p)\ x = 0$

**using**  $assms$  **by**  $(transfer'\ ,\ transfer')\ auto$

**lemma**  $coeff\text{-mpoly}\text{-of-poly}$ :

$coeff\ (mpoly\text{-of-poly}\ i\ p)\ m =$

$(poly.\text{coeff}\ p\ (Poly\text{-Mapping.lookup}\ m\ i)\ \text{when}\ keys\ m \subseteq \{i\})$

**by**  $(transfer'\ ,\ transfer')\ auto$

**lemma**  $poly\text{-mapping}\text{-single}\text{-eq}\text{-iff}$   $[simp]: Poly\text{-Mapping.single}\ i\ n = 0 \iff n =$

0

by (metis lookup-single-eq single-zero)

**lemma** *mpoly-of-poly-pCons* [simp]:

fixes  $p :: 'a :: \text{semiring-1}$  poly

shows  $\text{mpoly-of-poly } i \text{ (pCons } c \text{ } p) = \text{Const } c + \text{Var } i * \text{mpoly-of-poly } i \text{ } p$

**proof** (rule *mpoly-eqI*)

fix  $\text{mon} :: \text{nat} \Rightarrow_0 \text{nat}$

**define**  $\text{moni} :: \text{nat} \Rightarrow_0 \text{nat}$  **where**  $\text{moni} = \text{Poly-Mapping.single } i \text{ } 1$

**have**  $\text{coeff } (\text{Var } i * \text{mpoly-of-poly } i \text{ } p) \text{ } \text{mon} =$   
 $(\sum l. (1 \text{ when } l = \text{moni}) * (\sum q. \text{coeff } (\text{mpoly-of-poly } i \text{ } p) \text{ } q \text{ when } \text{mon} =$   
 $\text{moni} + q))$

**unfolding** *coeff-mpoly-times prod-fun-def coeff-Var moni-def*

**by** (rule *Sum-any.cong*) (auto simp: *when-def*)

**also have**  $\dots = (\sum a. \text{coeff } (\text{mpoly-of-poly } i \text{ } p) \text{ } a \text{ when } \text{mon} = \text{moni} + a)$

**by** (*subst Sum-any-left-distrib [symmetric]*) *simp-all*

**finally have**  $\text{eq: } \text{coeff } (\text{Var } i * \text{mpoly-of-poly } i \text{ } p) \text{ } \text{mon} = \dots$  .

**show**  $\text{coeff } (\text{mpoly-of-poly } i \text{ (pCons } c \text{ } p)) \text{ } \text{mon} = \text{coeff } (\text{Const } c + \text{Var } i * \text{mpoly-of-poly } i \text{ } p) \text{ } \text{mon}$

**proof** (*cases keys mon  $\subseteq \{i\}$* )

**case** *False*

**hence** [simp]:  $\text{mon} \neq 0$

**by** *auto*

**obtain**  $j$  **where**  $j \in \text{keys } \text{mon}$   $j \neq i$

**using** *False* **by** *auto*

**have**  $\text{coeff } (\text{mpoly-of-poly } i \text{ } p) \text{ } \text{mon}' = 0$  **if** *mon-eq: mon = moni + mon'* **for**  $\text{mon}'$

**proof** –

**have**  $\text{Poly-Mapping.lookup } \text{mon } j \neq 0$

**using**  $j$  **by** (*meson lookup-eq-zero-in-keys-contradict*)

**also have**  $\text{Poly-Mapping.lookup } \text{mon } j = \text{Poly-Mapping.lookup } \text{mon}' \text{ } j$

**unfolding** *mon-eq moni-def* **using**  $j$  **by** (*simp add: lookup-add lookup-single*)

**finally have**  $j \in \text{keys } \text{mon}'$

**by** (*meson lookup-not-eq-zero-eq-in-keys*)

**with**  $j$  **have**  $\neg \text{keys } \text{mon}' \subseteq \{i\}$

**by** *blast*

**thus** *?thesis* **by** *simp*

**qed**

**hence**  $\text{coeff } (\text{Var } i * \text{mpoly-of-poly } i \text{ } p) \text{ } \text{mon} = 0$

**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**  $n = \text{Poly-Mapping.lookup } \text{mon } i$

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

```

```

lemma mpoly-of-poly-1 [simp]: mpoly-of-poly i 1 = 1
  unfolding one-pCons mpoly-of-poly-pCons mpoly-of-poly-0 by simp

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma mapping-of-mpoly-of-poly [simp]: mapping-of (mpoly-of-poly i p) = mpoly-of-poly-aux
i p

```



by *transfer' simp*

**lemma** *vars-mpoly-of-poly*:  $\text{vars } (\text{mpoly-of-poly } i \ p) \subseteq \{i\}$   
**proof** –  
 have  $x = i$  if  $xa \in \text{keys } (\text{mpoly-of-poly-aux } i \ p)$   $x \in \text{keys } xa$  **for**  $x \ xa$   
 using *that*  
 by (*meson in-mono lookup-eq-zero-in-keys-contradict mpoly-of-poly-aux.rep-eq singletonD*)  
 thus *?thesis*  
 by (*auto simp: vars-def*)  
**qed**

**lemma** *mpoly-map-vars-mpoly-of-poly* [*simp*]:  
 assumes *bij f*  
 shows  $\text{mpoly-map-vars } f \ (\text{mpoly-of-poly } i \ p) = \text{mpoly-of-poly } (f \ i) \ p$   
**proof** (*rule mpoly-eqI, goal-cases*)  
 case (*1 mon*)  
 have  $f \text{ -- 'keys } mon \subseteq \{i\} \longleftrightarrow \text{keys } mon \subseteq \{f \ i\}$   
 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**

## 5 More facts about algebraic numbers

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

### 5.1 Miscellaneous

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

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

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

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

**lemma** *algebraic-sum* [*intro*]:  
 $(\bigwedge x. x \in A \implies \text{algebraic } (f \ x)) \implies \text{algebraic } (\text{sum } f \ A)$

by (induction A rule: infinite-finite-induct) (auto intro!: algebraic-plus)

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

**lemma algebraic-sum-list** [intro]:  
 $(\bigwedge x. x \in \text{set } xs \implies \text{algebraic } x) \implies \text{algebraic } (\text{sum-list } xs)$   
 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 algebraic (csqr x)  
 hence algebraic (csqr x  $\wedge$  2)  
 by (rule algebraic-power)  
 also have csqr x  $\wedge$  2 = x  
 by simp  
 finally show 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]: algebraic i  
 by (intro algebraic-int-imp-algebraic) auto

**lemma algebraic-int-fact** [simp, intro]: algebraic-int (fact n)

by (intro int-imp-algebraic-int fact-in-Ints)

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

**lemma** algebraic-add-cancel-left [simp]:  
 assumes algebraic  $x$   
 shows algebraic  $(x + y) \longleftrightarrow$  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  $\neq$  0*  
 shows *algebraic (x / y)  $\longleftrightarrow$  algebraic y*  
**proof** –  
 have *algebraic (x \* inverse y)  $\longleftrightarrow$  algebraic (inverse y)*  
 by (*intro algebraic-mult-cancel-left assms*)  
 also have *...  $\longleftrightarrow$  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  $\neq$  0*  
 shows *algebraic (x / y)  $\longleftrightarrow$  algebraic x*  
**proof** –  
 have *algebraic (x \* inverse y)  $\longleftrightarrow$  algebraic x*  
 using *assms* **by** (*intro algebraic-mult-cancel-right auto*)  
 thus *?thesis* **by** (*simp add: field-simps*)  
**qed**

## 5.2 Turning an algebraic number into an algebraic integer

### 5.3 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  $\neq$  0*  
 defines *c  $\equiv$  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 ( $\lambda i.$  if  $i = n$  then 1 else  $c \wedge (n - i - 1) * poly.coeff p i$ )*  
 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  $c \wedge (n - i - 1) * poly.coeff p i$* )  
 (*is - = ?f i*) **for** *i* **unfolding** *p'-def*  
**proof** (*subst poly.Abs-poly-inverse*)  
 have *eventually ( $\lambda i.$  poly.coeff p i = 0) cofinite*

**using** *MOST-coeff-eq-0* **by** *blast*  
**hence** *eventually*  $(\lambda i. ?f\ i = 0)$  *cofinite*  
**by** *eventually-elim* (*use* *assms* **in**  $\langle \text{auto simp: } n\text{-def} \rangle$ )  
**thus**  $?f \in \{f. \text{eventually } (\lambda i. f\ i = 0) \text{ cofinite}\}$  **by** *simp*  
**qed** *auto*

**have** *deg-p'*: *Polynomial.degree*  $p' = n$   
**proof** –  
**from** *assms* **have**  $(\lambda n. \forall i > n. \text{poly.coeff } p'\ i = 0) = (\lambda n. \forall i > n. \text{poly.coeff } p\ i = 0)$   
**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**  $0 = \text{of-int } (c \wedge (n - 1)) * (\sum i \leq n. \text{of-int } (\text{poly.coeff } p\ i) * x \wedge i)$   
**using** *assms unfolding n-def poly-altdef* **by** *simp*  
**also** **have**  $\dots = (\sum i \leq n. \text{of-int } (c \wedge (n - 1)) * \text{poly.coeff } p\ i) * x \wedge i$   
**by** (*simp add: sum-distrib-left sum-distrib-right mult-ac*)  
**also** **have**  $\dots = (\sum i \leq n. \text{of-int } (\text{poly.coeff } p'\ i) * (\text{of-int } c * x) \wedge i)$   
**proof** (*intro sum.cong, goal-cases*)  
**case**  $(2\ i)$   
**have**  $\text{of-int } (\text{poly.coeff } p'\ i) * (\text{of-int } c * x) \wedge i =$   
 $\text{of-int } (c \wedge i * \text{poly.coeff } p'\ i) * x \wedge i$   
**by** (*simp add: algebra-simps*)  
**also** **have**  $c \wedge i * \text{poly.coeff } p'\ i = c \wedge (n - 1) * \text{poly.coeff } p\ i$   
**proof** (*cases i = n*)  
**case** *True*  
**hence**  $c \wedge i * \text{poly.coeff } p'\ i = c \wedge n$   
**by** (*auto simp: coeff-p' simp flip: power-Suc*)  
**also** **have**  $n = \text{Suc } (n - 1)$   
**using**  $\langle n > 0 \rangle$  **by** *simp*  
**also** **have**  $c \wedge \dots = c * c \wedge (n - 1)$   
**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**  $\dots = \text{ipoly } p'\ (\text{of-int } c * x)$   
**by** (*simp add: poly-altdef n-def deg-p'*)  
**finally** **have**  $\text{ipoly } p'\ (\text{of-int } c * x) = 0 \dots$

```

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

end

```

## 6 Miscellaneous facts

```

theory Misc-HLW
imports
  Complex-Main
  HOL-Library.Multiset
  HOL-Library.FuncSet
  HOL-Library.Groups-Big-Fun
  HOL-Library.Poly-Mapping
  HOL-Library.Landau-Symbols
  HOL-Combinatorics.Permutations
  HOL-Computational-Algebra.Computational-Algebra
begin

```

```

lemma set-mset-subset-singletonD:
  assumes set-mset A ⊆ {x}
  shows A = replicate-mset (size A) x
  using assms by (induction A) auto

```

```

lemma image-mset-eq-replicate-msetD:
  assumes image-mset f A = replicate-mset n y
  shows  $\forall x \in \#A. f x = y$ 
proof –
  have f ' set-mset A = set-mset (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**  $\pi$  permutes  $A$

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

**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**  $\pi$  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**  $\pi$ :  $\pi$  permutes  $B$  **for**  $\pi$

**by** (auto simp: permutes-in-image[OF  $\pi$ ])

**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**  $\pi$  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**  $\pi$ :  $\pi$  permutes  $B$  **for**  $\pi$

