实现此处与彼处一阶逻辑

Jens Otten伯南
布哥大学 巴西累西腓
jeotten@leancop.de

Torsten Schaub波茨坦
大学 波茨坦，德国
torsten
@cs.uni-potsdam.de

我们介绍了用于此处与彼处一阶逻辑（HT）的自动定理证明器。这些证明器基于 HT 逻辑的原生序列演算，以及将 HT 逻辑公理化嵌入直觉主义逻辑。通过使用自由变量和斯科伦化，对序列演算中的解析证明搜索进行了优化。该嵌入与直觉主义一阶逻辑的序列演算、表演算和连接演算结合使用。我们在大型一阶公式基准测试集上评估了所有证明器，为开发更高效的 HT 证明器奠定了基础。

1 引言

此处与彼处逻辑 （*HT*；）是一种介于经典逻辑（CL）与直觉主义逻辑（IL）之间的中间逻辑。HT 弱于经典逻辑，因为它和直觉主义逻辑一样，不承认排中律 $A \vee \neg A$，因此具有更强的构造性。然而，HT 强于直觉主义逻辑，因为所谓的弱排中律 $\neg A \vee \neg\neg A$ 在 HT 中成立。

HT 在答案集编程 （ASP； [15]）中发挥着核心作用，并为其提供基本的逻辑基础。具体而言，逻辑程序 [10] 的稳定模型可以通过 HT 中一种简单的最小模型构造来刻画，并称为平衡模型 [27]。这种刻画有多项优势。值得注意的是，HT 中的逻辑等价意味着可以进行可替换，而不改变包含该公式的整体公式 [16]的平衡模型。这一性质至关重要，因为它允许独立于上下文简化逻辑程序的某个片段。此外，HT 便于刻画纳入外部语言构造和推理的答案集编程扩展 [1, 6, 2, 5]。文献 [28]介绍了第一个 HT 表列系统及其用于计算平衡模型的扩展。

尽管这些工作为答案集编程（ASP）的发展奠定了重要基石，但它们针对的是命题逻辑，与 ASP 编码的一阶特性形成对照。为此，研究者在 [29], 中引入了一阶 HT 扩展，并将上述部分工作推广到一阶情形 [17]。然而，HT 中的直接自动推理仍未得到发展。现有方法（例如 *anthem* [8, 7], ，它可判定两个 ASP 编码在 HT 中的等价性）依赖于将问题转换为布尔一阶逻辑，并使用经典一阶逻辑的传统定理证明器。

我们通过直接在 HT 中开发一阶定理证明技术来弥补这一不足，从而为 HT 带来了四种新的定理证明器。第一种证明器 leanHaT 基于 HT 的一阶序列演算。其余三种证明器 ileanSeP-HT、ileanTAP-HT 和 nanoCoP-i-HT，则基于将一阶此时彼时逻辑公理化嵌入一阶直觉主义逻辑，并分别使用直觉主义定理证明器 ileanSeP、ileanTAP 和 nanoCoP-i。这些直觉主义证明器分别基于单后件直觉主义序列演算、带前缀的表列演算和带前缀的非子句连接演算。

为 Here-and-There 逻辑开发的四种定理证明器都遵循自动定理证明器的 *lean* 方法，这一方法最初由针对经典一阶逻辑的 lean*TAP* [3] 和 leanCoP [25] 定理证明器推广；它们使用简洁而高效的 Prolog 代码实现。

M. Gebser、D.Inclezan、F. Ricca、M. Carro 和 M. Truszczynski:
第 41 届国际逻辑编程会议（ICLP 2025），EPTCS 439，2026 年，第 453–468 页，doi:10.4204/EPTCS.439.31

© J.Otten& T. Schaub 本作品采用知识共享署名许可协议授权。

本文结构如下。第 2 节介绍 Here-and-There 逻辑的序列演算 LHT，以及将 Here-and-There 逻辑公理化嵌入直觉主义逻辑的方法。第 3 节介绍证明搜索优化和方法，包括用于剪枝序列演算 LHT 搜索空间的技术，例如自由变量和斯科伦化，以及一种优化后的 Here-and-There 逻辑到直觉主义逻辑的公理化嵌入，它大幅减少了公理的数量。我们还详细介绍直觉主义逻辑的带前缀非子句连接演算，该演算由 nanoCoP-i-HT 使用，也是直觉主义逻辑中效率最高的证明搜索演算之一。第 4 节介绍为 Here-and-There 逻辑开发的四种定理证明器的实现细节。第 5 节对这些 Here-and-There 逻辑定理证明器在一大组基准公式上的表现进行了全面的实验评估，随后本文在第 6 节总结全文并展望未来。

2 预备知识

下一节将介绍 HT 逻辑的语法、语义和序列演算，以及 HT 逻辑向直觉主义逻辑的公理化嵌入。

2.1 HT 的语法和语义

HT 逻辑与经典逻辑和直觉主义逻辑具有相同的语法，但语义不同。命题公式由原子（命题）公式 (p, q) 和联结词 $\wedge$（合取）、$\vee$（析取）、$\rightarrow$（蕴含）及 $\neg$（否定）归纳定义。通过添加一阶谓词、变量、项以及量词 $\forall$（全称量词）和 $\exists$（存在量词），可将此定义扩展到一阶逻辑。项可以是变量 (x, y, z)，也可以是 $f(t1, \ldots, t_n)$，其中 f 是 n 元函数符号，$t1, \ldots, t_n$ 是项（$n \geq 0$）。一个（一阶）公式 (A, B, F, G, H) 归纳定义为以下形式之一：原子公式 $P(t1, \ldots, t_n)$，其中 P 是 n 元谓词符号，$t1, \ldots, t_n$ 是项（$n \geq 0$）；或者，对于公式 G、H 和变量 x，形式为 $G \wedge H$、$G \vee H$、$G \rightarrow H$、$\neg G$、$\forall x\, G$ 和 $\exists x\, G$ 的公式。文字 L 的形式为 A 或 $\neg A$，其中 A 是原子公式。公式中的自由变量是指不在任何量词辖域内的变量。$A[x\backslash t]$ 表示将项 t 代入 A 中 x 的所有自由出现位置后得到的公式。$\forall x1, \ldots, x_n$ 和 $\exists x1, \ldots, x_n$ 分别是 $\forall x_1 \ldots \forall x_n$ 和 $\exists x_1 \ldots \exists x_n$ 的简写。

HT 的语义由一个包含两个世界h、t（“此处”和“彼处”）的直觉主义克里普克结构定义。该结构具有恒定的个体域，且这两个世界之间的可及关系是自反且传递的。除蕴含和否定外，联结词和量词在每个单独的世界中均按经典方式解释；蕴含和否定则根据两个世界进行解释。在h 中为真的公式在t中也为真。更多详情请参见 [17] 。

2.2 HT 的序列演算

序列演算 允许我们推导给定逻辑中的所有有效公式。经典一阶逻辑和直觉主义一阶逻辑的序列演算最初由 Gentzen 引入 [11]。形式（序列）演算由规则和公理组成。每条规则都有一个或多个前提和一个结论；公理有一个结论，但没有前提。例如，经典逻辑的序列演算对每个联结词和量词各包含两条规则，以及一条公理。前提和结论都是相继式，形式为$\Gamma \vdash \Delta$，其中 Γ 和 Δ 是（一阶）公式的多重集。Γ, A 或 $A\ \Gamma$ 是 $\Gamma \cup \{A\}$的简写。（一阶）公式F 的相继式证明是相继式 $\vdash F$（即 $\{\} \vdash \{F\}$）的推导树，它应用序列演算的规则，且所有叶节点都是公理。

Mints 给出了 HT 的一阶逻辑的序列演算 [18]。它包含两条公理，以及对每个联结词和量词各两条规则 $\wedge$、 $\vee$、 $\rightarrow$、 $\forall$ 和 $\exists$。此外，否定 $\neg$与每个联结词和量词组合时，总共还有 12 条规则。

定义 1（HT 的序列演算 LHT） *HT* 的序列演算 *LHT* 的公理和规则见图 *1 [18]*。[1] t 是任意项；对于标有 * 的规则，必须遵守特征变量条件：特征变量 z 不得出现在结论中，即不得出现在 Γ、 Δ或 A 中。

The当且仅当F 存在F 在 LHT 中的相继式证明时，公式 n 在 Here-and-There 逻辑中有效 [18].

