形式化数学：Lean 入门与应用

张巨见[1] [2]

2026 年 8 月 19 日

[1] joz19@ic.ac.uk

[2] 刘俊杞，支丽红协助修改整理 liujunqi@amss.ac.cn，lzhi@mmrc.iss.ac.cn

目录

第一部分

基础理论

第一章　Lean 的语法速览

我们在本章节简单介绍 Lean 的语法[1]目的是让读者在阅读后面章节和相关代码时做好准备。我们将通过在 Lean 中如何描述和证明素数有无穷多（1.1）来学习 Lean 的一部分重要语法。代码片段 1.1可以在 PartOne/ChapterOne/Example.lean 中找到。

```
1  import Mathlib
2
3  variable (n : ℕ)
4
5  def nextPrime : ℕ := Nat.minFac (Nat.factorial n + 1)
6
7  lemma nextPrime_prime : Nat.Prime (nextPrime n) := by
8    fapply Nat.minFac_prime
9    simp only [ne_eq, Nat.add_eq_right]
10   exact Nat.factorial_ne_zero n
11
12 lemma nextPrime_ge : n ≤ nextPrime n := by
13   by_contra! ineq
14   have h₁ :  nextPrime n ∣ Nat.factorial n := by
15     refine Nat.dvd_factorial ?_ ?_
16     · exact Nat.minFac_pos ..
17     · linarith
18   have h₂ : nextPrime n ∣ 1 := by
19     rw [Nat.dvd_add_iff_right h₁]
20     exact Nat.minFac_dvd ..
21   have h₃ : ¬ nextPrime n ∣ 1 := Nat.Prime.not_dvd_one (nextPrime_prime n)
22   contradiction
23
24
25 theorem exists_infinite_primes : ∃ p, n ≤ p ∧ Nat.Prime p := by
26   use nextPrime n
27   fconstructor
```

[1]关于 Lean 的全部语法，见 [1]。

```
28    • exact nextPrime_ge n
29    • exact nextPrime_prime n
```

代码片段 1.1: 无穷多素数

1.1 变量，定义和定理

在本节中，我们将阅读代码片段 1.1中的定义、引理、定理和证明，以学习如何声明变量，定义函数和定理。

我们用variable关键字来引入新的变量，如在代码片段 1.1中的第三行，(n : ℕ)可以理解为“让 n 是任意一个自然数”。关键字def是用来引入新的定义的，格式如下：

```
def 定义名称 (输入参数 : 类型) ... : 输出类型 := 定义内容
```

其中输入参数的类型是数学概念上函数的定义域，输出类型对应函数的值域。如在第五行，对于任意自然数 n，我们定义nextPrime n 为 $n!+1$ 的最小素因子[2]；再比如若要定义平方函数则可写作

```
def square (x : ℝ) : ℝ := x^2
```

关键字 def 不仅可以用来定义数值，函数，它也可以用来定义命题、性质等，例如刻画自然数中的素数这一性质Nat.Prime在 Lean 中可以这样实现：

```
def Nat.Prime (p : ℕ) : Prop := 2 ≤ p ∧ ∀ m, m ∣ p → m = 1 ∨ m = p
```

关键字lemma和theorem 用来引入引理和定理的。引理和定理在 Lean 中并无区别，一般人们用theorem来标记更重要的结果 — 这样做通常是为了增加代码的可读性，读者可以一目了然地看到代码的哪些内容更为重要。二者的格式如下：

```
lemma 引理名称 (输入参数 : 类型) ... : 命题 := by
  证明
theorem 定理名称 (输入参数 : 类型) ... : 命题 := by
  证明
```

比如代码片段 1.1第 7 行和第 12 行分别引入了名为nextPrime_prime和nextPrime_ge的引理，其内容是对于任意自然数 n，nextPrime n均为素数且nextPrime n大于等于 n。由此两条引理，我们得出了一个名为 exists_infinite_primes 的定理，其内容是，对于任意自然数 n，都存在一个不小于 n 的素数 p。

Lean 中还提供了example关键字，可以用来定义例子。它可以给出无需名字的定义，定理等：

```
example (输入参数 : 类型) ... : 输出类型 := ...
```

[2]Nat.minFac和Nat.factorial是 Mathlib 中已经定义好的函数，分别用于求最小素因子和阶乘。

一般example用于需要写一些快速的实验性的或者以后不会用到代码，所以不要给这些代码起名字。

选读：def 和 lemma 的区别

在 Lean 中，def 和 lemma是有本质区别的，使用def时，输出类型（即冒号:右边到:=之间的内容）可以是任何类型，但是对于lemma 或 theorem，这一部分必须是命题：

```
lemma addOne (x : ℕ) : ℕ := x + 1
```

此代码片段会报错，Lean 返回提示错误信息：

```
type of theorem 'addOne' is not a proposition ℕ → ℕ
```

这是因为addOne输出一个自然数，而不是一个命题。

1.2 如何写形式化证明—基础 tactic 简介

在本节中，我们将通过分析代码片段 1.1的证明部分来学习如何在 Lean 中实现定理的形式化证明。注意到在lemma和theorem的叙述中，它们都是以by 结尾的。在 Lean 中by用来标记进入 tactic 模式，在此模式下用户可以用 tactic 来贴合自然语言书写形式化证明，结合 info view，用户可以清晰看出证明每一步的作用。我们以第 7 行的nextPrime_prime来举例。在第八行开始时，Lean 显示

```
1 goal                                  还有一个目标
n : ℕ                                   已知：n 为自然数
⊢ Nat.Prime (nextPrime n)               想证：nextPrime n 为素数
```

第 8 行使用了fapply 某引理，直观的来讲 fapply在数学上对应着根据“某引理，只需证明...”。此处 Nat.minFac_prime陈述如下：

```
theorem Nat.minFac_prime {n : ℕ} (n1 : n ≠ 1) : Nat.Prime (Nat.minFac n)
```

即,对于任意自然数 n,如果 n 不为 1, 则 n 的最小素因子是素数[3]。根据 Nat.minFac_prime内容，在fapply Nat.minFac_prime后（第 9 行），Lean 告诉你：

```
1 goal                                  还有一个目标
case n1                                 n1 是 Nat.minFac_prime 中假设 n ≠ 1 的名称
n : ℕ                                   已知：n 为自然数
⊢ n.factorial + 1 ≠ 1                   想证：n! + 1 不为 1
```

第 10 行中的simp是 Lean 试图自动简化的 tactic，其用法为

[3]这是因为鉴于 1 的因子只有 1，所以其最小素因子被规定 1

1. simp，Lean 会自动检索所有可以被用于simp的引理，并试图不断使用它们，直到不能再被化简。

2. simp only [引理名称，引理名称，...，引理名称]，Lean 仅使用列表内的引理来简化。

3. simp? 和 simp? only [引理名称，...，引理名称] 当 Lean 在执行完simp后，会把未使用的引理从列表中删除。

化简后（第 11 行），Lean 告诉我们：

```
1 goal                                              还有一个个目标
case n1
n : ℕ                                               已知：n 为自然数
⊢ ¬n.factorial = 0                                  想证：n! 不为 0
```

而这刚好和引理Nat.factorial_ne_zero的内容完全一样：

```
theorem Nat.factorial_ne_zero (n : ℕ) :n.factorial ≠ 0
```

所以 exact Nat.factorial_ne_zero就好了，所以第 10 行最后 Lean 会说：

```
No goals                                            没有目标了
```

在第 12 行的引理nextPrime_ge中，我们用到了by_contra! 假设，这里是在使用反证法。在第 13 行的开始时，Lean 提示如下：

```
1 goal
n : ℕ
⊢ n ≤ nextPrime n
```

在第 14 行开始时，因为使用了反证法by_contra! ineq，证明在这里变成了：

```
n : ℕ
ineq : nextPrime n < n                              假设命题为否，此假设命名为 ineq
⊢ False
```

第 14，18 和 21 行中，我们用have引入了三个新的命题，have的用法和lemma类似[4]，即：

```
have 名称 (输入参数 : 类型) ... : 命题 := by 证明
```

在第 14 至 22 行中，我们遇到了以下 tactic：

[4]在第 21 行的证明中，我们没有用到by，这是因为此命题的证明就是Nat.Prime.not_dvd_one (nextPrime_prime n)，所以没有必要进入 tactic 模式。在 tactic 模式中，此证明写为by exact Nat.Prime.not_dvd_one (nextPrime_prime n)

1. refine: refine的用法和exact类似，但是不需要一次性完成全部证明，可以用?_来留白。在第 15 行开始时，Lean 说：

```
1 goal  *@                                                    还有一个目标
n : ℕ
ineq : nextPrime n < n
⊢ nextPrime n ∣ n.factorial
```

因为Nat.dvd_factorial表述内容为：

```
theorem Nat.dvd_factorial{m n : ℕ} : 0 < m → m ≤ n → m ∣ n.factorial
```

即，对于任意自然数 m 和 n，如果 m 为正且 n 大于等于 m，则 m 整除 n 的阶乘。策略refine Nat.dvd_factorial ?_ ?_的含义是：我将使用Nat.dvd_factorial，并且将在下文中完成使用Nat.dvd_factorial的先决条件。所以证明就变成了：

```
2 goals                                                       还有两个目标
case refine_1                                                 第一个目标
n : ℕ
ineq : nextPrime n < n
⊢ 0 < nextPrime n
case refine_2                                                 第二个目标
n : ℕ
ineq : nextPrime n < n
⊢ nextPrime n ≤ n
```

因为当前有两个命题需要证明，我们用 · 来分隔开每个命题。

2. linarith是用来做不等式的证明的, 如证明不存在有理数 x，y，z 满足 $2x < 3y$，$-4x + 2z < 0$ 和 $12y - 4z < 0$

```
example (x y z : ℚ) (h1 : 2*x < 3*y) (h2 : -4*x + 2*z < 0)
        (h3 : 12*y - 4*z < 0) : False := by
  linarith
```

在第 17 行时，目标是 nextPrime n ≤ n，和 ineq的目标是完全一样的，所以用exact ineq也可以。

3. contradiction 当已知条件中有矛盾时，contradiction可以证明所有命题所谓[5]。在代码运行到第 22 行时，我们有：

```
1 goal
```

[5] *ex falso quodlibet*

```
n : ℕ
ineq : nextPrime n < n
h₁ : nextPrime n ∣ n.factorial
h₂ : nextPrime n ∣ 1
h₃ : ¬nextPrime n ∣ 1
⊢ False
```

h_2和h_3相互矛盾，所以contradiction完成了证明。

现在我们考虑代码片段 1.1中的第 25 到第 29 行，证明对于任意自然数 n，都存在一个自然数 p 使得 $n \leq p$ 且 p 是素数。在第 26 行开始时，我们的目标是

```
1 goal
n : ℕ
⊢ ∃ p, n ≤ p ∧ Nat.Prime p
```

第 26 行中的策略use就是用来处理存在性的命题的，在“使用”了nextPrime n后（第 27 行开始），Lean 的目标变成了证明n ≤ nextPrime n ∧ Nat.Prime (nextPrime n)：

```
1 goal
case h
n : ℕ
⊢ n ≤ nextPrime n ∧ Nat.Prime (nextPrime n)
```

因为现在的证明目标是一个逻辑和，我们用策略fconstructor将其拆分为两个子命题：

```
2 goals
case h.left
n : ℕ
⊢ n ≤ nextPrime n
case h.right
n : ℕ
⊢ Nat.Prime (nextPrime n)
```

并用 · 分隔开每个子命题的证明。

选读：have 和 let 的区别

在证明过程中，如果需要临时引入新的命题，可以使用策略have。如果需要引入新的数据则必须用策略let。因为 Lean 不会记住在have中引入内容的构造方式，而 Lean 会记住由let引入内容的构造方式。

```
1 example : true := by
2   have p : ℕ := 10
3   have hp : p = 10 := by rfl                                    失败
4   ...
```

第 3 行代码会报错，因为p是由have定义的，所以 Lean 不会记住除了p是自然数之外的任何信息。如果把第 2 行的have换成let后，运行第三行代码就不会报错了。

1.3 更多 tactic 及语法

在1.1中,我们记录了更多的 tactic 和示范用法。在这里,我们单独介绍calc和induction。

1.3.1 计算 calc

数学证明中我们也常常需要做一些计算。在 Lean 中calc是一种强大的工具，用来证明等式、不等式，和构造同构等。我们先看代数几何均值不等式的证明：

```
example (a b : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) : √(a * b) ≤ (a + b) / 2 := by
  have := calc (a + b) / 2 - √(a * b)
    _ = (√a ^ 2 + √b ^ 2 - 2 * √a * √b + 2 * √a * √b) / 2 - √(a * b) := by
      simp [Real.sq_sqrt ha, Real.sq_sqrt hb]
    _ = (√a - √b) ^ 2 / 2 + √a * √b - √(a * b) := by ring
    _ = (√a - √b) ^ 2 / 2 + √(a * b) - √(a * b) := by rw [Real.sqrt_mul ha]
    _ = (√a - √b) ^ 2 / 2 := by ring
    _ ≥ 0 := div_nonneg (sq_nonneg _) (by norm_num)
  linarith
```

在这个代码片段第 2 到 8 行中，我们首先证明不等式 $\frac{a+b}{2} - \sqrt{ab} \geq 0$。当然，我们也可以用calc来模仿下列自然语言证明。

$$\begin{aligned}
&\sqrt{ab} \leq \frac{a+b}{2} \\
\iff &ab \leq \left(\frac{a+b}{2}\right)^2 \\
\iff &ab \leq \frac{a^2}{4} + \frac{b^2}{4} + \frac{ab}{2} \\
\iff &0 \leq \frac{a^2}{4} + \frac{b^2}{4} - \frac{ab}{2} \\
\iff &0 \leq \left(\frac{a-b}{2}\right)^2
\end{aligned}$$

```
example (a b : ℝ) (ha : 0 ≤ a) (hb : 0 ≤ b) : √(a * b) ≤ (a + b) / 2 := by
  rw [calc √(a * b) ≤ (a + b) / 2
    _ ↔ a * b ≤ ((a + b) / 2) ^ 2 := by
      simp only [Real.sqrt_le_iff, and_iff_right_iff_imp]
      intro h
      linarith
    _ ↔ a * b ≤ a ^ 2 / 4 + b ^ 2 / 4 + a * b / 2 := by ring_nf
    _ ↔ 0 ≤ a ^ 2 / 4 + b ^ 2 / 4 - a * b / 2 := by rw [← sub_nonneg];
   ring_nf
    _ ↔ 0 ≤ ((a - b) / 2) ^ 2 := by ring_nf
    _ ↔ true := by
      simp only [iff_true]
      exact sq_nonneg ((a - b) / 2)]
```

从两个代码片段中，我们可以看出calc的用法是这样的

```
1 calc 式子
2 _ {=/≤/≥/≃/↔} 式子 := 证明
3 _ {=/≤/≥/≃/↔} 式子 := 证明
4 ...
5 _ {=/≤/≥/≃/↔} 式子 := 证明
```

calc也可以用来构造双射，下列代码片段用来构建偶数和自然数之间的双射和自然数和 $\mathbb{Q} \times \mathbb{Q}$ 之间的双射来构建偶数和 $\mathbb{Q} \times \mathbb{Q}$ 之间的双射：

```
example : {n : ℕ | Even n} ≃ ℚ × ℚ :=
  calc ({n : ℕ | Even n} : Type)
    _ ≃ ℕ :=
      { toFun n := n.1 / 2
        invFun n := ⟨2 * n, by simp⟩
```

```
        left_inv := by rintro ⟨_, ⟨m, rfl⟩⟩; grind
        right_inv := by intro n; simp }
    _ ≃ ℚ × ℚ := (Denumerable.eqv _).symm
```

需要注意的是所有步骤需要可以“串联”起来：比如说如果第 3 行是≤而第五行是≥是不可以的，因为数学意义上表达式 $a = b \leq c \geq d$ 是不能推出 a 和 d 之间的关系的。示例代码如下：

```
example (a b c d : ℝ) : False := by
  have ineq := calc a
    _ = b := by sorry
    _ ≥ c := by sorry
    _ ≤ d := by sorry -- 报错内容: invalid 'calc' step, failed to synthesize
    `Trans` instance Trans GE.ge LE.le ?m.483
  sorry
```

这是因为尽管小于等于和大于等于分别具有传递性（transitivity），两个在一起是没有传递性的。

自测 试证明 $1234^{123456} \equiv 10^{17} \mod 71$。也就是说补充下列代码片段：

```
example (p : ℕ) [Fact <| Nat.Prime p] (a : ZMod p) (h : a ≠ 0) :
    a ^ (p - 1) = 1 := ZMod.pow_card_sub_one_eq_one h

instance : Fact (Nat.Prime 71) where
  out := by decide

example : 1234 ^ 123456 ≡ 10 ^ 17 [MOD 71] := by
  rw [← ZMod.natCast_eq_natCast_iff, Nat.cast_pow, Nat.cast_pow]
  calc (1234 : ZMod 71) ^ 123456
    ...
    _ = 10 ^ 17 := by sorry
```

「请尽量不要使用无法自己口算出来的计算」。

1.3.2 选读：calc在范畴中的应用

同样的道理，calc也可以用来构造范畴中的同构，比如说如果 $\mathcal{F}$ 是一个对称幺半范畴而 A、B、C 都是 $\mathcal{F}$，则我们有

$$\begin{aligned} & A \otimes (B \otimes C) \\ \cong & (A \otimes B) \otimes C && \text{结合律} \\ \cong & (B \otimes A) \otimes C && \text{交换律} \otimes C \\ \cong & C \otimes (B \otimes A) && \text{交换律} \\ \cong & (C \otimes B) \otimes C && \text{结合律} \end{aligned}$$

$A \otimes (B \otimes C) \cong (C \otimes B) \otimes C$，在 Lean 中我们可以写作

```
open CategoryTheory CategoryTheory.Limits MonoidalCategory
noncomputable def huarongdao
    {ℱ : Type*} [Category ℱ] [MonoidalCategory ℱ] [SymmetricCategory ℱ]
    (A B C : ℱ) : A ⊗ (B ⊗ C) ≅ (C ⊗ B) ⊗ A :=
  calc A ⊗ B ⊗ C
  _ ≅ (A ⊗ B) ⊗ C := (α_ A B C).symm
  _ ≅ (B ⊗ A) ⊗ C := β_ _ _ ⊗ᵢ Iso.refl _
  _ ≅ C ⊗ (B ⊗ A) := β_ (B ⊗ A) C
  _ ≅ (C ⊗ B) ⊗ A := (α_ C B A).symm
```

这个定义在 R-模范畴和集合范畴中都和我们想象的是一样的：

```
open TensorProduct
example (R : Type) [CommRing R] (A B C : ModuleCat R) (a : A) (b : B) (c : C) :
    (huarongdao A B C).hom (a ⊗ (b ⊗ c)) = (c ⊗ b) ⊗ a := rfl


example (A B C : Type) (a : A) (b : B) (c : C) :
    (huarongdao A B C).hom (a, (b, c)) = ((c, b), a) := rfl
```

需注意的是如果不使用范畴论的语言，calc不可以用来搭建群环等代数结构的同构（GroupEquiv,RingEquiv,LinearEquiv等）。这是因为这些同构需要类型类的实例，而calc中需要用到的Trans是不能接受类型类的实例的：

```
class Trans (r : α → β → Sort u) (s : β →  → Sort v) (t : outParam (α →  →
   Sort w)) where
  trans : r a b → s b c → t a c
```

所以下列例子是不工作的

```
example (R A B C : Type) [CommRing R]
```

```
    [AddCommGroup A] [AddCommGroup B] [AddCommGroup C]
    [Module R A] [Module R B] [Module R C] :
    A ⊗[R] (B ⊗[R] C) ≃ₗ[R] (C ⊗[R] B) ⊗[R] A :=
  calc A ⊗[R] B ⊗[R] C
  _ ≃ₗ[R] (A ⊗[R] B) ⊗[R] C := sorry
  _ ≃ₗ[R] (B ⊗[R] A) ⊗[R] C := sorry
  _ ≃ₗ[R] C ⊗[R] (B ⊗[R] A) := sorry
  _ ≃ₗ[R] (C ⊗[R] B) ⊗[R] A := sorry
```

自测 若 $\mathcal{F}$ 的幺元记作 $\mathbb{1}_{\mathcal{F}}$，构造同构

$$(A \otimes B \otimes \mathbb{1}_{\mathcal{F}}) \otimes (\mathbb{1}_{\mathcal{F}} \otimes C \otimes D) \cong (A \otimes D) \otimes (C \otimes B).$$

即补充完整下列代码：

```
noncomputable def huarongdao' {𝓕 : Type*}
    [Category 𝓕]
    [MonoidalCategory 𝓕]
    [SymmetricCategory 𝓕]
    (A B C D : 𝓕) :
    (A ⊗ B ⊗ 𝟙_ 𝓕) ⊗
      (𝟙_ 𝓕 ⊗ C ⊗ D) ≅
    (A ⊗ D) ⊗ (C ⊗ B) := _

example (R : Type) [CommRing R] (r₁ r₂ : R)
    (A B C D : ModuleCat R)
    (a : A) (b : B) (c : C) (d : D) :
    (huarongdao' A B C D).hom
      ((a ⊗ (b ⊗ r₁)) ⊗ (r₂ ⊗ (c ⊗ d))) =
    (r₁ • a ⊗ d) ⊗ (r₂ • c ⊗ b) := by
  rfl
```

提示：在 Lean 中α_表示结合律，β_表示交换律，λ_表示左幺元，ρ_表示右幺元。

1.3.3 数学归纳 induction

在 Lean 中有一些引理被打上了elab_as_elim，如Nat.strong_induction_on

```
@[elab_as_elim]
theorem Nat.strong_induction_on {p : ℕ → Prop} (n : ℕ)
    (h : ∀ n, (∀ m, m < n → p m) → p n) : p n
```

凡是这类引理都可以被用在induction中。比如说

```
1 example (p : ℕ → Prop) : ∀ n, p n := by
2   intro n
3   induction n using Nat.strong_induction_on with
4   | h n ih => sorry
```

在使用induction前，我们需要证明：

```
1 goal
p : ℕ → Prop
n : ℕ
⊢ p n
```

而在使用induction后（第 4 行=>后），我们需要证明：

```
1 goal
case h
p : ℕ → Prop
n : ℕ
ih : ∀ m < n, p m
⊢ p n
```

这里 |h n ih => ...是因为Nat.strong_induction_on中出现了(h : ∀ n, (∀ m, m < n → p m) → p n)。

假如我们考虑Nat.strong_decreasing_induction:

```
theorem Nat.strong_decreasing_induction
    {P : ℕ → Prop}
    (base : ∃ n, ∀ m > n, P m)
    (step : ∀ (n : ℕ), (∀ m > n, P m) → P n)
  (n : ℕ) : P n
```

面对同样的例子我们就需要做相应的调整：

```
1 example (p : ℕ → Prop) : ∀ n, p n := by
2   intro n
3   induction n using Nat.strong_decreasing_induction with
4   | base => sorry
5   | step n ih => sorry
```

这里第 4 行出现的| base => 和第 5 行出现的| step n ih => 分别对应着Nat.strong_decreasing_ind的base和step。

自测 试证明，对于自然数 n 有 $169 \mid 3^{3n+3} - 26n - 27$

自测 试证明，对于自然数 n 有 $3(1^5+2^5+\cdots+n^5) \mid 1^3+2^3+\cdots+n^3$

1.3.4 完整的示例

在这里我们用完整的代数几何均值不等式来作为例子：对于任意自然数 n，和非负实数 $x_0,\ldots,x_{n+1}$，我们有

$$\left(\prod_i x_i\right)^{\frac{1}{n+1}} \le \frac{\sum_i x_i}{n+1}$$

这里我们要用 Cauchy 归纳，

```
theorem Nat.cauchy_induction {P : ℕ → Prop}
    (h : ∀ (n : ℕ), P (n + 1) → P n)
    (seed : ℕ) (hs : P seed)
    (f : ℕ → ℕ)
    (hf : ∀ (x : ℕ), seed ≤ x → P x → x < f x ∧ P (f x))
    (n : ℕ) :
  P n
```

也就是说若想证明对于任意自然数 n，$P(n)$ 都成立，我们需要固定某个自然数 s 和函数 $f:\mathbb{N}\to\mathbb{N}$ 并证明

1. `h`：如果 $P(n+1)$ 成立则 $P(n)$ 成立
2. `hs`：$P(s)$ 成立
3. `hf`：对于任意不小于 s 的自然数 x，如果 $P(x)$ 成立，则 $x<f(x)$ 且 $P(f(x))$ 成立。

在代数几何均值不等式中，我们取 $s=1$ 和 $f:x\mapsto 2x+1$。所以我们写成下面的形式：

```
1 theorem AMGM_full (n : ℕ) (x : Fin (n + 1) → ℝ) (hx : ∀ i, 0 ≤ x i) :
2       (∏ i, x i)^(1/(n + 1) : ℝ) ≤ (∑ i, x i) / (n + 1) := by
3   induction n using Nat.cauchy_induction (seed := 1) (f := (2 * • + 1)) with
4   | h n ih =>
5   sorry
6   | hs =>
7   sorry
8   | hf n hn ih =>
9   sorry
```

其中第 5 行的`sorry`需要证明如下

```
n : ℕ
ih : ∀ (x : Fin (n + 1 + 1) → ℝ),
```

```
  (∀ (i : Fin (n + 1 + 1)), 0 ≤ x i) → (∏ i, x i) ^ (1 / (↑(n + 1) + 1)) ≤
    (∑ i, x i) / (↑(n + 1) + 1)
x : Fin (n + 1) → ℝ
hx : ∀ (i : Fin (n + 1)), 0 ≤ x i
⊢ (∏ i, x i) ^ (1 / (↑n + 1)) ≤ (∑ i, x i) / (↑n + 1)
```

