2025 Lean 与数学形式化讲义(2B)

上海交通大学 AI4MATH 团队

3 朴素集合论

3.1 定义与定理

策略 1 unfold

unfold 策略可以将对象展开为其定义. 其实通常不影响证明, 但是会方便我们观察对象的定义.

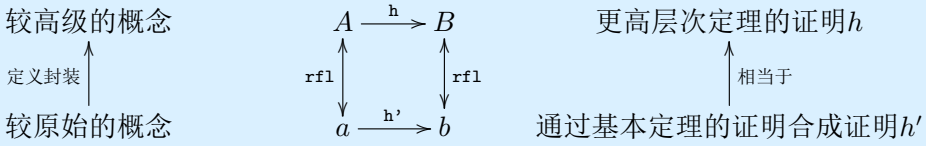
```
def a : Prop := sorry
def b : Prop := sorry
theorem b_of_a : a → b := by sorry

def A : Prop := a
def B : Prop := b
theorem th_name : A → B := by
  unfold A B
  exact b_of_a

theorem th_name2 : A → B := by
  intro hA
  have : a := Iff.rfl.mp hA
  have : b := b_of_a this
  have : B := Iff.rfl.mpr this
  exact this

theorem th_name3 : a → b := b_of_a
-- Iff.rfl and unfold tactic in fact does nothing.
```

在任何一个新定义被引入之初, 我们很少有工具能处理它, 此时大部分基本性质都通过还原概念的定义, 借助已成熟的理论框架证明. 而当一个成熟的概念已有诸多相关定理时, 便可直接使用已有定理.



例 3.1.1 展开定义

使用 unfold 策略展开定义, 方便我们查看定义的内容并用定义来证明命题.

```
def f (n : ℕ) : ℕ := n + 1
theorem th_name (n : ℕ) : f n = n + 1 := by
  unfold f            -- Goal: n + 1 = n + 1
  rfl                 -- No Goals
```

3.2 Mathlib 简介

大家应该或多或少在中学和本科的数学考试中遇到过“这个结论要不要证”的问题. 在数学考试中, 通常有一个设定的考核大纲, 其中一些主干的命题是教科书上已经证明的, 作为考试题目的命题被视为是还没有证明的, 而一些“二级结论”可能介于二者之间, 是否要为这些命题提供证明往往成为阅卷人和考生争议不休的问题. 这种情况主要源于中学和本科的数学考试的内容往往是人类几个世纪前早已建立的理论框架. 在数学研究中, 为了创造新的数学, 所有已被证明的数学均可为我们所用. 但是有一个问题始终存在: 我们有哪些命题的证明?

我们之前已经讲过, Lean 中借助已有结论证明定理等同于用已有命题的证明对象合成出新的命题的证明对象, 那么问题就变成了: 哪里存储了我们已有的证明. 对于 Lean 来说, 我们有一个社区维护的标准库, 即 Mathlib[1].

有许多可视化工具可以直观地感受 Mathlib 库的规模和其中复杂的关系, 例如 Mathlib Explorer[2]:

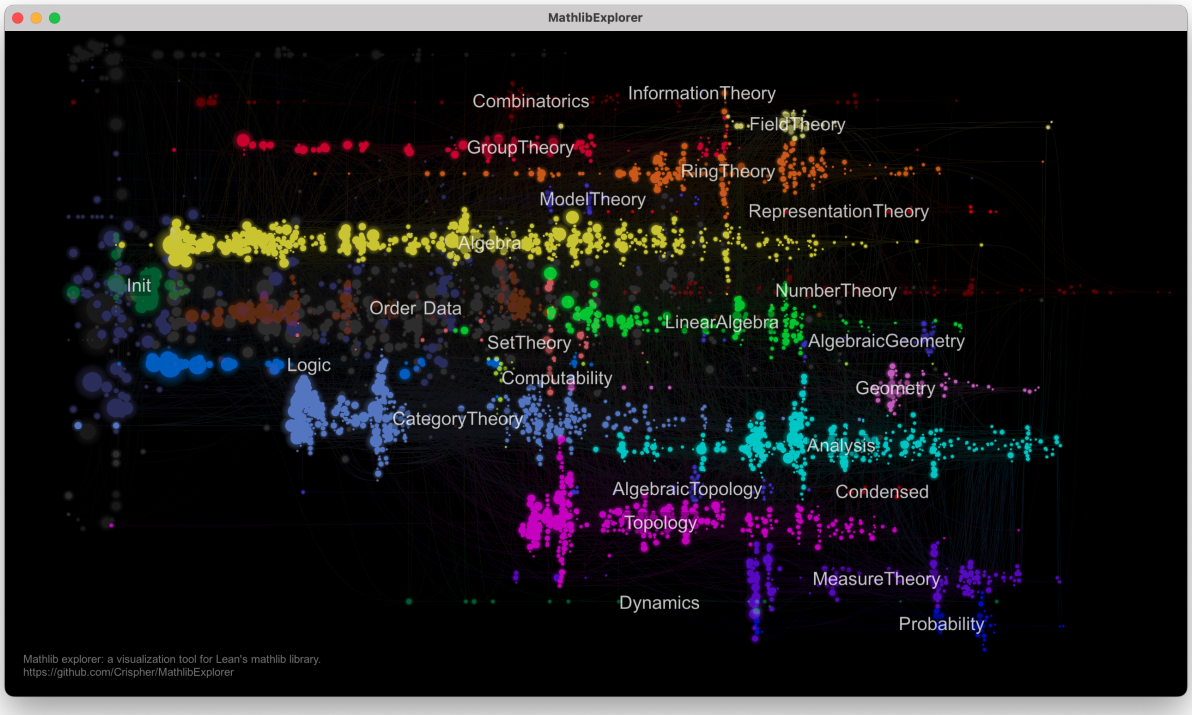


图 1: Mathlib Explorer 页面

图中每一个节点是 Mathlib 中的一个文件, 可以理解为一系列相近概念的集合. 节点之间通过连线刻画各个文件之间的依赖关系, 最左侧是最底层的数学基础, 右侧的概念依赖左侧概念建立.

Mathlib 中有逾 20 万个条目, 却仍然有诸多还未覆盖的数学领域以及各种缺陷. 在证明过程中使用 Mathlib 中的已有结论实在不是一件容易的事. 好在我们有一些方法来更加方便地查找和使用 Mathlib:

3.3 查找 Mathlib 中的内容

查找 Mathlib 中定理最基本的方式当然是直接查阅 Mathlib 库源代码. 如果要使用 Lean 进行特定领域的研究, 我们可能需要熟读相关领域的 Mathlib 源代码以了解一些常用结论的位置, 或是通过 `Ctrl +`

[1]关于 Mathlib 的详细信息请关注 Lean 社区主页或 Github 项目主页: https://github.com/leanprover-community/mathlib4

[2]Github 项目地址: https://github.com/Crispher/MathlibExplorer; 视频介绍: https://www.bilibili.com/video/BV1ex4y1r7tA

Click 跳转至一些概念的定义, 尝试从上下文和相邻文档中查找所需的定理. 目前 Mathlib 库缺乏简明易懂的自然语言文档[3], 这导致学习的成本十分高昂.

另一种方法是通过 Mathlib 中定理证明对象的变量命名习惯[4], 反向猜测所需命题证明的变量名, 借助 VS Code 的自动补全功能 (可通过 Ctrl + Space打开) 来查找.

例 3.3.1 Mathlib 中的定理命名习惯

一般严格小于号 $<$ 用 lt 表示, 而小于等于号 $\leq$ 用 le 表示, 函数类型用 of 表示, 算子之间用下滑线 _ 连接, 于是我们有: [a]

```
#check lt_of_le_of_lt        -- ... , a ≤ b → b < c → a < c
#check lt_of_lt_of_le        -- ... , a < b → b ≤ z → a < z
#check le_of_lt              -- ... , a < b → a ≤ b
```

[a]这里算子的顺序是: Goal, Arg1, Arg2, 以此类推

在策略证明环境中, 可以使用 apply? 策略:

策略 2 apply?