$$\frac{}{\Gamma, A \vdash A, \Delta}\ \text{公理}_1 \qquad \frac{}{\Gamma, A, \neg A \vdash \Delta}\ \text{公理}_2$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge\text{-左}\ (r1) \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge\text{-右}\ (r8)$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee\text{-左}\ (r9) \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee\text{-右}\ (r2)$$

$$\frac{\Gamma, \neg A \vdash \Delta \quad \Gamma \vdash A, \neg B, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta}\ \rightarrow\text{-左}\ (r14)$$

$$\frac{\Gamma, A \vdash B, \Delta \quad \Gamma, \neg B \vdash \neg A, \Delta}{\Gamma \vdash A \rightarrow B, \Delta}\ \rightarrow\text{-右}\ (r13)$$

$$\frac{\Gamma, \neg A \vdash \Delta \quad \Gamma, \neg B \vdash \Delta}{\Gamma, \neg(A \wedge B) \vdash \Delta}\ \neg\wedge\text{-左}\ (r10) \qquad \frac{\Gamma \vdash \neg A, \neg B, \Delta}{\Gamma \vdash \neg(A \wedge B), \Delta}\ \neg\wedge\text{-右}\ (r3)$$

$$\frac{\Gamma, \neg A, \neg B \vdash \Delta}{\Gamma, \neg(A \vee B) \vdash \Delta}\ \neg\vee\text{-左}\ (r4) \qquad \frac{\Gamma \vdash \neg A, \Delta \quad \Gamma \vdash \neg B, \Delta}{\Gamma \vdash \neg(A \vee B), \Delta}\ \neg\vee\text{-右}\ (r11)$$

$$\frac{\Gamma, \neg B \vdash \neg A, \Delta}{\Gamma, \neg(A \rightarrow B) \vdash \Delta}\ \neg\rightarrow\text{-左}\ (r5) \qquad \frac{\Gamma, \neg A \vdash \Delta \quad \Gamma \vdash \neg B, \Delta}{\Gamma \vdash \neg(A \rightarrow B), \Delta}\ \neg\rightarrow\text{-右}\ (r12)$$

$$\frac{\Gamma \vdash \neg A, \Delta}{\Gamma, \neg\neg A \vdash \Delta}\ \neg\neg\text{-左}\ (r6) \qquad \frac{\Gamma, \neg A \vdash \Delta}{\Gamma \vdash \neg\neg A, \Delta}\ \neg\neg\text{-右}\ (r7)$$

$$\frac{\Gamma, A[x\backslash t], \forall x\, A \vdash \Delta}{\Gamma, \forall x\, A \vdash \Delta}\ \forall\text{-左}\ (r25) \qquad \frac{\Gamma \vdash A[x\backslash z], \Delta}{\Gamma \vdash \forall x\, A, \Delta}\ \forall\text{-右}^*\ (r21)$$

$$\frac{\Gamma, A[x\backslash z] \vdash \Delta}{\Gamma, \exists x\, A \vdash \Delta}\ \exists\text{-左}^*\ (r22) \qquad \frac{\Gamma \vdash \exists x\, A, A[x\backslash t], \Delta}{\Gamma \vdash \exists x\, A, \Delta}\ \exists\text{-right}\ (r26)$$

$$\frac{\Gamma, \neg A[x\backslash z] \vdash \Delta}{\Gamma, \neg(\forall x A) \vdash \Delta}\ \neg\forall\text{-左}^*\ (r19) \qquad \frac{\Gamma \vdash \neg(\forall x\, A), \neg A[x\backslash t], \Delta}{\Gamma \vdash \neg(\forall x\, A), \Delta}\ \neg\forall\text{-右}\ (r23)$$

$$\frac{\Gamma, \neg A[x\backslash t], \neg(\exists x\, A) \vdash \Delta}{\Gamma, \neg(\exists x A) \vdash \Delta}\ \neg\exists\text{-左}\ (r24) \qquad \frac{\Gamma \vdash \neg A[x\backslash z], \Delta}{\Gamma \vdash \neg(\exists x\, A), \Delta}\ \neg\exists\text{-右}^*\ (r20)$$

图 1：HT 的一阶逻辑的序列演算 LHT。

[1]原演算的 $\neg\vee$-右规则略作修正：将前提序列式左侧的公式 $\neg A$ 和 $\neg B$移至右侧。否则，例如，在 HT 中有效的公式 $(p \vee p) \rightarrow p$将无法证明，而在 HT 中无效的公式 $\neg p \vee \neg(p \vee p)$ 却可以证明。

例 1 考虑公式 $F_1 = (p \to q) \vee (q \to p)$，这是 *Pelletier* 的"定理证明的 *75* 个问题" *[30]*中的第 *16* 题。下面给出了 F_1 在 *LHT* 中的相继式证明，因此 F_1 在 *Here-and-There* 逻辑中有效。

$$\dfrac{\dfrac{\dfrac{}{p,q \vdash p,q}\,axiom_1 \quad \dfrac{}{p,\neg p \vdash \neg q,q}\,axiom_2}{p \vdash q,\ q \to p}\,{\to}\text{-}right \quad \dfrac{\dfrac{}{\neg q,q \vdash p,\neg p}\,axiom_2 \quad \dfrac{}{\neg q,\neg p \vdash \neg q,\neg p}\,axiom_1}{\neg q \vdash \neg p,\ q \to p}\,{\to}\text{-}right}{\vdash p \to q,\ q \to p}\,{\to}\text{-}right$$
$$\dfrac{\vdash p \to q,\ q \to p}{\vdash (p \to q) \vee (q \to p)}\,{\vee}\text{-}right$$

考虑公式 $F_2 = \exists y \forall x\ (p(y) \to p(x))$，它是 *Pelletier [30]* 的第 *18* 题。下面给出了 F_2 在 *LHT* 中的相继式证明。ax_1 和 ax_2 分别是公理$_1$ 和公理$_2$，。为便于阅读，四个叶序列右侧的公式 F_2 均已省略；证明的主右分支上的相继式中，公式 $p(a) \to p(b)$ 也已省略。因此，F_2 在 *HT* 中有效。

$$\dfrac{\dfrac{\dfrac{}{p(a),p(b) \vdash p(b),p(c)}\,ax_1 \quad \dfrac{}{\neg p(b),p(b) \vdash \neg p(a),p(c)}\,ax_2}{p(b) \vdash F_2,\ p(a) \to p(b),\ p(c)}\,{\to}\text{-}right \quad \dfrac{\dfrac{\dfrac{\dfrac{}{p(c),\neg p(c) \vdash p(d),\neg p(b)}\,ax_2 \quad \dfrac{}{\neg p(d),\neg p(c) \vdash \neg p(c),\neg p(b)}\,ax_1}{\neg p(c) \vdash F_2,\ p(c) \to p(d),\ \neg p(b)}\,{\to}\text{-}right}{\neg p(c) \vdash F_2,\ \forall x\,(p(c) \to p(x)),\ \neg p(b)}\,{\forall}\text{-}right^{*}}{\neg p(c) \vdash \exists y \forall x\,(p(y) \to p(x)),\ \neg p(b)}\,{\exists}\text{-}right}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(b) \to p(c),\ p(a) \to p(b)}\,{\to}\text{-}right$$
$$\dfrac{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(b) \to p(c),\ p(a) \to p(b)}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(b) \to p(x)),\ p(a) \to p(b)}\,{\forall}\text{-}right^{*}$$
$$\dfrac{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(b) \to p(x)),\ p(a) \to p(b)}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(a) \to p(b)}\,{\exists}\text{-}right$$
$$\dfrac{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(a) \to p(b)}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(a) \to p(x))}\,{\forall}\text{-}right^{*}$$
$$\dfrac{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(a) \to p(x))}{\vdash \exists y \forall x\,(p(y) \to p(x))}\,{\exists}\text{-}right$$

这两个公式在直觉主义逻辑中都无效，但在经典逻辑中都有效。

2.3 HT的公理化嵌入

HT逻辑可以通过将其嵌入直觉主义逻辑来公理化。这种公理化由一组公理 A_1,$A2$，.....指定，并将其添加到给定公式 F 中，使得当且仅当 $(A_1 \wedge A_2 \wedge \ldots) \to F$ 在直觉主义逻辑中有效时，F 在 HT 中有效。

定义 2（HT的公理化嵌入）以下两个公理模式规定了将HT公理化嵌入到直觉主义逻辑中的公理集合 *[17, 18]*。

$$G \vee (G \to H) \vee \neg H \qquad \textit{(HOS)}$$
$$\exists \vec{x}\,(G \to \forall \vec{x}\,G) \qquad \textit{(SQHT)}$$