也就是说如果对于任意 $n+2$ 个非负实数 $x_0, x_1, ..., x_{n+2}$，代数几何均值不等式成立的话，试证明对于任意 $n+1$ 个非负实数 $x_0, ..., x_{n+1}$, 代数几何均值不等式都成立。其证明如下：设 $x = \frac{\sum_i x_i}{n+1}$，根据归纳假设，我们有

$$(x_0 x_1 ... x_{n+1} x)^{\frac{1}{n+2}} \leq \frac{x_0 + x_1 + \cdots + x_{n+1} + x}{n+2}$$

而我们又有

$$\frac{x_0 + x_1 + \cdots + x_{n+1} + x}{n+2} = x$$

所以，我们不失一般性的假设所有 x_i 均为正数

$$\begin{aligned}
&(x_0 x_1 \dots x_{n+1} x)^{\frac{1}{n+2}} \leq x \\
\Longrightarrow\ & x_0 x_1 \dots x_{n+1} x \leq x^{n+2} \\
\Longrightarrow\ & x_0 x_1 \dots x_{n+1} \leq x^{n+1} \\
\Longrightarrow\ & (x_0 x_1 \dots x_{n+1})^{\frac{1}{n+1}} \leq x \\
\Longrightarrow\ & (x_0 x_1 \dots x_{n+1})^{\frac{1}{n+1}} \leq \frac{x_0 + \cdots + x_{n+1}}{n+1}
\end{aligned}$$

在 Lean 中

```
  | h n ih =>
    let x_last := (∑ i, x i) / (n + 1)
    have ih := ih (Fin.cons x_last x) sorry
    rw [show ((n + 1 : ℕ) : ℝ) + 1 = n + 2 by
      simp only [Nat.cast_add, Nat.cast_one]; ring] at ih

    have eq := calc (∑ i, Fin.cons x_last x i) / (n + 2)
      _ = (∑ i, x i + (∑ i, x i / (n + 1))) / (n + 2) := sorry
      _ = (1 / (n + 1) * ((n + 1) * ∑ i, x i + ∑ i, x i)) / (n + 2) :=
        sorry
      _ = ((n + 2) * (1 / (n + 1) * ∑ i, x i)) / (n + 2) := by ring
      _ = 1 / (n + 1) * ∑ i, x i := sorry
      _ = (∑ i, x i) / (n + 1) := sorry

    rw [eq] at ih
```

```
have ineq := calc x_last * ∏ i, x i
  _ = ∏ i, Fin.cons x_last x i := sorry
  _ ≤  ((∑ i, x i) / (n + 1 : ℝ)) ^ (n + 2 : ℝ) := sorry
  _ = x_last ^ (n + 2 : ℝ) := rfl

wlog hx' : ∀ i, 0 < x i
• sorry

have hx'' : 0 < x_last := sorry
rw [show x_last ^ (n + 2 : ℝ) = x_last ^ (n + 2) by norm_cast,
  pow_succ, mul_comm, mul_le_mul_iff_of_pos_right hx''] at ineq

have h : MonotoneOn (fun x : ℝ => x ^ (1 / (n + 1) : ℝ)) {x | 0 ≤ x} :=
  sorry

have := h sorry sorry ineq
convert this using 1
sorry
```

自测 完成代数几何均值不等式的证明。

自测 完成带权重的几何均值不等式。

Tactic	作用	示例

`sorry`	跳过证明，承认定理是对的。也可以写作`admit`。	`example (n : ℕ) (hn : 3 ≤ n) :`
		`  ∀ (x y z : ℕ),`
		`  x^n + y^n = z^n →`
		`  x * y * z = 0 := by sorry`
		`/-`
		`Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos & generaliter nullam in infinitum ultra quadratum potestatem in duos eiusdem nominis fas est dividere cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.`
		`-/`

`abel`	解决 Abel 群中的等式，变体有 `abel_nf`用来将等式两边转换为正则形式。	`example {α} [AddCommMonoid α] (a b : α) :`
		`  a + (b + a) = a + a + b := by`
		`  abel`
		
		`example {α} [AddCommGroup α] (a : α) :`
		`  (3 : ℤ) • a = a + (2 : ℤ) • a := by`
		`  abel`

`aesop` 一个自动化的算法，利用所有标记为aesop标签的引理和定理来试图组合出证明。常见变体有aesop?可以打印证明步骤、aesop_cat用于范畴论。根据笔者的经验aesop_cat用来证明可以经由重复使用范性解决的问题是极好的

```
example (f : ℝ × ℝ → ℝ) (hf : ∀ r₁ r₂, f (r₁, r₂) = 0) : f = 0 := by
  aesop
```

`exact?` 在库中寻找一模一样的定理

```
example (n : ℕ) :  Σ i ∈ Finset.range n, i = n * (n - 1) / 2 := by
    exact?
```

`apply_fun` 将函数应用到当前的某个假设中，如有假设`h : a = b`，使用`apply_fun f at h`后会变成`h : f a = f b`。当伴有单射，单调等性质时，也可以用来把`f a = f b`转换为`a = b` 或 `a ≤ b` 转化为 `f a ≤ f b`。

```
example (X Y Z : Type) (f : X → Y) (g : Y → Z)
    (H : Function.Injective <| g ∘ f) :
    Function.Injective f := by
  intros x x' h
  apply_fun g at h
  exact H h
```

`ext`	用于证明等式，比如将函数的相等转换为处处相等或者把二元组的等式转换为两个等式	`example (X Y Z : Type) (x x' : X) (y y' : Y) (z z' : Z) : (x, y, z) = (x', y', z') := by ext • sorry • sorry • sorry`

表 1.1: 一些 tactic

第二章　什么是类型

Lean 的底层使用依赖类型（dependent type），而依赖类型论恰好可以作为数学的基础。这一章我们将简单地介绍依赖类型，并将其与集合做对比。不像集合论中所有元素均为集合，类型论中有项（term）和类型（type）的区别：如 $0 : \mathbb{N}$ 是指项 0 的类型是自然数 $\mathbb{N}$。在类型论中每一项有且只有一种类型：如 $0 : \mathbb{N}$ 和 $0 : \mathbb{Q}$ 是两个不同的项，前者是自然数零而后者是有理数零。只有同类的两项可以做等式推断，也就是说 $(0 : \mathbb{N}) = (0 : \mathbb{Q})$ 在 Lean 中是无意义的表达式。接下来我们将介绍 Curry-Howard 同构——类型和命题、项与证明之间的对应关系。我们不妨假设每个命题都对应着一个最多有一项的类型，且此类型的项即为命题的证明。也就是说真命题所对应的类型有且只有一项，而假命题对应的类型为空类型。真命题在 Lean 中记作 `True`；假命题记作 `False` 基于此假设，我们将构建出数学证明所需要的逻辑。

2.1　函数

给定任意两个类型α：`Type`、β：`Type`，我们可以形成从α到β的函数类型，记作α → β。在1.1中，我们已经见到过平方函数可以写作

```
def square (x : ℝ) : ℝ := x^2
```

如果我们想要强调`square`是一个$\mathbb{R}$ → $\mathbb{R}$的项时，我们可以写作

```
def square : ℝ → ℝ := fun x ↦ x^2
```

以上两种写法对于 Lean 来说是完全一样的。之前提到了 Lean 使用依赖类型作为理论基础，而依赖类型之所以叫依赖类型的原因之一是因为在 Lean 中函数的值域是可以依赖于函数的定义域。一个经典的例子是 $\hat{\mathbb{Z}} = \varprojlim \mathbb{Z}/n\mathbb{Z}$，即 $\mathbb{Z}/n\mathbb{Z}$ 的投射极限。在构造 $\hat{\mathbb{Z}}$ 的过程中，不可避免的我们需要考虑 $\mathbb{Z}/n\mathbb{Z}$ 的直积 $\prod_n \mathbb{Z}/n\mathbb{Z}$，在 Lean 中这个直积也是一个函数，其定义域是自然数$\mathbb{N}$，而值域是跟随定义域的输入变化的 — 当输入是`n`时，值域是 `ZMod n`[1]；在 Lean 中表示为

```
def PreZHat : Type := Π (n : ℕ), ZMod n
```

或

[1] `ZMod n` 是 Mathlib 中定义的 $\mathbb{Z}/n\mathbb{Z}$

```
def PreZHat : Type := ∀ (n : ℕ), ZMod n
```

或

```
def PreZHat : Type := (n : ℕ) → ZMod n
```

这三者完全等价。这种值域取决于定义域的函数类型被称为依赖函数或 Π-类型。在 Lean 中，函数类型`α → β`其实也是依赖函数，只不过 `β`是个常量罢了—`α → β`和`Π (a : α), β`是完全一致的。

函数相等 如果 $f, g : \alpha \to \beta$ 是两个函数，它们相等当且仅当对任意 $a : \alpha$，$f(a) = g(a)$。在 Lean 中，该定理地描述如下：

```
theorem funext {α} {β} {f g : (x : α) → β x} (h : ∀ (x : α), f x = g x) :
  f = g
```

在 Lean 中的 tactic 模式下，比起`apply funext`，可以使用`ext`：

```
1 example :
2     (fun x : ZMod 4 ↦ (2 * x + 1) ^ 2) =
3     (fun x : ZMod 4 ↦ 1) := by
4   ext x
5   ring_nf
6   simp [show (4 : ZMod 4) = 0 by rfl]
```

在第 4 行后，Lean 希望我们证明：

```
1 goal
case h
x : ZMod 4
⊢ (2 * x + 1) ^ 2 = 1
```

也就是说`ext x`引入了一个新的`x : ZMod 4`，并希望我们证明`(2 * x + 1)^2 = 1`。

选读：全函数

Lean 中定义的函数必须是全函数，即对于所有输入都必须有输出的定义。如果出于某种原因一定需要有未定义这一概念，可以使用`Option`在任意一个类型中添加一个“未定义值”。

2.1.1 蕴含关系

从逻辑的角度出发，（非依赖函数）函数代表着蕴含关系。假设 P 和 Q 是两个命题，那么蕴含关系 $P \implies Q$ 如果 P 为真则 Q 为真；另一种理解 $P \implies Q$ 的方式是

假如有一个命题 P 的证明则有一个命题 Q 的证明。也就是说 $P \implies Q$ 可以被理解为从 P 的证明到 Q 的证明的函数。而在依赖类型论中，P 和 Q 的证明就是 P 和 Q 所对应的类型的项罢了，所以 $P \implies Q$ 在 Lean 中就是函数类型`P → Q`。

2.1.2 逻辑非

一个命题 P 的否命题和 P 蕴含假命题 $P \implies \bot$ 是等价的。所以逻辑非在 Lean 中表示为 `P → False`，因为逻辑非的常用性，`P → False`也可以写作 `¬ P`。

自测 试证明 $P \implies \neg\neg P$.

选读：直觉主义逻辑

依赖类型论中并不支持（也不否定）双重否定表肯定 $\neg\neg P \implies P$，如果想要使用此演算法需要使用 `Classical.not_not`。Lean 的社群对使用经典逻辑（包括排中律、选择公理等）并不排斥，并没有严格的对于可计算性的要求。

2.1.3 全称量词

非依赖函数对应着逻辑蕴含关系，依赖函数则对应着全称量词。对于某一类 α，假如 P 是某个关于 α 的性质，即 $P : \alpha \to$ Prop。那么依赖函数`Π (a : α), P a`则代表着“任意 $a : \alpha$, $P(a)$ 都成立”；这是因为要想证明 $\forall a : \alpha, P(a)$，我们则需对任意 $a : \alpha$ 都构造出 $P(a)$ 的证明。用类型论的语言来说，对于任意项`a : α`，构造出类型为`P(a)`的项 — 换句话说，构造出类型为依赖函数`Π (a : α), P a`的项。当然`Π (a : α), P a`也可以用我们更熟悉的`∀ a, P a`的方式描述。

2.2 归纳类

在 Lean 中声明类型有两种方式，我们在本节中介绍归纳类（inductive type）。如果某个类型中的项可以被不相交的多种方式构造，那么这个类型就应该用归纳类。例如我们可以用归纳类来定义有限域 $\mathbb{F}_2$：

```
inductive F2 : Type
| zero : F2
| one : F2
```

这段代码构造了一个 $\mathbb{F}_2$ 的元素有两种方式，其一是 0, 其二是 1 且 $0 \neq 1$。因为这种类型定义支持数学归纳，所以将它定义为归纳类。比如我们想利用 $\mathbb{F}_2$ 构造更复杂的物件，我们只需要考虑如何利用 0 构造和如何利用 1 构造。在 Lean 中体现为 Lean 自动生成的 $\mathbb{F}_2$ 归纳法：`F2.rec`:

```
F2.rec {motive : F2 → Type}
  (zero : motive F2.zero)
  (one : motive F2.one) : ∀ x: F2, motive x
```

考虑和F2有关的某个构造motive : F2 → Type, 如果我们用 0 得出(zero : motive F2.zero)且用 1 得出(one : motive F2.one)，这样对于任意 $x : \mathbb{F}_2$，我们都获得了一个motive x；且当 $x = 0$ 时构造结果与zero一样；当 $x = 1$ 时的构造结果与one一样。

另一个典型的但要复杂一些的例子是自然数。自然数的项为 $0, 1, 2, 3, \ldots$。其中有两种方式构造自然数：第一种为“0 是一个自然数”；第二种是对于任意自然数 n，$n+1$ 也是一个自然数，此时 $n+1$ 被称作 n 的后继。并且注意到这两种构造方式是不相交的，要么一个自然数是 0 要么一个自然数是另一个自然数的后继，两种可能性不会同时发生。在 Lean 中，我们定义自然数为：

```
inductive Nat                                  定义类型 Nat
| zero : Nat                                   第一种构造方式是 0
| succ (n : Nat): Nat                          第二种构造方式是给定 (n : Nat)，构造 succ n。
```

代码片段 2.1: 自然数

和 $\mathbb{F}_2$ 一样，Lean 也会自动生成Nat的归纳法 Nat.rec:

```
Nat.rec {motive : Nat → Type}
  (zero : motive Nat.zero)
  (succ : (n : Nat) → motive n → motive n.succ) :
  ∀ x: Nat, motive x
```

这是在说如果想要获得某个利用自然数的构造motive : Nat → Type，我们需要规定 0 对应的zero : motive MyNat.zero；我们需要根据 n 对应着什么来规定 n 的后继对应着什么。如此一来，我们便规定完了全部自然数对应着什么。我们用阶乘为例：

```
def Nat.factorial : Nat → Nat :=
  Nat.rec 1 (fun n factorial_n ↦ (n + 1) * factorial_n)
```

这是在说如果给定任意自然数想要定义其阶乘，则需要定义 $0! = 1$，且如果 n 的阶乘为factorial_n，则 $n+1$ 的阶乘应该为 $(n+1) *$ factorial_n。因为归纳很常用，Lean 也提供以下语法

```
def Nat.factorial'
| 0 => 1
| succ n => (n + 1) * (factorial' n)
```

自测 考虑以下定义整数的方式

```
inductive Integer : Type
| nonneg : Nat → Integer                    每一个自然数 n 对应着整数 n
| neg : Nat → Integer                       每一个自然数 n 对应着整数 −(n+1)
```

试定义整数的加法和乘法。

2.2.1 逻辑或

从逻辑的角度来说，命题 P 和命题 Q 的逻辑与 $P \vee Q$ 也可以被定义为归纳类。我们提到了 P 作为类型时其项是 P 的证明；同理，Q 的项是 Q 的证明，$P \vee Q$ 的项是 $P \vee Q$ 的证明。那么如何证明命题 $P \vee Q$ 呢？我们要么证明 P 要么证明 Q—用类型论的语言来说，构造 $P \vee Q$ 的项的方法有两种：要么通过构造 P 的项，要么通过构造 Q 的项。所以 $P \vee Q$ 应该被定义成一个归纳类：

```
inductive Or (P Q : Prop) : Prop where
| inl (h : P) : Or P Q
| inr (h : Q) : Or P Q
```

在这段代码中，`Or P Q`有两种构造方式 `inl : P → Or P Q`和`inr : Q → Or P Q`分别对应着“如果有 P 的证明，就可以构造出 P 或 Q 的证明”和“如果有 Q 的证明，就可以构造出 P 或 Q 的证明”。由于逻辑与`Or`是如此常用，Lean 中可以用 `P ∨ Q`来表示`Or P Q`。

自测 证明逻辑或是交换的，也就是说构造如下函数

```
def Or.swap (P Q : Prop) : Or P Q → Or Q P := sorry
```

2.2.2 存在量词

假如 P 是关于某个类型 α 的性质，也就是说对 $P : \alpha \to \mathsf{Prop}$，那么可以形成命题“存在 $a : \alpha$ 满足 $P(a)$”。在 Lean 中，我们需要使用归纳类；这是因为要想证明“存在 $a : \alpha$ 满足 $P(a)$”我们需要举出某个元素 $a : \alpha$ 并证明 $P(a)$。在类型论中，相对应的就是存在量词的构造方式是取项`a : α`和项`h : P a`：

```
inductive Exists {α} (P : α → Prop) : Prop
| intro (a : α) (h : P a) : Exists p
```

因为存在量词的重要性，在 Lean 中我们也可以使用熟悉的`∃ a : α, P a`。

选择公理 假设我们有某个`∃ a : α, P a`的项, 也就是说我们证明存在 $a : \alpha$ 满足 $P(a)$。一种常见的诉求是“取出这个 a”，想要达到这个效果，Lean 提供了以下定义和定理。

```
def Exists.choose {α} {p : α → Prop} (P : ∃ (a : α), p a) : α

theorem Exists.choose_spec {α} {p : α → Prop} (P : ∃ (a : α), p a) :
  p P.choose
```

需注意的是即使证明∃ a : α, P a时使用了一个明确的a，这个值也不会被记住并且使用Exists.choose得出的值也和原a毫无关系。比如我们想要证明存在一个素数：在下列代码片段中第 3 行报错

```
1 theorem exists_prime : ∃ n, Nat.Prime n := ⟨2, Nat.prime_two⟩
2
3 example : exists_prime.choose = 2 := by rfl
```

也就是说，关于使用Exists.choose得出的值，我们唯一知道的信息就是它的类型是α和它符合性质P。

2.3 结构

在章节2.2中，我们介绍了归纳类；本章我们介绍另一种定义新类型的方式 — 通过复合多个类型来获得新类型。一个典型的例子是二维平面中的点

```
1 @[ext]
2 structure Point2D where
3   x : ℝ
4   y : ℝ
```

在这个例子中我们定义了一个新的类型Point2D，这个类型包含了x和y两个实数。如果P : Point2D是一个项，可以通过P.x和P.y来获取这两个实数值。如果我们需要构造一个Point2D，可以使用以下等价的方式：

```
def o0 : Point2D := Point2D.mk 0 0

def o1 : Point2D := ⟨0, 0⟩

example : o0 = o1 := by rfl
```

从数学的角度来看，这个类型无非是 $\mathbb{R} \times \mathbb{R}$，但是这种定义方式更具有可读性。通过第一行的@[ext]，Lean 会自动生成两个引理

```
lemma Point2D.ext {P Q : Point2D} : P.x = Q.x → P.y = Q.y → P = Q

lemma Point2D.ext_iff {P Q : Point2D} : P = Q ↔ P.x = Q.x ∧ P.y = Q.y
```

并且在使用ext这个 tactic 的时候，Lean 也会自动调用Point2D.ext。

一般来说，structure的语法如下

```
structure 结构名称 extends 结构名称 where
  数据 : 类型
  数据 : 类型
  数据 : 类型
  ...
```

这里extends是类似于“扩充”现有structure的概念，如果我们现在想要定义Point3D, 就可以使用

```
structuer Point3D extends Point2D where
  z : ℝ
```

2.3.1 卡式积

对于任意两个类型α和β来说，我们可以通过structure来定义他们的卡式积：

```
structure Prod (α β : Type) where
  fst : α
  snd : β
```

因为此构造较为常用，所以亦可以写作$\alpha \times \beta$。如果p : $\alpha \times \beta$是一个二元组，则可以使用p.1和p.2来取得第一个和第二个元素，显然p = (p.1, p.2)。

2.3.2 逻辑合取

P 和 Q 是两个命题，若想要证明 $P \wedge Q$，则需要证明 P 且证明 Q。用类型的语言来说，我们如果想要构造P ∧ Q项，则需要构造P的项和Q的项（命题对应着类型；证明对应着项）— 也就是说 $P \wedge Q$ 完全就是 P 和 Q 的卡式积。

自测 不使用tauto证明:

```
example {P Q R S T: Prop} (h : (P ∧ Q ∧ R) ∧ S ∧ T) : S ∧ Q := by sorry
```

2.3.3 选读：依赖卡式积

在2.1这一章中我们介绍了依赖函数，依赖函数是函数的推广。这里我们推广卡式积。目前卡式积中第二个元素的类型是不能依赖于第一项的，但是这个限制是无关紧要的。如果α是一个脚标类、β : α → Type是一系列类，那么我们也可以形成卡式积 (a : α) × β a 或 Σ a : α, β a。这个类型也被称为 Σ-类。其定义方式如下

```
structure Sigma {α : Type} (β : α → Type) where
  fst : α
  snd : β fst
```

我们在介绍存在量词时曾说过，存在量词的证明并不会记住构造过程

2.4 等价关系和商类

数学中另一个常用操作是商，例如环模理想形成的商环。在 Lean 中，我们用Setoid（集合体）来实现商的概念。首先，我们讨论等价关系（Equivalence）, 等价关系是指一个满足自反性、对称性和传递性的关系；在 Lean 中：

1. 在α上的关系是类型为$\alpha \to \alpha \to$ Prop的项。
2. 若r是α上的关系，r满足自反性是指对于任意α的项x都有r x x。
3. 若r是α上的关系，r满足对称性是指对于任意α的项x和y都有r x y蕴含r y x。
4. 若r是α上的关系，r满足传递性是指对于任意α的项x、y和z都有r x y和r y z蕴含r x z。

所以，等价关系被定义为

```
structure Equivalence {α} (r : α → α → Prop) : Prop where
  refl  : ∀ x, r x x
  symm  : ∀ {x y}, r x y → r y x
  trans : ∀ {x y z}, r x y → r y z → r x z
```

那么集合体就是一个带有等价关系的类型，即：

```
class Setoid (α) where
  r : α → α → Prop
  iseqv : Equivalence r
```

一旦我们有了某个α上的集合体r : Setoid α，我们就可以考虑 $\alpha/_r$，在 Lean 中表示为Quotient r。我们可以使用以下函数来定义定义域为商类型的函数：

```
def Quotient.recOn {α} {s : Setoid α} {motive : Quotient s → Type}
  (q : Quotient s)
  (f : (a : α) → motive (Quotient.mk s a))
  (h : ∀ (a b : α) (p : a ≈ b), Eq.ndrec (f a) (h p) = f b) : motive q
```

这里f指给定任意a : α，我们需要规定如何利用a的等价类来构造；而h是明确我们的构造方式需要是良定义的。Quotient.mk Quotient.mk'' 或 ⟦•⟧是去等价类的操作即$\alpha \to$ Quotient r。Quotient.sound和Quotient.sound'给出了以下结论：

```
theorem Quotient.sound {α} {s : Setoid α} {a b : α}, a ≈ b → ⟦a⟧ = ⟦b⟧
theorem Quotient.sound' {α} {s : Setoid α} {a b : α}, s a b → Quotient.mk'' a
    = Quotient.mk'' b
```

自测「难度高」 我们前面介绍了用归纳类来定义整数，我们也可以将整数定义为 $\mathbb{N}\times\mathbb{N}$ 的某个商。数学上我们考虑以下关系

$$(m,n)\sim(m',n') \iff m+n'=m'+n$$

则整数可以被定义为 $\mathbb{N}\times\mathbb{N}/_\sim$。参考小节2.6，定义这个整数并证明它和 Lean 里已经定义的整数是一一对应的。

2.5 类型类 Typeclass

在数学中，我们常常谈到“某个集合是一个群”、“某个类型是一个环”。这意味着：

- 这个集合本身只是一个**载体**；
- 我们还为它**配备**了额外的结构（运算与公理）。

在 Lean 中，`Typeclass` 提供了一种机制来表达这种“类型上附带结构”的思想。例如：

```
class Semigroup (G : Type) extends Mul G where
  mul_assoc : ∀ a b c : G, a * b * c = a * (b * c)
```

这里 `Semigroup` 是一个类型类，它描述了“成为一个半群”所需的数据（集合`G`和二元运算`Mul G`）和公理（`mul_assoc`）。如果需要声明类型类的实例，我们可以使用`instasnce`关键字，比如：

```
instance : Semigroup (ℕ → ℕ) where
  mul := (• ∘ •)
  mul_assoc := Function.comp_assoc
```

在这段代码片段中，我们实现了自然数到自然数函数集合上的半群结构，其二元运算是函数的复合。

类型类和结构（structure）是引入复合数据的方式。但不同的是 Lean 可以对类型类进行自动的推导（typeclass synthesis）：给定一些类型类的实例，Lean 可以自动推断出新的实例。比如我们可以实现半群的卡式积：

```
instance Semigroup.prod (H G : Type) [Semigroup H] [Semigroup G] : Semigroup
    (H × G) where
```

```
  mul := fun p q => (p.1 * q.1, p.2 * q.2)
  mul_assoc := by simp [mul_assoc]
```

在这段代码中，我们使用方括号来声明类型类的实例`[Semigroup H]`和`[Semigroup G]`分别是`H`和`G`上的半群实例；而`Semigroup.prod H G`就是`H × G`上的半群实例，即`Semigroup.prod H G`的实现是在给 Lean 添加了一条寻找`H × G`上半群结构的路径—通过分别寻找`H`和`G`上的半群结构。也就是说 Lean 是需要通过返回结果来反推需要哪些输入的参数的；所以以下代码不好：

```
instance (A B C : Type) [AddCommGroup A] [AddCommGroup B] [AddCommGroup C] :
    AddCommGroup (A × B) := _
```

这是因为仅从返回结果`AddCommGroup (A × B)`中来看的话，是无法看出`C`应该是什么的。

正是因为类型类是由 Lean 自动管理的，每个类型上的同一类型类的实例最多只能有一个。以下例子是很不好的

```
instance s1 : Semigroup ℤ where
  mul := (• * •)
  mul_assoc := mul_assoc

instance s2 : Semigroup ℤ where
  mul := (• + •)
  mul_assoc := add_assoc
```

