The Transcendence of e

Manuel Eberl

August 16, 2018

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

1 Proof of the Transcendence of e

theory *E-Transcendental*
imports
HOL−Analysis.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 *fact m dvd pochhammer* (*int n − int m + 1*) *m*
proof −
have (*real n gchoose m*) ∗ *fact m* = *of-int* (*pochhammer* (*int n − int m + 1*) *m*)
by (*simp add*: *gbinomial-pochhammer′ pochhammer-of-int* [*symmetric*])
also have (*real n gchoose m*) ∗ *fact m* = *of-int* (*int* (*n choose m*) ∗ *fact m*)
by (*simp add*: *binomial-gbinomial*)

finally have *int* (*n choose m*) ∗ *fact m* = *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* :: *semiring-char-0*) ⟷ *x* = *1*
by (*fact of-nat-eq-1-iff*)

lemma *prime-elem-int-not-dvd-neg1-power*:
prime-elem (*p* :: *int*) ⟹ ¬*p dvd* (−*1*) ^ *n*
by (*rule notI*, *frule* (*1*) *prime-elem-dvd-power*, *cases p* ≥ *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 dvd fact n ⟷ *p* ≤ *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 (λ*n*. *n* − *k* :: *nat*) *at-top at-top*
proof −
have *sequentially* = *filtermap* (λ*n*. *n* + *k*) *at-top*
by (*auto simp*: *filter-eq-iff eventually-filtermap*)
also have *filterlim* (λ*n*. *n* − *k* :: *nat*) *at-top* ...
by (*simp add*: *filterlim-filtermap filterlim-ident*)
finally show *?thesis* **.**
qed

lemma *power-over-fact-tendsto-0*:
(λ*n*. (*x* :: *real*) ^ *n* / *fact n*) ⟶ *0*
using *summable-exp*[*of x*] **by** (*intro summable-LIMSEQ-zero*) (*simp add*: *sums-iff field-simps*)

lemma *power-over-fact-tendsto-0'*:
(λ*n*. *c* ∗ (*x* :: *real*) ^ *n* / *fact n*) ⟶ *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* ⇒ *'a* :: *comm-ring-1 poly* **is** λ*g x*. *of-int* (*g x*)
by (*auto elim*: *eventually-mono*)

lemma *coeff-of-int-poly* [*simp*]: *coeff* (*of-int-poly p*) *n* = *of-int* (*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* (λ*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* (λ*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 ^ n*) = *of-int-poly p ^ 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 = (*of-int-poly q* :: *'a* :: {*comm-ring-1*, *ring-char-0*} *poly*) ⟷ *p* = *q*
by (*simp add*: *poly-eq-iff*)

lemma *of-int-poly-eq-0-iff* [*simp*]:
of-int-poly p = (*0* :: *'a* :: {*comm-ring-1*, *ring-char-0*} *poly*) ⟷ *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* :: {*comm-ring-1*, *ring-char-0*} *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* ^^ *n*) (*of-int-poly p*) = *of-int-poly* ((*pderiv* ^^ *n*) *p*)
by (*induction n*) *simp-all*

lemma *int-polyE*:
assumes ⋀*n*. *coeff* (*p* :: *'a* :: {*comm-ring-1*, *ring-char-0*} *poly*) *n* ∈ ℤ
obtains *p'* **where** *p* = *of-int-poly p'*
proof −
from *assms* **have** ∀ *n*. ∃ *c*. *coeff p n* = *of-int c* **by** (*auto simp*: *Ints-def*)
hence ∃ *c*. ∀ *n*. *of-int* (*c n*) = *coeff p n* **by** (*simp add*: *choice-iff eq-commute*)
then obtain *c* **where** *c*: *of-int* (*c n*) = *coeff p n* **for** *n* **by** *blast*
have [*simp*]: *coeff* (*Abs-poly c*) = *c*
proof (*rule poly.Abs-poly-inverse*, *clarify*)
have *eventually* (λ*n*. *n* > *degree p*) *at-top* **by** (*rule eventually-gt-at-top*)
hence *eventually* (λ*n*. *coeff p n* = *0*) *at-top*
by *eventually-elim* (*simp add*: *coeff-eq-0*)
thus *eventually* (λ*n*. *c n* = *0*) *cofinite*
by (*simp add*: *c* [*symmetric*] *cofinite-eq-sequentially*)
qed
have *p* = *of-int-poly* (*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* ^ *n*) = *smult* (*of-nat n*) (*p* ^ (*n* − *1*) ∗ *pderiv p*)
by (*cases n*) (*simp-all add*: *pderiv-power-Suc del*: *power-Suc*)

lemma *degree-prod-sum-eq*:

$(\bigwedge x.\ x \in A \Longrightarrow f\ x \neq 0) \Longrightarrow$
$degree\ (prod\ f\ A :: 'a :: idom\ poly) = (\sum x{\in}A.\ degree\ (f\ x))$
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*:] ^ *n* = [:*c* ^ *n*:]
by (*induction n*) (*simp-all add*: *power-commutes*)

lemma *monom-power*: *monom c n* ^ *k* = *monom* (*c* ^ *k*) (*n* ∗ *k*)
by (*induction k*) (*simp-all add*: *mult-monom*)

lemma *coeff-higher-pderiv*:
coeff ((*pderiv* ^^ *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 algebra-simps*)

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

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

lemma *higher-pderiv-0* [*simp*]: (*pderiv* ^^ *n*) *0* = *0*
by (*induction n*) *simp-all*

lemma *higher-pderiv-monom*:
m ≤ *n* + *1* ⟹ (*pderiv* ^^ *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* ⟹ (*pderiv* ^^ *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* ^^ *n*) (*sum f A*) = (∑ *x*∈*A*. (*pderiv* ^^ *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* ^^ *n*) *p*
proof −
have [:*fact n*:] *dvd* (*pderiv* ^^ *n*) (*monom c k*) **for** *c* :: *int* **and** *k* :: *nat*
by (*cases n* ≤ *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* ^^ *n*) (∑ *k*≤*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-monoms*)
qed

lemma *fact-dvd-poly-higher-pderiv-aux*:
(*fact n* :: *int*) *dvd poly* ((*pderiv* ^^ *n*) *p*) *x*
proof −
have [:*fact n*:] *dvd* (*pderiv* ^^ *n*) *p* **by** (*rule fact-dvd-higher-pderiv*)
then obtain *q* **where** (*pderiv* ^^ *n*) *p* = [:*fact n*:] ∗ *q* **by** (*erule dvdE*)
thus *?thesis* **by** *simp*
qed

lemma *fact-dvd-poly-higher-pderiv-aux'*:
m ≤ *n* ⟹ (*fact m* :: *int*) *dvd poly* ((*pderiv* ^^ *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* ≠ *0 poly* (*of-int-poly p*) *x* = *0*
proof −
from *assms* **obtain** *q* **where** ⋀*i*. *coeff q i* ∈ ℤ *q* ≠ *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* ≠ *0*
obtains *p* **where** *p* ≠ *0 coeff p 0* ≠ *0 poly* (*of-int-poly p*) *x* = *0*
proof −
from *assms*(*1*) **obtain** *p* **where** *p*: *p* ≠ *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*: *p* = *monom 1 n* ∗ *q* **by** (*erule dvdE*)
from *p* **have** *q* ≠ *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 ‹*q* ≠ *0*› **have** *poly q 0* ≠ *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* :: {*real-algebra-1*,*field-char-0*}) ⟷ *algebraic x*
proof
assume *algebraic* (*of-real x* :: *'a*)
then obtain *p* **where** *p* ≠ *0 poly* (*of-int-poly p*) (*of-real x* :: *'a*) = *0*
by (*erule algebraicE'*)
hence (*of-int-poly p* :: *real poly*) ≠ *0*
poly (*of-int-poly p* :: *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* ≠ *0 poly* (*of-int-poly p*) *x* = *0* **by** (*erule algebraicE'*)
hence *of-int-poly p* ≠ (*0* :: *'a 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* :: *complex* **and** *f* :: *complex poly*
defines *df* ≡ λ*n*. (*pderiv* ^^ *n*) *f*
defines *m* ≡ *degree f*
defines *I* ≡ λ*f u*. *exp u* ∗ (∑ *j*≤*degree f*. *poly* ((*pderiv* ^^ *j*) *f*) *0*) −
(∑ *j*≤*degree f*. *poly* ((*pderiv* ^^ *j*) *f*) *u*)
shows ((λ*t*. *exp* (*u* − *t*) ∗ *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* = λ*t*. *1* − *t* **and** *?f* = λ*t*. −*exp* (*t* ∗*R u*)
have ((λ*t*. *exp* ((*1* − *t*) ∗*R u*) ∗ *u*) *has-integral*
(*?f* ∘ *?g*) *1* − (*?f* ∘ *?g*) *0*) {*0..1*}
by (*rule fundamental-theorem-of-calculus*)
(*auto intro*!: *derivative-eq-intros simp del*: *o-apply*)
hence *aux-integral*: ((λ*t*. *exp* (*u* − *t* ∗*R u*) ∗ *u*) *has-integral exp u* − *1*) {*0..1*}
by (*simp add*: *algebra-simps*)