**by** (auto simp: permutes-in-image[OF  $\pi$ ])

**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**  $\text{bij-betw } f A B$

**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** 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 ∈ {X. set-mset X ⊆ A ∧ size X = n}
obtain xs where [simp]: X = mset xs
  by (metis ex-mset)
thus X ∈ mset ‘ {xs. set xs ⊆ A ∧ length xs = n}
  using X by auto
qed
next
show finite (mset ‘ {xs. set xs ⊆ A ∧ length xs = n})
  by (intro finite-imageI finite-lists-length-eq assms)
qed

lemma sum-mset-image-mset-sum-mset-image-mset:
  sum-mset (image-mset g (sum-mset (image-mset f A))) =
  sum-mset (image-mset (λx. sum-mset (image-mset g (f x))) A)
  by (induction A) auto

lemma sum-mset-image-mset-singleton: sum-mset (image-mset (λx. {#f x#}) A)
= image-mset f A
  by (induction A) auto

lemma sum-mset-conv-sum:
  sum-mset (image-mset f A) = (∑ x∈set-mset A. of-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 (λy. 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 ∉# A' count A' x = 0
      by (auto simp: A'-def)
    have n ≠ 0
      using x by (auto simp: n-def)

  have sum-mset (image-mset f A) = of-nat n * f x + sum-mset (image-mset f
A')
    by (simp add: A-eq)
  also have A' ⊂# A
    unfolding A'-def using x by (simp add: filter-mset-eq-conv subset-mset-def)
  with less.IH have sum-mset (image-mset f A') = (∑ x∈set-mset A'. of-nat
(count A' x) * f x)
    by simp
  also have ... = (∑ x∈set-mset A'. of-nat (count A x) * f x)
    by (intro sum.cong) (auto simp: A-eq)
  also have of-nat n * f x + ... = (∑ x∈insert x (set-mset A'). of-nat (count A

```



$x) * f x)$   
**by** (*subst sum.insert*) (*auto simp: A-eq*)  
**also from**  $\langle n \neq 0 \rangle$  **have**  $\text{insert } x \text{ (set-mset } A^\wedge) = \text{set-mset } A$   
**by** (*auto simp: A-eq*)  
**finally show** *?thesis* .  
**qed** *auto*  
**qed**

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

*sum-mset (image-mset f A) = Sum-any ( $\lambda x. \text{of-nat (count A } x) * f x$ )*

**proof** –

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

**by** (*rule sum-mset-conv-sum*)

**also have**  $\dots = \text{Sum-any } (\lambda x. \text{of-nat (count A } x) * f x)$

**proof** (*rule Sum-any.expand-superset [symmetric]*)

**show**  $\{x. \text{of-nat (count A } x) * f x \neq 0\} \subseteq \text{set-mset } A$

**proof**

**fix**  $x$  **assume**  $x \in \{x. \text{of-nat (count A } x) * f x \neq 0\}$

**hence**  $\text{count A } x \neq 0$

**by** (*intro notI*) *auto*

**thus**  $x \in \# A$

**by** *auto*

**qed**

**qed** *auto*

**finally show** *?thesis* .

**qed**

**lemma** *Sum-any-sum-swap*:

**assumes** *finite A*  $\wedge$  *finite*  $\{x. f x y \neq 0\}$

**shows**  $\text{Sum-any } (\lambda x. \text{sum (f } x) A) = (\sum_{y \in A} \text{Sum-any } (\lambda x. f x y))$

**proof** –

**have**  $\text{Sum-any } (\lambda x. \text{sum (f } x) A) = \text{Sum-any } (\lambda x. \text{Sum-any } (\lambda y. f x y \text{ when } y \in A))$

**unfolding** *when-def* **by** (*subst Sum-any.conditionalize*) (*use assms in simp-all*)  
**also have**  $\dots = \text{Sum-any } (\lambda y. \text{Sum-any } (\lambda x. f x y \text{ when } y \in A))$

**by** (*intro Sum-any.swap[of ( $\bigcup_{y \in A} \{x. f x y \neq 0\}) \times A]$*  *finite-SigmaI finite-UN-I* *assms*) *auto*

**also have**  $(\lambda y. \text{Sum-any } (\lambda x. f x y \text{ when } y \in A)) = (\lambda y. \text{Sum-any } (\lambda x. f x y) \text{ when } y \in A)$

**by** (*auto simp: when-def*)

**also have**  $\text{Sum-any } \dots = (\sum_{y \in A} \text{Sum-any } (\lambda x. f x y))$

**unfolding** *when-def* **by** (*subst Sum-any.conditionalize*) (*use assms in simp-all*)

**finally show** *?thesis* .

**qed**

**lemma** (*in landau-pair*) *big-power*:

**assumes**  $f \in L F g$

**shows**  $(\lambda x. f x \hat{=} n) \in L F (\lambda x. g x \hat{=} n)$

**using** *big-prod[of  $\{..<n\}$   $\lambda-. f F \lambda-. g]$  *assms* **by** *simp**

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

**lemma** *pairwise-imp-disjoint-family-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' ⊆ A› in auto)
  also have  $\dots = B - B'$ 
    using assms(1,2) by (auto simp: bij-betw-def)
  finally show  $f \text{ ` } (A - A') = B - B'$  .
qed

```

```

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

```

**end**

## 7 The Hermite–Lindemann–WeierstraSS Transcendence Theorem

```

theory Hermite-Lindemann

```

```

imports

```

```

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

```

```

begin

```

```

hide-const (open) Henstock-Kurzweil-Integration.content Module.smult

```

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.

## 7.1 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  $\beta_1, \dots, \beta_m$  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  $\beta_i$  are non-zero integers (as opposed to arbitrary algebraic numbers)
2. The exponents  $\alpha_i$  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  $\pi$ , we define some number  $J$  depending on the  $\alpha_i$  and  $\beta_i$  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**  $\text{Roots}'$  **where**  $\text{Roots}' = (\bigcup p \in P. \text{Roots } p)$

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

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

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

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

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

**using** *nonconstant* $[\text{of } 0]$  **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*: *rsquarefree* (of-int-poly  $q$  :: complex poly) **if**  $q \in P$  **for**  $q$   
**by** (rule irreducible-imp-rsquarefree-of-int-poly) (use that **in** ⟨auto intro: irred  
nonconstant⟩)

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**  $P$ :  $p \in P$   $q \in P$  **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  $p \cap$  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  $\Rightarrow$  nat (#-)  
**where** *n-roots* = ( $\lambda p$ . card (Roots  $p$ ))  
**define**  $n$  **where**  $n = (\sum p \in P. \#p)$   
**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  $p \neq \{\}$  **if**  $p \in P$  **for**  $p$   
**using** *nonconstant*[OF that] **by** (auto simp: Roots-def fundamental-theorem-of-algebra  
constant-degree)  
**have** Roots'  $\neq \{\}$   
**using** *Roots-nonempty* ⟨ $P \neq \{\}$ ⟩ **by** (auto simp: Roots'-def)  
**have**  $n > 0$   
**using** ⟨Roots'  $\neq \{\}$ ⟩ ⟨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  $q$ ) ( $\prod x \in$  Roots  $q$ . [ $-x$ ,  
1:])  
**if**  $q \in P$  **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  $\alpha$ :

**define**  $l$  **where**  $l =$  (LCM  $q \in P$ . Polynomial.lead-coeff  $q$ )  
**have** *alg-int*: algebraic-int (of-int  $l * x$ ) **if**  $x \in$  Roots' **for**  $x$   
**proof** –  
**from** that **obtain**  $q$  **where**  $q: q \in P$  *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 = (∏ q∈P. l ^ Polynomial.degree q div Polynomial.lead-coeff q)
have lc-factor: Polynomial.smult (of-int l ^ n) (∏ α'∈Roots'. [:-α', 1:]) =
  of-int-poly (Polynomial.smult lc-factor (∏ P))
proof -
  define lc where lc = (λq. Polynomial.lead-coeff q :: int)
  define d where d = (Polynomial.degree :: int poly ⇒ nat)
  have (∏ q∈P. of-int-poly q) =
    (∏ q∈P. Polynomial.smult (lc q) (∏ x∈Roots q. [:-x, 1:]) :: complex poly)
  unfolding lc-def by (intro prod.cong of-int-poly-P refl)
  also have ... = Polynomial.smult (∏ q∈P. lc q) (∏ q∈P. (∏ x∈Roots q. [:-x,
1:]))
  by (simp add: prod-smult)
  also have (∏ q∈P. (∏ x∈Roots q. [:-x, 1:])) = (∏ x∈Roots'. [:-x, 1:])
  unfolding Roots'-def using disjoint
  by (intro prod.UNION-disjoint [symmetric]) (auto simp: disjoint-family-on-def)
  also have Polynomial.smult (of-int lc-factor) (Polynomial.smult (∏ q∈P. lc q)
...) =
    Polynomial.smult (∏ q∈P. of-int (l ^ d q div lc q * lc q)) (∏ x∈Roots'.
pCons (- x) 1)
  by (simp add: lc-factor-def prod.distrib lc-def d-def)
  also have (∏ q∈P. of-int (l ^ d q div lc q * lc q)) = (∏ q∈P. of-int l ^ d q ::
complex)
proof (intro prod.cong, goal-cases)
  case (2 q)
  have lc q dvd l
  unfolding l-def lc-def using 2 by auto
  also have ... dvd l ^ d q
  using 2 nonconstant[of q] by (intro dvd-power) (auto simp: d-def)
  finally show ?case by simp
qed auto
also have ... = l ^ (∑ q∈P. d q)
  by (simp add: power-sum)
also have (∑ q∈P. d q) = (∑ q∈P. n-roots q)
proof (intro sum.cong, goal-cases)
  case (2 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 ... =  $n$ 
  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  $R :: real$  where  $R = Max (norm \text{' } Roots')$ 
have  $R \geq norm \alpha$  if  $\alpha \in Roots'$  for  $\alpha$ 
  unfolding  $R\text{-def}$  using that by (intro Max-ge) auto
have  $R \geq 0$ 
proof -
  from  $\langle Roots' \neq \{\} \rangle$  obtain  $\alpha$  where  $\alpha \in Roots'$ 
  by auto
  have  $0 \leq norm \alpha$ 
  by simp
  also have ...  $\leq R$ 
  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  $C :: nat \Rightarrow real$  where  $C = (\lambda p. l \wedge (n * p) * (2 * R) \wedge (n * p - 1))$ 
define  $C'$  where
   $C' = (\prod_{x \in Roots'} \sum_{q \in P. real\text{-of-int } |\beta \ q| * (\sum_{\alpha \in Roots} q. cmod \alpha * exp (cmod \alpha))$ 

```

We commence with the proof of the main inequality.

```

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

```

define  $f\text{-poly} :: complex \Rightarrow complex \text{ poly}$  where

