The Transcendence of π

Manuel Eberl

February 6, 2026

Abstract

This entry shows the transcendence of π based on the classic proof using the fundamental theorem of symmetric polynomials first given by von Lindemann in 1882, but the mostly formalisation follows the version by Niven [3]. The proof reuses much of the machinery developed in the AFP entry on the transcendence of e.

Contents

1 The Transcendence of π

theory *Pi-Transcendental*
imports
  E-Transcendental.E-Transcendental
  Symmetric-Polynomials.Symmetric-Polynomials
  HOL−Real-Asymp.Real-Asymp
begin

lemma *ring-homomorphism-to-poly* [*intro*]: *ring-homomorphism* ($\lambda i.$ [:*i*:])
  by *standard auto*

lemma (**in** *ring-closed*) *coeff-power-closed*:
  ($\bigwedge m.$ *coeff p m* $\in A$) $\Longrightarrow$ *coeff* (*p* ^ *n*) *m* $\in A$
  by (*induction n arbitrary*: *m*)
    (*auto simp*: *mpoly-coeff-1 coeff-mpoly-times intro*!: *prod-fun-closed*)

lemma (**in** *ring-closed*) *coeff-prod-closed*:
  ($\bigwedge x\ m.\ x \in X \Longrightarrow$ *coeff* (*f x*) *m* $\in A$) $\Longrightarrow$ *coeff* (*prod f X*) *m* $\in A$
  by (*induction X arbitrary*: *m rule*: *infinite-finite-induct*)
    (*auto simp*: *mpoly-coeff-1 coeff-mpoly-times intro*!: *prod-fun-closed*)

lemma *map-of-rat-of-int-poly* [*simp*]: *map-poly of-rat* (*of-int-poly p*) = *of-int-poly p*
  by (*intro poly-eqI*) (*auto simp*: *coeff-map-poly*)

Given a polynomial with rational coefficients, we can obtain an integer polynomial that differs from it only by a nonzero constant by clearing the denominators.

lemma *ratpoly-to-intpoly*:
  assumes $\forall$ *i*. *poly.coeff p i* $\in \mathbb{Q}$
  obtains *q c* **where** $c \neq 0$ *p* = *Polynomial.smult* (*inverse* (*of-nat c*)) (*of-int-poly q*)
proof (*cases p* = *0*)
  case *True*
  with *that*[*of 1 0*] **show** *?thesis* **by** *auto*
next
  case *False*
  from *assms* **obtain** *p′* **where** *p′*: *p* = *map-poly of-rat p′*
    using *ratpolyE* **by** *auto*
  define *c* **where** *c* = *Lcm* ((*nat* ∘ *snd* ∘ *quotient-of* ∘ *poly.coeff p′*) ‘ {..*Polynomial.degree p′*})
  have ¬*snd* (*quotient-of x*) ≤ *0* **for** *x*
    using *quotient-of-denom-pos*[*of x*, *OF surjective-pairing*] **by** *auto*
  hence $c \neq 0$ **by** (*auto simp*: *c-def*)
  define *q* **where** *q* = *Polynomial.smult* (*of-nat c*) *p′*

  have *poly.coeff q i* $\in \mathbb{Z}$ **for** *i*
  proof (*cases i* > *Polynomial.degree p′*)

```
    case False
    define m n
      where m = fst (quotient-of (poly.coeff p′ i))
        and n = nat (snd (quotient-of (poly.coeff p′ i)))
    have mn: n > 0 poly.coeff p′ i = of-int m / of-nat n
      using quotient-of-denom-pos[of poly.coeff p′ i, OF surjective-pairing]
            quotient-of-div[of poly.coeff p′ i, OF surjective-pairing]
      by (auto simp: m-def n-def)
    from False have n dvd c unfolding c-def
      by (intro dvd-Lcm) (auto simp: c-def n-def o-def not-less)
    hence of-nat c ∗ (of-int m / of-nat n) = (of-nat (c div n) ∗ of-int m :: rat)
      by (auto simp: of-nat-div)
    also have ... ∈ ℤ by auto
    finally show ?thesis using mn by (auto simp: q-def)
  qed (auto simp: q-def coeff-eq-0)
  with intpolyE obtain q′ where q′: q = of-int-poly q′ by auto
   moreover have p = Polynomial.smult (inverse (of-nat c)) (map-poly of-rat
(of-int-poly q′))
    unfolding smult-conv-map-poly q′[symmetric] p′ using ‹c ≠ 0›
    by (intro poly-eqI) (auto simp: coeff-map-poly q-def of-rat-mult)
  ultimately show ?thesis
    using q′ p′ ‹c ≠ 0› by (auto intro!: that[of c q′])
qed

lemma symmetric-mpoly-symmetric-sum:
  assumes ⋀π. π permutes A ⟹ g π permutes X
  assumes ⋀x π. x ∈ X ⟹ π permutes A ⟹ mpoly-map-vars π (f x) = f (g π
x)
  shows symmetric-mpoly A (∑x∈X. f x)
  unfolding symmetric-mpoly-def
proof safe
  fix π assume π: π permutes A
  have mpoly-map-vars π (sum f X) = (∑x∈X. mpoly-map-vars π (f x))
    by simp
  also have ... = (∑x∈X. f (g π x))
    by (intro sum.cong assms π refl)
  also have ... = (∑x∈g π ‘X. f x)
    using assms(1)[OF π] by (subst sum.reindex) (auto simp: permutes-inj-on)
  also have g π ‘ X = X
    using assms(1)[OF π] by (simp add: permutes-image)
  finally show mpoly-map-vars π (sum f X) = sum f X .
qed


lemma symmetric-mpoly-symmetric-prod:
  assumes g permutes X
  assumes ⋀x π. x ∈ X ⟹ π permutes A ⟹ mpoly-map-vars π (f x) = f (g x)
  shows symmetric-mpoly A (∏x∈X. f x)
  unfolding symmetric-mpoly-def
```

proof *safe*
  fix π **assume** π: π *permutes* A
  have *mpoly-map-vars* π (*prod f X*) = ($\prod x{\in}X.$ *mpoly-map-vars* π (f x))
    by *simp*
  also have ... = ($\prod x{\in}X.\ f\ (g\ x)$)
    by (*intro prod.cong assms* π *refl*)
  also have ... = ($\prod x{\in}g`X.\ f\ x$)
    using *assms* **by** (*subst prod.reindex*) (*auto simp*: *permutes-inj-on*)
  also have g ` $X = X$
    using *assms* **by** (*simp add*: *permutes-image*)
  finally show *mpoly-map-vars* π (*prod f X*) = *prod f X* **.**
qed

We now prove the transcendence of $i\pi$, from which the transcendence of π will follow as a trivial corollary. The first proof of this was given by von Lindemann [4]. The central ingredient is the fundamental theorem of symmetric functions.