G 和 H 是任意的（一阶）公式，其中可能包含自由变量，而 $\vec{x}$ 是 G 中任意变量的集合 x_1、.....、x_n。此外，还要对这些模式的公理公式应用全称闭包。更准确地说，如果 F_A 是根据某个公理模式得到的公式，并且 z_1、.....、z_m 是 F_A 中的自由变量，那么 $\forall z_1$、.....、$z_m F_A$ 就是所得的公理。

于是，当且仅当添加*HOS* 和*SQHT* 公理后的公式 F 在直觉主义逻辑中有效时，公式F 在 HT 中有效。

例 *2* 对于 $F_1 = (p \to q) \vee (q \to p)$，公式 $(p \vee (p \to q) \vee \neg q)$ 和 $(q \vee (q \to p) \vee \neg p)$ 是公理模式 *HOS* 的实例。对于 $F_2 = \exists y \forall x\,(p(y) \to p(x))$，公式 $\exists x\,(p(x) \to \forall y\,p(y))$ 和 $\exists x\,(\neg p(x) \to \forall y \neg p(y))$ 是公理模式 *SQHT* 的实例。

3 证明搜索

所介绍的序列演算 LHT 与 HT 的公理化嵌入，结合直觉主义逻辑的证明搜索演算，使我们能够进行证明搜索，即为 HT逻辑中的公式有效性寻找证明。为了进行高效 的证明搜索，还需要进行一些额外的优化，本节将对此进行介绍。

对于 HT 的公理化嵌入，我们需要确定一个公式在直觉主义逻辑中的有效性。直觉主义逻辑中最高效的证明方法之一（另见第 5 节中的结果）是直觉主义非子句连接演算，因此我们也在此对其作简要介绍。

3.1 序列演算 LHT 中的证明搜索

以下优化使第 2.2 节介绍的序列演算 LHT 的命题逻辑的片段（已经）能够实现更高效的证明搜索。

- **自底向上证明搜索**。 序列演算可采用自顶向下的方式推导所有有效公式。为了找到公式F的相继式证明，从序列 $\vdash F$ 开始，并以自底向上（或解析）的方式应用 LHT 的规则，直到推导树的所有叶节点都以 LHT 的公理结束，会更加高效。下文中，我们假设采用这种解析证明搜索。

- **可逆规则无需回溯**。在证明搜索过程中，如果一个序列式可以应用多条规则，那么回溯时（即先前应用的规则无法导出证明时）必须应用所有适用的（备选）规则。例如，对于序列式 $p \wedge q \vdash p \wedge q$ ，$\wedge$-左规则和 $\wedge$-右规则都适用。这样的选择会导致大量回溯。

  对于可逆规则，这种回溯并非必要。如果一条规则的结论可证时，该规则的所有前提也都可证（反方向显然成立），那么这条规则就称为可逆的 。
  例如，直觉主义相继式演算 LJ [11] 的以下（命题逻辑的）规则不可逆，因此需要回溯（例如，考虑序列式 $q \vdash p \vee q$ 的证明）：

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\text{-right} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee\text{-right}$$

  序列演算 LHT 的所有命题逻辑的规则都是可逆的（带有特征变量条件的量词规则也同样如此，标记为 *） [18]。因此，这些（可逆）规则无需在备选规则应用之间进行回溯。

- **优先应用非分支规则**。 如果某条序列演算规则有多个前提，证明搜索就会分成多个分支，而所有分支都必须得到证明。优先应用非分支规则，可以略微提高证明搜索的效率，因为这样能更快找到证明和失败的证明尝试。
- **限制公理公式**。 与经典和直觉主义序列演算一样，公理中的公式A 可以限制为文字和原子公式。可以证明，axiom1 中的A 可以限制为文字，axiom2 中的A 可以限制为原子公式（在 axiom2 中允许 A 为文字是多余的，因为在这种情况下可以应用 $\neg\neg$-left 和 axiom1）。由此得到两个新公理:

$$\frac{}{\Gamma, G \vdash G, \Delta}\ \text{axiom}_1 \qquad \frac{}{\Gamma, H, \neg H \vdash \Delta}\ \text{axiom}_2$$

  其中 G 是一个文字，H 是一个原子公式。在使用自由变量和合一时，只要分支通过公理闭合，就必须以这种方式限制公理公式（见下文）。

这些证明搜索优化为 Here-and-There 逻辑的命题逻辑的片段带来了一个判定过程。这是因为，在每个（分析式）证明步骤中，序列式中至少一个公式会被缩减为一个或多个更小的公式，而序列式中所有其他公式的大小和数量都保持不变。公式 F 的大小是其原子公式和联结词的数量。

HT 的一阶逻辑——与一阶直觉主义逻辑和一阶经典逻辑一样——由于量词规则 $\forall$-left、 $\exists$-right、 $\neg\forall$-right 和 $\neg\exists$-left，不可判定。尽管如此，以下优化使得在序列演算 LHT 中对一阶公式进行证明搜索更加高效。

- **自由变量**。 为了避免在应用规则 $\forall$-left、 $\exists$-right、 $\neg\forall$-right 和 $\neg\exists$-left 时“猜测”合适的项t （这会导致搜索空间极其庞大），我们改用自由变量。这样一来，我们便将对t 的选择推迟到到达公理时；此时使用项合一使G 和H （见上文）相同，从而得到合适的t值。我们用以下新的自由变量（量词）规则替换规则 $\forall$-left 和 $\exists$-right：

$$\frac{\Gamma, A[x\backslash x^*], \forall x A \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall\text{-left} \qquad \frac{\Gamma \vdash \exists x A, A[x\backslash x^*], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists\text{-right}$$

  其中 x^* 是一个新变量。相应地替换规则 $\neg\forall$-right 和 $\neg\exists$-left。

- **特征变量的斯科伦化**。 为了在使用自由变量和项合一时检查特征变量条件，我们采用动态斯科伦化 ，通过项合一的出现检查，检测量词规则的应用顺序是否违反特征变量条件。我们将规则 $\forall$-right* 和 $\exists$-left* 替换为以下新规则：

$$\frac{\Gamma \vdash A[x\backslash t^*], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall\text{-right}^* \qquad \frac{\Gamma, A[x\backslash t^*] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists\text{-left}^*$$

  其中 t^* 是斯科伦项 $f^*(x1,\ldots,x_n)$，其中 f^* 是一个新的函数符号，而$x1,\ldots,x_n$ 是 Γ 和 Δ中的所有自由变量。相应地替换规则 $\neg\forall$-left* 和 $\neg\exists$-right* 。

- **迭代加深**。 自由变量量词规则 （$\forall$-left、 $\exists$-right、 $\neg\forall$-right 和 $\neg\exists$-left）的应用可能导致证明分支不终止。为了恢复完备性，我们对证明推导中允许在一个分支上出现的自由变量数量进行迭代加深。如果给定公式中没有自由变量量词，则不进行迭代加深，从而为这类一阶公式提供判定过程。

3.2 HT 的受限公理化嵌入

在第 2.3 节介绍的 HT 公理模式中，G 和H 是任意公式。这会导致必须（逐步）将无限多个公理添加到给定的待证明公式中，从而造成巨大的搜索空间。为了尽量减少这类公理的数量，我们将G 限制为文字，并将H限制为原子公式。此外，我们可以将公理模式HOS 限制为公式$G \neq H$。

定义 3（HT 的受限公理嵌入）设 F 为一个（一阶）公式， $\mathscr{P}$ 为 F 中谓词符号的集合， $\langle P\rangle$ 为谓词符号 $P \in \mathscr{P}$的元数。以下针对公式 F 的公理集合定义了将HT 的受限公理嵌入到直觉主义逻辑中的方式。

$$\begin{aligned} AxiomsHT := \{ &\forall x_1,\ldots,x_n \forall y_1,\ldots,y_m\,(G \vee (G \to H) \vee \neg H),\ \exists x_1,\ldots,x_n\,(G \to \forall x_1,\ldots,x_n\,G) \mid \\ &G = P(x_1,\ldots,x_n) \text{ or } G = \neg P(x_1,\ldots,x_n), H = P'(y_1,\ldots,y_m) \text{ for } P,P' \in \mathscr{P} \text{ with } P \neq P', n = \langle P\rangle, m = \langle P'\rangle\} \end{aligned}$$

