A formalisation of transcendence of e

Jujian Zhang (CID: 01803812)

Supervisor: Prof. Kevin Buzzard

2020

Declaration

I declare that this report was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text.

Acknowledgement

I would like to thanks professor Kevin Buzzard for supervising this project whose advise has helped me to write more idiomatic `Lean` codes. He introduced me to the area of formalisation, hence I owe this unique experience to him. I am also thankful for the `Lean` community for providing `library_search`.

Abstract

The objective of this report is to present formalization of some basic theorems from transcendental number theory with `Lean` and `mathlib` in the hope that it will motivate and inspire other mathematicians, by igniting their curiosity about interactive theorem proving. The following theorems are formalized:

1. the set of algebraic numbers is countable, hence transcendental number exists:

```
theorem algebraic_set_countable : set.countable algebraic_set
theorem transcendental_number_exists :
  ∃ x : ℝ, transcendental x
```

2. all Liouville numbers are transcendental:

```
theorem liouville_numbers_transcendental :
  ∀ x : ℝ, liouville_number x → transcendental x
```

3. $\alpha := \sum_{i=0}^{\infty} \frac{1}{10^{i!}}$ is a Liouville number hence α is transcendental.

```
theorem liouville_α : liouville_number α
theorem transcendental_α : transcendental α :=
  liouville_numbers_transcendental α liouville_α
```

4. e is transcendental:

```
theorem e_transcendental : transcendental e
```

Contents

Chapter 1

Overview

1.1 Interactive theorem proving

Around 1920s, the German mathematician David Hilbert put forward a programme to seek:

1. an axiomatic foundation of mathematics;
2. a proof of consistency of the said foundation;
3. Entscheidungsproblem: an algorithm to determine if any proposition is universally valid given a set of axioms.

The first two aims were later proved to be impossible by Gödel and his celebrated incompleteness theorems. Via the completeness of first order logic, the Entscheidungsproblem can also be interpreted as an algorithm for producing proofs using deduction rules. Even without a panacea approach for mathematics, a computer still bears advantages against a carbon-based mathematician. Perhaps the most manifested advantage is the accuracy of a computer with which it execute its command and to recall its memories. Thus came the idea of **interactive theorem proving** — instead of hoping a computer algorithm can spit out some unfathomable proofs, assuming computers are given the ability to check the correctness of proofs, so human-comprehensible proofs can be verified by machines and thus guaranteed to be free of errors. With a collective effort, all theorems verified this way can be collected in an error-free library such that all mathematicians can utilise to prove further theorems, which can then be added to the collection, ad infinitum [Boy+94]. Curry-Howard isomorphism provided the crucial relationship between mathematical proofs and computer programmes, more specifically relationship between propositions and types, to make such project feasible [KK11]. The idea will be explained in section 2 along with `Lean`.

The proof of "Kepler's conjecture[1]" illustrates and exemplifies the utility of

[1] the most efficient way to pack spheres should be hexagonally

interactive theorem proving. As early as 1998, Thomas Hales had claimed a proof [Hal98; HUW14], however the proof is controversial in the sense that other mathematician even after great effort could not guarantee its correctness. A collaborative project using `Isabelle`[2] and `HOL Light`[3] verified the proof around 2014, hence settled the controversy in 2017 [Hal+17]. There is also Georges Gonthier with his teams using `Coq`[4] who formalised the four colour theorem and Feit-Thompson theorem where the latter is a step closer to the classification of simple groups [Gon08; Gon+13]. Additionally, using `Lean`[5], Buzzard, Commelin, and Massot were able to formalise the modern notion of perfectoid spaces [BCM20].

1.2 History of transcendental numbers

"Transcendence" as a mathematical jargon first appeared in a Leibniz's 1682 paper where he proved that sine is a transcendental function in the sense that for any natural number n there does not exist polynomials $p_0, \cdots, p_n$ such that

$$p_0(x) + p_1(x)\sin(x) + p_2(x)\sin(x)^2 + \cdots + p_n(x)\sin(x)^n = 0$$

holds for all $x \in \mathbb{R}$ [Bou98]. The Swiss mathematician Johann Heinrich Lambert in his 1768 paper proved the irrationality of e and π and he also conjectured their transcendence [Lam04]. It is not until 1844 that Joseph Liouville proved the existence of any transcendental number and until 1851 an explicit example of transcendental number was actually given by its decimal expansion:[Kem16]

$$\sum_{i=1}^{\infty} \frac{1}{10^{i!}} = 0.11000100000\cdots.$$

However, this construction is still artificial in nature. The first example of a real number proven to be transcendental that is not constructed for the purpose of being transcendental was e. Charles Hermite proved the transcendence of e in 1873 with a method applicable (with help of symmetric polynomial) to π in 1882 and later to be generalised to Lindemann-Weierstrass theorem in 1885 stating that if $\alpha_1, \cdots, \alpha_n$ are distinct algebraic numbers then $e^{\alpha_1}, \cdots, e^{\alpha_n}$ are linearly independent over the algebraic numbers [Bak90]. The transcendence of π was particularly celebrated because it immediately implied the impossibility of the ancient greek challenge of squaring the circle, i.e. it is not possible to construct a square, using compass and ruler only, with equal area to a circle. This question is plainly equivalent to construct $\sqrt{\pi}$, which is not possible for otherwise π is algebraic. Georg Cantor in 1874 proved that algebraic numbers are countable hence not only do transcendental numbers exist, they exist in

[2] a theorem prover relies extensively on dependent type theory and Curry-Howard correspondence.

[3] ibid.

[4] ibid.

[5] ibid.

a ubiquitous manner – there is a bijection from the set of all transcendental numbers to $\mathbb{R}$ [Can32; Can78].

In 1900, Hilbert proposed twenty-three questions, the 7th of which is regarding transcendental numbers: Is a^b transcendental, for any algebraic number a that is not 0 or 1 and any irrational algebraic number b? The answer is yes provided by Gelfond-Schneider theorem in 1934 [Gel34]. This led to some immediate consequences such that

1. $2^{\sqrt{2}}$ and its square root $\sqrt{2}^{\sqrt{2}}$ are transcendental;
2. e^π is transcendental for $e^\pi = \left(e^{i\pi}\right)^{-i} = (-1)^{-i}$;
3. $i^i = e^{-\frac{\pi}{2}}$ is transcendental etc.

In contrast, none of $\pi \pm e$, $\pi e, \dfrac{\pi}{e}$, π^π, π^e, etc were proven to be transcendental. It was also conjectured by Stephen Schanuel that given any n $\mathbb{Q}-$linearly independent $z_1, \cdots, z_n \in \mathbb{C}$, then $\operatorname{trdeg}\left(\mathbb{Q}(z_1, \cdots, z_n, e^{z_1}, \cdots, e^{z_n})/\mathbb{Q}\right)$ is at least n [Lan66]. If this was proven, the algebraic independence of e and π would follow immediately by setting $z_1 = 1$ and $z_2 = \pi i$ with Euler's identity.

Chapter 2

Brief introduction to `Lean`

`Lean` was developed by Leonardo de Moura at Microsoft Research Redmond in 2013 using dependent type theory and calculus of inductive constraint [AMK15]. In this chapter, basic ideas of Curry-Howard isomorphism will be demonstrated by some basic examples of mathematical theorem expressed in `Lean` using dependent type theory.

2.1 Simple type theory

Unlike set theory where everything from basic things like natural numbers to complex constructions like modular forms is essentially a set, type theory associate every expression with a `type`. In set theory, an element can belong to different sets, for example 0 is simultaneously in $\mathbb{N} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$. However an expression can only have one type. 0 without any context will have type $\mathbb{N}$ and, to specify the zero with type $\mathbb{R}$ we write $(0 : \mathbb{R})$. If a has type α, we write $a : \alpha$. By a universe of types we mean a collection of types. Types can be combined to form new types in the following way:

- let α and β be types then $\alpha \to \beta$ is the type of functions from α to β: the element of type $\alpha \to \beta$ is a function that for any element of α gives an element of β. For mathematicians this loosely means that for any two classes α and β, there is a new class $\hom(\alpha, \beta)$. Sometimes we are not bothered to give a function a name, we can use the λ notation: the expression $(\lambda x : \alpha, \text{expression})$ has type $\alpha \to \ldots$ depending on the content of expression. For example $(\lambda x : \mathbb{N}, x + 1) : \mathbb{N} \to \mathbb{N}$ is the function from natural numbers to natural numbers that add 1 to any input.

- let α and β be types then $\alpha \times \beta$ is the cartesian product of α and β: the element of type $\alpha \times \beta$ is an ordered tuple (a, b) where $a : \alpha$ and $b : \beta$.

- Let α be a type in universe $\mathcal{U}$ and $\beta : \alpha \to \mathcal{U}$ be a family of type that for any $a : \alpha$, $\beta(a)$ is a type in $\mathcal{U}$. Then we can form the Π-type

$$\prod_{a:\alpha} \beta(a)$$

whose element is of the form $f : \prod_{a:\alpha} \beta(a)$ such that for any $x : \alpha$, $f(x) : \beta(x)$. Note that function type is actually an example of Π-type where β is a constant family of types. For this reason, we also call Π-types dependent functions. For example if $\mathrm{Vec}(\mathbb{R}, n)$ is the type of $\mathbb{R}^n$, then

$$n \mapsto \underbrace{(1, \cdots, 1)}_{n \text{ times}} : \prod_{m:\mathbb{N}} \mathrm{Vec}(\mathbb{R}, m)$$

- We also have dependent cartesian product or Σ-type: Let α be a type in universe $\mathcal{U}$ and $\beta : \alpha \to \mathcal{U}$ be a family of types in $\mathcal{U}$, then the Σ-type

$$\sum_{a:\alpha} \beta(a)$$

whose element is of the form $(x, y) : \sum_{a:\alpha} \beta(a)$ such that $x : \alpha$ and $y : \beta(x)$. Similarly

$$\left(n, \underbrace{(1, \cdots, 1)}_{n \text{ times}}\right) : \sum_{m:\mathbb{N}} \mathrm{Vec}(\mathbb{R}, m)$$

2.1.1 Proposition as type

In type theory, a proposition p can be regarded as a type whose elements is a proof of p.

Example 1. $1 + 1 = 2$ is a proposition. `rfl` is an element of type $1 + 1 = 2$ where `rfl` is the assertion that every term equals to itself.

Example 2. For two propositions p and q, the implication $p \implies q$ then can be interpreted as function $p \to q$. To say $\mathrm{imp} : p \to q$ is to say for any $\mathrm{hp} : p$ we have $\mathrm{imp}(\mathrm{hp}) : q$, or equivalently given any hp, a *proof* of proposition p, $\mathrm{imp}(\mathrm{hp})$ is a *proof* of proposition q.

Example 3. If $p : \alpha \to$ proposition, the proposition $\forall x : \alpha, p(x)$ can be interpreted as a Π-type $\prod_{x:\alpha} p(x)$. To prove $\forall x : \alpha, p(x)$, we need to find an element of type $\prod_{x:\alpha} p(x)$; equivalently for any $x : \alpha$, we need to find an element of type $p(x)$; equivalently for any $x : \alpha$, we need to find a proof of $p(x)$.

Similarly, $\exists x : \alpha, p(x)$ can be interpreted as a Σ-type $\sum_{x:\alpha} p(x)$. To prove $\exists x : \alpha, p(x)$ is to find an element x of type α and prove $p(x)$, equivalent to find an element $x : \alpha$ and an element of type $p(x)$ and this is precisely $(x, p(x)) : \sum_{a:\alpha} p(a)$.

Theorems are true propositions. Using the interpretation above, theorems are inhabited types and to prove a theorem is to find an element of the required type.

2.2 Lean and mathlib

`mathlib` is the collection of mathematical definitions, theorems, lemmas built on `Lean`. `mathlib` includes topics in algebra, topology, manifolds and combinatorics etc. The following section will briefly explain how to use `Lean` with `mathlib`.

In `Lean`, new definition can be introduced with the following syntax:

```
def name (arg₁:type₁) ... (argₙ:typeₙ) : return_type := contents

def name' {arg₁:type₁} ... (argₙ:typeₙ) : return_type := contents
```

`return_type` is optional when it can be inferred from `contents`. If an argument is enclosed by curly brackets instead of round brackets, then when the definition is invoked the said argument is implicit, i.e. `name'` $\texttt{a}_2$ `...` $\texttt{a}_n$ where $\texttt{a}_i\texttt{:type}_i$. To explicitly mention the said argument, one needs to use `@name'` $\texttt{a}_1$ `...` $\texttt{a}_n$ where $\texttt{a}_i\texttt{:type}_i$. One can use "if then else" to introduce a function whose value depends on the value of the arguments:

```
def name args : return_type :=
  if (h args)
  then contents₁
  else contents₂

def name args : return_type :=
  ite (h args) contents₁ contents₂
```

New notations are introduced with the following syntax:

```
notation _`lhs`_ := _rhs_
```

so that `Lean` will treat every occurrence of ``_`lhs`_`` as `_rhs_` verbatim. For example ``notation ℤ`[X]` := polynomial ℤ`` will replace `polynomial ℤ` with a more familiar notation of $\mathbb{Z}[X]$.

For any type of α, we can introduce a subtype of α by:

```
def α' := {x : α // property_satisfied_by_x}
```

An element of type α' is of the form $\langle x, hx\rangle$ where $x : \alpha$ and hx is a proof that x satisfies the given property.

