

# The Transcendence of $e$

Manuel Eberl

December 14, 2021

## Abstract

This work contains a formalisation of the proof that Euler's number  $e$  is transcendental. The proof follows the standard approach of assuming that  $e$  is algebraic and then using a specific integer polynomial to derive two inconsistent bounds, leading to a contradiction.

This approach can be found in many different sources; this formalisation mostly follows a PlanetMath article [1] by Roger Lipsett.

## Contents

<b>1</b>	<b>Proof of the Transcendence of <math>e</math></b>	<b>1</b>
1.1	Various auxiliary facts . . . . .	1
1.2	Lifting integer polynomials . . . . .	2
1.3	General facts about polynomials . . . . .	4
1.4	Main proof . . . . .	7

## 1 Proof of the Transcendence of $e$

**theory** *E-Transcendental*

**imports**

*HOL-Complex-Analysis.Complex-Analysis*

*HOL-Number-Theory.Number-Theory*

*HOL-Computational-Algebra.Polynomial*

**begin**

### 1.1 Various auxiliary facts

**lemma** *fact-dvd-pochhammer*:

**assumes**  $m \leq n + 1$

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

**proof** –

**have**  $(real\ n\ gchoose\ m) * \text{fact } m = of-int\ (\text{pochhammer } (int\ n - int\ m + 1)\ m)$

**by** (*simp add: gbinomial-pochhammer' pochhammer-of-int [symmetric]*)

**also have**  $(real\ n\ gchoose\ m) * \text{fact } m = of-int\ (int\ (n\ choose\ m) * \text{fact } m)$

**by** (*simp add: binomial-gbinomial*)

**finally have**  $int\ (n\ choose\ m) * \text{fact } m = \text{pochhammer } (int\ n - int\ m + 1)\ m$

by (subst (asm) of-int-eq-iff)  
 from this [symmetric] show ?thesis by simp  
 qed

**lemma** of-nat-eq-1-iff [simp]: of-nat  $x = (1 :: 'a :: \text{semiring-char-0}) \iff x = 1$   
 by (fact of-nat-eq-1-iff)

**lemma** prime-elem-int-not-dvd-neg1-power:  
 prime-elem  $(p :: \text{int}) \implies \neg p \text{ dvd } (-1) ^ n$   
 by (rule notI, frule (1) prime-elem-dvd-power, cases  $p \geq 0$ ) (auto simp: prime-elem-def)

**lemma** nat-fact [simp]: nat (fact  $n$ ) = fact  $n$   
 by (subst of-nat-fact [symmetric]) (rule nat-int)

**lemma** prime-dvd-fact-iff-int:  
 $p \text{ dvd fact } n \iff p \leq \text{int } n$  if prime  $p$   
 using that prime-dvd-fact-iff [of nat  $|p|$   $n$ ]  
 by auto (simp add: prime-ge-0-int)

**lemma** filterlim-minus-nat-at-top:  
 filterlim  $(\lambda n. n - k :: \text{nat})$  at-top at-top  
**proof** –  
 have sequentially = filtermap  $(\lambda n. n + k)$  at-top  
 by (auto simp: filter-eq-iff eventually-filtermap)  
 also have filterlim  $(\lambda n. n - k :: \text{nat})$  at-top ...  
 by (simp add: filterlim-filtermap filterlim-ident)  
 finally show ?thesis .  
 qed

**lemma** power-over-fact-tendsto-0:  
 $(\lambda n. (x :: \text{real}) ^ n / \text{fact } n) \longrightarrow 0$   
 using summable-exp[of  $x$ ] by (intro summable-LIMSEQ-zero) (simp add: sums-iff field-simps)

**lemma** power-over-fact-tendsto-0':  
 $(\lambda n. c * (x :: \text{real}) ^ n / \text{fact } n) \longrightarrow 0$   
 using tendsto-mult[OF tendsto-const[of  $c$ ] power-over-fact-tendsto-0[of  $x$ ]] by  
 simp

## 1.2 Lifting integer polynomials

**lift-definition** of-int-poly :: int poly  $\Rightarrow 'a :: \text{comm-ring-1}$  poly is  $\lambda g x. \text{of-int } (g x)$   
 by (auto elim: eventually-mono)

**lemma** coeff-of-int-poly [simp]: coeff (of-int-poly  $p$ )  $n = \text{of-int } (\text{coeff } p \ n)$   
 by transfer' simp

**lemma** of-int-poly-0 [simp]: of-int-poly  $0 = 0$   
 by transfer (simp add: fun-eq-iff)

**lemma** *of-int-poly-pCons* [simp]:  $of-int-poly (pCons\ c\ p) = pCons (of-int\ c) (of-int-poly\ p)$   
**by** *transfer'* (*simp add: fun-eq-iff split: nat.splits*)

**lemma** *of-int-poly-smult* [simp]:  $of-int-poly (smult\ c\ p) = smult (of-int\ c) (of-int-poly\ p)$   
**by** *transfer simp*

**lemma** *of-int-poly-1* [simp]:  $of-int-poly\ 1 = 1$   
**by** (*simp add: one-pCons*)

**lemma** *of-int-poly-add* [simp]:  $of-int-poly (p + q) = of-int-poly\ p + of-int-poly\ q$   
**by** *transfer'* (*simp add: fun-eq-iff*)

**lemma** *of-int-poly-mult* [simp]:  $of-int-poly (p * q) = (of-int-poly\ p * of-int-poly\ q)$   
**by** (*induction p*) *simp-all*

**lemma** *of-int-poly-sum* [simp]:  $of-int-poly (sum\ f\ A) = sum (\lambda x. of-int-poly (f\ x))\ A$   
**by** (*induction A rule: infinite-finite-induct*) *simp-all*

**lemma** *of-int-poly-prod* [simp]:  $of-int-poly (prod\ f\ A) = prod (\lambda x. of-int-poly (f\ x))\ A$   
**by** (*induction A rule: infinite-finite-induct*) *simp-all*

**lemma** *of-int-poly-power* [simp]:  $of-int-poly (p \wedge n) = of-int-poly\ p \wedge n$   
**by** (*induction n*) *simp-all*

**lemma** *of-int-poly-monom* [simp]:  $of-int-poly (monom\ c\ n) = monom (of-int\ c)\ n$   
**by** *transfer* (*simp add: fun-eq-iff*)