第 5 节中的评估揭示了这是否是一项合理的限制。如果 G 和 H 都限制为原子公式，那么弱排中律 $\neg p \vee \neg\neg p$ 便不再可证。

示例 3 考虑公式 $F_1 = (p \rightarrow q) \vee (q \rightarrow p)$。*AxiomsHT* 集合包含六条公理（针对 $G \in \{p、q\ \neg p\ \neg q\}$ 以及 $H \in \{p、q\}$，其中 $G \neq H$），其中 $(p \vee (p \rightarrow q) \vee \neg q)$ 或 $(q \vee (q \rightarrow p) \vee \neg p)$ 是在直觉主义逻辑中证明 F_1 的必要充分条件。对于公式 $F_2 = \exists y \forall x\ (p(y) \rightarrow p(x))$，*AxiomsHT* 集合包含三条公理（针对 $G \in \{p(x)\ \neg p(x)\}$ 以及 $H \in \{p(y)\}$，其中 $G \neq H$），其中 $\exists x\ (p(x) \rightarrow \forall y\ p(y))$ 是在直觉主义逻辑中证明 F_2 的必要充分条件。

3.3 直觉主义非子句连接演算

与序列式和表演算（它们由连接词驱动）不同，连接演算 [4] 采用连接驱动的搜索策略，以目标为导向，因此能显著提高证明搜索的效率。连接是由具有相同谓词符号但极性不同的文字组成的集合 $\{A_1{}^0, A_2{}^1\}$（在经典逻辑中，极性 1 表示否定公式，即表示集合 $\{A1, \neg A_2\}$）。

非子句连接演算 [21] 适用于非子句矩阵，其中矩阵 M 是子句的集合，而子句 C 是文字 L 和（子）矩阵的集合。它表示一个处于否定范式的公式。前缀是由变量（V）和常量（a）组成的字符串，并分配给每个文字。它可视为一条世界路径，用于捕捉直觉主义逻辑的 Kripke语义。

对于公式 F、极性 pol $\in \{0,1\}$ 和前缀 p，带前缀公式 $\mathrm{F}^{pol} : \mathrm{p}$ 的直觉主义矩阵 M(F pol : p) 按照表 1 归纳定义。x^* 是一个新的项变量，t^* 是斯科伦项 $\mathrm{f}^*(\mathrm{x1}, \ldots, \mathrm{x}_n)$，$\mathrm{V}^*$ 是一个新的前缀变量，a^* 是形如 $\mathrm{f}^*(\mathrm{x1}, \ldots, \mathrm{x}_n)$ 的前缀常量，f^* 是一个新的函数符号，而 $\mathrm{x1}, \ldots, \mathrm{x}_n$ 是相应公式 $\mathrm{F}^{pol} : \mathrm{p}$ 中所有自由项变量和前缀变量。公式 F 的直觉主义非子句矩阵 M(F) 是直觉主义矩阵 $\mathrm{M}(\mathrm{F}^0 : \varepsilon)$。

表 1：直觉主义逻辑的（带前缀）非子句矩阵。

type	$F^{pol} : p$	$M(F^{pol} : p)$
原子的	$A^0 : p$	$\{\{A^0 : p\underline{a^*}\}\}$
α	$(G \wedge H)^1 : p$	$\{\{M(G^1 : p)\}\}, \{\{M(H^1 : p)\}\}$
	$(G \vee H)^0 : p$	$\{\{M(G^0 : p)\}\}, \{\{M(H^0 : p)\}\}$
	$(G \rightarrow H)^0 : p$	$\{\{M(G^1 : p\underline{a^*})\}\}, \{\{M(H^0 : p\underline{a^*})\}\}$
β	$(G \wedge H)^0 : p$	$\{\{M(G^0 : p), M(H^0 : p)\}\}$
	$(G \vee H)^1 : p$	$\{\{M(G^1 : p), M(H^1 : p)\}\}$
	$(G \rightarrow H)^1 : p$	$\{\{M(G^0 : pV^*), M(H^1 : p\underline{V^*})\}\}$

type	$F^{pol} : p$	$M(F^{pol} : p)$
原子的	$A^1 : p$	$\{\{A^1 : p\underline{V^*}\}\}$
α	$(\neg G)^0 : p$	$M(G^1 : p\underline{a^*})$
	$(\neg G)^1 : p$	$M(G^0 : p\underline{V^*})$
γ	$(\forall x G)^1 : p$	$M(G[x\backslash x^*]^1 : p\underline{V^*})$
	$(\exists x G)^0 : p$	$M(G[x\backslash x^*]^0 : p)$
δ	$(\forall x G)^0 : p$	$M(G[x\backslash t^*]^0 : p\underline{a^*})$
	$(\exists x G)^1 : p$	$M(G[x\backslash t^*]^1 : p)$

通常，项替换 σ_T 将项赋给变量。此外，前缀替换 σ_P 将字符串赋给前缀变量，并通过前缀合一 [23] 计算得到。对于直觉主义逻辑，若连接 $\{\mathrm{A}_1^0 : \mathrm{p}_1, \mathrm{A}_2^1 : \mathrm{p}_2\}$ 满足 σ-互补，当且仅当对于组合替换 $\sigma = (\sigma_T, \sigma_P)$，$\sigma_T(\mathrm{A}_1) = \sigma_T(\mathrm{A}_2)$ 且 $\sigma_P(\mathrm{p}_1) = \sigma_P(\mathrm{p}_2)$。直觉主义逻辑的非子句连接演算 [23] 如图 2 所示。F 的直觉主义连接证明是非子句连接演算中 ε,M(F) ε 的一棵推导树，其中所有叶节点都是公理。与形式子句连接演算 [25, 26], 相比，扩展规则得到了推广，并增加了一条分解规则 [21, 22]。

例 4 考虑两个公式 $F_3 = (p \rightarrow p)$ 和 $F_4 = (p \vee \neg p)$。它们的矩阵分别为 $M(F_3) = \{\{p^1 : a_1V_1\}, \{p^0 : a_1a_2\}\}$ 和 $M(F_4) = \{\{p^0 : a_1\}, \{p^1 : a_2V_1\}\}$。这些矩阵中唯一的连接分别是 $\{p^1 : a_1V_1, p^0 : a_1a_2\}$ 和 $\{p^0 : a_1, p^1 : a_2V_1\}$。前缀 $a_1V_1 = a_1a_2$ 可在替换 $\sigma_P(V_1) = a_2$ 下合一，而 a_1 和 a_2V_1 无法合一。因此，只有第一个连接是 σ-互补的，故 F_3 在直觉主义逻辑中有效，而 F_4 无效。

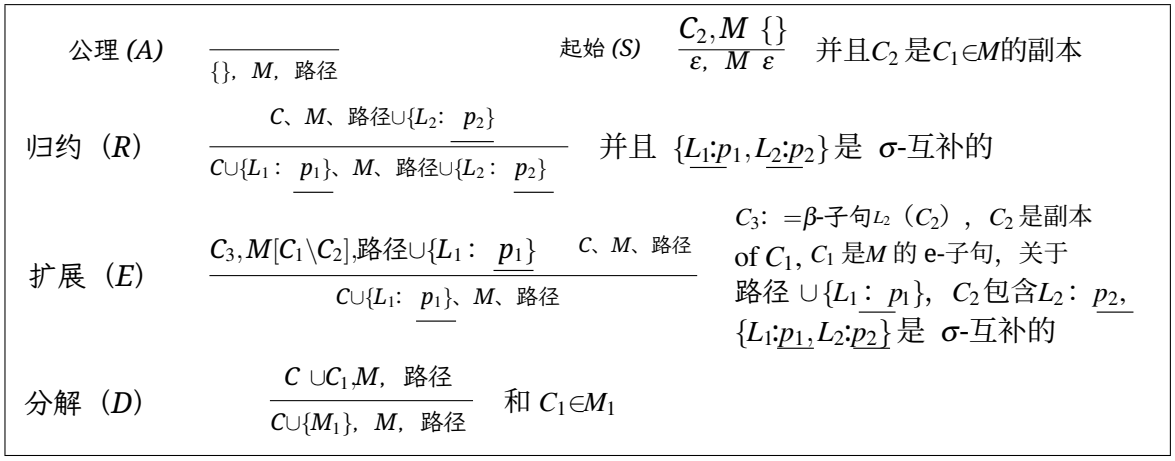

图 2：直觉主义逻辑的非子句连接演算。

4 实现

第 2.2 节介绍的序列演算 LHT，采用第 3.1 节所述的优化，已在 leanHaT 中实现。第 2.3 节介绍的 HT 的公理化嵌入，采用第 3.2 节所述的优化，已在 ileanSeP-HT、ileanTAP-HT 和 nanoCoP-i-HT 中实现。[2]

