用于 Prolog 验证的自动定理证明*

Fred Mesnard ThierryMarianne
Étienne PayetLIM，留尼汪大学，法国
{frederic.mesnard,thierry.marianne,etienne.payet}@univ-reunion.fr

LPTP（逻辑程序定理证明器）是一款交互式的、基于自然演绎的定理证明器，适用于带有失败即否定、带出现检查的合一，以及受限但可扩展的内置谓词集的纯 Prolog 程序。借助 LPTP，可以形式化证明此类 Prolog 程序的终止性和部分正确性。LPTP 由 Robert F. Stärk 于 20 世纪 90 年代中期设计。它以 ISO-Prolog 编写，并配有 Emacs 用户界面。

从理论角度看，Stärk 在关于 LPTP 的出版物中，为所研究的 Prolog 程序 P 关联了一组一阶公理 IND(P)。IND(P) 包含该程序 P 的 Clark 等式理论、对 P 中每个用户定义逻辑过程的成功、失败和终止的定义、联系这三个视角的公理，以及用于证明归纳性质的公理模式。因此，LPTP 是一款专用证明编辑器，这些公理已内置其中。

我们提议将这些公理转换为一阶公式（FOF），并运用自动定理证明器来检验所关注的性质。我们以 FOF 作为中间语言，尝试使用自动定理证明器进行 Prolog 程序验证。我们在一个基准测试上评估了这一方法，该基准测试包含约 400 个来自 LPTP 随附库的 Prolog 程序性质。用于根据给定的输入 Prolog 程序及其性质生成一组 FOF 文件的编译器，以及该基准测试，均已公开提供。

1 引言

20 世纪 90 年代中期，Robert F. Stärk 为 Prolog 验证定义了一个框架 [26, 29]。他研究了 ISO-Prolog [13] 的一个子集：带有失败即否定、采用出现检查的合一，并允许使用受限但可扩展的一组内置谓词的纯 Prolog 程序。他用公理模式给出了 Prolog 通常操作语义的一阶形式化。纳入终止条件的安全性条件要求在求值否定目标之前，这些目标必须是基项。他证明了关于终止、成功和失败的可靠性与完备性。该框架还允许通过归纳证明部分正确性性质，归纳时依据的是定义谓词的子句，并将这些子句视为归纳定义。第 3 节和第 4 节将讨论一些示例。逻辑理论被硬编码在一个名为 LPTP（逻辑程序定理证明器）的交互式专用一阶自然演绎定理证明器中。Stärk 使用 ISO-Prolog 实现了 LPTP，并提供了 Emacs 用户界面、HTML 和 TEX 管理器、详细的用户手册，以及一个包含皮亚诺数、整数、列表、排序算法等谓词及其众多已证明性质的库。LPTP 的副本可从 https://github.com/FredMesnard/lptp 获取。

三十年后，LPTP 仍可在任何 ISO-Prolog 处理器上运行，并保留其初始界面。如今，计算机程序的形式化验证已成为计算机科学中的一门成熟学科。然而，通过交互式定理证明进行程序验证仍然是一个缓慢的过程，并且需要非平凡的

*本文是 [21] 的更新版本，原文发表于 2024 年 LPAR 补充卷。

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 469–481 页，doi:10.4204/EPTCS.439.32

© F. Mesnard、T. Marianne 和 É. Payet 本作品采用知识共享署名许可协议授权。

技能。另一方面，在过去三十年中，计算能力的提升以及自动定理证明的发展都十分显著。例如，TPTP（定理证明器问题集， [30]）是一个用于自动定理证明的测试问题库。它提供在线工具，用于检查输入问题的语法，并应用用户选定的一组自动定理证明器。其中，E [25]和 Vampire [16] 是两个功能强大且可免费使用的自动定理证明器，多年来在许多国际竞赛中表现出色。交互式定理证明器的实现者得以利用这些进展，为其工具实现所谓的自动化工具，例如参见 [23, 3]。

这一发展引出了以下问题：我们也能否使用 TPTP FOF 世界语来表述 Stärk 为某个逻辑程序关联的逻辑理论？我们能否使用现成的TPTP 证明器，并在合理时间内获得自动证明？采用这种方法能否取得令人满意的成功率？

本文的主要贡献如下。我们使用 FOF（一阶形式，TPTP 提出的逻辑语言之一，参见 [31]）作为中间语言，据我们所知，首次开展了使用自动定理证明器 E 和 Vampire 进行 Prolog 程序验证的实验，涵盖终止性和部分正确性。我们针对约 400 项 Prolog 程序性质评估了这种方法。将 Stärk 的理论应用于给定输入 Prolog 程序及其性质，并将其编译为一组 FOF 文件的编译器，以及基准测试，均已公开发布于
https://github.com/atp-lptp/automated-theorem-proving-for-prolog-verification。

本文组织如下。下一节简要概述 LPTP 系统。第三节逐步介绍如何将 Prolog 程序、与其相关的 LPTP 公理以及一项关注的性质编译为 FOF 文件。随后，我们将介绍实验评估和相关工作，并作出总结。

2 符号表示

FOF（一阶形式）是 TPTP 中一种广为人知的逻辑语言，用于表示一阶逻辑（FOL）公理和猜想。公式写作 fof(名称,角色,公式)，其中 名称 是公式的名称，角色 为公理或猜想，而 公式 的非正式定义如下：

FOL	FOF	FOL	FOF
$A \wedge B$	A & B	$\neg p(x)$	~ p(X)
$A \vee B$	A | B	$\exists x.A$	?[X] : A
$A \to B$	A => B	$\forall x.A$	![X] : A