```

```

    f-poly = (λα. Polynomial.smult (l^(n*p)) ((∏ α'∈Roots'. [-α', 1:]^p) div
[-α, 1:]))
  have f-poly-altdef: f-poly α = Polynomial.smult (l^(n*p))
    ((∏ α'∈Roots'. [-α', 1:]^(if α' = α then p - 1 else p)))
  if α ∈ Roots' for α
  proof -
  have (∏ α'∈Roots'. [-α', 1:]^(if α'=α then p-1 else p)) * [-α, 1:] =
    [-α, 1:]^(p-1) * (∏ x∈Roots' - {α}. [-x, 1:]^p) * [-α, 1:]
    using that by (subst prod.If-eq) (auto simp: algebra-simps)
  also have ... = (∏ x∈Roots' - {α}. [-x, 1:]^p) * [-α, 1:]^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**  $J' :: \text{complex}$   
**where**  $J' = (\prod \alpha \in \text{Roots}' . J \alpha)$

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

**have**  $J\text{-eq}$ :  $J \alpha = -(\sum q \in P . \text{of-int } (\beta q) * (\sum x \in \text{Roots } q . \sum j < n * p . \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) x))$   
**if**  $\alpha \in \text{Roots}'$  **for**  $\alpha$   
**proof** –  
**have**  $n * p \geq 1 * 2$   
**using**  $\langle n > 0 \rangle \langle p > 1 \rangle$  **by**  $(\text{intro mult-mono}) \text{ auto}$   
**hence**  $[\text{simp}]$ :  $\{..n*p\text{-Suc } 0\} = \{..<n*p\}$   
**by**  $\text{auto}$   
**have**  $J \alpha = (\sum q \in P . \beta q * (\sum x \in \text{Roots } q . I \alpha x))$   
**unfolding**  $J\text{-def } ..$   
**also have**  $... = (\sum q \in P . \text{of-int } (\beta q) * (\sum x \in \text{Roots } q . \text{exp } x * (\sum j < n * p . \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) 0))) -$   
 $(\sum q \in P . \text{of-int } (\beta q) * (\sum x \in \text{Roots } q . \sum j < n * p . \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) x))$   
**unfolding**  $I\text{-def lindemann-weierstrass-aux.I-def}$   
**by**  $(\text{simp add: deg-f that ring-distrib sum-subtractf sum-distrib-left sum-distrib-right mult-ac})$   
**also have**  $... = -(\sum q \in P . \text{of-int } (\beta q) * (\sum x \in \text{Roots } q . \sum j < n * p . \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) x))$   
**unfolding**  $\text{sum-distrib-right } [\text{symmetric}] \text{mult.assoc } [\text{symmetric}] \text{sum-eq-0}$   
**by**  $\text{simp}$   
**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**  $J$ :  $\text{fact } (p - 1) \text{ alg-dvd } J \alpha \neg \text{of-nat } p \text{ alg-dvd } J \alpha$  **if**  $\alpha: \alpha \in \text{Roots}'$  **for**  $\alpha$   
**proof** –  
**define**  $h$  **where**  $h = (\lambda \alpha' j . \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) \alpha')$   
**from**  $\alpha$  **obtain**  $q$  **where**  $q: q \in P \alpha \in \text{Roots } q$   
**by**  $(\text{auto simp: Roots'-def})$   
  
**have**  $J \alpha = -(\sum (q, \alpha') \in \text{Sigma } P \text{ Roots} . \sum j < n * p . \text{of-int } (\beta q) * h \alpha' j)$   
**unfolding**  $J\text{-eq}[OF \alpha] h\text{-def sum-distrib-left}$  **by**  $(\text{subst } (2) \text{sum.Sigma}) \text{ auto}$   
**also have**  $... = -(\sum ((q, \alpha'), i) \in \text{Sigma } P \text{ Roots} \times \{..<n*p\} . \text{of-int } (\beta q) * h \alpha' i)$   
**by**  $(\text{subst } (2) \text{sum.Sigma } [\text{symmetric}]) (\text{auto simp: case-prod-unfold})$   
**finally have**  $J\text{-eq}'$ :  $J \alpha = -(\sum ((q, \alpha'), i) \in \text{Sigma } P \text{ Roots} \times \{..<n * p\} . \text{of-int } (\beta q) * h \alpha' i) .$

**have**  $h\text{-}\alpha\text{-pm1-eq}$ :  $h\ \alpha\ (p-1) = \text{of-int}\ (l^{\wedge}(n*p)) * \text{fact}\ (p-1) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. (\alpha - \alpha')^{\wedge} p)$   
**proof** –  
**have**  $h\ \alpha\ (p-1) = \text{of-int}\ (l^{\wedge}(n * p)) * \text{poly}\ ((\text{pderiv}\ \widehat{\wedge}(p-1)) (\prod \alpha' \in \text{Roots}'. [:-\alpha', 1:]^{\wedge} (\text{if}\ \alpha' = \alpha\ \text{then}\ p - 1\ \text{else}\ p)))\ \alpha$   
**unfolding**  $h\text{-def}\ f\text{-poly}\ \text{altdef}[OF\ \alpha]\ \text{higher-pderiv-smult}\ \text{poly-smult}\ ..$   
**also have**  $(\prod \alpha' \in \text{Roots}'. [:-\alpha', 1:]^{\wedge} (\text{if}\ \alpha' = \alpha\ \text{then}\ p - 1\ \text{else}\ p)) = [:-\alpha, 1:]^{\wedge}(p-1) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. [:-\alpha', 1:]^{\wedge} p)$   
**using**  $\alpha$  **by**  $(\text{subst}\ \text{prod.If-eq})\ \text{auto}$   
**also have**  $\text{poly}\ ((\text{pderiv}\ \widehat{\wedge}(p-1))\ \dots)\ \alpha = \text{fact}\ (p-1) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. (\alpha - \alpha')^{\wedge} p)$   
**by**  $(\text{subst}\ \text{poly-higher-pderiv-aux2})\ (\text{simp-all}\ \text{add:}\ \text{poly-prod})$   
**finally show**  $?thesis$  **by**  $(\text{simp}\ \text{only:}\ \text{mult.assoc})$   
**qed**

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

**have**  $\neg\text{of-nat}\ p\ \text{alg-dvd}\ \text{of-int}\ (\beta\ q) * h\ \alpha\ (p-1)$   
**unfolding**  $h\text{-}\alpha\text{-pm1-eq}\ \text{mult.assoc}\ [\text{symmetric}]\ \text{of-int-mult}\ [\text{symmetric}]$   
**proof**  
**define**  $r$  **where**  $r = (\lambda\alpha. \text{of-int}\ (l^{\wedge} n) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. \alpha - \alpha'))$   
**have**  $\text{alg-int-r}$ :  $\text{algebraic-int}\ (r\ \alpha)$  **if**  $\alpha \in \text{Roots}'$  **for**  $\alpha$   
**proof** –  
**have**  $\text{algebraic-int}\ (\text{of-int}\ l * (\prod \alpha' \in \text{Roots}' - \{\alpha\}. \text{of-int}\ l * \alpha - \text{of-int}\ l * \alpha'))$   
**by**  $(\text{intro}\ \text{algebraic-int-times}[OF\ \text{algebraic-int-of-int}]\ \text{algebraic-int-prod})$

*algebraic-int-power algebraic-int-diff alg-int that* auto  
**also have**  $\dots = \text{of-int } l * (\prod \alpha' \in \text{Roots}' - \{\alpha\}, \text{of-int } l * (\alpha - \alpha'))$   
**by** (*simp add: algebra-simps flip: power-mult-distrib*)  
**also have**  $\dots = \text{of-int } (l \wedge (1 + (n - 1))) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}, \alpha - \alpha')$   
**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**  $\langle n > 0 \rangle$  **by** auto  
**finally show** *algebraic-int* ( $r \alpha$ )  
**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*) =  
 $\text{Const } (\text{of-int } l \wedge n) * (\prod j \in \{..<n\} - \{i\}. \text{Var } (\pi \ i) - \text{Var } (\pi \ j))$   
**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*  $\mathbf{Q} :: \text{complex set} .$   
**show**  $\forall m. \text{MPoly-Type.coeff } R \ m \in \mathbf{Q}$   
 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**  $?lc = \text{of-int } (\prod p \in P. \text{Polynomial.lead-coeff } p) :: \text{complex}$   
**have**  $(\prod q \in P. \text{of-int-poly } q) = (\prod q \in P. \text{Polynomial.smult}$   
 $(\text{of-int } (\text{Polynomial.lead-coeff } q)) (\prod x \in \text{Roots } q. [-x, 1:]))$   
 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*  $?lc \in \mathbf{Q}$   
 by (*auto simp: field-simps simp flip: of-int-prod*)  
**qed auto**  
**also have** *insertion*  $\text{Root } R = (\prod i < n. \text{of-int } (l \hat{=} n) * (\prod j \in \{..<n\} - \{i\}.$   
 $\text{Root } i - \text{Root } j))$   
 by (*simp add: R-def insertion-prod insertion-mult insertion-power*  
*insertion-diff*)  
**also have**  $\dots = (\prod i < n. \text{of-int } (l \hat{=} n) * (\prod \alpha' \in \text{Roots}' - \{\text{Root } i\}.$   
 $\text{Root } i - \alpha'))$   
**proof** (*intro prod.cong, goal-cases*)  
**case** (2 *i*)  
**hence**  $(\prod j \in \{..<n\} - \{i\}.$   
 $\text{Root } i - \text{Root } j) = (\prod \alpha' \in \text{Roots}' - \{\text{Root } i\}.$   
 $\text{Root } i - \alpha')$   
 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**  $(\prod \alpha' \in \text{Roots}' . r \ \alpha') \in \mathbf{Q} .$   
**qed**  
**moreover have** *algebraic-int*  $(\prod \alpha' \in \text{Roots}' . r \ \alpha')$   
 by (*intro algebraic-int-prod alg-int-r*)  
**ultimately have** *is-int*:  $(\prod \alpha' \in \text{Roots}' . r \ \alpha') \in \mathbf{Z}$   
 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  $\langle l > 0 \rangle$  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  $(\beta q * l^{(n*p)}) * \text{fact } (p-1) * (\prod \alpha' \in \text{Roots}' - \{\alpha\}.$   
 $(\alpha - \alpha')^{\wedge p}$   
 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  $\dots = \text{of-int } (\beta q) * \text{fact } (p-1) * (\prod \alpha' \in \text{Roots}'. r \alpha')^{\wedge p}$   
 using  $\alpha$  by (subst (2) prod.remove[of -  $\alpha$ ]) (auto simp: mult-ac power-mult-distrib)  
 also have  $\dots = \text{of-int } (\beta q * \text{fact } (p-1) * R')^{\wedge p}$   
 by (simp add: R')  
 also have of-nat p = of-int (int p)  
 by simp  
 finally have int p dvd  $\beta q * \text{fact } (p-1) * R'^{\wedge p}$   
 by (subst (asm) alg-dvd-of-int-iff)  
 moreover have prime (int p)  
 using ⟨prime p⟩ by auto  
 ultimately have int p dvd  $\beta q \vee \text{int } p \text{ dvd } \text{fact } (p-1) \vee \text{int } p \text{ dvd } R'^{\wedge p}$   
 by (simp add: prime-dvd-mult-iff)  
 moreover have  $\neg \text{int } p \text{ dvd } \beta q$   
 proof  
 assume int p dvd  $\beta q$   
 hence int p  $\leq |\beta q|$   
 using  $\beta\text{-nz}[of q]$  dvd-imp-le-int[of  $\beta q$  int p] q by auto  
 with  $p\text{-ineqs}(1)$  q show False by auto  
 qed  
 moreover have  $\neg \text{int } p \text{ dvd } \text{fact } (p-1)$   
 proof -  
 have  $\neg p \text{ dvd } \text{fact } (p-1)$   
 using ⟨p > 1⟩ p by (subst prime-dvd-fact-iff) auto  
 hence  $\neg \text{int } p \text{ dvd } \text{int } (\text{fact } (p-1))$   
 by (subst int-dvd-int-iff)  
 thus ?thesis unfolding of-nat-fact .  
 qed  
 moreover have  $\neg \text{int } p \text{ dvd } R'^{\wedge p}$   
 proof  
 assume int p dvd  $R'^{\wedge p}$   
 hence int p dvd  $R'$   
 using ⟨prime (int p)⟩ prime-dvd-power by metis  
 hence int p  $\leq |R'|$   
 using  $\beta\text{-nz}[of q]$  dvd-imp-le-int[of  $R'$  int p] q by auto  
 hence real p  $\leq \text{real-of-int } |R'|$

```

    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  $\alpha' \in \text{Roots}' \alpha' \neq \alpha \vee j \neq p - 1$  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  $Q' = \text{Polynomial.smult (of-int (l^{n*p+j})) (pcompose Q$ 
 $[:0, 1 / \text{of-int } l:])$ 
      have  $\text{poly} ((\text{pderiv } \wedge j) Q) \alpha' / l \wedge j =$ 
 $\text{poly} ((\text{pderiv } \wedge j) (pcompose Q [:0, 1 / \text{of-int } l:]))) (l * \alpha')$ 
      using  $\langle l > 0 \rangle$  by (simp add: higher-pderiv-pcompose-linear poly-pcompose
field-simps)

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

    have  $h \alpha' j = \text{of-int (l^{n*p})} * \text{poly} ((\text{pderiv } \wedge j) Q) \alpha'$ 
      unfolding h-def f-poly-altdef[OF  $\alpha$ ] higher-pderiv-smult poly-smult e-def
Q-def ..
    also have  $\text{poly} ((\text{pderiv } \wedge j) Q) \alpha' =$ 
 $\text{of-int } l \wedge j * \text{poly} ((\text{pderiv } \wedge j) (pcompose Q [:0, 1 / \text{of-int } l:])))$ 
 $(l * \alpha')$ 
      using  $\langle l > 0 \rangle$  by (simp add: higher-pderiv-pcompose-linear poly-pcompose
field-simps)
    also have  $\text{of-int (l} \wedge (n * p)) * \dots = \text{poly} ((\text{pderiv } \wedge j) Q') (l * \alpha')$ 
      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 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 ... * poly.coeff  $Q i =$ 
     $(\sum X \in \{X. X \subseteq (\text{SIGMA } x:\text{Roots}'. \{..<e x\}) \wedge i = n * p - \text{Suc} (\text{card } X)\}$ 
     $\text{of-int } l^{(n * p + j - (n * p - \text{Suc} (\text{card } X)))} * ((- 1) ^ \text{card } X * \text{prod fst } X))$ 
    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 ...  $< n * p$ 
      using  $\langle n * p > 0 \rangle$  by simp

```

```

      finally have card-less: card X < n * p .
      have algebraic-int ((-1) ^ card X * of-int l ^ (j + 1) * (∏ x∈X. of-int
l * 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-dvd: of-nat p alg-dvd h α' j if α' ∈ Roots' α' ≠ α ∨ j ≠ p - 1 for α'
j
proof -
  have of-nat p alg-dvd (of-nat (fact p) :: complex)
    by (intro alg-dvd-of-nat dvd-fact) (use ⟨p > 1⟩ in auto)
  hence of-nat p alg-dvd (fact p :: complex)
    by simp
  also have ... alg-dvd h α' j
    using that by (intro fact-p-dvd)
  finally show ?thesis .
qed

show fact (p - 1) alg-dvd J α
  unfolding J-eq'
proof (intro alg-dvd-uminus-right alg-dvd-sum, safe intro!: alg-dvd-mult alge-
braic-int-of-int)
  fix q α' j
  assume q ∈ P α' ∈ Roots q j < n * p
  hence α' ∈ Roots'
    by (auto simp: Roots'-def)
  show fact (p - 1) alg-dvd h α' j
  proof (cases α' = α ∧ j = p - 1)
    case True
      thus ?thesis using ⟨fact (p - 1) alg-dvd h α (p - 1)⟩
        by simp
    next
      case False
        have of-int (fact (p - 1)) alg-dvd (of-int (fact p) :: complex)
          by (intro alg-dvd-of-int fact-dvd) auto
        hence fact (p - 1) alg-dvd (fact p :: complex)
          by simp
        also have ... alg-dvd h α' j
          using False ⟨α' ∈ Roots'⟩ by (intro fact-p-dvd) auto
        finally show ?thesis .
      qed
    qed
  qed

```



