arXiv:2611.29603v1 [cs.LO] 22 Nov 2026

模重写与绑定变量排列的定理哈希支持对 Mathlib 20 多万个定理进行等价性搜索

Samuel Leßmann[1(✉)] Simon Henniger[2]、Gabriel Poesia[3]和 Nada Amin[2]

[1] 剑桥大学，英国剑桥 sl2379@cam.ac.uk[2] 哈佛大学，美国马萨诸塞州剑桥 shenniger@fas.harvard.edu, namin@seas.harvard.edu[3] 密歇根大学，美国密歇根州安娜堡 poesía@umich.edu



摘要。 与所有代码一样，Lean 库在开发过程中会逐渐积累技术债务和结构性缺陷。一个值得特别关注的问题是重复定理：大型库的不同部分可能在无意间包含并证明同一个定理，造成重复的维护工作。工具可以帮助检测冗余并进行清理。例如，Mathlib 重复声明检查器会对顶层绑定符排序并擦除绑定信息和宇宙层级，然后以语法方式比较定理类型。如果检测等价需要展开、重新定向或逻辑变换，这种比较就无法发现。本研究提出了一种方法，用于检测在参数顺序以及依赖类型的柯里化／反柯里化方面等价的定理。我们为 Lean 实现了该方法，并将其应用于包含 20 多万个定理的 Mathlib 库，证明了该方法计算开销低且易于扩展。最后，我们探讨了这项工作如何助力减轻 Lean 库的维护负担。

关键词： 证明重构 · 类型同构 · 模重写 · Lean · 代码检查器 · 形式化验证 · 依赖类型论 · 交互式定理证明

1 引言

随着 Lean 日益普及，Lean 代码的数量也显著增加。以 Mathlib 为例，其规模已增长到超过 220 万行 Lean 代码。在这样的规模下，弃用和技术债务成为库持续开发的主要风险 [3,12]。用于代码维护的工具变得至关重要。人工智能的出现进一步加剧了这一问题，使较新的 Lean 项目能够以更快的速度变得更加庞大，例如 Anthropic 对费马大定理的形式化（1300 万行）[1,2]，或 OpenAI 为纳维–斯托克斯方程和欧拉方程有限时间爆破解析提供的 Lean 证书（641,000 行）[16,17]。

重复定理尤其令人担忧。位于不同位置的两个定理证明了同一件事，会导致混淆、代码膨胀以及社区工作重复。

然而，鉴于 Lean 类型系统的表达能力，自动查找重复项是一项困难的任务。一般而言，证明任意类型等价与证明任意定理同样困难，因此无法通过算法判定。

我们提出了一种方法，用于查找一部分可通过终止的句法重写找到的类型等价中的重复项。为此，我们借鉴了类型同构方面的一些理论成果，例如对带类型的 λ-演算中可证明同构的完备公理化 [8,11,13], 、它们在证明助手中的应用 [5], 以及它们向依赖积的扩展 [20]。不过，我们并未将这些成果用于搜索，而是为整个 Lean 库建立索引，以查找重复项。

我们计算的关系有意窄于逻辑等价。只有当一组固定的同构模式和定义展开应用于命题位置后，将两个陈述都重写为除重命名和重新排序外相同的范式时，我们才将它们视为相同。由于这种关系较为狭窄，因此可以在包含 20 万个 Mathlib 定理的库中以较低成本判定，同时仍能找到相关结果。

贡献。 我们作出以下贡献:

– 我们展示了一个包含八条规则的终止重写系统，能够找到定义相等检查所遗漏的相关重复定理。– 基于这一洞见，我们实现了一个工具，可在任意 Lean 项目中找到重复陈述，并使用内核检查的证明证明其等价性（第 3 节）。该工具对陈述进行规范化，依据同构不变量签名将范式分桶，在每个桶内判定同构关系，并利用通用传递引理构建证书。其元理论的性质在该工具实际使用的定义上进行了形式化和证明。– 我们在 Lean core、Batteries 和 Mathlib 库上评估了该工具，共处理 286,264 个陈述，并找到 1,747 个冗余证明（第 4 节）。

2 理论基础

为了在 Lean 的类型理论中一个定义良好的片段内开展工作，我们需要满足两个性质。首先，我们希望能够在该片段中忠实地表示 Lean 命题，并且能够将其提取出来。其次，我们希望在对绑定变量和假设进行重排与重命名后，仍能进行终止的规范化，并能够使用高效的图同构启发式方法来匹配它们。带有交换结构的项等式与图同构之间的联系已由 Basin [6] 研究。

2.1 项与图

我们在图1中定义的项语言 **Tm** 是 Lean 的一个句法片段，涵盖项、类型和命题。我们使用 **Tm** 并不是将其作为独立的类型理论，而是将其作为 Lean 命题的一个句法片段：它没有类型规则，也没有归约，并且每个项都是通过编码 Lean 语句获得的。因此，良类型性继承自 Lean，并在检查回读结果时由内核重新确立。该设计类似于 [8]和[21]。由于我们只想寻找重复定理，因此只处理命题。其主体为命题的 $\forall$（这也涵盖蕴含和否定）会依据规则展开；而陪域不是命题的 Π 则会作为惰性形式保留，因为任意宇宙中类型之间的同构会使理论复杂程度超出我们的使用需求。