4.1 序列演算证明器leanHaT

leanHaT 是一个用于带等号的 HT 一阶逻辑的定理证明器。它是第 2.2 节所介绍的序列演算 LHT 的精简 Prolog 实现，并采用了第 3.1 节所述的优化。公式 F 使用以下 Prolog 语法：联结词“,”（合取）、“;”（析取）、“~”（否定）、“=>”（蕴含）、“<=>”（等价），量词“all X:F”（全称量词）和“ex X:F”（存在量词）；原子公式用 Prolog 项表示。序列式 $\Gamma \vdash \Delta$表示为 Left > Right，其中多重集 Left 和 Right 用形如 [F1,...,Fn] 的 Prolog 列表表示。leanHaT 定理证明器的源代码见图 3。它分为序列规则的定义（第 *r1–r26* 行）和实际的证明搜索（第 *1–16* 行）。

谓词 fml(F,Pol,L1>R1,L2>R2,L3>R3,S,FreeV,NewX,Cpy1,Cpy2) 用于指定图 1 中序列演算 LHT 的规则。该演算的 22 条规则各自恰好对应谓词 fml 的一个 Prolog 子句，规则名称后括号中的数字即为该子句的编号。此外，还有四个额外的子句，用于表示等价式 $A \leftrightarrow B$（第 *r15–r18* 行），其定义为 $(A \rightarrow B) \wedge (B \rightarrow A)$。首先应用非分支规则，以及带有特征变量的量词规则。F 是规则结论的主公式，Pol 是其极性，取值为 0 或 1（即序列式的右侧或左侧）；Ln>Rn（其中 $n = 1$、2、3）是添加到规则第 n 个前提中的公式；S 是用作 Skolem 函子的唯一位置；FreeV 是序列式中自由变量的列表；NewX 是新的自由变量（否则为空列表）；Cpy1 会被复制并与 Cpy2 合一。

实际的证明搜索由 Prolog 谓词 prove(Left>Right,S,FreeV,VarLim) 完成。当且仅当在当前位置 S、当前自由变量列表 FreeV 的条件下，存在序列式 Left>Right 的相继式证明，且每个证明分支使用的自由变量不超过 VarLim 个时，该谓词才会成功。

[2]所有介绍的 定理证明器均可在 https://leancop.de/leanhat/ 和 https://leancop.de/ihat/ 获取。

```
        % - 序列规则的规范

(r1)    公式((A,B)  ,    1, [A,B]  > [],   [] > [],     [] > [], _,_,[],[],[]).
(r2)    公式((A;B),      0, [] > [A,B],    [] > [],     [] > [], _,_,[],[],[]).
(r3)    公式(~ (A,B) ,   0, [] > [~A,~B],  [] > [],     [] > [], _,_,[],[],[]).
(r4)    公式(~ (A;B),    1, [~A,~B] > [],  [] > [],     [] > [], _,_,[],[],[]).
(r5)    公式(~ (A=>B) ,  1, [~B] > [~A] ,  [] > [],     [] > [], _,_,[],[],[]).
(r6)    公式(~ (~A),     1, [] > [~A],     [] > [],     [] > [], _,_,[],[],[]).
(r7)    公式(~ (~A),     0, [~A] > [],     [] > [],     [] > [], _,_,[],[],[]).

(r8)    公式((A,B),      0, [] > [A],      [] > [B],    [] > [], _,_,[],[],[]).
(r9)    公式((A;B),      1, [A] > [],      [B] > [],    [] > [], _,_,[],[],[]).
(r10)   公式(~ (A,B),    1, [~A] > [],     [~B] > [],   [] > [], _,_,[],[],[]).
(r11)   公式(~ (A;B),    0, [] > [~A],     [] > [~B],   [] > [], _,_,[],[],[]).
(r12)   公式(~ (A=>B),   0, [~A] > [],     [] > [~B],   [] > [], _,_,[],[],[]).
(r13)   公式((A=>B),     0, [A] > [B],     [~B] > [~A], [] > [], _,_,[],[],[]).
(r14)   公式((A=>B),     1, [~A] > [],     [] > [A, ~B], [B]>[], _,_,[],[],[]).

(r15)   公式((A<=>B),    1, [((A=>B),(B=>A))]  > [],[]>[],[]>[], _,_,[],[],[]).
(r16)   公式((A<=>B),    0, [] > [((A=>B),(B=>A))],[]>[],[]>[], _,_,[],[],[]).
(r17)   公式(~ (A<=>B),1,  [~ ((A=>B),(B=>A))]>[],[]>[],[]>[], _,_,[],[],[]).
(r18)   公式(~ (A<=>B),0,  []>[~ ((A=>B),(B=>A))],[]>[],[]>[], _,_,[],[],[]).
(r19)   公式(~all X:A,   1, [~C] > [], []>[],  []>[],  S,自由变量,[],(X,A),(S~自由变量,C)).
(r20)   公式(~存在 X:A,  0, [] > [~C], []>[],  []>[],  S, 自由变量, [],(X,A),(S~自由变量,C)).
(r21)   fml( 对所有 X:A, 0, [] > [C],  []>[],  []>[],  S, 自由变量, [],(X,A),(S~自由变量,C)。
(r22)   公式( ex X:A,    1, [C] > [],  []>[],  []>[],  S,自由变量,[],(X,A),(S~自由变量,C)).

(r23)   公式(~all X:A,   0, [] > [~C,~所有 X:A], []>[],[]>[], _,_,[Y],X:A,Y:C).
(r24)   公式(~ex X:A,    1, [~C,~ex X:A] > [],  []>[],[]>[], _,_,[Y],X:A,Y:C).
(r25)   公式( all X:A    1, [C, 所有 X:A]  > [],  []>[],[]>[], _,_,[Y],X:A,Y:C).
(r26)   公式( ex X:A,    0, [] > [C,ex X:A],   []>[],[]>[], _,_,[Y],X:A,Y:C).


(a)     证明(F,I)      :- 证明([]>[F],s,[],I)。
(b)     证明(F,I) :- \+ no_mul([([F],0)]),          I1 is I+1, 证明(F,I1)。

        % - 证明搜索

(1)     prove(Left>Right,_,_,_) :-
(2)         member(A,Left),    ( member(B,Right) ; member(~B,Left) ),    (A==B ->
(3)         ! ;  ( A=(~C) ->  C=..[P|_]  ; A=..[P|_] ), P\=',',  P\=';',  P\='~',
(4)         P\='=>',  P\='<=>',  P\=all,  P\=ex,  unify_with_occurs_check(A,B) ).

(5)     prove(Left>Right,S,FreeV,VarLim) :-
(6)         公式(F,Pol,L1>R1,L2>R2,L3>R3,S,FreeV,V,Cpy1,Cpy2),
(7)         ( Pol=1, append(LeftA,[F|LeftB],Left),  RightA=Right,  RightB=[] ;
(8)           Pol=0,  append(RightA,[F|RightB],Right), LeftA=Left, LeftB=[] ),
(9)         ( V=[] ->! ; \+ length(FreeV,VarLim) ),
(10)        append(FreeV,V,FreeV1), 副本_项((Cpy1,FreeV),(Cpy2,FreeV)),
(11)        append(LeftA,LeftB,L0), append(RightA,RightB,R0),
(12)        append(L1,L0,Left1), append(R1,R0,Right1), append(L2,L0,Left2),
(13)        append(R2,R0,Right2), append(L3,L0,Left3), append(R3,R0,Right3),
(14)        prove(左1>右1,l(S),FreeV1,VarLim),
(15)        ( L2=[],R2=[]-> true ; prove(Left2>Right2,r(S),FreeV1,VarLim) ),
(16)          (L3=[],  R3=[]-> 真;  prove(Left3>Right3,x(S),FreeV1,VarLim))。
```

图 3：HT逻辑的 leanHaT 序列演算证明器源代码。

序列演算 LHT 的两个公理在前四行（第*1–4*行）中实现。对两个公理文字进行可靠的合一（因为使用了自由变量）。这是通过内置谓词 unify_with_occurs_check（第*4*行）完成的。如果无法应用任何公理，则选取演算中的一条规则（第*6*行），并检查它能否应用于当前序列式的左侧（第*7*行）或右侧（第*8*行）。如果应用的是可逆规则，则会截断任何回溯（使用 Prolog 的 cut"!"）；否则，会检查当前自由变量的数量是否在当前限制 VarLim （第 *9* 行）之内。

