arXiv:2603.06849v1 [cs.LO] 6 Mar 2026

Twitch：学习用于等式定理证明的抽象

Guy Axelrod[1], [2][0000−0002−1752−8069]、Moa Johansson[1], [2][0000−0002−1097−8278]和 NicholasSmallbone[1], [2][0000−0003−2880−6121]

[1] 瑞典哥德堡查尔姆斯理工大学[2] 瑞典哥德堡大学
{guya,jomoa,nicsma}@chalmers.se

摘要。 自动推理中的若干成功策略依赖于人类提供的指导，以确定哪些项或子句形状值得关注。本文旨在自动发现有趣的项形状。具体而言，我们发现抽象：即在相关证明中反复出现的项模式。我们介绍工具*Twitch*，它借助 Stitch 发现抽象。Stitch 最初是为在程序合成任务中发现可复用的库函数而开发的工具。Twitch 可以通过两种方式生成抽象：（1）从一个猜想的部分失败证明中生成；（2）从同一领域中其他定理的成功证明中生成。我们还扩展了等式定理证明器 Twee，使其能够使用这些抽象。我们在 TPTP 中的一组单元等式（UEQ）问题上评估了 Twitch，结果表明，它能够证明 12 个评级为 1 的问题，并显著加快了许多其他问题的求解速度。

关键词： 自动定理证明 · 等式定理证明 · 完备化· 抽象学习 · 证明引导搜索

1 引言

自动定理证明器必须在庞大的搜索空间中导航，选择有用的推理并舍弃无用的推理。例如，Twee[17] 中的一个证明可能需要考虑数亿个临界对，从中选出数万个作为活跃的重写规则，而最终证明中实际用到的或许只有几百个。因此，识别哪些临界对值得关注至关重要——但 Twee 为此采用的启发式方法过于简单，主要依据临界对的大小。要做到更智能并不容易，因为可能的临界对数量如此之多，每一个都必须被迅速判定。

这种情况在基于饱和的定理证明器中很常见：可能的推理非常多，并使用简单的度量进行排序。这种方法相当有效，但有时还需要更多智能。一种由 Otter [12] 推广的方法是，注入人工指导，指出哪些项或子句的形状 值得关注。用户指定一组值得关注的形状，生成的子句中符合这些形状的，会优先于不符合的子句。这个想法最初源自加权

策略 [14]，后来由共振策略 [23,3,8]和提示策略 [19,20]进一步改进。这三种策略都曾用于寻找难以找到的证明。

本文自动发现有趣的项形状。我们分析失败的证明尝试以及相关猜想的证明，以找出频繁出现的项形状，并将其称为抽象。我们已为 Twee 添加了对抽象的支持，借助自动生成的抽象，我们能够证明 TPTP [18]，中 12 个评级为 1 的问题，以及 Twee 先前无法解决的若干其他问题（见第 5 节）。我们的贡献如下：

– 一种从证明猜想的不成功尝试 中自动生成抽象的方法（见第 3.1 节）。其思路是从证明尝试中挑选有趣的引理，并找出这些引理的证明中反复出现的项模式。为找到这些模式，我们使用 Stitch [5]，它通过引入作为缩写的辅助函数定义，尝试压缩 一组项（此处为证明中的项）。– 一种使用 Stitch 从同一领域的成功证明语料库中生成抽象的方法（见第 3.2 节）。我们的假设是，在证明中反复出现的项模式，往往是同一领域其他猜想的良好抽象。这里的目标是创建一种受课程学习 [4] 启发的方法，用于证明困难的等式问题：先证明同一领域中的一些较简单问题，提炼这些证明中出现的共同模式，再用这些抽象来引导对困难问题的证明。– 在等式定理证明器 Twee[17]，中实现抽象，其设计受到加权策略 [14] 的启发（见第 4 节）。

Twitch 的代码以及所有数据和实验结果均在相关仓库 [1] 中提供。

1.1 动机示例

让我们考虑 TPTP 中的这个问题 `LAT075-1`：它涉及在一种仅含一个函数符号 f 的语言中，对模正交格进行等式公理化；该符号表示谢弗竖线（或与非运算符）。这里不必关注这个问题的细节，重要的是，在 Twee 找到的证明中，同一种项反复出现。以下是证明中的一个简短片段：

$$
\begin{aligned}
&\vdots \\
&= f(x, f(y, \boxed{f(f(z, \boxed{f(x,x)}), f(z, \boxed{f(x,x)}))})) \\
&= f(x, f(y, f(f(z, \boxed{f(x,x)}), f(\boxed{f(x,x)}, z)))) \\
&= f(x, f(y, \boxed{f(f(\boxed{f(x,x)}, z), f(\boxed{f(x,x)}, z))})) \\
\end{aligned}
$$

$$= f(\boxed{f(\boxed{f(x,\ x)},\boxed{f(x,x)})},\ f(y,\ \boxed{f(f(\boxed{f(x,x)}\ z)\ f(\boxed{f(x,x)}\ z))}))$$

$$\vdots$$

我们可以看到，形如 $f(t,\ t)$ 的项（我们已在上方用方框标出）在证明中反复出现。[3] 我们猜想，相同的项形态很可能也常见于其他关于谢弗竖线的定理证明中。

这为我们提供了证明一个困难定理的基本方法。首先，我们分析一个相关定理的证明，找出其中常见的项模式。然后，假设这些项模式也会出现在困难定理的证明中，我们便要求定理证明器优先采用涉及这些项模式的推理。剩下的问题是：如何找到这些项模式，以及定理证明器如何利用它们?

为了形式化证明中常见模式的概念，我们设想通过引入新定义来缩写某些项，从而压缩证明。例如，若定义 $g(x) := f(x,\ x)$，那么上面的证明就可以写得更紧凑；例如，第一个项可以写成 $f(x,\ f(y,\ g(f(z,\ g(x)))))$。然后我们要问：怎样的一组缩写能够实现最大程度的压缩，也就是说，怎样的一组定义能让我们尽可能紧凑地表示证明项?