项语言 **Tm**	Lean 读取
$\kappa ::= \exists \mid \wedge$	Σ 的两种读取方式
$\oplus ::= \vee \mid \leftrightarrow$	交换二元联结词
$t, a, b, A, B, T ::= i$	有界变量，de Bruijn 指数 i
$\mid$ 自由变量(n)	指向名为 n 的变量节点的指针
$\mid$ 原子$_k$	原子表中的条目 k （闭合、不透明）
$\mid\ t\ t'$	应用
$\mid\ \lambda A.\ b$	函数 $x :\ A \Rightarrow b$
$\mid\ \forall A.\ B$	$\forall x :\ A,\quad B$ 其中 B : 命题 (也包括 $A \to B,\quad \neg A$)
$\mid\ \Pi A.\ B$	$(x :\ A) \to B$ 与 B 不是命题
$\mid\ \Sigma^{\kappa} A_{\circ}\ B$	$\exists x :\ A,\quad B$ 如果 $\kappa = \exists$;　$A \wedge B$ 如果 $\kappa = \wedge$
$\mid\ A \oplus B$	$A \vee B,\quad A \leftrightarrow B$
$\mid\ a =_T b$	@等式 $T\ a\ b$
$\mid\ \top$	`True`
$\mid\ \star$	真.引入
$\mid\ \ a, b\ _T$	且.引入 $a\ b$, 存在.引入 $a\ b$ 在 Σ-类型 T
$\mid\ t.1 \mid t.2$	t 的投影：$A \wedge B$

图 1. Lean 的句法片段 **Tm** 中的项、类型和命题，其中局部绑定符使用德布鲁因指标，图节点使用命名指针。证明项（$\star$、$a, b_T, t.1, t.2$）会保留以便回读为 Lean。$\forall$、Π、$\Sigma^{\wedge}$ 上的边条件以及对偶注释由编码器保证，而非由语法保证。为便于阅读，示例将有界变量 $^{v}{}_{j}$ 记作其绑定符从外侧数起的深度 j，而非其索引，并将 fv(n) 写作 x_n。

$N ::= x_n :\ A$	类型为 A 的变量节点：名称 n
$\mid\ \Delta \longrightarrow C$	箭头：假设 Δ 推出结论 C
$\mid\ \Delta \oplus C$	图上的联结词，$\oplus \in \{\vee,\ \leftrightarrow\}$
$\mid\ a =_T b$	类型 T 上的方程
$\Delta,\ C ::= N_1,\ \ldots,\ N_m$	图：节点的有限积（$m \geq 0$）

图2. 图**G** 是由节点构成的积，其中积是一个望远镜，具有具名成员以及成员之间的指针。

我们在图2中定义了一个用于表示项的图：图是节点的有限积，而节点可以是带有类型的具名变量、从假设的积指向结论的积的箭头、两侧各带一个积的连接词节点，或无名的方程节点。积是 de Bruijn [9]意义下的望远镜：后面的成员可以按名称引用前面的成员，列表顺序就是绑定子顺序。当名称唯一、标签在局部闭合且每个指针都指向作用域内的成员时，图是良构的；不带指针的语句的初始单节点图 $x_0 :\ A$ 是良构的，并且每个重写步骤都会保持良构性。

当且仅当对名称进行双射重命名，并将其同时应用于节点名称和指针，再经过有限次图3中的步骤 $\leftrightsquigarrow$，使一个图变为另一个图时，这两个图才由 **Iso** 关联。一个步骤可以交换乘积中两个相邻的成员（Swap）、连接词节点两侧（Flip），或方程节点两侧（FlipEq）。这些步骤可应用于任意嵌套深度的每个乘积中，包括顶层、箭头的定义域或结论，以及连接词节点的一侧，但绝不会应用于标签内部。

标签内部的方程或连接词因此只能在重命名的意义下进行比较。**Iso** 是比较和判定范式所依据的关系。对于良构图，已证明重写系统 **R** 关于此 **Iso** 具有合流性，其形式为第 2.3 节所述的单源形式。

$$C \;::=\; \square \;\mid\; N, C \;\mid\; (C \to c), g \;\mid\; (d \to C), g \;\mid\; (C \oplus c), g \;\mid\; (d \oplus C), g$$

$$\frac{}{C[\,N,\,N',\,g\,] \;\leftrightsquigarrow\; C[\,N',\,N,\,g\,]}\ \textsc{Swap} \qquad \frac{}{C[\,(d \oplus c),\,g\,] \;\leftrightsquigarrow\; C[\,(c \oplus d),\,g\,]}\ \textsc{Flip}$$

$$\frac{}{C[\,a =_T b,\,g\,] \;\leftrightsquigarrow\; C[\,b =_T a,\,g\,]}\ \textsc{FlipEq} \qquad \frac{\varphi \text{ a bijection on names} \qquad \varphi(g) \;\leftrightsquigarrow^{*}\; g'}{g \;\mathbf{Iso}\; g'}\ \textsc{Iso}$$

图3。 关系 **Iso** 及其步骤和 **R** 的规则所适用的积上下文 C，其中 N 遍历节点， g、 d 和 c 遍历图，$\oplus$ 遍历 $\vee$ 和 $\leftrightarrow$。 $\varphi(g)$ 沿着 φ重命名 g 的每个节点名称和每个指针。三个步骤均无附加条件。

2.2 重写系统R

R的八条规则（图4）可以在任意积的任意成员处使用：在顶层、箭头的定义域或结论中，或连接词节点的一侧，但绝不能在标签内部使用。Pi 和 Sigma 将绑定子从标签中提升到周围的积中，同时选择新的名称。ArrowMerge 规则将嵌套箭头展平，从而表示柯里化。Unit、ArrowUnit 和 ArrowErase 移除 $\top$、空定义域以及具有空结论的箭头。Conn 和 Eq 将 $\vee$、 $\leftrightarrow$ 和 $=$ 转换为节点，以体现交换性。Pi、Conn 和 Eq 仅适用于整个图中没有任何指针引用的成员；也就是说，稍后在其他位置使用的假设绝不会被拆分。