have ((λ*t*. *exp* (*u* − *t* ∗*R u*) ∗ *u* ∗ *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-head-Suc) (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-head-Suc) (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 ⇒ complex where
 I u = exp u * (∑j≤degree f. poly ((pderiv ^^ j) f) 0) -
                (∑j≤degree f. poly ((pderiv ^^ j) f) u)

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

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

   hence exp (Re u - Re t) * norm (poly f t) ≤ exp (norm u) * C
```

```
    using assms t norm-t by (intro mult-mono) simp-all
  thus norm (exp (u − t) ∗ poly f t) ≤ 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 ^^ k) ([:−c, 1:] ^ n ∗ p)) c = 0
  using assms
proof (induction k arbitrary: n p)
  case (Suc k n p)
  from Suc.prems obtain n' where n: n = Suc n' by (cases n) auto
  from Suc.prems n have k < n' by simp
  have (pderiv ^^ Suc k) ([:− c, 1:] ^ n ∗ p) =
          (pderiv ^^ k) ([:− c, 1:] ^ n ∗ pderiv p + [:− c, 1:] ^ 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.prems ‹k < n'› 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:] ^ n dvd p
  shows   poly ((pderiv ^^ k) p) c = 0
proof −
  from assms(2) obtain q where p = [:−c, 1:] ^ n ∗ q by (elim dvdE)
  also from assms(1) have poly ((pderiv ^^ 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 ^^ n) ([:−c, 1:] ^ n ∗ p)) c = fact n ∗ poly p c
proof (induction n arbitrary: p)
  case (Suc n p)
  have (pderiv ^^ Suc n) ([:− c, 1:] ^ Suc n ∗ p) =
          (pderiv ^^ n) ([:− c, 1:] ^ Suc n ∗ pderiv p) +
            (pderiv ^^ n) ([:− c, 1:] ^ 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*:] ^ *Suc n* ∗ *pderiv p* = [:− *c*, *1*:] ^ *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* ≥ *n*
shows ∃ *q*. *poly* ((*pderiv* ^^ *k*) ([:−*c*, *1*:] ^ *n* ∗ *p*)) *c* = *fact n* ∗ *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* = *Suc n'*
have *poly* ((*pderiv* ^^ *Suc k*) ([:−*c*, *1*:] ^ *n* ∗ *p*)) *c* =
poly ((*pderiv* ^^ *k*) ([:− *c*, *1*:] ^ *n* ∗ *pderiv p*)) *c* +
of-nat n ∗ *poly* ((*pderiv* ^^ *k*) ([:−*c*, *1*:] ^ *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 ∃ *q1*. *poly* ((*pderiv* ^^ *k*) ([:−*c*, *1*:] ^ *n* ∗ *pderiv p*)) *c* = *fact n* ∗ *poly q1 c*
using *Suc.prems 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 *poly* ((*pderiv* ^^ *k*) ([:−*c*, *1*:] ^ *n* ∗ *pderiv p*)) *c* = *fact n* ∗ *poly q1 c* **..**
also from *Suc.IH*[*of n' p*] *Suc.prems* **obtain** *q2*
where *poly* ((*pderiv* ^^ *k*) ([:−*c*, *1*:] ^ *n'* ∗ *p*)) *c* = *fact n'* ∗ *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* ≥ *n* [:−*c*, *1*:] ^ *n dvd p*
shows *fact n dvd poly* ((*pderiv* ^^ *k*) *p*) *c*
proof −
from *assms*(*2*) **obtain** *q* **where** *p* = [:−*c*, *1*:] ^ *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* ≥ *0*
⋀*x*. *x* ∈ *closed-segment 0* (*of-nat n*) ⟹
norm (∏ *k*∈{*1*..*n*}. (*x* − *of-nat k* :: *complex*)) ≤ *C*