工具 Stitch [5] （见第 2 节）正是为了解答这个问题而设计的。给定一组项，它会搜索压缩效果最大的函数定义。下文中，我们将这类函数定义称为抽象。在本例中，Stitch 会发现 $g(x) := f(x,x)$ 是压缩效果最大的抽象（而在获得完整证明后，它还会找到其他有用的抽象，例如与运算符 $h(x,y) := f(f(x,y)\ f(x,y))$ 。

有了这些抽象后，我们如何利用它们来影响证明搜索? 一种做法是直接将函数定义添加为公理，在本例中即 $\forall x.\, g(x) = f(x,x)$。这样，定理证明器就可以将 $f(t,t)$ 形式的项重写为 $g(t)$。由于后者更小、代价更低，因此如果这类项出现在临界对中，该临界对就更有可能被选中。对于 `LAT075-1` ，这种方法效果很好，并将 Twee 的运行时间从$\sim 250s$ 缩短至 $\sim 10s$。

添加用于缩写常见项模式的定义性公理是一种已知且通常效果良好的技术。[4] 然而，添加新公理也会允许它们之间产生新的推理，从而扩大搜索空间。正如我们在第5.3节所示，添加超过少数几条此类公理会损害定理证明器的性能。

相反，我们扩展了 Twee，使其原生支持抽象（见第 4 节）。这些抽象会影响定理证明器的启发式函数，使与之匹配的推理更有可能被选中，但不会改变搜索空间本身。因此，我们可以提供更多抽象，但每个抽象的影响可能较小。在这种情况下，将 $f(x,\ x)$ 作为抽象传入，可将 Twee 的运行时间从 $\sim 250s$ 缩短至 $\sim 130s$。

[3] 这或许并不那么令人惊讶，因为 f（t,t）表示 t的否定。[4] 例如，参见TPTP中的 Robbins 问题变体 `ROB033-1` 。它也是Twee的目标方向启发式[17]的基础。

2 背景：Stitch

Stitch [5] 最初是为程序合成中的库学习而设计的系统，旨在从现有程序集合中自动发现可复用的函数抽象。给定一组项作为输入，它会构造能够概括这些项部分内容的函数，并用这些函数来简写这些项。Stitch 尝试构造压缩效果最大的抽象；具体而言，它最大化函数的大小与该函数可使用位置数量的乘积。

尽管 Stitch 的设计目标是处理程序代码，但它接受任意项集合作为输入。因此，给定一个 Twee 证明，我们可以将证明中出现的所有项组成的集合交给 Stitch，它便会为这些项找出抽象。

Stitch 的输入语言是一种高阶 λ 演算。因此，我们需要将证明项转换为 λ 项；具体做法是将函数符号视为常量，并对所有自由变量进行 λ 抽象。例如，我们示例中的项 $f(x,\ f(y,\ f(f(z, f(x, x))\ \ f(z, f(x, x)))))$ 会被转换为 $\lambda x.\lambda y.\lambda z.(f\ x\ (f\ y\ (f\ (f\ z\ (f\ x\ x))\ (f\ z\ (f\ x\ x)))))$. 在此情况下，Stitch 将返回抽象 $g(\alpha) := (f\ \alpha\ \alpha)$，其中 α 是一个模式参数，表示任意子项，可用于将输入项压缩为 $\lambda x.\lambda y.\lambda z.(f\ x\ (f\ y\ g(f\ z g(x))))$. 由于 g 的主体具有一阶形式，因此可以直接将其转换回来，从而得到抽象 $f(x, x)$。

一般来说，Stitch 抽象是任意的 λ-项，也可以是高阶的。在实践中，由于输入项是一阶的，Stitch 在我们的设置中很少生成高阶抽象。如果确实生成了，我们就会丢弃该抽象。

3 Twee + Stitch = Twitch

本节介绍 Twitch 的实现：这是一个用于自动发现可复用抽象的系统，这些抽象可用于帮助解决给定领域中的困难问题。Twitch 有两种使用模式：可以从部分证明尝试中发现抽象，如第 3.1 节所述；也可以从同一领域中较简单的证明中发现抽象，如第 3.2 节所述。我们将第一种模式生成的结果称为部分证明抽象，第二种模式生成的结果称为领域抽象。图 1 展示了领域抽象的推导过程概览。Twitch 找到的抽象用于调整 Twee 定理证明器中的搜索启发式，如第 4 节所述。

下文中，我们用 P 表示给定的输入问题——它由一组方程（公理）以及一个待证明的目标/猜想方程组成。

3.1 基于部分证明的抽象

我们探索的第一种方法是：给定一个单独的难题 P_h作为输入，发现其中的抽象；当我们没有相关问题的示例可供学习时，就可以使用这种方法。我们首先在 P_h 上运行 Twee（不使用抽象），运行固定时长。超时后，尽管它尚未证明目标，但它会

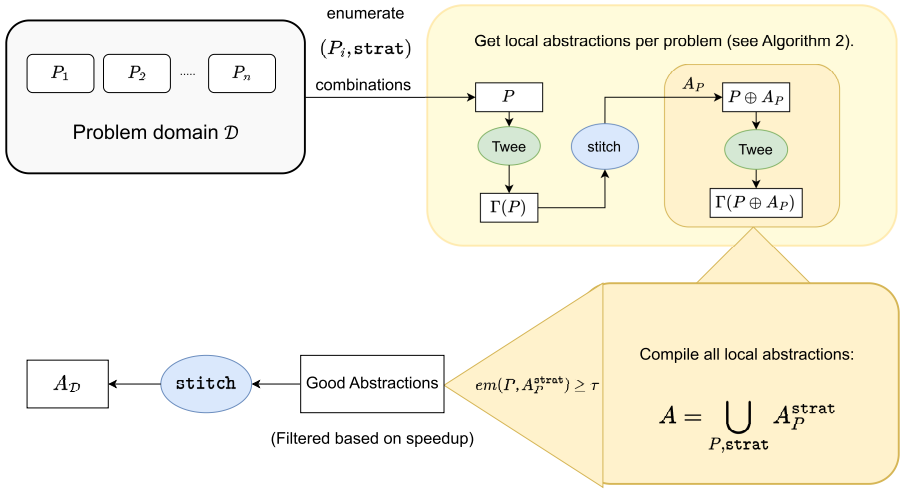


图1：通过Twitch进行领域抽象生成的概览（算法1及其子程序算法2）。

推导出大量重写规则。每条推导规则 $l \to r$都对应一个关于我们正在探索的理论的方程 $l = r$ 。其中许多方程会相当乏味，但有些会代表有趣的引理。我们的想法是只从最有趣的引理中学习抽象。

为了定义有趣性，我们首先需要一些辅助定义。对于任意引理 $l = r$，我们用 $T(l = r)$表示出现在其证明中的项的集合，包括该证明中所使用的任何子引理的证明。更精确地说， $l = r$ 的证明是一条链 $l = t_1 = \ldots = t_n = r$，其中每一步都可能由拥有自身证明的引理 $l_i = r_i$ 来论证。然后我们定义$T(l = r) = \{l, t_1, \ldots, t_n, r\} \cup \bigcup_i T(l_i = r_i)$。