Iso 忽略依赖顺序，因此它本身并不能保持良构性。不过，关键在于 **R** 作用于 **Iso**等价类：若 g 与 g' 在 **Iso** 意义下等价，且 g 经过一步变换得到 h，那么g' 只需一步便可变换到某个 h' **Iso** 等价于 h 的对象。该工具只比较良构范式，因此已识别对两侧的成员顺序都符合依赖关系。

考虑以下两个命题 T_1 和 T_2

$$T_1 \;=\; \forall m\, n : \mathbb{N}.\; 0 < n \to m \mid n \to m \le n$$

$$T_2 \;=\; \forall n\, m : \mathbb{N}.\; (\exists k.\, n = m \cdot k) \wedge 0 < n \to m \le n$$

其中 $m \mid n$ 展开为 $\exists k。\; n = m \cdot k$。从单节点图 $x_0 :\; T_1$开始，**R** 的规则会生成

$x_0 \;\; T_1 : (:\,) \longrightarrow (h_1 :\;\; \forall n :\;\; \mathbb{N}。\;\; 0 < n \to \cdots)$
$\to_{\mathbf{R}} \;\; m \;\; \mathbb{N}$ — PI

$\to_{\mathbf{R}} (m :\;\; \mathbb{N}) \longrightarrow ((n :\;\; \mathbb{N}) \longrightarrow (h_2 :\;\; 0 < n \to \cdots))$ — Π（在结论内部）

$\to_{\mathbf{R}} (m :\; \mathbb{N},\;\; n :\; \mathbb{N}) \longrightarrow (h_2 :\; 0 < n \to (\exists k。\; n = m \cdot k) \to m \le n)$ — ArrowMerge

$\to^{4}_{\mathbf{R}} (m :\; \mathbb{N},\;\; n :\; \mathbb{N},\;\; p :\; 0 < n,\;\; q :\; \exists k。\; n = m \cdot k) \longrightarrow (r :\; m \le n)$ — Π ×2, ArrowMerge ×2

$\to_{\mathbf{R}} (m :\; \mathbb{N}, n :\; \mathbb{N}, p :\; 0 < n, k :\; \mathbb{N}, e :\; n = m\, k) \longrightarrow (r :\; m \le n)$ — $\text{Sigma}^{\exists}$

$\to_{\mathbf{R}} \;(m :\;\; \mathbb{N},\;\; n :\;\; \mathbb{N},\;\; p :\;\; 0 < n,\;\; k :\;\; \mathbb{N},\;\; n =_{\mathbb{N}} m \cdot k) \;\longrightarrow\; (r :\;\; m \le n)$ — 等式 $=:\;\; N_{T_1}$

$$\frac{x, y \,\#\, C[\,x_i : \forall A.B,\ g\,] \qquad i \notin \mathsf{refs}(C[\,x_i : \forall A.B,\ g\,])}{C[\,x_i : \forall A.B,\ g\,] \ \to_{\mathbf{R}}\ C[\,(x_x : A) \to (x_y : B\{x\}),\ g\,]}\ \textsc{Pi}$$

$$\frac{x, y \,\#\, C[\,x_i : \Sigma^{\kappa} A.B,\ g\,]}{C[\,x_i : \Sigma^{\kappa} A.B,\ g\,] \ \to_{\mathbf{R}}\ C[\,x_x : A,\ x_y : B\{x\},\ g\,]\left[\langle \mathsf{fv}(x), \mathsf{fv}(y)\rangle_{\Sigma^{\kappa} A.B} \,/\, i\right]}\ \textsc{Sigma}$$

$$\frac{x, y \,\#\, C[\,x_i : A \oplus B,\ g\,] \qquad i \notin \mathsf{refs}(C[\,x_i : A \oplus B,\ g\,])}{C[\,x_i : A \oplus B,\ g\,] \ \to_{\mathbf{R}}\ C[\,(x_x : A) \oplus (x_y : B),\ g\,]}\ \textsc{Conn}$$

$$\frac{i \notin \mathsf{refs}(C[\,x_i : a =_T b,\ g\,])}{C[\,x_i : a =_T b,\ g\,] \ \to_{\mathbf{R}}\ C[\,a =_T b,\ g\,]}\ \textsc{Eq} \qquad \frac{}{C[\,x_i : \top,\ g\,] \ \to_{\mathbf{R}}\ C[\,g\,][\star/i]}\ \textsc{Unit}$$

$$\frac{}{C[\,(d \to (d' \to c)),\ g\,] \ \to_{\mathbf{R}}\ C[\,(d, d' \to c),\ g\,]}\ \textsc{ArrowMerge}$$

$$\frac{}{C[\,(\cdot \to c),\ g\,] \ \to_{\mathbf{R}}\ C[\,c,\ g\,]}\ \textsc{ArrowUnit} \qquad \frac{}{C[\,(d \to \cdot),\ g\,] \ \to_{\mathbf{R}}\ C[\,g\,]}\ \textsc{ArrowErase}$$

图4。 在图上定义的重写系统**R**，其中 C[·] 是乘积上下文（图3），# 表示新鲜性，refs 表示被引用名称的集合。

并且，根据 $x_0 :\ T_2$,

$$\begin{aligned}
&x_0 : T_2 \to^{5}_{\mathbf{R}} (n :\ \mathbb{N},\ \ m :\ \mathbb{N},\ \ h : (\exists k_{\circ}\ \ n = m \cdot k)\ \wedge 0 < n) \longrightarrow (r :\ m \le n) && \Pi \times 3,\ \text{ArrowMerge} \times 2\\
&\to_{\mathbf{R}} (n :\ \mathbb{N},\ m :\ \mathbb{N},\ q :\ \exists k.\, n = m \cdot k,\ p :\ 0 < n) \longrightarrow (r :\ m \le n) && \text{Sigma}^{\wedge}\\
&\to_{\mathbf{R}} (n :\ \mathbb{N},\ m :\ \mathbb{N},\ k :\ \mathbb{N},\ e :\ n = m \cdot k,\ p :\ 0 < n) \longrightarrow (r :\ m \le n) && \text{Sigma}^{\exists}\\
&\to_{\mathbf{R}} (n :\ \mathbb{N},\ \ m :\ \mathbb{N},\ \ k :\ \mathbb{N},\ \ n =_{\mathbb{N}} m \cdot k,\ \ p :\ 0 < n) \longrightarrow (r :\ m \le n) && \text{等式}\ =:\ N_{T_2}
\end{aligned}$$