接下来的各节中将出现许多示例。

设P 是一个纯逻辑程序，其中子句体中可以出现负文字（在 [19] 中也称为正则程序 ）。为简洁起见，除了等式 =/2 外，我们不考虑内置谓词（完整处理参见 [29] ）。我们从与P相关联的一阶语言 $\mathscr{L}$ 开始。 $\mathscr{L}$的目标如下：

$$G,H ::= \texttt{true} \mid \texttt{fail} \mid s = t \mid A \mid \texttt{\+}\ G \mid (G,H) \mid (G;H) \mid \texttt{some}\ x\ G$$

其中s 和t 是两个项，x 是一个变量，而A 是一个原子目标。 $\mathscr{L}$的目标具有 ISO-Prolog [13] 所规定的操作语义，并假定采用出现检查。

$\hat{}\mathscr{L}$ 是 LPTP 的规范语言。对于每个用户定义的谓词符号R，$\hat{}\mathscr{L}$ 不包含R，而是包含三个与R具有相同元数的谓词符号R^s、R^f、R^t，分别表示R的成功、失败和终止。 $\mathscr{L}$还包含一个用于表示基项性的 一元约束

gr，表示其参数是基项。$\hat{}\mathscr{L}$的公式为：

$$\phi, \psi ::= \top \mid \bot \mid s = t \mid R(\vec{t}) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\phi \mid \exists x\phi$$

其中 $\vec{t}$ 是由n 个项组成的序列，而R 表示$\hat{}\mathscr{L}$的一个n元谓词符号。$\hat{}\mathscr{L}$的语义是经典逻辑的一阶谓词演算。

对于逻辑程序P中任意由用户定义的逻辑过程R，$D^P_R(\vec{x})$表示其克拉克当且仅当完备定义，参见 [5, 19]。

为定义逻辑程序的声明式语义，Stärk 使用三个句法算子**S**、**F**$\hat{}$ 和**T**，它们将 $\mathscr{L}$ 中的目标映射为 $\mathscr{L}$公式。直观而言，**S***G* 表示*G* 成功（对*G* 进行任何广度优先求值都会成功），**F***G* 表示*G* 失败（ISO-Prolog 求值停止且没有任何答案），而**T***G* 表示*G* 终止（ISO-Prolog 求值产生有限个答案后停止）。这些算子的定义如下：

$$\mathbf{S}R(\vec{t}) := R^s(\vec{t}) \quad \mathbf{S}\,\texttt{true} := \top \quad \mathbf{S}\,\texttt{fail} := \bot \quad \mathbf{S}(s = t) := (s = t)$$
$$\mathbf{S}\backslash{+}G := \mathbf{F}G \quad \mathbf{S}(G,H) := \mathbf{S}G \wedge \mathbf{S}H \quad \mathbf{S}(G;H) := \mathbf{S}G \vee \mathbf{S}H \quad \mathbf{S}(\texttt{some}\ x\ G) := \exists x \mathbf{S}G$$

FR($\vec{t}$*) :=R f (*$\vec{t}$*) F* 真 *:=* ⊥*F* 失败 *:=* ⊤*F(s =t) :=* ¬*(s =t) F*|*+G :=SG F(G, H) :*
=FG ∨*FH F(G; H) :=FG* ∧*FH F(some x G) :=* ∀*xFG*

TR($\vec{t}$ *) :=R*^{t}*(*$\vec{t}$*) T true :=* ⊤*T fail :=* ⊤ *T(s =t) :*
= ⊤*T*|*+G :=TG* ∧*gr(G) T(G,H) :=TG* ∧*(FG* ∨
TH) T(G; H) :=TG ∧*TH T(some x G) :=* ∀*xTG*

请注意，终止性要求否定的安全使用，参见**T**\+*G* 的定义，其中目标*G* 必须在证明时被证明为终止且为基项。最后，我们补充*gr*的定义，它属于规范语言，且定义**T**\+*G*时需要用到：

gr(true) := ⊤ *gr((G,H)) :=gr(G)* ∧*gr(H) gr(fail) :=* ⊤ *gr((G;*
H)) :=gr(G) ∧*gr(H) gr(s =t) :=gr(s)*∧*gr(t) gr(some x G) :=* ∃
x gr(G) gr(R(t1, . . . ,tn)) :=gr(t$_1$*)* ∧ *. . .* ∧*gr(tn) gr(*|*+G) :=gr(G)*

关于 LPTP 的完整介绍，请读者参阅 Stärk 的论文 [26, 27, 28, 29]。

3 将 LPTP 公理编译为一阶形式

借助 LPTP，我们证明逻辑程序 *P* 相对于其归纳扩展IND(*P*)的性质。该归纳扩展包含 Clark完备化 [5]，以及沿谓词定义进行的归纳。Stärk 表明，归纳扩展总是一致的，并证明了相对于 Prolog [29] 操作语义的若干正确性和完备性结果。一阶理论 IND(*P*)（参见 [29], 第 253–254 页）由九个公理模式定义，我们现在将对其进行介绍，并说明它们如何转换为一阶形式。我们省略内置谓词的不动点公理。还需指出，LPTP 的规范语言 $\mathscr{L}$ 可以在逻辑层面扩展新的函数符号和谓词符号（没有与之关联的 Prolog 代码）。如 [21], 所示，这类函数和谓词定义也可以编译为一阶形式。

3.1 入门

我们以以下逻辑程序 ADD 作为运行示例。

```
nat(0). nat(s(X)) :- nat(X).
```

```
add(0,Y,Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

我们讨论 Stärk 提出的公理，并将其应用于 ADD 程序。

Clark 等式理论的公理

1. f (x1, . . . , x_n) = f (y1, . . . , y_n) → x_i = y_i [，如果 f 是 n 元函数且 1 ≤ i ≤ n]2. f (x1, . . . , x_n) ≠ g(y1, . . . , y_m) [，如果 n ≠ m 或 f ≠ g]3. t ≠ x [，如果 x 出现在 t 中且 t ≠ x]

前两个公理规定了由从待考察程序中提取的函数符号构造出的树的一些性质。第三个公理禁止无限树。请注意，它实际上是一个公理模式，即一组无限的一阶公理。我们将省略它，但仍能保证可靠性。以下是 FOF 版本（FOF 语法见第 2 节）：

