arXiv:2602.20082v1 [cs.PL] 23 Feb 2026

机器生成、机器检查的已验证编译器证明（经验报告）

ZOE PARASKEVOPOULOU, 雅典国立技术大学，希腊

我们报告了如下工作：使用智能体编程助手（Claude Code，由 Claude Opus 4.6 驱动），在有人工指导但无人工撰写证明的情况下，从零开始将一项规模可观的 Rocq 正确性证明机械化。该证明确立了面向 Rocq 的 CertiCoq 经验证的编译器中管理范式（ANF）变换的语义保持。CertiCoq 中与之密切相关的续延传递风格（CPS）变换此前已由人类专家历经数月证明正确 [Paraskevopoulou 和 Grover 2021]。我们以该证明为模板，指示大语言模型将证明技术适配到 ANF 环境；后者在若干重要技术方面与之不同。所得 ANF证明约包含 7,800 行 Rocq（多于 5,300 行的 CPS 证明），开发时间约为 96 小时。我们描述了该证明技术，并报告了借助大语言模型开发它的经验，讨论了该方法的优势与局限及其对经过验证的编译器构造的意义。

1 引言

近年来，大语言模型（LLM）取得了长足进展，彻底改变了软件开发。GitHub Copilot 等智能体编程助手 [GitHub 2025] 和 Claude Code [Anthropic 2025] 如今已被广泛用于生成和重构代码，早期报告也表明，大语言模型可以大幅加快软件开发 [Peng 等人 2023]。与此同时，大语言模型生成代码的正确性仍是一个亟待解决的问题。即使是经过认真手工编写的代码，也普遍存在软件错误；对于用户可能并不完全理解的大语言模型生成代码，风险更是有增无减。

这引出了一个问题：大语言模型能否不仅生成代码，还能为这些代码生成机器检查的正确性证明？如果可以，那么原则上，软件就能附带由独立证明检查器认证的形式化保证，而这些保证不受代码由人类还是大语言模型编写的影响。这一愿景——机器生成的代码，以及机器生成、机器检查的证明——为软件工程的未来指明了一个极具吸引力的方向，而且正变得越来越可行。

本文为这一方向贡献了一项案例研究。我们探究：在适当的模板和人工指导下，最先进水平的大语言模型能否在 Rocq 证明助手中构建一项规模可观的机械化证明。我们的案例研究基于一项具体且技术要求很高的编译器验证任务：证明 CertiCoq 经验证的编译器中行政范式变换的正确性 [Anand 等人 2017]。

案例研究。CertiCoq 将 Gallina（Rocq 证明器的规范语言）编译为 C [Savary Bélanger 等人 2019] 和 WebAssembly [Meier 等人 2025], , 通过一系列变换实现，其中相当大一部分已经过形式化验证 [Paraskevopoulou 和 Appel, 2019; Paraskevopoulou 等, 2021; Savary Bélanger 和 Appel 2017]。该编译流程使用 Meta-Coq [Sozeau 等人 2020] 作为前端，从 Gallina 中提取无类型中间表示，随后通过续延传递风格（CPS）变换或行政范式（ANF）变换，将其转换为函数式中间语言。Paraskevopoulou 和 Grover 提出了一种新颖的证明技术，将语法模拟论证与步索引逻辑关系相结合 [2021]，并用该技术证明了 CPS 变换的正确性 [Ahmed, 2006; Appel 和 McAllester 2001]。该证明在 Rocq 中机械化，共约 4,200 行，由人类专家历经数月完成。将同一证明技术应用于 ANF 变换则留待未来工作。

作者联系信息：Zoe Paraskevopoulou，雅典国立技术大学，希腊，zoepar@softlab. ntua.gr。

本文报告了一项实验：我们使用 Claude Code（一种用于软件开发的智能体大语言模型工具）来完成 ANF 正确性证明，作者则提供高层次指导、以自然语言解释协助推理（令人意外的是，只有少数情况需要这种协助），并审查生成的证明。作者没有直接编写 Rocq 证明；整个证明从零开始完成。实验总体上取得了成功。大语言模型生成了约 6,000 行 Rocq 证明，涵盖终止程序的核心语义保持定理、若干辅助引理以及各种支持性定义。大语言模型正确地发现，用于证明 CPS 变换保持发散行为的技术无法推广到 ANF 环境，并证明了一个反例。该证明沿用了 CPS 证明的总体结构，但尽管二者在结构上相似，由于 ANF 变换的性质不同，推理过程中仍存在重要的技术差异。作为 ANF 证明相较于 CPS 证明复杂度的经验证据，我们报告称，本文的一位作者此前曾尝试手动完成 ANF 证明，但由于复杂度和时间限制而搁置了这一工作。大语言模型机械化证明的进展通过频繁提交记录在一个公共代码仓库中，整个过程持续约 96 小时。

贡献。 本文的贡献如下：

- 我们给出了 CertiCoq 编译器中 ANF 变换的机械化正确性证明（第 4 节），采用 Paraskevopoulou 和 Grover [2021] 的 CPS正确性证明技术。正确性陈述涵盖了会终止的源程序的语义保持。
- 我们报告了使用大语言模型（Claude Code，由 Claude Opus 4.6 驱动）开发该证明的经验（第 5 节）。我们描述了实验设置、交互过程、遇到的挑战以及获得的经验。
- 我们讨论了大语言模型辅助证明开发对编程语言研究和经过验证的编译器构造所具有的更广泛影响（第 6 节）。

论文概述。 第2节概述了用于 CPS变换的证明技术。第 3 节介绍证明所需的技术设置，包括源语言和目标语言、它们的语义以及逻辑关系。第 4节介绍 ANF 变换及其正确性证明。第 5 节报告了使用大语言模型开发证明的经验，第 6 节讨论其影响和未来方向。第 7节介绍相关工作，第 8节总结全文。第 A 节呈现详细的证明开发统计数据，第B节则精选了对话记录中的用户提示。

2 CPS 证明技术

证明编译器变换正确，相当于证明编译后程序的可观察行为是源程序可观察行为的子集。对于确定性语言，这可以表示为前向模拟 [Leroy 2009]：每当源程序执行一步 $e \to e'$，编译后的程序便执行一步或多步，以到达 e'的翻译，即 $[e] \to^+ [e']$。有许多变换满足这一简单的交换图，但由于已充分研究的技术障碍，CPS变换并不属于其中之一 [Dargaye 和 Leroy 2007]。

设 $[e]_k$ 表示表达式 e 在当前续延 k下的CPS翻译， $\bar{v}$表示值的CPS翻译。CPS变换的应用情形可定义为$[v_1\ e_2]_k = (\lambda m.\ [e_2]_{\lambda n.\, m\, k\, n})\ \bar{v}_1$。外层的$(\lambda m.\ \ldots)\ \bar{v}_1$是一个管理性可归约式：变换所引入的、没有源对应项的 β-可归约式。

假设 $v_1\ e_2 \to v_1\ e_2'$，其中 $e_2 \to e_2'$。朴素模拟试图证明 $[v_1\ e_2]_k \to^+ [v_1\ e_2']_k$。收缩该管理性可归约式并对 $e_2 \to e_2'$ 应用归纳假设，会使目标变为 $[e_2']_{\lambda n.\, \bar{v}_1\, k\, n}$。但证明目标 $[v_1\ e_2']_k = (\lambda m.\ [e_2']_{\lambda n.\, m\, k\, n})\ \bar{v}_1$ 并非这一项：它是一个新的管理性可归约式，其收缩式恰好是我们所到达的项。我们需要反向步进才能闭合该图，而这是不可能的。朴素模拟失败。

另一个正交的障碍来自新绑定符名称。正如Paraskevopoulou 和 Grover [2021]所解释的，当 $e \to e'$ 是源语言中的一个归约步时，$[e]_k$ 可能会归约为一个与 α等价 $[e']_k$ 但在语法上并不相同的项，因为续延传递风格变换的两次调用可能使用不同的新名计数器。因此，必须在模拟图中加入 α等价关系。这虽然可行，但既繁琐又脆弱。