N_{T_1} 和 N_{T_2} 都是**R**-范式：剩余的变量节点具有原子类型（$\mathbb{N}$、 $0 < n$、$m \le n$），且没有规则可应用。它们通过置换定义域的成员而相关联，无需重命名，因为两条轨迹为对应节点选择了相同的名称。工具会自行生成名称，因此，图 5 中T_1 和 T_2 的范式也可通过重命名相关联。

2.3 在 Lean 中证明元理论

我们在 Lean 中形式化了 **Tm**、**R**、**G** 和**Iso**，并证明了它们的多项重要性质。所有定理的形式化方式都确保工具的实现使用的正是我们证明这些性质时所用的定义。我们对 **Tm**、**R** 和 **Iso** 的自定义义，与 Mathlib 和 CSLib [4]中关系和重写系统的规范定义相连接。这些定义在单独的文件夹中得到证明，并通过比较器 [14]验证。**R** 在所有图上都具有终止性，无须良构性假设；其依据是对脊构造子和权重定义的度量，且每条规则都会使该度量递减。**Iso** 按照第 2.2 节的含义，是**R** 的一种模拟。对于不含指针或松散绑定变量的每个标签，其初始图都满足良构性，而且每个**R**步都保持良构性。

在良构图上，**R** 在**Iso** 模下具有单源形式的合流性。若一个良构图 g 可约化为 h_1 和h_2，则h_1 和h_2 可约化为两个由**Iso**关联的图。这个抽象引理是纽曼论证在模拟假设下贯穿**Iso** 的特例，也是 Huet 模共尾合流引理的一个特例。因此，从同一个良构图可达的范式在**Iso**意义下唯一；正规化器依据终止性定理进行良基递归定义，不使用 fuel，并返回一个通过**R**步可达的**R**-范式；对于良构图，该范式还与每个可达范式都由**Iso** 关联。**Iso** 可判定；其参考判定器通过结构递归证明为全函数，并且在所有图上都可靠且完备。比较前用于对图分桶的键在**Iso**下保持不变。所有这些定理都不依赖通常的命题外延性、Classical.choice 和 Quot.sound 之外的任何公理。单页文件 Challenge.lean 仅导入 Mathlib 和 CSLib，重新陈述了**G**、**R**、**Iso**、良构性及相关定理陈述。

需要注意的是，在 Lean 中证明的内容与 Lean 命题的含义没有任何关联。图没有语义，对于类型同构理论也没有适切性或完备性结果。Lean 与此的联系，是通过等价性证书针对每个已识别对建立的。

3 实现

基于我们的洞见，我们构建了一个扫描 Lean 库以查找重复项的工具。重要的是，我们使用与证明 **Tm**、**R**、**G** 和 **Iso** 的性质相同的定义来实现该工具。图 5 展示了该工具在上文示例中的处理流程。

该工具扫描所有作为命题的 Lean 声明，并将其编码为 **Tm** 项。为了找到更多重复项，就像我们在示例中对 T_1 和 T_2 所做的那样，我们需要将 $m \mid n$ 展开为 $\exists k。 n = m \cdot k$。我们需要施加恰到好处的展开程度，而这 notoriously 难以实现。如果展开过多，项的大小就会爆炸。因此，我们只展开标注为可约或实例可约的定义（例如像 $m \mid n$ 这样的定义）。

将一个陈述编码为 **Tm** 项，并用它在 **G** 中构建初始图后，使用重写系统 **R** 将其归一化。我们此前已经证明，这一过程总会终止。

在默认的全库模式下，驱动程序使用已证明的 **Iso**不变签名来形成桶，并使用已证明的 **Iso** 判定过程将成员与类代表元进行比较。判定过程首先检查快速匹配器提出的见证；如果这一尝试无法确立同构，则回退到完整的参考判定器。尽管目前尚无用于判定图同构的已知多项式时间算法，但我们可以很好地利用一维 Weisfeiler–Leman 算法 [22]中的启发式颜色细化。范式中的每个节点都会先根据其标签（其中所有指针均置空）以及其在图中的位置计算出一个颜色。每一轮细化都会将节点的颜色替换为一个哈希值，该哈希值取决于其旧颜色、它所指向节点的颜色，以及对于每个传入指针，指向它的节点的颜色和该指针出现位置的组合。当颜色类不再发生分裂时，迭代停止。稳定的颜色成为节点哈希值，方程的两侧以交换方式合并，而望远镜中成员的哈希值则相加，