**lemma** *poly-of-int-poly* [simp]:  $poly (of-int-poly\ p) (of-int\ x) = of-int (poly\ p\ x)$   
**by** (*induction p*) *simp-all*

**lemma** *poly-of-int-poly-of-nat* [simp]:  $poly (of-int-poly\ p) (of-nat\ x) = of-int (poly\ p\ (int\ x))$   
**by** (*induction p*) *simp-all*

**lemma** *poly-of-int-poly-0* [simp]:  $poly (of-int-poly\ p)\ 0 = of-int (poly\ p\ 0)$   
**by** (*induction p*) *simp-all*

**lemma** *poly-of-int-poly-1* [simp]:  $poly (of-int-poly\ p)\ 1 = of-int (poly\ p\ 1)$   
**by** (*induction p*) *simp-all*

**lemma** *poly-of-int-poly-of-real* [simp]:  
 $poly (of-int-poly\ p) (of-real\ x) = of-real (poly (of-int-poly\ p)\ x)$   
**by** (*induction p*) *simp-all*

**lemma** *of-int-poly-eq-iff* [simp]:

*of-int-poly*  $p = (\text{of-int-poly } q :: 'a :: \{\text{comm-ring-1}, \text{ring-char-0}\} \text{ poly}) \longleftrightarrow p = q$   
**by** (*simp add: poly-eq-iff*)

**lemma** *of-int-poly-eq-0-iff* [simp]:

*of-int-poly*  $p = (0 :: 'a :: \{\text{comm-ring-1}, \text{ring-char-0}\} \text{ poly}) \longleftrightarrow p = 0$   
**using** *of-int-poly-eq-iff*[*of p 0*] **by** (*simp del: of-int-poly-eq-iff*)

**lemma** *degree-of-int-poly* [simp]:

*degree* (*of-int-poly*  $p :: 'a :: \{\text{comm-ring-1}, \text{ring-char-0}\} \text{ poly}$ ) = *degree*  $p$   
**by** (*simp add: degree-def*)

**lemma** *pderiv-of-int-poly* [simp]: *pderiv* (*of-int-poly*  $p$ ) = *of-int-poly* (*pderiv*  $p$ )

**by** (*induction p*) (*simp-all add: pderiv-pCons*)

**lemma** *higher-pderiv-of-int-poly* [simp]:

(*pderiv*  $\overset{\sim}{\sim} n$ ) (*of-int-poly*  $p$ ) = *of-int-poly* ((*pderiv*  $\overset{\sim}{\sim} n$ )  $p$ )  
**by** (*induction n*) *simp-all*

**lemma** *int-polyE*:

**assumes**  $\bigwedge n. \text{coeff } (p :: 'a :: \{\text{comm-ring-1}, \text{ring-char-0}\} \text{ poly}) n \in \mathbb{Z}$   
**obtains**  $p'$  **where**  $p = \text{of-int-poly } p'$

**proof** –

**from** *assms* **have**  $\forall n. \exists c. \text{coeff } p n = \text{of-int } c$  **by** (*auto simp: Ints-def*)

**hence**  $\exists c. \forall n. \text{of-int } (c n) = \text{coeff } p n$  **by** (*simp add: choice-iff eq-commute*)

**then obtain**  $c$  **where**  $c: \text{of-int } (c n) = \text{coeff } p n$  **for**  $n$  **by** *blast*

**have** [simp]:  $\text{coeff } (\text{Abs-poly } c) = c$

**proof** (*rule poly.Abs-poly-inverse, clarify*)

**have** *eventually*  $(\lambda n. n > \text{degree } p)$  *at-top* **by** (*rule eventually-gt-at-top*)

**hence** *eventually*  $(\lambda n. \text{coeff } p n = 0)$  *at-top*

**by** *eventually-elim* (*simp add: coeff-eq-0*)

**thus** *eventually*  $(\lambda n. c n = 0)$  *cofinite*

**by** (*simp add: c [symmetric] cofinite-eq-sequentially*)

**qed**

**have**  $p = \text{of-int-poly } (\text{Abs-poly } c)$

**by** (*rule poly-eqI*) (*simp add: c*)

**thus** *?thesis* **by** (*rule that*)

**qed**

### 1.3 General facts about polynomials

**lemma** *pderiv-power*:

*pderiv*  $(p \overset{\sim}{\sim} n) = \text{smult } (\text{of-nat } n) (p \overset{\sim}{\sim} (n - 1) * \text{pderiv } p)$

**by** (*cases n*) (*simp-all add: pderiv-power-Suc del: power-Suc*)

**lemma** *degree-prod-sum-eq*:

$(\bigwedge x. x \in A \implies f x \neq 0) \implies$

$\text{degree } (\text{prod } f A :: 'a :: \text{idom poly}) = (\sum_{x \in A}. \text{degree } (f x))$

**by** (*induction A* *rule: infinite-finite-induct*) (*auto simp: degree-mult-eq*)

**lemma** *pderiv-monom*:

$$pderiv (monom c n) = monom (of-nat n * c) (n - 1)$$

**by** (*cases n*)

(*simp-all add: monom-altdef pderiv-power-Suc pderiv-smult pderiv-pCons mult-ac del: power-Suc*)

**lemma** *power-poly-const* [*simp*]:  $[:c:] \hat{\ } n = [:c \hat{\ } n:]$

**by** (*induction n*) (*simp-all add: power-commutes*)

**lemma** *monom-power*:  $monom c n \hat{\ } k = monom (c \hat{\ } k) (n * k)$

**by** (*induction k*) (*simp-all add: mult-monom*)

**lemma** *coeff-higher-pderiv*:

$$coeff ((pderiv \hat{\ } m) f) n = pochhammer (of-nat (Suc n)) m * coeff f (n + m)$$

**by** (*induction m arbitrary: n*) (*simp-all add: coeff-pderiv pochhammer-rec alge-bra-simps*)

**lemma** *higher-pderiv-add*:  $(pderiv \hat{\ } n) (p + q) = (pderiv \hat{\ } n) p + (pderiv \hat{\ } n) q$

**by** (*induction n arbitrary: p q*) (*simp-all del: funpow.simps add: funpow-Suc-right pderiv-add*)

**lemma** *higher-pderiv-smult*:  $(pderiv \hat{\ } n) (smult c p) = smult c ((pderiv \hat{\ } n) p)$

**by** (*induction n arbitrary: p*) (*simp-all del: funpow.simps add: funpow-Suc-right pderiv-smult*)

**lemma** *higher-pderiv-0* [*simp*]:  $(pderiv \hat{\ } n) 0 = 0$

**by** (*induction n*) *simp-all*

**lemma** *higher-pderiv-monom*:

$$m \leq n + 1 \implies (pderiv \hat{\ } m) (monom c n) = monom (pochhammer (int n - int m + 1) m * c) (n - m)$$