在策略证明中使用 apply? 策略, 会返回一系列可用的证明步骤和剩余目标供选择:

```
theorem th_name : /- proposition -/ := by
  apply?          -- suggestions: ...
  sorry
```

定理查找方面最显著的成果当属定理查找器. 稍早的成果有 Moogle, 但并未开源; 北京大学团队开发了 LeanSearch[5] 工具用于 Mathlib 库的查找. LeanSearch 有两种使用方法, 可以通过网页版[6]直接使用: 或是通过 Lean 内置的 #leansearch 命令查询:

语法 3.3.1 #leansearch

使用 #leansearch 命令可以搜索 Mathlib

```
#leansearch "For integers a and b, if a + b = 0, then a = -b."
    -- LeanSearch Search Results:
    -- 1. ...
    -- 2. ...
```

类似的命令还有 #moogle, #loogle 等.

LeanSearch 借由 Mathlib 库代码, 对象之间的依赖关系和 Mathlib 文档注释生成提示词提交大语言模型, 递归地对 Mathlib 代码进行非形式化得到各个条目的自然语言描述, 再通过嵌入模型 (Embedding) 将自然语言转化为高维向量, 与由用户输入并经大语言模型增强过后的查询语句嵌入得到的高维向量进行比对, 通过计算余弦距离 (Cosine Distance) 得到与查询语句最为相似的若干个 Mathlib 代码条目.

[3]自动生成的 Lean 官方文档: https://leanprover-community.github.io/mathlib4_docs/index.html
[4]Mathlib 命名习惯: https://leanprover-community.github.io/contribute/naming.html
[5]https://arxiv.org/pdf/2403.13310
[6]https://leansearch.net/

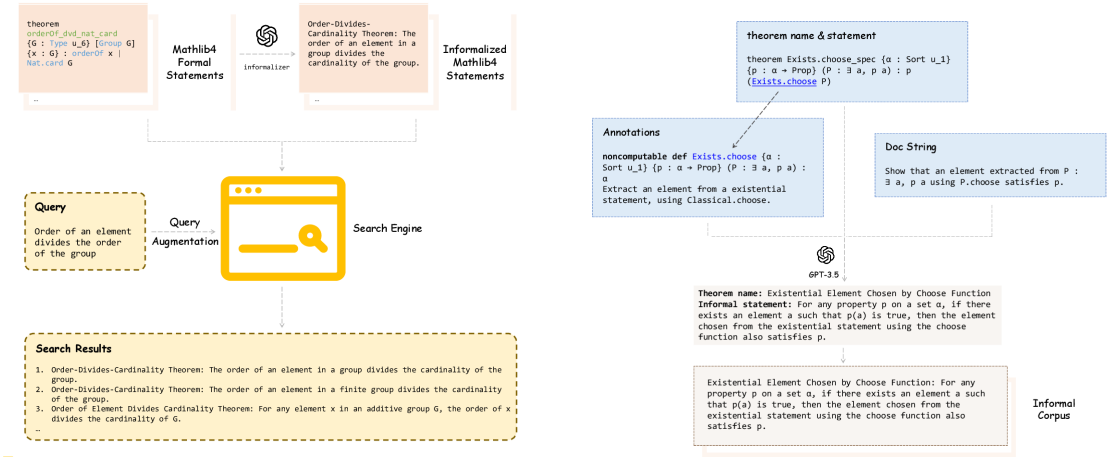


图 2: LeanSearch 工作原理 (图片引自 LeanSearch 论文*A Semantic Search Engine for Mathlib4*)

下面我们简单讲解 Mathlib 中朴素集合论的一些基本概念和定理. 由于 Mathlib 的规模实在太大, 我们无法一一介绍, 主要意在让大家对 Mathlib 的使用方法 (即 Lean 的常规使用风格) 有一个大致的概念.