Theorems or lemmas are introduced with the following syntax:

```
theorem name (arg₁:type₁) ... (argₙ:typeₙ) : content :=
begin
  -- proof of the theorem
end
```

To write a proof understandable to `Lean`, one need to use *tactic mode*. In `Lean`, one can use

- proof by induction: if the goal is a proposition about natural number n, `induction n with n IH` is to prove the proposition by induction. This command will change the current goal to two goals. The first is to prove the proposition for $n = 0$ and the second to prove the proposition for $n+1$ with the additional inductive hypothesis `IH`;

```
theorem awesome_theorem_about_natural_number (n : ℕ) :
↪  propositionₙ :=
begin
  induction n with n IH,

  a_proof_of_proposition₀

  -- (IH : propositionₙ) is now in context
  a_proof_of_propositionₙ₊₁
end
```

- proof by contradiction: if the goal is to prove proposition H, `by_contra absurdum` will add `absurdum :` $\neg H$ into the current context and turn the goal into proving `false`;

```
theorem awesome_theorem : awesome_proposition :=
begin
by_contra absurdum,

-- Now (absurdum : ¬ awesome_proposition) is in context and
↪  the goal is to prove falsehood.
a_proof_of_falsehood
end
```

- proof in a forward manner i.e. introduce new theorem into the current context or convert known theorem in the current context to approach the goal:
  - `have H := content` will introduce a new proposition whose proof is given by `content`.

    `have H : some_proposition` will add one more goal of proving the proposition then introduce the proved proposition to the current context.

- If `H` is in context then `replace H := content` will change `H` to (a proof of) the proposition that `content` is proving.

  `replace H : some_proposition` will add one more goal of proving `some_proposition` and then replace `H` to the proposition proven.
- If `H` is in context, `simp at H` will simplify `H` to using small lemmas[1]. `simp only [h1,…,hn]` is to simplify only using `h1 ... hn`.
- `rw` is for term rewriting. If we have `h : lhs = rhs` or `h : lhs↔rhs` and another `H` in context, then `rw h at H` will replace every occurrence of `lhs` with `rhs` in `H` and `rw ←h` will replace every occurrence of `rhs` with `lhs` in `H`.

  `rw [h1, h2, ... , hn] at H` is the same as `rw h1 at H, rw h2 at H, ... , rw hn at H`.
- The fact that `rw` and `simp` change all term occurrences sometimes creates an inconvenience. If `H` is in context, `conv_lhs at H {tactics}` will confine the scope of `tactics` only to the left hand side of H; similarly `conv_rhs at H {tactics}` will confine the scope to the right hand side of H.
- `generalise H : lhs = var_name` will set `var_name` to `lhs` and add (proof of) the proposition `H : lhs = var_name` to the current context.
- If `H : ∃ x : type, property_about_x` is in the current context, `choose x hx using H` will introduce `x:type` with the assumption `hx : property_about_x` to the current context.
- If `H :` $p \wedge q$ is in the current context, then `H.1` is (a proof of) p and `H.2` is (a proof of) q.
- If `H : ite h1 h2 h3` is in the current context, then `split_ifs at H` will turn the current goal into two goals, the first one is to prove the original goal with the additional assumption `h1` and `h2`; the second one is to prove the original with goal with the additional assumption ¬`h1` and `h3`.

- proof in a backward manner i.e. convert or replace the goal so that it is closer to what is known in context:
  - `unfold definition` is to unfold a definition to what is explicitly defined when the definition is introduced.
  - `simp`, `rw`, `conv_lhs {tactics}` and `conv_rhs {tactics}` is the same as above except now they change at goal.
  - Given (a proof of) proposition `H: h1 → h2`, then `apply H` will change the goal of proving `h2` to prove `h1`.

[1] to be more precise, lemma with `@[simp]` tag, i.e. lemmas declared in the following syntax `@[simp] lemma lemma_name args : Prop`. These lemma are usually trivial in nature such as `nat.add_zero` which asserts that $\forall n : \mathbb{N}, n + 0 = n$.

- `suffices H : some_proposition` ask for a proof of the current goal with additional `H`, then ask for a proof of `H`.
- `norm_cast` converts the type of numbers. For example the current goal is $(x : \mathbb{R}) < (y : \mathbb{R})$ where x and y are of type $\mathbb{N}$, then after `norm_cast` the goal will become $x < y$. This should be simpler because $\mathbb{R}$ in `Lean` is defined as equivalent classes of Cauchy sequence of $\mathbb{Q}$ while natural number is much easier to work with.

  `norm_num` is equivalent to `norm_cast, simp`.
- `ext` will convert the current goal with axioms of extensionality. For example if the goal is to prove equality of polynomial then after `ext` the goal would become to prove that every coefficient is equal; or if the goal is to prove equality of sets of type α $A = B$, then after `ext`, an arbitrary element `x` of type α will be introduced into context then the goal will become to prove $x \in A \iff x \in B$. `ext var_name` will force `Lean` to introduce a new variable under the identifier `var_name`.
- If the goal is to prove `ite h1 h2 h3` (or `ite h1 h2 lhs = rhs`), then `split_ifs at H` will turn the current goal into two goals, the first one is to prove `h2` (`lhs = rhs` resp.) with the additional assumption `h1`; the second one is to prove `h3` (`lhs = rhs` resp.) with the additional assumption $\neg$`h1`.

• when the goal is easily provable, one can use the following to finish a goal:

- `refl` (for reflexive) is used to prove propositions of the form `lhs = rhs` when `lhs` is **definitionally** equal to `rhs`. Definitional equality is more general than two string being literally identical but is less general than being (canonically) isomorphic. For example

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \sum_{j=0}^{\infty} \frac{1}{2^j}$$

  is a definitional equality but

$$\mathbb{R}^n = \mathrm{Func}\left(\{0, \cdots, n-1\}, \mathbb{R}\right)$$

  is not a definitional equality (strictly speaking perhaps not an equality at all).
- `exact H` will prove the current goal if the goal is definitionally equal to `H`.
- `ring` will try to prove the current goal using associativity and commutativity of addition and multiplication.
- `linarith` is used when proving inequality from context. `linarith` is semi-automated, so it can work with inequalities with symbols or variables but only to a degree. If `linarith` fails, one has to either

provide `linarith` with more propositions or use other tactics to change the goal into something more manageable the use `linarith`. `linarith [h1, ... , hn]` is equivalent to use `linarith` with additional (proofs of) propositions `h1` ... `hn`.

  – `tidy` is to ask `Lean` to try different tactics and finishes the goal if it possible.

- If there is multiple goals, one can use `{ }` to focus on the first one.
- If the entirety of proof is one line, one can replace `begin contents end` with `by contents`.

A proposition if not atomic is either a conjunction, a disjunction, an implication, an equivalence, a negation or a proposition with universal quantifier or existential quantifier.

2.2.1 prove a conjunction

If the goal is to prove a conjunction of the form $h_1 \wedge h_2$, `split` is used. It will change the current goal to two goals of proving h_1 and h_2 respectively. Then the general pattern is

```
theorem how_to_prove_conjunction (h1 : Prop) (h2 : Prop) : h1 ∧ h2 :=
begin
split,

proof_of_h1

proof_of_h2
end
```

2.2.2 prove a disjunction

If the goal is to prove a disjunction of the form $h_1 \vee h_2$, one can use `left` to change the goal to prove h_1 or `right` to change the goal to prove h_2. Let us assume h_1 is a true proposition :

```
theorem how_to_prove_disjunction (h1 : Prop) (h2 : Prop) : h1 ∨ h2 :=
begin
left,

proof_of_h1
end
```

2.2.3 prove an implication

If the goal is to prove an implication of the form $p \implies q$, one can use `intro` hp to add hp:p a proof of p into the context and convert the goal to proving q.

```
theorem how_to_prove_implication (p : Prop) (q : Prop) : p → q :=
begin
intro hp,

proof_of_q
end
```

If the goal is of the form $p_1 \to p_2 \to \ldots p_n$, one can use `intros hp₁ … hpₙ` as an abbreviation of `intro hp₁, intro hp₂,…, intro hpₙ`.

2.2.4 prove an equivalence

An equivalence of the form $p \iff q$ is by definition $p \implies q \wedge q \implies p$. Thus `split` will change the goal to two goals, one to prove $p \implies q$, the other to prove $q \implies p$. Then use section 2.2.3.

2.2.5 prove a negation

A negation of the form $\neg p$ is by definition $p \implies \perp$. Thus `intro hp` will add `hp:p` to the current context and convert the goal to proving a falsehood.

```
theorem how_to_prove_negation (p : Prop) : ¬p :=
begin
intro hp,

proof_of_falsehood
end
```

2.2.6 prove a proposition with ∀

A proposition of the form $\forall a : \alpha, p(a)$ where α is a type and $p : \alpha \to$ `Prop` can be proved also using `intro` x_0. This will add an arbitrary $x_0 : \alpha$ to the current context and change the goal to proving $p(x_0)$.

```
theorem how_to_proposition_with_universal_quantifier {α : Type} (p
↪   : α → Prop) : ∀ a : α, p a :=
begin
intro x₀,

a_proof_of_p(x₀)
end
```

If the goal is the form $\forall a_1 : \alpha_1, \forall a_2 : \alpha_2, \ldots, \forall a_n : \alpha_n, p\ a_1\ a_2\ \ldots\ a_n$ can be proved using `intros a₁ a₂ … aₙ` as an abbreviation of `intro a₁, intro a₂,…, intro aₙ`.

2.2.7 prove a proposition with ∃

A proposition of the form $\exists a : \alpha, p(a)$ where α is a type and $p : \alpha \to$ `Prop` can be proved by `use` x_0. This will convert the goal to proving $p(x_0)$.

```
theorem how_to_proposition_with_universal_quantifier {α : Type} (p : α → Prop) : ∃ a : α, p a :=
begin
a_construction_of_x_0

use x_0,

a_proof_of_p(x_0)
end
```

2.3 An example

To illustrate the above syntax and patterns, an example is presented here of defining mean value of two real numbers and proving some basic properties thereof.

```
import data.real.basic
import tactic

noncomputable theory
open_locale classical

def mean (x y : ℝ) : ℝ := (x + y) / 2

theorem min_le_mean : ∀ x y : ℝ, min x y ≤ (mean x y) :=
begin
intros x y,
have ineq1 : min x y ≤ x := min_le_left x y,
have ineq2 : min x y ≤ y := min_le_right x y,

unfold mean, rw le_div_iff, rw mul_two,
apply add_le_add,
exact ineq1, exact ineq2,

linarith,
end

theorem mean_le_max : ∀ x y : ℝ, (mean x y) ≤ max x y :=
begin
intros x y,
have ineq1 : x ≤ max x y := le_max_left x y,
have ineq2 : y ≤ max x y := le_max_right x y,

unfold mean, rw div_le_iff, rw mul_two,
apply add_le_add,
exact ineq1, exact ineq2,

linarith,
```

```
33 end
34
35 theorem a_number_in_between :
36   ∀ x y : ℝ, x ≤ y → ∃ z : ℝ, x ≤ z ∧ z ≤ y :=
37 begin
38 intros x y hxy,
39 have ineq1 := min_le_mean x y,
40 have ineq2 := mean_le_max x y,
41 have min_eq_x := min_eq_left hxy,
42 have max_eq_y := max_eq_right hxy,
43 use mean x y,
44 split,
45
46 { conv_lhs {rw ←min_eq_x}, exact ineq1, },
47 { conv_rhs {rw ←max_eq_y}, exact ineq2, },
48 end
```

Line 1 makes basic properties of real available to use and line 2 makes all the tactics we discussed amongst other more advanced tactics available to use. We add line 4 so that `lean` would ignore the issue of computability and line 5 so that we can use proof by contradiction[2].

We define the mean value of two real numbers on line 7. Then `mean`[3] has the type $\mathbb{R} \to \mathbb{R} \to \mathbb{R}$, `mean 1` has the type $\mathbb{R} \to \mathbb{R}$ and `mean 1 2` has the type $\mathbb{R}$.

We can introduce and prove theorems about `mean` stating that the mean value of two numbers is greater than or equal to the minimum of the two numbers but less than the maximum. This is from line 9 to line 33 where

- `min_le_left` is a proof of the proposition $\forall (x\ y : \alpha), \min(x, y) \le x$ where α is an implicit argument with a linear order. In this case, `Lean` infers from context that α is $\mathbb{R}$. Thus `min_le_left x y` is a proof of `min x y ≤ x`.

- `min_le_right` is a proof of the proposition $\forall (x\ y : \alpha), \min(x, y) \le y$ In this case, `min_le_right x y` is a proof of `min x y ≤ y`.

- Similarly, `le_max_left` is a proof of the proposition $\forall (x\ y : \alpha), x \le \max(x, y)$ where α is an implicit argument with a linear order. In this case, `le_max_left` is a proof of `x ≤ max x y`.

- Similarly, `le_max_right` is a proof of the proposition $\forall (x\ y : \alpha), y \le \max(x, y)$ where α is an implicit argument with a linear order. In this case, `le_max_right` is a proof of `y ≤ max x y`.

- `le_div_iff` is a proof that $0 < c \to (a \le \frac{b}{c} \iff a \times c \le b)$ where a, b, c are elements of a type with a linear ordered field structure. So by

[2] `Lean` by default use constructivism where $\neg\neg p \implies p$ is not an axiom of deduction. Thus the law of excluded middle is not by default a tautology.

[3] `mean` is not a function $\mathbb{R}^2 \to \mathbb{R}$ but a function $\mathbb{R} \to \mathrm{Func}(\mathbb{R}, \mathbb{R})$. This is called currying.

`rw le_div_iff`, the goal would change from `min x y ≤ (x + y) / 2` to `min x y * 2 ≤ x + y`. Since `le_div_iff` requires the assumption that $0 < c$, a new goal to prove that `0 < 2` is created after the original goal. This goal is proved by the final `linarith`.

- `div_le_iff` is proof that $0 < b \implies (\frac{a}{b} \leq c \iff a \leq c \times b)$ where a, b, c are elements of a type with a linear ordered field structure. So by `rw div_le_iff` the goal would change from `(x + y) / 2 ≤ max x y` to `x + y ≤ max x y * 2`. Since `div_le_iff` requires the assumption that $0 < b$, a new goal to prove $0 < 2$ is created after the original goal. This goal is proved by the final `linarith`.

- `mul_two` proves the lemma that $\forall n : \alpha, n \times 2 = n + n$ where α is a semiring. Thus `rw mul_two` would change the goal of proving `min x y * 2 ≤ x + y` (`x + y ≤ max x y * 2` resp.) to `min x y + min x y ≤ x + y` (`x + y ≤ max x y + max x y` resp.).