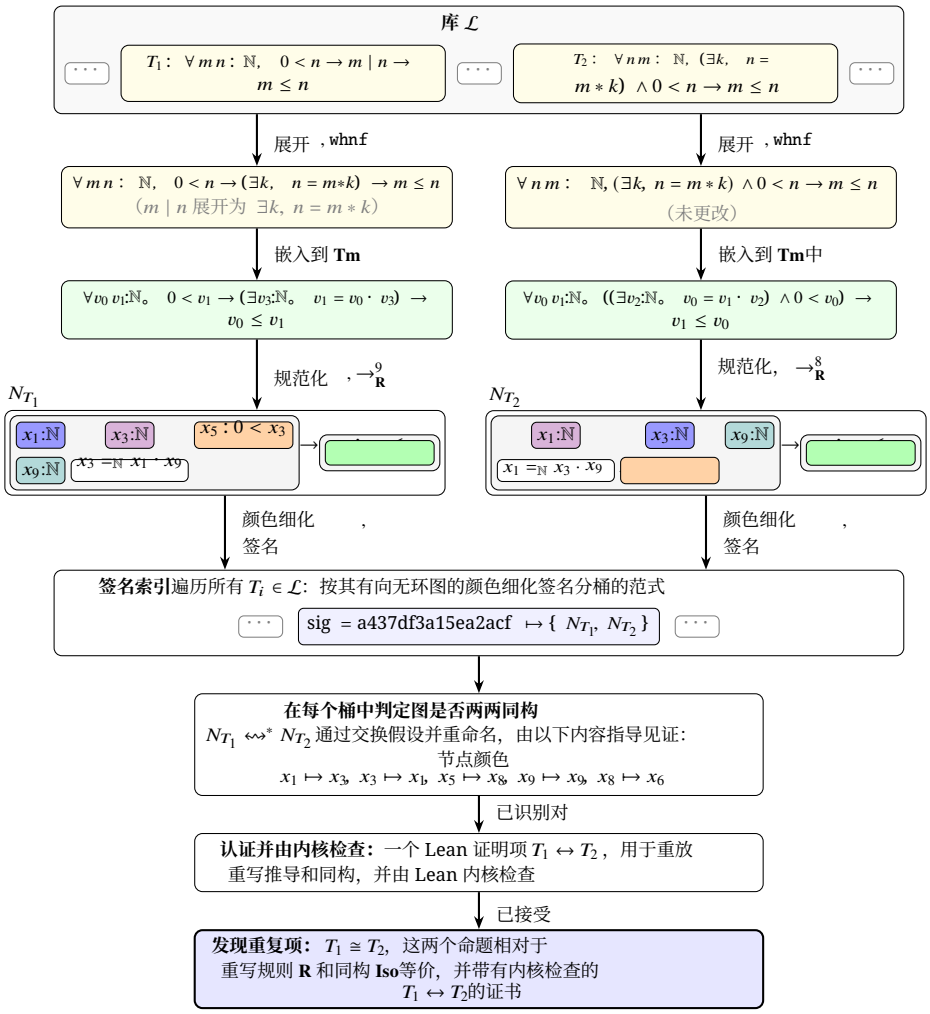


图 5。 在运行示例中检测库 L 中的重复项。每个定理都会展开其可约定义（此处只有 $m \mid n$），并由 Lean 将其转换为弱头范式，嵌入 **Tm**，由 **R** 进行规范化（此处将各个重写步骤合并显示），然后根据其范式对应的有向无环图的颜色细化签名插入索引。 N_{T_1} 和 N_{T_2} 中节点的填充色表示细化后的颜色类；相同的填充色表示相同的颜色值，方程节点不带颜色。只有同一桶中的定理才会进行比较，比较时会搜索模绑定符置换的图同构。 T_1 和 T_2 共享一个桶且彼此同构。对于已识别对，系统会根据两个重写推导和该同构构造一个 Lean 证明 $T_1 \leftrightarrow T_2$，并由 Lean 内核进行检查；只有通过检查后，才会将该对报告为重复项。

因此，最终的签名不依赖成员的顺序。在每个桶内，我们以节点哈希作为指引来计算同构。相同的签名和两两不同的成员哈希，使寻找见证的过程成为一系列无需回溯的强制步骤。在我们的数据中，这种启发式方法在所有测试案例中均有效。

对于桶中每个成员 B（其代表元为 A），工具都会构造一个类型为 $A \leftrightarrow B$ 的项，以证明这两个陈述确实等价。该证书由一系列针对任意命题和排序的通用等价引理、Iff.refl、Iff.trans 和 Iff.symm、逻辑常量，以及中间图的回读组成。它不引用这两个定理中的任何一个。由于这两个陈述都是定理，$A \leftrightarrow B$ 可由它们的证明直接证明。该证书通过不使用这两个证明来避免这种情况，因此即使 A 和 B 尚未证明，它也能通过类型检查；它见证的是二者的结构等价，而非二者都成立这一事实。

这种关系比逻辑等价严格更细，仅由上述模式和定义展开生成，不包含其他内容。例如，它不识别 $\forall$ 和 $\to$ 对 $\wedge$ 的分配律、幂等律、德摩根律、将 $\leftrightarrow$ 表示为两个蕴含、单点规则、对 Function.Injective 等普通定义的展开、宇宙参数的置换，以及 HEq 的对称性。因此，否定判定仅表示 **R** 和 **Iso** 无法识别这两种编码，并不意味着这些陈述不等价。

4 评估

为了评估我们的工具，我们在一个包含 286,264 条定理陈述的语料库上运行了该工具，这些陈述来自 Lean core、Batteries 和 Mathlib，其中包括已弃用的别名和生成的 API 变体[4]。在我们的硬件上，工具运行耗时约半小时，其中大部分时间用于证书构造和内核检查。代码检查器比较和报告生成又耗时 15 分钟。

4.1 重复类与证书

表 1 总结了此次运行结果。在提取的 286,264 条陈述中，286,224 条在配置的大小预算内完成了编码和规范化，另有 40 条从未进行比较。这衡量的是所选语料库的处理覆盖率，与固定模式未涵盖的等价关系无关。规范化后的陈述分为 7,525 个至少包含两个成员的重复类，共有 15,834 个成员。将每个非代表成员与其所属类的代表元配对，共得到 8,309 对，每对都有各自经内核检查的证书。在这 8,309 个成员中，根据 Meta.isDefEq，其中 6,293 个在定义上与其代表元相等，因此 76% 的证书对应的配对已被 Lean 接受为定义上相等；另有 2,016 个成员并非如此。