3.4 基本概念

`Mathlib.Data.Set` 中定义的集合与我们常见的 ZFC 公理集合论理论框架不完全相同, 更像是高中学习的朴素集合论, 只是通过类型论元语言规避了悖论. 这种做法也许是 Russell 先生最早提出类型论的初衷. 以下是一些简单定义说明:

定义 3.4.1 类型集合 (Set)
若 α 是一个类型, 定义 α 上的一个一元谓词为一个 **α-集合**, 记作 $\texttt{Set}(\alpha)$:

```
def Set (α : Type u) := α → Prop
```

定义 3.4.2 由谓词分离的类型集合 (setOf)
若 α 是一个类型, p 是一个 α 上的一元谓词, 定义由 α 上的一元谓词 p 分离的 α 集合为一元谓词 p 自身, 记作 $\{x : \alpha \mid p(x)\}$:

```
def setOf {α : Type u} (p : α → Prop) : Set α := p
```

定义 3.4.3 属于关系 (Mem)
对任何一个类型 α, 定义 α 的元素 a 与 α-集合 s 满足**属于关系 (Membership)**, 当且仅当 a 适合一元谓词 s, 记作 $a \in s$:

```
protected def Mem (s : Set α) (a : α) : Prop := s a

class Membership (α : outParam (Type u)) (γ : Type v) where
  mem : γ → α → Prop
```

```
notation:50 a:50 " ∈ " b:50 => Membership.mem b a

instance : Membership α (Set α) := ⟨Set.Mem⟩
```

定义 3.4.4 子集关系 (Subset)
对于 α-集合 s_1, s_2, 定义 s_1, s_2 满足**子集关系**当且仅当在 s_1 中的元素总是在 s_2中, 记作 $s_1 \subseteq s_2$:

```
protected def Subset (s₁ s₂ : Set α) := ∀ ⦃a⦄, a ∈ s₁ → a ∈ s₂

class LE (α : Type u) where
  le : α → α → Prop

infix:50 " ≤ "  => LE.le

instance : LE (Set α) := ⟨Set.Subset⟩

class HasSubset (α : Type u) where
  Subset : α → α → Prop
export HasSubset (Subset)

infix:50 " ⊆ " => Subset

instance : HasSubset (Set α) := ⟨(· ≤ ·)⟩
```

定义 3.4.5 空集 (EmptyCollection)
定义为将任意元素映射到 False 的函数, 记作 $\emptyset$ 或 {}:

```
class EmptyCollection (α : Type u) where
  emptyCollection : α

notation "{" "}" => EmptyCollection.emptyCollection
notation "∅"     => EmptyCollection.emptyCollection

instance : EmptyCollection (Set α) := ⟨fun _ ↦ False⟩
```

定义 3.4.6 宇集 (univ)
一个类型中的所有元素构成的集合称为该类型的**宇集 (Universe)**, 记作 α 的宇集 univ(α):

```
def univ : Set α := {_a | True}
```

定义 3.4.7 插入运算 (insert)
对于 α-集合 s 和 α 类型的元素 a, 定义将 a 到 s 的**插入**为 $\{a\} \cup s$:

```
protected def insert (a : α) (s : Set α) : Set α
  := {b | b = a ∨ b ∈ s}

class Insert (α : outParam <| Type u) (γ : Type v) where
  insert : α → γ → γ

instance : Insert α (Set α) := ⟨Set.insert⟩
```

定义 3.4.8 单元集 (singleton)
设 a 是一个 α 类型的元素, 定义由 a 生成的单元集为只包含 a 的 α-集合, 记作 $\{a\}$:

```
protected def singleton (a : α) : Set α := {b | b = a}

class Singleton (α : outParam <| Type u) (β : Type v) where
  singleton : α → β

instance instSingletonSet : Singleton α (Set α) := ⟨
   Set.singleton⟩
```

定义 3.4.9 并集 (Union)
并集是析取命题的封装, 记作 $\cup$.

```
protected def union (s₁ s₂ : Set α) : Set α
  := {a | a ∈ s₁ ∨ a ∈ s₂}

class Union (α : Type u) where
  union : α → α → α

infixl:65 " ∪ " => Union.union

instance : Union (Set α) := ⟨Set.union⟩
```

定义 3.4.10 交集 (Inter)
交集是合取命题的封装, 记作 $\cap$.

```
protected def inter (s₁ s₂ : Set α) : Set α
  := {a | a ∈ s₁ ∧ a ∈ s₂}

class Inter (α : Type u) where
```

```
  inter : α → α → α

infixl:70 " ∩ " => Inter.inter

instance : Inter (Set α) := ⟨Set.inter⟩
```