- `add_le_add` proves the lemma that $a \leq b \to c \leq d \to a + c \leq b + d$ where a, b, c and d are elements of an ordered additive commutative monoid. Since the goal now is to prove `min x y + min x y ≤ x + y`, by `apply add_le_add`, goal will be replaced by two goals of proving `min x y ≤ x` and `min x y ≤ y`. These are *exactly* `ineq1` and `ineq2`.

Chapter 3

Formalisation using `Lean`

Logistics of the formalisation

There are five main files in this project where

1. `small_things.lean` formalised results about the trivial embedding of $\mathbb{Z}[X] \subset \mathbb{R}[X]$ and manipulation of inequality in real numbers common to all three parts;

2. `algebraic_over_Z.lean` formalised countability of algebraic numbers with help of Schröder-Berstein theorem.

3. `liouville.lean` formalised Liouville's theorem and a construction of a Liouville's number;

4. `e_trans_helpers2.lean` formalised some results about differentiation and integration. Especially the formalisations of

$$\frac{\mathrm{d}^n}{\mathrm{d}x^n}uv = \sum_{i=0}^{n} \binom{n}{i} \frac{\mathrm{d}^i u}{\mathrm{d}x^i} \frac{\mathrm{d}^{n-i} v}{\mathrm{d}x^{n-i}}$$

   where u and v are differentiable function from $\mathbb{R}$ to $\mathbb{R}$ and

$$\int_0^t e^{t-x} f(x)\mathrm{d}x = e^t \sum_{i=0}^{m} f^{(i)}(0) - \sum_{i=0}^{m} f^{(i)}(t)$$

   where $f(X) \in \mathbb{Z}[X]$;

5. `e_transcendental.lean` formalised transcendence of e by assuming the algebraicity of e which resulted in two contradictory bounds using the results from `e_trans_helpers2.lean`.

3.1 Countability argument

The main caveat in this part is the internal specification of `mathlib`. A real number x in `Lean` is algebraic over $\mathbb{Z}$ if and only if there exists a nonzero polynomial $p(X) \in \mathbb{Z}[X]$ such that p is in the kernel of the unqiue $\mathbb{Z}$-algebra homomorphism $\mathbb{Z}[X] \to \mathbb{R}$ given by $X \mapsto x$.

```
∃ (p : ℤ[X]), p ≠ 0 ∧ ⇑(polynomial.aeval ℤ ℝ x) p = 0
```

Here the $\mathbb{Z}$-algebra homomorphism is `polynomial.aeval ℤ ℝ x`. ⇑ is to convert the homomorphism to a function applicable to `p`. The reason that a conversion is necessary is because an algebra homomorphism contains more information than a function, it is a structure containing the map and other fields containing (proofs of) the properties of algebra homomorphism. However in polynomial library of `mathlib`, the definition of root is as following :

```
def is_root (p : polynomial R) (a : R) : Prop := p.eval a = 0
```

Thus the first part of this formalisation is to unify the two evaluation methods – denote ι_x to be the unique $\mathbb{Z}$-algebra homomorphism $\iota_x : \mathbb{Z}[X] \to \mathbb{R}$ given by $X \mapsto x$ then for all polynomial $p(X) \in \mathbb{Z}[X]$ and for all $x \in \mathbb{R}, \mathrm{eval}_p(x) = \iota_x p$:

```
1  -- the trivial embedding ℤ[X] ⊆ ℝ[X]
2  def poly_int_to_poly_real (p : ℤ[X]) : polynomial ℝ :=
   ↪  polynomial.map ℤembℝ p
3
4  def poly_int_to_poly_real_wd (p : ℤ[X]) :=
5    ∀ x : ℝ, polynomial.aeval ℤ ℝ x p = (poly_int_to_poly_real
     ↪  p).eval x
6
7  theorem poly_int_to_poly_real_well_defined
8    (x : ℝ) (p : ℤ[X]) : poly_int_to_poly_real_wd p :=
9  begin
10   proof_omitted
11 end
```

Source Code 3.1: unifying two ways of evaluation

For any $p \in \mathbb{Z}[X]$, we can define the set of roots to be $\{x \in \mathbb{R} | \mathrm{eval}_p(x) = 0\}$ or $\{x \in \mathbb{R} | \iota_x p = 0\}$ where the former is builtin as `↑(poly_int_to_poly_real p).roots`[1] and the latter is defined as line 1 in source code 3.2. By line 7 in source code 3.1, the two sets must be equal, hence they have finite cardinality for a none-zero polynomial:

[1] `_.roots` in fact has type `finset ℝ`. The type `finset` is a `set` with a proof of finite cardinality. Here ↑ is used to convert a `finset` to `set` by discarding the proof of finite cardinality.

```
def roots_real (p : ℤ[X]) : set ℝ :=
  {x | polynomial.aeval ℤ ℝ x p}

theorem roots_real_eq_roots (p : ℤ[X]) (hp : p ≠ 0) :
  roots_real p = ↑(poly_int_to_poly_real p).roots :=
begin
  proof_omitted
end

theorem roots_finite (p : ℤ[X]) (hp : p ≠ 0) :
  set.finite (roots_real p) :=
begin
  proof_omitted
end
```

Source Code 3.2: two ways of defining roots

We define the set of all algebraic numbers over $\mathbb{Z}$ to be

```
def algebraic_set : set ℝ := {x | is_algebraic ℤ x}
```

To investigate the countability of `algebraic_set`, we compare it with

$$\bigcup_{n \in \mathbb{N}} \bigcup_{\substack{p \in \mathbb{Z}[X] \\ p \neq 0 \\ \deg p < n+1}} \{x \in \mathbb{R} | \iota_x p = 0\}. \tag{3.1}$$

To this end, we introduce some types of interest:

```
notation `int_n` n := fin n → ℤ
notation `nat_n` n := fin n → ℕ
notation `poly_n'` n := {p : ℤ[X] // p ≠ 0 ∧ p.nat_degree < n}
notation `int_n'` n := {f : fin n → ℤ // f ≠ 0}
notation `int'` := {r : ℤ // r ≠ 0}
```

where $\langle m, hm \rangle$ is an element of `fin n` if and only if m is a natural number and hm is a proof of $m < n$. Then `fin n` is the type of only n elements. Thus

- `int_n n` is $\mathbb{Z}^n$;
- `int_n' n` is $\mathbb{Z}^n - \{(0, \ldots, 0)\}$;
- `int'` is $\mathbb{Z} - \{0\}$;
- `nat_n n` is $\mathbb{N}^n$;
- `poly_n' n` is the type of non-zero integer polynomials with degree less than n.

Then $\mathbb{Z} \simeq \mathbb{Z} - \{0\}$ by the bijective function $s : \mathbb{Z} \to \mathbb{Z} - \{0\}$:

$$m \mapsto \begin{cases} m & \text{if } m < 0 \\ m+1 & \text{if } m \geq 0 \end{cases}$$

```
def strange_fun : ℤ → int' :=
  λ m, if h : m < 0
       then ⟨m, by linarith⟩
       else ⟨m + 1, by linarith⟩

theorem strange_fun_inj :
  function.injective strange_fun :=
begin
  proof_omitted
end

theorem strange_fun_sur :
  function.surjective strange_fun :=
begin
  proof_omitted
end

theorem int_eqiv_int' : ℤ ≃ int' :=
begin
  apply equiv.of_bijective strange_fun,
  split,
  exact strange_fun_inj,
  exact strange_fun_sur,
end
```

Source Code 3.3: $\mathbb{Z} \simeq \mathbb{Z} - \{0\}$

Then we prove that for all non-zero $n : \mathbb{N}$, non-zero integer polynomials of degree less than n bijectively correspond to $\mathbb{Z}^n - \{(0, \ldots, 0)\}$ via the function: $p \mapsto \mathbf{z}$ where the i-th coordinate of $\mathbf{z}$ is the i-th coefficient of p.

```
def identify (n : nat) : (poly_n' n) → (int_n' n) :=
  λ p, ⟨λ m, p.1.coeff m.1, a_proof_z_is_not_zero⟩

theorem sur_identify_n (n : nat) (hn : n ≠ 0) :
    function.surjective (identify n) :=
begin
  proof_omitted
end

theorem inj_identify_n (n : nat) (hn : n ≠ 0) :
  function.injective (identify n) :=
begin
  proof_omitted
end

theorem poly_n'_equiv_int_n' (n : nat) :
  (poly_n' n.succ) ≃ (int_n' n.succ) :=
begin
```

```
  apply equiv.of_bijective (identify n.succ),
  split,
  exact inj_identify_n n.succ (nat.succ_ne_zero n),
  exact sur_identify_n n.succ (nat.succ_ne_zero n),
end
```

†: `n.succ` means $n + 1$.

Source Code 3.4: non-zero integer polynomials with degree less than n has the same cardinality as $\mathbb{Z}^n - \{(0, \ldots, 0)\}$

Then we define two injective functions $F : \mathbb{Z}^{n+1} \to \mathbb{Z}^{n+1} - \{(0, \ldots, 0)\}$ and $G : \mathbb{Z}^{n+1} - \{(0, \ldots, 0)\} \to \mathbb{Z}^{n+1}$ by:

$$F(m_0, \ldots, m_n) = (s(m_0), \ldots, s(m_n))$$
$$G(m_0, \ldots, m_n) = (m_0, \ldots, m_n)$$

where $s : \mathbb{Z} \to \mathbb{Z} - \{0\}$ is defined previously. By Schröder-Berstein theorem, there is then a bijection $\mathbb{Z}^{n+1} \to \mathbb{Z}^{n+1} - \{(0, \ldots, 0)\}$ and thus $\mathbb{Z}^{n+1} \simeq \mathbb{Z}^{n+1} - \{(0, \ldots, 0)\}$:

```
def F (n : nat) : (int_n n.succ) → (int_n' n.succ) :=
  λ f, ⟨λ m, (strange_fun (f m)).1,
    a_proof_of_(s(m_0),...,s(m_n))_non-zero⟩
theorem F_inj (n : nat) : function.injective (F n) :=
begin
  proof_omitted
end

def G (n : nat) : (int_n' n.succ) → (int_n n.succ) :=
  λ f m, (f.1 m)
theorem G_inj (n : nat) : function.injective (G n) :=
begin
  proof_omitted
end

theorem int_n_equiv_int_n' (n : nat) :
  (int_n n.succ) ≃ int_n' n.succ :=
begin
  choose B HB using function.embedding.schroeder_bernstein (F_inj
  ↪  n) (G_inj n),
  apply equiv.of_bijective B HB,
end
```

Source Code 3.5: $\mathbb{Z}^{n+1} \simeq \mathbb{Z}^{n+1} - \{(0, \ldots, 0)\}$

For any natural number $n \geq 1$, we then construct two injective function $f_n : \mathbb{Z}^{n+2} \to \mathbb{Z}^{n+1} \times \mathbb{Z}$ and $g_n : \mathbb{Z}^{n+1} \times \mathbb{Z} \to \mathbb{Z}^{n+2}$:

$$f_n((m_0, \ldots, m_{n+1})) = ((m_0, \ldots, m_n), m_{n+1})$$
$$g_n(((m_0, \ldots, m_n), m_{n+1})) = (m_0, \ldots, m_{n+1})$$

Then by Schröder-Berstein theorem $\mathbb{Z}^{n+2} \simeq \mathbb{Z}^{n+1} \times \mathbb{Z}$ for all $n \geq 1$.

```
def fn (n : nat) :
  (int_n n.succ.succ) → (int_n n.succ) × ℤ :=  λ r,
  ⟨λ m, r (⟨m.1, nat.lt_trans m.2 (nat.lt_succ_self n.succ)⟩),
   r (⟨n.succ, nat.lt_succ_self n.succ⟩)⟩
theorem fn_inj (n : ℕ) : function.injective (fn n) :=
begin
  proof_omitted
end

def gn (n : nat) : (int_n n.succ) × ℤ → (int_n n.succ.succ) := λ r m,
begin
  by_cases (m.1 = n.succ),
    exact r.2,
    exact r.1 (⟨m.1, lt_of_le_of_ne (fin.le_last m) h⟩),
end
theorem gn_inj (n : nat) : function.injective (gn n) :=
begin
  proof_omitted
end

theorem aux_int_n (n : nat) :
  (int_n n.succ.succ) ≃ (int_n n.succ) × ℤ :=
begin
choose B HB using function.embedding.schroeder_bernstein (fn_inj n) (gn_inj n),
apply equiv.of_bijective B HB,
end
```

Source Code 3.6: $\mathbb{Z}^{n+2} \simeq \mathbb{Z}^{n+1} \times \mathbb{Z}$ for all $n \geq 1$

Now we are finally in the position of using equation 3.1 to prove the countability of all algebraic numbers. We first define the set of real roots of non-zero integer polynomials of degree less than n to be:

```
def algebraic_set'_n (n : ℕ) : set ℝ :=
  ⋃ p : (poly_n' n.succ), roots_real p.1
```

Hence by taking union over all natural numbers we can obtain an equivalent definition of all algebraic number over $\mathbb{Z}$:

```
def algebraic_set' : set real :=
  ⋃ n : ℕ, algebraic_set'_n n.succ

theorem algebraic_set'_eq_algebraic_set :
  algebraic_set' = algebraic_set :=
begin
  proof_omitted
end
```

We prove by induction that for any $n \in \mathbb{N}$, $\mathbb{Z}^{n+1}$ is denumerable (i.e. countably infinite) where the base case is $\mathbb{Z}^1 \simeq \mathbb{Z}$ and the inductive step is to prove

$\mathbb{Z}^{n+2}$ is denumerable using the denumerability of $\mathbb{Z}^{n+1}$. Since non-zero integer polynomials of degree less than $n+1$ bijectively corresponds to $\mathbb{Z}^{n+1}$, we infer that non-zero integer polynomials of degree less than $n+1$ are denumerable hence countable. Then the result of taking union over the countable set $\mathbb{N}$

$$\bigcup_{n\in\mathbb{N}} \bigcup_{\substack{p\in\mathbb{Z}[X] \\ p\neq 0 \\ \deg p<n+1}} \{x\in\mathbb{R}|\iota_x p=0\}$$

is still countable. Then finally the set of all algebraic numbers over $\mathbb{Z}$ is conclude to be countable. Since $\mathbb{R}$ is uncountable, transcendental number must exist:

```
theorem int_1_equiv_int : (int_n 1) ≃ ℤ :=
begin
  proof_omitted
end

theorem int_n_denumerable {n : nat} :
  denumerable (int_n n.succ) :=
begin
  proof_omitted
end

theorem poly_n'_denumerable (n : nat) :
  denumerable (poly_n' n.succ) :=
begin
  proof_omitted
end

theorem algebraic_set'_n_countable (n : nat) :
  set.countable (algebraic_set'_n n) :=
begin
  proof_omitted
end

theorem algebraic_set'_countable :
  set.countable algebraic_set' :=
  set.countable_Union
    (λ n, algebraic_set'_n_countable n.succ)

theorem algebraic_set_countable :
  set.countable algebraic_set :=
begin
  rw ←algebraic_set'_eq_algebraic_set,
  exact algebraic_set'_countable
end

theorem transcendental_number_exists :
  ∃ x : ℝ, transcendental x :=
begin
  proof_omitted
end
```

Source Code 3.7: algebraic numbers are countable, hence transcendental numbers exists.