这是因为`s2`会覆盖`s1`。为了解决加法乘法记号的问题，所有的乘法的结构都有对应的加法版本如`Semigroup`对应着`AddSemigroup`；`Group`对应着`AddGroup`等。为了在两个记号中转换，Lean 定义了`Multiplicative`和`Additive`的概念：如果`H`是一个乘法群，那么`Additive H`就是`H`但是其二元运算已经被换成了加法。比如我们考虑下面自测题:

自测题 不存在域 F 使得 F 的加法群结构和 F 的乘法群结构同构。

在 Lean 中我们需要这样实现：

```
variable (F : Type) [Field F]

example : IsEmpty (Units F ≃* Multiplicative F) := by
  by_contra r
  simp only [not_isEmpty_iff] at r
  obtain ⟨g⟩ := r
  have h1 : g 1 = Multiplicative.ofAdd 0 := by simp
  let a := (g (-1)).toAdd
  have eq1 := calc 2 • a
    _ = 2 • (g (-1)).toAdd := rfl
```

```
    _ = (g (-1)).toAdd + (g (-1)).toAdd := by rw [two_nsmul]
    _ = (g (-1) * g (-1)).toAdd := by simp
    _ = (g ((-1) * (-1))).toAdd := by rw [g.map_mul]
    _ = (g 1).toAdd := by simp
    _ = Multiplicative.toAdd 1 := by simp
    _ = 0 := by simp
  simp only [nsmul_eq_mul, Nat.cast_ofNat, mul_eq_zero] at eq1
  sorry
```

2.6 综合性例子：带点集（Pointed Set）

在这个小节中我们定义带点集。数学意义上带点集是一个二元组 (S, x) 满足 $x \in S$。两个带点集 (A, x) 和 (B, y) 之间的态射被定义为函数 $f : A \to B$ 满足 $f(x) = y$。在 Lean 中我们做以下定义：

```
1  structure PointedSet where
2    t : Type
3    base : t
4
5  instance : CoeSort PointedSet Type where
6    coe T := T.t
```

第 5 行中的`CoeSort`规定了如何将带点集合`S : PointedSet`看作类型，类型判断`x : S`也即是判断`x : S.t`。那么态射可以被定义为：

```
1  @[ext]
2  structure PointedFunc (S T : PointedSet) where
3    toFun : S → T
4    preserves' : toFun S.base = T.base
5
6  instance (S T : PointedSet) : FunLike (PointedFunc S T) S T where
7    coe f := f.toFun
8    coe_injective' := by
9      rintro ⟨f, _⟩ ⟨g, _⟩ rfl
10     rfl
```

第 6 行中的`FunLike`规定了对于任意两个带点集合`S : PointedSet`和`T : PointedSet`，它们之间的态射`f : PointedFunc S T`如何被看作一个函数，即对任何一个`s : S`，函数应用`f s`应该被理解为`f.toFun s`。有趣的是，对于任意两个带点集合 (A, x) 和 (B, y)，他

们之间的态射 $\mathrm{Hom}((A,x),(B,y))$ 也是一个带点集合[2]$(\mathrm{Func}(A,B),a\mapsto y)$。在 Lean 中

```
abbrev PointedSet.InternalHom (S T : PointedSet) : PointedSet where
  t := PointedFunc S T
  base := ⟨fun _ => T.base, rfl⟩
```

这里abbrev是说当 Lean 看到PointedSet.InternalHom应该自动展开成右手边。当然带点集合也是有卡式积的：即对于任意两个带点集合 (A,x) 和 (B,y)，$(A\times B,(x,y))$ 也是一个带点集合。在 Lean 中

```
def PointedSet.Product (S T : PointedSet) : PointedSet where
  t := S × T
  base := (S.base, T.base)
```

卡式积是有函子特性的：$f:(S,x)\to(T,y)$ 和 $g:(S',x')\to(T',y')$ 是两个带点集合之间的态射，那么 $f\times g:(S\times S',(x,x'))\to(T\times T',(y,y'))$ 也是一个态射。在 Lean 中这体现为

```
def PointedFunc.product {S S' T T' : PointedSet}
    (f : PointedFunc S S') (g : PointedFunc T T') :
    PointedFunc (S.Product T) (S'.Product T') where
  toFun p := (f p.1, g p.2)
  preserves' := by simp
```

除了卡式积外，对于任意两个带点集合 (A,x) 和 (B,y)，我们还可以形成 smash 积。具体操作如下：我们考虑 $A\times B$ 上的关系：

$$(a,b)\sim(a',b')\iff\begin{cases}\text{两点坐标中均有基准点：即（}a=x\text{或}b=y\text{）且（}a'=x\text{或}b'=y\text{）}\\ \text{两点相等：即}a=a'\text{且}b=b'\end{cases}$$

大家可以验证这确实是一个等价关系。

这个关系在 Lean 中可以这样实现：

```
1 abbrev baseInProduct {S T : PointedSet} (x : S.Product T) : Prop :=
2   x.1 = S.base ∨ x.2 = T.base
3
4 inductive smashedRel (S T : PointedSet) (x y : S × T) : Prop
5 | base (_ : baseInProduct x ∧ baseInProduct y )
6 | eq (_ : x = y)
7
8 lemma smashedRel_equivalence (S T : PointedSet) :
9     Equivalence (smashedRel S T) where
```

[2]即带点集合的范畴有 internal hom set

```
  refl _ := .eq rfl
  symm
  | .base h => .base h.symm
  | .eq h => .eq h.symm
  trans
  | .base h, .base h' => .base <| by tauto
  | .base h, .eq h' => .base <| by aesop
  | .eq h, .base h' => .base <| by aesop
  | .eq h, .eq h' => .eq <| by aesop
```

那么 smash 积就可以被定义成带点集合 $A \otimes B := ((A \times B)_{\sim}, [(x, y)])$。

```
def smashedSetoid (S T : PointedSet) : Setoid (S.Product T) where
  r := smashedRel S T
  iseqv := smashedRel_equivalence S T

@[simps]
def PointedSet.Smash (S T : PointedSet) : PointedSet where
  t := Quotient (smashedSetoid S T)
  base := ⟦(S.base, T.base)⟧
```

这里第 5 行的`@[simps]`是让 Lean 自动生成以下引理。

```
lemma PointedSet.Smash_t:
  ∀ (S T : PointedSet), (S.Smash T).t = Quotient (smashedSetoid S T)

lemma PointedSet.Smash_base:
  ∀ (S T : PointedSet), (S.Smash T).base = ⟦(S.base, T.base)⟧
```

不光卡式积有函子特性，smash 积也有函子特性：$f : (S, x) \to (S', x')$ 和 $g : (T, y) \to (T', y')$ 是两个带点集合之间的态射，我们可以定义态射 $f \otimes g : S \otimes T \to S' \otimes T'$ 为 $[(s, t)] \mapsto [(f(s), g(t))]$。在 Lean 中

```
def PointedFunc.smash {S S' T T' : PointedSet} (f : PointedFunc S S') (g :
   PointedFunc T T') :
     PointedFunc (S.Smash T) (S'.Smash T') where
   toFun := Quotient.map (f.product g) ...
   preserves' := ...
```

除了函子特性，smash 积是有伴随性质的（adjunction），也就是说对于任意三个带

点集合 (A,x)、(B,y) 和 (C,z)，我们有以下双射[3]：

$$\begin{aligned}\mathrm{Hom}(A\otimes B,C)&\cong\mathrm{Hom}(A,\mathrm{Hom}(B,C))\\ f&\mapsto\Big(a\mapsto b\mapsto f[(a,b)]\Big)。\\ \Big([(a,b)]\mapsto f(a)(b)\Big)&\mapsfrom f\end{aligned}$$

在 Lean 中我们说：

```
example (A B C : PointedSet) :
    PointedFunc (A.Smash B) C ≃ PointedFunc A (B.InternalHom C) where
  toFun f :=
    { toFun a :=
      { toFun b := f ⟦(a, b)⟧
        preserves' := by ... }
      preserves' := by ... }
  invFun f :=
  { toFun a := a.recOn (fun a => f a.1 a.2) <| by ...
    preserves' := by ... }
  left_inv := ...
  right_inv := ...
```

代码片段 2.2: 伴随性

自测 对于任何带点集合 (A,x)，定义态射 $\mathsf{id}:(A,x)\to(A,x)$。定义带点集合态射的复合。

自测 在代码片段2.2中，我们用的是双射的概念。请定义带点集合的同构，并将代码片段2.2升级为带点集合的同构。

2.7 选读：Type的类型是什么

正如集合论中没有包含所有集合的集合，Type的类型不能是Type。在 Lean 中，Type其实是Type 0的简写，而Type 0的类型是Type 1；一般来说Type u的类型是Type (u+1)。注意这里的0、1、u和u+1 并不是自然数，而是宇宙层级（universe level），虽然宇宙层级支持后继，但他们却不支持如加法之类的运算。如果当前的宇宙层级太小，Lean 会自动扩张宇宙。例如在定义所有环的类型时：

```
structure CommRingCat where
  carrier : Type u
```

[3]伴随函子还需要自然性质，在此我们忽略

```
[commRing : CommRing carrier]
```

Lean 会告诉你CommRingCat的类型是Type (u+1)。用集合论的语言，这就是说所有环不能形成一个集合。除去后继外，宇宙层级也支持max运算，比如如果我们有可能不在同一宇宙的两类A : Type u和B : Type v，他们之间的函数A → B的类型就是Type (max u v)。在本书中，除非绝对必要，我们尽量待在Type 0中；感兴趣的读者可以自己尝试找到最一般的等级限制。

2.8 选读：证明的无关性（Proof Irrelevance）

在 Lean 的类型系统中，对于任意命题P : Prop，它的任意两项都是相等的；或者说对于任意命题，它至多有一个证明。这个假设叫做证明的无关性。这个假设的提出是为了让 Lean 的类型论更贴近我们的数学直觉。考虑偶数的例子，假如对于一个数是否是偶数有多个不相等的证明，那么从偶数到自然数的自然映射就有可能不是单射；这是不符合数学直觉的。

2.9 练习册

1. 在 Lean 中Fin n是指 $\{m|m < n\}$ 的数，实现如下：

```
structure Fin (n : Nat) where
  val  : Nat
  isLt : LT.lt val n
```

Matrix (Fin n) (Fin n) ℕ是Fin n → Fin n → ℕ 一个 $n \times n$ 的方阵称为魔方阵，如果：

(a) 矩阵的每个元素都属于 $1, 2, \ldots, n^2$，且均不同。

(b) 每一行、每一列以及两条主对角线上的数字之和都相等。

尝试补充魔方阵的定义：

```
structure IsMagicSquare {n : ℕ} (M : Matrix (Fin n) (Fin n) ℕ) : Prop
    where
```

2. 尝试在 Lean4 中定义图结构。用你定义的图结构，构造柯尼斯堡七桥的图。(提示: 先给出顶点集，根据顶点集定义两点相连的性质，再定义边集)

3. 定义以下概念并证明它们的等价性：

(a) 单射函数和函数的左逆。

(b) 满射函数和函数的右逆。

(c) 双射函数和存在唯一逆函数。

4. 让我们来探索 Hom 的函子特性。给定两函数 $f : A' \to A$ 和 $g : B \to B'$，试定义

(a) $\mathrm{Hom}(A, -)$ 和 $\mathrm{Hom}(-, B)$ 「提示：函数的复合在 Lean 中可以使用 $\circ$ 表示」

(b) 请证明你的构造满足以下交换图：

$$\begin{array}{ccc} \mathrm{Hom}(A,B) & \xrightarrow{\mathrm{Hom}(f,B)} & \mathrm{Hom}(A',B) \\ {\scriptstyle \mathrm{Hom}(A,g)}\downarrow & & \downarrow{\scriptstyle \mathrm{Hom}(A',g)} \\ \mathrm{Hom}(A,B') & \xrightarrow{\mathrm{Hom}(f,B')} & \mathrm{Hom}(A',B') \end{array}$$

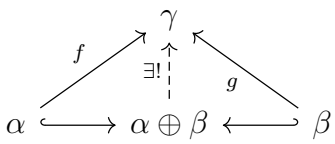
「$\mathrm{Hom}(A, B)$ 代表从 A 到 B 的函数。」

5. 余积（不交并）的构造。

(a) 对于任意类 α 和 β，Lean 中用$\alpha \oplus \beta$来表示以下构造：

```
inductive Sum (α β : Type) where
  | inl (val : α) : Sum α β
  | inr (val : β) : Sum α β
```

试证明余积的泛型：对于任意类型 γ、$f : \alpha \to \gamma$ 和 $g : \beta \to \gamma$ 都存在唯一函数 $f \oplus g : \alpha \oplus \beta \to \gamma$ 使得以下交换图成立：



(b) 试拓展到任意多个类型，即给定脚标类 $\iota : \mathsf{Type}$ 和一系列类 $A : \iota \to \mathsf{Type}$，试定义 $A_{i:\iota}$ 的余积并证明你的构造满足对应的泛型。

6. 让我们继续探索带点集合。定义 S_0 为带点集合 $(\{0, 1\}, 1)$，在 Lean 中

```
def S0 : PointedSet where
  t := Bool
  base := true
```

试证明带点集合的对称幺半范畴性，也就是说对于任意带点集合 (A, a)、(B, b)、(C, c) 和 (D, d) 构造以下同态：

(a) 结合 $\alpha : (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$

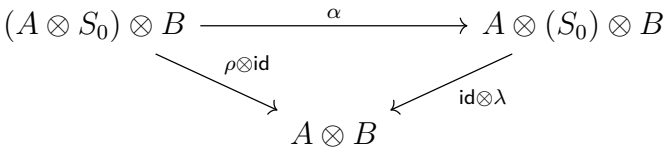
(b) 左单位 $\lambda : S_0 \otimes A \cong A$ 和右单位 $\rho : A \otimes S_0 \cong A$

(c) 交换 $\beta : A \otimes B \cong B \otimes A$

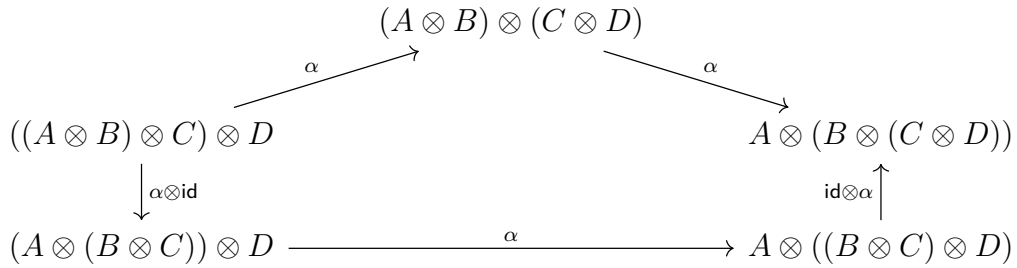
并证明他们满足

(a) $\beta \circ \beta = \mathsf{id}$

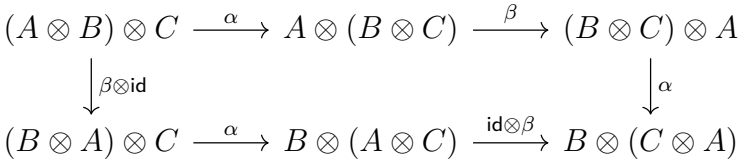
(b) 三角形



(c) 五边形



(d) 六边形



7. 开放式题目：我们可以定义归纳类`Option` 为

```
inductive Option (α : Type) where
| none : Option α
| some (val : α) : Option α
```

对比以下减法`optionSub : Nat → Nat → Option Nat`、`subInt : Nat → Nat → Int`和`sub : Nat → Nat → Nat`的各自好处和坏处；其中如果 $m < n$，`optionSub`将 $m - n$ 定义为`none`、`subInt`会将 m 和 n 视作整数并进行减法、`sub`将 $m - n$ 定义为 0。

第二部分

代数

第三章　群论

3.1　商群

在2.4和2.5这两小节中，我们介绍了怎么构建等价类和引入类型类。本章中我们通过 Coset 来实现商群[1]。首先我们声明变量：让 G 是任意群，H 是任意子群。

```
variable {G : Type} [Group G] (H : Subgroup G) [H.Normal]

open Pointwise
```

那么商群 G/H 其实就是所有左陪集的集合。

```
structure QGroup : Type where
  carrier : Set G
  isCoset : ∃ g : G, carrier = g • H
```

在之前代码片段中的`open Pointwise`是用来让 Lean 意识到`g • H`是指点在集合上的作用。为了方便之后证明`QGroup`确实是个群，我们先写一些有用的引理。

```
instance : SetLike (QGroup H) G where
  coe C := C.carrier
  coe_injective' := by
    intro C D; cases C; cases D; simp
```

`SetLike`类型类是用来说明某个类型的项可以被视作集合。在这里，我们将左陪集视作集合。

因为左陪集有生成元，也可以被任意`g : G`生成出来，所以我们定义

```
def QGroup.out (C : QGroup H) : G := Classical.choose C.isCoset

def QGroup.gen (g : G) : QGroup H := ⟨g • H, ⟨g, rfl⟩⟩
```

并定义商群上的归纳法则

```
@[elab_as_elim]
```

[1]Mathlib 中商群是通过 quotient type 实现的，相关文档在`QuotientGroup`或`AddQuotientGroup`里

```
theorem gen_induction {P : QGroup H → Prop}
    (h : ∀ g : G, P (gen H g)) :
    ∀ C : QGroup H, P C := by sorry
```

那么商群中 1，乘法和逆元就可以被定义为

```
instance : One (QGroup H) where
  one := ⟨H, ⟨1, by simp⟩⟩

instance : Mul (QGroup H) where
  mul C D := ⟨C.out • D.out • H, ⟨C.out * D.out, by simp [mul_smul, out_spec]⟩⟩

instance : Inv (QGroup H) where
  inv C := ⟨C.out⁻¹ • H, ⟨C.out⁻¹, rfl⟩⟩
```

自测 证明乘法和逆元的关键性质

1. 证明 $1 = 1H$

```
lemma gen_one : gen H 1 = 1 := sorry
```

2. 证明 $(gH)(g'H) = (gg')H$

```
lemma gen_mul_gen (g g' : G) : gen H g * gen H g' = gen H (g * g') := by
    sorry
```

3. 证明 $(gH)^{-1} = g^{-1}H$

```
lemma gen_inv (g : G) : (gen H g)⁻¹ = gen H g⁻¹ := by sorry
```

在上述习题中，需注意的是因为out是任意选出的一个代表元，我们无法证明(gen H g).out是g。在完成上述练习后，商群的群结构就显然了。

```
lemma mul_one' (C : QGroup H) : C * 1 = C := by
  induction C using gen_induction with | h g =>
  rw [← gen_one, gen_mul_gen, mul_one]

lemma one_mul' (C : QGroup H) : 1 * C = C := by
  induction C using gen_induction with | h g =>
  rw [← gen_one, gen_mul_gen, one_mul]
```

```
lemma mul_assoc' (C D E : QGroup H) : (C * D) * E = C * (D * E) := by
  induction C using gen_induction with | h g =>
  induction D using gen_induction with | h g' =>
  induction E using gen_induction with | h g'' =>
  simp [gen_mul_gen, mul_assoc]

lemma inv_mul_cancel' (C : QGroup H) : C⁻¹ * C = 1 := by
  induction C using gen_induction with | h g =>
  simp [gen_inv, gen_mul_gen]

instance : Group (QGroup H) where
  mul_assoc := mul_assoc'
  one_mul := one_mul'
  mul_one := mul_one'
  inv_mul_cancel := inv_mul_cancel'
```

3.1.1 Mathlib 中的群与商群

使用陪集来定义商群的劣势是过度使用`Classical.choice`。尽管我们可以通过完善相关接口来让陪集越来越好用，使用商来定义商群总会是更优雅的，因为我们可以通过 quotient type 的相关泛型来定义商群上的操作。Mathlib 中也是通过 quotient type 来实现商群的.

- `(Add)Subgroup.map`：任意同态 $\phi : G \to M$，包含了 $\phi(H)$ 的最小子群。
- `(Add)Subgroup.comap`：子群的前像（preimage）
- `(Add)Subgroup.closure`：包含了某个子集的最小子群
- `(Add)Subgroup.(add)SubgroupOf`：如果 H, K 是 G 的子群，那么`H.addSubgroupOf K`是 $H \sqcap K$ 视为 H 的子群。
- `Quotient(Add)Group.mk'`：$G \to G/H$ 的投射
- `Quotient(Add)Group.lift`：如果一个同态 $\phi : G \to M$ 的核（kernel）是 H 的子集，ϕ 可以下降到 $G/H \to M$。
- `Quotient(Add)Group.map`：如果一个同态 $\phi : G \to G'$ 满足 $H \leq \phi^{-1}(H')$，则 ϕ 下降到 $G/H \to G'/H'$。

3.1.2 同构定理

假如有态射 $\phi : G \to H$

```
variable (G H : Type) [Group G] [Group H] (φ : G →* H)
```

我们可以使用QuotientGroup.lift和 $\phi|_{\text{range}\,\phi} : G \to \text{range}\,\phi$ 来构造出 $G/\ker\phi \to H$ 的同态：

```
def quotKerToImage : G / φ.ker →* φ.range :=
  QuotientGroup.lift _ φ.rangeRestrict (by simp)
```

自测 试证明quotKerToImage是一个双射。

如果 H, N 是 G 的两个子群，且 N 正规，那么我们可以使用 $\iota : H \to HN$ 和QuotientGroup.map来构造一个 ${}^{H}/_{H\sqcap N} \to {}^{HN}/_{N}$ 的同态。

```
def quotIntersect :
    H / (N.subgroupOf H) →*
    ((H  N : Subgroup G) : Type) / (N.subgroupOf (H  N)) :=
  QuotientGroup.map _ _ (Subgroup.inclusion (by simp)) (by simp)
```

自测 试证明quotIntersect是个双射。

假如 G 是一个群，$N \le M$ 是两个正规子群。因为 $N \le M$，我们根据QuotientGroup.map可以构造出 $G/N \to G/M$；而此同态可以下降为 $\frac{G/N}{M/N} \to G/M$：

```
variable (G : Type) [Group G] (N : Subgroup G) [N.Normal] (M : Subgroup G)
    [M.Normal] (h : N ≤ M)

def quotQuotToQuot :
    (G / N) / (Subgroup.map (QuotientGroup.mk' N) M) →* G / M :=
  QuotientGroup.lift _
    (QuotientGroup.map _ _ (.id _) h) (by sorry)
```

需要注意的是我们需要 M/N 作为 G/N 的子群，所以我们这里取 M 在 $G \to G/N$ 的像。

自测 试证明quotQuotToQuot是双射。

3.1.3 Ext 群的描述

在本章的习题中，我们将继续练习关于群的操作。假如 A, B 是两个阿贝尔群

```
variable (A B : Type) [AddCommGroup A] [AddCommGroup B]
```

我们说 B 由阿贝尔群 E 扩张 A 是指一个短正合序列。

$$0 \longrightarrow A \xrightarrow{f} E \xrightarrow{g} B \longrightarrow 0$$

在 Lean 中，我们可以实现为

```
structure PreExtension where
  carrier : Type
  [ab : AddCommGroup carrier]
  f : A →+ carrier
  g : carrier →+ B
  injective : Function.Injective f
  surjective : Function.Surjective g
  exact : Function.Exact f g
```

两个群扩张 $0 \to A \to E \to B \to 0$ 和 $0 \to A \to E' \to B \to 0$ 的同态是指满足如下交换图的群同态 $h : E \to E'$

$$\begin{array}{ccccccccc} 0 & \longrightarrow & A & \longrightarrow & E & \longrightarrow & B & \longrightarrow & 0 \\ & & \| & & \downarrow{\scriptstyle h} & & \| & & \\ 0 & \longrightarrow & A & \longrightarrow & E' & \longrightarrow & B & \longrightarrow & 0 \end{array}$$

在 Lean 中我们实现为

```
structure PreExtensionHom extends E₁ →+ E₂ where
  comm_f : toAddMonoidHom.comp E₁.f = E₂.f
  comm_g : E₂.g.comp toAddMonoidHom = E₁.g
```

自然而然两个群扩张 $0 \to A \to E \to B \to 0$ 和 $0 \to A \to E' \to B \to 0$ 的同构就是互反的同态：

```
structure PreExtensionIso where
  hom : PreExtensionHom E₁ E₂
  inv : PreExtensionHom E₂ E₁
  hom_inv_id : hom.comp inv = .id _
  inv_hom_id : inv.comp hom = .id _
```

那么 $\mathrm{Ext}(A, B)$ 就被定义为了所有群扩张 $0 \to A \to E \to B \to 0$ 的等价类：

```
abbrev Extension :=  Quotient (α := PreExtension A B)
  { r E₁ E₂ := Nonempty (PreExtensionIso E₁ E₂)
    iseqv :=
    { refl E := ⟨PreExtensionIso.refl E⟩
```

```
      symm := Nonempty.map PreExtensionIso.symm
      trans := by intro _ _ _ ⟨e⟩ ⟨e'⟩; exact ⟨e.trans e'⟩ } }
```

其中的 0 元素是 $0 \to A \to A \times B \to B \to 0$ 的等价类

```
def PreExtension.zero : PreExtension A B where
  carrier := A × B
  ab := inferInstance
  f := AddMonoidHom.inl A B
  g := AddMonoidHom.snd A B
  injective := by rintro a a' ⟨⟩; rfl
  surjective b := ⟨(0, b), rfl⟩
  exact a := rfl

instance : Zero (Extension A B) where
  zero := Quotient.mk'' .zero
```

两个元素的加法被定义为

$$[0 \to A \to E \to B \to 0] + [0 \to A \to E' \to B \to 0]$$

$$=$$

$$[0 \to A \to \frac{E \times_g E'}{\{(f_E(a), -f_{E'}(a)) | a \in A\}} \to B \to 0]$$

```
def PreExtension.add (E E' : PreExtension A B) : PreExtension A B where
  carrier := AddSubgroup.pullback E.g E'.g /
    (AddMonoidHom.prod E.f (-E'.f) |>.range).addSubgroupOf _
  ab := inferInstance
  f :=
  { toFun a := QuotientAddGroup.mk' _ ⟨(E.f a, 0), by simp [E.exact]⟩
    map_zero' := sorry
    map_add' a a' := sorry
  g := QuotientAddGroup.lift _
    { toFun e := E.g e.1.1
      map_zero' := by simp
      map_add' :=  sorry } <| by sorry
  injective := by sorry
  surjective := by sorry
  exact := by sorry

instance : Add (Extension A B) where
  add := Quotient.map₂ PreExtension.add sorry
```