定义 3.4.11 补集 (compl)
补集是否定命题的封装, 记作 ^{C}.

```
protected def compl (s : Set α) : Set α := {a | a ∉ s}

class HasCompl (α : Type*) where
  compl : α → α
export HasCompl (compl)

postfix:1024 "^C" => compl
```

定义 3.4.12 差集 (diff)
差集

```
protected def diff (s t : Set α) : Set α := {a ∈ s | a ∉ t}

class SDiff (α : Type u) where
  sdiff : α → α → α

infix:70 " \\ " => SDiff.sdiff

instance : SDiff (Set α) := ⟨Set.diff⟩
```

定义 3.4.13 幂集 (powerset)
集合的全体子集构成的集合称为该集合的**幂集 (Powerset)**, α-集合 s 的幂集记作 $\mathcal{P}(s)$.

```
def powerset (s : Set α) : Set (Set α) := {t | t ⊆ s}
prefix:100 "𝒫" => powerset
```

定义 3.4.14 像集 (image)
设 α, β 是类型, f 是从 α 到 β 的映射, s 是 α-集合, s 全体元素在 f 下的像构成的集合称为 f 的**像集 (Image)**, 记作 $f(s)$.

```
def image {β : Type v} (f : α → β) (s : Set α) : Set β := {f a |
    a ∈ s}
```

```
infixl:80 " '' " => image
```

定义 3.4.15 原像集 (preimage)
设 α,β 是类型, f 是从 α 到 β 的映射, s 是 β-集合, 映射到 s 的全体元素构成的集合称为 f 的**原像 (Preimage)**, 记作 $f^{-1}(s)$.

```
def preimage (f : α → β) (s : Set β) : Set α := {x | f x ∈ s}
infixl:80 " ⁻¹' " => preimage
```

定义 3.4.16 单射 (Injective)
设 α,β 是类型, f 是从 α 到 β 的映射, 称 f 是**单射 (Injection)**, 当且仅当不同元素在 f 下的像不同.

```
def Injective (f : α → β) : Prop :=
  ∀ {a₁ a₂}, f a₁ = f a₂ → a₁ = a₂
```

定义 3.4.17 满射 (Surjective)
设 α,β 是类型, f 是从 α 到 β 的映射, 称 f 是**满射 (Surjection)**, 当且仅当 β 中的任何元素有原像.

```
def Surjective (f : α → β) : Prop :=
  ∀ b, ∃ a, f a = b
```

定义 3.4.18 双射 (Bijective)
既是单射又是满射的映射称为**双射 (Bijection)**.

```
def Bijective (f : α → β) :=
  Injective f ∧ Surjective f
```

在 MIL C04 中, 我们会用到的绝大部分定理出现在 Mathlib 的以下四个路径:
集合相关基本概念的定义:

`Mathlib.Data.Set.Defs`

单射, 满射, 双射等概念的定义:

`Mathlib.Logic.Function.Defs`

集合的基本性质, 包括集合的并集, 交集, 补集, 差集等运算的定义和性质:

`Mathlib.Data.Set.Basic`

像, 原像等概念的定义和基本定理:

`Mathlib.Data.Set.Operations`

3.5 自然语言证明的形式化与证明蓝图

现在我们尝试形式化证明讲义 1C 中通过 $\varepsilon-\delta$ 语言形式化的定理 `lim_uniq`. 我们先写一版自然语言证明, 再逐步构建出形式化证明的蓝图:

```
theorem lim_uniq
  {a_ : ℕ → ℝ}
  {a b : ℝ}
  (ha : lim a_ a)
  (hb : lim a_ b)
: a = b
:= by sorry
```

证明: 1. 运用反证法. 假设 $a \neq b$, 于是要么 $a > b$, 要么 $b < a$.

2. 不失一般性, 不妨设 $a > b$.