3.2 Liouville's theorem and Liouville numbers

General theory about Liouville numbers

A Liouville number is a real number that is "almost rational", i.e. for any $n \in \mathbb{N}$ there is a rational number[2] $\dfrac{a}{b} \in \mathbb{Q}$ such that $b > 1$ and $0 < |x - \dfrac{a}{b}| < \dfrac{1}{b^n}$.

```
def liouville_number (x : ℝ) :=
  ∀ n : ℕ, ∃ a b : ℤ,
    b > 1 ∧
    0 < abs(x - a / b) ∧ abs(x - a / b) < 1/b^n
```

Source Code 3.8: Definition of Liouville number

We first prove a lemma about irrational roots of an integer polynomial:

Lemma 3.2.1. if f is an integer polynomial with degree $m > 1$ and α is an irrational root for $i(f)$ where $i : \mathbb{Z}[X] \to \mathbb{R}[X]$ is the trivial embedding, then there is a postive real number A such that for every rational number $\dfrac{a}{b}$, $\left|\alpha - \dfrac{a}{b}\right| > \dfrac{A}{b^m}$:

```
lemma about_irrational_root (α : ℝ)
  (hα : irrational α) (f : ℤ[X])
  (f_deg : f.nat_degree > 1)
  (α_root : f_eval_on_ℝ f α = 0) :
  ∃ A : ℝ, A > 0 ∧ ∀ a b : ℤ, b > 0 → abs(α - a/b) >
  ↪  (A/b^(f.nat_degree)) :=
```

Proof. We will abuse the notation to denote f both as the integer polynomial and the real polynomial via trivial embedding.

```
begin
  have f_nonzero : f ≠ 0,
    proof_omitted
  generalize hfℝ: f.map ℤembℝ = f_ℝ,
  have hfℝ_nonzero : f_ℝ ≠ 0,
    proof_omitted
  generalize hDf: f_ℝ.derivative = Df_ℝ,
```

[2]Without losing generality, we are always assuming, for any rational number, the denominator is a strictly positive natural number.

Since $\mathrm{abs} \circ Df : \mathbb{R} \to \mathbb{R}$ given by $x \mapsto \left| \left. \frac{\mathrm{d}}{\mathrm{d}t} \right|_{t=x} f(t) \right|$ is a continuous function and $[\alpha - 1, \alpha + 1]$ is a non-empty compact subset of $\mathbb{R}$, $\mathrm{abs} \circ Df$ attains a maximum on $[\alpha - 1, \alpha + 1]$, denote it by M. Then $M > 0$ for otherwise $M = 0$ then $Df(x) = 0$ for all $x \in [\alpha - 1, \alpha + 1]$ implying that f is constant contradicting the degree of f.

```
have H := is_compact.exists_forall_ge
            a_proof_of_[α − 1, α + 1]_compact
            a_proof_of_[α − 1, α + 1]_not_empty
            a_proof_of_abs ∘ Df_continuous,

choose x_max hx_max using H,
generalize M_def: abs (Df_ℝ.eval x_max) = M,
have hM := hx_max.2, rw M_def at hM,
have M_non_zero : M ≠ 0,
  proof_omitted
have M_pos : M > 0,
  proof_omitted
```

Let us consider the smallest element B of the set $\left\{1, \frac{1}{M}\right\} \cup \{|\alpha - x| \,|\, f(x) = 0 \wedge x \neq \alpha\}$, then $B > 0$.

```
generalize roots_def :  f_ℝ.roots = f_roots,
generalize roots'_def : f_roots.erase α = f_roots',
generalize roots_distance_to_α : f_roots'.image (λ x, abs (α -
↪  x)) = distances,
generalize hdistances' : insert (1/M) (insert (1:ℝ) distances) =
↪  distances',
have hnon_empty: distances'.nonempty,
  proof_omitted
generalize hB : finset.min' distances' hnon_empty = B,
have allpos : ∀ x : ℝ, x ∈ distances' → x > 0,
  proof_omitted
have B_pos : B > 0,
  proof_omitted
```

Let $A = \frac{B}{2}$ then $A > B > 0$. We claim that A satisfies the lemma, i.e. $A > 0$ and for every rational number $\frac{a}{b}$, $|\alpha - a/b| > \frac{A}{b^m}$ where m is the degree of f.

```
generalize hA : B / 2 = A,
use A, split,
a_proof_of_A > 0
```

We proceed by assuming that there exists a rational number $\frac{a}{b}$ such that $\left|\alpha - \frac{a}{b}\right| \leq \frac{A}{b^m}$ for a contradiction. Since $b \geq 1$, we have $\left|\alpha - \frac{a}{b}\right| \leq A < B$. Then $\frac{a}{b}$ is not a root of f because otherwise $B \leq \left|\alpha - \frac{a}{b}\right|$.

```
by_contra absurd,
simp only [gt_iff_lt, classical.not_forall, not_lt,
↪  classical.not_imp] at absurd,
choose a ha using absurd,
choose b hb using ha,
have hb2 : b ^ f.nat_degree ≥ 1,
  proof_omitted
have hb21 : abs (α - a / b) ≤ A,
  proof_omitted
have hb22 : abs (α - a/b) < B,
  proof_omitted
have hab0 : (a/b:ℝ) ∈ set.Icc (α-1) (α+1),
  proof_omitted
have hab1 : (a/b:ℝ) ≠ α,
  proof_omitted
have hab2 : (a/b:ℝ) ∉ f_roots,
  proof_omitted
```

Since $\alpha \neq \dfrac{a}{b}$, we can assume without losing generality that $\dfrac{a}{b} < \alpha$. Since $\mathrm{eval}_f : \mathbb{R} \to \mathbb{R}$ given by $x \mapsto f(x)$ is differentiable, we can use mean value theorem to find $x_0 \in \left(\dfrac{a}{b}, \alpha\right)$ such that

$$\begin{aligned} Df(x_0) &= \frac{\mathrm{eval}_f(\alpha) - \mathrm{eval}_f(\frac{a}{b})}{\alpha - \frac{a}{b}} && \text{[Mean value theorem]} \\ &= -\frac{\mathrm{eval}_f(\frac{a}{b})}{\alpha - \frac{a}{b}} && [\alpha \text{ is a root of } i(f)] \end{aligned}$$

```
have hab3 := ne_iff_lt_or_gt.1 hab1,
cases hab3,
have H :=
  exists_deriv_eq_slope (λ x, f_ℝ.eval x) hab3 _ _,
choose x0 hx0 using H,
have hx0r := hx0.2,
rw [polynomial.deriv, hDf, ←hfℝ] at hx0r,
rw [f_eval_on_ℝ] at α_root, rw [α_root, hfℝ] at hx0r, simp only
↪  [zero_sub] at hx0r,
```

Then $|Df(x_0)| > 0$ hence $\left|\alpha - \dfrac{a}{b}\right| = \left|\dfrac{\mathrm{eval}_f(\frac{a}{b})}{Df(x_0)}\right|$ is non-zero. Since M is the maximum of $\mathrm{abs} \circ Df$ on $[\alpha - 1, \alpha + 1]$. We have $|Df(x_0)| \leq M$ and thus $\left|\alpha - \dfrac{a}{b}\right| \geq \dfrac{|\mathrm{eval}_f(\frac{a}{b})|}{M}$. If we write $f(X)$ as $\displaystyle\sum_{j=0}^{m} \lambda_j X^j$ then

$$\left|\mathrm{eval}_f\left(\frac{a}{b}\right)\right| = \left|\sum_{j=0}^{m} \lambda_j \frac{a^j}{b^j}\right| = \frac{1}{b^m}\left|\sum_{j=0}^{m} \lambda_j a^j b^{m-j}\right| \geq \frac{1}{b^m}$$

Hence we have $\left|\alpha - \frac{a}{b}\right| \geq \frac{1}{Mb^m} > \frac{A}{b^m}$. But we assumed $\left|\alpha - \frac{a}{b}\right| < \frac{A}{b^m}$ to start with, this is the desired contradiction.

```
  have Df_x0_nonzero : Df_ℝ.eval x0 ≠ 0,
    proof_omitted
  have H2 : abs(α - a/b) = abs((f_ℝ.eval (a/b:ℝ)) / (Df_ℝ.eval x0)),
    proof_omitted

  have ineq' : polynomial.eval (a/b:ℝ) (polynomial.map ℤembℝ f) ≠ 0,
    proof_omitted
  have ineq : abs (α - a/b) ≥ 1/(M*b^(f.nat_degree)),
    proof_omitted
  have ineq2 : 1/(M*b^(f.nat_degree)) > A / (b^f.nat_degree),
    proof_omitted
  have ineq3 : abs (α - a / b) > A / b ^ f.nat_degree,
    proof_omitted
  have ineq4 : abs (α - a / b) > abs (α - a / b),
    proof_omitted
  linarith,

  We omit the proof of differentiability of eval_f, continuity of abs ∘ Df and the case when a/b > α
  rest_omitted
end
```

□

We then prove the irrationality of Liouville numbers.

Lemma 3.2.2. Every Liouville number is irrational

```
lemma liouville_numbers_irrational:
  ∀ (x : ℝ), (liouville_number x) → irrational x :=
```

Proof. Let x be an arbitrary Liouville number and suppose for a contradiction that $x = \frac{a}{b}$, write $n = b + 1$ then $2^{n-1} > b$.

```
begin
  intros x liouville_x a b hb rid,
  replace rid : x = ↑a / ↑b, linarith,
  generalize hn : b.nat_abs + 1 = n,
  have b_ineq : 2 ^ (n-1) > b,
    proof_omitted
```

Since $x = \frac{a}{b}$ is a Liouville number we can find a rational number $\frac{p}{q}$ such that $q > 1$ and $0 < \left|\frac{a}{b} - \frac{p}{q}\right| < \frac{1}{q^n}$ or equivalently $0 < \frac{|aq - bp|}{bq} < \frac{1}{q^n}$. If $aq - bp = 0$, then $0 < 0$ is the desired contradiction.

```
  choose p hp using liouville_x n,
  choose q hq using hp, rw rid at hq,
  have q_pos : q > 0 := by linarith,
  rw [div_sub_div at hq, abs_div at hq],

  by_cases (abs (a*q-b*p:ℝ) = 0),
  intermediate_step_omitted
  linarith,
```

If $aq - bp \neq 0$ then $\dfrac{1}{bq} \leq \dfrac{|aq - bp|}{bq}$. But we also have $b < 2^{n-1}$ and $2^{n-1}q \leq q^n$ because $q \geq 2$. Hence $bq < q^n$, then $\dfrac{|aq - bp|}{bq} > \dfrac{1}{q^n}$. This is the desired contradiction.

```
  have ineq4 : 1 / (b * q : ℝ) ≤ (abs(a * q - b * p:ℝ)) / (b * q),
    proof_omitted
  have b_ineq'' : (b*q:ℝ) < (2:ℝ)^(n-1)*(q:ℝ),
    proof_omitted
  have q_ineq3 : 2 ^ (n - 1) * q ≤ q ^ n,
    proof_omitted
  have b_ineq2 : b * q < q ^ n, linarith,
  have rid'' :
    abs (a*q-b*p:ℝ) / (b*q:ℝ) > 1/q^n,
    proof_omitted,

  have hq22 := hq2.2,
  linarith,

  We manipulated inequalities involving division and multiplication hence we need to prove several things to be positive.
  proofs_omitted
end
```

□

With the above lemmas, we are ready to prove the transcendence of Liouville numbers.