一个元素 $[0 \to A \xrightarrow{f} E \to B \to 0]$ 被定义为 $[0 \to A \xrightarrow{-f} E \to B \to 0]$

```
def PreExtension.neg (E : PreExtension A B) : PreExtension A B where
  carrier := E
  ab := inferInstance
  f := - E.f
  g := E.g
  injective := sorry
  surjective := sorry
  exact a := sorry

instance : Neg (Extension A B) where
  neg := Quotient.map PreExtension.neg sorry
```

自测题 试证明上述表述的运算确实符合群的公理。

自测题 探索 $\mathrm{Ext}(-,-)$ 的双函子特性。

第四章　线性代数

4.1　基本设置

给定任意一个环 R，线性代数是研究 R-模的性质的学科。向量空间是在域上的模。

4.1.1　左模

在 Lean 中, 如果需要声明一个 M 是左 R-模，我们需要

```
variable (R : Type) [Ring R]
variable (M : Type) [AddCommGroup M] [Module R M]
```

对于模 M，N 和 P_i 来说，他们的乘积 $M \times N$，直和 $\bigoplus_i P_i$，直积 $\Pi_i P_i$ 在 Lean 中分别写作

```
variable (M : Type) [AddCommGroup M] [Module R M]
variable (N : Type) [AddCommGroup N] [Module R N]
variable {ι : Type} (P : ι → Type) [∀ i, AddCommGroup (P i)] [∀ i, Module R (P
    i)]

-- 模的乘积
#synth Module R (M × N)
-- 模的直积
#synth Module R (Π i, P i)
-- 模的直和
#synth Module R (⨁ i : ι, P i)
```

如果 R 是个**交换**环的话, 张量积[1] $M \otimes_R N$ 和 $\bigotimes_i P_i$ 和 R-线性映射 $\mathrm{Hom}_R(M, N)$ 也是 R-模：写作

```
-- R-线性映射 Hom_R(M, N)
#synth Module R (M →ₗ[R] N)

open TensorProduct
```

[1]非交换环上的张量积是存在的，目前 Lean 不支持

```
-- 张量积
#synth Module R (M ⊗[R] N)
#synth Module R (⨁[R] i : ι, P i)
```

4.1.2 右模

需注意的是，Lean 中没有右模的概念，如果需要声明一个右 R-模，我们需要

```
variable (Q : Type) [AddCommGroup Q] [Module R^mop Q]
```

即，Q 是一个左 R^{opp}-模。

4.1.3 双模

如果需要声明一个 (R, S)-双模，需要：

```
variable (B : Type) [AddCommGroup B] [Module R B] [Module S^mop B]
    [SMulCommClass R S^mop B]

open MulOpposite
example (r : R) (s : S) (b : B) :
    r • (op s) • b = op s • r • b := by rw [smul_comm]
```

即，B 是左 R-模，右 S-模且对于任意 $r \in R$, $s \in S$, $b \in B$, 我们都有 $r(bs) = (rb)s$。

4.1.4 向量空间

如果 F 是域，则 F-模被称作向量空间。在 Lean 中向量空间和模不做区分。如果需要声明一个有限维度的向量空间，需要

```
variable (F : Type) [Field F]
variable (V : Type) [AddCommGroup V] [Module F V]
variable [FiniteDimensional F V]
```

V 的维度是`Module.finrank F V`。

4.1.5 矩阵

一类重要的 R-模是矩阵，在 Lean 中矩阵的横坐标和纵坐标可以是任意的类型（不一定要是有限的, 比如

```
variable (R : Type) [Ring R]
variable {m n : Type}
```

```
#check Matrix m n R
```

规定了一个横坐标为 $\mathfrak{m}$ 纵坐标为 $\mathfrak{n}$ 的矩阵。这是因为 Lean 中的矩阵不过是函数 $\mathfrak{m} \to \mathfrak{n} \to R$ 的简写罢了。当我们需要有限的矩阵时，横纵坐标取`Fin m`和`Fin n`即可（`m`,`n`是自然数）。Lean 中提供了方便的方式书写矩阵，如 3×4 的整数矩阵

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

可以被写作

```
example : Matrix (Fin 3) (Fin 4) ℤ :=
!![1, 2, 3, 4;
   5, 6, 7, 8;
   9, 10, 11, 12]
```

4.2 线性映射，线性等价

假如 R 是一个交换环，A_i 和 B 是 R 模；我们在本章证明 $\mathrm{Hom}_R(\bigoplus_i A_i, B)$ 和 $\bigoplus_i \mathrm{Hom}_R(A_i, B)$ 作为 R-模是等价的。在 Lean 中

```
variable (R : Type) [CommRing R]
variable (ι : Type) [DecidableEq ι]
variable (A : ι → Type) [∀ i, AddCommGroup (A i)] [∀ i, Module R (A i)]
variable (B : Type) [AddCommGroup B] [Module R B]

open DirectSum

example : ((⨁ i, A i) →ₗ[R] B) ≃ₗ[R] (Π i : ι, A i →ₗ[R] B) :=
LinearEquiv.ofLinear
  - - - -
```

这里的`DecidableEq ι`是因为直和中需要脚标集合具有 decidable 的等式；不过在经典逻辑下所有的集合都有 decidable 的等式，所以我们不用太过担心。这里的四个下划线分别需要 R 线性映射 $F : \mathrm{Hom}_R(\bigoplus_i A_i, B) \to \prod_i \mathrm{Hom}_R(A_i, B)$、$G : \prod_i \mathrm{Hom}_R(A_i, B) \to \mathrm{Hom}_R(\bigoplus_i A_i, B)$ 并证明 $F \circ G = 1$、$G \circ F = 1$。让我们先来构造 F：

```
example : ((⨁ i, A i) →ₗ[R] B) ≃ₗ[R] (Π i : ι, A i →ₗ[R] B) :=
LinearEquiv.ofLinear
  { toFun f i := f ∘ₗ DirectSum.lof R ι A i
```

```
    map_add' := by intro f g; ext i a; rfl
    map_smul' := by intro r f; ext i a; rfl }
  _ _ _
```

toFun是说假如 $f : \bigoplus_i A_i \to B$ 是一个 R 线性映射，则 $\prod_i \mathrm{Hom}_R(A_i, B)$ 的第 i 项应该是 $x \mapsto f(0, ..., 0, 1, 0, ..., 0)$ 其中唯一非 0 项在第 i 处。这里$\circ_l$表示线性映射的复合，DirectSum.lof代表着典型的线性映射 $A_i \to \bigoplus_i A_i$。

现在我们来看 G

```
example : ((⨁ i, A i) →ₗ[R] B) ≃ₗ[R] (Π i : ι, A i →ₗ[R] B) :=
LinearEquiv.ofLinear
  _
  { toFun f := DirectSum.toModule _ _ _ f
    map_add' f g := by ext; simp
    map_smul' r f := by ext; simp }
  _ _
```

DirectSum.toModule是说想要从构建 $\bigoplus_i A_i \to B$ 的线性映射，只需要对于任意 i 构造出来 $A_i \to B$ 的线性映射。而这恰恰是 $\mathrm{Hom}_R(A_i, B)$ 的直积的含义。

习题 构建 $\mu : \bigoplus_i \mathrm{Hom}_R(B, A_i)$ 到 $\mathrm{Hom}_R(B, \bigoplus_i A_i)$ 的映射

习题 证明 μ 个单射。「可能需要DFinsupp.finset_sum_apply和DirectSum.sum_support_of」

4.3 线性映射和矩阵

假如 F 是一个域，V 是一个 n 维的向量空间且 $B = \{B_0, ..., B_{n-1}\}$ 是一组基，那么我们有 $\mathrm{End}_F V \cong \mathrm{Mat}_n(F)$。在这一小节中我们实现这个等价。首先我们声明 V 和 F

```
variable {F : Type} [Field F]
variable {V : Type} [AddCommGroup V] [Module F V]
variable {n : ℕ} (B : Module.Basis (Fin n) F V)
```

对于任意 $f : \mathrm{End}_F V$，我们将 $f(B_j)$ 写作 $\sum_i a_{ij} B_j$ 的形式，由此得到矩阵 $[a_{ij}]$。在 Lean 中，我们写作

```
def endToMatrix : Module.End F V → Matrix (Fin n) (Fin n) F :=
  fun f => Matrix.of fun i j => B.equivFun (f (B j)) i
```

这里的Matrix.of是将一个 $\{0, ..., n-1\} \to \{0, ..., n-1\} \to F$ 的函数看作一个矩阵。我们需要这个额外的一步是因为尽管矩阵被定义为了函数，但矩阵的乘法和函数的乘法不

相同，所以需要做这个区分。`B.equivFun`是从 B 这组基得到的同构 $V \cong F^n$。根据定义我们可以知道以下交换

$$\begin{array}{ccc} V & \xrightarrow{\texttt{B.equivFun}} & F^n \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \texttt{endToMatrix}(f)\texttt{.mulVecLin}} \\ V & \xrightarrow{\texttt{B.equivFun}} & F^n \end{array} \tag{4.1}$$

其中`Matrix.mulVecLin`是指矩阵和向量的乘法。

```
lemma endToMatrix_spec (f : Module.End F V) :
    B.equivFun ∘ₗ f = (endToMatrix B f).mulVecLin ∘ₗ B.equivFun := by
  apply B.ext
  intro j
  ext i
  simp [Matrix.mulVec, dotProduct, Finsupp.single_apply]
  rfl
```

对于任意矩阵 M, N 显然有 $(MN)v = M(Nv)$，上述交换图告诉我们`endToMatrix`保留乘法[2]:

```
lemma endToMatrix_mul (f g : Module.End F V) :
    endToMatrix B (f * g) = endToMatrix B f * endToMatrix B g := by
  have eq₁ := calc (endToMatrix B (f * g)).mulVecLin
    _ = B.equivFun ∘ₗ (f * g) ∘ₗ B.equivFun.symm := sorry
    _ =
      B.equivFun ∘ₗ
      (f ∘ₗ (B.equivFun.symm ∘ₗ B.equivFun) ∘ₗ g) ∘ₗ
      B.equivFun.symm := by
      sorry
    _ =
      (B.equivFun ∘ₗ f ∘ₗ B.equivFun.symm) ∘ₗ
      (B.equivFun ∘ₗ g ∘ₗ B.equivFun.symm) := by sorry
    _ = (endToMatrix B f).mulVecLin ∘ₗ (endToMatrix B g).mulVecLin := by sorry
  apply Matrix.ext_of_mulVec_single
  sorry
```

习题 假如我们考虑两个有限维向量空间之间的线性映射 $\mathrm{Hom}_R(V, W)$，并且 B_V 和 B_W 是向量空间 V, W 的基，请推广`endToMatrix_spec`

[2]以下代码需要一些 type hint，为了保持印刷的可读性再次省略

另外一方面,对于任意矩阵 M 来说 $V \longrightarrow F^n \xrightarrow{M.\text{mulVecLin}} F^n \longrightarrow V$ 是一个线性映射：

```
def matrixToEnd (M : Matrix (Fin n) (Fin n) F) : Module.End F V :=
  B.equivFun.symm ∘ₗ M.mulVecLin ∘ₗ B.equivFun
```

再次根据交换图4.1，不难看出endToMatrix和matrixToEnd是互逆的。也就是说，我们有群的同态

```
def endEquivMatrix :
    Module.End F V ≃* Matrix (Fin n) (Fin n) F where
  toFun := endToMatrix B
  invFun := matrixToEnd B
  left_inv := sorry
  right_inv := sorry
  map_mul' := sorry
```

习题 证明 $f+g$ 对应的矩阵是 f 对应的矩阵加 g 对应的矩阵，λf 对应的矩阵是 λ 乘 f 对应的矩阵。

4.4 如何定义模的示例

在本小节中，我们展示如何构建一个模。我们用的例子是，如果 R 是一个环，Q 是一个左 R-模，n 为自然数，则 Q^n 是一个左 $\mathrm{Mat}_n(R)$-模；定义为 $(m_{ij})\cdot(q_j)=\sum_j m_{ij}q_j$。在 Lean 中，我们需要验证 $1v=v$，$(MN)v=M(Nv)$，$M0=0$，$(M+N)v=Mv+Nv$，$0v=0$,$M(v+w)=Mv+Mw$，其中 M,N 是 R-矩阵，v,w 是 Q^n。

```
instance : Module (Matrix (Fin n) (Fin n) R) (Fin n → Q) where
  smul M v i :=  j, M i j • v j
  one_smul v := ...
  mul_smul M N v := ...
  smul_zero M := ...
  smul_add M v w := ...
  add_smul M N v := by
    ext i
    change  _, _ = ( j, M i j • v j) + ( _, _)
    simp [add_smul, Finset.sum_add_distrib]
  zero_smul v := ...
```

如果 $f:Q\to Q'$ 是 R 线性的，$Q^n\ni v\mapsto(f(v_i))\in Q'^n$ 则是 $\mathrm{Mat}_n(R)$-线性的：

```
variable (Q' : Type) [AddCommGroup Q'] [Module R Q']
variable (f : Q →ₗ[R] Q')

example : (Fin n → Q) →ₗ[Matrix (Fin n) (Fin n) R] (Fin n → Q') :=
{ toFun := fun v i => f (v i)
  map_add' := by intro v w; ext i; simp
  map_smul' := by
    intro M v
    ext i
    change f (  j, M i j • v j) =   j, _
    simp
```

习题 验证 $1 : Q \to Q$ 对应着 $1 : Q^n \to Q^n$。

习题 验证上述构造保线性映射的复合。

习题 实现：如果 n 不为 0，（`[NeZero n]`），则对于任意 $\mathrm{Mat}_n(R)$-模 P 来说，$E_{11}P$ 是一个 R-模。且如果有 $\mathrm{Mat}_n(R)$-线性的映射 $P \to P'$，就有对应的 R 线性映射 $E_{11}P \to E_{11}P'$。

第五章　环和模的局部化

在本章中，我们假设所有遇到的环都是交换环，所有的模都是左模。对于任意一个交换环 R, 如果 $S \subseteq R$ 是一个子幺半群[1]，我们构造 $S^{-1}R$ 为 $R \times S$ 以下关系的等价类：

$$(r,s) \sim (r',s') \iff \text{存在} t \in S \text{满足} t(rs' - r's) = 0$$

我们通常把 (r,s) 所在的等价类记作 $\frac{r}{s}$。所以 $S^{-1}R$ 就是以下集合

$$\left\{ \frac{r}{s} \middle| r \in R, s \in S \right\}$$

其中 $\frac{r}{s} = \frac{r'}{s'}$ 当且仅当存在 $t \in S$ 满足 $t(rs' - r's) = 0$。$S^{-1}R$ 上的加法和乘法定义为

$$\frac{r}{s} \cdot \frac{r'}{s'} = \frac{rr'}{ss'}, \ \frac{r}{s} + \frac{r'}{s'} = \frac{rs' + r's}{ss'}$$

同理，如果 M 是一个左 R-模，可以构造 $S^{-1}M$。我们可以看出 $S^{-1}M$ 是一个 $S^{-1}R$ 模：对于任意 $r \in R$,$s, s' \in S$,$m \in M$，我们有

$$\frac{r}{s} \cdot \frac{m}{s'} = \frac{r \cdot m}{ss'}$$

最常用的 R 子幺半群有 $\{f^n | n \in \mathbb{N}\}$ 和 $R - \mathfrak{p}$，此时没有歧义的情况下我们将局部环分别记作 R_f 和 $R_{\mathfrak{p}}$。有时也将 $S^{-1}R$ 记作 $R[S^{-1}]$

5.1　具体构造

在 Lean 中环的局部化和模的局部化分别使用`Localization`和`LocalizedModule`来实现：让`S`是任何一个子幺半群，我们用`Localization S`和`LocalizedModule S M`表示 $S^{-1}R$ 和 $S^{-1}M$。且 Mathlib 中已有 $S^{-1}R$ 的交换环实例和 $S^{-1}M$ 作为 R-模和 $S^{-1}R$-模的实例。

```
variable {R : Type} [CommRing R] (S : Submonoid R)
variable (M : Type) [AddCommGroup M] [Module R M]

#check Localization S
```

[1]即 $1 \in S$ 且对于任意两个 $s,t \in S$ 均有 $st \in S$

```
#synth CommRing (Localization S)

#check LocalizedModule S M

#synth Module R (LocalizedModule S M)

#synth Module (Localization S) (LocalizedModule S M)
```

我们可以通过Localization.add_mk、Localization.mk_mul、LocalizedModule.mk_add_mk、LocalizedModule.smul'_mk和LocalizedModule.mk_smul_mk来表示局部环的加法、乘法和 $S^{-1}M$ 的 R-模和 $S^{-1}R$-模结构。

```
example (r r' : R) (s s' : S) :
    Localization.mk r s + Localization.mk r' s' =
    Localization.mk (s * r' + s' * r) (s * s') := by
  rw [Localization.add_mk]

example (r r' : R) (s s' : S) :
    Localization.mk r s * Localization.mk r' s' =
    Localization.mk (r * r') (s * s') := by
  rw [Localization.mk_mul]

example (m m' : M) (s s' : S) :
    LocalizedModule.mk m s + LocalizedModule.mk m' s' =
    LocalizedModule.mk (s' • m + s • m') (s * s') := by
  rw [LocalizedModule.mk_add_mk]

example (r : R) (m : M) (s : S) :
    r • LocalizedModule.mk m s =
    LocalizedModule.mk (r • m) s := by
  rw [LocalizedModule.smul'_mk]

example (r : R) (m : M) (s s' : S) :
    Localization.mk r s • LocalizedModule.mk m s' =
    LocalizedModule.mk (r • m) (s * s') := by
  rw [LocalizedModule.mk_smul_mk]
```

习题 试证明 $S^{-1}R$ 是平凡的当且仅当 $0 \in S$.

5.2 泛性质

在交换代数中，我们通常不会特别在意环或者模的局部化的具体实现。我们在意的是以下泛性质：如果用 j 表示 $R \to S^{-1}R$, 令 $f : R \to T$ 为任意一个满足对于任意 $s \in S$, $f(s)$ 均为可逆的环同构，则有唯一一个 $g : S^{-1}R \to T$ 满足 $f = g \circ j$。在这里我们有

```
IsLocalization.lift {M : Submonoid R} {S : Type} [CommRing S]
  [Algebra R S] {P : Type u_3} [CommRing P] [IsLocalization M S] {g : R →+* P}
    (hg : ∀ (y : M), IsUnit (g y)) :
  S →+* P
```

比如我们可以用此来构造 $\mathbb{Z}[\mathbb{Z}^0]$ 到 $\mathbb{Q}$ 的同构：

```
def ex01 : Localization (nonZeroDivisors ℤ) →+* ℚ :=
IsLocalization.lift (M := nonZeroDivisors ℤ) (g := algebraMap ℤ ℚ) (by
  rintro ⟨z, hz⟩
  simpa using hz)
```

显然，这个同构 ϕ 是单射: 假如 $\phi(x/y) = 0$, 则有 $x/y \in \mathbb{Q}$ 为 0, 则 $x = 0$, 故 $x/y \in \mathbb{Z}[\mathbb{Z}^0]$ 为 0:

```
  rw [RingHom.injective_iff_ker_eq_bot, eq_bot_iff]
  intro x hx
  induction x using Localization.induction_on with | H x =>
  ...
```

这里的`induction`就是将任意一个在局部化环中的元素写成“分子分母”的形式。ϕ 也是满射因为 $m/n \in \mathbb{Q}(n \neq 0$ 且 $\gcd(m, n) = 1)$ 则，$\phi(m/n) = m/n$：

```
example : Function.Surjective ex01 := by
  rintro ⟨m, n, hn, hmn⟩
  refine ⟨Localization.mk m ⟨n, ?_⟩, ?_⟩
  • ...
  • ...
```

所以 $\mathbb{Q}$ 其实是 $\mathbb{Z}$ 的局部化。在 Lean 中这体现为`IsLocalization`：我们说某个 R 代数 S 是 R 在 M 的局部化当且仅当

1. 任意 $x \in M$ 在 R 中可逆；
2. 任意 $z \in S$ 都有 $r \in R$、$m \in M$ 使得 $z \cdot m = r$；
3. 如果 $x, y \in R$ 在 S 中相等，则存在 $c \in M$ 使得 $cx = cy$。

习题 试证明 $\mathbb{Q}$ 满足上述条件 $R=\mathbb{Z}$，$M=\mathbb{Z}^0$.

```
instance : IsLocalization (nonZeroDivisors ℤ) ℚ := sorry
```

同样的道理，我们可以考虑局部模的泛性质。如果 $S\subseteq R$ 是一个子幺半群、$f: M\to M'$ 是一个 R 线性同构，我们说 M' 是 M 在 S 处的局部化当且仅当

1. 任意 $x\in S$, x 作用在 M' 上的标量乘法作为 R 线性映射是可逆的;
2. 对于任意 $y\in M'$, 都有 $m\in M$ 和 $s\in S$ 使得 $s\cdot y=f(m)$;
3. 对于任意 $x_1,x_2\in M$，如果 $f(x_1)=f(x_2)$, 则存在 $c\in S$ 使得 $cx_1=cx_2$。

如果 $f: M\to M'$ 满足上述性质，则有以下泛性质：如果 $g: M\to M''$ 是 R 线性的，且对于任意 $x\in S$, x 作用在 M'' 上的标量乘法作为 R 线性映射是可逆的，则存在唯一一个 R 线性 $g': M'\to M''$ 满足 $g=g'\circ f$。

5.3 例子

需注意的是这里的例子通常不是最优解，是为了展示 API 而做的例子。

5.3.1 $S^{-1}M$ 和 $S^{-1}R\otimes_R M$ 之间的 R 同构

我们首先证明通过 $j: R\to S^{-1}R$，我们可以把 $S^{-1}R$ 看作 R 作为 R-模在 S 的局部化。

```
instance : IsLocalizedModule S (Algebra.ofId R (Localization S)).toLinearMap
    where
  map_units s := by
    rw [isUnit_iff_exists]
    let φ : Localization S ₗ→[R] Localization S :=
    { toFun x := x.liftOn (fun r t => Localization.mk r (s * t)) (by ...)
      map_add' := ...
      map_smul' := ... }
    refine ⟨φ, ?_, ?_⟩
    ...
  surj' := ...
  exists_of_eq := ...
```

这里`Algebra.ofId R (Localization S)`是 $R\to S^{-1}R$ 的代数同构。这里的难点是证明任意 $s\in S$，s 作用在 $S^{-1}R$ 上的标量乘法是可逆的。所以我们需要构造 $\phi: S^{-1}R\to S^{-1}R$ 的线性映射，我们定义为 $\frac{r}{t}\mapsto\frac{r}{st}$。我们用`divByAux s`和`divBy s`来表示 $r\mapsto\frac{r}{s}$ 的和 $\frac{r}{s}\mapsto\frac{r}{st}$ 映射:

```
def divByAux (s : S) : R →ₗ[R] Localization S :=
{ toFun r := Localization.mk r s
  map_add' := by intro r r'; simp [Localization.add_mk_self]
  map_smul' := by intro r x; simp [Localization.smul_mk] }

def divBy (s : S) : Localization S →ₗ[R] Localization S :=
IsLocalizedModule.lift S
  ((Algebra.ofId R (Localization S)).toLinearMap)
  (divByAux (s := s))
  (...)

@[simp]
lemma divBy_apply (r : R) (s t : S) :
    divBy S s (Localization.mk r t) = Localization.mk r (s * t) := ...
```

所以我们可以通过模局部化的泛性质将 $M \to S^{-1} \otimes_R M$ 诱导为 $S^{-1}M \to S^{-1} \otimes_R M$ 的映射。

```
def ex02 : LocalizedModule S M ₗ→[R] Localization S ⊗[R] M :=
IsLocalizedModule.lift S (LocalizedModule.mkLinearMap S M)
  { toFun m := 1 ⊗ₜ m
    map_add' := by simp [tmul_add]
    map_smul' := by simp }
  (by ...)
```

反过来，我们则需要构造 $S^{-1}R \to \mathrm{Hom}_R(M, S^{-1}M)$ 的双线性映射：由于 $S^{-1}M$ 的泛型，我们只需构造 $\mathrm{Hom}_R(R, \mathrm{Hom}(M, S^{-1}M))$ 的映射，我们定义为 $r \mapsto r \cdot \left(m \mapsto \frac{m}{1}\right)$

```
def ex03 : (Localization S ⊗[R] M) ₗ→[R] LocalizedModule S M :=
  TensorProduct.lift <| IsLocalizedModule.lift S ((Algebra.ofId R
    (Localization S)).toLinearMap)
    { toFun r := r • LocalizedModule.mkLinearMap S M
      map_add' := by simp [add_smul]
      map_smul' := by simp [mul_smul] } <| by ...
```

习题 试证明ex02和ex03是互逆的

```
def ex04 : LocalizedModule S M ≃ₗ[R] Localization S ⊗[R] M :=
  LinearEquiv.ofLinear (ex02 ..) (ex03 ..)
    sorry sorry
```

5.3.2 $(ST)^{-1}R$ 和 $\bar{S}^{-1}\left(T^{-1}R\right)$ 之间的同构

如果 S,T 是两个子幺半群，则一个常用的同态是 $(ST)^{-1}R \cong \bar{S}^{-1}\left(T^{-1}R\right)$，这里 $\bar{S}$ 是指 S 在 $T^{-1}R$ 下的像。在 Lean 中`S  T`用来表示 ST;`Submonoid.map`用来表示子幺半群的像。故我们需要构造

```
def ex05 : Localization (S  T) →+* Localization (S.map (algebraMap R
    (Localization T))) :=
```