proof −
let *?f* = λ*x*. (∏ *k*∈{*1*..*n*}. (*x* − *of-nat k*))
define *C* **where** *C* = *max 0* (*Sup* (*cmod* ‘ *?f* ‘ *closed-segment 0* (*of-nat n*)))
have *C* ≥ *0* **by** (*simp add*: *C-def*)
moreover **{**
fix *x* :: *complex* **assume** *x* ∈ *closed-segment 0* (*of-nat n*)
hence *cmod* (*?f x*) ≤ *Sup* ((*cmod* ∘ *?f*) ‘ *closed-segment 0* (*of-nat n*))
by (*intro cSup-upper bounded-imp-bdd-above compact-imp-bounded compact-continuous-image*)
(*auto intro*!: *continuous-intros*)
also have ... ≤ *C* **by** (*simp add*: *C-def*)
finally have *cmod* (*?f x*) ≤ *C* **.**
}
ultimately show *?thesis* **by** (*rule that*)
qed

theorem *e-transcendental-complex*: ¬ *algebraic* (*exp 1* :: *complex*)
proof
assume *algebraic* (*exp 1* :: *complex*)
then obtain *q* :: *int poly*
where *q*: *q* ≠ *0* *coeff q 0* ≠ *0* *poly* (*of-int-poly q*) (*exp 1* :: *complex*) = *0*
by (*elim algebraicE'-nonzero*) *simp-all*

define *n* :: *nat* **where** *n* = *degree q*
from *q* **have** [*simp*]: *n* ≠ *0* **by** (*intro notI*) (*auto simp*: *n-def elim*!: *degree-eq-zeroE*)
define *qmax* **where** *qmax* = *Max* (*insert 0* (*abs* ‘ *set* (*coeffs q*)))
have *qmax-nonneg* [*simp*]: *qmax* ≥ *0* **by** (*simp add*: *qmax-def*)
have *qmax*: |*coeff q k*| ≤ *qmax* **for** *k*
by (*cases k* ≤ *degree q*)
(*auto simp*: *qmax-def coeff-eq-0 coeffs-def simp del*: *upt-Suc intro*: *Max.coboundedI*)
obtain *C* **where** *C*: *C* ≥ *0*
⋀*x*. *x* ∈ *closed-segment 0* (*of-nat n*) ⟹ *norm* (∏ *k*∈{*1*..*n*}. (*x* − *of-nat k* :: *complex*)) ≤ *C*
by (*erule e-transcendental-aux-bound*)
define *E* **where** *E* = (*1* + *real n*) ∗ *real-of-int qmax* ∗ *real n* ∗ *exp* (*real n*) / *real n*
define *F* **where** *F* = *real n* ∗ *C*

have *ineq*: *fact* (*p* − *1*) ≤ *E* ∗ *F* ^ *p* **if** *p*: *prime p* *p* > *n p* > *abs* (*coeff q 0*) **for** *p*
proof −
from *p*(*1*) **have** *p-pos*: *p* > *0* **by** (*simp add*: *prime-gt-0-nat*)
define *f* :: *int poly*
where *f* = *monom 1* (*p* − *1*) ∗ (∏ *k*∈{*1*..*n*}. [:−*of-nat k*, *1*:] ^ *p*)
have *poly-f*: *poly* (*of-int-poly f*) *x* = *x* ^ (*p* − *1*) ∗ (∏ *k*∈{*1*..*n*}. (*x* − *of-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-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 :: *int* **where** $M = (-1) \hat{} (n * p) * fact\ n \hat{} p$
with p **have** *p-not-dvd-M*: $\neg int\ p\ dvd\ M$
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 :: *complex* **where** $J = (\sum k{\le}n.\ of\text{-}int\ (coeff\ q\ k) * I\ (of\text{-}nat\ k))$
define *idxs* **where** $idxs = (\{..n\}{\times}\{..m\}) - \{(0,\ p-1)\}$