对于项 t，我们将其权重 $|t|$ 定义为 t中函数应用的数量plus唯一变量的数量。我们将此扩展到方程，通过定义 $|l = r| = |l| + |r|$，并扩展到项的集合 T，通过定义$|T| = \sum_{t \in T} |t|$。然后我们将引理的“有趣性” $s(l = r)$定义如下：

$$s(l = r) := \frac{|T(l = r)|}{|l = r|^2}.$$

直观上，得分较高的引理通常是那些陈述简单、但需要较长证明的引理。当然，这不一定符合人类对有趣或有用内容的判断，但我们将其视为衡量这些概念的一种粗略启发式方法。

给定 P_h 的某个部分证明，我们按 s 降序排列发现的引理。然后提取前 k 个引理 $l_1 = r_1$、……，$l_k = r_k$。我们所说的 部分证明抽象 ，就是通过在 $\bigcup_{1 \leq i \leq k} T(l_i = r_i)$ 上运行 Stitch 得到的那些抽象。然后，我们可以在加入这些抽象的 P_h 上重新运行 Twee。

3.2 领域抽象

这种方法的目标是学习有助于在特定领域 $\mathcal{D}$ 中证明问题的抽象。我们需要一组问题，其中一些已经得到证明（“简单”问题），另一些尚未得到证明（“困难”问题）。算法如下：

1. 首先，我们在简单问题的证明上使用 Stitch，为每个问题找到抽象。这些抽象是局部的，即针对单个问题。2. 然后，我们为每个问题添加其局部抽象，并重新运行 Twee，以实证检验这些抽象是否良好，即能否让我们更快地重新证明该问题。3. 最后，我们构建一组包含所有良好局部抽象的集合，并用 Stitch 对其进行压缩，以发现共性并推导出更高层次的领域抽象。

这一过程如图 1 所示，伪代码见算法1。这里，我们用 $T(\Gamma(P))$ 表示 Twee 为问题 P 找到的证明 $\Gamma(P)$ 中出现的项集合。对 $T(\Gamma(P))$ 运行 Stitch 所得的抽象集合（其中 $\Gamma(P)$ 是使用特定的 Twee策略 `strat` 找到的）记为$A_P^{\texttt{strat}}$。我们用 $P \oplus A$ 表示添加了抽象 A 的 P。最后，使用效果度量 em 筛选局部抽象，该度量衡量通过添加该抽象增强问题后所获得的加速。更具体地说，

$$em(P,\ A) = \frac{t(\Gamma(P))}{t(\Gamma(P \oplus A))},$$

其中 $t(\Gamma(P))$ 表示 Twee 找到证明 $\Gamma(P)$ 所需的时间。只有那些带来的加速超过某个阈值 τ 的抽象才会被加入良好抽象集合。

我们看到，扩大良好抽象集合依赖于针对一组不同的策略运行 Local-Abstractions 子程序。这只是指为影响 Twee 运作方式的参数分配特定值。具体而言，策略参数包括：(1) 是否运行 Twee

算法1

过程领域抽象($\mathcal{D}$)
    good_absts ← 空集
    对于所有 $P \in \mathcal{D}$ 和 strat ∈ strategies 的组合**执行**
        $A_P^{\texttt{strat}}, em(P, A_P^{\texttt{strat}})$ ← 局部抽象(P, strat)
        如果 $em(P, A_P^{\texttt{strat}}) \geq \tau$ **则**
            将 α-规范化的$A_P^{\texttt{strat}}$添加到 good_absts
        结束 if
    结束 for
    $\mathcal{D}$ ← 在 good_absts 上运行 Stitch
    返回 $\mathcal{D}$
结束过程

算法2

过程 局部抽象(P, strat)
    在 P 上使用 strat运行 Twee。
    如果它未能在时间限制内终止，返回失败。
    否则，我们会得到证明 $\Gamma(P)$。
    将 $\Gamma(P)$ 中的项提取为字符串，得到列表 $T(\Gamma(P))$。
    对 $T(\Gamma(P))$ 运行 Stitch，得到抽象 A_P^{strat}。
    使用 strat 对 $P \oplus A_P^{\text{strat}}$运行 Twee。
    计算 $em(P,\ A_P^{\text{strat}})$。如果 Twee 在 $P \oplus A_P^{\text{strat}}$ 上超时，则将 $em(P,\ A_P^{\text{strat}})$ 设为 $= 0$。
    返回 A_P^{strat}、 $em(P,\ A_P^{\text{strat}})$。
结束过程

以及是否启用目标扁平化；(2) 抽象权重因子（见第 4.1 节）。通过遍历这些参数的各种组合，我们扩大了良好抽象的池，从中可以进一步推导出领域抽象。

然后，我们可以使用这些领域抽象来增强同一领域中的困难问题（见第5.2节）。在实践中，给定一个困难问题 $P_h \in \mathcal{D}$，我们在第二次运行Stitch之前（即算法1中第10行之前）对良好抽象执行过滤步骤。具体而言，我们会移除任何包含 P_h中不存在的函数符号的抽象，或包含元数不匹配的共享函数符号的抽象。这确保所得的领域抽象与 P_h兼容。

4 Twee中的支持抽象

Twee的架构基于不失败完备化 [11,2], ，其工作方式是将输入等式视为重写规则，然后推导出无法通过前向重写证明的新规则。例如，给定一些关于群论的公理，Twee可能会将它们定向为以下规则：

$$e \cdot x \to x \qquad x^{-1} \cdot x \to e \qquad (x \cdot y) \cdot z \to x \cdot (y \cdot z)$$

我们可以观察到，项$(x^{-1} \cdot x) \cdot y$ 具有两个范式：

$$(x^{-1} \cdot x) \cdot y \to e \cdot y \to y \qquad (x^{-1} \cdot x) \cdot y \to x^{-1} \cdot (x \cdot y),$$

因此，方程 $x^{-1} \cdot (x \cdot y) = y$ 为真，但无法通过前向重写证明，因为等式两边都是范式（这称为临界对）。因此，Twee 会添加新的重写规则 $x^{-1} \cdot (x \cdot y) \to y$。这个推导新重写规则的过程会持续进行，直到规则集饱和或目标得到证明。

一个问题是，临界对的数量通常会随着推导规则数量的增加而呈平方级增长：一个典型的证明可能会产生 $\sim 10^5$ 条规则，从而形成$\sim 10^{10}$ 个可能的临界对。因此，我们不能将所有 临界对都转化为重写规则。Twee 转而采用一种启发式方法，在每一步中将“最佳”临界对转化为重写规则。Twee 使用若干判据对临界对进行排序（见 [17]），其中最重要的是方程的权重，定义如下：

$$w(t = u) = w(t) + w(u)$$
$$w(x) = 1, \text{ if } x \text{ is a variable}$$
$$w(F(t_1, \ldots, t_n)) = 1 + \sum_{i=1}^{n} w(t_i), \text{ if } F \text{ is a function symbol}$$