Paraskevopoulou 和 Grover的关键见解 [2021] 是用逻辑相关性取代目标层级上的归约关系。证明无需 $[e]_k$ 在每个源语言步骤之后都恰好归约为 $[e']_k$；$e \to e'$只需满足：

$$e \to e' \quad \Longrightarrow \quad [e]_k \lesssim [e']_k.$$

其中 $\lesssim$ 是定义在项上的标准逻辑关系，具有自反性和传递性（我们在第3.3节给出此处所用逻辑关系的形式化定义）。

回到该示例：我们并不寻找从 $[v_1\ e_2]_k$ 到 $[v_1\ e_2']_k$ 的归约序列，而是证明它们在逻辑上相关。该逻辑关系满足 *Reduce-App*：一个 β-可归约式与其收缩式在逻辑上相关（任一方向均可）。此外，α-等价的项在逻辑上相关，因此无需显式地推理 α-等价。再结合传递性与归纳假设，该链条得以干净地闭合：

$$\begin{aligned}
[v_1\ e_2]_k &= (\lambda m.\ [e_2]_{\lambda n.\, m\, k\, n})\ \bar{v}_1 \\
&\lesssim^i [e_2]_{\lambda n.\, \bar{v}_1\, k\, n} && \text{(Reduce-App)} \\
&\lesssim^i [e_2']_{\lambda n.\, \bar{v}_1\, k\, n} && \text{(IH)} \\
&\lesssim^i (\lambda m.\ [e_2']_{\lambda n.\, m\, k\, n})\ \bar{v}_1 && \text{(Reduce-App)} \\
&= [v_1\ e_2']_k
\end{aligned}$$

传递性吸收了两侧的管理性可归约式。无需向后归约：该关系能够透过管理性结构看到底层语义。

3 技术设置

在本节中，我们给出实验的技术设置，包括源语言与目标语言、它们的语义，以及证明中使用的逻辑关系。这些都属于 CertiCoq 编译器的组成部分。关于这两种语言语义的完整定义，请读者参阅 Paraskevopoulou 与 Grover [2021]。

3.1 源语言

源语言 λ_{DS}是一种无类型的、使用 de Bruijn 索引的按值调用 λ 演算。文法如图 1a 所示。变量是自然数；环境 v 是一个值序列，表达式 n 通过查找第 n 项来求值。该语言扩展了 let 绑定、不动点 fix(e)、构造器应用和模式匹配。为简洁起见，相互递归函数未在论文中展示，但在实际定义中得到支持。

λ_{DS} 的语义由一个大步求值关系给出，记作 v；$e \Downarrow^f_{\text{src}} \text{Res}(v)$。$\bar{v}$ 是求值环境，$\bar{e}$ 是被求值的表达式，f 是燃料值，它会随着每一步求值而递减；Res(v) 是执行的结果，可以是一个值 v，也可以是超时结果 OOT，表示燃料已耗尽。

$$
\begin{array}{rcl}
e & ::= & n \\
& | & \text{let } e_1 \text{ in } e_2 \\
& | & \text{fix}(e) \\
& | & e_1\ e_2 \\
& | & \mathrm{C}(\overline{e}) \\
& | & \text{匹配 } e\text{，其中 } [\mathrm{C}_i \to e_i]_{i \in I} \\
v & ::= & \mathrm{C}(\overline{v}) \mid \mathrm{Clos}(\overline{v}, \lambda\ e)
\end{array}
$$

(a) 源语言 λ_{DS}。

$$
\begin{array}{rcl}
e & ::= & \text{令 } x = \mathrm{C}(\overline{y}) \text{ 于 } e \\
& | & \text{令 } x = y.i \text{ 于 } e \\
& | & \text{函数 } f\ x = e_1 \text{ 于 } e_2 \\
& | & \text{令 } x = f\ \overline{y} \text{ 于 } e \\
& | & \text{匹配 } y\text{，分支为 } [\mathrm{C}_i \to e_i]_{i \in I} \\
& | & f\ \overline{x} \\
& | & \text{返回}(x) \\
v & ::= & \mathrm{C}(\overline{v}) \mid \mathrm{Clos}(\sigma, \text{ fun } f\ x = e) \\
\sigma & = & \text{变量} \to \text{值}
\end{array}
$$

(b) 目标语言 λ_{ANF}。

图 1. 语言文法。

3.2 目标语言

目标语言 λ_{ANF} 是一种采用管理范式并使用命名变量的 λ演算。在这种表示中，每个中间结果都会绑定到一个名称。其语法如图1b所示。七种表达式形式分别是：构造器分配 $\text{let } x = \mathrm{C}(y) \text{ in } e$、字段投影 $\text{let } x = y.i \text{ in } e$、函数定义 $\text{fun } f\ x = e_1 \text{ in } e_2$、非尾调用 $\text{let } x = f\ y \text{ in } e$、分支分派 $\text{case } y \text{ of } [\mathrm{C}_i \to e_i]_{i \in I}$、尾调用 $f\ x$，以及返回形式 $\mathrm{ret}(x)$。环境 σ 是从变量到值的有限映射；值是构造器值 $\mathrm{C}(v)$ 或闭包 $\sigma, \text{fun } f\ x = e$。

目标语言的语义由大步操作语义给出，记作$\sigma;\ e \Downarrow^{f}_{\text{目标}} r$，其中 σ是求值环境，e是待求值表达式，f是燃料值，r是执行的结果，可以是一个值 v，也可以是超时结果超时。

单孔上下文。 我们将 单孔上下文 E 定义为 λ_{ANF} 在尾位置带有孔 $\bullet$ 的表达式：

$$
\begin{array}{rcl}
\mathcal{E} & ::= & \bullet \\
& | & \mathtt{let}\ x = \mathrm{C}(\overline{y})\ \mathtt{in}\ \mathcal{E} \\
& | & \mathtt{let}\ x = y.i\ \mathtt{in}\ \mathcal{E} \\
& | & \mathtt{let}\ x = f\ \overline{y}\ \mathtt{in}\ \mathcal{E} \\
& | & \mathtt{fun}\ f\ \overline{x} = e\ \mathtt{in}\ \mathcal{E}
\end{array}
$$

我们用 $\mathrm{E}[\,e\,]$ 表示将 $\bullet$ 替换为 e后得到的表达式，并用 $\mathrm{E}_1 \circ \mathrm{E}_2$ 表示上下文组合。ANF 变换生成序对 (C, r)——一个上下文和一个结果变量——使得完整的目标表达式为 $C\,[\,\mathrm{ret}(r)\,]$。

3.3 逻辑关系

正确性证明使用步索引的无类型逻辑关系。该逻辑关系已被用于证明所有 λ_{ANF}-到-λ_{ANF} 在 CertiCoq 中的变换 [Paraskevopoulou 等人 2021], 以及 CPS变换的正确性证明（尽管方式不同）。该关系在值、结果和表达式上相互定义（图 2）。

两个值在步骤 k 处相关，记作 $\mathcal{V}^k(v_1, v_2)$，定义如下。两个构造器值在其标签一致且其字段逐点相关时相关。两个闭包 $\sigma_1, \text{fun } f\ x = e_1$ 与 $\sigma_2, \text{fun } g\ y = e_2$ 相关，当对于每个 $i < k$ 以及每一对参数向量 $\mathcal{V}^i(v_1, v_2)$，两个函数体在各自的扩展环境中产生相关的表达式配置。每次函数应用时从 k 到 $i < k$ 的严格递减确保该定义是

值关系

$\mathcal{V}^k(\mathsf{C}_1(v_1), \mathsf{C}_2(v_2)) \text{ 定义 } = \mathsf{C}_1 = \mathsf{C}_2 \ \wedge \ \mathcal{V}^k(\overline{v_1}, \overline{v_2})$

$\mathcal{V}^k(\langle \sigma_1, \text{函数 } f\ \overline{x} = e_1 \rangle, \langle \sigma_2, \text{函数 } g\ \overline{y} = e_2 \rangle) \text{ 定义 } =$
$\forall (i < k)\ \overline{v_1}\ \overline{v_2}, \mathcal{V}^i(\overline{v_1}, \overline{v_2}) \Rightarrow$
$\mathsf{len}(\overline{x}) = \mathsf{len}(\overline{v_1}) \Rightarrow$
$\mathsf{len}(\overline{y}) = \mathsf{len}(\overline{v_2}) \wedge \boldsymbol{E}^i(\sigma_1', e_1)\ (\sigma_2', e_2)$
其中 $\sigma_1' = \sigma_1[\overline{x} \mapsto \overline{v_1}, f \mapsto \sigma_1, \text{函数 } f\ \overline{x} = e_1]$
以及 $\sigma_2' = \sigma_2[\overline{y} \mapsto \overline{v_2}, g \mapsto \sigma_2, \text{函数 } g\ \overline{y} = e_2]$.