随后，将新的自由变量（如果有）添加到自由变量列表中，并创建 Cpy1 的副本，再使用内置谓词 copy_term 将其与 Cpy2 合一（第*10*行）。每当应用量词规则时，都需要这些副本：(1) 对于自由变量量词规则 ∀-left、∃-right、¬∀-right 和 ¬∃-left，必须将主公式的一个新副本添加到前提的序列式中；(2) 对于特征变量量词规则 ∀-right*、∃-left*、¬∀-left* 和 ¬∃-right*，必须为特征变量插入一个斯科伦项。为避免自由变量被重命名，自由变量列表 FreeV 被作为"参数"添加到 copy_term 谓词的两侧。接下来的三行分别将任何新的公式 R*i*和 L*i*（对于$i = 1,2,3$）添加到第*i*个前提的右侧和左侧（第*11–13*行）。随后，证明搜索将依次继续处理第一个前提（第*14*行）、（如果存在）第二个前提（第*15*行）以及（如果存在）第三个前提（第*16*行）。

主谓词 prove(F,I) 实现迭代加深，并且当且仅当公式 F 存在相继式证明时成功。整数 I 是序列演算中每个分支上自由变量数量的（初始）上限。该谓词通过调用 prove([]>[F],s,[],I)，以起始序列式 []>[F]，起始位置 s、表示自由变量的空列表 [] 以及自由变量上限 I，来执行实际的证明搜索（第 *a* 行）。如果调用失败，则提高自由变量上限，并重新开始证明搜索（第 *b* 行）。谓词 no_mul([([F],0)]) 当且仅当 F 中不存在自由变量量词时成功，也就是说，不存在会导致应用自由变量量词规则的量词；此时迭代加深不会继续（第 *b* 行）（"\+"是 Prolog 的否定）。no_mul 谓词由六行代码实现，也是 leanHaT（核心）证明器中唯一未包含在图 3 中的代码。因此，leanHaT 也是不含自由变量量词的一阶公式的判定过程，也因此是 HT 的命题片段的判定过程。

如果给定的公式包含等号，则会在预处理步骤中添加必要的公理。

例 *5*（序列演算证明器 *leanHaT*）再来看这三个公式 $F_1 = (p \to q) \vee (q \to p)$、$F_2 = \exists y \forall x\,(p(y) \to p(x))$ 和 $F_4 = (p \vee \neg p)$。在 *ILTP*问题库 *[31]*（另见第 *5.1* 节）中，这些公式分别作为问题 *SYN416+1*、*SYN048+1*和 *SYN387+1*收录。为了确定这些公式是否有效，我们将 *leanHaT* 证明器加载到 *Prolog* 中（使用 [*leanhat*]），然后调用以下目标：*prove(((p=>q) ; (q=>p)), 1) , prove((ex Y: all X: (p(Y) => p(X))), 1)* 和 *prove((p ; ~ p), 1)*。对于前两个公式，*Prolog* 返回*"Yes"*；对于第三个公式，则返回*"No"*。因此，公式 F_1 和 F_2 在 *HT*逻辑中有效，而 F_4 无效。

4.2 公理化序列演算与表演算证明器ileanSeP-HT 和 ileanTAP-HT

定理证明器 ileanSeP-HT 是直觉主义序列演算证明器 ileanSeP 针对带等号的 HT 的一阶逻辑的公理化扩展。ileanSeP 是单后件直觉主义序列演算的紧凑 Prolog 实现。与表演算类似，它采用自底向上证明搜索、自由变量和动态斯科伦化来处理量词。结合项合一中的出现检查，这确保了特征变量条件得到满足。如有必要，会在预处理步骤中添加等号公理，并根据 HT逻辑对其进行处理。

定理证明器 ileanTAP-HT 是直觉主义表列证明器 ileanTAP 针对带等号的 HT 的一阶逻辑的公理化扩展。ileanTAP [19] 是带前缀的表格演算的紧凑 Prolog 实现，类似于 Fitting 的带前缀表演算 [9]。它通过添加前缀来捕捉直觉主义逻辑的 Kripke语义，从而扩展了经典演算 [32]；它使用自由前缀变量 [33]，并将斯科伦化扩展到前缀常量。首先，ileanTAP-HT 执行经典证明搜索，收集能够闭合分支的文字前缀。如果该搜索成功，则使用前缀合一来统一这些前缀，并确保给定公式在直觉主义下同样有效。如有必要，会在预处理步骤中添加等号公理，并根据 HT逻辑对其进行处理。

这两个 HT 定理证明器可能都不完备，也就是说，对于某些有效公式，它们无法找到证明，因为公理化在优化时考虑了所添加公理的数量（参见第 3.2 节）。

4.3 公理化连接证明器nanoCoP-i-HT

自动定理证明器 nanoCoP-i-HT 是直觉主义非子句连接证明器 nanoCoP-i 的公理化扩展，适用于带等号的 HT 一阶逻辑。nanoCoP-i 是一种紧凑的 Prolog 实现，用于直觉主义一阶逻辑的非子句连接演算 [23, 24]，也是经典证明器 nanoCoP 的扩展 [22]。它基于第 3.3 节介绍的带前缀非子句连接演算。首先，nanoCoP-i 执行经典证明搜索，并收集每个连接的前缀。如果搜索成功，则统一每个连接中文字的前缀。其他优化技术包括正则性、引理、受限回溯和策略调度 [20, 24]。必要时，会在预处理步骤中添加等号公理，并按照 HT 逻辑处理。目前，nanoCoP-i 2.0 是一阶直觉主义逻辑领域速度最快的定理证明器之一 [24]。同样，由于嵌入直觉主义逻辑时仅使用了第 3.2 节所述的有限数量的公理，nanoCoP-i-HT 可能不完备。[3]

5 实验评估

第 4 节介绍的四种实现都在现有基准库中的大量问题上进行了评估，并与用于直觉主义逻辑和经典逻辑的定理证明器进行了比较。

5.1 基准测试问题

目前，还没有用于测试和评测 HT 的一阶逻辑定理证明器的大型公式集。迄今为止，只有一小组问题可用，它们表示在*anthem* 系统 [8, 7]的背景下开发的逻辑程序强等价问题。由于 HT 的语法与经典逻辑和直觉主义逻辑相同，我们可以使用这些逻辑现有的基准公式，尽管这些公式上的结果并无明显意义。由于 HT 逻辑是直觉主义逻辑的扩展，我们使用直觉主义逻辑的 ILTP问题库 [31] 来测试和评测我们的四个 HT 定理证明器。HT、直觉主义逻辑和经典逻辑之间的关系如图4所示。

[3]本文介绍的证明演算、方法和定理证明器侧重于证明 公式。要有效地反驳 公式，则需要采用不同或额外的方法和技术。因此，（方法和定理证明器的）完备性并没有那么重要。例如，尽管 nanoCoP-i-HT（很可能）并不完备，但它证明的问题显著多于 leanHaT（见第 5.2 节）。此外，nanoCoP-i 使用的最成功的（证明搜索）技术之一（因此 nanoCoP-i-HT 也使用了该技术）并不完备 [24, 20]（总体而言，nanoCoP-i 是完备的，因为所采用的策略调度中也包含完备策略）。最后，与一阶经典逻辑和一阶直觉主义逻辑一样，HT 的一阶逻辑也是不可判定的。

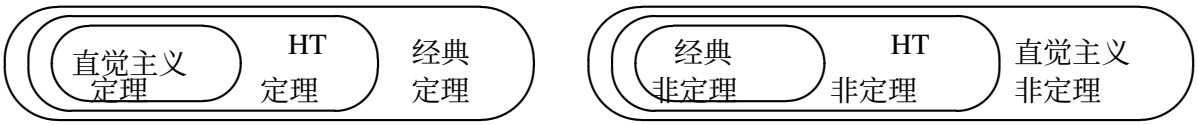

图4：直觉主义逻辑、Here-and-There 逻辑与经典逻辑之间的关系。

ILTP问题库的版本1.1.2包含 2550 个一阶公式，并提供经典逻辑和直觉主义逻辑的状态及难度评级信息。这些问题分为 24 个类别，涉及广泛的不同领域，例如智能体（AGT）、一般代数（ALG）、构造几何学（GEJ）、几何学（GEO）、知识表示（KRS）、管理（MGT）、自然语言处理（NLP）、数论（NUM）、集合论（SET）、软件验证（SWV）、句法（SYN）和直觉主义句法（SYJ）。