The proof can, by now, be considered folklore and one can easily find many similar variants of it, but we mostly follows the nice exposition given by Niven [3].

An independent previous formalisation in Coq that uses the same basic techniques was given by Bernard et al. [2]. They later also formalised the much stronger Lindemann–Weierstraß theorem [1].

lemma *transcendental-i-pi*: ¬*algebraic* (i ∗ *pi*)
proof
  — Suppose $i\pi$ were algebraic.
  assume *algebraic* (i ∗ *pi*)
  — We obtain some nonzero integer polynomial that has $i\pi$ as a root. We can assume w. l. o. g. that the constant coefficient of this polynomial is nonzero.
  then obtain p
    where p: *poly* (*of-int-poly p*) (i ∗ *pi*) = *0* $p \neq$ *0* *poly.coeff p 0* $\neq$ *0*
    by (*elim algebraicE′-nonzero*) *auto*
  define n **where** n = *Polynomial.degree p*

  — We define the sequence of the roots of this polynomial:
  obtain *root* **where** *Polynomial.smult* (*Polynomial.lead-coeff* (*of-int-poly p*))
      ($\prod i{<}n.$ [:−*root i* :: *complex*, *1*:]) = *of-int-poly p*
    using *complex-poly-decompose′*[*of of-int-poly p*] **unfolding** *n-def* **by** *auto*
  note *root* = *this* [*symmetric*]

  — We note that $i\pi$ is, of course, among these roots.
  from p **and** *root* **obtain** *idx* **where** *idx*: *idx* < n *root idx* = i ∗ *pi*
    by (*auto simp*: *poly-prod*)

  — We now define a new polynomial P', whose roots are all numbers that arise as a sum of any subset of roots of p. We also count all those subsets that sum up to 0 and call their number A.
  define *root′* **where** *root′* = ($\lambda X.$ ($\sum j{\in}X.$ *root j*))

```
define P where P = (λi. ∏ X | X ⊆ {..<n} ∧ card X = i. [:−root′ X, 1:])
define P′ where P′ = (∏ i∈{0<..n}. P i)
define A where A = card {X∈Pow {..<n}. root′ X = 0}
have [simp]: P′ ≠ 0 by (auto simp: P′-def P-def)

— We give the name Roots′ to those subsets that do not sum to zero and note
that there is at least one, namely {iπ}.
define Roots′ where Roots′ = {X. X ⊆ {..<n} ∧ root′ X ≠ 0}
have [intro]: finite Roots′ by (auto simp: Roots′-def)
have {idx} ∈ Roots′ using idx by (auto simp: Roots′-def root′-def)
hence Roots′ ≠ {} by auto
hence card-Roots′: card Roots′ > 0 by (auto simp: card-eq-0-iff)

have P′-altdef: P′ = (∏ X∈Pow {..<n} − {{}}. [:−root′ X, 1:])
proof −
  have P′ = (∏ (i, X)∈(SIGMA x:{0<..n}. {X. X ⊆ {..<n} ∧ card X = x}).
[:− root′ X, 1:])
    unfolding P′-def P-def by (subst prod.Sigma) auto
  also have … = (∏ X∈Pow{..<n} − {{}}. [:− root′ X, 1:])
    using card-mono[of {..<n}]
    by (intro prod.reindex-bij-witness[of - λX. (card X, X) λ(-, X). X])
      (auto simp: case-prod-unfold card-gt-0-iff intro: finite-subset[of - {..<n}])
  finally show ?thesis .
qed

— Clearly, A is nonzero, since the empty set sums to 0.
have A > 0
proof −
  have {} ∈ {X∈Pow {..<n}. root′ X = 0}
    by (auto simp: root′-def)
  thus ?thesis by (auto simp: A-def card-gt-0-iff)
qed

— Since e^iπ + 1 = 0, we know the following:
have 0 = (∏ i<n. exp (root i) + 1)
  using idx by force
— We rearrange this product of sums into a sum of products and collect all
summands that are 1 into a separate sum, which we call A:
also have … = (∑ X∈Pow {..<n}. ∏ i∈X. exp (root i))
  by (subst prod-add) auto
also have … = (∑ X∈Pow {..<n}. exp (root′ X))
  by (intro sum.cong refl, subst exp-sum [symmetric])
    (auto simp: root′-def intro: finite-subset[of - {..<n}])
also have Pow {..<n} = {X∈Pow {..<n}. root′ X ≠ 0} ∪ {X∈Pow {..<n}.
root′ X = 0}
  by auto
also have (∑ X∈…. exp (root′ X)) = (∑ X | X ⊆ {..<n} ∧ root′ X ≠ 0. exp
(root′ X)) +
                     (∑ X | X ⊆ {..<n} ∧ root′ X = 0. exp (root′ X))
```

by (*subst sum.union-disjoint*) *auto*
also have ($\sum X \mid X \subseteq \{..<n\} \wedge root'\ X = 0.\ exp\ (root'\ X)$) = *of-nat A*
by (*simp add*: *A-def*)
— Finally, we obtain the fact that the sum of exp(u) with u ranging over all the non-zero roots of P' is a negative integer.
finally have *eq*: ($\sum X \mid X \subseteq \{..<n\} \wedge root'\ X \neq 0.\ exp\ (root'\ X)$) = $-$*of-nat A*
by (*simp add*: *add-eq-0-iff2*)