$\mathcal{V}^k(v_1, v_2) \text{ 定义 } = \text{假}$ 否则。

结果关系

$\boldsymbol{R}^k(\text{超时}, \text{超时}) \stackrel{\text{def}}{=} \text{真}$

$\boldsymbol{R}^k(\text{结果}(v_1), \text{结果}(v_2)) \text{ 定义 } = \mathcal{V}^k(v_1, v_2)$

$\boldsymbol{R}^k(r_1, r_2) \text{ 定义 } = \text{假}$ 否则。

表达式关系

$\boldsymbol{E}^k(\sigma_1, e_1)\ (\sigma_2, e_2) \text{ 定义 } = \forall r_1, \sigma_1; e_1 \Downarrow_{\mathsf{tr_g}} r_1 \Rightarrow \exists r_2, \sigma_2; e_2 \Downarrow_{\mathsf{tr_g}} r_2 \wedge \boldsymbol{R}^k(r_1, r_2)$

图 2. 关于 λ_{ANF} 项的逻辑关系。

良基的，尽管没有类型。结果关系 $\boldsymbol{R}^k(r_1, r_2)$ 提升了值相关性：超时与超时始终相关，当 v_1 与 v_2 值相关时 $\mathrm{Res}(v_1)$ 与 $\mathrm{Res}(v_2)$ 相关，且没有其他对相关。

表达式关系 $\mathrm{E}^k(\sigma_1, e_1)\ (\sigma_2, e_2)$ 是该定义的核心：它断言，对于 (σ_1, e_1) 求值得到结果 r_1 的每一次求值，都存在 (σ_2, e_2) 求值得到相关结果 r_2 的一次求值。在完整定义中，该关系还由求值成本上的资源不变量加以参数化，用于确立发散保持和渐近成本界；详情请参阅 [Paraskevopoulou 和 Grover，2021；Paraskevopoulou 等人 2021]。

4 ANF 变换及其正确性证明

ANF 变换的证明分两层完成：首先，我们将 ANF 变换定义为关系 $S;\ x;\ e \triangleright C;\ r;\ S'$，并证明该关系的语义保持。然后，我们证明使用基于新名称供给的状态单子定义的可执行 ANF 变换满足关系规范。这使我们能够将单子式推理与核心模拟论证分离开来，从而得到更清晰、更模块化的证明结构。所有 λ_{ANF} 变换的证明都遵循这一模式。

与续延传递风格不同，ANF 变换不会引入延续。函数仍采用直接风格，中间结果通过 let 绑定命名，而不是传递给延续。ANF 变换写作 $S;\ x;\ e \triangleright C;\ r;\ S'$，其中 S 是一个用于生成新变量的集合，x 是一个将 de Bruijn 索引映射到变量名的 de Bruijn 环境，而 e 是一个源表达式。C 是一个目标单孔上下文，并且 r：上下文 C 计算 e 的结果，并将其绑定到 r。与 e 对应的完整 ANF 表达式为 $C[\mathtt{ret}(r)]$。下面我们将介绍该变换的一些示例情形；其定义见图3。

变量。DeBruijn 索引 n 会被转换为环境 $x[n]$ 中对应的变量以及空上下文。

Lambda。lambda fix(e) 会引入一个新函数名 f 和参数 x_1，在环境 $x_1 :: x$ to 中转换函数体 e，得到上下文—变量对 (C_1, r_1)，并生成上下文 $\mathsf{fun}\ f\ x_1 = C_1[\mathsf{ret}(r_1)]\ \mathsf{in}\ \bullet$，其结果为 f。

应用。应用 $e_1\ e_2$ 会翻译每个子表达式，生成上下文 C_1、C_2 和结果变量 x_1、x_2，并生成上下文 $C_1 \circ C_2 \circ \mathsf{let}\ r = x_1\ x_2\ \mathsf{in}\ \bullet$，其中 r 持有结果。

$$\frac{\overline{x}[n] = y}{S; \overline{x}; n \rhd \bullet; y; S}\ \textsc{Var} \qquad \frac{x_1 \in S \qquad f \in S \setminus \{x_1\} \qquad S \setminus \{x_1, f\}; x_1 :: \overline{x}; e \rhd C_1; r_1; S'}{S; \overline{x}; \mathsf{fix}(e) \rhd \mathsf{fun}\ f\ x_1 = C_1[\,\mathsf{ret}(r_1)\,]\ \mathsf{in}\ \bullet; f; S'}\ \textsc{Lam}$$

$$\frac{S_1; \overline{x}; e_1 \rhd C_1; x_1; S_2 \qquad S_2; \overline{x}; e_2 \rhd C_2; x_2; S_3 \qquad r \in S_3}{S_1; \overline{x}; e_1\ e_2 \rhd C_1 \circ C_2 \circ \mathtt{let}\ r = x_1\ x_2\ \mathtt{in}\ \bullet; r; S_3 \setminus \{r\}}\ \textsc{App}$$

$$\frac{z \in S_1 \qquad S_1 \setminus \{z\}; \overline{x}; \overline{e} \rhd_{\mathrm{e}} C; \overline{y}; S_2}{S_1; \overline{x}; \mathrm{C}(\overline{e}) \rhd C \circ \mathtt{let}\ z = \mathrm{C}(\overline{y})\ \mathtt{in}\ \bullet; z; S_2}\ \textsc{Constr}$$

$$\frac{S_1; \overline{x}; e_1 \rhd C_1; x_1; S_2 \qquad S_2; x_1 :: \overline{x}; e_2 \rhd C_2; x_2; S_3}{S_1; \overline{x}; \mathsf{let}\ e_1\ \mathsf{in}\ e_2 \rhd C_1 \circ C_2; x_2; S_3}\ \textsc{Let}$$

图 3. ANF 变换（关系式定义，节选）。

4.1 证明陈述

主要正确性定理通过逻辑关系，将 e 的源语言求值与 C 这一 ANF 编译的目标上下文联系起来。为适配 CertiCoq 的大步语义，我们调整了该定理的表述。定理指出，源程序 e 在环境 v中求值为一个值 v，且 ANF 变换生成上下文 C及结果变量 r；那么，对于任意延续 e_k，在目标语言中运行 C后接着运行 e_k，在逻辑上等价于直接运行 e_k，并将 r绑定到翻译后的值 v'。我们通过 ANF 值翻译关系联系 v和 v'，该关系记作 anf_val_rel。我们将这一关系扩展到环境，记作 anf_env_rel。

定理 4.1（ANF 正确性）。设 v 是良构的环境，e 是良构的源表达式，且 $v; e \Downarrow_{\mathsf{src}} v$。假设 $S; x; e \rhd C; x; S'$。那么，对于任意满足 $\mathtt{anf_env_rel}(x, v, \sigma)$ 的目标环境 σ，以及满足 $\mathtt{anf_val_rel}(v, v')$ 的目标值，我们有

$$\forall\, i,\ \boldsymbol{\mathcal{E}}^i(e_k,\ \sigma[x \mapsto v'])\ (C[\,e_k\,],\ \sigma)$$