```
fof(id1,公理,! [Xx4]: ! [Xx5] : (s(Xx4) = s(Xx5) => Xx4 = Xx5)). fof(id2,公
理,! [Xx3]: ~ ('0' = s(Xx3))).
```

gr/1 的公理

4. gr(c) [当c 是一个常量时]

5. gr(x_1) ∧ . . . ∧ gr(x_m) ↔ gr(f($x1$,...,x_m)) [f是 m元的]

实际上，LPTP 处理非基项项，并提供一个预定义谓词$gr/1$，我们可以将其视为约束。这个关系在处理失败即否定时很有用，例如，LPTP 只允许对基项目标使用失败即否定（参见定义**T**\+ G）。回到我们的例子，以下是 FOF 版本：

```
fof(id4,公理,gr('0')). fof(id5,公理,! [Xx6] : (gr(Xx6) <=>
gr(s(Xx6)))).
```

ADD 程序包含两个用户定义谓词：add/3 和 nat/1。LPTP 从三个角度考察每个用户定义谓词：失败、成功和终止。因此，LPTP 会创建以下谓词：add_fails/3、add_succeeds/3、add_terminates/3；nat/1 也同样如此。这三个视角通过以下公理相互关联，其中 R^s（分别为 R_f和 R^t）表示 R_- succeeds/3（分别表示 R_fails/3 和 R_terminates/3）。

唯一性公理和全称性公理

6. ¬(R^s($\vec{x}$) ∧ R^f($\vec{x}$)) [当R是用户定义谓词时]7. R^t($\vec{x}$) → (R^s($\vec{x}$) ∨ R_f($\vec{x}$)) [当R是用户定义谓词时]

公理 6 表明，对于任意一组（可能不是基项的）项，R 不可能同时成功和失败。公理 7 指出，若已终止，则必定成功或失败。总而言之，对于任意一组项 $\vec{x}$，假设其已终止，则 R($\vec{x}$) 要么成功，要么（且仅能）R($\vec{x}$) 失败。因此，对于我们的例子，可得：

fof(ida6,公理,! [Xx7,Xx8,Xx9] :~ ((add_成功(Xx7,Xx8,Xx9) & add_失败(Xx7,Xx8,Xx9)))). fof(ida7,公理,! [Xx7,Xx8,Xx9] : (add_终止(Xx7,Xx8,Xx9) =>(add_成功(Xx7,Xx8,Xx9) |add_失败(Xx7,Xx8,Xx9)))). fof(idn6,公理,! [Xx10] :~ ((nat_成功(Xx10)& nat_失败(Xx10)))). fof(idn7,公理,! [Xx10] : (nat_终止(Xx10) =>(nat_成功(Xx10) |nat_失败(Xx10)))).

用户定义谓词的不动点公理R

$$8.R^s(\vec{}x)\leftrightarrow \mathbf{S}D_R^P(\vec{}x),R\,f(\vec{}x)\leftrightarrow \mathbf{F}D_R^P(\vec{}x),R^t(\vec{}x)\leftrightarrow \mathbf{T}D_R^P(\vec{}x)$$

我们回顾一下，$D_R^P(\vec{}x)$ 表示逻辑程序 P 中用户定义过程 $R(\vec{}x)$ 的完备化定义 [5]。上一节中，我们介绍了如何将算子 **S**、**F** 和 **T** 应用于公式。例如，第一个等价式 $R^s(\vec{}x)\leftrightarrow \mathbf{S}D_R^P(\vec{}x)$ 定义了 $R^s(\vec{}x)$。回到运行示例，我们得到：

fof(idns8,公理,! [Xx1] : (自然数_succeeds(Xx1) <=>(? [Xx2] : (Xx1 = s(Xx2) & 自然数_succeeds(Xx2)) | Xx1 = '0'))). fof(idnf8,公理,! [Xx1] : (自然数_fails(Xx1) <=>(! [Xx2] : (~(Xx1 = s(Xx2)) |自然数_fails(Xx2)) & ~ (Xx1 = '0')))). fof(idnt8,公理,! [Xx1] : (自然数_terminates(Xx1) <=>(! [Xx2] : ((~ (Xx1 = s(Xx2)) | 自然数_terminates(Xx2)))))).

add/3 也同理。

最后，对于任何形式为 $\forall\vec{}x\,[R^s(\vec{}x)\rightarrow\phi(\vec{}x)]$, 的性质，其中 $R(\vec{}x)$ 是用户定义过程，而 $\phi(\vec{}x)$ 是一个 $\hat{}\mathscr{L}$-公式，我们都有一个归纳模式。交互式证明器 LPTP 能够在用户与其交互时，按需动态地生成归纳公理。采用我们的方法，如果待证猜想可以轻松改写为所需形式，我们就从该猜想静态地生成归纳公理，且只生成一次。这可能会导致不精确性，但我们仍能保证可靠性。让我们来看一个简单的例子。这正是使用 LPTP 时发生的情况，它对 [29] 做了轻微推广。通过下框中的直接递归用户定义谓词，我们排除了相互递归定义。当然，LPTP 能处理相互递归性质，[26] 中给出了一些例子。

用户定义谓词R的（简化）归纳模式

设 R 是一个直接递归的用户定义谓词，并设 $\phi(\vec{x})$ 是一个 $\mathscr{L}$ 公式，使得 $\vec{x}$ 的长度等于R的元数。
设 $sub(\phi(\vec{x})/R)$ 是待证明的公式 $\forall\vec{x}(R^s(\vec{x}) \rightarrow \phi(\vec{x}))$。
设 $closed(\phi(\vec{x})/R)$ 是由 $\forall\vec{x}(\mathbf{S}D^D_R(\vec{x}) \rightarrow R^s(\vec{x}))$ 替换后得到的公式

- $R_s(\vec{x})$ 根据 $\phi(\vec{x})$，替换 $\rightarrow$ 右侧的
- 所有出现在$R(\vec{t})$ 左侧的出现项，通过 $\rightarrow \phi(\vec{t}) \wedge R(\vec{t})$.

那么，归纳公理就是以下公式：

9. 闭合$(\phi(\vec{x})/R) \rightarrow sub(\phi(\vec{x})/R)$

我们将此公理应用于以下性质，其非正式表述为：对于任意项 x，若 nat(x) 成立，则 add(x,0,x)。用 LPTP 表示，即：对于任意项 x，若 nat_succeeds(x) 成立，则 add_- succeeds(x,0,x)，这恰好就是公理 9 的公式 $sub(\phi(\vec{x})/R)$。因此，$R \equiv$nat, $R^s \equiv$nat_- succeeds 且 $\phi(\vec{x}) \equiv$add_succeeds(x, 0, x)。

对于公理 9 的左侧，我们从以下内容开始：

$$\forall x(\mathbf{S}D^{ADD}_{nat}(x) \rightarrow \texttt{nat_succeeds}(x))$$

我们有 D^{ADD}_{nat}(x) $\equiv$ x $= 0 \vee \exists$y(x $=$ s(y)$\wedge$nat(y))。我们将 nat(y) 替换为 nat(y)$\wedge$add_succeeds(y, 0, y)。我们将 nat_succeeds(x) 替换为 add_succeeds(x, 0,x)。得到： $\forall$x (S [x $= 0 \vee \exists$y(x $=$ s(y) $\wedge$nat(y)$\wedge$add_ succeeds(y,0, y))] $\rightarrow$add_succeeds(x, 0, x))。应用 S 后得到： $\forall$x([x $= 0 \vee \exists$y(x $=$ s(y) $\wedge$nat_succeeds(y) $\wedge$ add_succeeds(y,0, y))] $\rightarrow$add_succeeds(x, 0,x))。

总结而言，在一阶形式中，与待证明性质相关联的内容为：