```

show  $\neg$ of-nat  $p$  alg-dvd  $J$   $\alpha$ 
  unfolding  $J$ -eq' alg-dvd-uminus-right-iff
proof (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 (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  $\alpha \in \text{Roots } (fst (fst z)) \cap \text{Roots } q \Rightarrow q \in P \text{ fst } (fst z) \in P$ 
    using that  $q$  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  $(x, xa) \Rightarrow$  (case  $x$  of  $(q, \alpha') \Rightarrow \lambda i. \text{of-int}$ 
 $(\beta q) * h \alpha' i$ )  $xa$ )
  using  $z$  by auto
qed (use  $\langle \neg$ 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  $\alpha_i$  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 ... = of-int-poly Q
  by (subst lc-factor) (simp-all add: Q-def of-int-poly-hom.hom-power)
also have ... div [-α, 1:] = synthetic-div (of-int-poly Q) α
  unfolding synthetic-div-altdef ..
also have ... = synthetic-div (map-poly (λp. ipoly p α) (map-poly (λx. [:x:]
Q)) (ipoly [:0, 1:] α))
  by (simp add: map-poly-map-poly o-def)
also have ... = map-poly (λp. ipoly p α) g
  unfolding g-def Q-def by (rule α.synthetic-div-hom)
finally show ?thesis ..
qed

obtain Q where Q: J α = -(∑ q∈P. of-int (β q) * eval-poly of-rat (Q q) α)
if α ∈ Roots' for α
proof -
  define g' :: nat ⇒ complex poly poly
    where g' = (λj. (map-poly of-int-poly ((pderiv ~ j) g)))
obtain root :: int poly ⇒ nat ⇒ complex
  where root: ∧q. q ∈ P ⇒ bij-betw (root q) {..<#q} (Roots q)
using ex-bij-betw-nat-finite[OF finite-Roots] unfolding n-roots-def atLeast0LessThan
  by metis
  have ∃ Q'. map-poly of-rat Q' = (∑ x∈Roots q. poly (g' j) [:x:]) if q: q ∈ P
for q j
  proof -
    define Q :: nat ⇒ complex poly mpoly
      where Q = (λj. (∑ i<#q. mpoly-of-poly i (g' j)))
    define ratpolys :: complex poly set where ratpolys = {p. ∀ i. poly.coeff p i
∈ Q}
    have insertion ((λx. [:x:]) ∘ root q) (Q j) ∈ ratpolys
    proof (rule symmetric-poly-of-roots-in-subring)
      show ring-closed ratpolys
      by standard (auto simp: ratpolys-def intro!: coeff-mult-semiring-closed)
      show ∀ m. MPoly-Type.coeff (Q j) m ∈ ratpolys
      by (auto simp: Q-def ratpolys-def Polynomial.coeff-sum coeff-mpoly-of-poly
when-def g'-def
      intro!: sum-in-Rats)
    show vars (Q j) ⊆ {..<#q} unfolding Q-def
    by (intro order.trans[OF vars-sum] UN-least order.trans[OF vars-mpoly-of-poly])
  auto
    show symmetric-mpoly {..<#q} (Q j) unfolding Q-def
      by (rule symmetric-mpoly-symmetric-sum[of - id]) (auto simp: per-
mutes-bij)
    interpret coeff-lift-hom: map-poly-idom-hom λx. [:x:]
      by standard
    define lc :: complex where lc = of-int (Polynomial.lead-coeff q)
    have of-int-poly q = Polynomial.smult (Polynomial.lead-coeff q) (∏ x∈Roots
q. [-x, 1:])
      by (rule of-int-poly-P) fact
    also have poly-lift ... = Polynomial.smult [:lc:] (∏ a∈Roots q. [-:a:],

```

1:]

by (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 (intro prod.reindex-bij-betw [symmetric] root q)

also have  $\dots = (\prod_{i < \#q} [:-((\lambda x. [x:]) \circ \text{root } q) \ i, 1:])$

by simp

finally show poly-lift (Ring-Hom-Poly.of-int-poly q) = Polynomial.smult

[lc:] ...

have  $lc \neq 0$

using q by (auto simp: lc-def)

thus [inverse lc:] \* [lc:] = 1

by (simp add: field-simps)

qed (auto simp: ratpolys-def coeff-pCons split: nat.splits)

also have insertion  $((\lambda x. [x:]) \circ \text{root } q) (Q \ j) = (\sum_{i < \#q} \text{poly } (g' \ j) [:\text{root } q \ i:])$

by (simp add: Q-def insertion-sum poly-sum)

also have  $\dots = (\sum_{x \in \text{Roots } q} \text{poly } (g' \ j) [x:])$

by (intro sum.reindex-bij-betw root q)

finally have  $\forall i. \text{poly.coeff } (\sum_{x \in \text{Roots } q} \text{poly } (g' \ j) [x:]) \ i \in \mathbb{Q}$

by (auto simp: ratpolys-def)

thus ?thesis

using ratpolyE by metis

qed

then obtain Q where  $Q: \bigwedge q \ j. q \in P \implies \text{map-poly of-rat } (Q \ q \ j) =$

$(\sum_{x \in \text{Roots } q} \text{poly } (g' \ j) [x:])$

by metis

define Q' where  $Q' = (\lambda q. \sum_{j < n * p} Q \ q \ j)$

have  $J \ \alpha = - (\sum_{q \in P} \text{of-int } (\beta \ q) * \text{eval-poly of-rat } (Q' \ q) \ \alpha)$  if  $\alpha: \alpha \in$

Roots' for  $\alpha$

proof -

have  $J \ \alpha = - (\sum_{q \in P} \text{of-int } (\beta \ q) * (\sum_{x \in \text{Roots } q} \sum_{j < n * p} \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) \ x))$

(is - = - ?S) unfolding J-eq[OF  $\alpha$ ] ..

also have ?S =  $(\sum_{q \in P} \text{of-int } (\beta \ q) * \text{eval-poly of-rat } (Q' \ q) \ \alpha)$

proof (rule sum.cong, goal-cases)

case q: (2 q)

interpret  $\alpha: \text{idom-hom } \lambda p. \text{ipoly } p \ \alpha$

by standard (auto simp: of-int-hom.poly-map-poly-eval-poly of-int-poly-hom.hom-mult)

have  $(\sum_{x \in \text{Roots } q} \sum_{j < n * p} \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) \ x) =$

$(\sum_{j < n * p} \sum_{x \in \text{Roots } q} \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) \ x)$

by (rule sum.swap)

also have  $\dots = (\sum_{j < n * p} \text{eval-poly of-rat } (Q \ q \ j) \ \alpha)$

proof (rule sum.cong, goal-cases)

case j: (2 j)

have  $(\sum_{x \in \text{Roots } q} \text{poly } ((\text{pderiv } \sim j) (f\text{-poly } \alpha)) \ x) =$

$(\sum_{x \in \text{Roots } q} \text{poly } (\text{poly } (g' \ j) [x:]) \ \alpha)$

```

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

have J'  $\in$  Q
proof -
  have ( $\prod \alpha \in \text{Roots } q$ . J  $\alpha$ )  $\in$  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' = -( $\sum q \in P$ . Polynomial.smult (of-int ( $\beta$  q)) (map-poly of-rat
(Q q)))

    have ( $\prod \alpha \in \text{Roots } q$ . J  $\alpha$ ) = ( $\prod \alpha \in \text{Roots } q$ . -( $\sum q \in P$ . of-int ( $\beta$  q) * eval-poly
of-rat (Q q)  $\alpha$ ))
    by (intro prod.cong refl Q) (auto simp: Roots'-def q)
    also have ... = ( $\prod \alpha \in \text{Roots } q$ . poly Q'  $\alpha$ )

```

```

    by (simp add: Q'-def poly-sum eval-poly-def)
  also have ... = (∏ i<#q. poly Q' (root i))
    by (intro prod.reindex-bij-betw [symmetric] root)
  also have ... = insertion root (∏ i<#q. mpoly-of-poly i Q')
    by (simp add: insertion-prod)
  also have ... ∈ ℚ
  proof (rule symmetric-poly-of-roots-in-subring)
    show ring-closed (ℚ :: complex set)
      by standard auto
    then interpret Q: ring-closed ℚ :: complex set .
    show ∀ m. MPoly-Type.coeff (∏ i<#q. mpoly-of-poly i Q') m ∈ ℚ
      by (auto intro!: Q.coeff-prod-closed sum-in-Rats
        simp: coe-eff-mpoly-of-poly when-def Q'-def Polynomial.coeff-sum)
    show symmetric-mpoly {..<#q} (∏ i<#q. mpoly-of-poly i Q')
      by (intro symmetric-mpoly-symmetric-prod'[of - id]) (auto simp: per-
        mutes-bij)
    show vars (∏ i<#q. mpoly-of-poly i Q') ⊆ {..<#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 (∏ x∈Roots q. [:- x, 1:])
      unfolding lc-def by (rule of-int-poly-P) fact
    also have (∏ x∈Roots q. [:- x, 1:]) = (∏ i<#q. [:- root i, 1:])
      by (intro prod.reindex-bij-betw [symmetric] root)
    finally show of-int-poly q = Polynomial.smult lc (∏ i<#q. [:- root i, 1:])
      .
    have lc ≠ 0
      using q by (auto simp: lc-def)
    thus inverse lc * lc = 1 inverse lc ∈ ℚ
      by (auto simp: lc-def)
  qed auto
  finally show ?thesis .
  qed
  hence (∏ q∈P. ∏ α∈Roots q. J α) ∈ ℚ
    by (rule prod-in-Rats)
  also have (∏ q∈P. ∏ α∈Roots q. J α) = J'
    unfolding Roots'-def J'-def using disjoint
  by (intro prod.UNION-disjoint [symmetric]) (auto simp: disjoint-family-on-def)
  finally show J' ∈ 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 ∈ 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 fact  $(p - 1) \wedge n$  alg-dvd  $J'$ 
proof -
  have fact  $(p - 1) \wedge n = (\prod \alpha \in \text{Roots}'. \text{fact } (p - 1))$ 
    by (simp add: n-altdef)
  also have ... alg-dvd  $J'$ 
    unfolding  $J'$ -def by (intro prod-alg-dvd-prod  $J(1)$ )
  finally show ?thesis .
qed

```

```

have 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 of-int ( $\text{fact } (p - 1) \wedge n$ ) = ( $\text{fact } (p - 1) \wedge n :: \text{complex}$ )
    by simp
  also have ... alg-dvd  $J'$ 
    by fact
  also have  $J' = \text{of-int } J''$ 
    by fact
  finally have  $\text{fact } (p - 1) \wedge n$  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 real-of-int  $|J''| \geq \text{real-of-int } |\text{fact } (p - 1) \wedge n|$ 
    by linarith
  also have 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  $\text{norm } J' \leq C' * C p ^ n$   
**proof** –  
 have  $\text{norm } J' = (\prod x \in \text{Roots}'. \text{norm } (J x))$   
 unfolding  $J'\text{-def prod-norm [symmetric] ..}$   
 also have  $\dots \leq (\prod x \in \text{Roots}'. \sum q \in P. \text{real-of-int } |\beta q| * (\sum \alpha \in \text{Roots } q. \text{cmod } \alpha * \exp (\text{cmod } \alpha) * C p))$   
**proof** (intro prod-mono conjI)  
 fix x assume x: x ∈ 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  $J\text{-def}$   
**proof** (intro sum-norm-le)  
 fix q assume q ∈ P  
 show  $\text{norm } (\text{of-int } (\beta q) * \text{sum } (I x) (\text{Roots } q)) \leq \text{real-of-int } |\beta q| * (\sum \alpha \in \text{Roots } q. \text{norm } \alpha * \exp (\text{norm } \alpha) * C p)$   
 unfolding  $\text{norm-mult norm-of-int of-int-abs}$   
**proof** (intro mult-left-mono sum-norm-le)  
 fix α assume α ∈ Roots q  
 hence α: α ∈ Roots'  
 using ⟨q ∈ P⟩ by (auto simp: Roots'-def)  
 show  $\text{norm } (I x \alpha) \leq \text{norm } \alpha * \exp (\text{norm } \alpha) * C p$   
 unfolding  $I\text{-def}$   
**proof** (intro lindemann-weierstrass-aux.lindemann-weierstrass-integral-bound)  
 fix t assume t ∈ closed-segment 0 α  
 also have closed-segment 0 α ⊆ cball 0 R  
 using ⟨R ≥ 0⟩ R-ge[OF α] by (intro closed-segment-subset) auto  
 finally have  $\text{norm } t \leq R$  by simp  
  
 have  $\text{norm-diff-le: norm } (t - y) \leq 2 * R$  if y ∈ 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 ⟨norm t ≤ R⟩ R-ge] that)  
 finally show ?thesis by simp  
 qed  
  