— Next, we show that P' is a rational polynomial since it can be written as a symmetric polynomial expression (with rational coefficients) in the roots of p.
define *ratpolys* **where** *ratpolys* = $\{p$::*complex poly*. $\forall i$. *poly.coeff p i* $\in \mathbb{Q}\}$
have *ratpolysI*: $p \in$ *ratpolys* **if** $\bigwedge i$. *poly.coeff p i* $\in \mathbb{Q}$ **for** p
using *that* **by** (*auto simp*: *ratpolys-def*)

have $P' \in$ *ratpolys*
proof −
define *Pmv* :: *nat* $\Rightarrow$ *complex poly mpoly*
where *Pmv* = ($\lambda i.\ \prod X \mid X \subseteq \{..<n\} \wedge card\ X = i.$ *Const* ([:*0*,*1*:]) −
($\sum i \in X$. *monom* (*Poly-Mapping.single i 1*) *1*))
define $P'mv$ **where** $P'mv$ = ($\prod i \in \{0<..n\}$. *Pmv i*)
have *insertion* (λi. [:*root i*:]) $P'mv \in$ *ratpolys*
proof (*rule symmetric-poly-of-roots-in-subring*[**where** $l = \lambda x$. [:x:]])
show *ring-closed ratpolys*
by *standard* (*auto simp*: *ratpolys-def coeff-mult*)
then interpret *ring-closed ratpolys* **.**
show $\forall m$. *coeff* $P'mv$ *m* $\in$ *ratpolys*
by (*auto simp*: *P'mv-def Pmv-def coeff-monom when-def mpoly-coeff-Const coeff-pCons' ratpolysI*
intro!: *coeff-prod-closed minus-closed sum-closed uminus-closed*)
show $\forall i$. [:*poly.coeff* (*of-int-poly p*) *i*:] $\in$ *ratpolys*
by (*intro ratpolysI allI*) (*auto simp*: *coeff-pCons'*)
show [:*inverse* (*of-int* (*Polynomial.lead-coeff p*)):] $*$
[:*of-int* (*Polynomial.lead-coeff p*) :: *complex*:] = *1*
using ‹$p \neq 0$› **by** (*auto intro*!: *poly-eqI simp*: *field-simps*)
next
have *symmetric-mpoly* $\{..<n\}$ (*Pmv k*) **for** *k*
unfolding *symmetric-mpoly-def*
proof *safe*
fix π :: *nat* $\Rightarrow$ *nat* **assume** π: π *permutes* $\{..<n\}$
hence *mpoly-map-vars* π (*Pmv k*) =
($\prod X \mid X \subseteq \{..<n\} \wedge card\ X = k.$ *Const* [:*0*, *1*:] −
($\sum x \in X$. *MPoly-Type.monom* (*Poly-Mapping.single* (π *x*) (*Suc 0*)) *1*))
by (*simp add*: *Pmv-def permutes-bij*)
also have ... = ($\prod X \mid X \subseteq \{..<n\} \wedge card\ X = k.$ *Const* [:*0*, *1*:] −
($\sum x \in \pi$‘X. *MPoly-Type.monom* (*Poly-Mapping.single x* (*Suc 0*)) *1*))
using π **by** (*subst sum.reindex*) (*auto simp*: *permutes-inj-on*)
also have ... = ($\prod X \in (\lambda X.\ \pi$‘$X)$‘$\{X.\ X \subseteq \{..<n\} \wedge card\ X = k\}$. *Const*

[:*0*, *1*:] −
(∑*x*∈*X*. *MPoly-Type.monom* (*Poly-Mapping.single x* (*Suc 0*)) *1*))
by (*subst prod.reindex*) (*auto intro*!: *inj-on-image permutes-inj-on*[*OF* π])
also have (λ*X*. π ‘*X*) ‘{*X*. *X* ⊆ {..<*n*} ∧ *card X* = *k*} = {*X*. *X* ⊆ π ‘ {..<*n*} ∧ *card X* = *k*}
using π **by** (*subst image-image-fixed-card-subset*) (*auto simp*: *permutes-inj-on*)
also have π ‘ {..<*n*} = {..<*n*}
by (*intro permutes-image* π)
finally show *mpoly-map-vars* π (*Pmv k*) = *Pmv k* **by** (*simp add*: *Pmv-def*)
qed
thus *symmetric-mpoly* {..<*n*} *P′mv*
unfolding *P′mv-def* **by** (*intro symmetric-mpoly-prod*) *auto*
next
show *vars-P′mv*: *vars P′mv* ⊆ {..<*n*}
unfolding *P′mv-def Pmv-def*
by (*intro order.trans*[*OF vars-prod*] *UN-least order.trans*[*OF vars-diff*]
Un-least order.trans[*OF vars-sum*] *order.trans*[*OF vars-monom-subset*]) *auto*
qed (*insert root*, *auto intro*!: *ratpolysI simp*: *coeff-pCons′*)
also have *insertion* (λ*i*. [:*root i*:]) (*Pmv k*) = *P k* **for** *k*
by (*simp add*: *Pmv-def insertion-prod insertion-diff insertion-sum root′-def P-def*
sum-to-poly del: *insertion-monom*)

hence *insertion* (λ*i*. [:*root i*:]) *P′mv* = *P′*
by (*simp add*: *P′mv-def insertion-prod P′-def*)
finally show *P′* ∈ *ratpolys* **.**
qed

— We clear the denominators and remove all powers of X from P' to obtain a new integer polynomial Q.
define *Q′* **where** *Q′* = (∏*X*∈*Roots′*. [:− *root′ X*, *1*:])
have *P′* = (∏*X*∈*Pow* {..<*n*}−{{}}. [:−*root′ X*, *1*:])
by (*simp add*: *P′-altdef*)
also have *Pow* {..<*n*}−{{}} = *Roots′* ∪
{*X*. *X* ∈ *Pow* {..<*n*} − {{}} ∧ *root′ X* = *0*} **by** (*auto simp*: *root′-def Roots′-def*)
also have (∏*X*∈. . . . [:−*root′ X*, *1*:]) =
Q′ ∗ [:*0*, *1*:] ^ *card* {*X*. *X* ⊆ {..<*n*} ∧ *X* ≠ {} ∧ *root′ X* = *0*}
by (*subst prod.union-disjoint*) (*auto simp*: *Q′-def Roots′-def*)
also have {*X*. *X* ⊆ {..<*n*} ∧ *X* ≠ {} ∧ *root′ X* = *0*} = {*X*. *X* ⊆ {..<*n*} ∧ *root′ X* = *0*} − {{}}
by *auto*
also have *card* . . . = *A* − *1* **unfolding** *A-def*
by (*subst card-Diff-singleton*) (*auto simp*: *root′-def*)
finally have *Q′*: *P′* = *Polynomial.monom 1* (*A* − *1*) ∗ *Q′*
by (*simp add*: *Polynomial.monom-altdef*)

have *degree-Q′*: *Polynomial.degree P′ = Polynomial.degree Q′* + (*A* − *1*)
by (*subst Q′*)
(*auto simp*: *Q′-def Roots′-def degree-mult-eq Polynomial.degree-monom-eq degree-prod-sum-eq*)