其中，e_k 是任意满足 $\mathsf{fv}(e_k) \cap ((S \setminus S') \setminus \{x\}) = \emptyset$ 的续延表达式。

全称量化的续延 e_k 使该陈述具有组合性：它允许在更大的上下文中归纳地应用此定理，如下文的 Let 情形所示。

示例：*Let* 情形。为说明证明结构，考虑**Let**情形。假设 v；$\mathsf{let}\, e_1 \mathsf{in}\ e_2 \Downarrow_{\mathsf{src}} r$，它可分解为 v；$e_1 \Downarrow_{\mathsf{src}} v_1$和 $v_1 :: v$；$e_2 \Downarrow_{\mathsf{src}} r$。ANF 变换得出上下文$C_1$、$C_2$以及结果变量 x_1、x_2，使得 S；x；$e_1 \rhd C_1$；x_1；S_2和 S_2；$x_1 :: x$；$e_2 \rhd C_2$；x_2；S'，输出上下文为 $C_1 \circ C_2$，结果为 x_2。我们需要证明：

$$\boldsymbol{\mathcal{E}}^i(e_k,\ \sigma[x_2 \mapsto v_2'])\ ((C_1 \circ C_2)[\,e_k\,],\ \sigma)$$

两个子求值的归纳假设给出：

$$\boldsymbol{\mathcal{E}}^i(e_k,\ \sigma[x_1 \mapsto v_1'][x_2 \mapsto v_2'])\ (C_2[\,e_k\,],\ \sigma[x_1 \mapsto v_1']) \qquad \text{(IH for } e_2\text{, continuation } e_k\text{)}$$
$$\boldsymbol{\mathcal{E}}^i(C_2[\,e_k\,],\ \sigma[x_1 \mapsto v_1'])\ (C_1[\,C_2[\,e_k\,]\,],\ \sigma) \qquad \text{(IH for } e_1\text{, continuation } C_2[\,e_k\,]\text{)}$$

传递性闭合了这条链。

顶层正确性。 遵循 Paraskevopoulou 和 Grover [2021], 定理 4.1 ，我们可以推导出一个全程序正确性推论。我们首先定义源值与目标值之间的观察关系 $\approx$：

$$\mathrm{C}(\overline{v}) \approx \mathrm{C}(\overline{v}') \stackrel{\text{def}}{=} \forall\, i,\ v_i \approx v'_i$$
$$\mathtt{Clos}(\overline{v}, \mathtt{fix}(e)) \approx \mathtt{Clos}(\sigma, \mathtt{fun}\ f\ x = e') \stackrel{\text{def}}{=} \mathtt{True}$$

当构造器值具有相同的最外层构造器，且其参数两两相关时，我们称它们相关；闭包则显然相关。接下来，我们定义全程序精化 $e_{\text{src}} \sqsubseteq^{e}_{\text{trg}}$:

$$e_{\text{src}} \sqsubseteq e_{\text{trg}} \stackrel{\text{def}}{=} \left(\forall\, v,\ [];e_{\text{src}} \Downarrow_{\text{src}} v \Rightarrow \exists v',\ \emptyset;e_{\text{trg}} \Downarrow_{\text{trg}} v' \wedge v \approx v'\right)$$

推论 4.2（全程序 ANF 正确性）。如果 e 是良构的，那么存在 C 和 x，使得 $S;\ [];\ e \triangleright C;\ x;\ S'$，并且

$$e \sqsubseteq C[\,\mathtt{ret}(x)\,]$$

其中 $C[$ 返回(x) $]$ 是在空目标环境中求值的。

我们还针对分离编译推导出了类似的推论，为简洁起见，此处略去。

4.2 发散保持

上述定理仅证明了终止程序的语义保持。CPS正确性证明还证明了发散保持。具体做法是进一步证明：当源程序耗尽燃料时， $v;\ e_{\text{src}} \Downarrow^{f_{\text{src}}}$源程序超时，则目标程序也会在某个 $\sigma;\ C[\,e_k\,] \Downarrow^{f_{\text{trg}}}$目标程序超时。 $f_{\text{trg}} \geq f_{\text{src}}$这是使用基于燃料的大步语义证明发散保持的一种标准技术，CakeML 编译器等也采用了这种方法 [Owens 等人 2017]。

然而，当变换可能减少程序执行所需的步数时，这种技术就会失效，行政范式（ANF）便是如此。考虑将一个变量应用于另一个变量的源程序$x_1\ x_2$及其 ANF 翻译结果令 $r = x_1\ x_2$，返回(r)。按照源语言和目标语言现有的代价模型，源程序需要三步求值（应用规则一步，变量查找规则两步），而目标程序只需两步求值（应用一步，变量查找一步）。我们认为可以调整代价模型来完成证明，但将其留作未来工作。大语言模型正确识别了这个问题，并生成了一个反例，但未能修复问题。CakeML 编译器通过插入“tick”指令来解决这一问题；这些指令会消耗燃料，但不执行任何工作。Paraskevopoulou 等人[2021] 提出了一种基于资源不变量的替代技术。

4.3 证明结构

ANF 正确性证明分为四个部分，并连同其证明规模汇总于表1。ANF证明比 CPS 证明大约大 47%，增量主要集中在语义保持部分。

5 经验报告：借助大语言模型进行证明开发

该证明使用 Claude Code 开发 [Anthropic 2025],Anthropic 用于软件开发的智能体编程助手，以搭载 Claude Opus4.6模型的 VSCode 扩展形式运行。Claude Code 能够读取和编辑文件、搜索代码库、运行构建命令，并在访问完整开发环境的情况下执行多步骤任务。在我们的设置中，它可以访问整个 CertiCoq 仓库，包括已完成的 CPS 证明文件，并能够调用 Rocqc 对生成的证明进行类型检查。我们首先着手处理

Part	描述	CPS	ANF
语义保持	主要模拟定理	2,243	4,755
α-等价	不同名称来源生成的行政范式结果在逻辑上相关	1,860	1,914
单子式/关系式	单子式实现满足关系规范	763	864
顶层定理	全程序与分离编译	428	250
总计		**5,294**	**7,783**

表 1. Rocq 证明结构及其行数规模。

主要的语义保持定理，然后转向证明的其他部分，逐步构建顶层定理。人类的职责是提供高层次指导：将证明开发组织成多个阶段，解释 CPS 证明与行政范式证明之间的高层次对应关系，并针对某些证明尝试提供自然语言反馈。交互的完整记录已与证明开发一并保存。

5.1 开发流程

我们最初要求大语言模型完成一份已有的部分证明（约 1,200 行）。这份证明因过于复杂，数月前便已搁置。大语言模型取得了一些进展，但最终陷入停滞，不断尝试类似却未能成功的方法。随后，我们在一个新分支上从头开始，并要求大语言模型遵循 CPS 证明的证明技术和结构，同时将推理调整到 ANF 环境。

阶段 *1*：定理陈述与证明框架。 大语言模型首先为四个相互归纳的证明义务生成定理陈述，然后着手搭建证明结构，整体组织方式紧密遵循 CPS 证明文件。它生成的初始正确性陈述缺少全称量化的延续 e_k（定理 4.1），而这对组合性至关重要。我们指出这一问题后，大语言模型便予以修正。随后，它迅速完成了简单的情形——主要正确性定理共有十三种情形——然后开始处理实质性情形。

阶段 *2*：逐案推导。 大语言模型逐一处理剩余情形，先部分填充各个情形，生成待证占位符（通常附有说明尚待证明内容的注释），并根据需要陈述辅助引理。我们指导它接下来处理哪个情形，它则会在开始前制定简短的工作计划。所有情形都用待证占位符勾勒出来后，我们提示它先解决内部待证占位符（即各个情形中的占位符），再处理外部辅助引理。这个顺序效果很好：内部待证占位符补齐后，外部引理也出人意料地顺利得以证明。