**proof** (*induction m arbitrary: c n*)

**case** (*Suc m*)

**thus** *?case*

**by** (*cases n*)

(*simp-all del: funpow.simps add: funpow-Suc-right pderiv-monom pochhammer-rec' Suc.IH*)

**qed** *simp-all*

**lemma** *higher-pderiv-monom-eq-zero*:

$$m > n + 1 \implies (pderiv \hat{\ } m) (monom c n) = 0$$

**proof** (*induction m arbitrary: c n*)

**case** (*Suc m*)

**thus** *?case*

**by** (*cases n*)

(*simp-all del: funpow.simps add: funpow-Suc-right pderiv-monom pochhammer-rec' Suc.IH*)

**qed** *simp-all*

**lemma** *higher-pderiv-sum*:  $(pderiv \ \overset{\sim}{\sim} \ n) \ (sum \ f \ A) = (\sum \ x \in \ A. \ (pderiv \ \overset{\sim}{\sim} \ n) \ (f \ x))$   
**by** (*induction A rule: infinite-finite-induct*) (*simp-all add: higher-pderiv-add*)

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

$[:fact \ n \ :: \ int:] \ dvd \ (pderiv \ \overset{\sim}{\sim} \ n) \ p$

**proof** –

**have**  $[:fact \ n:] \ dvd \ (pderiv \ \overset{\sim}{\sim} \ n) \ (monom \ c \ k)$  **for**  $c \ :: \ int$  **and**  $k \ :: \ nat$

**by** (*cases  $n \leq k + 1$* )

(*simp-all add: higher-pderiv-monom higher-pderiv-monom-eq-zero fact-dvd-pochhammer const-poly-dvd-iff*)

**hence**  $[:fact \ n:] \ dvd \ (pderiv \ \overset{\sim}{\sim} \ n) \ (\sum \ k \leq \ degree \ p. \ monom \ (coeff \ p \ k) \ k)$

**by** (*simp-all add: higher-pderiv-sum dvd-sum*)

**thus** *?thesis* **by** (*simp add: poly-as-sum-of-monom*)

**qed**

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

$(fact \ n \ :: \ int) \ dvd \ poly \ ((pderiv \ \overset{\sim}{\sim} \ n) \ p) \ x$

**proof** –

**have**  $[:fact \ n:] \ dvd \ (pderiv \ \overset{\sim}{\sim} \ n) \ p$  **by** (*rule fact-dvd-higher-pderiv*)

**then obtain**  $q$  **where**  $(pderiv \ \overset{\sim}{\sim} \ n) \ p = [:fact \ n:] * q$  **by** (*erule dvdE*)

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

**qed**

**lemma** *fact-dvd-poly-higher-pderiv-aux'*:

$m \leq n \implies (fact \ m \ :: \ int) \ dvd \ poly \ ((pderiv \ \overset{\sim}{\sim} \ n) \ p) \ x$

**by** (*rule dvd-trans[OF fact-dvd fact-dvd-poly-higher-pderiv-aux]*) *simp-all*

**lemma** *algebraicE'*:

**assumes** *algebraic*  $(x \ :: \ 'a \ :: \ field-char-0)$

**obtains**  $p$  **where**  $p \neq 0$  *poly* (*of-int-poly*  $p$ )  $x = 0$

**proof** –

**from** *assms* **obtain**  $q$  **where**  $\bigwedge i. \ coeff \ q \ i \in \mathbb{Z} \ q \neq 0$  *poly*  $q \ x = 0$

**by** (*erule algebraicE*)

**moreover from** *this(1)* **obtain**  $q'$  **where**  $q = of-int-poly \ q'$  **by** (*erule int-polyE*)

**ultimately show** *?thesis* **by** (*intro that[of  $q'$ ]*) *simp-all*

**qed**

**lemma** *algebraicE'-nonzero*:

**assumes** *algebraic*  $(x \ :: \ 'a \ :: \ field-char-0) \ x \neq 0$

**obtains**  $p$  **where**  $p \neq 0$   $coeff \ p \ 0 \neq 0$  *poly* (*of-int-poly*  $p$ )  $x = 0$

**proof** –

**from** *assms(1)* **obtain**  $p$  **where**  $p \neq 0$  *poly* (*of-int-poly*  $p$ )  $x = 0$

**by** (*erule algebraicE'*)

**define**  $n \ :: \ nat$  **where**  $n = order \ 0 \ p$

**have** *monom 1 n dvd p* **by** (*simp add: monom-1-dvd-iff p n-def*)

**then obtain**  $q$  **where**  $q \equiv p \equiv monom \ 1 \ n * q$  **by** (*erule dvdE*)

**from**  $p$  **have**  $q \neq 0$  *poly* (of-int-poly  $q$ )  $x = 0$  **by** (auto simp:  $q$  poly-monom  
*assms*(2))  
**moreover from** *this* **have** order 0  $p = n +$  order 0  $q$  **by** (simp add:  $q$  order-mult)  
**hence** order 0  $q = 0$  **by** (simp add:  $n$ -def)  
**with**  $\langle q \neq 0 \rangle$  **have** *poly*  $q$  0  $\neq 0$  **by** (simp add: order-root)  
**ultimately show** ?thesis **using** that[of  $q$ ] **by** (auto simp: poly-0-coeff-0)  
**qed**

**lemma** algebraic-of-real-iff [simp]:

algebraic (of-real  $x :: 'a :: \{\text{real-algebra-1, field-char-0}\}$ )  $\longleftrightarrow$  algebraic  $x$