 have  $\text{norm } (\text{poly } (f\text{-poly } x) t) = |\text{real-of-int } l| ^ (n * p) * (\prod y \in \text{Roots}'. \text{cmod } (t - y) ^ (\text{if } y = x \text{ then } p - 1 \text{ else } p))$   
 by (simp add: eval-f x f-def norm-mult norm-power flip: prod-norm)  
 also have  $\dots \leq |\text{real-of-int } l| ^ (n * p) * (\prod y \in \text{Roots}'. (2 * R) ^ (\text{if } y = x \text{ then } p - 1 \text{ else } p))$   
 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** *norm (poly (f-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  $\langle$ auto simp: C-def $\rangle$ )  
**qed** *auto*  
**qed**  
**qed** *auto*  
**also have**  $\dots = C' * C p^{\wedge n}$   
**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  $(p-1)^{\wedge n} \leq C' * C p^{\wedge n}$*  .  
**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  $(\lambda p. (\forall q \in P. \text{int } p > |\beta q|) \wedge \text{real } p > \text{norm } (\prod_{\alpha \in \text{Roots}'}. \text{of-int } (l^{\wedge n}) * (\prod_{\alpha' \in \text{Roots}' - \{\alpha\}}. (\alpha - \alpha'))))$*   
*sequentially*  
**by** (*intro eventually-ball-finite  $\langle$ finite P $\rangle$  ballI eventually-conj filterlim-real-sequentially eventually-compose-filterlim[OF eventually-gt-at-top] filterlim-int-sequentially*)  
**have** *frequently  $(\lambda p. \text{fact } (p-1)^{\wedge n} \leq C' * C p^{\wedge n})$  sequentially*  
**by** (*rule frequently-eventually-mono[OF freq ev]*) (*use ineq in blast*)  
**moreover have** *eventually  $(\lambda p. \text{fact } (p-1)^{\wedge n} > C' * C p^{\wedge n})$  sequentially*  
**proof** (*cases R = 0*)  
**case** *True*  
**have** *eventually  $(\lambda p. p * n > 1)$  at-top* **using**  $\langle n > 0 \rangle$   
**by** (*intro eventually-compose-filterlim[OF eventually-gt-at-top] mult-nat-right-at-top*)  
**thus** *?thesis*  
**by** *eventually-elim (use  $\langle n > 0 \rangle$  True in  $\langle$ auto simp: C-def power-0-left mult-ac $\rangle$ )*  
**next**



```

case False
hence  $R > 0$ 
  using  $\langle R \geq 0 \rangle$  by auto
define  $D :: \text{real}$  where  $D = (2 * R * |real-of-int l|) ^ n$ 
have  $D > 0$ 
  using  $\langle R > 0 \rangle \langle l > 0 \rangle$  unfolding  $D$ -def by (intro zero-less-power) auto

have  $(\lambda p. C' * C p ^ n) \in O(\lambda p. C p ^ n)$ 
  by simp
also have  $(\lambda p. C p ^ n) \in O(\lambda p. ((2 * R * l) ^ (n * p)) ^ n)$ 
proof (rule landau-o.big-power[OF bighetaD1])
  have  $np$ : eventually  $(\lambda p. p * n > 0)$  at-top using  $\langle n > 0 \rangle$ 
  by (intro eventually-compose-filterlim[OF eventually-gt-at-top] mult-nat-right-at-top)
  have eventually  $(\lambda p. (2 * R) * C p = (2 * R * l) ^ (n * p))$  at-top
    using  $np$ 
  proof eventually-elim
    case (elim p)
    have  $2 * R * C p = l ^ (n * p) * (2 * R) ^ (Suc (n * p - 1))$ 
      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  $(\lambda p. (2 * R) * C p) \in \Theta(\lambda p. (2 * R * l) ^ (n * p))$ 
  by (intro bighetaI-cong)
thus  $C \in \Theta(\lambda p. (2 * R * l) ^ (n * p))$ 
  using  $\langle R > 0 \rangle$  by simp
qed
also have  $\dots = O(\lambda p. (D ^ p) ^ n)$ 
  using  $\langle l > 0 \rangle$  by (simp flip: power-mult add: power2-eq-square mult-ac D-def)
also have  $(\lambda p. (D ^ p) ^ n) \in o(\lambda p. \text{fact } (p - 1) ^ n)$ 
proof (intro landau-o.small-power)
  have eventually  $(\lambda p. D ^ p = D * D ^ (p - 1))$  at-top
    using eventually-gt-at-top[of 0]
  by eventually-elim (use  $\langle D > 0 \rangle$  in  $\langle \text{auto simp flip: power-Suc} \rangle$ )
  hence  $(\lambda p. D ^ p) \in \Theta(\lambda p. D * D ^ (p - 1))$ 
    by (intro bighetaI-cong)
  hence  $(\lambda p. D ^ p) \in \Theta(\lambda p. D ^ (p - 1))$ 
    using  $\langle D > 0 \rangle$  by simp
  also have  $(\lambda p. D ^ (p - 1)) \in o(\lambda p. \text{fact } (p - 1))$ 
  by (intro smalloI-tendsto[OF filterlim-compose[OF power-over-fact-tendsto-0]]
      filterlim-minus-const-nat-at-top) auto
  finally show  $(\lambda p. D ^ p) \in o(\lambda x. \text{fact } (x - 1))$  .
qed fact+
finally have  $\text{smallo}$ :  $(\lambda p. C' * C p ^ n) \in o(\lambda p. \text{fact } (p - 1) ^ n)$  .
have eventually  $(\lambda p. |C' * C p ^ n| \leq 1/2 * \text{fact } (p - 1) ^ n)$  at-top
  using landau-o.smallD[OF smallo, of 1/2] by simp
thus eventually  $(\lambda p. C' * C p ^ n < \text{fact } (p - 1) ^ n)$  at-top

```

```

proof eventually-elim
  case (elim p)
  have  $C' * C p \wedge n \leq |C' * C p \wedge n|$ 
    by simp
  also have  $\dots \leq 1/2 * \text{fact } (p - 1) \wedge n$ 
    by fact
  also have  $\dots < \text{fact } (p - 1) \wedge n$ 
    by simp
  finally show ?case .
qed
qed
ultimately have frequently ( $\lambda p::\text{nat. False}$ ) sequentially
  by (rule frequently-eventually-mono) auto
thus False
  by simp
qed

```

## 7.2 Removing the restriction of full sets of conjugates

We will now remove the restriction that the  $\alpha_i$  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 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:  $(\sum_{x \in X. \text{of-int } (\beta x) * \text{exp } x}) = 0$ 
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]: finite Roots
  unfolding 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 squarefree (of-int-poly P :: complex poly)
  unfolding P-def of-int-poly-hom.hom-prod
proof (rule squarefree-prod-coprime; safe)
  fix x assume x ∈ X
  thus squarefree (of-int-poly (min-int-poly x)) :: complex poly
  by (intro squarefree-of-int-polyI) auto
next
  fix x y assume xy: x ∈ X y ∈ X min-int-poly x ≠ min-int-poly y
  thus Rings.coprime (of-int-poly (min-int-poly x)) (of-int-poly (min-int-poly y))
:: 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 = card Roots
obtain Root where Root: bij-betw Root {.. $n$ } Roots
  using ex-bij-betw-nat-finite[OF ⟨finite Roots⟩] unfolding n-def atLeast0LessThan
by metis
define unRoot :: complex ⇒ nat where unRoot = inv-into {.. $n$ } Root
have unRoot: bij-betw unRoot Roots {.. $n$ }
  unfolding unRoot-def by (intro bij-betw-inv-into Root)
have unRoot-Root [simp]: unRoot (Root i) = i if  $i < n$  for  $i$ 
  unfolding unRoot-def using Root that by (subst inv-into-f-f) (auto simp:
bij-betw-def)
have Root-unRoot [simp]: Root (unRoot x) = x if  $x \in \text{Roots}$  for  $x$ 
  unfolding unRoot-def using Root that by (subst f-inv-into-f) (auto simp:
bij-betw-def)
have [simp, intro]: Root i ∈ 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)

```

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

```

define convert-perm :: (nat ⇒ nat) ⇒ (complex ⇒ complex) where
  convert-perm = (λσ x. if x ∈ Roots then Root (σ (unRoot x)) else x)
have bij-convert: bij-betw convert-perm {σ. σ permutes {.. $n$ }} {σ. σ permutes
Roots}
  using bij-betw-permutations[OF Root] unfolding convert-perm-def unRoot-def
.
have permutes-convert-perm [intro]: convert-perm σ permutes Roots if σ permutes
{.. $n$ } for σ
  using that bij-convert unfolding bij-betw-def by blast
have convert-perm-compose: convert-perm (π ∘ σ) = convert-perm π ∘ con-
vert-perm σ
  if π permutes {.. $n$ } σ permutes {.. $n$ } for σ π
proof (intro ext)

```

```

fix  $x$  show convert-perm ( $\pi \circ \sigma$ )  $x = (\text{convert-perm } \pi \circ \text{convert-perm } \sigma) x$ 
proof (cases  $x \in \text{Roots}$ )
  case True
  thus ?thesis
    using permutes-in-image[OF that(2), of unRoot x]
    by (auto simp: convert-perm-def bij-betw-def)
qed (auto simp: 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 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
   $\text{Roots-ms} = \{X. \text{set-mset } X \subseteq \text{Roots} \wedge \text{size } X = \text{fact } n\}$ 
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  $\Rightarrow ((\text{complex} \Rightarrow \text{complex}) \Rightarrow \text{complex}) \text{ set}$  where
   $\text{tuples} = (\lambda X. \{f \in \text{perms} \rightarrow_E \text{Roots}. \text{image-mset } f (\text{mset-set } \text{perms}) = X\})$ 
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  $\times ((\text{complex} \Rightarrow \text{complex}) \Rightarrow \text{complex})$ ) set
where
   $\text{tuples}' = (\text{SIGMA } X:\text{Roots-ms}. \text{tuples } X)$ 

```

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  $\sigma: \sigma \text{ permutes } \text{Roots}$  for  $\sigma$   $X$ 
proof (rule bij-betwI)
  have  $*$ : ( $\lambda f. f \circ (\circ) \sigma$ )  $\in \text{tuples } X \rightarrow \text{tuples } X$  if  $\sigma: \sigma \text{ permutes } \text{Roots}$  for  $\sigma$ 
  proof
    fix  $f$  assume  $f: f \in \text{tuples } X$ 
    show  $f \circ (\circ) \sigma \in \text{tuples } X$ 
    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  $\neg(\sigma \circ \sigma') \text{ permutes } Roots$ 
  proof
    assume  $(\sigma \circ \sigma') \text{ permutes } Roots$ 
    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  $(f \circ (\circ) \sigma) (mset\text{-set } perms) =$ 
    image-mset  $f (image\text{-mset } ((\circ) \sigma) (mset\text{-set } perms))$ 
    by (rule multiset.map-comp [symmetric])
  also have image-mset  $((\circ) \sigma) (mset\text{-set } perms) = mset\text{-set } perms$ 
    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  $f \dots = X$ 
    using  $f$  by (auto simp: tuples-def)
  finally show image-mset  $(f \circ (\circ) \sigma) (mset\text{-set } perms) = X$  .
  qed
qed

show  $(\lambda f. f \circ (\circ) \sigma) \in tuples\ X \rightarrow tuples\ X$ 
  by (rule *) fact
show  $(\lambda f. f \circ (\circ) (inv\text{-into UNIV } \sigma)) \in tuples\ X \rightarrow tuples\ X$ 
  by (intro * permutes-inv) fact
show  $f \circ (\circ) \sigma \circ (\circ) (inv\text{-into UNIV } \sigma) = f$  if  $f \in tuples\ X$  for  $f$ 
  by (auto simp: fun-eq-iff o-def permutes-inverses[OF  $\sigma$ ])
show  $f \circ (\circ) (inv\text{-into UNIV } \sigma) \circ (\circ) \sigma = f$  if  $f \in tuples\ X$  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**  $R :: complex\ multiset \Rightarrow complex\ multiset$  **where**  
 $R = (\lambda X. image\text{-mset } (\lambda f. \sum_{\sigma \in perms. \sigma} (f \sigma)) (mset\text{-set } (tuples\ X)))$

We show that, for each such multiset, there is a content-free integer polyno-

mial 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 :: complex multiset  $\Rightarrow$  int poly
  where Q:  $\bigwedge X. X \in \text{Roots-ms} \implies \text{poly-roots (of-int-poly (Q X))} = R X$ 
          $\bigwedge X. X \in \text{Roots-ms} \implies \text{content (Q X)} = 1$ 
proof -
  {
    fix X :: complex multiset
    assume X: X  $\in$  Roots-ms
    define Q :: 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 ratpolys :: complex poly set where ratpolys = {p.  $\forall i. \text{poly.coeff } p \ i \in \mathbb{Q}$ }

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

```

```

next
  define lc :: complex where lc = of-int (Polynomial.lead-coeff P)
  show [:inverse lc:] ∈ ratpolys
    by (auto simp: ratpolys-def coeff-pCons lc-def split: nat.splits)
  show ∀ i. [:poly.coeff (of-int-poly P) i:] ∈ ratpolys
    by (auto simp: ratpolys-def coeff-pCons split: nat.splits)
  have lc ≠ 0
    by (auto simp: lc-def)
  thus [:inverse lc:] * [:lc:] = 1
    by auto
  have rsquarefree (of-int-poly P :: complex poly)
    using ‹squarefree (of-int-poly P :: complex poly)› by (intro square-
free-imp-rsquarefree)
  hence of-int-poly P = Polynomial.smult lc (∏ x∈Roots. [:-x, 1:])
    unfolding lc-def Roots-def of-int-hom.hom-lead-coeff[symmetric]
    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 lc (∏ i<n. [:- Root i, 1:]) .
next
  show symmetric-mpoly {..<n} Q
    unfolding symmetric-mpoly-def
  proof safe
    fix π assume π: π permutes {..<n}
    have mpoly-map-vars π Q = (∏ f∈tuples X. Const (pCons 0 1) - (∑ σ
| σ permutes {..<n}.
      Var ((π ∘ σ) (unRoot (f (convert-perm σ))))))
    by (simp add: Q-def permutes-bij[OF π])
    also have ... = (∏ f∈tuples X. Const (pCons 0 1) - (∑ σ | σ permutes
{..<n}.
      Var ((π ∘ σ) (unRoot ((f ∘ (λσ. convert-perm π ∘ σ)) (convert-perm
σ))))))
    using π 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 (∑ σ | σ permutes {..<n}. Var ((π ∘ σ) (unRoot ((f ∘ (λσ.
convert-perm π ∘ σ)) (convert-perm σ)))) =
      (∑ σ | σ permutes {..<n}. Var ((π ∘ σ) (unRoot (f (convert-perm
(π ∘ σ))))))
    using π by (intro sum.cong refl, subst convert-perm-compose) simp-all
    also have ... = (∑ σ | σ permutes {..<n}. Var (σ (unRoot (f
(convert-perm σ))))))
    using π by (rule setum-permutations-compose-left [symmetric])
  finally show ?case by simp
  qed auto
  finally show mpoly-map-vars π Q = Q .