我们对抽象的处理方法受到加权策略 [14]的启发：每当临界对中的某个项或子项与一个抽象匹配时，就降低其计算出的权重。具体来说，我们将 w 扩展，加入以下子句：

$$w(A\sigma) = w_A + \sum_{x \in dom(\sigma)} w(x\sigma), \text{ if } A \text{ is an abstraction and } \sigma \text{ is a substitution}$$

这里， w_A 是表示抽象权重的常数。我们会在第 4.1 节中再讨论这一点，但现在先假设 $w_A = 1$。

例如，假设我们正在研究两个函数 $times$ 和 $plus$。为了促使定理证明器对分配律进行推理，我们可以采用以下抽象 A，它表示一个可以应用分配律的项。

$$plus(times(x, y), times(x, z))$$

现在考虑下面这个方程，它可能是作为临界对出现的：

$$times(c, \underline{plus(times(g(f(a), b), f(x)), times(g(f(a), b), g(y, z)))}) = g(a, b)$$

如果没有抽象 A，这个方程的权重将为 21。使用 A 后，带下划线的项可以匹配为 $A\sigma$，其中 $\sigma = \{x \mapsto g(f(a)\ b)\ \ y \mapsto f(x)\ \ z \mapsto g(y, z)\}$。因此，总权重为 $1 + w(c) + w(A\sigma) + w(g(a, b)) = 2 + w_A + w(g(f(a)\ b)) + w(f(x)) + w(g(y, z)) + 3 = 15$。这意味着，如果该方程作为临界对产生，它就更有可能被选中。

请注意，使用 A 会通过两种方式降低上述方程的代价：(1) 抽象的“骨架” $plus(times(_, _), times(_, _))$ 的权重只计为 1，而不是 3；(2) 尽管 x 在抽象中出现两次， $x\sigma$ 只计一次，因此重复的子项 $g(f(a)\ b)$ 也只计一次。性质（2）对于让抽象发挥缩写的作用至关重要。例如，在第 1.1 节中，我们定义了抽象 $f(x,\ x)$，其中 f 是谢弗竖线，表示 x 的否定。因此，我们希望 $f(t,\ t)$ 的权重与假设项 $not(t)$ 相同，这就需要性质（ 2 ）。

顺便一提，我们还在 Twee 中实现了项级共振器[23] 。它们的工作方式与抽象完全相同，但只有在 σ 中的每个变量都映射到一个变量（而非复合项）时才会匹配。

4.1 抽象的代价

在上面的示例中，我们假设抽象的权重 w_A 为 1 个符号。实际上，我们允许用户在两种计算权重的方法之间进行选择。这两种方法都以用户选择的常量 k 作为参数：

固定权重 每个抽象 A 都会获得一个固定权重 $w_A = k$。
权重因子 这种方法会为较大的抽象赋予更高的权重。给定抽象 A，我们首先计算 A 的骨架 $w(skel(A))$ 的权重。若将所有变量的权重都计为 0（即忽略变量），这就是 A 的权重。在上面的例子中，$w(skel(A)) = 3$。然后我们设置 $w_A = w(skel(A)) \times k$。

5 TPTP 实验

我们在一组来自 TPTP v9.2.1 [18] 的不可满足单元等式（UEQ）问题上评估了 Twitch。这些问题来自九个 TPTP 领域，汇总于表 1。主要目的是衡量 Twitch 发现的抽象是否有助于解决 TPTP 中的问题。

第 5.1 节衡量部分证明抽象和领域抽象是否能缩短 Twee 的运行时间。对于领域抽象，用户必须定义哪些问题构成该领域；在我们的实验中（除第 5.2 节提到的一种情况外），我们简单地将给定 TPTP 领域中的问题集视为该领域（例如群论（GRP）、格论（LAT）等）。

第 5.2 节报告了 Twitch 的抽象使我们能够求解的困难问题。在此，我们将困难问题定义为：其 TPTP 评级至少为 0.9（即 90% 的自动定理证明器无法在 300 秒内求解），且 Twee（不使用抽象）无法在 1000 秒内求解。表 1 中共有 70 个此类问题。

第 5.3 节考察了多种消融实验，用以检验 Twitch 架构与策略的若干方面。

有关所用相关参数值的详细信息，请参见附录 A。

5.1 总体运行时间改进

图 2a 和图 2b 展示了 Twee 在表 1 所列 TPTP 问题上的运行时间仙人掌图。我们对 Twee 进行了基准测试，分别考察其是否采用目标

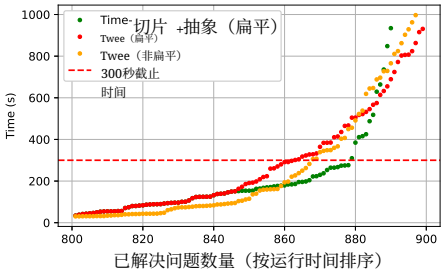
领域	数量
ALG（一般代数）	20
BOO（布尔代数）	56
COL（组合逻辑）	120
GRP（群论）	481
LAT（格论）	126
LCL（逻辑演算）	84
REL（关系代数）	81
RNG（环论）	48
ROB（罗宾斯代数）	25
总计	1041

表 1：TPTP v9.2.1 各领域中不可满足的 UEQ问题

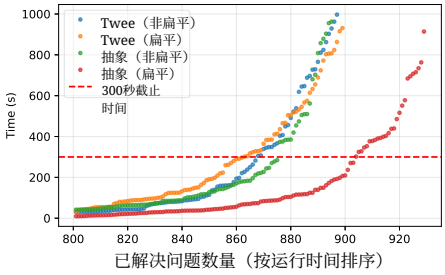
扁平化 策略 [17], ，以了解该策略与抽象的相互作用。在两张图中，“Twee（扁平化）”和“Twee（不扁平化）”曲线显示了 Twee 在不使用抽象的情况下（分别采用和不采用目标扁平化）运行至完整的 1000 秒时间限制的结果。我们在 300 秒处绘制了一条线，这是 ATP 基准测试中常见的超时时间，但评估仍持续到 1000 秒。

图 2a 展示了使用部分证明抽象的结果。我们先运行 Twee（不进行目标扁平化）150 秒；如果超时，则从失败的证明中提取部分证明抽象，并在剩余时间内重新运行 Twee（进行目标扁平化），同时加入这些抽象。