**proof**

**assume** algebraic (of-real  $x :: 'a$ )

**then obtain**  $p$  **where**  $p \neq 0$  *poly* (of-int-poly  $p$ ) (of-real  $x :: 'a$ ) = 0

**by** (erule algebraicE')

**hence** (of-int-poly  $p :: \text{real poly}$ )  $\neq 0$

*poly* (of-int-poly  $p :: \text{real poly}$ )  $x = 0$  **by** simp-all

**thus** algebraic  $x$  **by** (intro algebraicI[of of-int-poly  $p$ ]) simp-all

**next**

**assume** algebraic  $x$

**then obtain**  $p$  **where**  $p \neq 0$  *poly* (of-int-poly  $p$ )  $x = 0$  **by** (erule algebraicE')

**hence** of-int-poly  $p \neq (0 :: 'a \text{ poly})$  *poly* (of-int-poly  $p$ ) (of-real  $x :: 'a$ ) = 0

**by** simp-all

**thus** algebraic (of-real  $x$ ) **by** (intro algebraicI[of of-int-poly  $p$ ]) simp-all

**qed**

## 1.4 Main proof

**lemma** lindemann-weierstrass-integral:

**fixes**  $u :: \text{complex}$  **and**  $f :: \text{complex poly}$

**defines**  $df \equiv \lambda n. (\text{pderiv } \overset{\sim}{\sim} n) f$

**defines**  $m \equiv \text{degree } f$

**defines**  $I \equiv \lambda f u. \text{exp } u * (\sum_{j \leq \text{degree } f. \text{poly } ((\text{pderiv } \overset{\sim}{\sim} j) f) 0) -$   
 $(\sum_{j \leq \text{degree } f. \text{poly } ((\text{pderiv } \overset{\sim}{\sim} j) f) u)$

**shows**  $((\lambda t. \text{exp } (u - t) * \text{poly } f t)$  has-contour-integral  $I f u$ ) (linepath 0  $u$ )

**proof** –

**note** [derivative-intros] =

*exp-scaleR-has-vector-derivative-right vector-diff-chain-within*

**let** ? $g = \lambda t. 1 - t$  **and** ? $f = \lambda t. -\text{exp } (t *_R u)$

**have**  $((\lambda t. \text{exp } ((1 - t) *_R u) * u)$  has-integral

$(?f \circ ?g) 1 - (?f \circ ?g) 0$ ) {0..1}

**by** (rule fundamental-theorem-of-calculus)

(auto intro!: derivative-eq-intros simp del: o-apply)

**hence** aux-integral:  $((\lambda t. \text{exp } (u - t *_R u) * u)$  has-integral  $\text{exp } u - 1$ ) {0..1}

**by** (simp add: algebra-simps)

**have**  $((\lambda t. \text{exp } (u - t *_R u) * u * \text{poly } f (t *_R u))$  has-integral  $I f u$ ) {0..1}

**unfolding**  $df$ -def  $m$ -def

**proof** (induction degree  $f$  arbitrary:  $f$ )

case 0

```

then obtain c where c: f = [:c:] by (auto elim: degree-eq-zeroE)
have ((λt. c * (exp (u - t *R u) * u)) has-integral c * (exp u - 1)) {0..1}
  using aux-integral by (rule has-integral-mult-right)
with c show ?case by (simp add: algebra-simps I-def)
next
case (Suc m)
define df where df = (λj. (pderiv ~ j) f)
show ?case
proof (rule integration-by-parts[OF bounded-bilinear-mult])
  fix t :: real assume t ∈ {0..1}
  have ((?f ∘ ?g) has-vector-derivative exp (u - t *R u) * u) (at t)
    by (auto intro!: derivative-eq-intros simp: algebra-simps simp del: o-apply)
  thus ((λt. -exp (u - t *R u)) has-vector-derivative exp (u - t *R u) * u)
(at t)
    by (simp add: algebra-simps o-def)
next
  fix t :: real assume t ∈ {0..1}
  have (poly f ∘ (λt. t *R u) has-vector-derivative u * poly (pderiv f) (t *R u))
(at t)
    by (rule field-vector-diff-chain-at) (auto intro!: derivative-eq-intros)
  thus ((λt. poly f (t *R u)) has-vector-derivative u * poly (pderiv f) (t *R u))
(at t)
    by (simp add: o-def)
next
  from Suc(2) have m: m = degree (pderiv f) by (simp add: degree-pderiv)
  from Suc(1)[OF this] this
  have ((λt. exp (u - t *R u) * u * poly (pderiv f) (t *R u)) has-integral
    exp u * (∑ j=0..m. poly (df (Suc j)) 0) - (∑ j=0..m. poly (df (Suc
j)) u)) {0..1}
    by (simp add: df-def funpow-swap1 atMost-atLeast0 I-def)
  also have (∑ j=0..m. poly (df (Suc j)) 0) = (∑ j=Suc 0..Suc m. poly (df
j) 0)
    by (rule sum.shift-bounds-cl-Suc-ivl [symmetric])
  also have ... = (∑ j=0..Suc m. poly (df j) 0) - poly f 0
    by (subst (2) sum.atLeast-Suc-atMost) (simp-all add: df-def)
  also have (∑ j=0..m. poly (df (Suc j)) u) = (∑ j=Suc 0..Suc m. poly (df j
u)
    by (rule sum.shift-bounds-cl-Suc-ivl [symmetric])
  also have ... = (∑ j=0..Suc m. poly (df j) u) - poly f u
    by (subst (2) sum.atLeast-Suc-atMost) (simp-all add: df-def)
  finally have ((λt. - (exp (u - t *R u) * u * poly (pderiv f) (t *R u)))
has-integral
    - (exp u * ((∑ j = 0..Suc m. poly (df j) 0) - poly f 0) -
      ((∑ j = 0..Suc m. poly (df j) u) - poly f u)) {0..1}
    (is (- has-integral ?I) -) by (rule has-integral-neg)
  also have ?I = - exp (u - 1 *R u) * poly f (1 *R u) -
    - exp (u - 0 *R u) * poly f (0 *R u) - I f u
    by (simp add: df-def algebra-simps Suc(2) atMost-atLeast0 I-def)
  finally show ((λt. - exp (u - t *R u) * (u * poly (pderiv f) (t *R u)))