并非每一对都是冗余证明。我们区分由内核认证的陈述对，以及分析脚本对证明独立性和可能维护价值所作的标签。相关发现是冗余的独立证明，按类统计，

[4]具体而言，使用的版本为 Lean v4.35.0-rc2，源码修订版为 23c81b5，Batteries 版本为 ed9b316，Mathlib 版本为 0653561。我们使用了一台通用虚拟专用服务器，配备 8 个虚拟 CPU 和 32 GB 内存。

表 1. 工具在 Lean core、Batteries 和固定版本的 Mathlib 上的一次运行。所有数值均由该次运行的分析脚本根据其输出计算得出，并可使用所提供的制品复现。定义相等性依据 Meta.isDefEq 在默认透明度下对代表元与成员进行宇宙参数位置重命名后的判定。

提取的定理陈述（非私有）	286,264
在大小预算内编码并规范化	286,224
超出预算，从未进行比较	40
重复类（至少包含两个成员）	7,525
这些类中的成员	15,834
（代表元，成员）对，每对各有一个内核检查的证书	8,309
与代表元定义上相等的成员	6,293（76%）
与代表元定义上不相等的成员	2,016
头部对（代表元，定义等价组的头部）	1,912
双方各自都带有证明的配对	7,612
且双方都不是由对方生成的	3,003
且两个证明都不引用对方	2,051
且两侧并未通过其他成员连接	1,896
且两侧均未被弃用（独立对）	1,742
冗余证明（每类的独立证明 − 1）	1,747 个，分布于 1,687 个类中
对于定义上相等的陈述 / 仅通过 **Iso** 相等	965 / 782
未标记为刻意设计的 API 双胞胎	844 个类中有 873 个
定义上相等 / 仅在同构意义下相等	586 / 287
额外发现的结果，其中一部分已弃用	109
相关发现，包括已弃用的结果	1,856
横跨多个软件包	涉及 162 个类中的 192 项
Mathlib 重复声明检查器未找到配对项	1,184 个类中的 1,226 项
Mathlib 的代码检查器发现 / 排除 / 漏掉的独立对	481 / 241 / 1,020

模同构的陈述。如果类成员引用的有向图有 k 个证明根，则该类贡献 $k-1$ 项相关发现；证明根自身有证明，不会到达其他成员，不是由其他成员生成的，并且未被弃用。通过别名或 to_additive 从其他成员生成的证明，或通过同一命令与其他成员一同生成的证明，均不计入。如果某个陈述也由第三个声明通过宏或属性生成，那么手写证明仍符合条件。

按类统计，共有 1,747 个冗余证明，分布在 1,687 个类中；其中 965 个对应定义上相等的陈述，782 个对应只有通过同构才相等的陈述。

自动分类将 1,747 个冗余证明中的 874 个标记为刻意设计的 API 双胞胎，因此有 873 个候选项分布在 844 个类中，属于这一类别之外。这些标签有助于确定检查优先级，但并不能证明其余声明是无意引入的，也不能证明它们应当被移除。

4.2 示例发现

下面这一对是同一文件中两个定理的方向孪生；将 = 两侧翻转后，它们相等。两者都是有意保留的，但由于第二个陈述没有使用第一个定理的证明，因此被标记出来。可以用 map_zipIdx.symm 替换其独立证明，以消除重复并表明两个定理之间的联系。

```
-- Init/Data/List/Nat/Range.lean 定理 map_zipIdx {f : α → β} {l : List
 α} {k : Nat} : map (Prod.map f id)(zipIdx l k) = zipIdx (l.map f) k :=
by induction l generalizing k <;>simp_all

-- Init/Data/List/Nat/Range.lean，往下 31 行 定理 zipIdx_map {l : List α} {
k : Nat} {f : α → β} : zipIdx (l.map f) k =(zipIdx l k).map (Prod.map f id) :
= by induction l with|nil => rfl| cons hd tl IH =>rw [map_cons,
zipIdx_cons', zipIdx_cons', map_cons, map_map, IH, map_map]rfl
```

这是 Mathlib 两个文件中跨文件的一对定理，在重新结合存在量词与合取并重新排列假设后得到，具体如下：

```
-- Mathlib/代数/余极限/张量积.lean 定理
Submodule.FG.exists_rTensor_fg_inclusion_eq {N : R 模 P}(hN :
N.FG) {x y : N ⊗[R] M} (等式 : N.subtype.rTensor M x =
N.subtype.rTensor M y) : ∃N', N'.FG ∧ h : N ≤ N', (N.inclusion
h).rTensor M x = (N.inclusion h).rTensor M y : 由此可证...
```

```
-- Mathlib/环论/张量积/DirectLimitFG.lean 定理
TensorProduct.eq_of_fg_of_subtype_eq {t' : P ⊗[R] N} (h : r
Tensor N P.subtype t = rTensor N P.subtype t') : ∃(Q : R 模 M)
(hPQ : P ⩽ Q), Q.FG ∧ rTensor N (inclusion hPQ) t = rTensor N
(inclusion hPQ) t' : 由此可证...
```

这两个陈述都表达了这样一点：若两个张量在整体张量积中的像相等，那么它们在某个更大的有限生成子模中的像也相等。它们的证明以不同形式呈现了同一个直极限论证，因此这对定理可能适合在不同文件间复用证明。这一例子说明，比较时不仅要考虑绑定子顺序，还必须考虑依赖关系，因为存在量词包含证明引用了所选子模。证书为这些陈述提供了结构上的联系；至于是否共享它们的证明，则需要另行审查该库的 API 和导入依赖。