图 2b 展示了使用领域抽象时的结果。与之前一样，“Twee（扁平化）”和“Twee（不扁平化）”展示的是 Twee 未使用抽象时的性能。“抽象（扁平化）”和“抽象（不扁平化）”展示的是加入抽象后 Twee 的性能。与部分证明抽象不同，这里没有进行时间分片。图中未计入针对特定领域学习抽象所花费的时间，因为该时间由该领域中的所有问题共享。



（a）使用部分证明抽象。



（b）使用领域抽象。

图 2：TPTP 问题的 Twee 运行时间仙人掌图。这些图中的每个点都对应表 1 中的一个特定问题 P。点的 y 值表示 Twee 找到 P 的证明所用的时间（可能加入 Twitch 领域抽象），相应的 x 值则是问题按运行时间递增排序后的索引 P。

我们可以看到，仅对问题应用领域抽象只能带来适度的加速。然而，同时应用领域抽象和目标扁平化，则能显著加速求解，300 秒内大约能多解决 25 个问题。仅比较基线 Twee 能在 300 秒内解决的问题时，加入领域抽象和目标扁平化后，运行时间大约缩短了一半。因此，领域抽象似乎与目标扁平化特别契合。

我们可以看到，部分证明抽象确实缩短了运行时间，并且在 300 秒后能解决更多问题，但效果不如领域抽象。不过，在许多情况下，它们仍然很有用，因为无需进行任何预处理即可找到这些抽象。此外，它们还具有

适用于任何问题的优势，即使是那些没有其他证明可供学习的问题。

5.2 求解困难问题

除总体运行时间的改进外，我们发现加入 Twitch 抽象后能够证明 18 个困难问题。已解决的困难问题完整列表见附录 C。我们在此概述结果，并着重介绍一些有趣的案例。

部分证明抽象 给定一个困难问题，我们在其上运行 Twee，最多 500 秒。然后，我们从部分证明中排名靠前的引理生成抽象，并将其提供给 Twee。这使我们解决了 11 个困难问题——每个问题的总时间限制为 1000 秒，其中包括部分证明的运行。其中大多数（11 个中的 6 个）是 LCL 问题，它们包含将霍恩子句转换为等式逻辑的翻译（使用 [7] 中介绍的技术）。在这些情况下，我们发现形式为$ifeq(x, true, y, true)$ 的部分证明抽象总会被发现——即以 $ifeq$ 函数表示的蕴含编码。通过人工测试，我们发现正是这些抽象在帮助解决困难问题方面真正发挥了作用。

领域抽象领域抽象在1000秒内解决了18个困难问题，其中包括部分证明抽象所解决的每一个困难问题。例如，关于谢弗竖线公理化的问题 `LAT075-1` （评分：1.00）在32秒内（未使用目标扁平化）通过提供以下领域抽象得以解决：

$$
\begin{aligned}
&- \;\; f(A, A) - \;\; f(f(B, A)\;\; f(B, A)) - \;\; f(f \\
&(A, B)\;\; f(A, C)) - \;\; f(f(D, C)\;\; f(B, A)) - \\
&\;\; f(f(f(C, f(C, f(f(A, A)\;\; B)))\;\; B)\;\; A)
\end{aligned}
$$

经人工检查，我们发现第一和第三个抽象对于在合理时间内找到证明至关重要。

我们注意到，至少前三个抽象在语义上是有意义的。第一个对应于非运算符，第二个对应于与。谢弗竖线满足一种自分配律（其中与非对与非进行分配以产生或非），而第三个时间表示一个可应用该定律的项。

进一步检查完整结果（见源代码仓库）表明，在少数情况下（确切地说是三个），抽象列表相当长。这些情况凸显了这样一个事实：在某些场景中，跳过对良好抽象进行抽象（过程 1 的第 10 行），而直接将 A_D 取为所有良好抽象的集合，会更有帮助。

我们还注意到，除了这 18 个问题之外，问题 `GRP677-1.p` 也借助领域抽象得到了求解（用时 180 秒）。不过，这需要将领域 $\mathcal{D}$ 限制为仅包含与待求解难题具有非常相似公理的问题。因此，我们看到更大的领域未必更好。这表明，当目标是求解某个特定难题时，对提供给 Twitch 的领域进行筛选，可能是未来工作的一个有用方向。

5.3 抽象与定义性公理

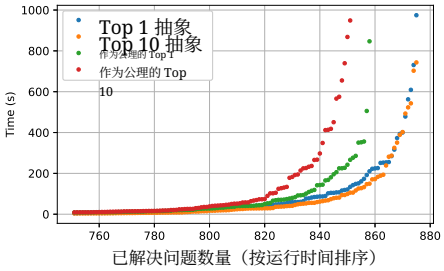
使用 Stitch 抽象来增强问题P 有两种方式：将其作为抽象提供给 Twee，或将其作为定义性公理添加进去（如第 1.1 节所述）。Twitch 采用的是抽象方法。我们在此介绍旨在验证这一选择的实验，通过比较这两种方法来进行评估。实验是在表 1 所列的 TPTP 问题上运行算法2。

图 3a 展示了算法2 第 7 行中每个问题的运行时间仙人掌图。不同颜色对应于使用 A 增强 P 的不同方法。图例中的“前 k”指的是采用 Stitch 在算法2 第 6 行找到的前k 个抽象。 [5]。

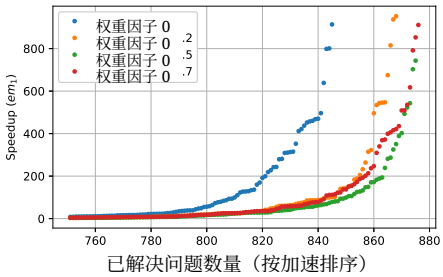
总体而言，与定义性公理相比，抽象能带来更短的运行时间，尤其是能减少因增强而导致 Twee 超时的情况。因此，抽象方法为 Twitch 提供了更大的良好抽象池，以便从中导出领域抽象。此外，我们发现，将抽象数量从 1 增至 10，在使用公理时会显著增加超时次数，而抽象方法对此类增长的鲁棒性要强得多。因此，使用从 Twitch 导出的领域抽象作为 Twee 抽象来增强某些难题，可以提供更多抽象，同时降低造成显著减速的可能性。

然而，对于某些问题，将抽象作为公理提供，相较于使用 Twee 的抽象机制，仍能带来更显著的加速。例如，我们之前用于说明问题的 `LAT075-1` 问题就是一例。 [6] 这表明，未来的工作或许可以将这两种方法结合起来。