```



```

      has-integral ...) {0..1} by (simp add: algebra-simps)
    qed (auto intro!: continuous-intros)
  qed
  thus ?thesis by (simp add: has-contour-integral-linepath algebra-simps)
qed

locale lindemann-weierstrass-aux =
  fixes f :: complex poly
begin

definition I :: complex  $\Rightarrow$  complex where
  I u = exp u * ( $\sum_{j \leq \text{degree } f} \text{poly } ((\text{pderiv } \sim j) f) 0) -$ 
    ( $\sum_{j \leq \text{degree } f} \text{poly } ((\text{pderiv } \sim j) f) u$ )

lemma lindemann-weierstrass-integral-bound:
  fixes u :: complex
  assumes C  $\geq$  0  $\wedge$  t  $\in$  closed-segment 0 u  $\implies$  norm (poly f t)  $\leq$  C
  shows norm (I u)  $\leq$  norm u * exp (norm u) * C
proof -
  have I u = contour-integral (linepath 0 u) ( $\lambda t. \text{exp } (u - t) * \text{poly } f t$ )
    using contour-integral-unique[OF lindemann-weierstrass-integral[of u f]] unfolding I-def ..
  also have norm ...  $\leq$  exp (norm u) * C * norm (u - 0)
  proof (intro contour-integral-bound-linepath)
    fix t assume t: t  $\in$  closed-segment 0 u
    then obtain s where s: s  $\in$  {0..1} t = s *R u by (auto simp: closed-segment-def)
    hence s * norm u  $\leq$  1 * norm u by (intro mult-right-mono) simp-all
    with s have norm-t: norm t  $\leq$  norm u by auto

    from s have Re u - Re t = (1 - s) * Re u by (simp add: algebra-simps)
    also have ...  $\leq$  norm u
    proof (cases Re u  $\geq$  0)
      case True
        with  $\langle s \in \{0..1\} \rangle$  have (1 - s) * Re u  $\leq$  1 * Re u by (intro mult-right-mono)
    simp-all
        also have Re u  $\leq$  norm u by (rule complex-Re-le-cmod)
        finally show ?thesis by simp
      next
      case False
        with  $\langle s \in \{0..1\} \rangle$  have (1 - s) * Re u  $\leq$  0 by (intro mult-nonneg-nonpos)
    simp-all
        also have ...  $\leq$  norm u by simp
        finally show ?thesis .
    qed
  finally have exp (Re u - Re t)  $\leq$  exp (norm u) by simp

  hence exp (Re u - Re t) * norm (poly f t)  $\leq$  exp (norm u) * C
    using assms t norm-t by (intro mult-mono) simp-all
  thus norm (exp (u - t) * poly f t)  $\leq$  exp (norm u) * C

```

by (simp add: norm-mult exp-diff norm-divide field-simps)  
 qed (auto simp: intro!: mult-nonneg-nonneg contour-integrable-continuous-linepath  
 continuous-intros assms)  
 finally show ?thesis by (simp add: mult-ac)  
 qed  
 end

**lemma poly-higher-pderiv-aux1:**  
 fixes  $c :: 'a :: idom$   
 assumes  $k < n$   
 shows poly ((pderiv  $\overset{\sim}{\sim}$  k) ([:- c, 1:]  $\wedge$  n \* p)) c = 0  
 using assms  
**proof** (induction k arbitrary: n p)  
 case (Suc k n p)  
 from Suc.prem1 obtain n' where n: n = Suc n' by (cases n) auto  
 from Suc.prem2 have k < n' by simp  
 have (pderiv  $\overset{\sim}{\sim}$  Suc k) ([:- c, 1:]  $\wedge$  n \* p) =  
 (pderiv  $\overset{\sim}{\sim}$  k) ([:- c, 1:]  $\wedge$  n \* pderiv p + [- c, 1:]  $\wedge$  n' \* smult (of-nat  
 n) p)  
 by (simp only: funpow-Suc-right o-def pderiv-mult n pderiv-power-Suc,  
 simp only: n [symmetric]) (simp add: pderiv-pCons mult-ac)  
 also from Suc.prem3 have poly ... c = 0  
 by (simp add: higher-pderiv-add Suc.IH del: mult-smult-right)  
 finally show ?case .  
 qed simp-all

**lemma poly-higher-pderiv-aux1':**  
 fixes  $c :: 'a :: idom$   
 assumes  $k < n$  [- c, 1:]  $\wedge$  n dvd p  
 shows poly ((pderiv  $\overset{\sim}{\sim}$  k) p) c = 0  
**proof** -  
 from assms(2) obtain q where p = [- c, 1:]  $\wedge$  n \* q by (elim dvdE)  
 also from assms(1) have poly ((pderiv  $\overset{\sim}{\sim}$  k) ...) c = 0  
 by (rule poly-higher-pderiv-aux1)  
 finally show ?thesis .  
 qed

**lemma poly-higher-pderiv-aux2:**  
 fixes  $c :: 'a :: \{idom, semiring-char-0\}$   
 shows poly ((pderiv  $\overset{\sim}{\sim}$  n) ([:- c, 1:]  $\wedge$  n \* p)) c = fact n \* poly p c  
**proof** (induction n arbitrary: p)  
 case (Suc n p)  
 have (pderiv  $\overset{\sim}{\sim}$  Suc n) ([:- c, 1:]  $\wedge$  Suc n \* p) =  
 (pderiv  $\overset{\sim}{\sim}$  n) ([:- c, 1:]  $\wedge$  Suc n \* pderiv p) +  
 (pderiv  $\overset{\sim}{\sim}$  n) ([:- c, 1:]  $\wedge$  n \* smult (1 + of-nat n) p)  
 by (simp del: funpow.simps power-Suc add: funpow-Suc-right pderiv-mult  
 pderiv-power-Suc higher-pderiv-add pderiv-pCons mult-ac)  
 also have [- c, 1:]  $\wedge$  Suc n \* pderiv p = [- c, 1:]  $\wedge$  n \* ([:- c, 1:] \* pderiv p)

by (simp add: algebra-simps)  
 finally show ?case by (simp add: Suc.IH del: mult-smult-right power-Suc)  
 qed simp-all