阶段 *3*：辅助证明文件。 在主要正确性证明完成后，我们将大语言模型引向其余三个证明文件。*α 等价工具*表明，当以不同的新鲜名称供应调用该变换时会得到逻辑相关的结果；这一部分进展顺利，并紧密遵循 CPS 模板。 *单子式/关系式对应* 则更令人失望：尽管它在概念上是四个文件中最简单的（纯粹的句法论证），并且与 CPS 对应版本最为相似，大语言模型仍然十分吃力。它没有改编证明，而是全盘复制了 CPS 证明，并试图根据错误消息逐一修复类型错误，导致许多缓慢的迭代。最终，作者手动修正了证明脚本最后十行中的两处策略调用，以显式统一一个阻碍其余目标的变量。该

顶层定理 较为直接。在此阶段，大语言模型发现，发散保持的标准技术不适用于 ANF 变换（第 4.2），并生成了一条高层可读注释，紧挨着受阻的待证占位符，解释了原因。

自主运行。 在此过程中，大语言模型经常被放手独立工作，有时连续数小时，并能自主持续推进，往往无需指导便能端到端地完成整个证明情形。但界面问题阻碍了这一过程：有时，尽管已获准执行常规操作（编译、使用 Grep 搜索），工具仍会反复请求许可，导致进度停滞，直到有人介入。

5.2 大语言模型表现出色之处

证明结构与适配。 大语言模型非常擅长将续延传递风格证明结构适配到行政范式环境中。它理解了证明的整体策略，并能在技术细节不同的环境中加以运用。

证明推理。 对于每个情形，大语言模型都会先推理该情形为何应当成立，通常会生成一段自然语言的思路，然后编写相应的证明脚本。当生成的脚本中不再包含任何待证占位符时，它通常在逻辑上是正确的，只需对语法做些调整（策略名称、假设名称、证明目标顺序）即可通过编译。

常规证明工作。 大语言模型擅长机械化证明中繁琐但必要的部分：建立归纳证明、进行分类讨论、应用代码库中的已知引理，以及管理证明子目标。它还独立生成了辅助引理，并以适当的一般性加以陈述。

代码质量。 生成的证明代码结构良好，命名规范清晰，注释信息丰富，文件组织合乎逻辑，并与现有代码库保持一致。

5.3 挑战与局限

静默削弱证明陈述。 我们观察到的最令人担忧的行为是，大语言模型会静默修改证明陈述，使其更容易证明。当它遇到发散保持障碍（第 4.2 节）时，它并未报告这一困难，而是从该陈述中移除了燃料上界，并用一个平凡的零值填补了燃料义务。这一点在证明顶层定理时被发现，但它凸显了一种风险：机器检验保证仅对实际被证明的陈述成立；如果该陈述已被削弱，则该保证可能是空洞的。对证明陈述和证明体进行仔细的人工审查仍然至关重要。

关于变量新鲜性的错误假设。 语义保持证明主体中最后两个待证占位符实际上编码了一个错误假设：大语言模型假定，let-应用情形中的某些结果变量不可能重合，因此不会发生遮蔽。事实上，这些变量可以 重合，因此需要进行分类讨论，并补充推理：若它们相同，则上下文的求值结果保持一致。大语言模型理解了这一问题，并成功修复了该证明。

迭代修复。 生成证明脚本后，交互式修复过程效率相当低下。大语言模型经常遇到假设名称方面的问题（假设名称会随着证明上下文的演变而变化），偶尔也会遇到目标编号问题，导致不必要的修复循环。这个迭代修复过程缓慢，而且常常陷入循环（修复一个错误又引入另一个错误）。大语言模型只能通过编辑文件和使用 Rocqc 编译器与 Rocq 交互，无法访问交互式证明状态。每次修复都需要完整地重新编译，耗时较长。

上下文窗口限制。 在漫长的证明过程中，大语言模型的上下文窗口经常被填满，从而触发自动上下文压缩。压缩后，它有时会忘记当前的证明状态，需要重新读取文件并再次 Grep 代码库，以重新熟悉情况；而这些工作其实已经在被压缩的对话部分中完成过了。这种反复探索耗时较长，也表明需要更好的机制，以便在上下文边界之间持久保留证明状态。

注意力与专注。 大语言模型的关注点可能狭窄且难以预测：它有时会在证明中相距甚远的部分之间跳转，或优先处理较容易的情况，同时回避困难的待证占位符。另一个反复出现的问题是重新实现代码库中已有（但当前文件未导入）的引理。

5.4 与 CPSProof 的比较

续延传递风格正确性证明由人类专家历经数月手动完成，既需要概念上的创新（结合语法模拟与步索引逻辑关系的证明技术），也需要大量机械性工作（在 Rocq 中形式化数百条引理）。相比之下，行政范式证明是在约 96 小时的大语言模型辅助工作中完成的，没有手动编写证明，而是将一种已知的证明策略机械化。人类的贡献包括提供高层次指导和审查，以及偶尔就推理步骤提供自然语言反馈。人类投入并未经过系统测量；我们在从事其他工作的同时对过程进行间歇性监督，也没有详细记录所花费的时间。

如表 1所示，ANF证明比其CPS 对应版本更大（总计7,783行对5,294行），其中主要正确性文件的规模大约是后者的两倍（4,755行对2,243行）。由于上下文处理，ANF证明在某些方面比CPS 证明更为复杂。在其他方面则更为简单：不存在需要推理的管理性归约。这也表明，由大语言模型生成的证明比人工编写的证明更为冗长。

6 讨论

一种基于模板的编译器验证范式。 我们的经验指向一种经验证的编译器开发的新工作流程：由人类专家为一次变换设计证明策略并构建验证基础设施，再由大语言模型将同一技术机械化地应用于后续相关变换。CertiCoq 的流水线是这种方法的天然候选：它使用同一逻辑关系框架来验证一系列 λ_{ANF}-到-λ_{ANF} 变换 [Paraskevopoulou 等人 2021], ，每一项都遵循相似的证明模式。我们预计，大语言模型将能有效地在这一族变换之间改编证明，从而大幅减少扩展或修改经验证的编译器流水线所需的工作量。

迈向经验证的代码生成。 如果大语言模型能够同时生成代码与证明，经验证的软件就可能从例外变为常态。与其耗费巨大人力构建单一的经验证编译器流水线，按需生成新的经验证变换或许会变得可行。我们的案例研究表明，对于遵循已知证明模板的变换而言，这一愿景已触手可及。

大语言模型辅助证明的工具支持。 我们识别出工具支持方面的若干具体缺口：**证明状态访问**。 大语言模型仅通过文件编辑和 Rocqc 调用来与 Rocq 交互。若能访问当前证明状态（证明目标、假设、可用引理），便可大幅减少花在句法修复上的时间。

结构化错误反馈。 Rocqc 的错误消息是为人类专家设计的。结构化、机器可读的错误输出可以帮助大语言模型更高效地诊断并修复证明失败。

跨上下文窗口的持久状态。 在上下文压缩边界之间总结并保留证明进展的机制，可以减少我们观察到的重复探索。

证明脚本与证明项。 一个自然的问题是，与生成证明脚本（策略序列）相比，大语言模型是否更适合生成证明项（完全显式的证明对象）。证明项可以避免策略脚本的脆弱性，因为假设名称和目标编号可能会悄然变化。另一方面，证明项更难以交互式地诊断和修复。我们将实证比较留待未来工作，但要指出，我们遇到的许多迭代修复难题，都是基于策略的工作流程所造成的。