Theorem 3.2.1. Every Liouville number is transcendental

```
theorem liouville_numbers_transcendental : ∀ x : ℝ,
    liouville_number x → transcendental x :=
```

Proof. Let x be an arbitrary Liouville number then x is irrational. Assume for a contradiction that x is algebraic, let f be the non-zero integer polynomial admitting x as root as a $\mathbb{R}$-polynomial. Then since x is irrational, f has degree at least 2.

```
begin
  intros x liouville_x,
  have irrational_x : irrational x := liouville_numbers_irrational
  ↪  x liouville_x,
  intros rid, rw is_algebraic at rid,
  choose f hf using rid,
  have f_deg : f.nat_degree > 1,
    proof_omitted
```

By using lemma 3.2.1 we can find a real number $A > 0$ such that for any rational number $\frac{p}{q}$, $\left|x - \frac{p}{q}\right| > \frac{A}{q^n}$ where n is the degree of f.

```
  have about_root : f_eval_on_ℝ f x = 0,
    proof_omitted
  choose A hA using about_irrational_root x irrational_x f f_deg
  ↪  about_root,
  have A_pos := hA.1,
```

Since $\mathbb{R}$ is an Archimedean field, we can find an $r \in \mathbb{N}$ such that $\frac{1}{A} \leq 2^r$. Then consider $m := r + n$. Since x is a Liouville number, there is a rational number $\frac{a}{b}$ such that $b > 1$ and $0 < \left|x - \frac{a}{b}\right| < \frac{1}{b^m} = \frac{1}{b^r b^n}$.

```
  have exists_r := pow_big_enough A A_pos,
  choose r hr using exists_r,
  have hr' : 1/(2^r) ≤ A,
    proof_omitted
  generalize hm : r + f.nat_degree = m,
  replace liouville_x := liouville_x m,
  choose a ha using liouville_x,
  choose b hb using ha,

  have ineq : abs (x-a/b:ℝ) < 1/((b:ℝ)^r)*(1/(b:ℝ)^f.nat_degree),
    proof_omitted
```

Since $b \geq 2$, we have $\frac{1}{b^r} \leq \frac{1}{2^r} \leq A$. Thus $\left|x - \frac{a}{b}\right| < \frac{1}{b^r b^n} \leq \frac{A}{b^n}$. This contradicts lemma 3.2.1 stating that $\left|x - \frac{a}{b}\right| > \frac{A}{q^n}$.

```
  have ineq3 : 1/(b:ℝ)^r ≤ A,
    proof_omitted,
  have ineq4 : 1 /(b:ℝ)^r * (1/(b:ℝ)^ f.nat_degree) ≤ (A /
  ↪  (b:ℝ)^f.nat_degree),
    proof_omitted
  have ineq5 :  abs (x - a/b:ℝ) < A/(b:ℝ)^f.nat_degree, linarith,
  have rid := hA.2 a b _, linarith, linarith,
end
```

□

Construction of a Liouville number

Knowing that all Liouville numbers are transcendental, we now focus on constructing a Liouville number

$$\alpha = \sum_{j=0}^{\infty} \frac{1}{10^{j!}}$$

hence obtain a concrete example of transcendental number α.

Lemma 3.2.3. α converges.

Proof. Since for any $n \in \mathbb{N}$ we have $\frac{1}{10^n}$ is none-negative and $\frac{1}{10^n} \leq \frac{1}{10^{n!}}$, we can use comparison test against $\sum_{j=0}^{\infty} \frac{1}{10^j}$ to deduce the convergence of α.

```
def ten_pow_n_fact_inverse (n : ℕ) : ℝ :=
  (1/10)^n.fact
def ten_pow_n_inverse (n : ℕ) : ℝ :=
  (1/10)^n

lemma summable_ten_pow_n_fact_inverse : summable
↪  ten_pow_n_fact_inverse :=
begin
  exact @summable_of_nonneg_of_le _
    ten_pow_n_inverse
    ten_pow_n_fact_inverse
    a_proof_of_ 1/10^n ≥ 0
    a_proof_of_ 1/10^n ≤ 1/10^{n!}
    a_proof_of_∑_{j=0}^∞ 1/10^n _converges,
end

def α := ∑' n, ten_pow_n_fact_inverse n
```

†: In Lean, $\sum'$ is to indicate infinite sum while $\sum$ is for finite sum. □

Lemma 3.2.4. For every $k \in \mathbb{N}$, there exists some $p_k \in \mathbb{N}$ such that

$$\sum_{j=0}^{k} \frac{1}{j^{k!}} = \frac{p_k}{10^{k!}}$$

```
notation `α_k` k := ∑ ii in finset.range(k+1),
↪  ten_pow_n_fact_inverse ii

notation `α_k_rest` k :=
```

```
4    ∑' ii, ten_pow_n_fact_inverse (ii+(k+1))
5
6  theorem α_k_rat (k:ℕ) :
7    ∃ (p:ℕ), α_k k = (p:ℝ)/((10:ℝ)^k.fact) :=
```

Proof. We prove by induction on k. For $k = 0$, the zeroth partial sum is $\frac{1}{10^{0!}} = \frac{1}{10}$. Thus we can pick $p_0 = 1$.

```
8  begin
9    induction k with k IH,
10
11   simp only [ten_pow_n_fact_inverse, pow_one, finset.sum_singleton,
   ↪   finset.range_one, nat.fact_zero],
12   use 1, norm_cast,
```

Assuming that $\sum_{j=0}^{k} \frac{1}{10^{j!}} = \frac{p_k}{10^{k!}}$, let $m := 10^{(k+1)!-k!}$, then we can set $p_{k+1} := p_k m + 1$ then

$$\sum_{j=0}^{k+1} \frac{1}{10^{j!}} = \frac{p_k}{10^{k!}} + \frac{1}{10^{(k+1)!}} = \frac{p_k m + 1}{10^{(k+1)!}} = \frac{p_{k+1}}{10^{(k+1)!}}$$

```
13   choose pk hk using IH,
14   rw α_k at hk ⊢,
15   generalize hm : 10^((k+1).fact - k.fact) = m,
16   generalize hp : pk * m + 1 = p,
17   use p,
18     proof_omitted
19 end
```

† : In line 1 and 4 above, we use `ii` as indexing variable is to avoid clashes.
‡ : `finset.range n` ranges over $\{0, \ldots, n-1\}$. □

Theorem 3.2.2. α is a Liouville number

```
1  theorem liouville_α : liouville_number α :=
```

Proof. We need to prove that for an arbitrary $n \in \mathbb{N}$, there exists a rational number $\frac{p(n)}{q(n)}$ such that $p(n) > 1$ and $0 < \left|\alpha - \frac{p(n)}{q(n)}\right| < \frac{1}{q(n)^n}$. By lemma 3.2.4 We know that for some $p \in \mathbb{N}$,

$$\alpha = \sum_{j=0}^{n} \frac{1}{10^{j!}} + \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} = \frac{p}{10^{n!}} + \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}}.$$

We take $p(n)$ to be p and $q(n)$ to be $10^{n!}$. Then $10^{n!} > 1$, thus it suffices to prove $0 < \left| \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} \right| < \left(\frac{1}{10^{n!}} \right)^n$

```
begin
  intro n,
  have lemma1 := α_k_rat n,
  have lemma2 : (α_k_rest n) = α - α_k n,
    proof_omitted
  choose p hp using lemma1,
  use p, use 10^(n.fact),
  suffices : 0 < abs (α_k_rest n) ∧
    abs (α_k_rest n) < 1/(10^n.fact)^n,
    split,
    a_proof_of_10^{n!} > 1,
    tidy,
  split,
```

Since each summand is strictly positive, $\left| \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} \right| = \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} > 0$. Then we prove $\left| \sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} \right| < \left(\frac{1}{10^{n!}} \right)^n$, or equivalently $\sum_{j=0}^{\infty} \frac{1}{10^{(j+n+1)!}} < \left(\frac{1}{10^{n!}} \right)^n$ instead. Because for all $j \in \mathbb{N}$, $10^j \times 10^{(n+1)!} \leq 10^{(j+(n+1))!}$, we have

$$\sum_{j=0}^{\infty} \frac{1}{10^{(j+(n+1))!}} \leq \sum_{j=0}^{\infty} \left(\frac{1}{10^j} \frac{1}{10^{(n+1)!}} \right) = \frac{1}{10^{(n+1)!}} \sum_{j=0}^{\infty} \frac{1}{10^j}$$
$$= \frac{10}{9} \frac{1}{10^{(n+1)!}} < \frac{2}{10^{(n+1)!}} < \left(\frac{1}{10^{n!}} \right)^n$$

```
  rw [α_k_rest, abs_of_pos (α_k_rest_pos n)],

  have ineq2 :
    (∑' (j:ℕ), ten_pow_n_fact_inverse (j+(n+1))) ≤
    (∑' (i:ℕ), (1/10:ℝ)^i * (1/10:ℝ)^(n+1).fact),
    proof_omitted
  have ineq3 :
    (∑' (i:ℕ), (1/10:ℝ)^i * (1/10:ℝ)^(n.fact*n.succ)) ≤
    ↪  (2/10^n.succ.fact:ℝ),
    proof_omitted
  have ineq4 : (2 / 10 ^ (n.fact*n.succ):ℝ) <
  ↪  (1/((10:ℝ)^n.fact)^n),
    proof_omitted,
  have ineq5 :
```

```
    (∑' (j : ℕ), ten_pow_n_fact_inverse (j+(n+1))) <
    ↪  (1/((10:ℝ)^n.fact)^n),
    proof_omitted
  tidy,
end
```

□

The transcendence of α follows immediately from theorem 3.2.1 and theorem 3.2.2.

Corollary 3.2.1. α is a transcendental number.

```
theorem transcendental_α : transcendental α :=
↪  liouville_numbers_transcendental α liouville_α
```

3.3 Hermite's theorem

Throughout this section f will be an integer polynomial with degree d, and t is a none-negative real number.

Definition 3.3.1. we define

$$I(f,t) := \int_0^t e^{t-x}\mathrm{eval}_f(x)\mathrm{d}x$$

```
def II (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) : ℝ :=
  ∫ x in set.Icc 0 t, real.exp(t-x)*(f_eval_on_ℝ f x)
```

If $f(X) = \sum_{j=0}^{d} \lambda_j X^j$, we define $\bar{f}(X) := \sum_{j=0}^{d} |\lambda_j| X^j$

```
def f_bar (f : ℤ[X]) : ℤ[X] :=
{ support            := f.support,
  to_fun             := λ n, abs (f.coeff n),
  mem_support_to_fun := proof_omitted }
```

†: In Lean, an integer polynomial is a function $\mathbb{N} \to \mathbb{Z}$ with finite support such that for any $n \in \mathbb{N}$ the value of the said function at n is not zero if and only if n is in the support of the said function. Thus to define $\bar{f}$, not only need we to specify the support and the function, a proof of n-th coefficient being non-zero if and only if n being in the support is needed as well.

Let us estimate an upper bound for $|I(f,t)|$ using $\bar{f}$.

Lemma 3.3.1. If $x \in [0,t]$, then $|\text{eval}_f(x)| \leq \text{eval}_{\bar{f}}(t)$

```
lemma f_bar_ineq (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) :
  ∀ x ∈ set.Icc 0 t, abs (f_eval_on_ℝ f x) ≤ f_eval_on_ℝ (f_bar f) t :=
```

Proof. If we write $f(X) = \displaystyle\sum_{j=0}^{d} \lambda_j X^j$, then for any $x \in [0,t]$, we have $|\text{eval}_f(x)| = \left|\displaystyle\sum_{j=0}^{d} \lambda_j x^j\right| \leq \displaystyle\sum_{j=0}^{d} \left|\lambda_j x^j\right|$.

```
  intros x hx,
  have lhs : f_eval_on_ℝ f x = ∑ i in f.support, (f.coeff i : ℝ) * x ^ i,
    proof_omitted
  rw lhs,

  have ineq1 : abs (∑ (i : ℕ) in f.support, (f.coeff i:ℝ) * x ^ i) ≤ ∑ i in f.support, (abs (f.coeff i:ℝ) * (x ^ i)),
    proof_omitted
```

On the right hand side, $\text{eval}_{\bar{f}}(t) = \displaystyle\sum_{j=0}^{d} |\lambda_j|\, t^j$. We conclude by noting that for any $n \in \mathbb{N}$, $x^n \leq t^n$.

```
  have rhs : f_eval_on_ℝ (f_bar f) t = ∑ i in (f_bar f).support, abs (f.coeff i:ℝ) * t ^ i,
    proof_omitted
  rw rhs,

  have ineq2 : ∑ (i : ℕ) in f.support, abs (f.coeff i:ℝ) * x ^ i ≤ ∑ i in (f_bar f).support, abs (f.coeff i:ℝ) * t ^ i,
  {
      suffices : x ^ n ≤ t ^ n,
        proof_omitted
  },
  exact le_trans ineq1 ineq2,
```

$\square$

Theorem 3.3.1.

$$|I(f,t)| \leq te^t \text{eval}_{\bar{f}}(t)$$

```
theorem abs_II_le2 (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) :
  abs (II f t ht) ≤ t*t.exp*(f_eval_on_ℝ (f_bar f) t)
```

Proof.

$$\begin{aligned}
|I(f,t)| &= \left|\int_0^t e^{t-x}\mathrm{eval}_f(x)\mathrm{d}x\right| \\
&\le \int_0^t \left|e^{t-x}\mathrm{eval}_f(x)\right|\mathrm{d}x \\
&\le te^t\mathrm{eval}_{\bar{f}}(t)
\end{aligned}$$

where the last inequality is due to $e^{t-x} \le e^t$ for all $x \in [0,t]$ and lemma 3.3.1.

```
begin
  have ineq1 :
    abs (II f t ht) ≤ ∫ (x : ℝ) in set.Icc 0 t, abs ((t-x).exp *
    ↪  f_eval_on_ℝ f x),
    proof_omitted
  have ineq2 :
    (∫ (x : ℝ) in set.Icc 0 t,
      abs ((t-x).exp * f_eval_on_ℝ f x)) ≤
    t * t.exp * f_eval_on_ℝ (f_bar f) t,
    proof_omitted
  exact le_trans ineq1 ineq2,
end
```

□

Lemma 3.3.2.

$$I(f,t) := \int_0^t e^{t-x}\mathrm{eval}_f(x)\mathrm{d}x = e^t\mathrm{eval}_f(0) - \mathrm{eval}_f(t) + I(f',t)$$

```
lemma II_integrate_by_part
  (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) :
  (II f t ht) = (real.exp t) * (f_eval_on_ℝ f 0) - (f_eval_on_ℝ f
  ↪  t) + (II f.derivative t ht)
```

Proof. Since $e^{t-x} = \frac{\mathrm{d}}{\mathrm{d}x}\left(-e^{t-x}\right)$, we can use integration by part.

```
  rw II,
  have eq :
    (∫ x in set.Icc 0 t,
      (t-x).exp * f_eval_on_ℝ f x) =
    (∫ x in set.Icc 0 t,
      f_eval_on_ℝ f x * (deriv (λ x, -(real.exp (t-x))) x)),
    proof_omitted
  rw eq,
  replace eq := integrate_by_part (f_eval_on_ℝ f) (λ (x : ℝ), -(t -
  ↪  x).exp) 0 t ht,
  intermediate_steps_omitted
  rw eq, ring,
```