**lemma** poly-higher-pderiv-aux3:

fixes  $c :: 'a :: \{idom, semiring-char-0\}$   
 assumes  $k \geq n$   
 shows  $\exists q. \text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n * p)) c = \text{fact } n * \text{poly } q c$   
 using *assms*  
**proof** (induction k arbitrary: n p)  
 case (Suc k n p)  
 show ?case  
**proof** (cases n)  
 fix  $n'$  assume  $n: n = \text{Suc } n'$   
 have  $\text{poly } ((pderiv \widehat{\sim} \text{Suc } k) ([: -c, 1:] \widehat{\sim} n * p)) c =$   
 $\text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n * pderiv p)) c +$   
 $\text{of-nat } n * \text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n' * p)) c$   
 by (simp del: funpow.simps power-Suc add: funpow-Suc-right pderiv-power-Suc  
 pderiv-mult n pderiv-pCons higher-pderiv-add mult-ac higher-pderiv-smult)  
 also have  $\exists q1. \text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n * pderiv p)) c = \text{fact } n * \text{poly } q1 c$   
 using *Suc.prem* *Suc.IH*[of n pderiv p]  
 by (cases  $n' = k$ ) (auto simp: n poly-higher-pderiv-aux1 simp del: power-Suc  
 of-nat-Suc  
 intro: *exI*[of - 0::'a poly])  
 then obtain  $q1$   
 where  $\text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n * pderiv p)) c = \text{fact } n * \text{poly } q1 c ..$   
 also from *Suc.IH*[of  $n'$  p] *Suc.prem* obtain  $q2$   
 where  $\text{poly } ((pderiv \widehat{\sim} k) ([: -c, 1:] \widehat{\sim} n' * p)) c = \text{fact } n' * \text{poly } q2 c$   
 by (auto simp: n)  
 finally show ?case by (auto intro!: *exI*[of -  $q1 + q2$ ] simp: n algebra-simps)  
 qed auto  
 qed auto

**lemma** poly-higher-pderiv-aux3':

fixes  $c :: 'a :: \{idom, semiring-char-0\}$   
 assumes  $k \geq n$   $[: -c, 1:] \widehat{\sim} n \text{ dvd } p$   
 shows  $\text{fact } n \text{ dvd } \text{poly } ((pderiv \widehat{\sim} k) p) c$   
**proof** -  
 from *assms*(2) obtain  $q$  where  $p = [: -c, 1:] \widehat{\sim} n * q$  by (*elim dvdE*)  
 with *poly-higher-pderiv-aux3*[OF *assms*(1), of  $c$   $q$ ] show ?thesis by auto  
 qed

**lemma** e-transcendental-aux-bound:

obtains  $C$  where  $C \geq 0$   
 $\bigwedge x. x \in \text{closed-segment } 0 \text{ (of-nat } n) \implies$   
 $\text{norm } (\prod_{k \in \{1..n\}}. (x - \text{of-nat } k :: \text{complex})) \leq C$   
**proof** -  
 let  $?f = \lambda x. (\prod_{k \in \{1..n\}}. (x - \text{of-nat } k))$

```

define  $C$  where  $C = \max 0 (Sup (cmod \text{ ' } ?f \text{ ' } closed\text{-segment } 0 (of\text{-nat } n)))$ 
have  $C \geq 0$  by (simp add: C-def)
moreover {
  fix  $x :: complex$  assume  $x \in closed\text{-segment } 0 (of\text{-nat } n)$ 
  hence  $cmod ( ?f x) \leq Sup ((cmod \circ ?f) \text{ ' } closed\text{-segment } 0 (of\text{-nat } n))$ 
  by (intro cSup-upper bounded-imp-bdd-above compact-imp-bounded compact-continuous-image)
    (auto intro!: continuous-intros)
  also have  $\dots \leq C$  by (simp add: C-def image-comp)
  finally have  $cmod ( ?f x) \leq C$  .
}
ultimately show ?thesis by (rule that)
qed

```

**theorem** *e-transcendental-complex*:  $\neg algebraic (exp\ 1 :: complex)$

**proof**

**assume** *algebraic* ( $exp\ 1 :: complex$ )

**then obtain**  $q :: int\ poly$

**where**  $q: q \neq 0\ coeff\ q\ 0 \neq 0\ poly (of\text{-int}\text{-poly } q) (exp\ 1 :: complex) = 0$

**by** (elim algebraicE'-nonzero) simp-all

**define**  $n :: nat$  **where**  $n = degree\ q$

**from**  $q$  **have** [simp]:  $n \neq 0$  **by** (intro notI) (auto simp: n-def elim!: degree-eq-zeroE)

**define**  $qmax$  **where**  $qmax = Max (insert\ 0 (abs \text{ ' } set (coeffs\ q)))$

**have**  $qmax\text{-nonneg}$  [simp]:  $qmax \geq 0$  **by** (simp add: qmax-def)

**have**  $qmax$ :  $|coeff\ q\ k| \leq qmax$  **for**  $k$

**by** (cases  $k \leq degree\ q$ )

(auto simp: qmax-def coeff-eq-0 coeffs-def simp del: upt-Suc intro: Max.coboundedI)

**obtain**  $C$  **where**  $C: C \geq 0$

$\bigwedge x. x \in closed\text{-segment } 0 (of\text{-nat } n) \implies norm (\prod_{k \in \{1..n\}} (x - of\text{-nat } k :: complex)) \leq C$

**by** (erule *e-transcendental-aux-bound*)

**define**  $E$  **where**  $E = (1 + real\ n) * real\text{-of}\text{-int } qmax * real\ n * exp (real\ n) / real\ n$

**define**  $F$  **where**  $F = real\ n * C$

**have** *ineq*:  $fact (p - 1) \leq E * F ^ p$  **if**  $p$ : *prime*  $p\ p > n\ p > abs (coeff\ q\ 0)$  **for**  $p$

**proof** –

**from**  $p(1)$  **have**  $p\text{-pos}$ :  $p > 0$  **by** (simp add: prime-gt-0-nat)

**define**  $f :: int\ poly$

**where**  $f = monom\ 1 (p - 1) * (\prod_{k \in \{1..n\}} [:-of\text{-nat } k, 1:] ^ p)$

**have**  $poly\text{-f}$ :  $poly (of\text{-int}\text{-poly } f) x = x ^ (p - 1) * (\prod_{k \in \{1..n\}} (x - of\text{-nat } k)) ^ p$

**for**  $x :: complex$  **by** (simp add: f-def poly-prod poly-monom prod-power-distrib)

**define**  $m :: nat$  **where**  $m = degree\ f$

**from**  $p\text{-pos}$  **have**  $m$ :  $m = (n + 1) * p - 1$

**by** (simp add: m-def f-def degree-mult-eq degree-monom-eq degree-prod-sum-eq degree-linear-power)

**define**  $M :: \text{int}$  **where**  $M = (-1) \wedge (n * p) * \text{fact } n \wedge p$   
**with**  $p$  **have**  $p\text{-not-dvd-}M: \neg \text{int } p \text{ dvd } M$   
**by** (*auto simp: M-def prime-elem-int-not-dvd-neg1-power prime-dvd-power-iff prime-gt-0-nat prime-dvd-fact-iff-int prime-dvd-mult-iff*)