和

```
def ex06 : Localization (S.map (algebraMap R (Localization T))) →+*
    Localization (S  T) :=
```

根据泛性质，若想构造 $(ST)^{-1}R \to \bar{S}^{-1}(T^{-1}R)$，我们只需要构造 $\phi : R \to \bar{S}^{-1}(T^{-1}R)$，并证明可逆性质。显然我们可以取 ϕ 为 $\bar{S}^{-1}(T^{-1}R)$ 的 R-代数结构映射。$\phi(st)$ 的可逆性是因为 $\phi(s)$ 的逆为 $\frac{1}{\frac{s}{1}}$ 而 $\phi(t)$ 的逆为 $\frac{\frac{1}{t}}{1}$。

```
def ex05 : Localization (S  T) →+* Localization (S.map (algebraMap R
    (Localization T))) :=
  IsLocalization.lift (M := S  T) (g := algebraMap R _) <| by
    rintro ⟨x, hx⟩
    rw [← SetLike.mem_coe, Submonoid.coe_sup] at hx
    rcases hx with ⟨s, hs, t, ht, rfl⟩
    simp only [map_mul, IsUnit.mul_iff] at hx ⊢
    constructor
    · rw [isUnit_iff_exists]
      refine ⟨Localization.mk 1 ⟨algebraMap _ _ s, by simpa using ⟨s, hs,
    rfl⟩⟩, ?_, ?_⟩ <;> sorry
    · rw [isUnit_iff_exists]
      refine ⟨Localization.mk (Localization.mk 1 ⟨t, ht⟩) ⟨1, by simp⟩, ?_, ?_⟩
    <;> sorry
```

另一方面，构造 $\bar{S}^{-1}(T^{-1}R) \to (ST)^{-1}R$ 的映射，我们需要构造 $\psi : T^{-1}R \to (ST)^{-1}R$ 的映射并证明任意 $t \in T$,$\psi(t)$ 都是可逆的。根据 $T^{-1}R$ 的泛性质，由于 $T \leq ST$，我们可以讲 $R \to R$ 的显然态射下降到 $T^{-1}R \to (ST)^{-1}R$。

```
def ex06 : Localization (S.map (algebraMap R (Localization T))) →+*
    Localization (S  T) :=
  IsLocalization.lift (M := S.map (algebraMap R (Localization T)))
    (g := IsLocalization.map (M := T) (T := S  T) _ (RingHom.id _) ...) <| by
    ...
```

习题 证明上述的同构互逆，即

```
def ex07 : Localization (S   T) ≃+* Localization (S.map (algebraMap R
    (Localization T))) :=
  RingEquiv.ofHomInv (ex05 ..) (ex06 ..) sorry sorry
```

第六章　初等数论

6.1　同余和整除

6.1.1　Int.ModEq

在 Lean 中，讨论余数可以使用Int.ModEq或Nat.ModEq，其定义为

```
def Int.ModEq (n a b : ℤ) :=
  a % n = b % n

def Nat.ModEq (n a b : ℕ) :=
  a % n = b % n
```

也可以写作

```
a ≡ b [ZMOD n]
a ≡ b [MOD n]
```

值得注意的是这个记号是可用在calc中的，在这种场景下配合congr和gcongr是很方便的。在这里我们定义 $a_n = 2^{2n+1} - 2^{n+1} + 1$ 并证明如果 $n = 4k$，则 $a_n \equiv 1 \mod 5$：

```
1  def a (n : ℕ) : ℤ :=
2    2 ^ (2 * n + 1) - 2 ^ (n + 1) + 1
3
4  example (k : ℕ) : a (4 * k) ≡ 1 [ZMOD 5] :=
5    calc a (4 * k)
6      _ ≡ 2 ^ (8 * k + 1) - 2 ^ (4 * k + 1) + 1 [ZMOD 5] := by
7        simp only [a]
8        congr 1
9        ring
10     _ ≡ (2 ^ 4) ^ (2 * k) * 2 - (2 ^ 4) ^ k * 2 + 1 [ZMOD 5] := by
11       simp only [pow_add, pow_mul, Int.reducePow, pow_one, Int.ModEq.refl]
12     _ ≡ 1 ^ (2 * k) * 2 - 1 ^ k * 2 + 1 [ZMOD 5] := by
13       gcongr
14       · rfl
```

```
15        • rfl
16      _ ≡ 2 - 2 + 1 [ZMOD 5] := by
17        gcongr;
18        • simp
19        • simp
20      _ ≡ 1 [ZMOD 5] := by
21        simp
```

之前我们已经介绍过congr可以用来把一个等式的证明转化成多个等式证明的 tactic，比如 $x+y=x'+y'$ 会被转换成 $x=x'$ 和 $y=y'$。这里的gcongr的作用是把证明一个同余转化为多个同余的证明，比如在第 12 行，gcongr把

```
  (2 ^ 4) ^ (2 * k) * 2 - (2 ^ 4) ^ k * 2 + 1
≡ 1 ^ (2 * k) * 2 - 1 ^ k * 2 + 1 [ZMOD 5]
```

的证明转化为了

```
2 goals
case ...
k : ℕ
⊢ 2 ^ 4 ≡ 1 [ZMOD 5]
case ...
k : ℕ
⊢ 2 ^ 4 ≡ 1 [ZMOD 5]
```

这是因为gcongr通过对比式子左右两边,发现只需要证明 $2^4=1$ 并利用Int.ModEq.refl就可以完成证明。同样地道理，gcongr也可以被用来证明不等式：

```
example (n : ℕ) (hn : 2 ≤ n) : n ^ 2 + 5 ≤ (n + 1) ^ 2  := by
  calc n ^ 2 + 5
    _ = n ^ 2 + 2 * 2 + 1 := by ring
    _ ≤ n ^ 2 + 2 * n + 1 := by gcongr
    _ = (n + 1) ^ 2 := by ring
```

在这里gcongr比较了不等式左右两边发现了只需要使用hn就可以完成不等式的证明。这两个例子中都是因为有一些引理、定理（如Int.ModEq.refl和add_le_add）被打上了@[gongr]的标签所以gcongr就会使用这些引理。

习题 设 $a_n=2^{2n+1}-2^{n+1}+1$，$b_n=2^{2n+1}+2^{n+1}+1$。对于任意 n，证明 a_nb_n 被 5 整除。

完整的示例

在本小节，我们证明对于任意非零自然数 n,$\phi(n)|n!$。

```
theorem totient_dvd_factorial (n : ℕ) (hn : 0 < n) : n.totient ∣ n.factorial :=
    by
  let s := Nat.factorization n
```

我们将 $s : \mathbb{N} \to \mathbb{N}$ 定义为 $p \mapsto v_p(n)$，即 $p^{s(p)} \mid n$ 且 $p^{s(p)+1} \nmid n$。这里只有有限多个 p 使得 $s(p)$ 不为零，这些 p 的集合被记作`s.support`。我们有

$$\phi(n) = \phi\left(\prod_p p^{s(p)}\right) = \prod_p p^{s(p)-1}(p-1) = \left(\prod_p p^{s(p)-1}\right)\left(\prod_p (p-1)\right)$$

```
have eq1 : n.totient = (∏ p ∈ s.support, p ^ (s p - 1)) * (∏ p ∈ s.support,
    (p - 1)) := by
    rw [Nat.totient_eq_prod_factorization (by omega), ←
    Finset.prod_mul_distrib]
    rfl
```

另一方面，显然 $n! = n \cdot (n-1)!$

```
  have eq2 : n.factorial = n * (n - 1).factorial := by
    conv_lhs => rw [show n = n - 1 + 1 by omega, Nat.factorial_succ, show n -
    1 + 1 = n by omega]
```

所以我们只需要证明 $\prod_p p^{s(p)-1} \mid n$ 且 $\prod_p (p-1) \mid (n-1)!$

```
  rw [eq1, eq2]
  gcongr
```

其中 $\prod_p p^{s(p)-1} \mid n$ 是因为 $n = \prod_p p^{s(p)}$ 且 $s(p) - 1 \le s(p)$:

```
  • rw [show n = ∏ p ∈ s.support, p ^ (s p) from
      Nat.factorization_prod_pow_eq_self (n := n) (by omega) |>.symm]
    apply Finset.prod_dvd_prod_of_dvd
    intro p hp
    exact Nat.pow_dvd_pow _ (by omega)
```

而 $\prod_p (p-1) \mid (n-1)!$ 则是因为 $\prod_p (p-1) = \prod_{\{1 \le j < n \mid s(j+1) \neq 0\}} j$

```
    ...
    rw [show ∏ p ∈ s.support, (p - 1) = ∏ j ∈ Finset.Ico 1 n with j + 1 ∈
    s.support, j by
      fapply Finset.prod_bij
      • exact fun p _ => p - 1
      • intro p hp
        simp only [Nat.support_factorization, Nat.mem_primeFactors, ne_eq,
    Finset.mem_filter,
```

```
        Finset.mem_Ico, s] at hp ⊢
      have p_ineq1 : 2 ≤ p := hp.1.two_le
      have p_ineq2 : p ≤ n := by
        exact Nat.le_of_dvd (by omega) hp.2.1
      simp only [show p - 1 + 1 = p by omega]
      exact ⟨⟨by omega, by omega⟩, hp.1, hp.2.1, by omega⟩
    · intro p hp q hq h
      simp only [Nat.support_factorization, Nat.mem_primeFactors, ne_eq, s]
  at hp hq ⊢
      have p_ineq : 2 ≤ p := hp.1.two_le
      have q_ineq : 2 ≤ q := hq.1.two_le
      omega
    · rintro i hi
      simp only [Finsupp.mem_support_iff, ne_eq, Finset.mem_filter,
  Finset.mem_Ico] at hi
      exact ⟨i + 1, by simpa using hi.2, by omega⟩
    · simp]
  ...
```

习题 补全上述证明

习题 当 n 是奇数时，证明 $n \mid 2^{n!} - 1$。

6.1.2 $\mathbb{Z}/n\mathbb{Z}$

另一种同余类的定义为

```
def ZMod : ℕ → Type
  | 0 => ℤ
  | n + 1 => Fin (n + 1)
```

也就是说ZMod n为 $\mathbb{Z}/n\mathbb{Z}$。当 p 为一个素数时, $\mathbb{Z}/p\mathbb{Z}$ 是一个域; 在 Lean 中，我们写作

```
variable (p : ℕ) [hp : Fact p.Prime]

#synth Field (ZMod p)
```

这里需要用到Fact，是因为Nat.Prime并不是一个类型类，所以我们用Fact p.Prime将这个条件加到类型类的搜索中。

当然ZMod和Nat.ModEq和Int.ModEq之间可以通过 ZMod.intCast_eq_intCast_iff 和 ZMod.natCast_eq_natCast_iff来链接:

```
#check ZMod.intCast_eq_intCast_iff
/-
ZMod.intCast_eq_intCast_iff (a b : ℤ) (c : ℕ) : ↑a = ↑b ↔ a ≡ b [ZMOD ↑c]
-/

#check ZMod.natCast_eq_natCast_iff
/-
ZMod.natCast_eq_natCast_iff (a b c : ℕ) : ↑a = ↑b ↔ a ≡ b [MOD c]
-/
```

这里的上箭头↑分别代表着整数和自然数到 $\mathbb{Z}/n\mathbb{Z}$ 的转换。反过来ZMod.val则是 $\mathbb{Z}/n\mathbb{Z}$ 到 $\mathbb{Z}$ 的类型转换。

二次特性

在本小节我们证明对于任意奇素数 p，我们都有 $\{x \in \mathbb{Z}/p\mathbb{Z} | x$是非零的平方数$\}$ 个数是 $\frac{p-1}{2}$。首先我们定义群同构 $\mathsf{sq} : (\mathbb{Z}/p\mathbb{Z})^\times \to (\mathbb{Z}/p\mathbb{Z})^\times$。

```
@[simps]
def sq : (ZMod p) →* (ZMod p) :=
  { toFun := fun x => x ^ 2
    map_one' := by simp
    map_mul' x y := by ext; simpa using by ring }
```

我们想要证明

```
example : Set.ncard { x : (ZMod p) | IsSquare x } = (p - 1) / 2 := by
```

这里Set.ncard是把集合的基看作自然数（无穷集取 0）。

我们首先证明 $\{x \in (\mathbb{Z}/p\mathbb{Z})^\times | x$是个平方$\}$ 等于 sq 的像。

```
  have set_eq : { x : (ZMod p) | IsSquare x } = MonoidHom.range (sq p) := by
    ext x; simp [IsSquare, pow_two, eq_comm]
  rw [set_eq]
```

为了更方便地用群同构定理，我们证明 $\ker \phi = 1, -1$

```
  have card_eq₀ : Set.ncard {x : (ZMod p) | x ^ 2 = 1} = 2 := by
    sorry
  rw [Set.ncard_eq_toFinset_card, Set.Finite.toFinset_setOf] at card_eq₀
```

这里Set.ncard_eq_toFinset_card是用来在Finset.card和Set.ncard中做转换。

又因为 $(\mathbb{Z}/p\mathbb{Z})^\times / \ker \mathsf{sq} \cong \operatorname{range} \mathsf{sq}$，我们有 $|(\mathbb{Z}/p\mathbb{Z})^\times / \ker \mathsf{sq}| = |\operatorname{range} \mathsf{sq}|$。

```
let e : (ZMod p) ⧸ MonoidHom.ker (sq p) ≃* MonoidHom.range (sq p) :=
    QuotientGroup.quotientKerEquivRange _

  have card_eq₁ := Fintype.card_congr e.toEquiv
```

因为 $p-1=|(\mathbb{Z}/p\mathbb{Z})^\times|=|(\mathbb{Z}/p\mathbb{Z})^\times/\ker \mathsf{sq}|\cdot|\ker \mathsf{sq}|$，我们可以得出想要的结论。

```
have eq := (MonoidHom.ker (sq p)).card_eq_card_quotient_mul_card_subgroup
```

习题 补全上述证明

6.2 高斯整数

对于任何一个不是平方的整数 d，我们可以考虑 $\mathbb{Z}[\sqrt{d}]:=\{a+b\sqrt{d}|a,b\in\mathbb{Z}\}$。在 Lean 中我们可以定义为

```
@[ext]
structure Zsqrt (d : ℤ) where
  re : ℤ
  im : ℤ
```

这里的`@[ext]`是让 Lean 自动生成在 $\mathbb{Z}[\sqrt{d}]$ 中，$x=y$ 当且仅当 $\mathfrak{Re}x=\mathfrak{Re}y$ 且 $\mathfrak{Im}x=\mathfrak{Im}y$。我们可以定义 $\mathbb{Z}[\sqrt{d}]$ 上的加法为 $(a+b\sqrt{d})+(a'+b'\sqrt{d})=(a+a')+(b+b')\sqrt{d}$，负数被定义为 $-(a+b\sqrt{d})=-a+(-b)\sqrt{d}$。

```
instance (d : ℤ) : Add (Zsqrt d) where
  add x y := ⟨x.re + y.re, x.im + y.im⟩

@[simp]
lemma add_re (d : ℤ) (x y : Zsqrt d) : (x + y).re = x.re + y.re := rfl

@[simp]
lemma add_im (d : ℤ) (x y : Zsqrt d) : (x + y).im = x.im + y.im := rfl

instance (d : ℤ) : Neg (Zsqrt d) where
  neg x := ⟨-x.re, -x.im⟩

@[simp]
lemma neg_re (d : ℤ) (x : Zsqrt d) : (-x).re = -x.re := rfl

@[simp]
```

```
lemma neg_im (d : ℤ) (x : Zsqrt d) : (-x).im = -x.im := rfl
```

同理 0 和 1 被定义为 $0+0\sqrt{d}$ 和 $1+0\sqrt{d}$。

```
instance (d : ℤ) : Zero (Zsqrt d) where
  zero := ⟨0, 0⟩

@[simp]
lemma zero_re (d : ℤ) : (0 : Zsqrt d).re = 0 := rfl

@[simp]
lemma zero_im (d : ℤ) : (0 : Zsqrt d).im = 0 := rfl

@[simp]
lemma one_re (d : ℤ) : (1 : Zsqrt d).re = 1 := rfl
@[simp]
lemma one_im (d : ℤ) : (1 : Zsqrt d).im = 0 := rfl
```

乘法被定义做 $(a+b\sqrt{d})\cdot(a'+b'\sqrt{d})=(aa'+dbb')+(ab'+a'b)\sqrt{d}$

```
instance (d : ℤ) : Mul (Zsqrt d) where
  mul x y := ⟨x.re * y.re + d * x.im * y.im, x.re * y.im + x.im * y.re⟩

@[simp]
lemma mul_re (d : ℤ) (x y : Zsqrt d) :
    (x * y).re = x.re * y.re + d * x.im * y.im := rfl

@[simp]
lemma mul_im (d : ℤ) (x y : Zsqrt d) :
    (x * y).im = x.re * y.im + x.im * y.re := rfl
```

这些定义给 $\mathbb{Z}[\sqrt{d}]$ 一个交换环的结构。

事实上，这个环结构不过是 $\mathbb{Z}[X]/\langle X^2-d\rangle$。这是因为我们有环同构 $\mathbb{Z}[X]\to\mathbb{Z}[\sqrt{d}]$ 定义为 $e:p\mapsto p(0+1\sqrt{d})$

```
abbrev toQuotPoly (d : ℤ) : ℤ[X] →+* Zsqrt d := Polynomial.aeval (⟨0, 1⟩ :
   Zsqrt d) |>.toRingHom
```

其中 $\ker e$ 是 $\langle X^2-d\rangle$：

```
lemma ker_toQuotPoly (d : ℤ) :
    RingHom.ker (toQuotPoly d) = Ideal.span {X ^ 2 - Polynomial.C d} :=
```

习题 试证明 e 是满射的。

第七章　代数数论

7.1　整数环

代数数论的研究对象之一是代数数域（有理数的有限扩张且特征为 0）和其整数环。在 Lean 中，要是想声明某个K是一个数域，我们需要:

```
variable (K : Type*) [Field K] [NumberField K]
```

其整数环被定义做NumberField.RingOfIntegers K或$\mathcal{O}$ K;其中后者需要open NumberField或者open scoped NumberField。需注意的是整数环的实现方式是

```
def RingOfIntegers : Type _ :=
  integralClosure ℤ K
```

也就是说，$\mathcal{O}_K$ 是 $\mathbb{Z}$ 在 K 中的（任意一个）整闭包。这就意味着下面的 Lean 代码证不出来：

```
example : ℤ = 𝓞 ℚ := by sorry
```

我们最好也只能证明二者是同构的（显然，他们都是 $\mathbb{Z}$ 在 $\mathbb{Q}$ 中的整闭包）

```
noncomputable example : ℤ ≃ [ℤ] 𝓞 ℚ := IsIntegralClosure.equiv ℤ ℤ ℚ _
```

$\mathcal{O}_K$ 的基础性质都是已经实现了的比如，K 是 $\mathcal{O}_K$ 的分式域, 整闭包。

```
#synth IsFractionRing (𝓞 K) K
-- RingOfIntegers.instIsFractionRing

#synth IsIntegralClosure (𝓞 K) ℤ K
-- RingOfIntegers.instIsIntegralClosureInt
```

例子　我们考虑 $\mathbb{Q}(i) \subseteq \mathbb{C}$。在 Lean 中，有很多种方法表示它

1. IntermediateField.adjoin或者写作 ℚ⟮I⟯。这是指包含了 i 的最小的 $\mathbb{C}/\mathbb{Q}$ 的子域扩张。

2. AdjoinRoot (X^2 + 1)。这是指商环 $\mathbb{Q}[X]/(X^2+1)$。

3. 自定义的结构

```
structure Q_I where
  re : ℚ
  im : ℚ
```

其中这些方式各有优劣。第一种来说，Lean 不能自动合成它是一个代数数域，但它是真正的子代数；第二种来说，它不是真正意义上的 $\mathbb{C}$ 的子 $\mathbb{Q}$-代数，但 Lean 知道它是代数数域；第三种来说，Lean 什么都不知道，但它的构造非常明确地指出了实数部分和虚数部分是怎样的，方便构造 norm 和 trace 或其他计算。

norm 和 trace

在 Lean 中，norm 和 trace 的定义依赖于左乘对应的矩阵的 determinant 和 trace 来定义，所以计算起来不是很容易。我们这里计算 i 在 $\mathbb{Q}(i)$ 中的 norm 和 trace 来举例。首先，我们用`IntermediateField`来举例，Lean 不知道`ℚ⟮I⟯`是个代数数域的原因是 Lean 不知道它作为 $\mathbb{Q}$-向量空间是有限维的。为了解决这个问题，我们先来构建一组基：$\{1, I\}$：

```
noncomputable def B : Module.Basis (Fin 2) ℚ ℚ⟮I⟯ :=
  .mk (v := ![1, ⟨I, mem_adjoin_simple_self ℚ I⟩]) sorry sorry
```

因为我们需要把 i 构造成为 $\mathbb{Q}(I)$ 的元素，所以我们需要`⟨I, mem_adjoin_simple_self ℚ I⟩`，第一个元素是复数 i，第二个元素是 $i \in \mathbb{Q}(i)$ 的证明。余下两个`sorry`分别证明线性不相关性和 $\{1, i\}$ 生成所有 $\mathbb{Q}(i)$。有了这一组基后，我们便可以得到代数数域的实例。

```
instance : NumberField ℚ⟮I⟯ where
  to_charZero := charZero ℚ⟮I⟯
  to_finiteDimensional := FiniteDimensional.of_fintype_basis B
```

而从 B 可以明显看出乘以 i 所对应的矩阵为

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

```
@[simp]
lemma B_leftMulMatrix :
    Algebra.leftMulMatrix B ⟨I, mem_adjoin_simple_self ℚ I⟩ =
    !![0, -1; 1, 0] := by
  ext i j
  fin_cases i <;> fin_cases j <;> sorry
```

所以 i 的 norm 就应该是 1

```
@[simp]
lemma norm_eq1 : Algebra.norm ℚ (⟨I, mem_adjoin_simple_self ℚ I⟩ : ℚ⟮I⟯) = 1 :=
    by
  rw [Algebra.norm_eq_matrix_det B, Matrix.det_fin_two]
  simp
```

需注意的是这里右边的 1 是一个有理数 $\mathbb{Q}$。如果我们需要整数的话，则我们需要考虑使用RingOfIntegers.norm

```
#check RingOfIntegers.norm
-- RingOfIntegers.norm {L} (K) [Field K] [Field L] [Algebra K L]
    [FiniteDimensional K L] [Algebra.IsSeparable K L] : 𝒪 L →* 𝒪 K
```

它的定义其实就是将Algebra.norm限制在分别的整数环上，所以我们只要有norm_eq1, 想要计算 i 在 $\mathcal{O}_{\mathbb{Q}}$ 的 norm 就很简单了。首先我们需要定义一个 $\mathcal{O}_{\mathbb{Q}(i)}$ 的元素 i

```
open Polynomial
abbrev i1 : 𝒪 ℚ⟮I⟯ :=
  ⟨⟨I, mem_adjoin_simple_self ℚ I⟩, ⟨X^2 + C 1, monic_X_pow_add_C _ (by
    norm_num), by simpa using by ext; simp⟩⟩
```

然后答案就呼之欲出了

```
example : RingOfIntegers.norm ℚ i1 = 1 := by
  ext
  simp [i1]
```

需注意的是这里的右边的 1 的类型其实是 $\mathcal{O}_{\mathbb{Q}}$，如果想要获得真正的整数还需要在左边复合相应的 $\mathcal{O}_{\mathbb{Q}} \cong \mathbb{Z}$。

相应的我们也可以在AdjoinRoot中实现相同的事情

```
@[simp]
lemma norm_eq2 :
  Algebra.norm ℚ (AdjoinRoot.root (X ^ 2 + 1 : ℚ[X])) = 1 := by sorry

noncomputable def i2 : 𝒪 (AdjoinRoot (X ^ 2 + 1 : ℚ[X])) :=
  ⟨AdjoinRoot.root _, ⟨X^2 + 1, monic_X_pow_add_C _ (by norm_num), by
    have := AdjoinRoot.isRoot_root (X ^ 2 + 1 : ℚ[X])
    simpa using this⟩⟩

example : RingOfIntegers.norm ℚ i2 = 1 := by
  ext
  simp [i2]
```

这里的i2是通过AdjoinRoot的 i 在 $\mathcal{O}_{\mathbb{Q}(i)}$ 的实现。相比于IntermediateField来说，我们已经通过AdjoinRoot.powerBasis来快速获得一组由根生成的基：

```
#check AdjoinRoot.powerBasis

-- AdjoinRoot.powerBasis {K} [Field K] {f : K[X]} (hf : f ≠ 0) :
-- PowerBasis K (AdjoinRoot f)
```

在我们的例子中 f 是 X^2+1, 显然不为 0

```
@[simp]
lemma norm_eq2 : Algebra.norm ℚ (AdjoinRoot.root (X ^ 2 + 1 : ℚ[X])) = 1 := by
  let B := AdjoinRoot.powerBasis (f := (X ^ 2 + 1 : ℚ[X]))
    (X_pow_add_C_ne_zero (by norm_num) _)

  have dim_eq : B.dim = 2 := by simp [B]; exact natDegree_X_pow_add_C
  have gen_eq : B.gen = AdjoinRoot.root (X ^ 2 + 1 : ℚ[X]) := by simp [B]
  rcases B with ⟨gen, dim, B, hB⟩
  simp only at *
  subst gen_eq
  subst dim
```

这是我们可以看到 infoview 中 B 的相关信息：

```
B : Module.Basis (Fin 2) ℚ (AdjoinRoot (X ^ 2 + 1))
hB : ∀ (i : Fin 2), B i = AdjoinRoot.root (X ^ 2 + 1) ^ ↑i
```

也就是说 B 是 $\{1, i\}$ 这一组基，剩下的也都显然了。

第八章　组合数学

8.1　什么是有限

在 Lean 中表示有限的方法有很多，它们各个都很相似却又不同。我们在此总结：

- `Finset` 有限集合在 Lean 中是列表（List）的商类的子类：商类上的等价关系是两个列表等价当且仅当这两个列表是彼此的排列；子类是指 Finset 是没有重复元素的列表的等价类。