分类标签根据源文本、证明项和 Lean 环境，按照规则自动分配。这些标签尚未经过人工核查，因此所报告候选项的维护价值仍有待评估。对发现结果的非正式检查表明，在未被标记为有意设计的 API 孪生的 873 个候选项中，有相当一部分提供了通过证明复用和更清晰地呈现声明之间联系来提升代码质量的机会。检查还发现，绑定子顺序存在差异，也存在使用自动隐式绑定符（Mathlib 之外，因为 Mathlib 不使用它们）以及单独声明变量的情况。这些发现还揭示了定理命名约定不一致的问题。

4.3 与 Mathlib 重复声明检查器的比较

Mathlib 的 DuplicateDecls 是一种语法工具：它会在抹去宇宙层级和绑定符信息，并对顶层绑定符排序后，按照相同的键对定理类型进行分组。它会忽略绑定符名称来比较所得的项，但不会展开定义、重新定向联结词或应用逻辑变换。我们将 Mathlib 的 DuplicateDecls 比较与筛选规则应用于同一环境。在我们的分析报告的 1,742 个独立对中，代码检查器找出 481 对（27.6 %），排除 241 对（13.8 %），漏掉 1,020 对（58.6 %）。这里的比较统计的是代表元–成员对，而上文的 1,747 个冗余证明是按每个类中的证明根计数的。

对于被排除的一对，其中至少有一个声明被筛掉：160 对仅由元编程名称筛选器排除，78 对仅由别名筛选器排除，另有 3 对被两者同时排除。元编程筛选器会跳过 Lean.Grind、Lean.Omega 和 Mathlib.Tactic 等命名空间。别名筛选器会跳过仅由一个常量应用于定理假设而构成的证明，这也会排除真正的一行重证明。在这 241 对中，有 118 对在禁用筛选器时比较键会相同。这种筛选策略合理地将报告重点放在数学声明上，并避免列出常规别名，因此我们将有意排除的项与键未匹配的情况区分开来，并报告重叠情况，而非总体召回率。

在保留但被代码检查器漏掉的 1,020 对中，527 对被诊断为方向变化，316 对尽管键不同却定义上相等。其余

177 对涉及绑定子顺序、展开、柯里化、注释或其他重写。这些诊断取决于键放宽的顺序，而非对每个原因的唯一判定。

反过来，我们的工具将代码检查器的 782 个定理组中的 646 个识别为完全位于单个重复类中。在这些组内的 809 对定理中，有 142 对未被我们的工具识别，其中 119 对不在我们的语料库中，另有 23 对在宇宙层级上存在差异；代码检查器会擦除宇宙层级，而我们的工具会保留它们。

5 相关工作

类型同构与定理搜索。 将类型用作模同构的库搜索键可以追溯到 Rittri [19]。SearchIsos 将这一方法应用于 Rocq 中的依赖类型 [10], ，而 Loogle 支持 Lean 中基于模式的定理搜索 [7]。Takimoto 等人比较了这些工具以及 Rocq、Idris 和 Lambdapi 的搜索能力，并为依赖积与依赖和的模同构合一提出了一种半算法 [21]。他们的方法将参数重排和柯里化/反柯里化与元变量实例化相结合，使查询能够匹配更一般的陈述或带有额外前提的陈述。我们的工作处理的是不同的任务：在整个库中，依据固定的结构等价对现有定理陈述进行分组，而不进行基于元变量的泛化。我们将他们的重写模式改编为带注释的图表示，并加入箭头到 Unit 的律，以及命题联结词和等式的显式对称性。

Lean 中的重复项检测。Mathlib 的 DuplicateDecls 是现有工具中用途最接近的一款。它通过对顶层绑定符排序并擦除绑定符信息和宇宙层级，对声明类型进行语法比较，同时过滤别名，以避免报告有意保留的重复项。我们的方法将比较扩展到命题内部的结构变换，包括柯里化与反柯里化、合取与存在量词的分解，以及联结词的对称性。依赖图在保留绑定符之间引用关系的同时，允许在重命名和置换意义下比较范式。我们还会选择性地在命题位置展开定义。每个报告的匹配项都附有一个内核检查的等价性证明，该证明由重写见证和同构见证构造而成，且不引用任何一个定理。

基础与范围。 我们的方法建立在类型化 λ演算中可证明的类型同构 [8] 及其向依赖积 [20]扩展的相关研究之上。采用模等价重写遵循经典的模重写框架 [18]。我们的机械化结果证明了所选图规则在**Iso** 下的终止性和合流性，并为**Iso**提供了判定过程。这些结果针对图表示：它们并未确立一般类型同构理论或逻辑等价的完备性。与 Lean 命题的联系，则通过每个所报告配对的等价性证书分别进行验证。

6 结论

我们介绍了用于通过结构重写和保持依赖关系的绑定符置换检测重复定理陈述的工具。其重写系统和图同构判定过程拥有机器验证的元理论，并且每个所报告的配对都附有内核验证的等价性证书。对286,264条定理陈述的评估展示了库级别的检测能力，并发现了873个维护候选项；我们的标签表明，值得检查这些候选项，以寻找提升代码质量的机会。这些发现为审查证明复用以及明确声明之间的联系提供了具体机会。证书确立了陈述等价性，但决定应共享哪些证明或保留哪些声明，仍需要谨慎的人工审查。

下一步可以将经认证的重复项报告整合到库的维护工作流程中，例如 Mathlib 的技术债务跟踪，并与维护者更详细地评估由此产生的证明重构。扩展该项目的另一个方向，是完善所支持的模式和证书构造，使其适用范围从命题扩展到一般的类型同构。此外，除了等价之外，若能识别特化或蕴含也会很有意义。例如，如果等价 $A \leftrightarrow B$ 已得到证明，那么 $A \to B$ 的证明也可能是多余的。