**interpret** *lindemann-weierstrass-aux of-int-poly f* .  
**define**  $J :: \text{complex}$  **where**  $J = (\sum k \leq n. \text{of-int } (\text{coeff } q \ k) * I \ (\text{of-nat } k))$   
**define**  $\text{idxs}$  **where**  $\text{idxs} = (\{..n\} \times \{..m\}) - \{(0, p - 1)\}$

**hence**  $J = (\sum k \leq n. \text{of-int } (\text{coeff } q \ k) * \text{exp } 1 \wedge k) * (\sum n \leq m. \text{of-int } (\text{poly } ((\text{pderiv } \wedge n) \ f) \ 0)) -$   
 $\text{of-int } (\sum k \leq n. \sum n \leq m. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{int } k))$   
**by** (*simp add: J-def I-def algebra-simps sum-subtractf sum-distrib-left m-def exp-of-nat-mult [symmetric]*)

**also have**  $(\sum k \leq n. \text{of-int } (\text{coeff } q \ k) * \text{exp } 1 \wedge k) = \text{poly } (\text{of-int-poly } q) \ (\text{exp } 1$   
 $:: \text{complex})$   
**by** (*simp add: poly-altdef n-def*)  
**also have**  $\dots = 0$  **by fact**  
**finally have**  $J = \text{of-int } (-\sum (k,n) \in \{..n\} \times \{..m\}. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{int } k))$   
**by** (*simp add: sum.cartesian-product*)

**also have**  $\{..n\} \times \{..m\} = \text{insert } (0, p - 1) \ \text{idxs}$  **by** (*auto simp: m idxs-def*)  
**also have**  $-\sum (k,n) \in \dots. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{int } k) =$   
 $-(\text{coeff } q \ 0 * \text{poly } ((\text{pderiv } \wedge (p - 1)) \ f) \ 0) -$   
 $(\sum (k, n) \in \text{idxs}. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{of-nat } k))$   
**by** (*subst sum.insert*) (*simp-all add: idxs-def*)

**also have**  $\text{coeff } q \ 0 * \text{poly } ((\text{pderiv } \wedge (p - 1)) \ f) \ 0 = \text{coeff } q \ 0 * M * \text{fact } (p$   
 $- 1)$   
**proof** -  
**have**  $f = [:-0, 1:] \wedge (p - 1) * (\prod k = 1..n. [:- \text{of-nat } k, 1:] \wedge p)$   
**by** (*simp add: f-def monom-altdef*)  
**also have**  $\text{poly } ((\text{pderiv } \wedge (p - 1)) \ \dots) \ 0 =$   
 $\text{fact } (p - 1) * \text{poly } (\prod k = 1..n. [:- \text{of-nat } k, 1:] \wedge p) \ 0$   
**by** (*rule poly-higher-pderiv-aux2*)  
**also have**  $\text{poly } (\prod k = 1..n. [:- \text{of-nat } k :: \text{int}, 1:] \wedge p) \ 0 = (-1) \wedge (n * p) *$   
 $\text{fact } n \wedge p$   
**by** (*induction n*) (*simp-all add: prod.nat-ivl-Suc' power-mult-distrib mult-ac power-minus' power-add del: of-nat-Suc*)

**finally show** *?thesis* **by** (*simp add: mult-ac M-def*)  
**qed**  
**also obtain**  $N$  **where**  $(\sum (k, n) \in \text{idxs}. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{int } k)) = \text{fact } p * N$   
**proof** -  
**have**  $\forall (k, n) \in \text{idxs}. \text{fact } p \ \text{dvd} \ \text{poly } ((\text{pderiv } \wedge n) \ f) \ (\text{of-nat } k)$   
**proof clarify**  
**fix**  $k \ j$  **assume**  $\text{idxs}: (k, j) \in \text{idxs}$   
**then consider**  $k = 0 \ j < p - 1 \mid k = 0 \ j > p - 1 \mid k \neq 0 \ j < p \mid k \neq 0 \ j$   
 $\geq p$

```

    by (fastforce simp: idxs-def)
  thus fact p dvd poly ((pderiv  $\sim$  j) f) (of-nat k)
proof cases
  case 1
  thus ?thesis
    by (simp add: f-def poly-higher-pderiv-aux1' monom-altdef)
  next
  case 2
  thus ?thesis
  by (simp add: f-def poly-higher-pderiv-aux3' monom-altdef fact-dvd-poly-higher-pderiv-aux')
  next
  case 3
  thus ?thesis unfolding f-def
    by (subst poly-higher-pderiv-aux1 '[of - p])
      (insert idxs, auto simp: idxs-def intro!: dvd-mult)
  next
  case 4
  thus ?thesis unfolding f-def
    by (intro poly-higher-pderiv-aux3') (insert idxs, auto intro!: dvd-mult
simp: idxs-def)
  qed
  qed
  hence fact p dvd ( $\sum (k, n) \in \text{idxs}. \text{coeff } q \ k * \text{poly } ((\text{pderiv } \sim n) f) (\text{int } k)$ )
    by (auto intro!: dvd-sum dvd-mult simp del: of-int-fact)
  with that show thesis
    by blast
  qed
  also from p have  $-(\text{coeff } q \ 0 * M * \text{fact } (p - 1)) - \text{fact } p * N =$ 
     $-\text{fact } (p - 1) * (\text{coeff } q \ 0 * M + p * N)$ 
    by (subst fact-reduce[of p]) (simp-all add: algebra-simps)
  finally have J:  $J = -\text{of-int } (\text{fact } (p - 1) * (\text{coeff } q \ 0 * M + p * N))$  by simp

  from p q(2) have  $\neg p \text{ dvd } \text{coeff } q \ 0 * M + p * N$ 
  by (auto simp: dvd-add-left-iff p-not-dvd-M prime-dvd-fact-iff-int prime-dvd-mult-iff
dest: dvd-imp-le-int)
  hence  $\text{coeff } q \ 0 * M + p * N \neq 0$  by (intro notI) simp-all
  hence  $\text{abs } (\text{coeff } q \ 0 * M + p * N) \geq 1$  by simp
  hence  $\text{norm } (\text{of-int } (\text{coeff } q \ 0 * M + p * N) :: \text{complex}) \geq 1$  by (simp only:
norm-of-int)
  hence  $\text{fact } (p - 1) * \dots \geq \text{fact } (p - 1) * 1$  by (intro mult-left-mono) simp-all
  hence J-lower:  $\text{norm } J \geq \text{fact } (p - 1)$  unfolding J norm-minus-cancel
of-int-mult of-int-fact
  by (simp add: norm-mult)

  have  $\text{norm } J \leq (\sum k \leq n. \text{norm } (\text{of-int } (\text{coeff } q \ k) * I (\text{of-nat } k)))$ 
  unfolding J-def by (rule norm-sum)
  also have  $\dots \leq (\sum k \leq n. \text{of-int } q_{\max} * (\text{real } n * \text{exp } (\text{real } n) * \text{real } n \wedge (p -$ 
1) *  $C \wedge p))$ 
  proof (intro sum-mono)