[5] 请注意，我们实际上不一定会使用所有 k 抽象来扩充问题，因为其中一些抽象可能无法反向转换（参见第 2 节中的转换讨论）。尽管如此，这一比较仍然有参考意义，因为在本实验中，对于给定问题，实际使用的抽象数量在抽象方法和公理方法中相同。[6]此外，还有少数问题（确切地说是 15 个）中，公理带来了加速，而抽象却导致速度变慢。取 P 为 LAT141-1，我们可以从默认的 Twee（未进行目标扁平化）在 ~ 160 秒内得到一个证明 $\Gamma(P)$。Stitch 在该证明中找到的最佳抽象是 $f(X,\ Y,\ Z) = meet(X,\ join(Y,\ Z))$。将其添加为公理后，可在 $\sim$ 12 秒内得到 $P \oplus A$ 的证明；而使用抽象机制时，则需 ~ 260 秒才能得到证明。



（a）不同数量的抽象以及将其提供给 Twee 的不同方式（作为公理或抽象）下的运行时间仙人掌图。



(b) 不同权重因子的运行时间仙人掌图（见第 4 节）。

图 3：展示局部抽象效果的仙人掌图。图中的每个点对应表 1 中的一个特定问题 P。点的 y 值表示在加入局部抽象后，Twee 证明 P 所需的时间；这些局部抽象即从默认 Twee 对 P 的现有证明中导出的抽象。点的 x 值表示问题按加入局部抽象后的 Twee 运行时间 P 递增排序时所处的序号。换言之，这些点对应算法 2 第 7 行的运行时间。

最后，我们展示图 3b，其中给出了不同权重因子下、加入抽象后的运行时间仙人掌图。权重因子设为约 0.5（将匹配缩写的项的代价减半）似乎效果最佳。这与我们的直觉一致：较低的权重因子意味着更激进，因此会导致更多性能下降（尤其是超时）。不过，对某些个别问题而言，权重因子较低时加速效果最大。

6 相关工作

我们在 Twee 中实现的抽象，是一种自动控制的加权策略 [14]实例。加权后来影响了 Wos 的共振策略 [23]研究。共振器是子句模式，能够使子句选择偏向人类专家认为有希望的形式。这种方法在困难领域中证明了其有效性 [3,8]。大约在同一时期，Veroff 提出了提示策略 [19]，以及证明草图 方法 [20], ：定理证明器首先证明一个较容易或相关的定理，然后重用该证明中的子句作为提示，引导搜索更难的目标。我们的抽象在理念上与 Veroff 的提示恰恰相反：提示子句是可能只推导一次的关键子句，而我们的抽象则是反复出现的关键项结构。

与我们使用 Stitch 密切相关并为之提供灵感的，是此前有关自动化证明压缩的研究。Urban 等人 [21] 提议引入新

定义，以缩短饱和证明器找到的证明。他们的目标是进行证明重构和压缩，而我们主要将压缩用作发现引导搜索模式的机制。类似地，Wernhard 和 Zombori [22] 最近探索了基于语法的 Metamath 证明压缩。

与从内部证明结构中挖掘信息以改进后续搜索这一思路相关的，还有关于饱和证明器的基于学习的引导的研究，例如 ENIGMA [9,6]。不过，这些方法依赖统计模型进行子句选择，而我们的方法则提取显式的句法抽象，为等式完备化提供轻量且易于理解的引导。

Puzis 等人 [16],研究了如何自动评估有趣的推导结果：他们使用自动推理生成并筛选可能有趣的定理。他们的工作启发了我们为从部分 Twee 证明中提取的引理进行排序的简单启发式方法；在我们的场景中进一步探索他们提出的度量方法，也是一个值得研究的方向。

自动推理领域还探索了从相关问题中学习的其他方法。例如，*Octopus* 系统 [13] 将学习与并行搜索相结合，利用较简单问题中的信息来引导较难问题的证明。这与我们的领域抽象方法相契合；该方法受到课程学习 [4]的启发：领域中的较简单问题能够为较难的问题提供可复用的结构模式。

自动定理证明器正越来越多地被用作代数领域中真正的研究工具，以获得数学上有意义的结果。例如，拟群与圈理论中的许多结果都是在 Prover9 和 Otter[10,15] 的辅助下获得的。在此类场景中，自动定理证明器的用处不仅在于找到证明，还在于生成能够被理解并以标准数学术语重新表述的推导。证明草图和证明简化等技术正是为了满足这一需求。我们的抽象在精神上与之相关：Twitch 识别出的反复出现的项模式，理想情况下应对应于代数上有意义的构造；而将证明搜索引导向这些模式，可能有助于生成结构更清晰、更易于分析的证明。

7 局限性与未来工作

我们所提出的系统在许多方面相当简单，并且有许多可以改进之处。首先，我们的领域构建相当粗糙：我们仅限于同一 TPTP 理论中共享具体符号的问题，并且并未尝试以证明草图的方式自动生成相关或弱化的猜想。同样，引理评分以及从证明中进行的项提取都基于简单的启发式方法，这些方法很可能可以进一步改进，从而获得更有用的抽象。

此外，抽象选择被视为一个单目标优化问题，主要关注运行时间加速。采用其他判据或多目标判据，例如证明长度缩减或策略间的鲁棒性，可能会提供更好的指导。直接从累积的（问题、抽象）训练数据中学习抽象提议也可能颇有成效，或许可以借助语言

模型。尽管做了这些简化，我们认为，研究结果表明，即使是相对简单的抽象处理流程，也能生成带来显著改进的抽象，并解决此前无法处理的问题；这也说明，自动化的基于结构的抽象发现是等式定理证明未来研究中一个颇有前景的方向。我们还注意到，对于某些问题，同时提供领域抽象和部分证明抽象时，运行时间最短。具体而言，另一个较难的问题——LCL351-10 ——也通过这种方式得到解决，而其他所有方法均未能解决，这表明这种组合可能带来更多益处。

此外，Twee 中抽象的实现目前相当脆弱。问题在于，我们只会对包含抽象项实例的临界对给予奖励。然而，Twee（与超位置定理证明器一样）会对临界对进行去模化，使用生成的重写规则对其进行简化。因此，与某个抽象匹配的项很容易被重写成不匹配的项。例如，考虑抽象 $f(x,\ x)$。如果 f 也是结合的（$f(f(x,\ y)\ z) \to f(x,\ f(y,\ z))$），那么项 $f(f(x,\ x)\ y)$（它与该抽象匹配）就会被重写为 $f(x,\ f(x,\ y))$（它不与该抽象匹配）。我们正在试验一种类临界对机制：当抽象以某种方式与重写规则重叠，导致重写破坏该抽象时，生成额外的**导出抽象**。通过这种方式，我们能够生成许多额外的抽象，但生成的抽象可能与用户提供的抽象大不相同，而且尚不清楚它们是否符合用户的意图。