```

```

qed
qed auto
also have insertion ( $\lambda x. [:\text{Root } x:]$ )  $Q = Q1$ 
  unfolding  $Q\text{-def } Q1\text{-def insertion-prod insertion-sum insertion-diff inser-$ 
 $\text{tion-Const insertion-Var}$ 
proof (intro prod.cong, goal-cases)
  case  $f: (2 f)$ 
  have ( $\sum \sigma \mid \sigma \text{ permutes } \{..<n\}. [:\text{Root } (\sigma (\text{unRoot } (f (\text{convert-perm } \sigma)))):]$ )
=
  ( $\sum \sigma \mid \sigma \text{ permutes } \{..<n\}. [:\text{convert-perm } \sigma (f (\text{convert-perm } \sigma))]$ )
proof (rule sum.cong, goal-cases)
  case ( $2 \sigma$ )
  have  $\text{convert-perm } \sigma \text{ permutes Roots}$ 
  using  $\text{bij-convert } 2$  by (auto simp:  $\text{bij-betw-def}$ )
  hence  $f (\text{convert-perm } \sigma) \in \text{Roots}$ 
  using  $f$  by (auto simp:  $\text{tuples-def perms-def}$ )
  thus ?case by (simp add:  $\text{convert-perm-def}$ )
qed simp-all
also have ... = ( $\sum \sigma \mid \sigma \text{ permutes Roots. } [:\sigma (f \sigma):]$ )
  by (rule sum.reindex-bij-betw[ $OF \text{ bij-convert}$ ])
finally show ?case
  by (simp flip:  $p\text{Cons-one coeff-lift-hom.hom-sum}$ )
qed simp-all
finally have  $Q1 \in \text{ratpolys}$ 
  by auto
then obtain  $Q2 :: \text{rat poly}$  where  $Q2: Q1 = \text{map-poly of-rat } Q2$ 
  unfolding  $\text{ratpolys-def}$  using  $\text{ratpolyE[of } Q1]$  by blast

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

have  $\text{poly-roots } (\text{of-int-poly } Q3) = \text{poly-roots } (\text{map-poly } (\text{of-rat } \circ \text{of-int}) Q3)$ 
  by simp
also have  $\text{map-poly } (\text{of-rat } \circ \text{of-int}) Q3 = \text{map-poly of-rat } (\text{map-poly of-int } Q3)$ 
  by (subst  $\text{map-poly-map-poly}$ ) auto
also have  $\text{poly-roots } \dots = \text{poly-roots } (\text{Polynomial.smult } (\text{of-rat } lc) \dots)$ 
  using  $\langle lc > 0 \rangle$  by simp
also have  $\text{Polynomial.smult } (\text{of-rat } lc) (\text{map-poly of-rat } (\text{map-poly of-int } Q3))$ 
=
   $\text{map-poly of-rat } (\text{Polynomial.smult } lc (\text{map-poly of-int } Q3))$ 
  by (simp add:  $\text{of-rat-hom.map-poly-hom-smult}$ )
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  $\langle \text{content } Q3 = 1 \rangle$  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  $\beta''$ -def using sum.not-neutral-contains-not-neutral by fastforce

```

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

have  $\beta''\ q = (\sum X \in \{Y\}. \text{int } (\text{count } (\text{prime-factorization } (Q\ X))\ q) * \text{prod-mset}$ 
 $(\text{image-mset } \beta'\ X))$ 
  unfolding  $\beta''\text{-def}$ 
proof  $(\text{intro } \text{sum.mono-neutral-right ballI})$ 
  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  $(\text{cases } \text{set-mset } Y' \subseteq X)$ 
  case  $Y'\text{-subset: True}$ 
  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  $(\text{intro } \text{dvd-imp-poly-roots-subset of-int-poly-hom.hom-dvd})$ 
auto
  with  $q(2)$  have  $(\sum \sigma \in \text{perms. } \sigma\ (f\ \sigma)) \in \# \text{poly-roots } (\text{of-int-poly } (Q\ Y'))$ 
  by  $(\text{meson } \text{mset-subset-eqD})$ 
  also have  $\text{poly-roots } (\text{of-int-poly } (Q\ Y')) = R\ Y'$ 
  using  $Q(1)[of\ Y']\ Y'$  by  $\text{auto}$ 
  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  $R\text{-def}$  using  $\text{fin-tuples}[of\ Y']$  by  $\text{auto}$ 

moreover have  $(\sum \sigma \in \text{perms. } \sigma\ (g\ \sigma)) <_{\mathbf{c}} (\sum \sigma \in \text{perms. } \sigma\ (f\ \sigma))$ 
proof  $(\text{rule } \text{sum-strict-mono-ex1-complex-lex})$ 
show  $le: \forall \sigma \in \text{perms. } \sigma\ (g\ \sigma) \leq_{\mathbf{c}} \sigma\ (f\ \sigma)$ 
proof
  fix  $\sigma$  assume  $\sigma: \sigma \in \text{perms}$ 
  hence  $\sigma': \sigma$  permutes  $\text{Roots}$ 
  by  $(\text{auto } \text{simp: perms-def})$ 
  have  $\text{image-mset } g\ (\text{mset-set } \text{perms}) = Y'$ 
  using  $g$  by  $(\text{auto } \text{simp: tuples-def})$ 
  also have  $\text{set-mset } \dots \subseteq X$ 
  by  $\text{fact}$ 
  finally have  $g\ ' \text{perms} \subseteq X$ 

```

```

    using ‹finite perms› by auto
  hence  $\sigma (g \sigma) \leq_{\mathbf{C}} \text{complex-lex.Max} (\sigma \text{ ' } X)$ 
    using ‹finite perms›  $\sigma$ 
    by (intro complex-lex.Max.coboundedI finite-imageI imageI)
      (auto simp: tuples-def)
  also have ... =  $\sigma (f \sigma)$ 
    using  $\sigma$  by (simp add: f-def permutes-inverses[OF  $\sigma$ ])
  finally show  $\sigma (g \sigma) \leq_{\mathbf{C}} \sigma (f \sigma)$  .
qed

have image-mset g (mset-set perms)  $\neq$  image-mset f (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  $\sigma$  ‹finite perms› by (auto simp: perms-def)
have  $\sigma (g \sigma) \neq \sigma (f \sigma)$ 
  using permutes-inj[OF ‹ $\sigma$  permutes Roots›]  $\sigma$  by (auto simp: inj-def)
moreover have  $\sigma (g \sigma) \leq_{\mathbf{C}} \sigma (f \sigma)$ 
  using le  $\sigma$  ‹finite perms› by auto
ultimately have  $\sigma (g \sigma) <_{\mathbf{C}} \sigma (f \sigma)$ 
  by simp
thus  $\exists \sigma \in \text{perms. } \sigma (g \sigma) <_{\mathbf{C}} \sigma (f \sigma)$ 
  using  $\sigma$  ‹finite perms› by auto
qed (use ‹finite perms› in simp-all)
ultimately show False by simp
qed
thus ?thesis by auto
qed (auto simp:  $\beta'$ -def)
qed (use ‹ $Y \in \text{Roots-ms}\beta' Y$ )
  by simp
also have ...  $\neq 0$ 
  using q nz ‹finite X› ‹ $X \neq \{\}$ › ‹finite perms› 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 =  $(\sum_{x \in X}. \beta x * \text{exp } x)$ 
  using sum0 ..
also have ... =  $(\sum_{x \in \text{Roots}. } \beta' x * \text{exp } x)$ 
  by (intro sum.mono-neutral-cong-left ‹ $X \subseteq \text{Roots}$ ›) (auto simp:  $\beta'$ -def)
also have ... dvd  $(\prod_{\sigma \in \text{perms}. } \sum_{x \in \text{Roots}. } \beta' x * \text{exp } (\sigma x))$ 
  by (rule dvd-prodI[OF ‹finite perms›])
  (use permutes-id[of Roots] in ‹simp-all add: id-def perms-def›)

```

**also have**  $\dots = (\sum f \in \text{perms} \rightarrow_E \text{Roots}. \prod \sigma \in \text{perms}. \beta' (f \sigma) * \exp (\sigma (f \sigma)))$   
**by** (*rule prod-sum-PiE*) *auto*  
**also have**  $\dots = (\sum f \in \text{perms} \rightarrow_E \text{Roots}. (\prod \sigma \in \text{perms}. \beta' (f \sigma)) * \exp (\sum \sigma \in \text{perms}. \sigma (f \sigma)))$   
**using**  $\langle \text{finite perms} \rangle$  **by** (*simp add: prod.distrib exp-sum*)  
**also have**  $\dots = (\sum (X, f) \in \text{tuples}'. (\prod \sigma \in \text{perms}. \beta' (f \sigma)) * \exp (\sum \sigma \in \text{perms}. \sigma (f \sigma)))$   
**using**  $\langle \text{finite perms} \rangle$   
**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**  $\dots = (\sum (X, f) \in \text{tuples}'. (\prod x \in \#X. \beta' x) * \exp (\sum \sigma \in \text{perms}. \sigma (f \sigma)))$   
**proof** (*safe intro!: sum.cong*)  
**fix**  $X :: \text{complex multiset}$  **and**  $f :: (\text{complex} \Rightarrow \text{complex}) \Rightarrow \text{complex}$   
**assume**  $(X, f) \in \text{tuples}'$   
**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**  $\dots = (\sum X \in \text{Roots-ms}. \text{of-int } (\prod x \in \#X. \beta' x) * (\sum f \in \text{tuples } X. \exp (\sum \sigma \in \text{perms}. \sigma (f \sigma))))$   
**by** (*simp add: sum-distrib-left*)  
**also have**  $\dots = (\sum X \in \text{Roots-ms}. \text{of-int } (\prod x \in \#X. \beta' x) * (\sum x \in \#R \text{ } X. \exp x))$   
**by** (*simp only: R-def multiset.map-comp o-def sum-unfold-sum-mset*)  
**also have**  $\dots = (\sum X \in \text{Roots-ms}. \text{of-int } (\prod x \in \#X. \beta' x) * (\sum x \in \#\text{poly-roots } (\text{of-int-poly } (Q X)). \exp x))$   
**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 ... =  $(\sum p \in \# \text{prime-factorization } (Q X). \sum x \in \# \text{poly-roots } (\text{of-int-poly } p)). \text{exp } x)$   
 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  $(\sum p \in \# \text{prime-factorization } (Q X). \sum x \in \# \text{poly-roots } (\text{of-int-poly } p)). \text{exp } x) =$   
 $(\sum p \in \# \text{prime-factorization } (Q X). \sum x \mid \text{ipoly } p x = 0. \text{exp } (x :: \text{complex}))$   
**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 ... =  $(\sum p. \text{count } (\text{prime-factorization } (Q X)) p * (\sum x \mid \text{ipoly } p x = 0. \text{exp } (x :: \text{complex})))$   
 = 0. exp (x :: complex))  
 by (rule sum-mset-conv-Sum-any)  
 also have of-int  $(\prod x \in \# X. \beta' x) * \dots =$   
 $(\sum p. \text{of-int } (\text{int } (\text{count } (\text{prime-factorization } (Q X)) p) * (\prod x \in \# X. \beta' x))) * (\sum x \mid \text{ipoly } p x = 0. \text{exp } x)$   
 by (subst Sum-any-right-distrib) (auto simp: mult-ac)  
**finally show** ?case by simp  
**qed** auto  
 also have ... =  $(\sum q. \text{of-int } (\beta'' q) * (\sum x \mid \text{ipoly } q x = 0. \text{exp } x))$   
**unfolding**  $\beta''$ -def of-int-sum  
 by (subst Sum-any-sum-swap [symmetric]) (auto simp: sum-distrib-right)  
 also have ... =  $(\sum q \mid \beta'' q \neq 0. \text{of-int } (\beta'' q) * (\sum x \mid \text{ipoly } q x = 0. \text{exp } x))$   
 by (intro Sum-any.expand-superset finite-subset[OF supp- $\beta''$ ]) auto  
**finally have**  $(\sum q \mid \beta'' q \neq 0. \text{of-int } (\beta'' q) * (\sum x \mid \text{ipoly } q x = 0. \text{exp } (x :: \text{complex}))) = 0$   
 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**  $(\sum q \mid \beta'' q \neq 0. \text{of-int } (\beta'' q) * (\sum x \mid \text{ipoly } q x = 0. \text{exp } (x :: \text{complex}))) \neq 0$   
**proof** (rule Hermite-Lindemann-aux1)  
 show finite {q.  $\beta'' q \neq 0$ }  
 by (rule finite-subset[OF supp- $\beta''$ ]) auto  
**next**  
 show pairwise Rings.coprime {q.  $\beta'' q \neq 0$ }  
**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 ⟨{x. β'' x ≠ 0} ≠ {}⟩ in auto)

  ultimately show False by contradiction
qed