```

```

fix  $k$  assume  $k: k \in \{..n\}$ 
have  $n > 0$  by (rule ccontr) simp
{
  fix  $x :: \text{complex}$  assume  $x: x \in \text{closed-segment } 0 \text{ (of-nat } k)$ 
  then obtain  $t$  where  $t: t \geq 0 \ t \leq 1 \ x = \text{of-real } t * \text{of-nat } k$ 
    by (auto simp: closed-segment-def scaleR-conv-of-real)
  hence norm  $x = t * \text{real } k$  by (simp add: norm-mult)
  also from  $\langle t \leq 1 \rangle k$  have  $*$ :  $\dots \leq 1 * \text{real } n$  by (intro mult-mono) simp-all
  finally have  $x'$ :  $\text{norm } x \leq \text{real } n$  by simp
  from  $t \langle n > 0 \rangle *$  have  $x''$ :  $x \in \text{closed-segment } 0 \text{ (of-nat } n)$ 
    by (auto simp: closed-segment-def scaleR-conv-of-real field-simps
      intro!: exI[of -  $t * \text{real } k / \text{real } n$ ] )
  have norm (poly (of-int-poly f) x) =
    norm  $x \wedge^{(p-1)} * \text{cmod} (\prod_{i=1..n} x - i) \wedge^p$ 
    by (simp add: poly-f norm-mult norm-power)
  also from  $x \ x' \ x''$  have  $\dots \leq \text{of-nat } n \wedge^{(p-1)} * C \wedge^p$ 
    by (intro mult-mono C power-mono) simp-all
  finally have norm (poly (of-int-poly f) x)  $\leq \text{real } n \wedge^{(p-1)} * C \wedge^p$  .
} note  $A = \text{this}$ 

have norm (I (of-nat k))  $\leq$ 
  cmod (of-nat k) * exp (cmod (of-nat k)) * (of-nat  $n \wedge^{(p-1)}$ ) *
 $C \wedge^p$ )
  by (intro lindemann-weierstrass-integral-bound[OF - A]
    C mult-nonneg-nonneg zero-le-power) auto
also have  $\dots \leq \text{cmod} (\text{of-nat } n) * \text{exp} (\text{cmod} (\text{of-nat } n)) * (\text{of-nat } n \wedge^{(p-1)}) * C \wedge^p$ 
  using  $k$  by (intro mult-mono zero-le-power mult-nonneg-nonneg C) simp-all
finally show cmod (of-int (coeff q k) * I (of-nat k))  $\leq$ 
  of-int qmax * (real n * exp (real n) * real  $n \wedge^{(p-1)}$ ) *  $C \wedge^p$ )
  unfolding norm-mult
  by (intro mult-mono) (simp-all add: qmax of-int-abs [symmetric] del:
of-int-abs)
qed
also have  $\dots = E * F \wedge^p$  using p-pos
  by (simp add: power-diff power-mult-distrib E-def F-def)
finally show fact (p - 1)  $\leq E * F \wedge^p$  using J-lower by linarith
qed

have ( $\lambda n. E * F * F \wedge^{(n-1)} / \text{fact } (n-1)$ )  $\longrightarrow 0$  (is ?P)
  by (intro filterlim-compose[OF power-over-fact-tendsto-0' filterlim-minus-nat-at-top])
also have ?P  $\longleftrightarrow (\lambda n. E * F \wedge^n / \text{fact } (n-1)) \longrightarrow 0$ 
  by (intro filterlim-cong refl eventually-mono[OF eventually-gt-at-top[of 0::nat]])
  (auto simp: power-Suc [symmetric] simp del: power-Suc)
finally have eventually ( $\lambda n. E * F \wedge^n / \text{fact } (n-1) < 1$ ) at-top
  by (rule order-tendstoD) simp-all
hence eventually ( $\lambda n. E * F \wedge^n < \text{fact } (n-1)$ ) at-top by eventually-elim simp
then obtain  $P$  where  $P: \bigwedge n. n \geq P \implies E * F \wedge^n < \text{fact } (n-1)$ 
  by (auto simp: eventually-at-top-linorder)

```

**have**  $\exists p. \text{prime } p \wedge p > \text{Max } \{ \text{nat } (\text{abs } (\text{coeff } q \ 0)), n, P \}$  **by** (rule bigger-prime)  
**then obtain**  $p$  **where**  $\text{prime } p \wedge p > \text{Max } \{ \text{nat } (\text{abs } (\text{coeff } q \ 0)), n, P \}$  **by** blast  
**hence**  $\text{int } p > \text{abs } (\text{coeff } q \ 0) \wedge p > n \wedge p \geq P$  **by** auto  
**with**  $\text{ineq}[of \ p] \ \langle \text{prime } p \rangle$  **have**  $\text{fact } (p - 1) \leq E * F ^ p$  **by** simp  
**moreover from**  $\langle p \geq P \rangle$  **have**  $\text{fact } (p - 1) > E * F ^ p$  **by** (rule P)  
**ultimately show** False **by** linarith  
**qed**

**corollary** *e-transcendental-real*:  $\neg \text{algebraic } (\text{exp } 1 :: \text{real})$

**proof** –

**have**  $\neg \text{algebraic } (\text{exp } 1 :: \text{complex})$  **by** (rule e-transcendental-complex)

**also have**  $(\text{exp } 1 :: \text{complex}) = \text{of-real } (\text{exp } 1)$  **using** exp-of-real[of 1] **by** simp

**also have**  $\text{algebraic } \dots \longleftrightarrow \text{algebraic } (\text{exp } 1 :: \text{real})$  **by** simp

**finally show** ?thesis .

**qed**

**end**

## References

- [1] R. Lipsett. Planetmath. <http://planetmath.org/prooffindemannweierstrasstheoremandthateandpiaretranscendental>, 2007.