```
fof(lemma,conjecture,
! [Xx] : (自然数_succeeds(Xx) => add_succeeds(Xx,'0',Xx))).
```

我们得到如下归纳公理：

```
fof(induction,axiom,
    ! [Xx] :
        ((? [Xx2] : (Xx = s(Xx2) & (nat_succeeds(Xx2)
                                    & add_succeeds(Xx2,'0',Xx2)))
        | Xx = '0') => add_succeeds(Xx,'0',Xx))
=>
    ! [Xx] : (自然数_succeeds(Xx) => ADD_succeeds(Xx,'0',Xx)))).
```

我们可以将全部 15 条公理（包括定义 add_success/3、add_fails/3 和 add_terminates/3 的公理）、猜想及其归纳公理汇集到一个文件中，例如 test.fof，然后提交给 E 定理证明器或 Vampire。在标准笔记本电脑上，这两个系统都能在不到一秒的时间内找到反驳。

这使我们能够得出结论：对于任意项 x，若 nat(x) 成立，则 add(x,0,x) 为真。在操作层面，对于任意自然数 n，在与目标 add(s^n(0),0,s^n(0)) 对应的 Prolog 搜索树中，会出现空子句。假设程序会终止（这一点将在后文证明），这意味着用户使用任何 ISO-Prolog 系统执行查询 :- add(s^n(0),0,s^n(0)).时，至少会得到一个肯定答案。

以下是该性质在 LPTP 版本中的手动证明（一个 Prolog 文件），随后是由 LPTP 生成的 TEX 版本。使用 interactiveLPTP Emacs 模式，我们通过调用

ind 策略开始了这项证明，要求进行归纳证明。基础情形和归纳情形均由 LPTP 自动生成并完成。

:- lemma(add:x_0_x, all [x]: succeeds nat(?x) => succeeds add(?x,0,?x),
归纳([all x: succeeds nat(?x) => succeeds add(?x,0,?x)] ,[step([] , [] , [] ,
succeeds add(0,0,0)), step([x] , [succeeds add(?x,0,?x), succeeds nat(?x)
] , [] ,succeeds add(s(?x),0,s(?x)))])).

引理 [*add:x_0_x*] $\forall x$ (**S** 自然数(x) $\rightarrow$ **S** add(x,0, x))。

证明。
归纳$_0$: $\forall x$(**S** 自然数(x) $\rightarrow$ **S**
add(x,0,x)). 假设$_1$: 无。结论$_1$: **S**
add(0,0,0). 假设$_1$: **S** add(x,0,x) 且 **S** 自
然数(x). 结论$_1$: **S** add(s(x),0,s(x)). □

3.2 第二个性质

现在考虑以下性质：对于任意x、y和z，若 nat(x)、nat(y) 且 add($s(x)$,y,z)，则有 add(x,$s(y)$,z)。我们先将前述性质断言为公理，自动定理证明器便可自由使用它；接着得到新的猜想：

fof('lemma-(add:x_0_x)',公理 ,
! [Xx] : (自然数_succeeds(Xx) => ADD_succeeds(Xx,'0',Xx))).

fof('lemma-(add:succ)',猜想, ! [Xx,Xy,Xz] : (((nat_
succeeds(Xx) & nat_succeeds(Xy)) & add_
succeeds(s(Xx),Xy,Xz))=> add_succeeds(Xx,s(Xy),Xz))).

为了针对该性质生成归纳公理，我们首先将其改写为形式 $\forall \vec{x}\ [R^s(\vec{x}) \rightarrow \phi(\vec{x})$，并对用户定义谓词应用简化归纳模式。得到：fof(induction,axiom,(

! [Xx]: ((? [Xy25] : (Xx = s(Xy25) & (自然数_succeeds(Xy25) & ! [
Xy,Xz] : ((ADD_succeeds(s(Xy25),Xy,Xz) & 自然数_succeeds(Xy))=>
ADD_succeeds(Xy25,s(Xy),Xz))))| Xx ='0') =>! [Xy,Xz] : ((ADD_
succeeds(s(Xx),Xy,Xz) & 自然数_succeeds(Xy))=> ADD_
succeeds(Xx,s(Xy),Xz)))=> ! [Xx] : (自然数_succeeds(Xx)=> ! [Xy,Xz] :
((ADD_succeeds(s(Xx),Xy,Xz) & 自然数 _succeeds(Xy))=> ADD_
succeeds(Xx,s(Xy),Xz))))).

同样，我们可以将所有公理、猜想及其归纳公理汇集到一个文件中，并提交给 Vampire；它会在约一分钟内找到反驳。

3.3 皮亚诺加法的交换律

现在我们已经可以考虑皮亚诺加法的交换性：对于任意 x、y、z，如果 add(x，y，z) 成立，那么 add(y，x，z) 也成立。当然，这样表述的性质并不成立。我们需要确保 x 和 y 都是皮亚诺数。因此，我们先将前面两个性质作为公理加入。以下是与其归纳公理相对应的新猜想：