ILTP 库中的问题采用 TPTP语法，由一个猜想 公式以及（可选的）n个公理公式公理1、...、公理$_n$（$n \geq 0$）组成。于是，待证明的公式为（公理$_1 \wedge \ldots \wedge$公理$_n$） $\rightarrow$猜想。如果没有猜想 （公式），则必须判定公式（公理$_1 \wedge \ldots \wedge$公理$_n$）的不可满足性。

5.2 评估

第 4 节介绍的四个 Here-and-There 逻辑定理证明器，均在 ILTP 库的 2550 个问题上进行了评估。测试在一台 LIFEBOOK U9311 上进行，该设备配备 4 核 i7-1185G7 CPU 和 16GB 内存，运行 Linux 5.15.0 和 ECLiPSe Prolog 5.10 版。[4]

表2 展示了在 CPU时间限制为 10 秒的条件下，对 ILTP 库 v1.1.2 中全部 2550 个一阶问题进行评估的结果。[5] 其中包括 Here-and-There 逻辑证明器 leanHaT 1.0、ileanSeP-HT 1.0、ileanTAP-HT 1.0 和 nanoCoP-i-HT 2.0。还包括直觉主义逻辑证明器 ileanSeP 1.0 、 ileanTAP 1.17 [19] 和 nanoCoP-i 2.0 [24]。这些直觉主义逻辑证明器为使用类似演算的（原生）Here-and-There 逻辑定理证明器提供了下限。例如，Here-and-There 逻辑的（原生）序列式证明器或语义树证明器应至少解决与 ileanSeP 同样多的问题，而 Here-and-There 逻辑的（原生）连接证明器应至少解决与 nanoCoP-i 同样多的问题。该表列出了每个证明器已证明的问题总数、在特定时间内已证明的问题数、在最重要领域中已证明的问题数（见第 5.1 节）、仅由该证明器（相对于其逻辑）证明的“独有”问题数、已证伪（无效）问题数以及错误数（通常是 ECLiPSe Prolog 的全局栈/trail栈溢出）。由于 Here-and-There 逻辑的公理化证明器很可能并不完备，因此它们不会证伪任何（无效）问题。

在 HT 证明器中，nanoCoP-i-HT 证明的问题数量遥遥领先。序列演算证明器 leanHaT 位居第二，而基于 ileanSeP 和 ileanTAP 的证明器表现则差得多。nanoCoP-i-HT 和 leanHaT 已证明的独有问题数量均居首位，涵盖了 ileanSeP-HT 所证明的问题。在 nanoCoP-i-HT 的第三种策略中加入猜想起始子句技术（即使用 [conj,scut,cut]，而非 [scut,cut],，见 [24]），已证明问题的数量便增至 635。总而言之，nanoCoP-i-HT 和 leanHaT 共证明了 652 个问题。

[4]ECLiPSe Prolog 5.x 可从 https://eclipseclp.org/Distribution/Builds/ 获取；较新版本的 ECLiPSe Prolog 缺少重要功能，性能也明显较低。[5]由于底层证明搜索问题的时间复杂度较高，延长时间限制只能略微增加已证明的问题/公式数量。只有 nanoCoP-i 和 nanoCoP-i-HT 定理证明器会从延长时间限制中稍微受益，因为这样策略调度便能执行全部 12 种（内置）策略，而不只是前九种。

表2：HT 定理证明器在 ILTP 库上的结果。

定理证明器	ileanTAP	ileanTAP-HT	ileanSeP	ileanSeP-HT	leanHaT	nanoCoP-i	nanoCoP-i-HT
逻辑	IL	**HT**	IL	**HT**	**HT**	IL	**HT**
已证明	312	154	300	209	**378**	799	**626**
0 to 1秒。	306	145	273	184	**345**	699	**528**
1到10秒。	6	9	27	25	**33**	100	**98**
AGT	0	0	4	3	4	15	12
ALG	6	6	0	0	3	12	9
GEJ	11	0	7	3	11	85	59
GEO	0	0	1	0	1	10	7
KRS	18	0	17	3	31	91	41
MGT	6	0	1	0	1	32	16
NLP	11	11	3	0	1	13	8
NUM	1	0	0	0	0	49	30
SET	14	1	19	12	16	112	76
SWV	51	48	82	71	107	134	127
SYN	107	56	102	93	134	125	139
SYJ	81	28	62	23	67	102	90
唯一	0	3	2	0	**24**	413	**238**
已证伪	4	0	4	0	**31**	88	**0**
错误	60	27	151	114	**0**	3	**3**

HT 的定理证明器也可用于反驳直觉主义逻辑中的问题（如果 F 在 HT 中无效，那么 F 在直觉主义逻辑中也无效）。例如，ILTP 库中 18 个可扩展（且在经典 逻辑中有效）的问题 SYJ207 至 SYJ212，即使对命题逻辑的直觉主义证明器而言，也很难反驳。leanHaT 轻而易举地在不到一秒内反驳了所有这些问题。

表 3 展示了五个经典逻辑定理证明器的结果：经典定理证明器（原生）leanTAP 2.3 [3] 和 nanoCoP 2.0 [20], ，以及三个直觉主义定理证明器 ileanTAP、ileanSeP 和 nanoCoP-i。后者通过向直觉主义逻辑中添加用于排中律的公理模式 $A \vee \neg A$（对于所有原子公式A）来实现经典逻辑到直觉主义逻辑的嵌入。这些结果为公理化方法提供了一个上限。例如，结合 HT 的公理化嵌入与 nanoCoP-i 定理证明器时，nanoCoP-i-C 的性能就是我们所能期待的最佳表现。

表 3：经典定理证明器在 ILTP 库上的结果。

定理证明器	ileanTAP-C	ileanSeP-C	leanTAP	nanoCoP-i-C	nanoCoP
逻辑	CL	CL	CL	CL	CL
已证明	252	274	516	855	1086
0 to 1秒。	242	248	499	728	942
1到10秒。	10	26	17	127	144
已证伪	0	0	0	5	86
错误	82	105	375	3	1

6 结论

尽管 HT逻辑为答案集编程提供了逻辑基础，但在开发能够自动化 HT 中形式推理的工具和高效演算方面，迄今为止所做的工作仍然很少。本文通过为 HT 的一阶 逻辑引入四个自动定理证明器来解决这一问题：leanHaT 为 HT 实现了优化的原生序列演算；ileanSeP-HT、ileanTAP-HT 和 nanoCoP-i-HT 则将公理化嵌入到直觉主义逻辑中，并分别结合直觉主义定理证明器 ileanSeP、ileanTAP 和 nanoCoP-i。就我们所知，这些是首批无需对给定公式进行翻译的 HT一阶 逻辑定理证明器。

值得注意的是，所介绍的四种定理证明器都基于非子句形式证明演算，即它们不需要将给定公式转换为（合取或析取）子句形式，而是在整个证明搜索过程中保留原始公式的结构。如果要返回可读的证明，这一点尤其重要。

针对 HT 的四种定理证明器均在 ILTP问题库上进行了评估，提供了全面的性能结果。与类似的直觉主义逻辑序列式或语义树证明器相比，leanHaT 能证明更多问题。leanHaT 能解决（即证明或反驳）的问题比经典语义树证明器 lean*TAP*少约 20%；考虑到 lean*TAP*证明的一些公式在 HT 中并非有效，这已是相当扎实的成绩。

对于公理化嵌入，需要在给定公式中添加大量公理。例如，对于一个含有 25 个不同谓词符号的命题公式，需要添加 1200 多条公理；这一数字已经考虑了我们优化后的公理化方案。正如预期的那样，也与类似方法的结果一致，这些附加公理显著扩大了搜索空间。只有与直觉主义连接证明器 nanoCoP-i 结合使用时，这种方法才能带来令人信服的性能，nanoCoP-i-HT 证明器的结果便展示了这一点。

哪些公理是将 HT 完备且可靠地公理化嵌入直觉主义逻辑所必需的，这仍是一个悬而未决的问题。我们没有发现任何证据表明，针对命题逻辑的 HT 的优化公理化是不完备的。对于一阶 HT，它很可能是不完备的。例如，公式$\exists y(\exists x\, p(x) \rightarrow p(y))$（ILTP 库中的问题 SYN971+1）很容易由 leanHaT 证明，但任何用于 HT 的公理化定理证明器都无法证明。要回答这一悬而未决的问题，还需要进一步开展理论和实践研究。

尽管 leanHaT 和 nanoCoP-i-HT 的性能表现十分可喜，本文研究的重点并非效率，而是着手开发 HT 的一阶逻辑定理证明器，并为今后开发效率更高的定理证明器奠定基础。无论是（原生）序列式演算或表演算，还是公理化嵌入，都无法作为开发高效 HT 定理证明器的基础。为此，需要开发基于目标导向证明搜索的原生证明方法，例如连接演算。这将是未来工作的一个方向。