have ∀ *i*. *poly.coeff Q′ i* ∈ ℚ
proof
fix *i* :: *nat*
have *poly.coeff Q′ i* = *Polynomial.coeff P′* (*i* + (*A* − *1*))
by (*simp add*: *Q′ Polynomial.coeff-monom-mult*)
also have . . . ∈ ℚ **using** ‹*P′* ∈ *ratpolys*› **by** (*auto simp*: *ratpolys-def*)
finally show *poly.coeff Q′ i* ∈ ℚ **.**
qed
from *ratpoly-to-intpoly*[*OF this*] **obtain** *c Q*
where [*simp*]: *c* ≠ *0* **and** *Q*: *Q′* = *Polynomial.smult* (*inverse* (*of-nat c*)) (*of-int-poly Q*)
by *metis*
have [*simp*]: *Q* ≠ *0* **using** *Q Q′* **by** *auto*
have *Q′*: *of-int-poly Q* = *Polynomial.smult* (*of-nat c*) *Q′*
using *Q* **by** *simp*
have *degree-Q*: *Polynomial.degree Q = Polynomial.degree Q′*
by (*subst Q*) *auto*
have *Polynomial.lead-coeff* (*of-int-poly Q* :: *complex poly*) = *c*
by (*subst Q′*) (*simp-all add*: *degree-Q Q′-def Polynomial.lead-coeff-prod*)
hence *lead-coeff-Q*: *Polynomial.lead-coeff Q* = *int c*
using *of-int-eq-iff*[*of Polynomial.lead-coeff Q of-nat c*] **by** (*auto simp del*: *of-int-eq-iff*)
have *Q-decompose*: *of-int-poly Q* =
Polynomial.smult (*of-nat c*) (∏ *X*∈*Roots′*. [:− *root′ X*, *1*:])
by (*subst Q′*) (*auto simp*: *Q′-def lead-coeff-Q*)
have *poly* (*of-int-poly Q*) (i ∗ *pi*) = *0*
using ‹{*idx*} ∈ *Roots′*› ‹*finite Roots′*› *idx*
by (*force simp*: *root′-def Q-decompose poly-prod*)
have *degree-Q*: *Polynomial.degree* (*of-int-poly Q* :: *complex poly*) = *card Roots′*
by (*subst Q′*) (*auto simp*: *Q′-def degree-prod-sum-eq*)
have *poly* (*of-int-poly Q*) (*0* :: *complex*) ≠ *0*
by (*subst Q′*) (*auto simp*: *Q′-def Roots′-def poly-prod*)
hence [*simp*]: *poly Q 0* ≠ *0* **by** *simp*
have [*simp*]: *poly* (*of-int-poly Q*) (*root′ Y*) = *0* **if** *Y* ∈ *Roots′* **for** *Y*
using *that* ‹*finite Roots′*› **by** (*auto simp*: *Q′ Q′-def poly-prod*)

— We find some closed ball that contains all the roots of *Q*.
define *r* **where** *r* = *Polynomial.degree Q*
have *r* > *0* **using** *degree-Q card-Roots′* **by** (*auto simp*: *r-def*)
define *Radius* **where** *Radius* = *Max* ((λ*Y*. *norm* (*root′ Y*)) ‘ *Roots′*)
have *Radius*: *norm* (*root′ Y*) ≤ *Radius* **if** *Y* ∈ *Roots′* **for** *Y*
using ‹*finite Roots′*› *that* **by** (*auto simp*: *Radius-def*)
from *Radius*[*of* {*idx*}] **have** *Radius* ≥ *pi*
using *idx* **by** (*auto simp*: *Roots′-def norm-mult root′-def*)

hence *Radius-nonneg*: *Radius* $\geq$ *0* **and** *Radius* $>$ *0* **using** *pi-gt3* **by** *linarith+*

— Since this ball is compact, Q is bounded on it. We obtain such a bound.
have *compact* (*poly* (*of-int-poly Q* :: *complex poly*) ' *cball 0 Radius*)
by (*intro compact-continuous-image continuous-intros*) *auto*
then obtain *Q-ub*
where *Q-ub*: *Q-ub* $>$ *0*
$\bigwedge$*u* :: *complex*. *u* $\in$ *cball 0 Radius* $\Longrightarrow$ *norm* (*poly* (*of-int-poly Q*) *u*) $\leq$ *Q-ub*
by (*auto dest*!: *compact-imp-bounded simp*: *bounded-pos cball-def*)

— Using this, define another upper bound that we will need later.
define *fp-ub*
where *fp-ub* = (λ*p*. |*c*| ^ (*r* $*$ *p* − *1*) / *fact* (*p* − *1*) $*$ (*Radius* ^ (*p* − *1*) $*$ *Q-ub* ^ *p*))
have *fp-ub-nonneg*: *fp-ub p* $\geq$ *0* **for** *p*
unfolding *fp-ub-def* **using** ‹*Radius* $\geq$ *0*› *Q-ub*
by (*intro mult-nonneg-nonneg divide-nonneg-pos zero-le-power*) *auto*
define *C* **where** *C* = *card Roots′* $*$ *Radius* $*$ *exp Radius*

— We will now show that any sufficiently large prime number leads to *C* $*$ *fp-ub p* $\geq$ *1*, from which we will then derive a contradiction.
define *primes-at-top* **where** *primes-at-top* = *inf-class.inf sequentially* (*principal* {*p*. *prime p*})
have *eventually* (λ*p*. $\forall$ *x*$\in${*nat* |*poly Q 0*|, *c*, *A*}. *p* $>$ *x*) *sequentially*
by (*intro eventually-ball-finite ballI eventually-gt-at-top*) *auto*
hence *eventually* (λ*p*. $\forall$ *x*$\in${*nat* |*poly Q 0*|, *c*, *A*}. *p* $>$ *x*) *primes-at-top*
unfolding *primes-at-top-def eventually-inf-principal* **by** *eventually-elim auto*
moreover have *eventually* (λ*p*. *prime p*) *primes-at-top*
by (*auto simp*: *primes-at-top-def eventually-inf-principal*)
ultimately have *eventually* (λ*p*. *C* $*$ *fp-ub p* $\geq$ *1*) *primes-at-top*
proof *eventually-elim*
case (*elim p*)
hence *p*: *prime p p* $>$ *nat* |*poly Q 0*| *p* $>$ *c p* $>$ *A* **by** *auto*
hence *p* $>$ *1* **by** (*auto dest*: *prime-gt-1-nat*)