```
fof('theorem-(add:commutative)',猜想,！［Xx,Xy,Xz］:(((自然
数_succeeds(Xx) & 自然数_succeeds(Xy)) & add_
succeeds(Xx,Xy,Xz))=> add_succeeds(Xy,Xx,Xz))).

fof(归纳,公理,
(！［Xx］:((？［Xy26］:(Xx =s(Xy26) & (自然数_succeeds(Xy26) & ！［
Xy,Xz］:((add_succeeds(Xy26,Xy,Xz)& 自然数_succeeds(Xy))=> add_
succeeds(Xy,Xy26,Xz))))∥ Xx ='0') =>！［Xy,Xz］:((add_
succeeds(Xx,Xy,Xz)& 自然数_succeeds(Xy))=> add_
succeeds(Xy,Xx,Xz)))

=>
！［Xx］:(自然数_succeeds(Xx) =>！［Xy,Xz］:((add_
succeeds(Xx,Xy,Xz)& 自然数_succeeds(Xy))=> add_
succeeds(Xy,Xx,Xz))))).
```

Vampire 在不到一秒的时间内证明了这个猜想。

3.4 一些终止证明

最后，让我们证明一些关于 add/3 的终止性质。显然，当 nat(x) 或 nat(z) 成立时，add(x，y，z) 的 Prolog 证明会终止。我们通过陈述两个引理来证明这一点，并将它们汇总为一个定理。以下是 LPTP 性质及其证明（第二个证明从略）。

引理 [*add:*终止*:1*] $\forall x, y, z\ (\mathbf{S}\text{自然数}(x) \rightarrow \mathbf{T}\,\mathtt{add}(x, y, z))$。 **证明**。

归纳$_0$: $\forall x(\mathbf{S}\text{自然数}(x) \rightarrow \forall y, z\,\mathbf{T}\,\mathtt{add}(x, y, z)$。

假设$_1$：无。结论$_1$：$\forall y, z\,\mathbf{T}\,\mathtt{add}(0, y, z)$。假设$_1$：$\forall y, z\,\mathbf{T}\,\mathtt{add}(x, y, z)$，且$\mathbf{S}\,\mathtt{nat}(x)$。结论$_1$：$\forall y, z\,\mathbf{T}\,\mathtt{add}(\mathtt{s}(x), y, z)$。

□

引理 [*ADD:*项*:3*] $\forall x, y, z\ (\mathbf{S}\text{自然数}(z) \rightarrow \mathbf{T}\,\mathrm{ADD}(x, y, z))$。 **证明**。 类似。 □

定理 [*ADD:*项] ∀*x, y, z* (**S** 自然数(*x*) ∨**S** 自然数(*z*) →**T** ADD(*x, y, z*))。 **证明**。
假设0：S 自然数(x) ∨S 自然数(z)。情形1：S 自然数(x)。
根据引理 1 [add:项:*1*]，T add(x, y,z)。情形1：S 自然数
(z)。根据引理 2 [add:项:*3*]，T add(x, y, z)。因此1，在所
有情形下：T add(x, y, z)。故0：S 自然数(x) ∨S 自然数
(z) →T add(x, y, z)。 □

Vampire 在不到一秒的时间内证明了这三个陈述中的每一个。我们的编译器会为每个引理生成一个归纳公理实例，而不会为定理生成。例如，下面是第一个猜想及其归纳公理：

fof('lemma-(add:term:1)',猜想,！ [Xx,Xy,Xz] : (自然数_succeeds(Xx) => add_
terminates(Xx,Xy,Xz))).

fof(induction,axiom,(！ [Xx]: ((? [Xx2] : (Xx =s(Xx2) & (nat_成功
(Xx2) &！ [Xy,Xz] :add_终止(Xx2,Xy,Xz)))| Xx ='0')=>！ [Xy,Xz] :
add_ 终止 (Xx,Xy,Xz))=>

！ [Xx] : (nat_成功(Xx) =>！ [Xy,Xz]: add_终止(Xx,Xy,Xz)))).

4 实验结果

我们将前几节介绍的模式应用于 LPTP 提供的各种程序库，现总结如下。程序库 nat 定义了一些具有预期性质的基本 Peano关系。程序库 ack 定义了 Ackermann函数的关系版本，并具备三项性质（见下文）。程序库 gcd 定义了最大公约数关系的一个版本，并附有完整的正确性证明。程序库 int 定义了整数。程序库 list 提供了一些关于列表的基本关系及其性质。程序库 suffix 定义了子列表关系的两个版本：一个将其定义为后缀的前缀，另一个将其定义为前缀的后缀，并证明了这两个版本在终止、成功和失败方面等价。类似地，程序库 reverse 定义了逆序关系的两个经典版本：一个使用追加关系，另一个使用累加器，并证明了二者完全等价。程序库 permutation 定义了置换关系及一些有用的性质，用于证明程序库 sort 和 mergesort 中所定义的排序算法的正确性。程序库 taut 定义了一个用于命题公式的重言式检查器，并附有其正确性证明（详细说明见 [27] ）。

我们如何处理这类文件？给定 LPTP 库中的一个程序，我们首先列出尝试证明其关联的 LPTP 证明文件中所列性质时所需的条件。所需条件包括逻辑程序 *P* 及其关联的 LPTP 证明文件。如果 *P* 依赖其他逻辑程序，我们必须将它们也包括在内。如果关联的 LPTP 证明文件使用了其他证明文件，我们也必须将它们包括在内。我们假设不存在循环性，例如在尝试证明某个引理之前就先假定该引理成立。我们使用这些条件来构建目标逻辑程序 *P′* 和目标 LPTP 证明文件。然后将 *P′* 编译为 IND(*P′*) 的 FOF 版本。每个事实（即引理、推论或定理）都被编译为一个 FOF 猜想（可能附带其归纳公理），并存储在单个文件中。此类文件还包含逻辑理论