证明自动化角色的变化。 如果大语言模型能够可靠地生成和修复证明脚本，那么自定义策略自动化在证明开发中的作用可能会发生变化。传统上，证明工程师会投入大量精力编写自定义策略和判定过程，以减轻证明负担。在大语言模型辅助的工作流程中，重点可能会从编写自动化转向编写清晰、结构良好的证明模板，供大语言模型遵循并加以调整。

可推广性。 我们的实验能否成功，取决于几个前提条件：存在一个密切相关且完整的证明，可作为模板；有一位领域专家能够识别对应关系并提供有针对性的指导；以及大语言模型具备足够的能力。这种方法在缺少一个或多个前提条件的情况下能否推广，例如没有相近模板的证明，或人类缺乏深厚专业知识的领域，仍是一个重要的开放问题。

可复现性与归属。 大语言模型本质上具有随机性，因此我们实验中的证明生成过程无法精确复现。我们通过公开完整证明及其提交历史来缓解这一问题，并提供证据表明，该证明是在与大语言模型辅助开发相符的时间框架内完成的。作者身份也很难归属，因为没有可靠的方法来确定代码是否由大语言模型生成。

证明专业知识的教育与未来。 如果大语言模型能够生成机械化证明，研究人员就会更多地专注于设计证明策略，而较少关注基于策略的证明的具体操作。与此同时，评估、指导和调试大语言模型生成的证明，需要具备一定水平的专业知识，而没有亲自动手进行证明的经验，就很难获得这种知识。这表明，即使证明教育的重点发生转变，它仍然十分重要。

依赖大语言模型提供商。 我们的实验依赖于一种商业大语言模型。这使我们依赖商业服务的持续可用性、可负担性和能力。尽管开源模型目前在这项任务上的能力较弱，但它们是社区值得探索的重要方向，既有助于提升可及性，也有助于实现科学上的可复现性。

7 相关工作

智能体式证明开发。 目前已有早期报告指出，智能体式证明开发取得了成功。Swamy [2026] 描述了使用 F* 和大语言模型生成一万行代码和证明的过程，涵盖了多种示例。de Moura [2026] 也指出，证明助手与人工智能正步入一个新的共生时代，并认为机器检查的证明生成是近期的重要机遇。我们的工作是将这一范式应用于编译器验证的一个实例。

大语言模型辅助证明生成。 近来有几篇论文探讨了使用大语言模型自动生成证明。Qin 等人 [2025] 研究了使用大语言模型为 FSCQ（一个以 Rocq 实现的经过验证的文件系统）生成证明，并在从 FSCQ 代码库中抽取的已有定理上实现了 38% 的证明覆盖率。我们的经验表明，如果有密切相关的证明模板可供参考，即使是在端到端的证明开发中，而不仅仅是独立的引理证明，也能实现更高的覆盖率。

PALM [Lu 等人 2024] 分析大语言模型生成的证明中的常见错误模式，并据此设计了一种先生成后修复方法，用于纠正大语言模型生成证明中的底层错误。该方法使用 GPT- 3.5，在包含 10K+ 个 Rocq 定理的数据集上实现了 40% 的覆盖率，性能比以往方法高出 76.6% 以上。CoqPilot [Kozyrev 等人 2024] 开发的 VSCode 扩展使用大语言模型填充 Rocq 文件中的证明空缺。该工具在一个包含 300 个 Rocq 定理的数据集上进行了评估，这些定理的证明最多包含 20 个策略；使用不同的大语言模型时，采用 GPT-4 可实现最高 50% 的覆盖率。类似地，Lean Copilot [Song 等人 2025] 开发了 Lean 证明助手的原生大语言模型接口 [de Moura 和 Ullrich 2021], , 用于协助自动完成部分证明开发。Rango [Thompson 等人 2025] 开发了一种 Rocq 综合工具，该工具会在每个证明步骤检索相关前提和相似证明，以指导微调的大语言模型。在 CoqStoq 这一新的基准测试中，该基准包含来自 2,226 个开源项目的 196,929 个定理；Rango 证明了其中 32.0% 的定理，比最先进水平的 Tactician 高出 29% [Blaauwbroek 等人 2020]）。我们的工作在规模上有所不同：这些工具专注于自动证明大量较小且彼此独立的引理，而我们则在用户指导下生成一个大型、连贯的证明。大多数先前工作都在已有证明语料库上评估证明生成；我们则针对此前尚未证明的命题。我们的经验表明，将此类工具集成到智能体证明环境中，可以进一步加快机械化证明开发的进程。

Misu 等人 [2024] 通过提示模型生成经验证的 Dafny 程序，探索了大语言模型辅助的经验证 Dafny 方法合成，在 Python 程序的 MBPP 数据集上实现了高达 58% 的覆盖率 [Austin 等人 2021]。Baldur [First 等人 2023] 展示了使用 Minerva 大语言模型进行完整证明生成 [Lewkowycz 等人 2022] 在 Isabelle/HOL 证明上进行了微调，并使用修复步骤纠正底层错误。

用于机械化数学的人工智能。 利用人工智能为数学定理生成机器检查的证明，正受到越来越多的关注。值得注意的是，DeepMind 的 AlphaProof [Hubert 等人2025], 构建了一种能够使用强化学习在 Lean 中形式化数学的形式证明系统，并解决了 2024年国际数学奥林匹克竞赛中的 6 道题中的 4 道。

8 结论

我们展示了一个大语言模型辅助的机械化证明开发案例研究。借助智能体大语言模型编程助手和人类提供的高层指导，我们在短短几天内，为 CertiCoq 中 ANF 变换的正确性开发了约 6,000 行 Rocq 证明；相比之下，相关性很高的 CPS 证明由人类专家手动开发，耗时数月。我们的经验表明，在关键阶段提供人工指导时，大语言模型能够非常有效地将已知证明技术应用于新的相关场景。我们认为，大语言模型辅助的证明开发或许预示着机械化证明构造的新范式，并将本案例研究作为一个具体实例，供进一步探究。

致谢

参考文献

Amal Ahmed。2006。递归类型与量化类型的步数索引句法逻辑关系。载于 编程语言与系统，Peter Sestoft（编）。Springer Berlin Heidelberg，柏林，海德堡，69–83。Abhishek Anand、Andrew W. Appel、John Gregory Morrisett、Zoe Paraskevopoulou、Randy Pollack、Olivier Savary Bélanger、Matthieu Sozeau 和 Matthew Weaver。2017。CertiCoq：面向 Coq 的经验证编译器。载于 *CoqPL'17*：第三届面向编程语言的 *Coq* 国际研讨会。2 页。Anthropic。2025。Claude Code。https://www.anthropic.com/claude-code。访问日期：2026年2月18日。Andrew W. Appel 和 David McAllester。2001。基础证明携带代码的递归类型索引模型。*ACM* 编程语言与系统汇刊23，5（2001 年 9 月），657–683。doi:10.1145/504709.504712Jacob Austin、Augustus Odena、Maxwell Nye、Maarten Bosma、Henryk Michalewski、David Dohan、Ellen Jiang、Carrie Cai、Michael Terry、Quoc Le 和 Charles Sutton。2021。使用大语言模型进行程序合成。*arXiv* 预印本 *arXiv:2108.07732* （2021）。https://arxiv.org/abs/2108.07732