□

Lemma 3.3.3. For any $m \in \mathbb{N}$,

$$I(f,t) := \int_0^t e^{t-x} \mathrm{eval}_f(x) \mathrm{d}x = e^t \sum_{j=0}^{m} \mathrm{eval}_{f^{(j)}}(0) - \sum_{j=0}^{m} \mathrm{eval}_{f^{(j)}}(t) + I(f^{(m+1)}, t)$$

```
lemma II_integrate_by_part_m (f : ℤ[X]) (t : ℝ)
  (ht : t ≥ 0) (m : ℕ) :
  II f t ht = t.exp * (∑ i in finset.range (m+1), (f_eval_on_ℝ
  ↪  (deriv_n f i) 0)) - (∑ i in finset.range (m+1), f_eval_on_ℝ
  ↪  (deriv_n f i) t) + (II (deriv_n f (m+1)) t ht) :=
```

Proof. We prove by induction on m. The base case is lemma 3.3.2

```
begin
  induction m with m ih,
  rw [deriv_n, II_integrate_by_part],
  simplification_steps_omitted
```

The inductive steps is to apply lemma 3.3.2 to $f^{(m+1)}$ and regroup.

```
  rw [ih, II_integrate_by_part],
  simplification_steps_omitted
end
```

□

By the previous lemma, we obtain an alternative formulation of $I(f,t)$

Theorem 3.3.2.

$$I(f,t) = e^t \left(\sum_{j=0}^{d} \mathrm{eval}_{f^{(j)}}(0) \right) - \sum_{j=0}^{d} \mathrm{eval}_{f^{(j)}}(t)$$

```
def I (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) : ℝ :=
  t.exp * (∑ i in finset.range f.nat_degree.succ, (f_eval_on_ℝ
  ↪  (deriv_n f i) 0)) - (∑ i in finset.range f.nat_degree.succ,
  ↪  (f_eval_on_ℝ (deriv_n f i) t))

theorem II_eq_I (f : ℤ[X]) (t : ℝ) (ht : t ≥ 0) :
  II f t ht = I f t ht
```

Proof. We use lemma 3.3.3 with $m := d$, the degree of f. Then we get

$$I(f,t) := \int_0^t e^{t-x} \mathrm{eval}_f(x) \mathrm{d}x = e^t \sum_{j=0}^{d} \mathrm{eval}_{f^{(j)}}(0) - \sum_{j=0}^{d} \mathrm{eval}_{f^{(j)}}(t) + I(f^{(d+1)}, t)$$

together with $f^{(d+1)}$ is the zero polynomial so that $I(f^{(d+1)}, t) = 0$.

```
begin
  have II_integrate_by_part_m :=
    II_integrate_by_part_m f t ht f.nat_degree,
  have triv : deriv_n f (f.nat_degree + 1) = 0,
    proof_omitted
  rw I, rw [triv, II_0, add_zero] at II_integrate_by_part_m,
  assumption,
end
```

□

Transcendence of e

To prove the transcendence of e, we will assume the algebraicity for the hope of a contradiction.

Definition 3.3.2. For any prime number p and natural number n, we define an integer polynomial $f_{p,n}(X) := X^{p-1}\prod_{i=1}^{n}(X-i)^p$. For any integer polynomial g with degree n whose i-th coefficient is denoted by g_i, we define $J_p(g) = \sum_{j=0}^{n} g_j I(f_{p,n}, j)$

```
def f_p (p : ℕ) (hp : nat.prime p) (n : ℕ): ℤ[X] :=
  polynomial.X ^ (p - 1) *
  (∏ i in finset.range n,
    (polynomial.X - (polynomial.C (i+1:ℤ)))^p)

def J (g : ℤ[X]) (p : ℕ) (hp : nat.prime p) : ℝ :=
  ∑ i in finset.range g.nat_degree.succ,
    (g.coeff i : ℝ) * (II (f_p p hp g.nat_degree) i (nonneg_nat i))
```

Let us evaluate an upper bound for $J_p(g)$

Theorem 3.3.3. Let g and $f_{p,n}$ be as above. Define

$$M := (d+1)\left(\max\{1, |g_0|, \dots, |g_m|\}(n+1)e^{n+1}\left(2(n+1)\right)^{1+n}\right).$$

Then

$$|J_p(g)| \leq M^p$$

```
def M (g : ℤ[X]) : ℝ :=
  g.nat_degree.succ * ((max_abs_coeff_1 g) * (g.nat_degree+1) *
  ↪  ((g.nat_degree:ℝ)+1).exp *
  ↪  (2*(g.nat_degree+1))^(1+g.nat_degree))

```

```
theorem abs_J_upper_bound
  (g : ℤ[X]) (p : ℕ) (hp : nat.prime p) :
  abs (J g p hp) ≤ (M g)^p :=
```

Proof.

$$
\begin{aligned}
|J_p(g)| &= \left|\sum_{j=0}^{n} g_j I(f_{p,n}, j)\right| && \text{[by definition]}\\
&\le \sum_{j=0}^{n} |g_j I(f_{p,n}, j)| \\
\texttt{ineq1} \quad &\le \sum_{j=0}^{n} |g_j|\, j e^j \operatorname{eval}_{\bar{f}_{p,d}}(j) && \text{[by theorem 3.3.1]}\\
&\le \sum_{j=0}^{n} \max\{1, |g_0|, \ldots, |g_m|\} \cdot (d+1)e^{d+1} \left(j^{p-1} \prod_{i=1}^{n} (j-i)^p\right)\\
&\le \sum_{j=0}^{n} \max\{1, |g_0|, \ldots, |g_m|\} \cdot (d+1)e^{d+1} \left((2d+1)^p \prod_{i=1}^{n} (2d+1)^p\right)\\
\texttt{ineq2} \quad &= (n+1) \left(\max\{1, |g_0|, \ldots, |g_m|\} \cdot (n+1)e^{n+1}\, (2n+1)^{p(1+n)}\right)\\
\texttt{ineq3} \quad &\le (n+1)^p \left(\max\{1, |g_0|, \ldots, |g_m|\}^p \cdot (n+1)^p e^{p(n+1)}\, (2n+1)^{p(1+n)}\right)\\
&= M^p
\end{aligned}
$$

```
theorem abs_J_upper_bound (g : ℤ[X]) (p : ℕ) (hp : nat.prime p) :
↪  abs (J g p hp) ≤ (M g)^p :=
begin
  have ineq1 := abs_J_ineq1'' g p hp,
  have ineq2 := sum_ineq_1 g p hp,
  have ineq3 := sum_ineq_2 g p hp,
  have ineq4 := le_trans (le_trans ineq1 ineq2) ineq3,
  rw [M, mul_pow, mul_pow, mul_pow, mul_pow, ←pow_mul, add_mul,
  ↪  one_mul],
  have eq1 : g.nat_degree * p = p * g.nat_degree, rw mul_comm, rw
  ↪  eq1, exact ineq4,
end
```

†: Later we will see that as long as there exists for some $c > 0$, $|J_p(g)| < c^p$, we can prove the transcendence of e. Thus here M is chosen to be quite rough on purpose to trivialise the small inequalities needing to be proved such as $j^{p-1} < (2(n+1))^p$ for any $j = 0, \ldots, d$. ☐

For lower bound of $J_p(g)$ where $\operatorname{eval}_g(e) = 0$, we need to work with more precision.

Lemma 3.3.4. For any prime number p and natural number n, $f_{p,n}(X)$ has degree $(n+1)p-1$.

```
lemma deg_f_p (p : ℕ) (hp : nat.prime p) (n : ℕ) : (f_p p hp n).nat_degree = (n+1)*p - 1 :=
begin
  proof_omitted
end
```

Theorem 3.3.4. Let $g \in \mathbb{Z}[X]$ with degree n whose i-th coefficient is denoted by g_i such that $\mathrm{eval}_g(e) = 0$. Let $m = (n+1)p-1$. Then

$$J_p(g) = -\sum_{j=0}^{m}\sum_{k=0}^{n} g_k \mathrm{eval}_{f_{p,n}^{(j)}}(k) \tag{3.2}$$

```
theorem J_eq' (g : ℤ[X])
  (e_root_g : (polynomial.aeval ℤ ℝ e) g = 0) (p : ℕ) (hp : nat.prime p) :
  (J g p hp) =
  - ∑ j in finset.range (f_p p hp g.nat_degree).nat_degree.succ,
    (∑ k in finset.range g.nat_degree.succ,
      (g.coeff k : ℝ) * (f_eval_on_ℝ (deriv_n (f_p p hp g.nat_degree) j) (k:ℝ))) :=
```

Proof. We consider the following equalities

$$\begin{aligned}
J_p(g) &= \sum_{k=0}^{n} g_k I(f_{p,n}, k) && [\text{definition}] \\
\texttt{J_eq1} \quad &= \sum_{k=0}^{n} g_k \left[e^k \left(\sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(0) \right) - \sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(k) \right] && [\text{by lemma 3.3.2}] \\
\texttt{J_eq2} \quad &= \sum_{k=0}^{n} g_k e^k \left(\sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(0) \right) - \sum_{k=0}^{n} g_k \sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(k) \\
&= \left(\sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(0) \right) \sum_{k=0}^{n} g_k e^k - \sum_{k=0}^{n} g_k \sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(k) \\
\texttt{J_eq3} \quad &= -\sum_{k=0}^{n} g_k \sum_{j=0}^{m} \mathrm{eval}_{f_{p,n}^{(j)}}(k) && [\mathrm{eval}_g(e) = 0] \\
&= -\sum_{j=0}^{m}\sum_{k=0}^{n} g_k \mathrm{eval}_{f_{p,n}^{(j)}}(k)
\end{aligned}$$

```
begin
  rw [J_eq1, J_eq2, J_eq3, finset.sum_comm],
```

```
  simp only [zero_sub, neg_inj],
  apply congr_arg, ext, rw finset.mul_sum,
  assumption,
end
```

†: Types too long to be displayed with clarity, thus the information about `J_eq`i was moved to the start of the proof. □

The summation in 3.2 actually starts at $j = p - 1$ because of the following lemmas.

Lemma 3.3.5. Let $g, p, n, f_{p,n}$ be like above. If $j < p-1$, then $\mathrm{eval}_{f_{p,n}^{(j)}}(0) = 0$.

```
lemma deriv_f_p_k_eq_zero_k_eq_0_when_j_lt_p_sub_one
  (p : ℕ) (hp : nat.prime p) (n j : ℕ) (hj : j < p-1):
  polynomial.eval 0 (deriv_n (f_p p hp n) j) = 0 :=
```

Proof. Let us agree to write $f_{p,n}(X) = X^{p-1}\Pi_{p,n}$ as a short hand. Then

$$\begin{aligned} f_{p,n}^{(j)}(X) &= \sum_{i=0}^{j} \binom{j}{i} (X^{p-1})^{(j-i)} \Pi_{p,n}^{(i)} \\ \mathrm{eval}_{f_{p,n}^{(j)}}(0) &= \sum_{i=0}^{j} \binom{j}{i} \mathrm{eval}_{(X^{p-1})^{(j-i)}}(0) \mathrm{eval}_{\Pi_{p,n}^{(i)}}(0) \end{aligned} \tag{3.3}$$

We prove that for all $i = 0, \ldots, j$, since $j - i < p - 1$,

$$(X^{p-1})^{(j-i)} = \left(\prod_{k=0}^{j-i-1} (p-1) - k \right) X^{p-1-(j-i)}. \tag{3.4}$$

Thus by substituting 0, we get $\mathrm{eval}_{f_{p,n}^{(j)}}(0) = \sum_{i=0}^{j} \binom{j}{i} 0 = 0$

```
begin
  corresponding to equation (3.3)
  rw [deriv_n_poly_prod, eval_sum',polynomial.eval_mul],
  intermediate_steps_omitted

  corresponding to equation (3.4)
  rw deriv_X_pow',
  rest_omitted
end
```

□

Similarly, we have the following lemma:

Lemma 3.3.6. Let $g, p, n, f_{p,n}$ be like above. If $j < p$, then $\text{eval}_{f_{p,n}^{(j)}}(x) = 0$ for all $1 \le x \le n$.

```
lemma deriv_f_p_when_j_lt_p (p : ℕ) (hp : nat.prime p) (n : ℕ) :
  ∀ x : ℕ, ∀ j : ℕ, j < p → x > 0 → x < n.succ →
    polynomial.eval (x:ℤ) (deriv_n (f_p p hp n) j) = 0 :=
```

Proof. We prove this by induction on n. For $n = 0$, there is no $1 \le x \le 0$, there is nothing to prove.

```
begin
  induction n with n hn,
  intros x j hj hx1 hx2,
  linarith,
```

For the inductive step, assume $\text{eval}_{f_{p,n}^{(j)}}(k) = 0$ for all $1 \le k \le n$. Then for any $1 \le x \le n+1$

$$f_{p,n+1} = f_{p,n}(X - (n+1))^p$$

$$f_{p,n+1}^{(j)} = \sum_{i=0}^{j} \binom{j}{i} f_{p,n}^{(j-i)} \left((X - (n+1))^p\right)^{(i)}$$

$$\text{eval}_{f_{p,n+1}^{(j)}}(x) = \sum_{i=0}^{j} \binom{j}{i} \text{eval}_{f_{p,n}^{(j-i)}}(x) \text{eval}_{((X-(n+1))^p)^{(i)}}(x).$$

We will prove that for any $0 \le y \le j$,

$$\text{eval}_{f_{p,n}^{(j-y)}}(x)\text{eval}_{((X-(n+1))^p)^{(y)}}(x) = 0$$

```
  intros x j hj hx1 hx2,
  rw [f_p_n_succ, deriv_n_poly_prod, eval_sum'],
  apply finset.sum_eq_zero, intros y hy,
```

Here we have that either $x \le n$ or $x = n+1$. For $x \le n$, by inductive hypothesis we have $\text{eval}_{f_{p,n}^{(j-y)}}(x) = 0$ then of course $\text{eval}_{f_{p,n}^{(j-y)}}(x)\text{eval}_{((X-(n+1))^p)^{(y)}}(x) = 0$

```
  cases hx2,
    simp only [int.cast_coe_nat, int.cast_add,
    ↪  ring_hom.eq_int_cast, gt_iff_lt, int.coe_nat_eq_zero,
    ↪  int.cast_one, mul_eq_zero] at *,
    rw IH x (j-y) (gt_of_gt_of_ge hj (nat.sub_le j y)) hx1 hx2,
    tauto,
```

For $x = n+1$, we show $\text{eval}_{((X-(n+1))^p)^{(y)}}(x) = 0$. This is true because

$$((X - (n+1))^p)^{(y)} = \left(\prod_{i=0}^{y}(p-i)\right)(X - (n+1))^{p-y}$$

and $p - y > 0$ hence $0^{p-y} = 0$.

```
  intermediate_steps_omitted
  rw [hx2, deriv_X_sub_pow, polynomial.eval_mul, polynomial.eval_pow],
  simp only [polynomial.eval_X, polynomial.eval_C, int.coe_nat_succ, polynomial.eval_sub, sub_self, mul_eq_zero],
  right, apply zero_pow (nat.sub_pos_of_lt (gt_of_ge_of_gt hj hy))
  exact le_of_lt (gt_of_ge_of_gt hj hy),
```

□

Combining the previous lemmas 3.3.5 and 3.3.6, we have the following corollary.