IND(*P′*) 被编译为 FOF 公理。先前处理过的 FOF 猜想也会被转换为 FOF 公理。因此，初始 LPTP 证明文件中的每个事实都会对应生成一个 FOF 文件。最后，E定理证明器和 Vampire 都会对每个 FOF 文件运行，所用命令分别为 vampire –mode casc -m 16384 –cores 7 -t $TO $FILE 和 eprover –delete-bad-limit=2000000000–definitional-cnf -s –auto-schedule=8 –proof-object –cpu-limit=$TO $FILE。

我们将结果汇总于表1。第一列列出库的名称。第二列列出相应证明文件中的（引理/推论/定理）数量。其余九列可分为三组。在一台配备 8 核 M2 芯片、24 GB 内存并运行 macOS Sonoma 的 MacBook Pro 上，第一组给出了 E 定理证明器（E-1s 列）、Vampire（V-1s 列）以及两种证明器组合（EV-1s 列）在 1 秒超时设置下的成功率。第二组（第三组）给出了在 10 秒（60 秒）超时设置下的成功率。

lib	#	E-1s	V-1s	EV-1s	E-10s	V-10s	EV-10s	E-60秒	V-60秒	EV-60秒
nat	91	70%	88%	88%	76%	95%	95%	78%	97%	97%
gcd	11	45%	45%	45%	45%	45%	45%	45%	45%	45%
ack	3	33%	33%	33%	33%	33%	33%	33%	33%	33%
int	67	76%	82%	87%	79%	88%	90%	79%	91%	91%
list	84	75%	94%	94%	80%	96%	96%	81%	99%	99%
后缀	31	94%	100%	100%	94%	100%	100%	97%	100%	100%
反向	25	72%	88%	88%	84%	100%	100%	84%	100%	100%
排列。	42	48%	71%	71%	60%	79%	81%	62%	86%	86%
sort	42	45%	62%	62%	50%	74%	74%	55%	76%	76%
合并。	24	79%	88%	88%	79%	92%	92%	79%	100%	100%
taut	43	65%	81%	81%	70%	84%	84%	74%	84%	84%

表1：实验评估

下面对这些结果作一些说明。gcd Prolog 文件包含谓词 gcd/3 和 gcd_leq/3 的相互递归定义。目前，我们的翻译模式尚不支持证明此类定义的性质。

ack 证明文件包含以下三个性质。第一个已成功验证。后两个无法通过我们的简化归纳模式证明。事实上，LPTP 证明在顶层归纳内部使用了归纳，这超出了我们的翻译模式所支持的范围。

引理 [*ackermann:types*] $\forall m, n, k\ (\mathbf{S}\,\mathtt{ackermann}(m, n, k) \wedge \mathbf{S}\,\text{自然数}(n) \rightarrow \mathbf{S}\,\text{自然数}(k))$。

引理 [*ack:existence*] $\forall m, n\ (\mathbf{S}\,\text{自然数}(m) \wedge \mathbf{S}\,\text{自然数}(n) \rightarrow \exists k\ \mathbf{S}\,\text{阿克曼函数}(m, n, k))$。

引理 [*ack:termination*] $\forall m, n, k\ (\mathbf{S}\,\text{自然数}(m) \wedge \mathbf{S}\,\text{自然数}(n) \rightarrow \mathbf{T}\,\text{阿克曼函数}(m, n, k))$。

5 相关工作

Prolog验证框架为数不少，例如 [7, 10, 2, 24] ，以及较新的 [8]。它们大多旨在支持纸笔 证明。尽管这些框架可能提供有趣而优雅的方法，但证明的有效性依赖于通常以自然语言书写的数学写作，而且证明并未

自动验证。在我们看来，与本文介绍的一键式方法相比，撰写和验证这类手写证明既耗时又容易出错。

最近，有一些颇有意思的工作报告了在现代一阶定理证明器中加入数据类型、考虑其取值的无环性，以及采用归纳的方法，参见例如 [4, 12]。我们尚未在自己的框架中测试这些扩展。

对于答案集编程（一种采用 Prolog 语法、面向知识表示和搜索问题的声明式规范语言），[9] 介绍了一种验证方法，其中 Vampire 会检查答案集程序的等价性。

有些程序设计语言在设计时便内置了自动验证工具。例如，Dafny[17] 大量采用 SMT 求解。Why3 系统 [11] 允许将验证条件导出至众多自动定理证明器和交互式定理证明器。

关于自动与交互式定理证明集成的早期论述见 [1]。正如引言中所述，如今大多数交互式定理证明器都支持运行一些自动定理证明器。以 Isabelle 为先例， [20, 23, 3, 22], 自动化工具也可见于 ACL2, [14], Coq、 [6] 和 Lean [18]等系统中。

6 结论

让我们回顾一下引言中的问题，并在这项实验之后给出我们的答案：

- 我们能否也使用 TPTP 一阶形式 世界语 来表述 Stärk 与逻辑程序相关联的逻辑理论？可以。我们没有实现一种公理模式：禁止有理项的公理 3。另一种则只实现了部分功能：用于归纳的公理 9。事实上，采用我们的方法，无法在归纳证明中进行归纳论证。虽然精确性有所损失，但这两种情况下我们的方法仍然是可靠的。