未来工作还将包括与*anthem* 系统的比较，以及该系统将 Here-and-There 逻辑转换为一阶经典逻辑的做法。此外，开发一个 Here-and-There 逻辑的基准库，其中也包含来自实际应用的问题，例如表示逻辑程序的强等价，将十分有益，并能进一步推动高效的 Here-and-There 逻辑定理证明器的开发。

致谢

作者谨向波茨坦大学、内布拉斯加大学奥马哈分校和德克萨斯大学奥斯汀分校 *anthem* 小组的成员致谢，感谢他们进行的许多富有启发性的讨论。也感谢匿名审稿人提出的宝贵意见。

参考文献

[1] Felicidad Aguado、Pedro Cabalar、Martin Diéguez、Gilberto Pérez 和 Concepción Vidal（2013）：时序平衡逻辑：综述。《应用非经典逻辑杂志》23(1–2)，第 2–24 页，doi:10.1080/11663081.2013.798985。[2] Harald Beck、Minh Dao-Tran 和 Thomas Eiter（2016）：等价流推理程序。载于 Kambhampati [14],第 929–935 页。[3] Bernhard Beckert 和 Joachim Posegga（1995）：leanTAP：精简表 tableau 演绎。《自动推理杂志》15(3)，第 339–358 页，doi:10.1007/BF00881804。[4] Wolfgang Bibel（1987）：自动定理证明。《人工智能》，F. Vieweg und Sohn，威斯巴登，doi:10.1007/978-3-322-90102-6。[5] Pedro Cabalar、Jorge Fandinno 和 Luis Fariñas del Cerro（2020）：自认知答案集编程。《人工智能》289，第 103382 页，doi:10.1016/j.artint.2020.103382。[6] Pedro Cabalar、Roland Kaminski、Max Ostrowski 和 Torsten Schaub（2016）：带约束默认推理的 ASP 语义。载于 Kambhampati [14],第 1015–1021 页。[7] Jorge Fandinno、Christoph Glinzer、Zachary Hansen、Jan Heuer、Yuliya Lierler、Vladimir Lifschitz、Torsten Schaub 和 Tobias Stolzmann（2025）：Anthem 2.0：答案集编程的自动推理。可见于 https://arxiv.org/abs/2507.11704。（即将发表于《逻辑编程理论与实践》2025 年卷。）[8] Jorge Fandinno、Vladimir Lifschitz、Patrick Lühne 和 Torsten Schaub（2020）：使用 anthem 和 Vampire 验证紧逻辑程序。《逻辑编程理论与实践》20(5)，第 735–750 页，doi:10.1017/S1471068420000344。[9] Melvin Fitting（1983）：模态逻辑与直觉主义逻辑的证明方法。D. Reidel，多德雷赫特，doi:10.1007/978-94-017-2794-5。[10] Michael Gelfond 和 Vladimir Lifschitz（1990）：含经典否定的逻辑程序。载于 David Warren 和 Péter Szeredi（编）：第七届国际逻辑编程会议论文集（ICLP’90），麻省理工学院出版社，第 579–597 页。[11] Gerhard Gentzen（1935）：关于逻辑推理的研究。《数学杂志》39，第 176–210、405–431 页，doi:10.1007/BF01201353。[12] Kurt Gödel（1932）：论直觉主义命题演算。《维也纳科学院通报》，第 65–66 页。[13] Arend Heyting（1930）：直觉主义逻辑的形式规则。载于：《普鲁士科学院会议报告》，柏林德意志科学院，第 42–56 页。[14] Subbarao Kambhampati（编，2016）：第二十五届国际人工智能联合会议论文集（IJCAI’16）。IJCAI/AAAI 出版社。[15] Vladimir Lifschitz（2019）：答案集编程。施普林格，尚，doi:10.1007/978-3-030-24658-7。[16] Vladimir Lifschitz、David Pearce 和 Agustín Valverde（2001）：强等价逻辑程序。《ACM 计算逻辑汇刊》2(4)，第 526–541 页，doi:10.1145/383779.383783。[17] Vladimir Lifschitz、David Pearce 和 Agustín Valverde（2007）：含变量逻辑程序强等价的刻画。载于 Chitta Baral、Gerhard Brewka 和 John Schlipf（编）：第九届国际逻辑编程与非单调推理会议论文集（LPNMR’07），《人工智能讲义》4483，施普林格，海德堡，第 188–200 页，doi:10.1007/978-3-540-72200-7_- 17。[18] Grigori Mints（2010）：带量词的此处与彼处逻辑的无切割形式系统。《纯粹与应用逻辑年刊》162(3)，第 237–242 页，doi:10.1016/J.APAL.2010.09.009。[19]Jens Otten（1997）：ileanTAP：直觉主义定理证明器。载于 Didier Galmiche（编）：TABLEAUX 1997，《人工智能讲义》1227，施普林格，海德堡，第 307–312 页，doi:10.1007/BFb0027422。

[20] Jens Otten（2010年）：限制连接演算中的回溯。《人工智能通讯》23(2–3)，第159–182页，doi:10.3233/AIC-2010-0464。[21] Jens Otten（2011年）：非子句连接演算。载于 Kai Brünnler 和 George Metcalfe（编）：TABLEAUX 2011，《人工智能讲义》6793，施普林格，海德堡，第226–241页，doi:10.1007/978-3-642-22119-4 18。[22] Jens Otten（2016年）：nanoCoP：一种非子句连接证明器。载于 Nicola Olivetti 和 Ashish Tiwari（编）：IJCAR 2016，《人工智能讲义》9706，施普林格，海德堡，第302–312页，doi:10.1007/978-3-319-40229-1 21。[23] Jens Otten（2017年）：非经典逻辑的非子句连接演算。载于 Renate Schmidt 和 Cláudia Nalon（编）：TABLEAUX 2017，《人工智能讲义》10501，施普林格，尚，第209–227页，doi:10.1007/978-3-319-66902-1_13。[24] Jens Otten（2021年）：用于经典逻辑、直觉主义逻辑和模态逻辑的 nanoCoP 2.0 连接证明器。载于 Anupam Das 和 Sara Negri（编）：TABLEAUX 2021，《人工智能讲义》12842，施普林格，尚，第236–249页，doi:10.1007/978-3-030-86059-2_14。[25] Jens Otten 和 Wolfgang Bibel（2003）：leanCoP：精简的基于连接的定理证明。《符号计算杂志》36(1–2)，第139–161页，doi:10.1016/S0747-7171(03)00037-3。[26] Jens Otten 和 Wolfgang Bibel（2017年）：基于连接的自动定理证明进展。载于 Mike Hinchey、Jonathan P. Bowen 和 Ernst-Rüdiger Olderog（编）：《可证明正确的系统》，NASA系统与软件工程专著，施普林格，尚，第211–241页，doi:10.1007/978-3-319-48628- 4 9。[27] David Pearce（2006年）：平衡逻辑。《数学与人工智能年刊》47(1–2)，第3–41页，doi:10.1007/s10472-006-9028-z。[28] David Pearce、Inmaculada de Guzmán 和 Augustín Valverde（2000年）：平衡蕴涵的表演算。载于 Roy Dyckhoff（编）：第九届解析表及相关方法自动推理国际会议论文集（TABLEAUX 2000），《计算机科学讲义》1847，施普林格，海德堡，第352–367页，doi:10.1007/10722086_28。[29] David Pearce 和 Augustín Valverde（2005年）：构造逻辑的一阶非单调扩展。《逻辑研究》30(2–3)，第321–346页，doi:10.1007/s11225-005-8473-8。[30] Francis Jeffry Pelletier（1986年）：测试自动定理证明器的七十五个问题。《自动推理杂志》2(2)，第191–216页，doi:10.1007/BF02432151。[31] Thomas Raths、Jens Otten 和 Christoph Kreitz（2007年）：直觉主义逻辑的 ILTP 问题库。《自动推理杂志》38，第261–271页，doi:10.1007/s10817-006-9060-z。[32] Raymond M. Smullyan（1968年）：一阶逻辑。《数学及其边界领域成果》，施普林格，柏林、海德堡、纽约，doi:10.1007/978-3-642-86718-7。[33] Lincoln A. Wallen（1990年）：非经典逻辑中的自动演绎。麻省理工学院出版社，马萨诸塞州剑桥。