Corollary 3.3.1. Let $g, p, n, f_{p,n}$ be like above. If $j < p-1$, then $\text{eval}_{f_{p,n}^{(j)}}(k) = 0$ for all $0 \le k \le n$.

Thus

$$\sum_{j=0}^{p-2} \sum_{k=0}^{n} g_k \text{eval}_{f_{p,n}^{(j)}}(k) = 0$$

```
theorem deriv_f_p_k_eq_zero_when_j_lt_p_sub_one
  (p : ℕ) (hp : nat.prime p) (n j : ℕ)
  (hj : j < p - 1) (k : ℕ)
  (hk : k ∈ finset.range n.succ):
  polynomial.eval (k:ℤ) (deriv_n (f_p p hp n) j) = 0 :=
begin
  cases k,
  exact deriv_f_p_k_eq_zero_k_eq_0_when_j_lt_p_sub_one p hp n j hj,
  apply deriv_f_p_when_j_lt_p p hp n k.succ j (nat.lt_of_lt_pred hj) (nat.succ_pos k) (finset.mem_range.mp hk),
end

theorem J_partial_sum_from_one_to_p_sub_one
  (g : ℤ[X]) (p : ℕ) (hp : nat.prime p) :
  ∑ (j : ℕ) in finset.range (p - 1),
  ∑ (k : ℕ) in finset.range g.nat_degree.succ,
    g.coeff k * polynomial.eval ↑k (deriv_n (f_p p hp g.nat_degree) j) = 0 :=
begin
  rw finset.sum_eq_zero, intros, rw finset.sum_eq_zero, intros,
  rw mul_eq_zero, right,
  rw deriv_f_p_k_eq_zero_when_j_lt_p_sub_one, simp only [finset.mem_range] at H, exact H, exact H_1,
end
```

When $j = p - 1$, we can express $\text{eval}_{f_{p,n}^{(p-1)}}(0)$ in a closed form.

Theorem 3.3.5. Let $g, p, n, f_{p,n}$ be like above. Then

$$\text{eval}_{f_{p,n}^{(p-1)}}(0) = (p-1)!(-1)^{np}(n!)^p$$

```
theorem deriv_f_p_zero_when_j_eq_p_sub_one
  (p : ℕ) (hp : nat.prime p) (n : ℕ) :
  polynomial.eval 0 (deriv_n (f_p  p hp n) (p-1)) =
  (p-1).fact * (-1)^(n*p)*(n.fact)^p :=
```

Proof. We have the following equalities:

$$
\begin{aligned}
f_{p,n}^{(p-1)}(X) &= \sum_{i=0}^{p-1} \binom{p-1}{i} (X^{p-1})^{(p-1-i)} \Pi_{p,n}^{(i)} \\
\mathrm{eval}_{f_{p,n}^{(p-1)}}(0) &= \sum_{i=0}^{p-1} \binom{p-1}{i} \mathrm{eval}_{(X^{p-1})^{(p-1-i)}}(0)\mathrm{eval}_{\Pi_{p,n}^{(i)}}(0) \\
&= \binom{p-1}{0} \mathrm{eval}_{(X^{p-1})^{(p-1)}}(0)\mathrm{eval}_{\Pi_{p,n}}(0)
\end{aligned}
$$

where the last equality is due to $\mathrm{eval}_{(X^{p-1})^{(p-1-i)}}(0) = 0$ for $i \neq 0$.

```
begin
  rw [f_p, deriv_n_poly_prod, eval_sum'],
  rw finset.sum_eq_single 0,

  a_proof_of_$\binom{p-1}{0}\mathrm{eval}_{(X^{p-1})^{(p-1)}}(0)\mathrm{eval}_{\Pi_{p,n}}(0) = (p-1)!(-1)^{np}(n!)^p$

  a_proof_of_$\mathrm{eval}_{(X^{p-1})^{(p-1-i)}}(0) = 0$ for all $i \neq 0$
end
```

□

Combine theorem 3.3.5 with lemma 3.3.6, we get the following corollary:

Corollary 3.3.2. Let $g, p, n, f_{p,n}$ be like above. Then

$$\sum_{k=0}^{n} g_k \mathrm{eval}_{f_{p,n}^{(p-1)}}(k) = g_0(p-1)!(-1)^{np}(n!)^p$$

```
theorem J_partial_sum_from_p_sub_one_to_p
  (g : ℤ[X]) (e_root_g : (polynomial.aeval ℤ ℝ e) g = 0)
  (p : ℕ) (hp : nat.prime p) :
  ∑ (k : ℕ) in finset.range g.nat_degree.succ, g.coeff k *
  ↪  polynomial.eval ↑k (deriv_n (f_p p hp g.nat_degree) (p - 1))
  ↪  =
   g.coeff 0 * (↑((p - 1).fact) * (-1) ^ (g.nat_degree * p) *
    ↪  ↑(g.nat_degree.fact) ^ p) :=
begin
  rw finset.sum_eq_single 0,

  simp only [int.coe_nat_zero],
```

```
  rw deriv_f_p_zero_when_j_eq_p_sub_one p hp g.nat_degree,

  intros i hi1 hi2, rw mul_eq_zero, right,
  apply deriv_f_p_when_j_lt_p p hp g.nat_degree,
  rest_omitted
end
```

The final piece of the puzzle is to evaluate $f_{p,n}^{j}$ when $j \geq p$. We first consider when $k = 0$:

Lemma 3.3.7. Let $g, p, n, f_{p,n}$ be like above. Then if $j \geq p$ then $p! \mid \mathrm{eval}_{f_{p,n}^{(j)}}(0)$.

```
lemma k_eq_0_case_when_j_ge_p (p : ℕ) (hp : nat.prime p) (n:ℕ) :
  ∀ j : ℕ, j ≥ p → (p.fact:ℤ) ∣ polynomial.eval 0 (deriv_n (f_p p
  ↪  hp n) j) :=
```

Proof. Using equation 3.3, we need to prove that for all $0 \leq x \leq j$,

$$p! \mid \binom{j}{x} \mathrm{eval}_{(X^{p-1})^{(j-x)}}(0) \mathrm{eval}_{\Pi_{p,n}^{(x)}}(0)$$

```
begin
  rw f_p, intros j j_ge_p, rw [deriv_n_poly_prod, eval_sum'],
  apply finset.dvd_sum, intros x hx,
  simp only [polynomial.eval_C, polynomial.C_add, polynomial.C_1,
  ↪  polynomial.eval_mul, nat.fact],
```

If $j - x = p - 1$, then $\mathrm{eval}_{(X^{p-1})^{(j-x)}}(0) = (p-1)!$, so it suffices to prove that $p \mid \mathrm{eval}_{\Pi_{p,n}^{(x)}}(0)$. In this case, $x \neq 0$, otherwise $j = p - 1 > p$. For $x \geq 1$,

$$\Pi_{p,n}^{(x)} = \left(\left(\prod_{i=1}^{n} (X - i) \right)^{p} \right)^{(x)} = p \left(\left(\prod_{i=1}^{n} (X - i) \right)^{p-1} \left(\prod_{i=1}^{n} (X - i) \right)' \right)^{(x-1)} \tag{3.5}$$

```
  by_cases j - x = p - 1,
  rw [h, deriv_X_pow'], simp only [mul_one, polynomial.eval_C,
  ↪  nat.sub_self, pow_zero],
  rw ←fact_eq_prod',
  suffices :
    (p:ℤ) ∣ polynomial.eval 0 (deriv_n (∏ (x : ℕ) in finset.range
    ↪  n, (polynomial.X - (polynomial.C ↑x + 1)) ^ p) x),
    proof_omitted
  cases x,
    simplification_omitted, linarith,
    rw finset.prod_pow,
    apply dvd_poly_pow_deriv, corresponding to equation 3.5
```

If $j - x \neq p - 1$, either $j - x < p - 1$ or $j - x > p - 1$. If $j - x < p - 1$ then $\text{eval}_{(X^{p-1})^{j-x}}(0) = 0$; if $j - x > p - 1$, $(X^{p-1})^{j-x}$ is the zero polynomial.

```
replace h : j - x < p - 1 ∨ j - x > p - 1, exact lt_or_gt_of_ne
↪  h,
cases h,
  rw [(deriv_X_pow' (p-1) (j-x) (le_of_lt h)),
  ↪  polynomial.eval_mul],
  simp only [polynomial.eval_X, polynomial.eval_C,
  ↪  polynomial.eval_pow],
  rw (rw (zero_pow (nat.sub_pos_of_lt h))),
  simp only [zero_mul, mul_zero, dvd_zero],

  rw deriv_X_pow_too_much,
  simp only [zero_mul, mul_zero, polynomial.eval_zero, dvd_zero],
  assumption,
```

□

Lemma 3.3.8. Let $g, p, n, f_{p,n}$ be like above. Then $p! \mid \text{eval}_{\Pi_{p,n}^{(j)}}(k)$ for any $0 < k \leq n$ and any $p > 0$, prime or composite.

```
lemma p_fact_dvd_prod_part (n : ℕ) :
∀ j : ℕ, ∀ k : ℕ, ∀ p : ℕ, p > 0 → k > 0 → k < n.succ →
(p.fact:ℤ) ∣ polynomial.eval (k:ℤ) (deriv_n (∏ i in finset.range
↪  n, (polynomial.X - polynomial.C (↑i + 1))^p) j) :=
```

Proof. We proceed by using strong induction on j. For $j = 0$ we need to prove $p! \mid \text{eval}_{\Pi_{p,n}^{(0)} p,n}(k) = \text{eval}_{\Pi_{p,n}}(k)$ for any $0 < k \leq n$. This is true because $\text{eval}_{\Pi_{p,n}}(k) = 0$

```
intros j,
apply nat.case_strong_induction_on j,
intros k p hp hk1 hk2,
rw zeroth_deriv, simp only [int.cast_coe_nat, int.cast_add,
↪  ring_hom.eq_int_cast, int.cast_one, nat.fact],
suffices : polynomial.eval (k:ℤ) (∏ (i : ℕ) in finset.range n,
↪  (polynomial.X - (↑i + 1)) ^ p) = 0,
  rw this, exact dvd_zero ↑(nat.fact p),
a_proof_of_eval_Π_{p,n}(k) = 0
```

For inductive case, we assume $p! \mid \text{eval}_{\Pi_{p,n}^{(m)}}(k)$ for all $m \leq j$, $p > 0$ and $0 < k \leq$

n. This is certainly true for $p = 1$, for $p! = 1$. For $p > 1$,

$$\begin{aligned}
\Pi_{p,n}^{(j+1)} &= \left(\left(\left(\prod_{i=0}^{n}(X-(i+1))\right)^{p}\right)'\right)^{(j)} \\
&= p\left(\left(\prod_{i=0}^{n}(X-(i+1))\right)^{p-1}\left(\prod_{i=0}^{n}(X-(i+1))\right)'\right)^{(j)} \\
&= p\sum_{i=0}^{j}\binom{j}{i}\Pi_{p-1,n}^{(j-i)}\left(\Pi_{1,n}'\right)^{(i)}
\end{aligned}$$

By inductive hypothesis $(p-1)! \mid \Pi_{p-1,n}^{j-i}$ for any $i = 0, \dots, j$. Thus $p! \mid \Pi_{p,n}^{(j+1)}$

```
  intros j IH k p hp hk1 hk2,
  rw [deriv_n, function.iterate_succ_apply, ←deriv_n,
  ↪  finset.prod_pow, poly_pow_deriv, deriv_n_poly_prod,
  ↪  eval_sum'],
  apply finset.dvd_sum,
  intros x hx,
  by_cases (p=1), rw h, norm_num,

  replace IH := IH (j-x) _ k (p-1) _ hk1 hk2,
  intermediate_steps_omitted,
  exact IH,
```

□

Immediately bys lemma 3.3.7, 3.3.8 and equation 3.3, we have:

Corollary 3.3.3. Let $g, p, n, f_{p,n}$ be like above. If $j \geq p$ then for all $0 \leq k \leq n$ we have $p! \mid \mathrm{eval}_{f_{p,n}^{(j)}}(k)$. Then

$$p! \mid \sum_{j=p}^{m}\sum_{k=0}^{n} g_k \mathrm{eval}_{f_{p,n}^{(j)}}(k)$$

```
lemma k_ge_1_case_when_j_ge_p (p : ℕ) (hp : nat.prime p) (n:ℕ) :
  ∀ j : ℕ, j ≥ p → ∀ k : ℕ, k < n.succ → k > 0 → (p.fact:ℤ) |
  ↪  polynomial.eval (k:ℤ) (deriv_n (f_p p hp n) j) :=
begin
  intros j hj k hk1 hk2,
  rw [f_p, deriv_n_poly_prod, eval_sum'], apply finset.dvd_sum,
  intros x hx,
  rw polynomial.eval_mul, rw polynomial.eval_mul,
  apply dvd_mul_of_dvd_right,
  apply p_fact_dvd_prod_part n _ _ _ (nat.prime.pos hp) hk2 hk1,
end

theorem when_j_ge_p_k (p : ℕ) (hp : nat.prime p) (n:ℕ) :
```

```
  ∀ j : ℕ, j ≥ p → ∀ k : ℕ, k ∈ finset.range n.succ → (p.fact:ℤ)
  ↪  | polynomial.eval (k:ℤ) (deriv_n (f_p p hp n) j) :=
begin
  intros j j_ge_p k hk,
  simp only [finset.mem_range] at hk,
  cases k,
    exact k_eq_0_case_when_j_ge_p p hp n j j_ge_p,
    exact k_ge_1_case_when_j_ge_p p hp n j j_ge_p k.succ hk
     ↪  (nat.succ_pos k),
end

theorem J_partial_sum_rest (g : ℤ[X]) (e_root_g : (polynomial.aeval
↪  ℤ ℝ e) g = 0) (p : ℕ) (hp : nat.prime p) :
  (p.fact:ℤ) |
    ∑ (j : ℕ) in finset.Ico p (f_p p hp
    ↪  g.nat_degree).nat_degree.succ,
    ∑ (k : ℕ) in finset.range g.nat_degree.succ, g.coeff k *
    ↪  polynomial.eval (k:ℤ) (deriv_n (f_p p hp g.nat_degree) j)
    ↪  :=
begin
  apply finset.dvd_sum, intros x hx,
  apply finset.dvd_sum, intros y hy,
  apply dvd_mul_of_dvd_right,
  apply when_j_ge_p_k, simp only [finset.Ico.mem] at hx,
  exact hx.1, exact hy,
end
```

We finally have everything we need to evaluate equation 3.2: by previous corollaries 3.3.1 3.3.2 we have:

Corollary 3.3.4. Let $g, p, n, f_{p,n}$ be like above, there is some $M \in \mathbb{Z}$,

$$J_p(g) = -g_0(p-1)!(-1)^{np}(n!)^p + p! \cdot M$$

```
theorem J_eq_final
  (g : ℤ[X]) (e_root_g : (polynomial.aeval ℤ ℝ e) g = 0)
  (p : ℕ) (hp : nat.prime p) :
  ∃ M : ℤ, (J g p hp) = ℤembℝ ((-(g.coeff 0 * (↑((p - 1).fact) *
  ↪  (-1) ^ (g.nat_degree * p) * ↑(g.nat_degree.fact) ^ p))) +
  ↪  (p.fact:ℤ) * M) :=
begin
  have J_eq := J_eq'' g e_root_g p hp, rw J_eq, rw
  ↪  ←ring_hom.map_neg,
  have seteq : finset.range (f_p p hp g.nat_degree).nat_degree.succ
  ↪  = finset.range (p-1) ∪ {p-1} ∪ finset.Ico p (f_p p hp
  ↪  g.nat_degree).nat_degree.succ,
    proof_omitted
  rw seteq, rw finset.sum_union, rw finset.sum_union,
  rw J_partial_sum_from_one_to_p_sub_one g, rw zero_add, rw
  ↪  finset.sum_singleton,
  rw J_partial_sum_from_p_sub_one_to_p g e_root_g,

  have H3 := J_partial_sum_rest g e_root_g p hp,
  rw dvd_iff_mul at H3,
```

```
  choose c eq3 using H3,
  rw eq3, rw neg_add, use -c, rw neg_mul_eq_mul_neg ↑(p.fact),

  rest_omitted
end
```

We are now ready to prove a lower bound for $|J_p(g)|$.

Theorem 3.3.6. Let $g, p, n, f_{p,n}$ be like above, if we further assume $g_0 \neq 0$, $p > n$ and $p > |g_0|$. Then $(p-1)! \leq |J_p(g)|$

```
theorem abs_J_lower_bound
  (g : ℤ[X]) (e_root_g : (polynomial.aeval ℤ ℝ e) g = 0)
  (coeff_nonzero : (g.coeff 0) ≠ 0)
  (p : ℕ) (hp : nat.prime p)
  (hp2 : p > g.nat_degree ∧ p > (g.coeff 0).nat_abs) :
  ((p-1).fact:ℝ) ≤ (abs (J g p hp)) :=
```

Proof. By the previous theorem, for some $c \in \mathbb{Z}$

$$|J_p(g)| = (p-1)! \, |(-g_0(-1)^{np}(n!)^p) + pc| \, .$$

Thus to prove $|J_p(g)| > (p-1)!$, we prove $|(-g_0(-1)^{np}(n!)^p) + pc| \geq 1$, equivalently, $(-g_0(-1)^{np}(n!)^p) + pc \neq 0$. Let us assume otherwise, i.e. assume $(-g_0(-1)^{np}(n!)^p) + pc = 0$

```
  simplification_steps_omitted
  intro rid,
```

Then since $p \mid 0$, we have $p \mid (-g_0(-1)^{np}(n!)^p) + pc$ then $p \mid (-g_0(-1)^{np}(n!)^p)$.

```
  have rid2 : (p:ℤ) ∣ g.coeff 0 * ((-1) ^ (g.nat_degree * p) *
  ↪  -↑(g.nat_degree.fact ^ p)) + ↑p * c,
  rw rid, exact dvd_zero ↑p,

  replace rid2 : (p:ℤ) ∣ g.coeff 0 * ((-1) ^ (g.nat_degree * p) *
  ↪  -↑(g.nat_degree.fact ^ p)),
    refine (dvd_add_iff_left _).2 rid2, exact dvd.intro c rfl,
```

Assume $(-1)^{np} = 1$, then we have $p \mid g_0(n!)^p$, then either $p \mid |g_0|$ or $p \mid (n!)^p$. If $p \mid |g_0|$ then $p \leq |g_0|$.

```
  have triv : (-1:ℤ) ^ (g.nat_degree * p) = 1 ∨ (-1:ℤ) ^
  ↪  (g.nat_degree * p) = -1 := neg_one_pow_eq_or _,
  cases triv,
  simplification_steps_omitted
  rw nat.prime.dvd_mul at rid2,
  cases rid2,
    simplification_steps_omitted
    have hm : p*m = (g.coeff 0).nat_abs, proof_omitted,
    replace hm : p ≤ (g.coeff 0).nat_abs, proof_omitted,
```

If $p \mid (n!)^p$, since p is a prime number, $p \mid n!$ implies $p \leq n$.

```
  intermediate_steps_omitted
  have H : p | ↑(g.nat_degree.fact).nat_abs,
  rw nat.prime.dvd_fact at H,
```

For $(-1)^{np} = -1$, the same proof works mutatis mutandis,

```
  rest_omitted
end
```

□

To fullfill the requirement of g having a non-zero coefficient, we divide g by a suitable power of X as following:

```
def min_degree_term (f : ℤ[X]) (hf : f ≠ 0) : ℕ :=
  finset.min' (f.support) (non_empty_supp f hf)
def make_const_term_nonzero (f : ℤ[X]) (hf : f ≠ 0) : ℤ[X] :=
{ support := finset.image (λ i : ℕ, i-(min_degree_term f hf)) f.support,
  to_fun := (λ n, (f.coeff (n+(min_degree_term f hf)))),
  mem_support_to_fun := begin
    intro n, split, intro hn, rw finset.mem_image at hn, choose a ha using hn, rw ←ha.2, rw nat.sub_add_cancel,
    have eq2 := (f.3 a).1 ha.1, exact eq2,
    rw min_degree_term, exact finset.min'_le f.support (non_empty_supp f hf) a ha.1,
    intro hn, rw finset.mem_image, use n + min_degree_term f hf,
    split,
    exact (f.3 (n + min_degree_term f hf)).2 hn, simp only [nat.add_sub_cancel],
  end,}
```

In other words, we divide g by X^m where m is the degree of the lowest non-zero monomial of g.

Because for any $x \geq 0$, $\lim_{n\to\infty} \dfrac{x^n}{n!} = 0$ and there is an infinite amount of primes, we have the following theorem serving the coup de grace of attacking the algebraicity of e.

Theorem 3.3.7. For any integer z and non-negative real number M, there is some prime number $p > z$ such that $(p-1)! > M^p$

```
theorem coup_de_grace (M : ℝ) (hM : M ≥ 0) (z : ℤ) : ∃ p : nat.primes, (p.val:ℤ) > z ∧ ((p.val-1).fact:ℝ) > M^p.val
```

Theorem 3.3.8. e is transcendental.

```
theorem e_transcendental : transcendental e :=
```

Proof. We prove by contradiction. Assume e is algebraic then there is an integer polynomial g such that $\text{eval}_g(e) = 0$. Divide g by some suitable power of X if necessary, we can assume that g has a non-zero constant coefficent.

```
begin
  by_contra e_algebraic,
  rw is_algebraic at e_algebraic,
  choose g' g'_def using e_algebraic,
  have g'_nonzero := g'_def.1,
  have e_root_g' := g'_def.2,
  generalize g_def : make_const_term_nonzero g' g'_nonzero = g,
  have coeff_zero_nonzero : (g.coeff 0) ≠ 0,
    rw ←g_def, apply coeff_zero_after_change,
  have e_root_g : (polynomial.aeval ℤ ℝ e) g = 0,
    rw ←g_def,
    apply non_zero_root_same, rw e, exact (1:ℝ).exp_ne_zero, exact
    ↪  e_root_g',
```

There is a prime number p such that $p > n$, $p > |g_0|$ and $(p-1)! > M^p$ where M is defined as in theorem 3.3.3. Then $(p-1)! > M^p \geq |J_p(g)| > (p-1)!$ is the desired contradiction.

```
  have contradiction := contradiction (M g) _ (max g.nat_degree
  ↪  (abs (g.coeff 0))),
  choose p Hp using contradiction,
  have abs_J_lower_bound := abs_J_lower_bound g e_root_g
  ↪  coeff_zero_nonzero p.val p.property _,
  have rid := le_trans abs_J_lower_bound abs_J_upper_bound,
  simplification_steps_omitted
```

□

Conclusion and future work

This project lives in an interdisciplinary area between mathematics and computer science. Even interactive theorem proving is not necessarily suitable for every mathematician, its pedagogical value is enormous because by formalising theorems the logic of proofs are disentangled and one is forced to be precise and explicit. Hence a proof in `Lean` often challenge users to prove intuitions in a rigourous manner. For example, Alan Baker deemed the evaluation of $J_p(g)$ in equation 3.2 to be clear, but by formalising one has to get hands dirty by actually performing the evaluation [Bak90]. Then anyone else interested in the proof can choose to read or discard relavent sections depending on whether she/he shares the same intuitions with the author.

This is perhaps both a blessing and a curse – by forbidding to use words like "trival" or "clearly", one often finds her/himself proving truly trivial propositions as well, for example to prove that $\mathbb{Z}^1 \simeq \mathbb{Z}$. Hence there are at least two directions in which future works could take. One is to formalise more theorems of interest for example the transcendence of π; the other is to use the (meta-)programming facilities of `Lean` to make more tactics so that learning and using `Lean` could be more effortless.

Bibliography

[AMK15] Jeremy Avigad, Leonardo de Moura, and Soonho Kong. *Theorem proving in Lean*. 2015.

[AMR16] Jeremy Avigad, Leonardo de Moura de Moura, and Jared Roesch. *Programming in Lean*. 2016.

[Bak90] Alan Baker. *Transcendental number theory*. Cambridge university press, 1990.

[BCM20] Kevin Buzzard, Johan Commelin, and Patrick Massot. "Formalising perfectoid spaces". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2020, pp. 299–312.

[Bou98] Nicolas Bourbaki. *Elements of the History of Mathematics*. Springer Science & Business Media, 1998.

[Boy+94] Robert Boyer et al. "The QED manifesto". In: *Automated Deduction–CADE* 12 (1994), pp. 238–251.

[Can32] G Cantor. *Uber eine Eigenschaft des lnbegriffes aller reelen algebraischen Zahlen. Crelles J. 77 (1874), 258-262; also in Gesammelte A bhandlungen*. 1932.

[Can78] Georg Cantor. "Ein beitrag zur mannigfaltigkeitslehre". In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1878.84 (1878), pp. 242–258.

[Gel34] Aleksandr Gelfond. "Sur le septieme probleme de Hilbert". In: *Известия Российской академии наук. Серия математическая* 4 (1934), pp. 623–634.

[Gon+13] Georges Gonthier et al. "A machine-checked proof of the odd order theorem". In: *International Conference on Interactive Theorem Proving*. Springer. 2013, pp. 163–179.

[Gon08] Georges Gonthier. "Formal proof–the four-color theorem". In: *Notices of the AMS* 55.11 (2008), pp. 1382–1393.

[Hal+17] Thomas Hales et al. "A formal proof of the Kepler conjecture". In: *Forum of mathematics, Pi*. Vol. 5. Cambridge University Press. 2017.

[Hal98] Thomas C Hales. "The kepler conjecture". In: *arXiv preprint math.MG/9811078* (1998).

[HUW14] John Harrison, Josef Urban, and Freek Wiedijk. "History of Interactive Theorem Proving." In: *Computational Logic*. Vol. 9. 2014, pp. 135–214.

[Kem16] Aubrey J. Kempner. "On Transcendental Numbers". In: *Transactions of the American Mathematical Society* 17.4 (1916), pp. 476–482. ISSN: 00029947. URL: http://www.jstor.org/stable/1988833.

[KK11] Juliette Kennedy and Roman Kossak. *Set Theory, Arithmetic, and Foundations of Mathematics: Theorems, Philosophies*. Vol. 36. Cambridge University Press, 2011.

[Lam04] M Lambert. "Mémoire sur quelques propriétés remarquables des quantités transcendentes circulaires et logarithmiques". In: *Pi: A Source Book*. Springer, 2004, pp. 129–140.

[Lan66] Serge Lang. *Introduction to transcendental numbers*. Addison-Wesley Pub. Co., 1966.

[Zha20] Jujian Zhang. *e is a transcendental number*. 2020. URL: https://jjaassoonn.github.io/e_transcendence_doc.html.

[Zor00] Vladimir A Zorich. *Mathematical Analysis I, volume 1. Universitext*. 2000.