- `Fintype`

```
class Fintype (α : Type) where
  elems : Finset α
  complete : ∀ x : α, x ∈ elems
```

要想说明一个类是`Fintype`需要列举出此类的所有项**列举**。

- `Finite`类型类：

```
class inductive Finite (α : Sort) : Prop
  | intro {n : ℕ} : α ≃ Fin n → Finite _
```

即一个类型是`Finite`的当且仅当它和某个`Fin n`之间存在双射。

- `Set.Finite`集合的有限性

```
def Finite (s : Set α) : Prop := Finite s
```

也就是说，一个集合是有限的当且仅当它作为一个类型是有限（`Finite`）的。

`Fintype`和`Finite`之间可以利用`Fintype.finite`和`Fintype.ofFinite`相互转化。需注意的是后者是不可计算的，也就是说一个类型是`Finite`的并不能保证我们一定知道它的全部元素是什么。

由于我们先验地知道Finset一定是有限的，它的基数Finset.card是一个自然数。而对于一般集合Set来说，我们有Set.encard返回 $\mathbb{N} \cup \infty$ 和Set.ncard返回自然数其中无限集合（不满足Set.Finite）的 ncard为 0。

习题 （超难）你会怎么定义有限和基数两个概念。用你的定义可以证明每个有限的类都有唯一一个基数吗？

8.2 算两次

我们在这里展示两个组合恒等式 $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$ 和 $\binom{m+n}{k} = \sum_{r=0}^{k} \binom{m}{k}\binom{n}{r-k}$。这里我们用到的主要定理是Fintype.card_congr

```
Fintype.card_congr {α β: Type u_4} (f : α ≃ β) :
  Fintype.card α = Fintype.card β
```

即任意两个有限（Fintype）类型之间的双射都保证了两类基数相等。

第一个组合恒等式

等式左边是 $\{0, \ldots, n\}$ 的 $k+1$-子集的数量；在 Lean 中我们说

```
Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1)))
```

那么同理，等式右边的 $\binom{n}{k}$ 和 $\binom{n}{k+1}$ 就分别是

```
Finset.powersetCard k (Finset.univ (α := Fin n))
```

和

```
Finset.powersetCard (k + 1) (Finset.univ (α := Fin n))
```

所以我们首先构造一个函数

```
Finset.powersetCard k (Finset.univ (α := Fin n)) ⊕
Finset.powersetCard (k + 1) (Finset.univ (α := Fin n)) →
Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1)))
```

这里⊕是用来表示不交并的。一旦我们能够证明这个函数是双射的，我们的定理也就结束了。所以在 Lean 中的框架可以写作

```
example (n k : ℕ) :
    (n + 1).choose (k + 1) = n.choose k + n.choose (k + 1) := by
  have lhs_eq : (n + 1).choose (k + 1) =
      (Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1)))).card :=
    by ...
```

```
  let e :
      Finset.powersetCard k (Finset.univ (α := Fin n)) ⊕
      Finset.powersetCard (k + 1) (Finset.univ (α := Fin n)) →
      Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1))) := ...
  have inj : Function.Injective e := ...
  have surj : Function.Surjective e := ...

  have bij : Function.Bijective e := ⟨inj, surj⟩
  have card_eq := Fintype.card_congr (Equiv.ofBijective e bij)
  simp only [...] at card_eq
  exact card_eq.symm
```

为了构建需要的函数，我们需要用到Sum.rec。这个定义是用来构建从不交并出发的函数：

```
Sum.rec {α β : Type u}{motive : α ⊕ β → Type} (inl : (val : α) → motive
    (Sum.inl val))
  (inr : (val : β) → motive (Sum.inr val)) (t : α ⊕ β) : motive t
```

所以我们需要两个函数一个Finset.powersetCard k (Finset.univ (α := Fin n)) → Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1)))另一个Finset.powersetCard (k + 1) (Finset.univ (α := Fin n)) → Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1)))。从数学上这两个函数分别是将一个 k-子集 $S \subseteq \{0,..,n-1\}$ 送到 $\{n\} \cup S$；另一个是将 $k+1$-子集 $S \subseteq \{0,..,n-1\}$ 送到 S。需注意的是因为Fin n和Fin (n + 1)的类型转换问题，我们用到了Fin.castSucc : Fin n → Fin (n + 1)。所以我们的e被构造为：

```
let e :
      Finset.powersetCard k (Finset.univ (α := Fin n)) ⊕
      Finset.powersetCard (k + 1) (Finset.univ (α := Fin n)) →
      Finset.powersetCard (k + 1) (Finset.univ (α := Fin (n + 1))) :=
    Sum.rec
      (fun s => ⟨Finset.cons (Fin.last _) (s.1.map ⟨Fin.castSucc,
   Fin.castSucc_injective n⟩) (by simp),
        by simpa [-Finset.coe_mem] using s.2⟩)
      (fun s => ⟨s.1.map ⟨Fin.castSucc, Fin.castSucc_injective n⟩,
        by simpa [-Finset.coe_mem] using s.2⟩)
```

至于单射和满射的性质都只需要分类讨论即可。

第二个恒等式

我们将要证明

$$\binom{m+n}{r} = \sum_{k=0}^{r+1} \binom{m}{k}\binom{n}{n-r}$$

等是左边是 $0, ..., m+n-1$ 的 r-子集的个数。如果我们将 ι 的 r 子集记作 S_r^n，那么我们有下列的双射

$$\begin{aligned}
& S_r^{\mathsf{Fin}(m+n)} \\
\cong & S_r^{\mathsf{Fin}m \oplus \mathsf{Fin}n} \\
\cong & \{(s,t) | s \subseteq \mathsf{Fin}m \wedge t \subseteq \mathsf{Fin}n \wedge |s| + |t| = r\} \\
\cong & \bigcup_{k=0}^{r} \{(s,t) | s \subseteq \mathsf{Fin}m \wedge t \subseteq \mathsf{Fin}n \wedge |s| = k \wedge |t| = r-k\} \\
\cong & \bigcup_{k=0}^{r} S_k^{\mathsf{Fin}m} \times S_{r-k}^{\mathsf{Fin}n}
\end{aligned}$$

而等式右边刚好是 $\bigcup_{k=0}^{r} S_k^{\mathsf{Fin}m} \times S_{r-k}^{\mathsf{Fin}n}$ 的基数。所以我们只需要在 Lean 里面写下这一串双射就可以了。我们用 Σ-类型来实现 $\bigcup_{k=0}^{r} S_k^{\mathsf{Fin}m} \times S_{r-k}^{\mathsf{Fin}n}$ — 在 Lean 中是`Σ k : Fin (r + 1), Finset.powersetCard k (Finset.univ (α := Fin m)) × Finset.powersetCard (r - k) (Finset.univ (α := Fin n))`

```
let e₁ :
      Finset.powersetCard r (Finset.univ (α := Fin (m + n))) ≃
      Finset.powersetCard r (Finset.univ (α := Fin m ⊕ Fin n)) :=
    { toFun s := ⟨s.1.map finSumFinEquiv.symm.toEmbedding, sorry⟩
      invFun s := ⟨s.1.map finSumFinEquiv.toEmbedding, sorry⟩
      left_inv s := by simp [Finset.map_map]
      right_inv s := by simp [Finset.map_map] }

  let e₂ : Finset.powersetCard r (Finset.univ (α := Fin m ⊕ Fin n)) ≃
    { p : Finset (Fin m) × Finset (Fin n) | p.1.card + p.2.card = r } :=
    { toFun s := ⟨⟨Finset.univ.filter fun i => Sum.inl i ∈ s.1,
        Finset.univ.filter fun i => Sum.inr i ∈ s.1⟩, sorry⟩
      invFun s := ⟨Finset.disjUnion
          (s.1.1.map ⟨Sum.inl, Sum.inl_injective⟩)
          (s.1.2.map ⟨Sum.inr, Sum.inr_injective⟩)
          (by rw [Finset.disjoint_iff_ne]; aesop), sorry⟩
      left_inv := sorry
      right_inv := sorry }

  let e₃ : { p : Finset (Fin m) × Finset (Fin n) | p.1.card + p.2.card = r } ≃
```

```
  Σ k : Fin (r + 1), { p : Finset (Fin m) × Finset (Fin n) | p.1.card = k ∧
 p.2.card = r - k } :=
  { toFun x := ⟨⟨x.1.1.card, sorry⟩, ⟨⟨x.1.1, x.1.2⟩, rfl, sorry⟩⟩
    invFun x := ⟨x.2.1, sorry⟩
    left_inv := sorry
    right_inv := sorry }

let e :
  (Σ k : Fin (r + 1), { p : Finset (Fin m) × Finset (Fin n) | p.1.card = k ∧
   p.2.card = r - k }) ≃
  Σ k : Fin (r + 1), Finset.powersetCard k (Finset.univ (α := Fin m)) ×
    Finset.powersetCard (r - k) (Finset.univ (α := Fin n)) :=
  { toFun x := ⟨x.1, ⟨⟨x.2.1.1, by simp [x.2.2.1]⟩, ⟨x.2.1.2, by simp
 [x.2.2.2]⟩⟩⟩
    invFun x := ⟨x.1, ⟨⟨x.2.1, x.2.2⟩, sorry, sorry⟩⟩
    left_inv := sorry
    right_inv := sorry }

let e := e₁.trans <| e₂.trans <| e₃.trans e
```

习题 利用构造双射的方法来计算下列恒等式：

- $\binom{n}{k} = \binom{n}{n-k}$
- $2^n = \sum_{k=0}^{n} \binom{n}{k}$
- $2 + 2 + 2 = 3 \times 2$

习题 $\{(x, y, z) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} | x + y + z = n\}$ 这个集合的基数是多少。

8.3 母函数（Generating function）

在 Lean 中，我们用形式幂级数来实现母函数。我们需要经常使用下列构造

- `PowerSeries.mk`：把 $a : \mathbb{N} \to R$ 的函数转为 $\sum a_i X^i \in R[[X]]$。
- `PowerSeries.eq_inv_iff_mul_eq_one`，如果 $q \in R[[X]]$ 的常数项不为 0 则 $p = q^{-1}$ 当且仅当 $pq = 1$.
- `PowerSeries.X`：$R[[X]]$ 中的未知数
- `PowerSeries.C`：$R \to R[[X]]$ 的映射

- `PowerSeries.coeff_C_mul`：如果 $c \in R$，$c \cdot q$ 的 n 次系数和 $c \cdot q_n$ 相等其中 q_n 是 q 的第 n 次系数。
- `PowerSeries.coeff_X_pow_mul'`:$X^n q$ 的 m 次系数和 q 的 $m-n$ 次系数相等。（若 $n > m$ 则为 0).

我们先看第一个例子 $-\frac{1}{1-X}$ 是 $-1, -1, -1, \ldots$ 的母函数:

```
1 lemma eq0 : -(1 - X : PowerSeries ℝ)⁻¹ = .mk fun _ => -1 := by
2   rw [neg_eq_iff_eq_neg, eq_comm, PowerSeries.eq_inv_iff_mul_eq_one (by simp),
     mul_comm]
3   ext n
4   ...
```

第 2 行，我们先把问题转化为 $(1-X)\cdot\sum -X^i = 1$。然后通过`ext n`将问题转化为求证所有系数相等。然后我们可以通过`PowerSeries.coeff_C_mul`和`PowerSeries.coeff_X_pow_mul'`等引理来完成命题。

同样地我们来证明 $\frac{1}{1-2X} = \sum 2^i X^i$

```
lemma eq1 : (1 - 2 * X : PowerSeries ℝ)⁻¹ = .mk fun n => 2 ^ n := by
  rw [eq_comm, PowerSeries.eq_inv_iff_mul_eq_one (by simp), mul_comm]
  ext n
  simp only [show (2 : PowerSeries ℝ) = C ℝ 2 from rfl, ...]
  ...
```

和上面不同的是我们这里为了方便使用`PowerSeries.coeff_C_mul`，将 $2 \in R[[X]]$ 写作了 $2 \in R$ 在`PowerSeries.C`的像。

有了这两个引理，我们就可以来证明 $f_0 = 0,\ f_1 = 3,\ f_{n+2} = 3f_{n+1} - 2f_n$ 的通项公式了。首先我们定义 f

```
def f : ℕ → ℤ
| 0 => 1
| 1 => 3
| n + 2 => 3 * f (n + 1) - 2 * f n
```

我们把 F 定义为 $\sum f_i X^i$

```
def F : PowerSeries ℝ := .mk fun n => f n
```

首先我们证明 $F = \frac{-1}{1-X} + \frac{2}{1-2X}$。

```
1 lemma F_eqn : F = - (1 - X)⁻¹ + 2 * (1 - 2 * X)⁻¹ := by
2   calc F
3     _ = (1 - 3 * X + 2 * X ^ 2)⁻¹ := ...
4     _ = - (1 - X)⁻¹ + 2 * (1 - 2 * X)⁻¹ := by
```

```
5       ...
6       have eq1 : (1 - X : PowerSeries ℝ)⁻¹ * (1 - 3 * X + 2 * X ^ 2) = 1 - 2 *
      X := ...
7       have eq2 : (1 - 2 * X : PowerSeries ℝ)⁻¹ * (1 - 3 * X + 2 * X ^ 2) = 1
      - X := ...
8       ...
```

第 3 行的等式和eq0和eq1的证明方式是一模一样的。第 4 行的等式中我们需要手动 “通分” 证明 $\frac{1-3X+2X^2}{1-X} = 1-2X$ 和 $\frac{1-3X+2X^2}{1-2X} = 1-X$。所以 f_n 就是 F 的第 n 次系数。根据这个引理和eq0和eq1, 我们知道 f_n 应该是 $2^{n+1}-1$:

```
lemma f_eq (n : ℕ) : f n = 2 ^ (n + 1) - 1 := by
  have := F_eqn
  have eq := congr($(F_eqn).coeff ℝ n)
  rw [map_add, eq1, eq0, coeff_mk, show (2 : PowerSeries ℝ) = C ℝ 2 by rfl,
    coeff_C_mul,
     coeff_mk] at eq
  simp only [F, coeff_mk] at eq
  rify
  rw [eq]
  ring
```

习题 定义 s_n 为 $\{(x,y,z) \in \mathbb{N}\times\mathbb{N}\times\mathbb{N} | x+y+z=n\}$ 的基数。通过找到 s_n 的母函数的方法，求 s 的通向公式。

习题 定义 s_n 为 $\{(x,y,z) \in \mathbb{N}\times\mathbb{N}\times\mathbb{N} | x+2y+2z=n\}$ 的基数。通过找到 s_n 的母函数的方法，求 s 的通向公式。

第三部分

分析

第九章　极限

在 Mathlib 中的极限概念使用滤子来实现的，在此我们不讨论其数学原理，而是直接通过实例来看 Lean 中如何实现极限的概念。滤子的数学原理可以参见 Bourbaki 的《General Topology》。本章中我们倾向于使用熟悉的 $\epsilon - \delta$ 概念，因为我们不直接使用滤子，所以有些证明并不是最简单的。

1. `Filter.atTop` 用来表达趋向于正无穷。
2. `Filter.atBot` 用来表达趋向于负无穷。
3. `nhds`或者$\mathcal{N}$ `x`用来表达 x 的邻域。
4. $\mathcal{N}$`[≠]x`用来表达 x 的去心邻域。
5. $\mathcal{N}$`[≥]x`用来表达 x 的邻域与 $[x, \infty)$ 的交集;$\mathcal{N}$`[>]x`用来表达 x 的邻域与 (x, ∞) 的交集。
6. $\mathcal{N}$`[≤]x`用来表达 x 的邻域与 $(-\infty, x]$ 的交集;$\mathcal{N}$`[<]x`用来表达 x 的邻域与 $(-\infty, x)$ 的交集.

其中 3-6 需要`open Filter Topology`

我们先来看一个最基础的例子 $\lim_{n\to\infty} \frac{1}{n} = 0$。

```
example : Tendsto (fun n : ℕ => (1 / n : ℝ)) atTop (𝓝 0) := by
  rw [tendsto_atTop_nhds]
```

`tendsto_atTop_nhds`的作用是将滤子语言的目标转化为更为熟悉的极限语言：对于任意包含 0 的开集 U，对于足够大的 n 我们有 $\frac{1}{n} \in U$。

```
⊢ ∀ (U : Set ℝ), 0 ∈ U → IsOpen U → ∃ N, ∀ (n : ℕ), N ≤ n → 1 / ↑n ∈ U
```

```
  intro U mem oU
  rw [Metric.isOpen_iff] at oU
```

而在度量空间中开集一般含有半径为 r 的球（`Metric.isOpen_iff`），其中 r 足够小。

```
U : Set ℝ
mem : 0 ∈ U
oU : ∀ x ∈ U, ∃ ε > 0, Metric.ball x ε ⊆ U
r : ℝ
r_pos : r > 0
ball_subset : Metric.ball 0 r ⊆ U
⊢ ∃ N, ∀ (n : ℕ), N ≤ n → 1 / ↑n ∈ U
```

现在这已经是我们熟悉的 $\epsilon - N$ 语言了，我们只需要找到合适的 N，因为自然数的 Archimedean 性质，存在 N 使得 $1 < N \cdot r$，我们使用 $\max\{1, N\}$。

```
1  obtain ⟨r, r_pos, ball_subset⟩ := oU _ mem
2  obtain ⟨N, hN⟩ := exists_lt_nsmul r_pos 1
3  use max N 1
4  ...
```

自测题 试证明:

1. $\lim_{n\to\infty} n = \infty$
2. $\lim_{n\to\infty} (-1)^n \cdot n \neq \infty$
3. $\lim_{n\to\infty} (-1)^n \cdot n \neq -\infty$

函数的极限也是同理，我们用 $\lim_{x\to 0^+} \frac{1}{x} = +\infty$ 举例子：

```
1 example : Tendsto (fun x : ℝ => 1 / x) (𝓝[>] 0) atTop := by
2   rw [tendsto_atTop]
3   intro R
```

我们先尽量简化滤子语言

```
1 R : ℝ
2 ⊢ ∀ᶠ (a : ℝ) in 𝓝[>] 0, R ≤ 1 / a
```

这里的∀ᶠ (a : ℝ) in 𝓝[>] 0 是表达终将（eventually）的概念，也就是说对于足够接近 0 并且在 $(0, \infty)$ 中的 a 来说，$R \leq \frac{1}{a}$。同样的道理∀ᶠ (a : ℝ) in atTop是表达对于任意足够大的实数 a。在这里我们不放假设 $0 < R$，因为如果 R 是负数，此题显然：

```
wlog hR : 0 < R
• simp only [not_lt] at hR
  apply eventually_of_mem (U := Set.Ioi 0)
  ...
rw [eventually_nhdsWithin_iff, eventually_nhds_iff]
```

```
  simp only [one_div]
```

这里`eventually_of_mem`是说如果因为所有 $(0,\infty)$ 中的 a 都满足 $R \le \frac{1}{a}$ 且 $(0,\infty) \in \mathcal{N}_{>0}$ 这个邻域，那么“所有在 $\mathcal{N}_{>0}$ 中的 a 都终将满足 $R \le \frac{1}{a}$”成立。而当 R 是正数时，我们需要证明

```
1 R : ℝ
2 hR : 0 < R
3 ⊢ ∃ t, (∀ y ∈ t, y ∈ Set.Ioi 0 → R ≤ y⁻¹) ∧ IsOpen t ∧ 0 ∈ t
```

也就是说存在某个包含 0 的开集 t 满足对于任意 $y \in t \cap (0,\infty)$ 都有 $R \le y^{-1}$。显然 $(-\frac{1}{R}, \frac{1}{R})$ 满足此条件：

```
1   use Set.Ioo (-1/R) (1 / R)
2   ...
```

自测题 试证明

1. $\lim_{x\to 0^-} \frac{1}{x} = -\infty$
2. $\lim_{x\to 0} \frac{1}{x^2} = \infty$

自测题 证明或证伪对于任意 $f : \mathbb{R} \to \mathbb{R}$，$\lim_{x\to 0} f(x)$ 存在当且仅当 $\lim_{x\to 0^+} f(x)$ 和 $\lim_{x\to 0^-} f(x)$ 存在且相等。

9.1 级数

在 Lean 中，我们有如下定义来表示无穷级数的收敛：

```
variable {α β : Type} [AddCommMonoid α][TopologicalSpace α]
def HasProd (f : β → α) (a : α) : Prop :=
  Tendsto (fun s ↦ ∏ b ∈ s, f b) atTop (𝓝 a)
```

注意这里的 s 的类型为`Finset β`，而 $s \to \infty$ 是说越来越大的有限集合，所以 $\lim_{s\to\infty} \sum_{b\in s} f(b)$ 表达了当 s 包含的项越来越多时级数的有限和是如何变化的。而`Summable f`就是`∃ a, HasSum f a`的简写。在 Lean 中`∑' b : β, f b`表示无穷级数的和；若此级数不收敛则为 0。

我们使用 $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$ 来举例。在 Lean 中，因为我们通常使用整个自然数作为脚标，所以我们考虑

$$\sum_{n\in\mathbb{N}} \frac{1}{(n+1)(n+2)}$$

首先我们注意到此级数确实收敛，因为 $\frac{1}{(n+1)(n+2)} \le \frac{1}{(n+1)^2}$，且后者收敛：

```
lemma s1 : Summable (fun n : ℕ => 1 / ((n + 1) * (n + 2) : ℝ)) := by
  fapply Summable.of_nonneg_of_le
  • exact fun n => 1 / (n + 1)^2
  • ...
  • ...
  • simpa using Real.summable_one_div_nat_add_rpow 1 2
```

这里Real.summable_one_div_nat_add_rpow是说对于任意实数 a, s 我们都有 $\sum_{n:\mathbb{N}} |n+a|^s$ 收敛当且仅当 $1 < s$。

为证明此级数和为 1, 我们证明 $\lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)(i+2)} = 1$:

```
1 example : ∑' n : ℕ, (1 / ((n + 1) * (n + 2)) : ℝ) = 1 := by
2   suffices : Tendsto (fun n => ∑ i ∈ Finset.range n, 1 / ((i + 1) * (i + 2)
      : ℝ)) atTop (𝓝 1)
3   • exact tendsto_nhds_unique s1.tendsto_sum_tsum_nat this
```

这是因为根据定义，当 $n \to \infty$, $\sum_{i=0}^{n-1} \frac{1}{(i+1)(i+2)}$ 趋向于∑' n : ℕ, (1 / ((n + 1) * (n + 2)) : ℝ)，所以只需要证明其极限为 1 即可。我们的第一步是

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)(i+2)} = \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)} - \frac{1}{(i+2)}$$

```
1   rw [tendsto_congr (f₂ := fun n =>  i ∈ Finset.range n, (1 / (i + 1) - 1 /
      (i + 2) : ℝ))
2     (fun n => Finset.sum_congr rfl fun i hi => by field_simp; norm_num)]
```

下一步是

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)} - \frac{1}{(i+2)} = \lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)} - \sum_{i=0}^{n-1} \frac{1}{(i+2)}$$

```
1   rw [tendsto_congr (f₂ := fun n =>
2       i ∈ Finset.range n, (1 / (i + 1) : ℝ) -
3       i ∈ Finset.range n, (1 / (i + 2) : ℝ)) (fun n => Finset.sum_sub_distrib
      _ _)]
```

下一步是

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} \frac{1}{(i+1)} - \sum_{i=0}^{n-1} \frac{1}{(i+2)} = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{(i+1)} - \sum_{i=0}^{n} \frac{1}{(i+2)}$$

```
1   rw [← tendsto_add_atTop_iff_nat 1]
```

这里tendsto_add_atTop_iff_nat是说对于任意数列 a_n 和 a_{n+k} 当 $n \to \infty$ 时有相同的极限。下一步是

$$\lim_{n\to\infty}\sum_{i=0}^{n}\frac{1}{(i+1)}-\sum_{i=0}^{n}\frac{1}{(i+2)}=\lim_{n\to\infty}\left(1+\sum_{i=0}^{n-1}\frac{1}{i+2}\right)-\sum_{i=0}^{n}\frac{1}{(i+2)}$$

```
1  rw [tendsto_congr (f₂ := fun n =>
2    1 +  i ∈ Finset.range n, (1 / (i + 2) : ℝ) -
3    ( i ∈ Finset.range (n + 1), (1 / (i + 2) : ℝ))) sorry]
```

下一步是

$$\lim_{n\to\infty}\left(1+\sum_{i=0}^{n-1}\frac{1}{i+2}\right)-\sum_{i=0}^{n}\frac{1}{(i+2)}=\lim_{n\to\infty}\left(1+\sum_{i=0}^{n-1}\frac{1}{i+2}\right)-\sum_{i=0}^{n-1}\frac{1}{(i+2)}+\frac{1}{n+2}$$

```
1  rw [tendsto_congr (f₂ := fun n =>
2    1 +  i ∈ Finset.range n, (1 / (i + 2) : ℝ) -
3    (( i ∈ Finset.range n, (1 / (i + 2) : ℝ)) + 1 / (n + 2))) (fun n => by
4    rw [Finset.sum_range_succ])]
```

下一步就是

$$\lim_{n\to\infty}\left(1+\sum_{i=0}^{n-1}\frac{1}{i+2}\right)-\sum_{i=0}^{n-1}\frac{1}{(i+2)}+\frac{1}{n+2}=\lim_{n\to\infty}1-\frac{1}{n+2}$$

```
1  rw [tendsto_congr (f₂ := fun n : ℕ => (1 - 1 / (n + 2 : ℕ) : ℝ)) (fun n =>
     by simp; ring)]
```

下面的就可以利用极限的知识解决了

自测题

1. 完成 $\sum_{n\in\mathbb{N}}\frac{1}{(n+1)(n+2)}=1$ 的证明。
2. 试证明几何级数的求和公式。

9.2 微分

在 Lean 中对于任意函数 $f:\mathbb{R}\to\mathbb{R}$，deriv f x代表了 f 在 x 处的导数，如果导数不存在，则默认为 0。大部分情况下需要证明 f 和 g 的可导性才能使用相应的定理：