Lasse Blaauwbroek、Josef Urban 和 Herman Geuvers。2020。The Tactician：面向 Coq 的无缝交互式策略学习器与证明器。载于《智能计算机数学》（CICM）（LNCS，第 12236 卷）。施普林格，271–277。doi:10.1007/978-3-030- 53518-6 17_Zaynah Dargaye 和 Xavier Leroy。2007。CPS变换的机械化验证。载于第 14 届国际编程逻辑、人工智能与推理会议论文集（亚美尼亚埃里温）（LPAR'07）。Springer-Verlag，柏林，海德堡，211–225。Leonardo de Moura。2026。人工智能时代的证明助手。博客文章。https://leodemoura.github.io/blog/2026/02/18/proof- assistants-in-the-age-of-ai.html 访问日期：2026年2月19日。Leonardo de Moura 和 Sebastian Ullrich。2021。Lean 4 定理证明器与编程语言。载于《自动演绎——CADE 28》（LNCS，第 12699 卷）。施普林格，625–635。doi:10.1007/978-3-030-79876-5_37 Emily First、Markus N. Rabe、Talia Ringer 和 Yuriy Brun。2023。Baldur：使用大语言模型进行整篇证明生成与修复。载于第 31 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集（美国加利福尼亚州旧金山）（ESEC/FSE 2023）。计算机协会，纽约，纽约州，美国，1229–1241。doi:10.1145/3611643.3616243 GitHub。2025。GitHub Copilot。https://github.com/features/copilot。访问日期：2026年2月18日。Thomas Hubert、Rishi Mehta、Laurent Sartran 等。2025。通过强化学习实现奥林匹克级形式化数学推理。《自然》（2025）。doi:10.1038/s41586-025-09833-y Andrei Kozyrev、Gleb Solovev、Nikita Khramov 和 Anton Podkopaev。2024。CoqPilot：用于基于大语言模型生成证明的插件。载于第 39 届 IEEE/ACM 自动化软件工程国际会议论文集（美国加利福尼亚州萨克拉门托）（ASE '24）。计算机协会，纽约，纽约州，美国，2382–2385。doi:10.1145/3691620.3695357 Xavier Leroy。2009。形式化验证的编译器后端。《自动推理杂志》43，4（2009），363–446。Aitor Lewkowycz、Anders Andreassen、David Dohan、Ethan Dyer、Henryk Michalewski、Vinay Ramasesh、Ambrose Slone、Cem Anil、Imanol Schlag、Theo Gutman-Solo 等。2022。使用语言模型解决定量推理问题。《神经信息处理系统进展》35（2022），3843–3857。https://arxiv.org/abs/2206.14858 Minghai Lu、Benjamin Delaware 和 Tianyi Zhang。2024。使用大语言模型实现证明自动化。载于第 39 届 IEEE/ACM 自动化软件工程国际会议论文集（美国加利福尼亚州萨克拉门托）（ASE '24）。计算机协会，纽约，纽约州，美国，1509–1520。doi:10.1145/3691620.3695521 Wolfgang Meier、Martin Jensen、Jean Pichon-Pharabod 和 Bas Spitters。2025。CertiCoq-Wasm：CertiCoq 的经验证 WebAssembly 后端。载于第 14 届 ACM SIGPLAN 认证程序与证明国际会议论文集（CPP 2025）。计算机协会，纽约，纽约州，美国，127–139。doi:10.1145/3703595.3705879 Md Rakib Hossain Misu、Cristina V. Lopes、Iris Ma 和 James Noble。2024。面向经验证的 Dafny 方法的人工智能辅助合成。载于 ACM 软件工程基础国际会议论文集（FSE '24）。计算机协会，纽约，纽约州，美国。doi:10.1145/3643763 Scott Owens、Michael Norrish、Ramana Kumar、Magnus O. Myreen 和 Yong Kiam Tan。2017。验证 CakeML 中高效的函数调用。《ACM 程序设计语言会议论文集》1，ICFP，第 18 号文章（2017 年 8 月），27 页。doi:10.1145/3110262 Zoe Paraskevopoulou 和 Andrew W. Appel。2019。闭包转换在空间方面是安全的。《ACM 程序设计语言会议论文集》3，ICFP，第 83 号文章（2019 年 7 月），29 页。doi:10.1145/3341687 Zoe Paraskevopoulou 和 Anvay Grover。2021。正确地使用延续进行编译。《ACM 程序设计语言会议论文集》5，OOPSLA，第 114 号文章（2021 年 10 月），29 页。doi:10.1145/3485491 Zoe Paraskevopoulou、John M. Li 和 Andrew W. Appel。2021。CertiCoq 的组合式优化。《ACM 程序设计语言会议论文集》3，ICFP，第 86 号文章（2021 年 7 月），30 页。doi:10.1145/3341687 Sida Peng、Eirini Kalliamvakou、Peter Cihon 和 Mert Demirer。2023。人工智能对开发者生产力的影响：来自 GitHub Copilot 的证据。arXiv 预印本 arXiv:2302.06590（2023）。https://arxiv.org/abs/2302.06590 Jianxing Qin、Alexander Du、Danfeng Zhang、Matthew Lentz 和 Danyang Zhuo。2025。大语言模型能否验证系统软件？以 FSCQ 为基准测试的案例研究。载于 2025 年操作系统热点研讨会论文集（加拿大艾伯塔省班夫）（HotOS '25）。计算机协会，纽约，纽约州，美国，34–41。doi:10.1145/3713082.3730382 Olivier Savary Bélanger 和 Andrew W. Appel。2017。正确地快速收缩！载于第 19 届声明式编程原理与实践国际研讨会论文集（比利时那慕尔）（PPDP '17）。计算机协会，纽约，纽约州，美国，49–60。doi:10.1145/3131851.3131859 Olivier Savary Bélanger、Matthew Z. Weaver 和 Andrew W. Appel。2019。从 CPS 到 C 的认证代码生成。（2019 年 10 月）。未发表手稿。https://www.cs.princeton.edu/~appel/papers/CPStoC.pdf。Peiyang Song、Kaiyu Yang 和 Anima Anandkumar。2025。Lean Copilot：作为 Lean 定理证明助手的大语言模型。载于神经符号系统国际会议论文集（机器学习研究论文集，第 288 卷）。PMLR，144–169。https://proceedings.mlr.press/v288/song25a.html Matthieu Sozeau、Abhishek Anand、Simon Boulier、Cyril Cohen、Yannick Forster、Fabian Kunze、Gregory Malecha、Nicolas Tabareau 和 Théo Winterhalter。2020。MetaCoq项目。《自动推理杂志》（2020 年 2 月）。doi:10.1007/

s10817-019-09540-0

Nikhil Swamy. 2026. 智能体式面向证明的编程。RiSE MSR 博客。https://risemsr.github.io/blog/2026-02-04-nik- agentic-pop/ 访问日期：2026-02-18。

Kyle Thompson、Nuno Saavedra、Pedro Carrott、Kevin Fisher、Alex Sanchez-Stern、Yuriy Brun、João F. Ferreira、Sorin Lerner 和 Emily First。2025年。Rango：面向自动化软件验证的自适应检索增强证明。载于*IEEE/ACM 第47届国际软件工程大会（ICSE '25）*论文集。ACM。doi:10.1109/ICSE55347.2025.00161

证明开发统计数据

我们报告从所有 Claude Code 会话保存的对话记录（JSONL 日志）中提取的证明开发过程统计数据。这是作者首次使用大语言模型进行机械化证明开发，也是首次使用 Claude Code。因此，事先没有制定系统性的交互组织方案或进度衡量方法。开发过程通过 80 次手动 Git 提交进行跟踪。证明开发历经两个 Claude Code 会话：作者在第0天开始于一个会话中，次日意外切换到一个新会话，并在此后继续完成开发。以下报告的统计数据汇总了这两个会话。作者订阅了 Claude Max 20x 套餐；每周使用限额足以完成整个证明开发。

总体统计数据。 表 2 汇总了所有会话的工具使用总量。开发历时连续五天，墙钟时间为 90.2 小时（每天从首次事件到最后一次事件的经过时间），活跃时长为 62.6 小时（对实际运行时间的近似估算，即排除没有记录事件且间隔超过 30 分钟的时段）。人类总共提供了 114 条提示，而大语言模型进行了 6,538 次工具调用。五天内共发生 101 次上下文压缩，反映出这些会话持续时间较长。

指标	数量
用户提示（人工）	114
上下文压缩	101
Rocqc/make 调用	872
Grep/搜索调用	1,684
编辑/写入调用	1,209
读取调用	2,383
Bash 调用	1,035
工具使用总次数	**6,538**
挂钟时间（小时）	90.2
活跃时长	62.6