最后，我们希望探索的不仅是生成抽象，还包括生成提示。这些是在证明中必不可少、但定理证明器可能难以找到的关键步骤。一个好的提示示例是较大的重写规则（因此不太可能被 Twee 选中），但它能够导出某个猜想或重要引理的证明。

8 结论

我们已经证明，通过压缩现有证明自动生成抽象，是一种很有前景的等式证明策略。为 Twee 增添预先学习的领域抽象后，我们得以证明若干困难的猜想，并将定理证明器在更典型猜想上的运行时间大致缩短了一半。

目前，要创建高质量的抽象，需要有同一领域中的一批较简单问题。如果不存在这类问题，我们也可以从失败的证明尝试中生成抽象，但目前生成的质量较低。由于通常只需要少数几个简单的领域抽象，我们希望更精密的提取机制能够解决这一问题。

显然，该方法仍有很大的改进潜力，无论是抽象的发现方式，还是定理证明器对它们的利用方式。即便如此，我们相信这些结果颇具前景，并激励我们进一步探索这一想法。

致谢。 本研究部分由瓦伦堡人工智能、自主系统与软件计划（WASP）资助，该计划由克努特和爱丽丝

瓦伦堡基金会资助；同时也得到瑞典研究理事会（VR）项目 2025-06153“数学的语义引导的定理证明”的资助。

参考文献

1. Twitch 实验代码库。https://github.com/WeAreDevo/ijcar26-twee_抽象 2. Bachmair, L., Dershowitz, N., Plaisted, D.A.：无失败完备化。载于：重写技术，第 1–30 页。Academic Press（1989），https://doi.org/10.1016/B978-0-12-046371-8.50007-9 3. Beeson, M., Wos, L.：在塔斯基几何中寻找证明。《自动推理期刊》**58**(1), 181207（2017年1月）。https://doi.org/10.1007/s10817-016-9392-2, https://doi.org/ 10 2 4 1007 10817 016 9392 . /s - - - . Bengio, Y., Louradour, J., Collobert, R., Weston, J.：课程学习。载于：第26届机器学习年度国际会议论文集。第 4148 页。ICML ’09，美国计算机协会，纽约州纽约（2009）。https://doi.org/10.1145/1553374.1553380, https://doi.org/10.1145/ 1553374.1553380 5. Bowers, M., Olausson, T.X., Wong, L., Grand, G., Tenenbaum, J.B., Ellis, K., Solar-Lezama, A.：用于库学习的自顶向下合成。《ACM 编程语言论文集》**7**(POPL), 11821213（2023年1月）。https://doi.org/10.1145 3571234 6 3571234, 10 1145 / http://dx.doi.org/ . / . Chvalovský, K., Jakubův, J., Suda, M., Urban, J.：ENIGMA-NG：面向 E 的高效神经网络与梯度提升推理引导。载于：自动演绎——第27届自动演绎国际会议：2019年8月27日至30日，巴西纳塔尔，论文集。第 197–215 页。施普林格出版社，柏林、海德堡（2019）。https://doi.org/10.1007/978-3-030-29436-6_12, https://doi.org/10.1007/978-3-030-29436-6 127 . Claessen, K., Smallbone, N.：一阶霍恩公式在等式逻辑中的高效编码。载于：Galmiche, D., Schulz, S., Sebastiani, R.（编）《自动推理》。第 388–404 页。Springer International Publishing，查姆（2018）8. Fitelson, B., Wos, L.：找到缺失的证明。《自动推理期刊》**27**(2), 201–225（2001）9. Jakubův, J., Urban, J.：Enigma：基于高效学习的推理引导机器。载于：Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O.（编）《智能计算机数学》。第 292–302 页。Springer International Publishing，查姆（2017）10. Kinyon, M., P hillips, J.：矩形拟群与矩形环。《计算机与应用数学》**49**(11), 1679–1685（2005）。https:// doi.org/https://doi.org/10.1016/j.camwa.2005.01.018, https://www.sciencedirect. com/science/article/pii/S0898122105001896 11. Knuth, D.E., Bendix, P.B.：《泛代数中的简单字问题》，第 342–376 页。Springer Berlin Heidelberg，柏林、海德堡（1983）。https://doi.org/10 1007 978 3 642 81955 1_23 . / - - - - . McCune, W.：OTTER 参考手册。技术报告（2003）13. Newborn, M., Wang, Z.：Octopus：结合学习与并行搜索。《自动推理期刊》**33**(2), 171–218（2004）14. Overbeek, R., McCharen, J., Wos, L.：自动定理证明程序的复杂性及相关改进。《计算机与应用数学》**2**(1), 1–16（1976）。https://doi.org/https://doi.org/10.1016/0898-1221(76) 90002-X, https://www.sciencedirect.com/science/article/pii/089812217690002X

15. Phillips, J.D., Stanovský, D.：拟群与圈理论中的自动定理证明。《AI Commun.》**23**(2-3), 267–283（2010年4月）16. Puzis, Y., Gao, Y., Sutcliffe, G.：有趣定理的自动生成。载于：FLAIRS。第 49–54 页（2006）17. Smallbone, N.：Twee：一种等式定理证明器。载于：CADE。第 602–613 页（2021）18. Sutcliffe, G., Suttner, C.：TPTP问题库。《自动推理期刊》**21**(2), 177–203（1998）19. Veroff, R.：使用提示提高自动推理程序的有效性：案例研究。《自动推理期刊》**16**(3), 223–239（1996）20. Veroff, R.：利用证明草图解决开放问题及其他挑战性问题。《自动推理期刊》**27**(2), 157–174（2001）21. Vyskočil, J., Stanovsk`y, D., Urban, J.：通过发明新定义实现自动化证明压缩。载于：程序设计、人工智能与推理逻辑国际会议。第 447–462 页。Springer（2010）22. Wernhard, C., Zombori, Z.：作为语法压缩证明项的数学知识库：探索 Metamath 证明结构（2025），https: //arxiv.org/abs/2505.12305 23. Wos, L.：共振策略。《计算机与应用数学》**29**(2), 133–178（1995）。
https://doi.org/https://doi.org/10.1016/0898-1221(94) 00220-F,
https://www.sciencedirect.com/science/article/pii/089812219400220F

A 实验参数

运行实验时，有多种参数会影响 Twitch 和 Twee 的性能。我们在此列出这些参数。Stitch 和 Twee 的通用参数：

– 允许 Stitch 返回的抽象数量。– Stitch 返回的抽象的最大元数。– 运行 Twee 时是否进行（`flattening` $= True$）或不进行（`flattening` $= False$）目标扁平化。– 影响抽象效果的 Twee `weight-factor` 参数。