1. deriv_add: 若 f,g 在 x 可导，则有 $(f+g)'(x)=f'(x)+g'(x)$
2. deriv_mul: 若 f,g 在 x 可导，则有 $(fg)'(x)=f'(x)g(x)+f(x)g'(x)$

3. `deriv_pow`: 若 f 在 x 可导，则有 $(f^n)'(x) = nf(x)^{n-1}f'(x)$

4. `deriv_rpow_const`: 若 f 在 x 可导，则有 $(f^n)'(x) = nf(x)^{n-1}f'(x)$ 其中 n 是不小于 1 的实数或 $f(x) \neq 0$。

5. `deriv_comp`: 若 f 在 x 可导，g 在 $f(x)$ 可导，则 $(g \circ f)'(x) = g'(f(x))f'(x)$。

在 Lean 中做微分还是比较容易的：我们用 $\sin^n x$ 和 $\sin(x^n)$ 来举例：

```
example (n : ℕ) (x : ℝ) :
    deriv (fun x : ℝ => sin x ^ n) x =  n * sin x ^ (n - 1) * cos x :=
  calc deriv (fun x : ℝ => sin x ^ n) x
    _ = deriv (sin ^ n) x := by rfl
    _ = n * sin x ^ (n - 1) * deriv sin x := by rw [deriv_pow]; exact
    differentiableAt_sin
    _ = n * sin x ^ (n - 1) * cos x := by rw [Real.deriv_sin]


example (n : ℕ) (x : ℝ) :
    deriv (fun x : ℝ => sin (x ^ n)) x = n * cos (x ^ n) * x ^ (n - 1) :=
  calc deriv (fun x : ℝ => sin (x ^ n)) x
    _ = deriv (sin ∘ fun x : ℝ => x ^ n) x := by rfl
    _ = deriv sin (x ^ n) * deriv (fun x : ℝ => x ^ n) x := by
      rw [deriv_comp]
      · exact differentiableAt_sin
      · exact differentiableAt_pow n
    _ = cos (x ^ n) * n * x ^ (n - 1) := by
      simp only [Real.deriv_sin, differentiableAt_fun_id, deriv_fun_pow,
    deriv_id'', mul_one]
      ring
    _ = _ := by ring
```

自测题 证明当 $x > 0$ 时 x^x 的导数是 $x^x + x^x \log x$。提示需要使用`Filter.EventuallyEq.deriv_eq`。

9.2.1 `HasDerivAt`相关的内容

读者如果一层一层地打开`deriv`的定义的话，会发现有下列两个有用的定义: 对于 $f : \mathbb{R} \to \mathbb{R}$ 和 $x, f' \in \mathbb{R}$ 和 $s \subseteq \mathbb{R}$

1. `HasDerivAt f f' x`是说 f 在 x 处的导数是 f'

2. `HasDerivWithinAt f f' s x`是说 f 在 x 处的导数是 f' 其中极限只能在集合 s 中取。这个概念可以用来实现单边极限等概念：如取 $s = (x, \infty)$。

和 `deriv`一样，Mathlib 中有非常丰富的相关引理包括但不限于

- `HasDerivAt.add`、`HasDerivAt.sub`、`HasDeriv.mul`、`HasDeriv.div`表示导数的加减乘除
- `HasDerivAt.comp`表示链式法则
- `Real.hasDerivAt_cos`，`Real.hasDerivAt_sin`等具体实例

我们这里的例子是 x^x 在 1 的导数: 我们先假设我们已经知道了 $e^{x \log x}$ 在 1 的导数是 1 (`claim1`).

```
example : HasDerivAt (fun x : ℝ => x ^ x) 1 1 := by
  have claim1 : HasDerivAt (fun x : ℝ => rexp (x * log x)) 1 1 := by
    sorry

  refine claim1.congr_of_eventuallyEq ?_
  refine eventually_of_mem (U := Set.Ioo 0 2) (Ioo_mem_nhds (by simp) (by
    simp)) ?_

  ...
```

这里我们用的`HasDerivAt.congr_of_eventuallyEq`是说如果 f 和 g 在某个邻域上相等，则他们的导数相等。这里我们用的邻域是 $(0, 2)$。这是因为在 0 处 x^x 和 $e^{x \log x}$ 都没有良好的定义所以我们避开它。现在我们来证明 $e^{x \log x}$ 的导数

```
example : HasDerivAt (fun x : ℝ => x ^ x) 1 1 := by
  have claim1 : HasDerivAt (fun x : ℝ => rexp (x * log x)) 1 1 := by
    have d0 : HasDerivAt log 1 1 := by simpa using Real.hasDerivAt_log (x :=
    1) (by simp)
    have d1 : HasDerivAt (fun x : ℝ => x) 1 1 := hasDerivAt_id' 1
    have d2 : HasDerivAt (fun x : ℝ => x * log x) 1 1 := by simpa using
    d1.mul d0

    simpa using Real.hasDerivAt_exp _ |>.comp 1 d2
  sorry
```

这里我们证明（`d0`）log 在 1 的导数是 1;`d1`证明 $x \mapsto x$ 在 1 的导数是 1；`d2`用导数的乘法结合`d0`、`d1`证明 $x \mapsto x \log x$ 的导数是 1。因为`hasDerivAt_exp`是用来求 e^x 的导数的，所以我们将它和`d2`通过链式法则`HasDerivAt.comp`可以得到我们想要的答案。

9.3 积分

9.3.1 定积分

在 Lean 中 $f : \mathbb{R} \to \mathbb{R}$ 从 a 到 b 的定积分被写作`∫ x in a..b,f x`。我们用最简单的例子来开始

```
example : ∫ x in (0)..1, x = 1 / 2 := by simp
```

这里的0需要括号是因为 0. 会造成浮点数歧义。在 Lean 中，定积分需要用到`Set.uIcc a b`代表着`Set.Icc a b`如果 $a \le b$ 不然的话则是`Set.Icc b a`

下面我们着重介绍微积分基本定理和其变体。

- 如果 f' 是 f 在 $[a,b]$ 区间上的导数，则 $\int_a^b f'(x)\mathrm{d}x = f(b) - f(a)$

```
intervalIntegral.integral_eq_sub_of_hasDerivAt
    {E : Type}
    [NormedAddCommGroup E] [NormedSpace ℝ E]
    {a b : ℝ}
    [CompleteSpace E]
    {f f' : ℝ → E}
    (hderiv : ∀ x ∈ Set.uIcc a b, HasDerivAt f (f' x) x)
    (hint : IntervalIntegrable f' MeasureTheory.volume a b) :
    ∫ (y : ℝ) in a..b, f' y = f b - f a
```

- 如果 f' 是 f 在 (a,b) 区间上的导数，且 f' 在 a 处的右极限是 f_a 而 f' 在 b 处的左极限是 f_b 则 $\int_a^b f'(x)\mathrm{d}x = f_b - f_a$

```
intervalIntegral.integral_eq_sub_of_hasDerivAt_of_tendsto
    {E : Type} [NormedAddCommGroup E] [NormedSpace ℝ E]
    {a b : ℝ} [CompleteSpace E]
    {f f' : ℝ → E}
    (hab : a < b)
    {fa fb : E}
    (hderiv : ∀ x ∈ Set.Ioo a b, HasDerivAt f (f' x) x)
    (hint : IntervalIntegrable f' MeasureTheory.volume a b)
    (ha : Filter.Tendsto f (nhdsWithin a (Set.Ioi a)) (nhds fa))
    (hb : Filter.Tendsto f (nhdsWithin b (Set.Iio b)) (nhds fb)) :
    ∫ (y : ℝ) in a..b, f' y = fb - fa
```

在 Mathlib 中，这样的变体还有很多，大家可以自行探索。

另一个重要的积分手段是换元法

- 如果 $f, f', g, g' : \mathbb{R} \to \mathbb{R}$ 符合以下性质

  1. f 在闭区间 $[a, b]$ 上连续
  2. f' 在闭区间 $[a, b]$ 上是 f 的导数
  3. f' 在闭区间 $[a, b]$ 上连续
  4. g' 是连续函数
  5. 对于任意 $x \in [a, b]$, g 在 $f(x)$ 处的导数是 $g'(f(x))$

  则 $\int_a^b f'(x)g'(f(x))\mathrm{d}x = g(f(b)) - g(f(a))$

```
theorem integral_deriv_comp_smul_deriv (hf : ∀ x ∈ uIcc a b, HasDerivAt
    f (f' x) x)
    (hg : ∀ x ∈ uIcc a b, HasDerivAt g (g' (f x)) (f x)) (hf' :
    ContinuousOn f' (uIcc a b))
    (hg' : Continuous g') : (∫ x in a..b, f' x • (g' ∘ f) x) = (g ∘ f) b
    - (g ∘ f) a :=
```

- 如果 $f, f', g, g' : \mathbb{R} \to \mathbb{R}$ 符合以下性质

  1. f 在闭区间 $[a, b]$ 上连续
  2. f' 在开区间 (a, b) 上是 f 的右导数
  3. f' 在闭区间 $[a, b]$ 上连续
  4. g 在闭区间 $[f(a), f(b)]$ 上连续
  5. 对于任意 $x \in (a, b)$,g 在 $f(x)$ 处的右导数是 $g'(f(x))$
  6. g' 在 $f([a, b])$ 上连续

  则 $\int_a^b f'(x)g'(f(x))\mathrm{d}x = g(f(b)) - g(f(a))$

```
theorem integral_deriv_comp_smul_deriv' (hf : ContinuousOn f [[a, b]])
    (hff' : ∀ x ∈ Ioo (min a b) (max a b), HasDerivWithinAt f (f' x)
    (Ioi x) x)
    (hf' : ContinuousOn f' [[a, b]]) (hg : ContinuousOn g [[f a, f b]])
    (hgg' : ∀ x ∈ Ioo (min (f a) (f b)) (max (f a) (f b)),
    HasDerivWithinAt g (g' x) (Ioi x) x)
    (hg' : ContinuousOn g' (f '' [[a, b]])) :
    (∫ x in a..b, f' x • (g' ∘ f) x) = (g ∘ f) b - (g ∘ f) a := by
```

```
    rw [integral_comp_smul_deriv'' hf hff' hf' hg',
      integral_eq_sub_of_hasDeriv_right hg hgg' (hg'.mono
      _).intervalIntegrable]
    exacts [rfl, intermediate_value_uIcc hf]
```

在 Mathlib 中，这样的变体还有很多，大家可以自行探索。

我们这里用的例子是

$$\int_1^n \frac{\log x}{x}\mathrm{d}x = \frac{\log^2 n}{2}$$

我们这里取 $f = \log, f' = x \mapsto \frac{1}{x}, g = x \mapsto \frac{x^2}{2}$, $g' := x \mapsto x$。

```
lemma integral_log_div (n : ℕ) (hn : 2 ≤ n) :
    ∫ x in (1)..n, log x / x = (log n) ^ 2 / 2 := by
  have eq := intervalIntegral.integral_deriv_comp_smul_deriv
    (g := fun x => x ^ 2 / 2) (g' := id)
    (f := log) (f' := fun x => 1 / x)
    (a := 1) (b := n)
    sorry
    sorry
    sorry
    sorry
  ...
```

接下来是分部积分: 如果 u, v, u', v' 是 $\mathbb{R} \to \mathbb{R}$ 的函数且满足以下性质：

- u, v 在闭区间 $[a, b]$ 上连续
- u', v' 在开区间 (a, b) 上分别是 u 和 v 的导数
- u', v' 在 $[a, b]$ 上可积则

$$\int_a^b u(x)v(x)\mathrm{d}x = u(b)v(b) - u(a)v(a) - \int_a^b u'(x)v(x)\mathrm{d}x$$

```
theorem integral_mul_deriv_eq_deriv_mul_of_hasDerivAt
    (hu : ContinuousOn u [[a, b]]) (hv : ContinuousOn v [[a, b]])
    (huu' : ∀ x ∈ Ioo (min a b) (max a b), HasDerivAt u (u' x) x)
    (hvv' : ∀ x ∈ Ioo (min a b) (max a b), HasDerivAt v (v' x) x)
    (hu' : IntervalIntegrable u' volume a b) (hv' : IntervalIntegrable v'
   volume a b) :
    ∫ x in a..b, u x * v' x = u b * v b - u a * v a - ∫ x in a..b, u' x * v x
    :=
```

```
integral_mul_deriv_eq_deriv_mul_of_hasDeriv_right hu hv
      (fun x hx ↦ (huu' x hx).hasDerivWithinAt) (fun x hx ↦ (hvv' x
  hx).hasDerivWithinAt) hu' hv'
```

这里我们证明欧拉-麦克劳林求和公式的一个特例: 设自然数 $n \geq 2$，$f : \mathbb{R} \to \mathbb{R}$ 在 $[1, n]$ 上连续且在 $(1, n)$ 上可导且其导数 f' 在 $[1, n]$ 上连续，则我们有

$$\sum_{k=1}^{n} f(k) = \frac{f(1)+f(n)}{2} + \int_1^n f(x)\mathrm{d}x + \int_1^n f'(x)\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$$

```
lemma EM (n : ℕ) (hn : 2 ≤ n) (f : ℝ → ℝ)
    (continuous_f : ContinuousOn f (Set.Icc 1 n))
    (diff_f : DifferentiableOn ℝ f (Set.Ioo 1 n))
    (continuous_deriv_f : ContinuousOn (deriv f) (Set.Icc 1 n)) :
     k ∈ Finset.Ico 1 (n + 1), f k =
    (f 1 + f n) / 2 + (∫ x in (1)..n, f x) + (∫ x in (1)..n, deriv f x * (x -
    x - 2⁻¹)) := by
```

需注意的是∫的优先级和括号的配合其证明如下：

1. 先证明
$$\sum_{k=1}^{n-1} \frac{f(k)+f(k+1)}{2} = \sum_{i=1}^{n} f(i) - \frac{f(1)+f(n)}{2}$$

2. 对于任意 $1 \leq k \leq n$，用分部积分法证明
$$\int_k^{k+1} \left(x - k - \frac{1}{2}\right) f'(x)\mathrm{d}x = \frac{f(k)+f(k+1)}{2} - \int_k^{k+1} f(x)\mathrm{d}x$$

3. 证明
$$\int_1^n \left(x - \lfloor x \rfloor - \frac{1}{2}\right) f'(x)\mathrm{d}x = \sum_{k=1}^{n-1} \int_k^{k+1} \left(x - k - \frac{1}{2}\right) f'(x)\mathrm{d}x$$

4. 利用 3, 证明
$$\int_1^n \left(x - \lfloor x \rfloor - \frac{1}{2}\right) f'(x)\mathrm{d}x = \sum_{k=1}^{n} f(k) - \frac{f(1)+f(n)}{2} - \int_1^n f(x)\mathrm{d}x$$

5. 通过代数变换命题得证

根据这个证明，我们 Lean 的骨架应该是

```
-- 第1步
lemma sum0 (n : ℕ) (hn : 2 ≤ n) (f : ℝ → ℝ) :
     k ∈ Finset.Ico 1 n, ((f k + f (k + 1)) / 2) =
    ( i ∈ Finset.Ico 1 (n + 1), f i) - ((f 1 + f n) / 2) := by sorry
```

```
lemma EM (n : ℕ) (hn : 2 ≤ n) (f : ℝ → ℝ)
    (continuous_f : ContinuousOn f (Set.Icc 1 n))
    (diff_f : DifferentiableOn ℝ f (Set.Ioo 1 n))
    (continuous_deriv_f : ContinuousOn (deriv f) (Set.Icc 1 n)) :
     k ∈ Finset.Ico 1 (n + 1), f k =
    (f 1 + f n) / 2 + (∫ x in (1)..n, f x) + (∫ x in (1)..n, deriv f x * (x -
    x - 2⁻¹)) := by
  -- 第2步
  have eq0 (k : ℕ) (hk : k ∈ Finset.Ico 1 n) :
      ∫ x in (k : ℝ)..(k + 1 : ℝ), (x - k - 2⁻¹) * deriv f x =
      (f k + f (k + 1)) / 2 - ∫ x in (k : ℝ)..(k + 1 : ℝ), f x := by sorry

  -- 第3步
  have eq1 : ∫ x in (1)..n, (x - x - 2⁻¹) * deriv f x =
       k ∈ Finset.Ico 1 n, ∫ x in (k : ℝ)..(k + 1 : ℝ), (x - k - 2⁻¹) *
   deriv f x := by sorry

  -- 第4步
  have eq2 := calc ∫ x in (1)..n, (x - x - 2⁻¹) * deriv f x
    _ = (( k ∈ Finset.Ico 1 (n + 1), f k) - (f 1 + f n) / 2)
      - ∫ x in (1)..n, f x := by sorry

  --第5步
  grind
```

其中第一步

```
  have eq0 (k : ℕ) (hk : k ∈ Finset.Ico 1 n) :
      ∫ x in (k : ℝ)..(k + 1 : ℝ), (x - k - 2⁻¹) * deriv f x =
      (f k + f (k + 1)) / 2 - ∫ x in (k : ℝ)..(k + 1 : ℝ), f x := by
    rw [intervalIntegral.integral_mul_deriv_eq_deriv_mul_of_hasDerivAt
      (u := fun x => x - k - 2⁻¹) (u' := 1)
      (v := f) (v' := deriv f) (a := k) (b := k + 1)
      sorry
      sorry
      sorry
      sorry
      sorry
      sorry]
    ...
```

这里我们将 $u := x \mapsto x - k - \frac{1}{2}, u' = 1, v = f, v' = f'$。

第二步和第三步中最重要的引理是

```
theorem sum_integral_adjacent_intervals_Ico {a : ℕ → ℝ} {m n : ℕ} (hmn : m ≤
    n)
    (hint : ∀ k ∈ Ico m n, IntervalIntegrable f μ (a k) (a <| k + 1)) :
      k ∈ Finset.Ico m n, ∫ x in a k..a <| k + 1, f x  μ = ∫ x in a m..a n, f
    x  μ := by
```

这个引理可以把相邻区间的积分叠加。

习题 设 $1 \ll R$, 求 $\int_1^R \frac{1}{x^2}\mathrm{d}x$。用Tendsto的语言求出此极限。

习题 设 $0 \ll R$, 求 $\int_0^R xe^{-x^2}\mathrm{d}x$。用Tendsto的语言求出此极限。

习题 设 $0 < \epsilon \ll 1$, 求 $\int_0^{3-\epsilon} \frac{1}{\sqrt{3-x}}\mathrm{d}x$。用Tendsto的语言求出此单边极限。

9.3.2 反常积分

我们首先定义

$$I_N := \int_1^N \frac{1-\log x}{x^2}\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$$

```
def I (N : ℕ) : ℝ := ∫ x in (1)..N, (1 - log x) / x^2 * (x -  x  - 2⁻¹)
```

我们的目标是证明 $I_N \to \int_0^\infty \frac{1-\log x}{x^2}\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$ 当 $N \to \infty$ 为此，我们先引入一个记号

```
local notation:max "Iω" => ∫ x in Set.Ioi 1, (1 - log x) / x^2 * (x -  x  -
    2⁻¹)
```

我们试图形式化

```
lemma I_converges : Tendsto I atTop (nhds Iω) := by sorry
```

这个看似是废话的命题背后其实需要的是证明 $x \mapsto \frac{1-\log x}{x^2}\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$ 在 $[1, \infty)$ 是可积的, 这里我们用到的是

```
intervalIntegral_tendsto_integral_Ioi
```

```
lemma I_converges :
    Tendsto I atTop (nhds Iω) := by
  refine intervalIntegral_tendsto_integral_Ioi
      (f := fun x => (1 - log x) / x^2 * (x -  x  - 2⁻¹)) (μ := volume)
```

```
      (a := 1) (b := fun n : ℕ => n)
      (l := atTop) ?_ tendsto_natCast_atTop_atTop
  sorry
```

又因为我们知道在 $[1,\infty)$ 上

$$\begin{aligned}&\left|\frac{1-\log x}{x^2}\left(x-\lfloor x\rfloor-\frac{1}{2}\right)\right|\\ \le&\frac{1+\log x}{x^2}\left|x-\lfloor x\rfloor-\frac{1}{2}\right|\\ \le&\frac{1+\log x}{x^2}\frac{1}{2}\\ =&\frac{1}{2}\frac{1+\log x}{x^2}\\ &\dagger\end{aligned}\tag{9.1}$$

我们只需要证明后者可积即可

```
lemma I_converges :
    Tendsto I atTop (nhds Iω) := by
  refine intervalIntegral_tendsto_integral_Ioi
      (f := fun x => (1 - log x) / x^2 * (x - ⌊x⌋ - 2⁻¹)) (μ := volume)
      (a := 1) (b := fun n : ℕ => n)
      (l := atTop) ?_ tendsto_natCast_atTop_atTop
  fapply Integrable.mono (g := fun x => 2⁻¹ * (log x + 1) / x ^ 2)
  <;> sorry
```

为此我们分别证明 $x\mapsto\frac{\log x}{x^2}$ 和 $\frac{1}{x^2}$ 在 $[1,\infty)$ 上可积

```
lemma log_div_integrableOn : IntegrableOn (fun x ↦ log x / x ^ 2) (Set.Ioi 1)
    := by sorry

lemma one_div_sq_integrableOn : IntegrableOn (fun x : ℝ ↦ 1 / x ^ 2)
    (Set.Ioi 1) := by sorry

lemma I_converges :
    Tendsto I atTop (nhds Iω) := by
  refine intervalIntegral_tendsto_integral_Ioi
      (f := fun x => (1 - log x) / x^2 * (x - ⌊x⌋ - 2⁻¹)) (μ := volume)
      (a := 1) (b := fun n : ℕ => n)
      (l := atTop) ?_ tendsto_natCast_atTop_atTop
  fapply Integrable.mono (g := fun x => 2⁻¹ * (log x + 1) / x ^ 2)
  • simp_rw [mul_div_assoc, add_div]
    refine .const_mul (.add log_div_integrableOn one_div_sq_integrableOn) _
```

- sorry --- 可测性质
- sorry --- † 几乎处处成立

这里的可测性质来自连续性需要使用

```
ContinuousOn.aestronglyMeasurable
```

鉴于我们前面已经看过很多ContinuousOn的相关应用，我们在此略过。我们的重头戏是两个可积性，这两个例子都可以使用

```
#check integrableOn_Ioi_deriv_of_nonneg

MeasureTheory.integrableOn_Ioi_deriv_of_nonneg {g g' : ℝ → ℝ} {a l : ℝ}
    (hcont : ContinuousWithinAt g (Set.Ici a) a)
  (hderiv : ∀ x ∈ Set.Ioi a, HasDerivAt g (g' x) x) (g'pos : ∀ x ∈ Set.Ioi a,
    0 ≤ g' x) (hg : Tendsto g atTop (nhds l)) :
  IntegrableOn g' (Set.Ioi a) volume
```

即如果一个函数 g 在 $[a,\infty)$ 上连续且在无穷处有极限，若其在 $[a,\infty)$ 上有非负导数 g'，则 g' 可导. 有了这个引理, 则两个可积性就变成了两个极限

$$\lim_{x\to\infty} -\frac{\log x+1}{x} \to 0$$

和

$$\lim_{x\to\infty} -\frac{1}{x} \to 0$$

```
lemma log_div_integrableOn : IntegrableOn (fun x ↦ log x / x ^ 2) (Set.Ioi 1)
    := by
  fapply integrableOn_Ioi_deriv_of_nonneg
  • exact fun x => - (log x + 1) / x
  • exact 0
  • sorry --连续性
  • sorry --导数
  • sorry --非负性质
  • rw [show nhds (0 : ℝ) = nhds (-(0 + 0)) by simp]
    simp_rw [neg_div, add_div]
    refine .neg <| .add ?_ ?_
    • simpa using Real.tendsto_pow_log_div_mul_add_atTop 1 0 1 (by simp)
    • rw [show HDiv.hDiv (1 : ℝ) = fun x => x⁻¹ by ext; simp]
      exact tendsto_inv_atTop_zero

lemma one_div_sq_integrableOn : IntegrableOn (fun x : ℝ ↦ 1 / x ^ 2)
    (Set.Ioi 1) := by
```

```
  fapply integrableOn_Ioi_deriv_of_nonneg
  • exact fun x => - 1 / x
  • exact 0
  • sorry --连续性
  • sorry --导数
  • sorry --非负性质
  • rw [show nhds (0 : ℝ) = nhds (-0) by simp]
    simp_rw [neg_div]
    refine .neg ?_
    rw [show HDiv.hDiv (1 : ℝ) = fun x => x⁻¹ by ext; simp]
    exact tendsto_inv_atTop_zero
```

根据上节课的欧拉麦克劳林公式我们可以定义

$$E_N = \sum_{k=1}^{N} \frac{\log k}{k} - \frac{\log^2 N}{N}$$

```
def E (N : ℕ) : ℝ := ( k ∈ Finset.Ico 1 (N + 1), log k / k) - (log N) ^ 2 / 2
```

并立刻获得当 $2 \leq n$

$$E_n = \frac{\log n}{2n} + I_n$$

那么显然我们可以获得

$$\lim_{n\to\infty} E_n \to \int_0^{\infty} \frac{1-\log x}{x^2}\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$$

即

```
def E (N : ℕ) : ℝ := (∑ k ∈ Finset.Ico 1 (N + 1), log k / k) - (log N) ^ 2 /
    2

lemma E_eq (n : ℕ) (hn : 2 ≤ n) :
    E n = log n / (2 * n) + I n := by
  delta E I
  rw [EM_special n hn, show ∀ a b c : ℝ, (a + b + c) - b = a + c by intros;
    ring]

lemma E_eq' (n : ℕ) :
    E (n + 2) = log (n + 2) / (2 * (n + 2)) + I (n + 2) := by
  have hn : 2 ≤ n + 2 := by omega
  rw [E_eq (n + 2) hn]
  simp
```

```
lemma E_converges : Tendsto E atTop (nhds Iω) := by
  ...
  have lim0 := Real.tendsto_pow_log_div_mul_add_atTop 1 0 1 (by simp)
  ...
```

这里我们用到的关键引理是

```
#check Real.tendsto_pow_log_div_mul_add_atTop

Real.tendsto_pow_log_div_mul_add_atTop (a b : ℝ) (n : ℕ) (ha : a ≠ 0) :
  Tendsto (fun x ↦ log x ^ n / (a * x + b)) atTop (nhds 0)
```