— We define the polynomial $f(X) = \frac{c^s}{(p-1)!} X^{p-1} Q(X)^p$, where c is the leading coefficient of Q. We also define $F(X)$ to be the sum of all its derivatives.
define *s* **where** *s* = *r* $*$ *p* − *1*
define *fp* :: *complex poly*
where *fp* = *Polynomial.smult* (*of-nat c* ^ *s* / *fact* (*p* − *1*))
(*Polynomial.monom 1* (*p* − *1*) $*$ *of-int-poly Q* ^ *p*)
define *Fp* **where** *Fp* = ($\sum$ *i*$\leq$*s*+*p*. (*pderiv* ^^ *i*) *fp*)
define *f F* **where** *f* = *poly fp* **and** *F* = *poly Fp*
have *degree-fp*: *Polynomial.degree fp* = *s* + *p* **using** *degree-Q card-Roots′* ‹*p* $>$ *1*›
by (*simp add*: *fp-def s-def degree-mult-eq degree-monom-eq*
degree-power-eq r-def algebra-simps)

— Using the same argument as in the case of the transcendence of e, we now consider the function

$$I(u) := e^u F(0) - F(u) = u \int_0^1 e^{(1-t)x} f(tx)\,\mathrm{d}t$$

whose absolute value can be bounded with a standard "maximum times length" estimate using our upper bound on f. All of this can be reused from the proof for e, so there is not much to do here. In particular, we will look at $\sum I(x_i)$ with the x_i ranging over the roots of Q and bound this sum in two different ways.

interpret *lindemann-weierstrass-aux fp* **.**
have *I-altdef*: *I* = (λ*u*. *exp u* ∗ *F 0* − *F u*)
  by (*intro ext*) (*simp add*: *I-def degree-fp F-def Fp-def poly-sum*)

— We show that *fp-ub* is indeed an upper bound for f.
have *fp-ub*: *norm* (*poly fp u*) ≤ *fp-ub p* **if** *u* ∈ *cball 0 Radius* **for** *u*
proof −
  have *norm* (*poly fp u*) = |*c*| ^ (*r* ∗ *p* − *1*) / *fact* (*p* − *1*) ∗ (*norm u* ^ (*p* − *1*) ∗
    norm (*poly* (*of-int-poly Q*) *u*) ^ *p*)
  by (*simp add*: *fp-def f-def s-def norm-mult poly-monom norm-divide norm-power*)
  also have ... ≤ *fp-ub p*
    unfolding *fp-ub-def* **using** *that Q-ub* ‹*Radius* ≥ *0*›
    by (*intro mult-left-mono*[*OF mult-mono*] *power-mono zero-le-power*) *auto*
  finally show *?thesis* **.**
qed

— We now show that the following sum is an integer multiple of p. This argument again uses the fundamental theorem of symmetric functions, exploiting that the inner sums are symmetric over the roots of Q.
have (∑ *i*=*p*..*s*+*p*. ∑ *Y*∈*Roots′*. *poly* ((*pderiv* ^^ *i*) *fp*) (*root′ Y*)) / *p* ∈ ℤ
proof (*subst sum-divide-distrib*, *intro Ints-sum*[*of* {*a*..*b*} **for** *a b*])
  fix *i* **assume** *i*: *i* ∈ {*p*..*s*+*p*}
  then obtain *roots′* **where** *roots′*: *distinct roots′ set roots′* = *Roots′*
    using *finite-distinct-list* ‹*finite Roots′*› **by** *metis*
  define *l* **where** *l* = *length roots′*
  define *fp′* **where** *fp′* = (*pderiv* ^^ *i*) *fp*
  define *d* **where** *d* = *Polynomial.degree fp′*
  — We define a multivariate polynomial for the inner sum $\sum f(x_i)/p$ in order to show that it is indeed a symmetric function over the x_i.
  define *R* **where** *R* = (*smult* (*1* / *of-nat p*) (∑ *k*≤*d*. ∑ *i*<*l*. *smult* (*poly.coeff fp′ k*)
    (*monom* (*Poly-Mapping.single i k*) (*1* / *of-int* (*c* ^ *k*)))) :: *complex mpoly*)

  — The j-th coefficient of the i-th derivative of f are integer multiples of $c^j p$ since $i \geq p$.
  have *integer*: *poly.coeff fp′ j* / (*of-nat c* ^ *j* ∗ *of-nat p*) ∈ ℤ **if** *j* ≤ *d* **for** *j*
  proof −

define fp′′ **where** fp′′ = Polynomial.monom 1 (p − 1) ∗ Q ^ p
define x
where x = c ^ s ∗ poly.coeff ((pderiv ^^ i) (Polynomial.monom 1 (p − 1) ∗ Q ^ p)) j
have [:fact p:] dvd ([:fact i:] :: int poly) **using** i
by (auto intro: fact-dvd)
also have [:fact i:] dvd ((pderiv ^^ i) (Polynomial.monom 1 (p − 1) ∗ Q ^ p))
by (rule fact-dvd-higher-pderiv)
finally have c ^ j ∗ fact p dvd x **unfolding** x-def of-nat-mult **using** that i
by (intro mult-dvd-mono)
(auto intro!: le-imp-power-dvd simp: s-def d-def fp′-def degree-higher-pderiv degree-fp)
hence of-int x / (of-int (c ^ j ∗ fact p) :: complex) ∈ ℤ
by (intro of-int-divide-in-Ints) auto
also have of-int x / (of-int (c ^ j ∗ fact p) :: complex) =
poly.coeff fp′ j / (of-nat c ^ j ∗ of-nat p) **using** ‹p > 1›
unfolding x-def fp′-def fp-def
by (simp add: fact-reduce[of p] field-simps hom-distribs higher-pderiv-smult
flip: of-int-hom.coeff-map-poly-hom)
finally show ?thesis **.**
qed

— Evaluating R yields is an integer since it is symmetric.
have insertion (λi. c ∗ root′ (roots′ ! i)) R ∈ ℤ
proof (intro symmetric-poly-of-roots-in-subring-monic allI)
define Q′ **where** Q′ = of-int-poly Q ∘ₚ [:0, 1 / of-nat c :: complex:]
show symmetric-mpoly {..<l} R **unfolding** R-def
by (intro symmetric-mpoly-smult symmetric-mpoly-sum[of {..d}] symmetric-mpoly-symmetric-sum)
(simp-all add: mpoly-map-vars-monom permutes-bij permutep-single bij-imp-bij-inv permutes-inv-inv)
show MPoly-Type.coeff R m ∈ ℤ **for** m **unfolding** R-def coeff-sum coeff-smult sum-distrib-left
using integer **by** (auto simp: R-def coeff-monom when-def intro!: Ints-sum)
show vars R ⊆ {..<l} **unfolding** R-def
by (intro order.trans[OF vars-smult] order.trans[OF vars-sum] UN-least
order.trans[OF vars-monom-subset]) auto
show ring-closed ℤ **by** standard auto

have (∏ i<l. [:− (of-nat c ∗ root′ (roots′ ! i)), 1:]) =
(∏ Y←roots′. [:− (of-nat c ∗ root′ Y), 1:])
by (subst prod.list-conv-set-nth) (auto simp: atLeast0LessThan l-def)
also have ... = (∏ Y∈Roots′. [:− (of-nat c ∗ root′ Y), 1:])
using roots′ **by** (subst prod.distinct-set-conv-list [symmetric]) auto
also have ... = (∏ Y∈Roots′. Polynomial.smult (of-nat c) ([:−root′ Y, 1:])) ∘ₚ [:0, 1 / c:]
by (simp add: pcompose-prod pcompose-pCons)
also have (∏ Y∈Roots′. Polynomial.smult (of-nat c) ([:−root′ Y, 1:])) =