hence $J = (\sum k{\le}n.\ of\text{-}int\ (coeff\ q\ k) * exp\ 1 \hat{} k) * (\sum n{\le}m.\ of\text{-}int\ (poly\ ((pderiv \hat{\hat{}} n)\ f)\ 0)) -$
$of\text{-}int\ (\sum k{\le}n.\ \sum n{\le}m.\ coeff\ q\ k * poly\ ((pderiv \hat{\hat{}} n)\ f)\ (int\ k))$
by (*simp add*: *J-def I-def algebra-simps sum-subtractf sum-distrib-left m-def exp-of-nat-mult* [*symmetric*])
also have $(\sum k{\le}n.\ of\text{-}int\ (coeff\ q\ k) * exp\ 1 \hat{} k) = poly\ (of\text{-}int\text{-}poly\ q)\ (exp\ 1 :: complex)$
by (*simp add*: *poly-altdef n-def*)
also have $\ldots = 0$ **by** *fact*
finally have $J = of\text{-}int\ (-(\sum (k,n){\in}\{..n\}{\times}\{..m\}.\ coeff\ q\ k * poly\ ((pderiv \hat{\hat{}} n)\ f)\ (int\ k)))$
by (*simp add*: *sum.cartesian-product*)
also have $\{..n\}{\times}\{..m\} = insert\ (0,\ p-1)\ idxs$ **by** (*auto simp*: *m idxs-def*)
also have $-(\sum (k,n){\in}\ldots.\ coeff\ q\ k * poly\ ((pderiv \hat{\hat{}} n)\ f)\ (int\ k)) =$
$-\ (coeff\ q\ 0 * poly\ ((pderiv \hat{\hat{}} (p-1))\ f)\ 0) -$
$(\sum (k,\ n){\in}idxs.\ coeff\ q\ k * poly\ ((pderiv \hat{\hat{}} n)\ f)\ (of\text{-}nat\ k))$
by (*subst sum.insert*) (*simp-all add*: *idxs-def*)
also have $coeff\ q\ 0 * poly\ ((pderiv \hat{\hat{}} (p-1))\ f)\ 0 = coeff\ q\ 0 * M * fact\ (p-1)$
proof −
have $f = [:-0,\ 1:] \hat{} (p-1) * (\prod k = 1..n.\ [:-\ of\text{-}nat\ k,\ 1:] \hat{} p)$
by (*simp add*: *f-def monom-altdef*)
also have $poly\ ((pderiv \hat{\hat{}} (p-1))\ \ldots)\ 0 =$
$fact\ (p-1) * poly\ (\prod k = 1..n.\ [:-\ of\text{-}nat\ k,\ 1:] \hat{} p)\ 0$
by (*rule poly-higher-pderiv-aux2*)
also have $poly\ (\prod k = 1..n.\ [:-\ of\text{-}nat\ k :: int,\ 1:] \hat{} p)\ 0 = (-1) \hat{} (n*p) * fact\ n \hat{} p$
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 have $\exists N.\ (\sum (k,\ n){\in}idxs.\ coeff\ q\ k * poly\ ((pderiv \hat{\hat{}} n)\ f)\ (int\ k)) = fact\ p * N$
proof −
have $\forall (k,\ n){\in}idxs.\ fact\ p\ dvd\ poly\ ((pderiv \hat{\hat{}} n)\ f)\ (of\text{-}nat\ k)$
proof *clarify*
fix $k\ j$ **assume** *idxs*: $(k,\ j) \in idxs$

```
      then consider k = 0 j < p − 1 | k = 0 j > p − 1 | k ≠ 0 j < p | k ≠ 0
j ≥ p
        by (fastforce simp: idxs-def)
      thus fact p dvd poly ((pderiv ^^ 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 (∑(k, n)∈idxs. coeff q k * poly ((pderiv ^^ n) f) (int k))
      by (auto intro!: dvd-sum dvd-mult simp del: of-int-fact)
    thus ?thesis by (blast elim: dvdE)
  qed
  then guess N .. note N = this
  also from p have − (coeff q 0 * M * fact (p − 1)) − fact p * N =
                    − fact (p − 1) * (coeff q 0 * M + p * N)
    by (subst fact-reduce[of p]) (simp-all add: algebra-simps)
   finally have J: J = −of-int (fact (p − 1) * (coeff q 0 * M + p * N)) by
simp

  from p q(2) have ¬p dvd 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 coeff q 0 * M + p * N ≠ 0 by (intro notI) simp-all