3. 根据极限定义, 得:

$$\lim_{n\to+\infty} a_n = a \iff \forall \varepsilon > 0, \exists N : \mathbb{N}, \forall n > N, -\varepsilon < a_n - a < \varepsilon$$

$$\lim_{n\to+\infty} a_n = b \iff \forall \varepsilon > 0, \exists N : \mathbb{N}, \forall n > N, -\varepsilon < a_n - b < \varepsilon$$

4. 在上两式中取 $\varepsilon = \dfrac{a-b}{2}$, 满足 $\varepsilon > 0$.

5. 将两式的存在量词分别实例化为 N_a, N_b, 得:

$$\forall n > N_a, -\varepsilon < a_n - a < \varepsilon$$

$$\forall n > N_b, -\varepsilon < a_n - b < \varepsilon$$

6. 取 $n = \max(N_a, N_b) + 1$, 有 $n > N_a \wedge n > N_b$, 得:

$$-\varepsilon < a_n - a < \varepsilon$$

$$-\varepsilon < a_n - b < \varepsilon$$

7. 由 ε 定义, 得:

$$-\frac{a-b}{2} < a_n - a < \frac{a-b}{2}$$
$$-\frac{a-b}{2} < a_n - b < \frac{a-b}{2}$$

8. 化简得到:

$$a_n > \frac{a+b}{2}$$
$$a_n < \frac{a+b}{2}$$

9. 矛盾! □

现在我们逐步分析证明, 并将其形式化为 Lean 代码.

1. 反证法, 对应的证明状态变化是:

$$\boxed{(a_n : \mathbb{N} \to \mathbb{R}), (a, b : \mathbb{R}), \left(h_a : \lim_{n\to+\infty} a_n = a\right), \left(h_b : \lim_{n\to+\infty} a_n = b\right) \vdash a = b}$$

$$\downarrow \texttt{by\_contra!}(h_{ab})$$

$$\boxed{(a_n : \mathbb{N} \to \mathbb{R}), (a, b : \mathbb{R}), \left(h_a : \lim_{n\to+\infty} a_n = a\right), \left(h_b : \lim_{n\to+\infty} a_n = b\right) (h_{ab} : a \neq b) \vdash \texttt{False}}$$

将假设 $(h_{ab} : a \neq b)$ 改写为 $a - b > 0 \vee a - b < 0$, 可能需要用到定理 $(a, b : \mathbb{R}), (a \neq b) : a > b \vee a < b$ 的证明. 通过 LeanSearch 查询, 需要的定理是全序结构上的比较定理 `lt_or_gt_of_ne`, 证明状态变化为:

$$\boxed{\sim, (h_{ab} : a \neq b), \sim}$$

$$\downarrow \texttt{apply lt\_or\_gt\_of\_ne at } h_{ab}$$

$$\boxed{\sim, (h_{ab} : a < b \vee a > b), \sim}$$

2. 不失一般性是个比较麻烦的问题, 我们一般不愿意把两种等同的情况分别证明. 这种情况可以用 `wlog` 策略处理, 我们可以"不妨"引入一个条件, 需要说明为何其他情况可以划归为这个条件下的情况, 接着用引入的条件继续证明.

$$\boxed{\sim \vdash \sim}$$

$$\texttt{wlog} \cdots \swarrow \searrow$$

$$\boxed{(h : \neg a > b), (wlog : \dots), \sim} \qquad \boxed{(h : a > b), \sim}$$

我们要解决第一个新增分支目标, 即证明为何 $\neg a > b$ 的情况可以用"不妨设"后的条件证明, 只需要反向代入 h_b, h_a 即可. 但是一些命题的顺序改变了, 不想写复杂的项证明的话, 用 `apply` 策略保留语法缺口, 进一步拆分证明状态:

$$\boxed{\sim \vdash \texttt{False}}$$

$$\swarrow \texttt{apply } wlog(h_b)(h_a) \searrow$$

$$\boxed{\sim \vdash b < a \vee b > a} \qquad \boxed{\sim \vdash b > a}$$

这两个目标的证明已经十分简单了, 完成分支目标的证明, 便成功引入了 "不妨设 $a > b$" 的条件, 可以回到正轨上来.