Polynomial.smult (*of-nat c* ^ *card Roots'*) ($\prod$ *Y*∈*Roots'*. [:−*root'* *Y*, *1*:])
by (*subst prod-smult*) *auto*
also have ... = *Polynomial.smult* (*of-nat c* ^(*card Roots'* − *1*))
(*Polynomial.smult c* ($\prod$ *Y*∈*Roots'*. [:−*root' Y*, *1*:]))
using ‹*finite Roots'*› **and** ‹*Roots'* ≠ {}›
by (*subst power-diff*) (*auto simp*: *Suc-le-eq card-gt-0-iff*)
also have *Polynomial.smult c* ($\prod$ *Y*∈*Roots'*. [:−*root' Y*, *1*:]) = *of-int-poly* *Q*
using *Q-decompose* **by** *simp*
finally show *Polynomial.smult* (*of-nat c* ^(*card Roots'* − *1*)) *Q'* =
($\prod$ *i*<*l*. [:− (*of-nat c* ∗ *root'* (*roots'* ! *i*)), *1*:])
by (*simp add*: *pcompose-smult Q'-def*)
fix *i* :: *nat*
show *poly.coeff* (*Polynomial.smult* (*of-nat c* ^(*card Roots'* − *1*)) *Q'*) *i* ∈ ℤ
proof (*cases i Polynomial.degree Q rule*: *linorder-cases*)
case *greater*
thus *?thesis* **by** (*auto simp*: *Q'-def coeff-pcompose-linear coeff-eq-0*)
next
case *equal*
thus *?thesis* **using** ‹*Roots'* ≠ {}› *degree-Q card-Roots' lead-coeff-Q*
by (*auto simp*: *Q'-def coeff-pcompose-linear lead-coeff-Q power-divide* *power-diff*)
next
case *less*
have *poly.coeff* (*Polynomial.smult* (*of-nat c* ^(*card Roots'* − *1*)) *Q'*) *i* =
of-int (*poly.coeff Q i*) ∗ (*of-int* (*c* ^(*card Roots'* − *1*)) / *of-int* (*c* ^ *i*))
by (*auto simp*: *Q'-def coeff-pcompose-linear power-divide*)
also have ... ∈ ℤ **using** *less degree-Q*
by (*intro Ints-mult of-int-divide-in-Ints*) (*auto intro*!: *le-imp-power-dvd*)
finally show *?thesis* **.**
qed
qed *auto*
— Moreover, by definition, evaluating R gives us $\sum f(x_i)/p$.
also have *insertion* (λ*i*. *c* ∗ *root'* (*roots'* ! *i*)) *R* =
($\sum$ *Y*←*roots'*. *poly fp'* (*root' Y*)) / *of-nat p*
by (*simp add*: *insertion-sum R-def poly-altdef d-def sum-list-sum-nth* *atLeast0LessThan*
l-def power-mult-distrib algebra-simps
sum.swap[*of* - {..*Polynomial.degree fp'*}] *del*: *insertion-monom*)
also have ... = ($\sum$ *Y*∈*Roots'*. *poly* ((*pderiv* ^^ *i*) *fp*) (*root' Y*)) / *of-nat p*
using *roots'* **by** (*subst sum-list-distinct-conv-sum-set*) (*auto simp*: *fp'-def* *poly-pcompose*)
finally show ... ∈ ℤ **.**
qed
then obtain *K* **where** *K*: ($\sum$ *i*=*p*..*s*+*p*. $\sum$ *Y*∈*Roots'*.
poly ((*pderiv* ^^ *i*) *fp*) (*root' Y*)) = *of-int K* ∗ *p*
using ‹*p* > *1*› **by** (*auto elim*!: *Ints-cases simp*: *field-simps*)

— Next, we show that $F(0)$ is an integer and coprime to p.

```
obtain F0 :: int where F0: F 0 = of-int F0 coprime (int p) F0
proof −
  have (∑ i=p..s + p. poly ((pderiv ^^ i) fp) 0) / of-nat p ∈ ℤ
    unfolding sum-divide-distrib
  proof (intro Ints-sum)
    fix i assume i: i ∈ {p..s+p}
    hence fact p dvd poly ((pderiv ^^ i) ([:0, 1:] ^ (p − 1) ∗ Q ^ p)) 0
      by (intro fact-dvd-poly-higher-pderiv-aux') auto
     then obtain k where k: poly ((pderiv ^^ i) ([:0, 1:] ^ (p − 1) ∗ Q ^ p))
0 = k ∗ fact p
      by auto

    have (pderiv ^^ i) fp = Polynomial.smult (of-nat c ^ s / fact (p − 1))
            (of-int-poly ((pderiv ^^ i) ([:0, 1:] ^ (p − 1) ∗ Q ^ p)))
   by (simp add: fp-def higher-pderiv-smult Polynomial.monom-altdef hom-distribs)
    also have poly ... 0 / of-nat p = of-int (c ^ s ∗ k)
      using k ‹p > 1› by (simp add: fact-reduce[of p])
    also have ... ∈ ℤ by simp
    finally show poly ((pderiv ^^ i) fp) 0 / of-nat p ∈ ℤ .
  qed
  then obtain S where S: (∑ i=p..s + p. poly ((pderiv ^^ i) fp) 0) = of-int
S ∗ p
    using ‹p > 1› by (auto elim!: Ints-cases simp: field-simps)

  have F 0 = (∑ i≤s + p. poly ((pderiv ^^ i) fp) 0)
    by (auto simp: F-def Fp-def poly-sum)
  also have ... = (∑ i∈insert (p − 1) {p..s + p}. poly ((pderiv ^^ i) fp) 0)
  proof (intro sum.mono-neutral-right ballI)
    fix i assume i: i ∈ {..s + p} − insert (p − 1) {p..s + p}
    hence i < p − 1 by auto
    have Polynomial.monom 1 (p − 1) dvd fp
      by (auto simp: fp-def intro: dvd-smult)
    with i show poly ((pderiv ^^ i) fp) 0 = 0
        by (intro poly-higher-pderiv-aux1'[of - p − 1]) (auto simp: Polyno-
mial.monom-altdef)
  qed auto
  also have ... = poly ((pderiv ^^ (p − 1)) fp) 0 + of-int S ∗ of-nat p
    using ‹p > 1› S by (subst sum.insert) auto
  also have poly ((pderiv ^^ (p − 1)) fp) 0 = of-int (c ^ s ∗ poly Q 0 ^ p)
    using poly-higher-pderiv-aux2[of p − 1 0 of-int-poly Q ^ p :: complex poly]
    by (simp add: fp-def higher-pderiv-smult Polynomial.monom-altdef)
  finally have F 0 = of-int (S ∗ int p + c ^ s ∗ poly Q 0 ^ p)
    by simp
  moreover have coprime p c coprime (int p) (poly Q 0)
    using p by (auto intro!: prime-imp-coprime dest: dvd-imp-le-int[rotated])
  hence coprime (int p) (c ^ s ∗ poly Q 0 ^ p)
    by auto
```