- 我们能否使用现成的 TPTP 证明器，并在合理的时间内获得自动证明？可以。我们使用 Vampire 和 E 定理证明器的最基本选项，本质上就是设置超时。尽管 Vampire 似乎能更快找到反驳，但有时 E 定理证明器能找到证明，而 Vampire 却无法在时限内得出结论。因此，这两种证明器各有所长。目前，我们尚未尝试使用证明器提供的高级功能，例如 [15] 中提出的功能。

- 采用这种方法，我们能否获得可接受的成功率？可以。E 定理证明器和 Vampire 并行运行时，在标准笔记本电脑上设置一分钟超时，我们的基准测试平均成功率约为 83%。

与使用交互式定理证明器手动、费力地推导某些证明所耗费的精力相比，使用最先进的自动定理证明器显然能节省时间。对于这次初步实验能取得如此高的成功率，我们原本并未抱有期待。我们认为，这可以由多种原因解释。显然，如今笔记本电脑的计算能力十分强大，而且自动定理证明器也得到了长足改进。此外，多亏了 Stärk 的理念，LPTP 所针对的 Prolog 纯子集以及 LPTP 规范语言（本质上是一阶逻辑）都具有简洁明了的语义，因此可以直接转换为一阶形式。最后但同样重要的是，Stärk 巧妙的证明方法——将 LPTP 库中性质的证明拆分为易于处理的引理——无疑也对我们取得的成功率产生了影响。

最后，本文所呈现的工作仍有改进空间。根据 [3]，可以将其视为迈向 LPTP 自动化证明工具的初步尝试。自动化证明工具的第一步——前提选择器，用于筛选 LPTP 库中可能对证明有用的子部分——以及第三步——证明

重构模块，用于将自动证明器找到的证明改写为 LPTP 证明格式——尚待研究。

致谢。感谢 Manuel Hermenegildo、Daniel Jurjo、Pedro López-Garcia 和 Jose Morales 就 LPTP 展开富有启发性的讨论，感谢 Geoff Sutcliffe 在 TPTP 方面提供的帮助，也感谢审稿人提出的建设性意见。

参考文献

[1] W. Ahrendt, B. Beckert, R. Hähnle, W. Menzel, W. Reif, G. Schellhorn & P. Schmitt (1998年)：整合自动与交互式定理证明。《自动演绎——应用基础：第二卷：系统与实现技术》，第 97–116 页。Springer。[2] K. R. Apt & E. Marchiori (1994年)：关于 Prolog 程序的推理：从模式到类型，再到断言。《计算机与形式化》6(6A)，第 743–765 页，doi:10.1007/BF01213601。[3] J. C. Blanchette, C. Kaliszyk, L. C. Paulson & J. Urban (2016年)：迈向 QED 的自动化证明。《形式化推理杂志》9(1)，第 101–148 页，doi:10.6092/ISSN.1972-5787/4593。[4] J. C. Blanchette, N. Peltier & S. Robillard (2018年)：带数据类型和余数据类型的超位置。载于 D. Galmiche, S. Schulz & R. Sebastiani 编：《自动推理——第九届国际联合会议 IJCAR 2018，作为联合逻辑会议 FloC 2018 的一部分举行，英国牛津，2018年7月14日至17日，论文集》，《计算机科学讲义》10900，Springer，第 370–387 页，doi:10.1007/978-3-319-94205-6 25。[5] K. L. Clark (1977年)：失败即否定。载于 H. Gallaire & J. Minker 编：《逻辑与数据库：逻辑与数据库研讨会》，图卢兹研究中心，法国，1977年，《数据库理论进展》，Plemum Press，纽约，第 293–322 页，doi:10.1007/978-1-4684-3384-5_11。[6] L. Czajka, B. Ekici & C. Kaliszyk (2018年)：用 Coq 和 CoqHammer 实现具体语义。载于 F. Rabe, W. M. Farmer, G. O. Passmore & A. Youssef 编：CICM，《计算机科学讲义》11006，Springer，第 53–59 页，doi:10.1007/978-3-319-96812-4 5。[7] P. Deransart (1993年)：确定子句程序声明性性质的证明方法。《理论计算机科学》118(2)，第 99–166 页，doi:10.1016/0304-3975(93)90107-5。[8] W. Drabent (2016年)：逻辑程序的正确性与完备性。《ACM 计算逻辑汇刊》17(3)，第 18 页，doi:10.1145/2898434。[9] J. Fandinno, V. Lifschitz, P. Lühne & T. Schaub (2020年)：使用 Anthem 和 Vampire 验证紧逻辑程序。《逻辑程序理论与实践》20(5)，第 735–750 页，doi:10.1017/S1471068420000344。[10] G. Ferrand & P. Deransart (1993年)：正规逻辑程序部分正确性与弱完备性的证明方法。《逻辑程序杂志》17(2/3&4)，第 265–278 页，doi:10.1016/0743-1066(93)90033-D。[11] J.-C. Filliâtre & A. Paskevich (2013年)：Why3——程序与证明器的交汇之处。载于 M. Felleisen & P. Gardner 编：ESOP，《计算机科学讲义》7792，Springer，第 125–128 页，doi:10.1007/978-3-642-37036-6_8。[12] M. Hajdú, P. Hozzová, L. Kovács & A. Voronkov (2021年)：超位置中的递归定义归纳。载于：《计算机辅助设计中的形式化方法》，FMCAD 2021，美国康涅狄格州纽黑文，2021年10月19日至22日，电气电子工程师学会，第 1–10 页，doi:10.34727/2021/ISBN.978-3-85448-046-4_34。[13] ISO/IEC 13211-1 (1995年)：信息技术——程序设计语言——Prolog——第 1 部分：通用核心。[14] S. J. C. Joosten, C. Kaliszyk & J. Urban (2014年)：在 ACL2 问题上使用 TPTP 风格自动定理证明器的初步实验。载于 F. Verbeek & J. Schmaltz 编：ACL2 国际研讨会，EPTCS 152，第 77–85 页，doi:10.4204/EPTCS.152.6。