领域抽象的专用参数：

– 局部抽象阶段使用的策略。– 局部抽象阶段中 Twee 运行的时限。– 用于确定良好抽象的阈值 τ（参见算法1第9行）。– 是否执行算法1第10行中的 Stitch 最终步骤，以压缩良好抽象；或者直接将所有良好抽象作为领域抽象。

部分证明抽象的特定参数：

– 运行 Twee 以生成部分证明所需的时间 t_{par} 。– 从部分证明中选取的排名靠前的引理数量 k 。

在整个实验过程中，Stitch 的 maximumarity 参数固定为 5。同样，允许 Stitch 返回的抽象数量固定为 10。

为了生成全文使用的领域抽象，LocalAbstractions 阶段针对一组固定的策略，对每个领域问题运行 1000 秒。若将一个策略表示为一对 (`flattening`, `weight-factor`)，那么我们使用的策略集合就是乘积$\{True, False\} \times \{0, 0.2, 0.5, 0.7\}$。

在图 2a 中，我们选取排名前 50 的部分证明引理，并设为 `weight-factor` $=$ 0。2。在图 2b 中，我们启用了 `weight-factor` $= 0.2$、$\tau = 0.2$ 和 Stitch 最终步骤。

为了生成使用部分证明抽象求解困难问题的结果，我们运行 Twee（将目标扁平化设为 `flattening`）t_{par} 秒，以生成部分证明；随后从该证明中选取排名前 k 的引理，用于生成抽象。然后，我们在加入部分证明抽象的困难问题上重新运行 Twee（将目标扁平化设为 `flattening`，并设为 `weight-factor` $= 0.2$），运行剩余的 $(1000 - t_{\text{par}})$ 秒。我们对所有 (`flattening`, t_{par}, k) $\in \{True, False\} \times \{50, 150, 500\} \times \{50, 100\}$ 都进行了上述操作。

为了得出借助领域抽象解决困难问题的结果，我们遍历了相关参数值的 8 种组合，每种组合在不同问题上的成功率各不相同。具体而言，我们对每个问题测试了以下参数组合：

- `flattening` $= True$; `abstraction-weight-factor` $= 0.2$; $\tau = 0.7$; Stitch 最终步骤已启用。
- `flattening` $= False$; `abstraction-weight-factor` $= 0$; $\tau = 0.2$; Stitch 最终步骤已禁用。
- `flattening` $= False$; `abstraction-weight-factor` $= 0$; $\tau = 0.7$; Stitch 最终步骤已禁用。
- `flattening` $= True$; `abstraction-weight-factor` $= 0$; $\tau = 0.5$; Stitch 最终步骤已禁用。
- `flattening` $= False$; `abstraction-weight-factor` $= 0.5$; $\tau = 0.5$; Stitch 最终步骤已启用。
- `flattening` $= False$; `abstraction-weight-factor` $= 0.2$; $\tau = 0.2$; Stitch 最终步骤已启用。
- `flattening` $= False$; `abstraction-weight-factor` $= 0.5$; $\tau = 0.5$; Stitch 最终步骤已启用。
- `flattening` $= True$; `abstraction-weight-factor` $= 0.2$; $\tau = 0.2$; Stitch 最终步骤已禁用。

此处， `flattening` 指的是在经领域抽象增强的困难问题上运行 Twee 时的目标扁平化参数。

图 3a 和图 3b 中的所有运行均使用 `flattening` $= False$ 完成。图 3a 中，`abstraction-weight-factor` 设为 0。5（这对定义公理运行没有影响）。

B 硬件

所有实验均在 AMD Zen 4 EPYC 9354 核心上运行。

C 已解决的困难问题

以下困难问题在 Twitch 派生的抽象帮助下得以解决。请注意，评级依据的是 TPTP v9.2.1 版本。完整的成功运行列表及其对应的提示和实验参数值，请参阅代码仓库 `data/experiments/hard_successes` 目录中的 `json` 文件：https://github.com/WeAreDevo/ijcar26-twee_abstractions。

C.1 部分证明抽象

使用部分证明抽象：

– `GRP723-1.p` (评分：1.00)– 最佳时间：418.47秒– `LAT075-1.p` (评分：1.00)– 最佳时间：141.11秒– `LCL375-10.p` (评分：1.00) – 最佳时间：399.95秒– `LCL395-10.p` (评分：1.00) – 最佳时间：376.37秒– `LCL225-10.p` (评分：0.96) – 最佳时间：11.55秒– `LCL245-10.p` (评分：0.96) – 最佳时间：114.44秒– `LCL289-10.p` (评分：0.96) – 最佳时间：130.55秒– `LCL348-10.p` (评分：0.96) – 最佳时间：11.56秒– `ROB031-2.p` (评分：0.96)– 最佳时间：12.78秒– `ROB035-1.p` (评分：0.96)– 最佳时间：12.30秒– `LAT400-1.p` (评分：0.91)– 最佳时间：698.88 秒

C.2 领域抽象

使用领域抽象：

– `GRP672-11.p` (评分：1.00) – 最佳时间：510.31秒– `GRP723-1.p` (评分：1.00)– 最佳时间：708.57秒– `GRP732-1.p` (评分：1.00)– 最佳时间：549.19秒– `LAT075-1.p` (评分：1.00)– 最佳时间：31.94秒– `LCL304-10.p` (评分：1.00) – 最佳时间：119.69秒– `LCL331-10.p` (评分：1.00) – 最佳时间：40.88秒

– LCL375-10.p (评分：1.00) – 最佳时间：119.14秒– LCL395-10.p (评分：1.00) – 最佳时间：522.33 秒 – ROB032-2.p (评分：1.00) – 最佳时间：41.19秒– ROB034-1.p (评分：1.00) – 最佳时间：42.35秒– LCL225-10.p (评分：0.96) – 最佳时间：295.57秒– LCL245-10.p (评分：0.96) – 最佳时间：125.09秒– LCL289-10.p (评分：0.96) – 最佳时间：105.07秒– LCL339-10.p (评分：0.96) – 最佳时间：247.18秒– LCL348-10.p (评分：0.96) – 最佳时间：31.57秒– ROB031-2.p (评分：0.96) – 最佳时间：33.79秒– ROB035-1.p (评分：0.96) – 最佳时间：32.73秒– LAT400-1.p (评分：0.91) – 最佳时间：160.89秒 使用一组精选的领域问题：– GRP677-1.p (评分：1.00) – 最佳时间：180秒 将领域抽象与部分证明结合使用：

– LCL351-10.p (评分：1.00) – 最佳时间：200s