hence *coprime* (*int p*) (*S* ∗ *int p* + *c* ^ *s* ∗ *poly Q 0* ^ *p*)
unfolding *coprime-iff-gcd-eq-1 gcd-add-mult* **by** *auto*
ultimately show *?thesis* **using** *that*[*of S* ∗ *int p* + *c* ^ *s* ∗ *poly Q 0* ^ *p*] **by** *blast*
qed

— Putting everything together, we have shown that $\sum I(x_i)$ is an integer coprime to p, and therefore a nonzero integer, and therefore has an absolute value of at least 1.

have ($\sum$ *Y*∈*Roots'*. *I* (*root' Y*)) = *F 0* ∗ ($\sum$ *Y*∈*Roots'*. *exp* (*root' Y*)) − ($\sum$ *Y*∈*Roots'*. *F* (*root' Y*))
by (*simp add*: *I-altdef sum-subtractf sum-distrib-left sum-distrib-right algebra-simps*)
also have . . . = −(*of-int* (*F0* ∗ *int A*) +
($\sum$ *i*≤*s*+*p*. $\sum$ *Y*∈*Roots'*. *poly* ((*pderiv* ^^ *i*) *fp*) (*root' Y*)))
using *F0* **by** (*simp add*: *Roots'-def eq F-def Fp-def poly-sum sum.swap*[*of* - {..*s*+*p*}])
also have ($\sum$ *i*≤*s*+*p*. $\sum$ *Y*∈*Roots'*. *poly* ((*pderiv* ^^ *i*) *fp*) (*root' Y*)) =
($\sum$ *i*=*p*..*s*+*p*. $\sum$ *Y*∈*Roots'*. *poly* ((*pderiv* ^^ *i*) *fp*) (*root' Y*))
proof (*intro sum.mono-neutral-right ballI sum.neutral*)
fix *i Y* **assume** *i*: *i* ∈ {..*s*+*p*} − {*p*..*s*+*p*} **and** *Y*: *Y* ∈ *Roots'*
have [:−*root' Y*, *1*:] ^ *p dvd of-int-poly Q* ^ *p*
by (*intro dvd-power-same*) (*auto simp*: *dvd-iff-poly-eq-0 Y*)
hence [:−*root' Y*, *1*:] ^ *p dvd fp*
by (*auto simp*: *fp-def intro*!: *dvd-smult*)
thus *poly* ((*pderiv* ^^ *i*) *fp*) (*root' Y*) = *0*
using *i* **by** (*intro poly-higher-pderiv-aux1'*) *auto*
qed *auto*
also have . . . = *of-int* (*K* ∗ *int p*) **using** *K* **by** *simp*
finally have ($\sum$ *Y*∈*Roots'*. *I* (*root' Y*)) = −*of-int* (*K* ∗ *int p* + *F0* ∗ *int A*)
by *simp*
moreover have *coprime p A*
using *p* ‹*A* > *0*› **by** (*intro prime-imp-coprime*) (*auto dest*!: *dvd-imp-le*)
hence *coprime* (*int p*) (*F0* ∗ *int A*)
using *F0* **by** *auto*
hence *coprime* (*int p*) (*K* ∗ *int p* + *F0* ∗ *int A*)
using *F0* **unfolding** *coprime-iff-gcd-eq-1 gcd-add-mult* **by** *auto*
hence *K* ∗ *int p* + *F0* ∗ *int A* ≠ *0*
using *p* **by** (*intro notI*) *auto*
hence *norm* (−*of-int* (*K* ∗ *int p* + *F0* ∗ *int A*) :: *complex*) ≥ *1*
unfolding *norm-minus-cancel norm-of-int* **by** *linarith*
ultimately have *1* ≤ *norm* ($\sum$ *Y*∈*Roots'*. *I* (*root' Y*)) **by** *metis*

— The M–L bound on the integral gives us an upper bound:
also have *norm* ($\sum$ *Y*∈*Roots'*. *I* (*root' Y*)) ≤
($\sum$ *Y*∈*Roots'*. *norm* (*root' Y*) ∗ *exp* (*norm* (*root' Y*)) ∗ *fp-ub p*)
proof (*intro sum-norm-le lindemann-weierstrass-integral-bound fp-ub fp-ub-nonneg*)
fix *Y u* **assume** ∗: *Y* ∈ *Roots' u* ∈ *closed-segment 0* (*root' Y*)
hence *closed-segment 0* (*root' Y*) ⊆ *cball 0 Radius*

using ‹*Radius* ≥ *0*› *Radius*[*of Y*] **by** (*intro closed-segment-subset*) *auto*
with ∗ **show** *u* ∈ *cball 0 Radius* **by** *auto*
qed
also have . . . ≤ (∑ *Y*∈*Roots*′. *Radius* ∗ *exp* (*Radius*) ∗ *fp-ub p*)
using *Radius* **by** (*intro sum-mono mult-right-mono mult-mono fp-ub-nonneg* ‹*Radius* ≥ *0*›) *auto*
also have . . . = *C* ∗ *fp-ub p* **by** (*simp add*: *C-def*)
finally show *1* ≤ *C* ∗ *fp-ub p* **.**
qed