即当 $a \neq 0$

$$\lim_{x\to\infty} \frac{\log x^n}{ax+b} \to 0$$

习题 （其实和积分无关）定义 $S_N = \sum_{n=2}^{N}(-1)^n \frac{\log n}{n}$ 通过证明当 $2 \le n$ 时

$$S_{2n} = \sum_{j=1}^{n} \frac{1}{j} - \log n + (E_n - E_{2n}) - \frac{1}{2}\log^2 2$$

得出 $\lim_{n\to\infty} S_n = \gamma \log 2 - \frac{1}{2}\log^2 2$ 其中 γ 为欧拉麦克劳林常数。

9.3.3 控制收敛定理

这里我们定义

$$T_N = \sum_{m=1}^{N}\sum_{n=1}^{N} \frac{1}{mn(m+n)}$$

为

```
def T (N : ℕ) : ℝ :=
  ∑ ⟨m, n⟩ ∈ (Finset.Ico 1 (N + 1) ×ˢ Finset.Ico 1 (N + 1)),
    1 / (m * n * (m + n))
```

我们本节将通过积分的方式证明 $\lim_{n\to\infty} T_n = 2\zeta(3)$。因为本证明的长度，我们将略过一些形式化:

- $T_N = \int_0^1 \frac{1}{x}\left(\sum_{m=1}^{N} \frac{x^m}{m}\right)\left(\sum_{n=1}^{N} \frac{x^n}{n}\right)$。这里只是在交换有限的和，所以没有问题

  ```
  lemma T_eq (N : ℕ) :
      T N =
      ∫ x in (0)..1, x⁻¹ *
          (∑ m ∈ Finset.Ico 1 (N + 1), x^m / m) *
          (∑ n ∈ Finset.Ico 1 (N + 1), x^n / n) := by sorry
  ```

```
lemma T_eq' :
    T =
    fun N : ℕ => ∫ x in (0)..1, x⁻¹ *
        (∑ m ∈ Finset.Ico 1 (N + 1), x^m / m) *
        (∑ n ∈ Finset.Ico 1 (N + 1), x^n / n) :=
  funext T_eq
```

- 任意区间 $[a,b]$（$0<a\le b<1$），都有 $x\mapsto \frac{\log^2(1-x)}{x}$ 可积，因为连续性

```
lemma intervalIntegrable_fact (a b : ℝ) (ha : 0 < a) (hb : b < 1) (hab :
    a ≤ b) :
    IntervalIntegrable (fun x => log (1 - x) ^ 2 / x) volume a b := by
    sorry
```

- 对于任意自然数 n, 我们都有

$$\int_{\frac{1}{n+2}}^{1-\frac{1}{n+2}} \frac{\log^2(1-x)}{x} \le \frac{1}{2}+\log^2 2+2\log 2+2$$

这是因为我们把积分拆成 $\int_{\frac{1}{n+2}}^{\frac{1}{2}}$ 和 $\int_{\frac{1}{2}}^{1-\frac{1}{n+2}}$ 并分别估计。（这里具体数值不重要，只需要有界性质）

```
lemma bddAbove_integral_fact (n : ℕ)  :
    ∫ x in (1 / (n + 2) : ℝ)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2)
    / x ≤
    2⁻¹ + (log 2 ^ 2 + 2 * log 2 + 2) := by
  calc ∫ x in (1 / (n + 2) : ℝ)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^
    2) / x
    _ = (∫ x in (1 / (n + 2) : ℝ)..(1 / 2), (log (1 - x) ^ 2) / x)
      + ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2) / x :=
    sorry
    _ ≤ (∫ x in (1 / (n + 2) : ℝ)..(1 / 2), 4 * x)
      + ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2) / x :=
    sorry
    _ ≤ (∫ x in (0)..(1 / 2), 4 * x)
      + ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2) / x :=
    sorry
    _ = 2⁻¹ + ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2) /
    x := sorry
```

```
    _ ≤ 2⁻¹ + ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^ 2) *
     2 := sorry
    _ = 2⁻¹ + 2 * ∫ x in (1 / 2)..(1 - 1 / (n + 2) : ℝ), (log (1 - x) ^
    2) := sorry
    _ = 2⁻¹ + 2 * (-∫ x in (1 / 2)..(1 / (n + 2) : ℝ), log x ^ 2) :=
    sorry
    _ = 2⁻¹ + 2 * ∫ x in (1 / (n + 2) : ℝ)..(1 / 2), log x ^ 2 := sorry
    _ = 2⁻¹
      + 2 *
        (2⁻¹ * (log 2 ^ 2 + 2 * log 2 + 2)
          - (n + 2 : ℝ)⁻¹ * (log (n + 2) ^ 2 + 2 * log (n + 2) + 2)) :=
    sorry
    _ = 2⁻¹
      + ((log 2 ^ 2 + 2 * log 2 + 2)
          - 2 * (log (n + 2) ^ 2 / (n + 2) + 2 * log (n + 2) / (n + 2) +
    2 / (n + 2))) := sorry
    _ ≤ 2⁻¹ + (log 2 ^ 2 + 2 * log 2 + 2) := sorry
```

这里需要用到换元、分部积分等已经讲过的技巧

- $\int_{\frac{1}{n+2}}^{1-\frac{1}{n+2}} \frac{\log^2(1-x)}{x} \le \frac{1}{2} + \log^2 2 + 2\log 2 + 2$ 当 $n \to \infty$ 时有极限

```
lemma limit_integral_fact :
    ∃ L : ℝ, Tendsto (fun n : ℕ => ∫ x in (1 / (n + 2) : ℝ)..(1 - 1 /
    (n + 2) : ℝ),
      (log (1 - x) ^ 2) / x) atTop (nhds L) := by
  fapply Real.tendsto_of_bddAbove_monotone <;> sorry
```

这里用到单调收敛定理

- $\int_{\frac{1}{n+2}}^{n+2} \frac{u^2 e^{-u}}{1-e^{-u}} = \int_{1-e^{-\frac{1}{n+2}}}^{1-e^{-(n+2)}} \frac{\log^2(1-x)}{x}$

```
lemma integral_calc_aux0 (n : ℕ) :
    ∫ u in ((n + 2)⁻¹)..(n + 2), (u ^ 2 * rexp (-u)) / (1 - rexp (-u)) =
    ∫ x in (1 - rexp (- (n + 2 : ℝ)⁻¹))..(1 - rexp (- (n + 2 : ℝ))),
    (log (1 - x) ^ 2) / x := sorry
```

这里我们利用换元法

我们的第一个重头戏来了

```
lemma integrable_fact :
    IntervalIntegrable (fun x => log (1 - x) ^ 2 / x) volume (0 : ℝ) 1 := by
```

即 $x \mapsto \frac{\log^2(1-x)}{x}$ 在 $(0,1)$ 上可积。这里我们用到的关键引理是

```
#check AECover.integrable_of_integral_tendsto_of_nonneg_ae

MeasureTheory.AECover.integrable_of_integral_tendsto_of_nonneg_ae.{u_1, u_2}
    {α : Type u_1} {ι : Type u_2}
  [MeasurableSpace α] {μ : Measure α} {l : Filter ι} [l.NeBot]
    [l.IsCountablyGenerated] {φ : ι → Set α}
  (hφ : AECover μ l φ) {f : α → ℝ} (I : ℝ) (hfi : ∀ (i : ι), IntegrableOn f
    (φ i) μ) (hnng : ∀ᵐ (x : α) ∂μ, 0 ≤ f x)
  (htendsto : Tendsto (fun i ↦ ∫ (x : α) in φ i, f x ∂μ) l (nhds I)) :
    Integrable f μ
```

即给定

- 几乎处处非负函数 f
- 集合 ϕ_i 当 i 趋近于无穷时，ϕ_i 几乎处处覆盖 $(0,1)$
- f 在任意 ϕ_i 上可积
- $\int_{x\in\phi_i} f$ 有极限

则 f 可积。这里我们的 ϕ 自然是 $\phi_n = [\frac{1}{n+2}, 1-\frac{1}{n+2}]$

```
lemma integrable_fact :
    IntervalIntegrable (fun x => log (1 - x) ^ 2 / x) volume (0 : ℝ) 1 := by
  obtain ⟨L, L_spec⟩ := limit_integral_fact
  rw [intervalIntegrable_iff_integrableOn_Ioo_of_le (by simp)]
  change Integrable _ _
  fapply AECover.integrable_of_integral_tendsto_of_nonneg_ae
    (l := atTop (α := ℕ)) (φ := fun n => Set.Icc (1 / (n + 2 : ℝ)) (1 - 1 /
    (n + 2 : ℝ)))
    (I := L)
  <;> sorry
```

那么 T 的极限就是

$$\int_0^1 \frac{\log^2(1-x)}{x}$$

这里我们肯定要用到某种控制收敛定理：

```
#check intervalIntegral.tendsto_integral_filter_of_dominated_convergence
```

这个定理说的是假如有

- 区间 $[a,b]$

- 实函数 f 和 M 且 M 可积
- 实函数序列 F_n 且 F_n 几乎处处可积
- 当 $n \to \infty$ 时, 几乎处处 $x \in [a,b]$, 有 $|F_n(x)| \le M(x)$
- 几乎出处 $x \in [a,b]$, 都有 $F_n(x) \to f(x)$

则 $\int_a^b F_n(x) \to \int_a^b f(x)$ 这里我们只需要 M 取 $\frac{\log^2(1-x)}{x}$。我们需要用到等式

$$T_N = \int_0^1 \frac{1}{x} \left(\sum_{m=1}^{N} \frac{x^m}{m} \right) \left(\sum_{n=1}^{N} \frac{x^n}{n} \right)$$

和 $\log(1-x)$ 的泰勒展开。

```
lemma T_lim0 :
    Tendsto T atTop (nhds <| ∫ x in (0)..1, (log (1 - x) ^ 2) / x) := by
  rw [T_eq']
  fapply intervalIntegral.tendsto_integral_filter_of_dominated_convergence
  · exact fun x => log (1 - x) ^ 2 / x
  · refine .of_forall fun _ => by fun_prop
  · refine .of_forall fun n => eventually_of_mem (U := {1}ᶜ) (by simp
    [mem_ae_iff]) ?_
    ...
    intro x hx0 hx1 hx2
    rw [show log (1 - x) ^ 2 / x = |x⁻¹| * -log (1 - x) * -log (1 - x) by ...]
    gcongr
    · ...
    all_goals
    · rw [abs_of_nonneg (by positivity)]
      have lim0 := Complex.hasSum_taylorSeries_neg_log (z := x)
        (by simpa [abs_lt] using ⟨by linarith, lt_of_le_of_ne ‹_› ‹_›⟩)
      ...
  · exact integrable_fact
  · ...
```

同理我们可以通过AECover.integrable_of_integral_tendsto_of_nonneg_ae得到 $u \mapsto \frac{u^2 e^{-u}}{1-e^{-u}}$ 在 $[0,\infty)$ 上可积。根据另一个控制收敛定理

```
#check AECover.integral_tendsto_of_countably_generated
```

已知

- 实函数 f

- 集合 ϕ_i 当 $i \to \infty$,ϕ_i 几乎处处覆盖整个空间
- f 可积

则 $\lim_i \int_{x\in\phi_i} f(x) = \int f(x)$。我们立刻可以得到推论

$$\lim_{n\to\infty} \int_{\frac{1}{n+2}}^{1-\frac{1}{n+2}} \frac{\log^2(1-x)}{x} \to \int_0^1 \frac{\log^2(1-x)}{x}$$

且

$$\lim_{n\to\infty} \int_{1-e^{-\frac{1}{n+2}}}^{1-e^{-(n+2)}} \frac{u^2e^{-u}}{1-e^{-u}} \to \int_0^\infty \frac{u^2e^{-u}}{1-e^{-u}}$$

```
lemma integral_calc0_aux1 :
    Tendsto (fun n : ℕ => ∫ u in ((n + 2)⁻¹)..(n + 2), (u ^ 2 * rexp (-u)) /
    (1 - rexp (-u)))
      atTop (nhds <| ∫ u in Set.Ioi 0, (u ^ 2 * rexp (-u)) / (1 - rexp (-u)))
    := by
  have :=
    AECover.integral_tendsto_of_countably_generated
      (l := atTop (α := ℕ))
      (φ := fun n => Set.Icc ((n + 2 : ℝ)⁻¹) (n + 2))
      (μ := volume.restrict (Set.Ioi 0))
      (f := fun u => (u ^ 2 * rexp (-u)) / (1 - rexp (-u)))
      ...
  sorry

lemma integral_calc0_aux2 :
    Tendsto
      (fun n : ℕ => ∫ x in (1 - rexp (- (n + 2 : ℝ)⁻¹))..(1 - rexp (- (n + 2
    : ℝ))),
        (log (1 - x) ^ 2) / x)
      atTop (nhds <| ∫ x in (0)..1, (log (1 - x) ^ 2) / x) := by
  have :=
    AECover.integral_tendsto_of_countably_generated
      (l := atTop (α := ℕ))
      (φ := fun n => Set.Icc (1 - rexp (- (n + 2 : ℝ)⁻¹)) (1 - rexp (- (n + 2
    : ℝ))))
      (μ := volume.restrict (Set.Ioo 0 1))
      (f := fun x => (log (1 - x) ^ 2) / x)
      ...

  sorry
```

所以我们得到结论

$$\int_0^1 \frac{\log^2(1-x)}{x} = \int_0^\infty \frac{u^2 e^{-u}}{1-e^{-u}}$$

```
lemma integral_calc0 :
    ∫ u in Set.Ioi 0, (u ^ 2 * rexp (-u)) / (1 - rexp (-u)) =
    ∫ x in (0)..1, (log (1 - x) ^ 2) / x := by
  have lim0 := integral_calc0_aux1
  have lim1 := integral_calc0_aux2
  simp_rw [integral_calc_aux0] at lim0
  exact tendsto_nhds_unique lim0 lim1
```

接下来我们通过将 $\frac{1}{1-e^{-u}}$ 看作几何级数可以得到

$$\int_0^\infty \frac{u^2 e^{-u}}{1-e^{-u}} = \int_0^\infty \sum_n u^2 e^{-(n+1)u}$$

这里不涉及控制收敛，因为被积分的部分相等

习题 利用`integral_tsum_of_summable_integral_norm`交换积分和求和符号证明

$$\int_0^\infty \frac{u^2 e^{-u}}{1-e^{-u}} = \sum_n \int_0^\infty u^2 e^{-(n+1)u}$$

并完成 $T_N \to \sum_n \frac{2}{(n+1)^3}$ 的证明

习题 补全略过的证明

第十章　拓扑

10.1　基础概念

如果 (X,τ) 是一个拓扑空间，我们是说 $\tau \subseteq \mathcal{P}(X)$ 且满足:

1. $X \in \tau$
2. $S,T \in \tau$ 意味着 $S \cap T \in \tau$;
3. $\forall i, S_i \in \tau$ 意味着 $\bigcup_i S_i \in \tau$。

如此一来 τ 中的元素被称为开集，τ 被称作 X 上的拓扑。在 Mathlib 中，如果 X 是任意一个类型，则 X 上的拓扑

```
class TopologicalSpace (X : Type u) where
  protected IsOpen : Set X → Prop
  protected isOpen_univ : IsOpen univ
  protected isOpen_inter : ∀ s t, IsOpen s → IsOpen t → IsOpen (s ∩ t)
  protected isOpen_sUnion : ∀ s, (∀ t ∈ s, IsOpen t) → IsOpen (⋃₀ s)
```

数学中的关于拓扑的基础概念在 Mathlib 中已经比较完备，其中开集（`IsOpen`）、闭集（`IsClosed`）、稠密（`Dense`）、闭包（`closure`）等自然不必说。值得一提的是关于紧性 (`IsCompact`) 的定义是用滤子的语言来完成的：

```
def IsCompact (s : Set X) :=
  ∀ {f} [NeBot f], f ≤ 𝒫 s → ∃ x ∈ s, ClusterPt x f
```

而我们孰知的用有限覆盖的定义则需要使用

```
#check isCompact_iff_finite_subcover

/-
isCompact_iff_finite_subcover {X : Type u} [TopologicalSpace X] {s : Set X} :
  IsCompact s ↔
  ∀ {ι : Type u} (U : ι → Set X), (∀ (i : ι), IsOpen (U i)) → s ⊆ ⋃ i, U i → ∃
    t, s ⊆ ⋃ i ∈ t, U i
-/
```

这里需要注意的是指标集 ι 的宇宙层级需要和拓扑空间 X 相同。

度量空间上的拓扑 我们最常见的拓扑之一便是度量空间上由球来定义的拓扑，在 Lean 中表现为

```
variable {X : Type*} [MetricSpace X]

#synth TopologicalSpace X

#check Metric.isOpen_iff
/-
Metric.isOpen_iff.{u} {α : Type u} [PseudoMetricSpace α] {s : Set α} : IsOpen
    s ↔ ∀ x ∈ s, ∃ ε > 0, Metric.ball x ε ⊆ s
-/
```

子空间上的拓扑 如果 X 本身是一个拓扑空间，则 X 的任意子集都是一个拓扑空间

```
variable {X : Type*} [TopologicalSpace X] (S : Set X)

#synth TopologicalSpace S
```

这是因为 Mathlib 中定义了拓扑的引导（induced topology）：如果 $f : X \to Y$ 是一个函数，其中 Y 上有拓扑，则 f 引导出来的拓扑中的开集 S 是说 S 是某个 Y 中的开集的原像

```
#check isOpen_induced_iff

/-
isOpen_induced_iff {α β} [TopologicalSpace β] {s : Set α} {f : α → β} :
  IsOpen s ↔ ∃ t, IsOpen t ∧ f ⁻¹' t = s
-/
```

在子空间的拓扑中,f 是自然映射。

相对于引导拓扑来说，还有余引导拓扑 (coinduced topology): 如果 $: X \to Y$ 是从拓扑空间 X 到 Y 的函数，则余引导拓扑下的 Y 的开集是原像是开集的集合

```
isOpen_coinduced {α β} {TopologicalSpace α} {s : Set β} {f : α → β} :
  IsOpen s ↔ IsOpen (f ⁻¹' s)
```

如果 $\mathfrak{T}$ 是所有 X 上的拓扑的集合，则 $\mathfrak{T}$ 本身形成了一个完整的 lattice。这里的大小比较关系是 $\tau_1 \leq \tau_2$ 当且仅当 τ_2 中的开集在 τ_1 中也是开集。既然拓扑是个 lattice，它就有最大和最小

```
-- discrete topology
#check ( : TopologicalSpace X)

-- indiscrete topology
#check ( : TopologicalSpace X)
```

其中最小值是离散拓扑（所有集合都是开集）和最大值是 indiscrete topology 即只有空集和整个空间是开集。

卡式积上的拓扑 如果 X,Y 是拓扑空间，$X\times Y$ 的拓扑是 $X\times Y\to X$ 的余引导空间和 $X\times Y\to Y$ 的余引导空间的 infinmum。用平常话说 $S\subseteq X\times Y$ 是开集当且仅当任意 $(a,b)\in X\times Y$ 都存在开集 $a\in A\subseteq X, b\in B\subseteq Y$ 且 $A\times B\subseteq S$。

```
instance instTopologicalSpaceProd [t₁ : TopologicalSpace X] [t₂ :
    TopologicalSpace Y] :
    TopologicalSpace (X × Y) :=
  induced Prod.fst t₁  induced Prod.snd t₂

#check isOpen_prod_iff
/-
isOpen_prod_iff {X Y} [TopologicalSpace X] [TopologicalSpace Y] {s : Set (X ×
    Y)} :
  IsOpen s ↔ ∀ (a : X) (b : Y), (a, b) ∈ s → ∃ u v, IsOpen u ∧ IsOpen v ∧ a ∈
    u ∧ b ∈ v ∧ u ×ˢ v ⊆ s
-/
```

商空间 商空间是 $X\to X/_\sim$ 的余引导拓扑。

```
instance instTopologicalSpaceQuotient {s : Setoid X} [t : TopologicalSpace X] :
    TopologicalSpace (Quotient s) :=
  coinduced Quotient.mk' t
```

习题 「难」怎么把拓扑空间粘起来？

10.2 一个用拓扑试的证明无限素数的方法

首先我们定义一个在 $\mathbb{Z}$ 上的拓扑，我们规定 $U\subseteq\mathbb{Z}$ 是开集当且仅当对于任意 $x\in U$ 都存在 $1\le m\in\mathbb{Z}$ 使得 $m\mathbb{Z}+x\subseteq U$。在这个奇怪的拓扑上，我们发现任意非空开集都是无限的，且任意等差数列 $\{mx+b|1\le m,x\in\mathbb{Z}\}$ 都又是开集也是闭集。那么假如自

然数只有有限的素数我们就会发现 $\bigcup_{p\text{ 素数}} p\mathbb{Z}$ 是个闭集（有限闭集的集合），所以它的补集是开集，但它的补集正是 $\{-1,1\}$。这是个矛盾，因为非空开集一定无限。

首先我们定义开集，也就是包含等差数列这一概念，并证明这些集合构成拓扑

```
def ContainsArithProgression (U : Set ℤ) : Prop :=
∀ (x : ℤ), x ∈ U → ∃ (m : ℤ), 1 ≤ m ∧ ∀ (k : ℤ), m * k + x ∈ U

instance weird_top_on_int : TopologicalSpace ℤ where
  IsOpen := ContainsArithProgression
  isOpen_univ := sorry
  isOpen_inter := sorry
  isOpen_sUnion := sorry

lemma isOpen_iff_weird (s : Set ℤ) : IsOpen s ↔ ContainsArithProgression s :=
    Iff.rfl
```

接下来我们证明任意非空开集都是无限的

```
lemma nonempty_open_is_infinite (s : Set ℤ) (hs1 : IsOpen s) (hs2 :
    s.Nonempty) :
  s.Infinite := by
  rw [isOpen_iff_weird] at hs1
  rcases hs2 with ⟨x, hx⟩
  obtain ⟨m, hm1, hm2⟩ := hs1 x hx
  set f : ℤ → s := fun z => ⟨_, hm2 z⟩ with f_eq
  haveI infinite1 := Infinite.of_injective f (by
    rintro a b hab
    rw [f_eq, Subtype.ext_iff_val] at hab
    rwa [add_left_inj, mul_right_inj'] at hab
    linarith)
  rwa [Set.infinite_coe_iff] at infinite1
```

这是因为如果 $x\in s$，则存在 $1\le m$ 使得任意整数 k 都有 $mk+x\in s$。显然 $f:k\mapsto mk+x$ 是从 $\mathbb{Z}\to s$ 的单射。那么 s 是无限的。下一步我们证明任意等差数列都是又开又闭的

```
def arithProgression (a m : ℤ) := { z : ℤ | ∃ k, m * k + a = z }

lemma arithProgression_open (a m : ℤ) (hm : 1 ≤ m) : IsOpen (arithProgression
    a m) := by
  rw [isOpen_iff_weird]
  rintro _ ⟨k, rfl⟩
  exact ⟨m, hm, fun k' => ⟨k + k', by ring⟩⟩
```

为了证明 $mk+a|1 \le m, k \in \mathbb{Z}$ 是闭集，我们证明补集是开集。令 x 不再 $\{mk+a|1 \le m, k \in \mathbb{Z}\}$ 中，我们需要证明存在 $1 \le n$ 使得 $nk+x$ 不在 $\{mk+a|1 \le m, k \in \mathbb{Z}\}$，这是显然的：我们取 n 为 m，则 $mk+x$ 如果等于 $mk'+a$, 则有 $m(k'-k)+a=x$，这和 x 不再 $\{mk+a|1 \le m, k \in \mathbb{Z}\}$ 中矛盾

```
\begin{lstlisting}
lemma arithProgression_closed (a m : ℤ) (hm : 1 ≤ m) : IsClosed
    (arithProgression a m) := by
  rw [←isOpen_compl_iff, isOpen_iff_weird]
  intros x hx
  rw [arithProgression, Set.mem_compl_iff, Set.mem_setOf_eq, not_exists] at hx
  refine ⟨m, hm, fun k r => ?_⟩
  obtain ⟨k', hk'⟩ := r

  refine hx (k' - k) ?_
  grind
```

下一步我们证明 $\bigcup_p \{kp|k \in \mathbb{Z}\}$ 的补集是 $\{-1, 1\}$

```
lemma Seteq1 : ⋃( (p : ℕ) (_ : Nat.Prime p), arithProgression 0 p)  = {1, -1}
    := by sorry
```

这里只需要用到除去 $1, -1$ 之外的整数都有素因子即可。那么 $\bigcup_p \{kp|k \in \mathbb{Z}\}$ 不是闭集

```
lemma not_closed : ¬ IsClosed ⋃( (p : ℕ) (_ : Nat.Prime p), arithProgression
    0 p) := by
  rw [←isOpen_compl_iff.not, Seteq1]
  intro r
  have h1 := nonempty_open_is_infinite {1, -1} r ⟨1, by simp⟩
  have h2 : ({1, -1} : Set ℤ).Finite := by simp
  rw [←Set.not_infinite] at h2
  exact h2 h1


lemma not_closed' :  ¬ IsClosed ⋃( (p : setOf Nat.Prime), arithProgression 0
    (p : ℤ)) := by
  simp only [Set.mem_setOf_eq, Set.iUnion_coe_set]
  exact not_closed
```

所以素数必须是无限的

```
lemma infinite_prime : (setOf Nat.Prime).Infinite := by
  by_contra! r
  rw [Set.not_infinite] at r
```

```
  refine not_closed' ?_
  haveI : Finite (setOf Nat.Prime) := Set.finite_coe_iff.mpr r
  refine isClosed_iUnion_of_finite (fun i => arithProgression_closed _ _ ?_)
  norm_cast
  linarith [show (2 : ℕ) ≤ i by exact_mod_cast Nat.Prime.two_le i.2]
```

参考文献

[1] The Lean Developers. The Lean Language Reference, version 4.22.0. https://lean-lang.org/doc/reference/4.22.0/.