[15] L. Kovács, S. Robillard & A. Voronkov (2017)：应对量化推理。载于 G. Castagna & A. D. Gordon 编：POPL 2017，ACM，第 260–270 页，doi:10.1145/3009837.3009887。[16] L. Kovács & A. Voronkov (2013年)：一阶定理证明与 Vampire。载于 N. Sharygina & H. Veith 编：CAV 2013，《计算机科学讲义》8044，Springer，第 1–35 页，doi:10.1007/978-3-642-39799-8_1。[17] K. R. M. Leino (2012)：使用 Dafny 开发经验证程序。载于 R. Joshi, P. Müller & A. Podelski 编：VSTTE，《计算机科学讲义》7152，Springer，第 82 页，doi:10.1007/978-3-642-27705-4_7。[18] P. Lippe (2019)：Lean Hammer。https://github.com/phlippe/Lean_hammer。访问日期：2025年4月。[19] J. W. Lloyd (1987)：《逻辑程序设计基础》，第 2 版。Springer，doi:10.1007/978-3-642-83189-8。[20] J. Meng & L. C. Paulson (2004)：使用归结支持交互式证明的实验。载于 D. A. Basin & M. Rusinowitch 编：《自动推理——第二届国际联合会议 IJCAR 2004，爱尔兰科克，2004年7月4日至8日，论文集》，《计算机科学讲义》3097，Springer，第 372–384 页，doi:10.1007/978-3-540-25984-8_28。[21] F. Mesnard, T. Marianne & É. Payet (2024)：用于 Prolog 验证的自动定理证明。载于 N. S. Bjørner, M. Heule & A. Voronkov 编：LPAR 2024 补充卷，《计算领域 Kalpa 出版物》18，EasyChair，第 137–151 页，doi:10.29007/C25R。[22] L. C. Paulson (2022)：Sledgehammer：一些历史和技巧。https://lawrencecpaulson.github.io/2022/04/13/Sledgehammer.html。访问日期：2025年4月。[23] L. C. Paulson & J. C. Blanchette (2010)：使用 Sledgehammer 的三年经验：连接自动与交互式定理证明器的实用工具。载于 G. Sutcliffe, S. Schulz & E. Ternovska 编：IWIL，《计算领域 EPiC 系列》2，EasyChair，第 1–11 页，doi:10.29007/36DT。[24] D. Pedreschi & S. Ruggieri (1999)：逻辑程序验证。《逻辑程序杂志》39(1-3)，第 125–176 页，doi:10.1016/S0743-1066(98)10035-3。[25] S. Schulz, S. Cruanes & P. Vukmirovic (2019)：更快、更高、更强：E 2.3。载于 P. Fontaine 编：《自动演绎——CADE 27——第 27 届自动演绎国际会议》，巴西纳塔尔，2019年8月27日至30日，论文集，《计算机科学讲义》11716，Springer，第 495–507 页，doi:10.1007/978-3-030-29436-6 29。_[26] R. F. Stärk (1995)：带否定的纯 Prolog 程序的一阶理论。《数学逻辑档案》34(2)，第 113–144 页，doi:10.1007/BF01270391。[27] R. F. Stärk (1996)：逻辑程序的完全正确性：一种形式化方法。载于 R. Dyckhoff, H. Herre & P. Schroeder-Heister 编：ELP’96，《计算机科学讲义》1050，Springer，第 237–254 页，doi:10.1007/3-540-60983-0_17。[28] R. F. Stärk (1997)：逻辑程序的形式化验证：基础与实现。载于 S. I. Adian & A. Nerode 编：《计算机科学的逻辑基础》，第四届国际研讨会 LFCS’97，俄罗斯雅罗斯拉夫尔，1997年7月6日至12日，论文集，《计算机科学讲义》1234，Springer，第 354–368 页，doi:10.1007/3-540-63045-7_36。[29] R. F. Stärk (1998年)：LPTP（逻辑程序定理证明器）的理论基础。《逻辑程序杂志》36(3)，第 241–269 页，doi:10.1016/S0743-1066(97)10013-9。[30] G. Sutcliffe (2017)：TPTP 问题库及其相关基础设施——从合取范式到 TH0，TPTP v6.4.0。《自动推理杂志》59(4)，第 483–502 页，doi:10.1007/S10817-017-9407-7。[31] G. Sutcliffe (2023)：TPTP 世界中的逻辑语言。《IGPL 逻辑杂志》31(6)，第 1153–1169 页，doi:10.1093/JIGPAL/JZAC068。