```

### 7.3 Removing the restriction to integer coefficients

Next, we weaken the restriction that the  $\beta_i$  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 :: complex set and β :: complex ⇒ rat
  assumes finite X
  assumes nz: ∧x. x ∈ X ⇒ β x ≠ 0
  assumes alg: ∧x. x ∈ X ⇒ algebraic x
  assumes sum0: (∑ x∈X. of-rat (β x) * exp x) = 0
  shows X = {}

```

**proof** –

```

  define l :: int where l = Lcm ((snd ∘ quotient-of ∘ β) ` X)
  have [simp]: snd (quotient-of r) ≠ 0 for r
    using quotient-of-denom-pos'[of r] by simp
  have [simp]: l ≠ 0
    using ⟨finite X⟩ by (auto simp: l-def Lcm-0-iff)

```

have of-int l \* β x ∈  $\mathbb{Z}$  if x ∈ X for x

**proof** –

```

  define a b where a = fst (quotient-of (β x)) and b = snd (quotient-of (β x))
  have b > 0
    using quotient-of-denom-pos'[of β x] by (auto simp: b-def)
  have β x = of-int a / of-int b
    by (intro quotient-of-div) (auto simp: a-def b-def)
  also have of-int l * ... = of-int (l * a) / of-int b
    using ⟨b > 0⟩ by (simp add: field-simps)

```

**also have**  $\dots \in \mathbf{Z}$  **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**  $\beta'$  **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**  $\text{of-int } l * \text{of-rat } (\beta x) = \text{of-rat } (\text{of-int } l * \beta x)$   
**by** (*simp add: of-rat-mult*)  
**also have**  $\text{of-int } l * \beta x = \text{of-int } (\beta' x)$   
**using**  $2$  **by** (*rule  $\beta'$  [symmetric]*)  
**finally show** *?case* **by** (*simp add: mult-ac*)  
**qed** *simp-all*  
**finally show**  $\dots = 0$  ..  
**next**  
**fix**  $x$  **assume**  $x \in X$   
**hence**  $\text{of-int } (\beta' x) \neq (0 :: \text{rat})$  **using** *nz*  
**by** (*subst  $\beta'$  auto*)  
**thus**  $\beta' x \neq 0$   
**by** *auto*  
**qed** (*use alg ⟨finite X⟩ in auto*)  
**qed**

Next, we weaken the restriction that the  $\beta_i$  must be rational to them being algebraic. Similarly to before, this is done by multiplying over all possible permutations of the  $\beta_i$  (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**  $\beta :: \text{complex} \Rightarrow \text{complex}$   
**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**  $X: X \neq \{\}$   
**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  $\beta$  as roots. Note that this is a slight deviation from Baker's proof, who picks one polynomial for each  $\beta$  independently. I did it this way because, as Bernard [2] observed, it makes the proof a bit easier.

```

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

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  $(\text{auto simp: Roots-def } P\text{-def of-int-poly-hom.hom-prod poly-prod})$ 
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  $(\text{intro coprime-of-int-polyI}[OF \text{primes-coprime}]) \text{ auto}$ 
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  $(\text{auto simp: } n\text{-def } m\text{-def})$ 
have  $[\text{fin1 } [\text{simp}]: \text{finite } (X \rightarrow_E \text{Roots})$ 
by  $\text{auto}$ 
have  $[\text{simp}]: \text{card } (X \rightarrow_E \text{Roots}) = n \wedge m$ 
by  $(\text{subst card-PiE}) (\text{auto simp: } m\text{-def } n\text{-def})$ 

```

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



than  $n$ :

```

obtain Root where Root: bij-betw Root  $\{..<n\}$  Roots
using ex-bij-betw-nat-finite[OF  $\langle$ finite Roots $\rangle$ ] unfolding n-def atLeast0LessThan
by metis
define unRoot :: complex  $\Rightarrow$  nat where unRoot = inv-into  $\{..<n\}$  Root
have unRoot: bij-betw unRoot Roots  $\{..<n\}$ 
unfolding unRoot-def by (intro bij-betw-inv-into Root)
have unRoot-Root [simp]: unRoot (Root  $i$ ) =  $i$  if  $i < n$  for  $i$ 
unfolding unRoot-def using Root that by (subst inv-into-f-f) (auto simp:
bij-betw-def)
have Root-unRoot [simp]: Root (unRoot  $x$ ) =  $x$  if  $x \in$  Roots for  $x$ 
unfolding unRoot-def using Root that by (subst f-inv-into-f) (auto simp:
bij-betw-def)
have [simp, intro]: Root  $i \in$  Roots if  $i < n$  for  $i$ 
using Root that by (auto simp: bij-betw-def)
have [simp, intro]: unRoot  $x < n$  if  $x \in$  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 =  $\{Y. \text{set-mset } Y \subseteq X \wedge \text{size } Y = n \wedge m\}$ 
have [intro]: finite Roots-ms
unfolding Roots-ms-def by (rule finite-multisets-of-size) auto
define tuples :: complex multiset  $\Rightarrow$  (complex  $\Rightarrow$  complex)  $\Rightarrow$  complex set
where tuples =  $(\lambda Y. \{f \in (X \rightarrow_E \text{Roots}) \rightarrow_E X. \text{image-mset } f (\text{mset-set } (X \rightarrow_E \text{Roots})) = Y\})$ 
have [intro]: finite (tuples  $Y$ ) for  $Y$ 
unfolding tuples-def by (rule finite-subset[of - (X  $\rightarrow_E$  Roots)  $\rightarrow_E$  X]) auto

```

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

```

define convert-perm :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  (complex  $\Rightarrow$  complex) where
  convert-perm =  $(\lambda \sigma x. \text{if } x \in \text{Roots} \text{ then } \text{Root } (\sigma (\text{unRoot } x)) \text{ else } x)$ 
have bij-convert: bij-betw convert-perm  $\{\sigma. \sigma \text{ permutes } \{..<n\}\}$   $\{\sigma. \sigma \text{ permutes } \text{Roots}\}$ 
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  $(\lambda f. \text{restrict } (\lambda g. f (\text{restrict } (\pi \circ g) X)) (X \rightarrow_E \text{Roots})) (\text{tuples } Y)$ 
(tuples  $Y$ )
if  $\pi$ :  $\pi$  permutes Roots and  $Y \in$  Roots-ms for  $\pi$   $Y$ 

```

```

proof (rule bij-betwI)
  have *: (λf. restrict (λg. f (restrict (π ∘ g) X)) (X →E Roots)) ∈ tuples Y →
tuples Y
  if π: π permutes Roots for π
  proof
    fix f assume f: f ∈ tuples Y
    hence f': f ∈ (X →E Roots) →E X
      by (auto simp: tuples-def)
    define f' where f' = (λg. f (restrict (π ∘ g) X))
    have f' ∈ (X →E Roots) → X unfolding f'-def
      using f' bij-betw-apply[OF bij-betw-compose-left-perm-PiE[OF π, of X]] by
blast
    hence restrict f' (X →E Roots) ∈ (X →E Roots) →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  $\pi$ ])
  thus ?thesis using True
    by (auto simp: permutes-in-image[OF permutes-inv[OF  $\pi$ ]])
qed (use f in ⟨auto simp: tuples-def⟩)
qed
show ( $\lambda g \in X \rightarrow_E \text{Roots}. (\lambda g \in X \rightarrow_E \text{Roots}. f (\text{restrict } (\pi \circ g) X))$ 
  ( $\text{restrict } (\text{inv-into UNIV } \pi \circ g) X)) = f$  if  $f \in \text{tuples } Y$  for  $f$ 
  using *[OF that  $\pi$ ].
show ( $\lambda g \in X \rightarrow_E \text{Roots}. (\lambda g \in X \rightarrow_E \text{Roots}. f (\text{restrict } (\text{inv-into UNIV } \pi \circ g)$ 
 $X))$ 
  ( $\text{restrict } (\pi \circ g) X)) = f$  if  $f \in \text{tuples } Y$  for  $f$ 
  using *[OF that permutes-inv[OF  $\pi$ ]] permutes-inv-inv[OF  $\pi$ ] by simp
qed

```

We show that the coefficients in the expanded new sum are rational – again using the fundamental theorem of symmetric polynomials.

```

define  $\beta' :: \text{complex multiset} \Rightarrow \text{complex}$ 
  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  $Q :: \text{complex mpoly}$ 
  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 ( $\mathbb{Q} :: \text{complex set}$ )
      by standard auto
    then interpret ring-closed  $\mathbb{Q} :: \text{complex set}$ .
    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  $\pi Q = (\sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots}. \text{Var } (\pi$ 
      ( $\text{unRoot } (g (f g))))$ )
        unfolding Q-def by (simp add: permutes-bij[OF  $\pi$ ])
      also have  $\dots = (\sum f \in \text{tuples } Y. \prod g \in X \rightarrow_E \text{Roots}. \text{Var } (\text{unRoot } (g (f$ 
      ( $\text{restrict } (\pi' \circ g) X))))$ )
        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  $(\prod g \in X \rightarrow_E \text{Roots}. \text{Var } (\pi (\text{unRoot } (g (f g)))) =$ 

```

```

      (∏ g ∈ X →E Roots. Var (π (unRoot (restrict (π' ∘ g) X (f (restrict
(π' ∘ g) X))))))
    using π' by (intro prod.reindex-bij-betw [symmetric] bij-betw-compose-left-perm-PiE)
    also have ... = (∏ g ∈ X →E Roots. Var (unRoot (g (f (restrict (π' ∘ g)
X))))))
  proof (intro prod.cong refl arg-cong[of - - Var])
    fix g assume g: g ∈ X →E Roots
    have restrict (π' ∘ g) X ∈ X →E Roots
      using bij-betw-compose-left-perm-PiE[OF π', of X] g unfolding
bij-betw-def by blast
    hence *: f (restrict (π' ∘ g) X) ∈ X
      by (rule PiE-mem[OF f])
    hence **: g (f (restrict (π' ∘ g) X)) ∈ 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)))) ∈
{..E Roots. Var (unRoot (g ((λ g ∈ 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 ⊆ {..

```

**have**  $lc \neq 0$   
**unfolding**  $lc\text{-def}$  **by**  $auto$   
**thus**  $inverse (of\text{-int } lc) * (of\text{-int } lc :: complex) = 1$  **and**  $inverse (of\text{-int } lc) \in$   
**Q**  
**by**  $auto$   
**have**  $rsquarefree (of\text{-int-poly } P :: complex\ poly)$   
**using**  $\langle squarefree (of\text{-int-poly } P :: complex\ poly) \rangle$  **by**  $(intro\ squarefree\text{-imp}\text{-rsquarefree})$   
**hence**  $of\text{-int-poly } P = Polynomial.smult (of\text{-int } lc) (\prod_{x \in Roots.} [:-x, 1:])$   
**unfolding**  $lc\text{-def}$   $of\text{-int-hom.hom-lead-coeff}$   $[symmetric]$   $Roots\text{-def}$   
**by**  $(rule\ complex\text{-poly-decompose}\text{-rsquarefree } [symmetric])$   
**also have**  $(\prod_{x \in Roots.} [:-x, 1:]) = (\prod_{i < n.} [:-Root\ i, 1:])$   
**by**  $(rule\ prod.reindex\text{-bij-betw}[OF\ Root, symmetric])$   
**finally show**  $of\text{-int-poly } P = Polynomial.smult (of\text{-int } lc) (\prod_{i < n.} [:-Root$   
 $i, 1:])$  .  
**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**  $(simp\ add: Q\text{-def}$   $insertion\text{-sum}$   $insertion\text{-prod})$   
**also have**  $\dots = \beta' Y$   
**unfolding**  $\beta'\text{-def}$  **by**  $(intro\ sum.cong\ prod.cong\ refl\ Root\text{-unRoot})$   $(auto\ simp:$   
 $tuples\text{-def})$   
**finally show**  $?thesis$  .  
**qed**  
**hence**  $\forall Y \in Roots\text{-ms.} \exists x. \beta' Y = of\text{-rat } x$   
**by**  $(auto\ elim!: Rats\text{-cases})$   
**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  $\beta''$ :

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

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**  $(rule\ dvd\text{-prodI})$   $(use\ \langle \beta\ ' X \subseteq Roots \rangle$  **in**  $\langle auto\ simp: id\text{-def} \rangle)$   
**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**  $(rule\ prod\text{-sum}\text{-PiE})\ auto$