— It now only remains to show that this inequality is inconsistent for large *p*. This is obvious, since the upper bound is an exponential divided by a factorial and therefore clearly tends to zero.
have (λ*p*. *C* ∗ *fp-ub p*) ∈ Θ(λ*p*. (*C* / (*Radius* ∗ |*c*|)) ∗ (*p* / *2* ^ *p*) ∗
((*2* ∗ |*c*| ^ *r* ∗ *Radius* ∗ *Q-ub*) ^ *p* / *fact p*))
(**is** - ∈ Θ(*?f*)) **using** *degree-Q card-Roots*′ ‹*Radius* > *0*›
by (*intro bigthetaI-cong eventually-mono*[*OF eventually-gt-at-top*[*of 0*]])
(*auto simp*: *fact-reduce power-mult* [*symmetric*] *r-def*
fp-ub-def power-diff power-mult-distrib)
also have *?f* ∈ *o*(λ*p*. *1* ∗ *1* ∗ *1*)
proof (*intro landau-o.big-small-mult landau-o.big-mult*)
have (λ*x*. (*real-of-int* (*2* ∗ |*c*| ^ *r*) ∗ *Radius* ∗ *Q-ub*) ^ *x* / *fact x*) ⟶ *0*
by (*intro power-over-fact-tendsto-0*)
thus (λ*x*. (*real-of-int* (*2* ∗ |*c*| ^ *r*) ∗ *Radius* ∗ *Q-ub*) ^ *x* / *fact x*) ∈ *o*(λ*x*. *1*)
by (*intro smalloI-tendsto*) *auto*
qed *real-asymp*+
finally have (λ*p*. *C* ∗ *fp-ub p*) ∈ *o*(λ-. *1*) **by** *simp*
from *smalloD-tendsto*[*OF this*] **have** (λ*p*. *C* ∗ *fp-ub p*) ⟶ *0* **by** *simp*
hence *eventually* (λ*p*. *C* ∗ *fp-ub p* < *1*) *at-top*
by (*intro order-tendstoD*) *auto*
hence *eventually* (λ*p*. *C* ∗ *fp-ub p* < *1*) *primes-at-top*
unfolding *primes-at-top-def eventually-inf-principal* **by** *eventually-elim auto*
moreover note ‹*eventually* (λ*p*. *C* ∗ *fp-ub p* ≥ *1*) *primes-at-top*›
— We therefore have a contradiction for any sufficiently large prime.
ultimately have *eventually* (λ*p*. *False*) *primes-at-top*
by *eventually-elim auto*

— Since sufficiently large primes always exist, this concludes the theorem.
moreover have *frequently* (λ*p*. *prime p*) *sequentially*
using *primes-infinite* **by** (*simp add*: *cofinite-eq-sequentially*[*symmetric*] *Inf-many-def*)
ultimately show *False*
by (*auto simp*: *frequently-def eventually-inf-principal primes-at-top-def*)
qed

lemma *pcompose-conjugates-integer*:
assumes ⋀*i*. *poly.coeff p i* ∈ ℤ
shows *poly.coeff* (*pcompose p* [:*0*, i:] ∗ *pcompose p* [:*0*, −i:]) *i* ∈ ℤ
proof −
let *?c* = λ*i*. *poly.coeff p i* :: *complex*

```
  have poly.coeff (pcompose p [:0, i:] * pcompose p [:0, −i:]) i =
          i ^ i * (∑ k≤i. (−1) ^(i − k) * ?c k * ?c (i − k))
    unfolding coeff-mult sum-distrib-left
    by (intro sum.cong) (auto simp: coeff-mult coeff-pcompose-linear power-minus′
                                    power-diff field-simps intro!: Ints-sum)
  also have (∑ k≤i. (−1) ^(i − k) * ?c k * ?c (i − k)) =
          (∑ k≤i. (−1) ^ k * ?c k * ?c (i − k)) (is ?S1 = ?S2)
   by (intro sum.reindex-bij-witness[of - λk. i − k λk. i − k]) (auto simp: mult-ac)
  hence ?S1 = (?S1 + ?S2) / 2 by simp
  also have ... = (∑ k≤i. ((−1) ^ k + (−1) ^(i − k)) / 2 * ?c k * ?c (i − k))
    by (simp add: ring-distribs sum.distrib sum-divide-distrib [symmetric])
  also have ... = (∑ k≤i. (1 + (−1) ^ i) / 2 * (−1) ^ k * ?c k * ?c (i − k))
    by (intro sum.cong) (auto simp: power-add power-diff field-simps)
  also have i ^ i * ... ∈ ℤ
  proof (cases even i)
    case True
    thus ?thesis
      by (intro Ints-mult Ints-sum assms) (auto elim!: evenE simp: power-mult)
  next
    case False
    hence 1 + (−1) ^ i = (0 :: complex) by (auto elim!: oddE simp: power-mult)
    thus ?thesis by simp
  qed
  finally show ?thesis .
qed

lemma algebraic-times-i:
  assumes algebraic x
  shows   algebraic (i * x) algebraic (−i * x)
proof −
  from assms obtain p where p: poly p x = 0 ∀ i. poly.coeff p i ∈ ℤ p ≠ 0
    by (auto elim!: algebraicE)
  define p′ where p′ = pcompose p [:0, i:] * pcompose p [:0, −i:]
  have p′: poly p′ (i * x) = 0 poly p′ (−i * x) = 0 p′ ≠ 0
     by (auto simp: p′-def poly-pcompose algebra-simps p pcompose-eq-0-iff dest:
pcompose-eq-0)
  moreover have ∀ i. poly.coeff p′ i ∈ ℤ
    using p unfolding p′-def by (intro allI pcompose-conjugates-integer) auto
  ultimately show algebraic (i * x) algebraic (−i * x) by (intro algebraicI[of p′];
simp)+
qed

lemma algebraic-times-i-iff: algebraic (i * x) ⟷ algebraic x
  using algebraic-times-i[of x] algebraic-times-i[of i * x] by auto

theorem transcendental-pi: ¬algebraic pi
  using transcendental-i-pi by (simp add: algebraic-times-i-iff)

end
```

References

[1] S. Bernard. Formalization of the Lindemann-Weierstrass Theorem. In *Interactive Theorem Proving*, Brasilia, Brazil, Sept. 2017.

[2] S. Bernard, Y. Bertot, L. Rideau, and P.-Y. Strub. Formal proofs of transcendence for e and pi as an application of multivariate and symmetric polynomials. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, pages 76–87, New York, NY, USA, 2016. ACM.

[3] I. Niven. The transcendence of π. *The American Mathematical Monthly*, 46(8):469–471, 1939.

[4] F. von Lindemann. Ueber die Zahl π. *Mathematische Annalen*, 20(2):213–225, Jun 1882.