  hence abs (coeff q 0 * M + p * N) ≥ 1 by simp
  hence norm (of-int (coeff q 0 * M + p * N) :: complex) ≥ 1 by (simp only:
norm-of-int)
 hence fact (p − 1) * ... ≥ fact (p − 1) * 1 by (intro mult-left-mono) simp-all
    hence J-lower: norm J ≥ fact (p − 1) unfolding J norm-minus-cancel
of-int-mult of-int-fact
    by (simp add: norm-mult)

  have norm J ≤ (∑k≤n. norm (of-int (coeff q k) * I (of-nat k)))
    unfolding J-def by (rule norm-sum)
```

```
  also have ... ≤ (∑ k≤n. of-int qmax * (real n * exp (real n) * real n ^ (p −
1) * C ^ p))
  proof (intro sum-mono)
    fix k assume k: k ∈ {..n}
    have n > 0 by (rule ccontr) simp
    {
      fix x :: complex assume x: x ∈ closed-segment 0 (of-nat k)
      then obtain t where t: t ≥ 0 t ≤ 1 x = of-real t * of-nat k
        by (auto simp: closed-segment-def scaleR-conv-of-real)
      hence norm x = t * real k by (simp add: norm-mult)
     also from ‹t ≤ 1› k have *: ... ≤ 1 * real n by (intro mult-mono) simp-all
      finally have x': norm x ≤ real n by simp
      from t ‹n > 0› * have x'': x ∈ closed-segment 0 (of-nat n)
        by (auto simp: closed-segment-def scaleR-conv-of-real field-simps
              intro!: exI[of - t * real k / real n] )
      have norm (poly (of-int-poly f) x) =
            norm x ^ (p − 1) * cmod (∏ i = 1..n. x − i) ^ p
        by (simp add: poly-f norm-mult norm-power)
      also from x x' x'' have ... ≤ of-nat n ^ (p − 1) * C ^ p
        by (intro mult-mono C power-mono) simp-all
      finally have norm (poly (of-int-poly f) x) ≤ real n ^ (p − 1) * C ^ p .
    } note A = this

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

 have (λn. E * F * F ^ (n − 1) / fact (n − 1)) ⟶ 0 (is ?P)
  by (intro filterlim-compose[OF power-over-fact-tendsto-0' filterlim-minus-nat-at-top])
 also have ?P ⟷ (λn. E * F ^ n / fact (n − 1)) ⟶ 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 (λn. E * F ^ n / fact (n − 1) < 1) at-top
  by (rule order-tendstoD) simp-all
```

hence *eventually* $(\lambda n.\ E * F\ \hat{}\ n < fact\ (n - 1))$ *at-top* **by** *eventually-elim simp*
then obtain P **where** P: $\bigwedge n.\ n \geq P \Longrightarrow E * F\ \hat{}\ n < fact\ (n - 1)$
by (*auto simp*: *eventually-at-top-linorder*)

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

corollary *e-transcendental-real*: $\neg\ algebraic\ (exp\ 1 :: real)$
proof −
have $\neg algebraic\ (exp\ 1 :: complex)$ **by** (*rule e-transcendental-complex*)
also have $(exp\ 1 :: complex) = of\text{-}real\ (exp\ 1)$ **using** *exp-of-real*[*of 1*] **by** *simp*
also have $algebraic\ \ldots \longleftrightarrow algebraic\ (exp\ 1 :: real)$ **by** *simp*
finally show *?thesis* **.**
qed

end

References

[1] R. Lipsett. Planetmath. http://planetmath.org/proofoflindemannweierstrasstheoremandthateandpiaretranscendental, 2007.