```

also have ... =  $(\sum_{f \in (X \rightarrow_E \text{Roots})} \rightarrow_E X. (\prod_{g \in X \rightarrow_E \text{Roots}. g} (f g)) * \text{exp}$ 
 $(\sum_{g \in X \rightarrow_E \text{Roots}. f} g))$ 
  by (simp add: prod.distrib exp-sum)
also have ... =  $(\sum_{(Y,f) \in \text{Sigma Roots-ms tuples}.}$ 
 $(\prod_{g \in X \rightarrow_E \text{Roots}. g} (f g)) * \text{exp} (\sum_{g \in X \rightarrow_E \text{Roots}. f} g))$ 
  by (intro sum.reindex-bij-witness[of - snd  $\lambda f. (\text{image-mset } f (\text{mset-set } (X \rightarrow_E$ 
Roots)), f])
  (auto simp: Roots-ms-def tuples-def)
also have ... =  $(\sum_{(Y,f) \in \text{Sigma Roots-ms tuples}.} (\prod_{g \in X \rightarrow_E \text{Roots}. g} (f g))$ 
 $* \text{exp} (\sum_{\#} Y))$ 
  by (intro sum.cong) (auto simp: tuples-def sum-unfold-sum-mset)
also have ... =  $(\sum_{Y \in \text{Roots-ms}.} \beta' Y * \text{exp} (\sum_{\#} Y))$ 
  unfolding  $\beta'$ -def sum-distrib-right by (rule sum.Sigma [symmetric]) auto
also have ... =  $(\sum_{Y \in \text{Roots-ms}.} \text{of-rat} (\beta'' Y) * \text{exp} (\sum_{\#} Y))$ 
  by (intro sum.cong) (auto simp:  $\beta''$ )
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**  $\alpha\text{-max}$  **where**  $\alpha\text{-max} = \text{complex-lex.Max } X$   
**have**  $[\text{simp}]$ :  $\alpha\text{-max} \in X$   
**unfolding**  $\alpha\text{-max-def}$  **using**  $\langle X \neq \{\} \rangle$  **by**  $(\text{intro complex-lex.Max-in})$  *auto*  
**define**  $Y\text{-max} :: \text{complex multiset}$  **where**  $Y\text{-max} = \text{replicate-mset } (n \wedge m)$   
 $\alpha\text{-max}$   
**define**  $f\text{-max}$  **where**  $f\text{-max} = \text{restrict } (\lambda-. \alpha\text{-max}) (X \rightarrow_E \text{Roots})$   
**have**  $[\text{simp}]$ :  $Y\text{-max} \in \text{Roots-ms}$   
**by**  $(\text{auto simp: } Y\text{-max-def Roots-ms-def})$   
**have**  $\text{tuples } Y\text{-max} = \{f\text{-max}\}$   
**proof** *safe*  
**have**  $\text{image-mset } (\lambda \in X \rightarrow_E \text{Roots}. \alpha\text{-max}) (\text{mset-set } (X \rightarrow_E \text{Roots})) =$   
 $\text{image-mset } (\lambda-. \alpha\text{-max}) (\text{mset-set } (X \rightarrow_E \text{Roots}))$   
**by**  $(\text{intro image-mset-cong})$  *auto*  
**thus**  $f\text{-max} \in \text{tuples } Y\text{-max}$   
**by**  $(\text{auto simp: } f\text{-max-def tuples-def } Y\text{-max-def image-mset-const-eq})$   
**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**  $(\text{auto simp: } Y\text{-max-def fun-eq-iff } f\text{-max-def})$   
**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**  $\beta'''$ -def  $\text{Roots-ms-def}$  ..  
**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**  $\text{set-mset-subset-singletonD}[\text{of } Y \alpha\text{-max}]$  *that*  
**by**  $(\text{auto simp: Roots-ms-def } Y\text{-max-def split: if-splits})$   
**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**  $y \leq_C \alpha\text{-max}$   
**using**  $y$  **unfolding**  $\alpha\text{-max-def}$  **by**  $(\text{intro complex-lex.Max-ge})$  *auto*  
**with**  $y$  **have**  $y <_C \alpha\text{-max}$

by *auto*  
 have \*:  $Y = \{\#y\# \} + (Y - \{\#y\# \})$   
 using *y by simp*  
 have *sum-mset*  $Y = y + \text{sum-mset } (Y - \{\#y\# \})$   
 by (*subst \**) *auto*  
 also have ...  $<_C \alpha\text{-max} + \text{sum-mset } (Y - \{\#y\# \})$   
 by (*intro complex-lex.add-strict-right-mono*) *fact*  
 also have ...  $\leq_C \alpha\text{-max} + \text{sum-mset } (\text{replicate-mset } (n \wedge m - 1) \alpha\text{-max})$   
 unfolding  $\alpha\text{-max-def}$  using that  $y \langle \text{set-mset } (Y - \{\#y\# \}) \subseteq X \rangle$   
 by (*intro complex-lex.add-left-mono sum-mset-mono-complex-lex*  
*rel-mset-replicate-mset-right complex-lex.Max-ge*)  
 (*auto simp: Roots-ms-def size-Diff-singleton*)  
 also have ...  $= \text{of-nat } (\text{Suc } (n \wedge m - 1)) * \alpha\text{-max}$   
 by (*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 ...  $= \beta'' Y\text{-max}$   
 by (*auto simp: when-def Y-max-def*)  
 also have *of-rat* ...  $= \beta' Y\text{-max}$   
 using  $\beta''[\text{of } Y\text{-max}]$  by *auto*  
 also have ...  $= (\prod_{g \in X \rightarrow_E \text{Roots.}} g (f\text{-max } g))$   
 by (*auto simp: \beta'-def \langle tuples Y-max = \{f-max\} \rangle*)  
 also have ...  $= (\prod_{g \in X \rightarrow_E \text{Roots.}} g \alpha\text{-max})$   
 by (*intro prod.cong*) (*auto simp: f-max-def*)  
 also have ...  $\neq 0$   
 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

## 7.4 The final theorem

We now additionally allow some of the  $\beta_i$  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 ... = 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  $(\sum x \in I. \beta x * \exp (\alpha x)) = 0$ 
  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  $0 = (\sum x \in I. \beta x * \exp (\alpha x))$ 
    using assms(5) ..
  also have ... =  $(\sum x \in I. (\beta \circ f) (\alpha x) * \exp (\alpha x))$ 
    by (intro sum.cong) auto
  also have ... =  $(\sum x \in \alpha' I. (\beta \circ f) x * \exp x)$ 
    using assms(4) by (subst sum.reindex) auto
  finally show  $(\sum x \in \alpha' I. (\beta \circ f) x * \exp x) = 0$  ..
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:  $(\sum (c,\alpha) \leftarrow xs. c * \exp \alpha) = 0$ 
  shows  $\forall c \in (\text{fst } \leftarrow \text{set } xs). c = 0$ 
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  $i \in \{..<n\}$ 
  hence ( $\text{fst } (xs ! i), \text{snd } (xs ! i)$ )  $\in \text{set } xs$ 
    by (auto simp: n-def)
  with alg show algebraic ( $\text{fst } (xs ! i)$ ) algebraic ( $\text{snd } (xs ! i)$ )
    by blast+
qed auto

show ?thesis
proof (intro ballI, elim imageE)
  fix c x assume  $cx: c = \text{fst } x \ x \in \text{set } xs$ 
  then obtain i where  $i \in \{..<n\} \ x = xs ! i$ 
    by (auto simp: set-conv-nth n-def)
  with  $* \ cx$  show  $c = 0$  by blast
qed
qed

```

## 7.5 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  $\mathbb{Z}$ , then  $e^{\alpha_1}, \dots, e^{\alpha_n}$  are algebraically independent over  $\mathbb{Q}$ .

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

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

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  $\longleftrightarrow$   
 $(\forall p. \text{vars } p \subseteq \{..<n\} \wedge (\forall m. \text{coeff } p \ m \in \mathbf{Q}) \wedge \text{insertion } a \ p = 0 \longrightarrow p = 0)$

**corollary** *Hermite-Lindemann-original*:

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

**proof** *safe*

**fix** p **assume** p: vars p  $\subseteq \{..<n\} \forall m. \text{coeff } p \ m \in \mathbf{Q}$  *insertion* ( $\lambda i. \text{exp } (\alpha \ i)$ ) p  
 $= 0$

**define**  $\alpha'$  **where**  $\alpha' = (\lambda m. \sum_{i < n. \text{of-nat } (\text{lookup } m \ i) * \alpha \ i)$

**define** I **where**  $I = \{m. \text{coeff } p \ m \neq 0\}$

**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**  $\forall x \in I. \text{coeff } p \ x = 0$

**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 = ( $\lambda m \ i. \text{int } (\text{lookup } m \ i)$ )

**interpret** *int*: *Modules.module*  $\lambda r \ x. \text{of-int } r * (x :: \text{complex})$

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

```

## 7.6 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*: *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*  $x$   $\exp y = (x :: \text{complex})$   $x \neq 1$

**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*  $x$   $x \neq 0$   $x \neq 1$

**shows**  $\neg \text{algebraic } (\text{Ln } x)$

**by** (*rule transcendental-complex-logarithm*) (*use assms in auto*)

The transcendence of  $e$  and  $\pi$ , 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**  $\neg$ algebraic (i \* pi)  
**by** (rule transcendental-complex-logarithm[of -1]) auto  
**thus** ?thesis **by** simp  
**qed**

## 7.7 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  $z \neq 0$   
**shows**  $\neg$ algebraic (sinh  $z ::$  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  $z \neq 0$   
**shows**  $\neg$ algebraic (cosh  $z ::$  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  $z \neq 0$   
**shows**  $\neg$ algebraic (sin  $z ::$  complex)  
**unfolding** sin-conv-sinh **using** transcendental-sinh[of i \* z] *assms* **by** simp

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

**lemma** *tan-square-neq-neg1*: tan ( $z ::$  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 ≠ 0*  
**shows**  $\neg$ *algebraic (tan z :: complex)*  
**proof**  
**assume** *algebraic (tan z)*

**have** *nz1: real-of-int n + 1 / 2 ≠ 0 for n*  
**proof** –  
**have** *real-of-int (2 \* n + 1) / real-of-int 2 ∉ ℤ*  
 by (*intro fraction-not-in-ints*) *auto*  
**also have** *real-of-int (2 \* n + 1) / real-of-int 2 = real-of-int n + 1 / 2*  
 by *simp*  
**finally show**  $\dots \neq 0$   
 by *auto*  
**qed**

**have** *nz2: 1 + tan z<sup>2</sup> ≠ 0*  
**using** *tan-square-neq-neg1 [of z]* **by** (*subst add-eq-0-iff*)

**have** *nz3: cos z ≠ 0*  
**proof**  
**assume** *cos z = 0*  
**then obtain** *n where z = complex-of-real (real-of-int n \* pi) + complex-of-real pi / 2*  
 by (*subst (asm) cos-eq-0*) *blast*  
**also have**  $\dots = \text{complex-of-real } ((\text{real-of-int } n + 1 / 2) * \text{pi})$   
 by (*simp add: ring-distrib*)  
**also have** *algebraic*  $\dots \iff$  *algebraic*  $((\text{real-of-int } n + 1 / 2) * \text{pi})$   
 by (*rule algebraic-of-real-iff*)  
**also have**  $\neg$ *algebraic*  $((\text{real-of-int } n + 1 / 2) * \text{pi})$   
**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** *algebraic*  $(\tan z^2 / (1 + \tan z^2))$

using  $\langle \text{algebraic } (\tan z) \rangle$  by auto  
 finally have algebraic  $(\sin z ^ 2)$  .  
 hence algebraic  $(\sin z)$   
 by simp  
 thus False  
 using transcendental-sin[OF  $\langle \text{algebraic } z \rangle \langle z \neq 0 \rangle$ ] by contradiction  
 qed

**lemma transcendental-cot:**  
 assumes algebraic  $z$   $z \neq 0$   
 shows  $\neg$ algebraic  $(\cot z :: \text{complex})$   
**proof** –  
 have  $\neg$ algebraic  $(\tan z)$   
 by (rule transcendental-tan) fact+  
 also have algebraic  $(\tan z) \longleftrightarrow$  algebraic  $(\text{inverse } (\tan z))$   
 by simp  
 also have  $\text{inverse } (\tan z) = \cot z$   
 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  $(\tanh z :: \text{complex})$   
 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  $(\text{Arcsin } z)$   
**proof** –  
 have  $i * z + \text{csqrt } (1 - z^2) \neq 0$   
 using Arcsin-body-lemma by blast  
 moreover have  $i * z + \text{csqrt } (1 - z^2) \neq 1$   
**proof**  
 assume  $i * z + \text{csqrt } (1 - z^2) = 1$   
 hence  $\text{Arcsin } z = 0$   
 by (simp add: Arcsin-def)  
 hence  $\sin (\text{Arcsin } z) = 0$   
 by (simp only: sin-zero)  
 also have  $\sin (\text{Arcsin } z) = z$   
 by simp  
 finally show False using  $\langle z \neq 0 \rangle$  by simp  
 qed  
 ultimately have  $\neg$  algebraic  $(\text{Ln } (i * z + \text{csqrt } (1 - z^2)))$   
 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  $z + i * \text{csqrt } (1 - z^2) \neq 0$ 
    using Arccos-body-lemma by blast
  moreover have  $z + i * \text{csqrt } (1 - z^2) \neq 1$ 
  proof
    assume  $z + i * \text{csqrt } (1 - z^2) = 1$ 
    hence Arccos  $z = 0$ 
      by (simp add: Arccos-def)
    hence  $\cos$  (Arccos  $z$ ) = 1
      by (simp only: cos-zero)
    also have  $\cos$  (Arccos  $z$ ) =  $z$ 
      by simp
    finally show False using  $\langle z \neq 1 \rangle$  by simp
  qed
  ultimately have  $\neg$  algebraic (Ln ( $z + i * \text{csqrt } (1 - z^2)$ ))
    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  $\neg$ algebraic (Arctan  $z$ )
proof -
  have  $i * z \neq 1$   $1 + i * z \neq 0$ 
    using assms(2) by (auto simp: complex-eq-iff)
  hence  $\neg$  algebraic (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.