数据可用性声明。 本文的制品——包含该工具及完整评估复现内容的 Docker 镜像——可在 Zenodo [15] 上获取。

致谢。 本研究部分由 Coefficient Giving 资助。Simon Henniger 获得了哈佛大学约翰·A·保尔森工程与应用科学学院（SEAS）奖学金以及德国学术奖学金组织的资助，后者由德国联邦经济事务与能源部提供资金。

参考文献

1. Anthropic：Fermats-last-theorem：Lean 4 中的费马大定理。https://github.com/anthropics/fermats-last-theorem（2026）2. Anthropic：在 Lean 中形式化费马大定理：Claude 推理过程的时间线及精选摘录。技术报告，Anthropic（2026 年 9 月）3. Baanen, A., Ballard, M.R., Commelin, J., Chen, B.G., Rothgang, M., Testa, D.：Mathlib 的发展：大型数学库的维护（2025 年 8 月）。https://doi.org/10.48550/arXiv.2508.21593 4. Barrett, C., Chaudhuri, S., Montesi, F., Grundy, J., Kohli, P., de Moura, L., Rademaker, A., Yingchareonthawornchai, S.：CSLib：Lean 计算机科学库（2026 年 2 月）。https://doi.org/10 04846 5 48550 2602 . /arXiv. . . Barthe, G., Pons, O.：依赖类型论中的类型同构与证明复用。载于：Honsell, F., Miculan, M.（编）《软件科学与计算结构基础》。第 57–71 页。施普林格，柏林，海德堡（2001）。https://doi.org/10.1007/3-540-45315-6_46. Basin, D.A.：一个等价于图同构的项等式问题。《信息处理快报》**51**(2)，61–66（1994 年 7 月）。https://doi.org/10.1016/0020-0190(94)00084-0 7. Breitner, J.：Loogle。https://loogle.lean-lang.org/（2023）

8. Bruce, K.B., Di Cosmo, R., Longo, G.：类型的可证明同构。《计算机科学中的数学结构》**2**(2)，231–247（1992 年 6 月）。https://doi.org/10.1017/ S0960129500001444 9. de Bruijn, N.G.：类型化 λ 演算中的望远镜映射。《信息与计算》**91**(2)，189–204（1991 年 4 月）。https://doi.org/10.1016/0890-5401(91)90066-B 10. Delahaye, D.：在 Coq 证明库中使用类型同构进行信息检索。载于：Coquand, T., Dybjer, P., Nordström, B., Smith, J.（编）《证明与程序的类型》。第 131 147 2000 10 1007 3 540 44557 9_8 – 页。施普林格，柏林，海德堡（）。https://doi.org/ . / - - - . Di Cosmo, R.：类型同构简述。《计算机科学中的数学结构》**15**0960129505004871 12 5 825 838 2005 10 1017 ( )，–（10 月）。https://doi.org/ . /S . van Doorn, F., Ebner, G., Lewis, R.Y.：形式数学库的维护。载于：Benzmüller, C., Miller, B.（编）《智能计算数学》（CICM 2020）。《计算机科学讲义》，第 12236 卷，第 251–267 页。施普林格，Cham（2020）。https://doi. org/ 10 1007 978 3 030 53518 6_16 . / - - - - . Fiore, M., Di Cosmo, R., Balat, V.：关于带空类型和和类型的类型化 λ 演算中的同构的几点说明。载于：第 17 届 IEEE 计算机科学中的逻辑年度研讨会论文集。第 147–156 页。IEEE 计算机学会，哥本哈根，丹麦（2002）。https: //doi.org/10 1029824 14 1109 2002 . /LICS. . . Lean FRO：比较器。https://github.com/leanprover/comparator（2025）15. Leßmann, S., Henniger, S., Poesia, G., Amin, N.：lean-piso-trip：模重写与绑定符置换的 Mathlib 定理等价性搜索（VMCAI 2027 工件）（2026）。https://doi.org/10 23064352 16 5281. /zenodo. . OpenAI：NavierStokesAndEuler：随纳维–斯托克斯和欧拉结果附带的 Lean 证书。https://github.com/openai/NavierStokesAndEuler（2026 年 9 月）17. OpenAI：关于纳维–斯托克斯千禧年大奖难题。https://openai.com/index/ navier-stokes-solution/（2026 年 9 月）18. Peterson, G.E., Stickel, M.E.：若干等式理论的完备归约集。《美国计算机学会会刊》（JACM）**28**(2)，233–264（1981 年 4 月）。https://doi.org/10.1145/322248. 322251 19. Rittri, M.：在函数库中将类型用作搜索键。载于：第四届函数式编程语言与计算机体系结构国际会议论文集。第 174–183 页。FPCA ’89，美国计算机协会，纽约，纽约州，美国（1989 年 11 月 99384 20 10 1145 99370）。https://doi.org/ . / . . Soloviev, S.：类型化逻辑框架中依赖积的同构。载于：Her- belin, H., Letouzey, P., Sozeau, M.（编）第 20 届证明与程序的类型国际会议（TYPES 2014）。莱布尼茨国际信息学论文集（LIPIcs），第 39 卷，第 274–287 页。达格施图尔城堡－莱布尼茨信息学中心，德国达格施图尔（2015 274 21 10 4230 2014）。https://doi.org/ . /LIPIcs.TYPES. . . Takimoto, S., Moriguchi, S., Watanabe, T.：用于基于类型的库搜索的依赖类型间模同构合一。载于：第 10 届 ACM SIGPLAN 类型驱动开发国际研讨会论文集。第 15–25 页。TyDe ’25，美国计算机协会，纽约，纽约州，美国（2025 年 10 月）。https://doi.org/10.1145/3759538. 3759651 22. Weisfeiler, B.Y., Leman, A.A.：将图归约为规范形式及其中出现的代数。《科学技术信息》**2**(9)，12–16（1968）