3. 这一步本质上是在展开极限定义, 用 `unfold` 策略即可:

$$\boxed{\left(h_a : \lim_{n\to+\infty} a_n = a\right), \left(h_b : \lim_{n\to+\infty} a_n = b\right), \sim}$$

$$\downarrow \texttt{unfold lim at *}$$

$$\boxed{(h_a : \forall \varepsilon > 0, \dots), (h_b : \forall \varepsilon > 0, \dots), \sim}$$

4. 取值, 使用 `let` 策略引入一个新变量 ε.

$$\boxed{\sim \vdash \sim}$$

$$\downarrow \texttt{let}\ \varepsilon := (a-b)/2$$

$$\boxed{(\varepsilon : \mathbb{R} := (a-b)/2), \sim}$$

后面要用到 $\varepsilon > 0$ 的证明, 我们提前引入一个引理准备好, 这个引理的证明应该十分简单.

$$\boxed{\sim \vdash \sim}$$

$$\downarrow \texttt{have}\ \varepsilon\_\texttt{pos} : \varepsilon > 0 := \dots$$

$$\boxed{(\varepsilon\_\texttt{pos} : \varepsilon > 0), \sim}$$

接着代入消去 h_a, h_b 的全称量词, 化简条件.

$$\boxed{\sim, (h_a : \forall \varepsilon > 0, \dots), (h_b : \forall \varepsilon > 0, \dots), \sim}$$

$$\downarrow \texttt{have}\ h_a := h_a(\varepsilon)(\varepsilon\_\texttt{pos})$$

$$\boxed{\sim, (h_a : \exists N : \mathbb{N}, \dots), (h_b : \forall \varepsilon > 0, \dots), \sim}$$

$$\downarrow \texttt{have}\ h_b := h_b(\varepsilon)(\varepsilon\_\texttt{pos})$$

$$\boxed{\sim, (h_a : \exists N : \mathbb{N}, \dots), (h_b : \exists N : \mathbb{N}, \dots), \sim}$$

5. 全称量词实例化, 运用 `rcases` 或 `obtain` 策略:

$$\boxed{\sim, (h_a : \exists N : \mathbb{N}, \dots), (h_b : \exists N : \mathbb{N}, \dots), \sim}$$

$$\downarrow \texttt{rcases}\ h_a\ \texttt{with}\ \langle N_a, h_a \rangle$$

$$\boxed{\sim, (h_a : \forall n > N, \dots), (h_b : \exists N : \mathbb{N}, \dots), \sim}$$

$$\downarrow \texttt{rcases}\ h_b\ \texttt{with}\ \langle N_b, h_b \rangle$$

$$\boxed{\sim, (N_a : \mathbb{N}), (h_a : \forall n > N_a, \dots), (N_b : \mathbb{N}), (h_b : \forall n > N_b, \dots), \sim}$$

6. 取 $n = \max(N_a, N_b) + 1$, 运用 `let` 策略引入一个新变量 n, 证明状态变化为:

$$\boxed{\sim \vdash \sim}$$

$$\downarrow \texttt{let}\ N := \max(N_a, N_b)$$

$$\boxed{(N : \mathbb{N}), \sim}$$

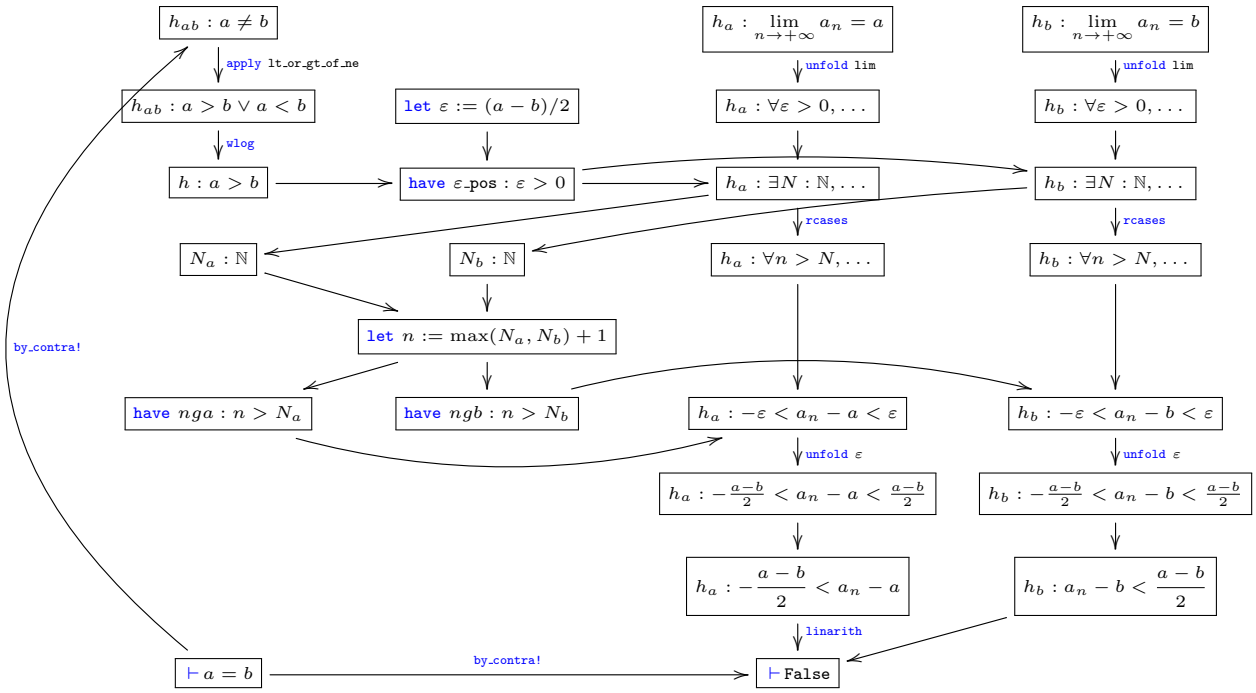
用 `have` 策略引入引理证明 $n > N_a$ 和 $n > N_b$, 并代入全称量词, 与之前相同, 不再赘述.

7. 运用 `unfold` 策略展开 ε 的定义, 与之前相同.

8. 用 `have` 策略分别取出 ha 与 hb 的一侧, 构造矛盾不等式组.

9. 如果不想一步步化简证明, 可以用 `linarith` 策略直接完成证明.

但是这样一步步追踪证明状态的变化实在太麻烦了, 我们可以用蓝图来表示整个证明:



通过绘制证明蓝图, 我们可以很清晰的查看一个定理证明的结构, 以及证明状态中各个变量, 假设以及目标随证明步骤推进的变化流程. 对于较为宏大的数学问题, 可能不需要绘制如此细致的蓝图, 但也有助于我们理清证明的思路并构建 Lean 项目, 让数学家之间更好地进行分工合作.

陶哲轩 (Terrence Tao) 等人通过 Patrick Massot 开发的蓝图工具为 Polynomial Freiman-Ruzsa 猜想 (PFR conjecture) 的形式化证明绘制了蓝图[7], 本文所示的蓝图与之不尽相同, 但大约是这样一个概念.

3.6 Shroeder-Bernstein 定理

作为实践, 我们将证明 Shroeder-Bernstein 定理. 大家或许或多或少学习过该定理的集合论表述, 在此我们将证明其大同小异的类型论版本. 尝试写出它的一个自然语言证明, 画出形式化证明的蓝图, 最后将它改写为 Lean 代码. [8]

定理 3.6.1 Shroeder-Bernstein 定理 (Shroeder-Bernstein Theorem)
设 α, β 是类型, f 是 α 映射到β, $g : \beta \to \alpha$, f, g 均为单射, 则存在映射 $h : \alpha \to \beta$, 使得 h 是双射.

```
theorem schroeder_bernstein
  {α β : Type*}              --  α β are types
  [Nonempty β]               --  β is non-empty
  {f : α → β}                --  f is a function from α to β
  {g : β → α}                --  g is a function from β to α
  (hf : Injective f)         --  f is injective
  (hg : Injective g)         --  g is injective
: ∃ h : α → β, Bijective h   --  Exists a bijection h : α → β
:= by sorry
```

[7] https://teorth.github.io/pfr/

[8] 收到反馈前两节课讲得太难了, S-B 定理的构造需要用到归纳类型, 那就更难了, 作为拓展, 有兴趣的同学可以自学 MIL 完成.