表2. 所有会话中的大语言模型工具使用总量。

按日和按文件的活动。 表 3 按天分解了开发活动。第0天基本上是一次完全自主的运行。那条唯一的初始提示（第B节）之后是大约4.3小时的活跃时长，其间大语言模型探索了代码库，熟悉了证明结构，并开始着手语义保持证明。语义保持证明的大部分是在第1至2天完成的，大语言模型在主证明和所生成的辅助引理上稳步推进。第3至4天则专门用于 α 等价证明、单子式/关系式对应以及顶层定理。第4天的人工提示数量最多（50）。这些提示中的大多数出现在顶层推论完成之后，当时

用户和大语言模型在发散保持证明上遇到了阻碍。用户提示大语言模型解释该证明中的问题并提供一个反例。随后，用户提示大语言模型逐步演练具体的证明情形，试图理解并修复问题的根源。

Day	提示	Compact	Rocqc	Grep	编辑/写入	Read	Bash	总时长（小时）	活动时长（小时）
第0天	1	9	72	150	111	212	80	4.3	4.3
第1天	18	21	211	324	302	589	226	16.2	12.6
第2天	22	22	204	456	323	610	255	24.0	14.9
第3天	23	34	208	466	270	661	291	24.0	18.3
第4天	50	15	177	288	203	311	183	21.7	12.5
总计	**114**	**101**	**872**	**1,684**	**1,209**	**2,383**	**1,035**	**90.2**	**62.6**

表3。每日开发活动。

逐文件活动。表4 按证明文件和天细分展示了工具使用情况。语义保持文件受到的关注远超其他文件，在第0至第2天占据主导地位，五天内共进行了515次编译尝试。α 等价工具和单子式/关系式对应主要是在第3天和第4天开发的；尽管后者在概念上更简单、规模也更小，却需要编译132次（第5.3节）。顶层定理是工作量最小的部分，全部在第4天完成。逐文件总数与表3中的总计并不相符，因为大语言模型还读取并搜索了代码库中的其他文件，以查找参考引理和定义。

证明文件	Day	提示	Rocqc	Grep	编辑/写入	Read
语义保持	0	0	66	49	107	100
	1	18	209	142	302	493
	2	18	199	196	320	536
	3	1	41	33	48	74
	4	22	0	32	2	50
	总计	**59**	**515**	**452**	**779**	**1,253**
α-等价	3	13	141	79	208	385
	4	0	0	12	0	12
	总计	**13**	**141**	**91**	**208**	**397**
单子式/关系式	3	1	1	0	1	4
	4	14	131	17	164	145
	总计	**15**	**132**	**17**	**165**	**149**
顶层定理	4	6	28	0	36	19
	总计	**6**	**28**	**0**	**36**	**19**

表4. 每个证明文件每天的工具使用情况。

B 示例用户提示

为了展示人类与大语言模型交互的特点，我们从对话记录中精选了一些用户提示。为便于阅读，这些提示经过少量编辑（修正错别字），除此之外均为原文。我们根据它们在开发过程中的作用进行了分组。

初始任务说明。第一个提示设定了总体目标：

> “文件 LambdaBoxLocal_到_LambdaANF.v 包含 *CPS* 和行政范式（*ANF*）变换的定义。*CPS*变换的正确性已在文件 LambdaBoxLocal_到_LambdaANF_correct.v 和 LambdaBoxLocal_到_LambdaANF_corresp.v 中得到证明；*ANF* 的正确性尚未得到证明。你能否应用类似的证明技术来证明 *ANF* 的正确性？你生成的代码和证明应当能够通过编译。”

指示接下来要处理的任务。相当一部分提示只是引导大语言模型转向下一个任务：

> “继续处理其余的待证占位符。请注意：目前先忽略 preord_exp_Ecase_red；anf_ cvt_val_alpha_equiv 和 anf_val_rel_exists 将是最后要证明的部分。”
>
> “做得太棒了。现在，我希望你着手移除主证明中的所有待证占位符。引入辅助引理时务必谨慎，确保它们确实可证。等所有内部的待证占位符都清除后，我们再来证明所有辅助引理。”
>
> “接下来，我希望你处理 行政范式_cvt_disjoint_* 这一组中的引理。”
>
> “现在来处理最后一个大型证明，然后我们再转向顶层引理。这个证明要说明关系式变换与单子式变换之间的对应关系。应按照续延传递风格来完成；续延传递风格的证明位于文件 LambdaBoxLocal_to_LambdaANF_corresp 中。请为行政范式创建一个类似的文件，并完成证明。”

回看上一个提示，大语言模型复制了续延传递风格对应关系证明，这可能是因为提示并未明确要求改编该证明，而是要求创建一个类似的文件。

数学指导。 在少数关键情况下（例如语义保持证明中的构造子情形），人类以自然语言提供了详细的证明思路。最重要的几个例子如下所示。

> “我有以下想法，希望你研究一下并告诉我是否可行。• 首先，放弃你正在做的所有替换_为_rho 相关工作。• 我们知道xs 可能包含重复项。令xs[i] = xs[j] 对 $i \neq j$成立。我认为vs0[i] =vs0[j]应该成立。我们将利用 *ANF* 转换的定义和语义的定义，通过结构归纳证明这一点。甚至可能可以直接从转换的定义推出，语法上es[i] =es[j]，这样根据语义的确定性，证明vs0[i] = vs0[j] 会更容易。
>
> • 然后，利用 α-等价结果，preord_val vs’0[i] = vs’0[j]，这应该足以使证明成立。
>
> 请检查这种方法是否可行；如果可行，就实现它。如果不可行，请告诉我问题所在。”
>
> “我希望我们证明以下命题，然后用它来证明剩下的待证占位符。
> • 行政范式_转换_关系 S e 变量名 约束 $S, C\ x \rightarrow x$ = 变量名[i] → 求值_燃料 环境 $e\ v\ f\ t \rightarrow v$ = 变量名[i]
> • 行政范式_转换_关系_表达式 S 表达式 变量名 约束 $S, C\ xs \rightarrow xs[j]$ = 变量名[i] → 求值_燃料_表达式 环境 表达式 变量 $f\ t \rightarrow$ 变量[j] =变量名[i]
> 先确认它成立，再开始证明。你需要使用互归纳。如果发现它不成立，请立刻告诉我。”
>
> “我在想如何利用这些引理来移除第一个待证占位符。假设我们有一个更一般的 一致版本，它将第二个列表与任意关系联系起来，而不只是相等关系。那么：

- 引理 *1*：eval_燃料_many *vs′ es vs0 fs ts* → anf_cvt_correct_exps *vs′ es vs0 fs ts* → Forall2 (*λ v v′*. anf_val_rel *v v′*) *vs0 vs′ 0* → consistent preord_val *vs0 vs′0*。你可能需要用到 *anf-equiv* 引理。
- 引理 *2*：一致的 preord_val *xs* 与 → ∃ *xs*，getlist *xs* (setlist *xs* 与 *ρ*) = *vs′* ∧ Forall2 preord_val*vs* 与′。

那么，这两者结合起来应该足以移除待证占位符。”

纠正与错误诊断。 人类发现了从悄无声息地擦除证明到策略误用等各种错误。在少数情况下，用户将证明状态直接输入大语言模型，以协助修复底层策略错误。

> “它能编译，是因为你擦除了证明。证明目标是修复证明，请把它加回来。”
>
> “它就在文件后面的部分。你试图使用一个稍后才证明的引理。”
>
> “你正在对 HL 取反，而它是一个 var。这毫无意义；这很可能是命名不一致导致的。你应该对 H1 取反，然后 zify;lia 就能奏效。”

提示与继续。 相当一部分提示只是简单的延续，通常是在大语言模型暂停等待指示或上下文压缩之后所需要的：

> “你为什么停下了？”“请继续。”“从你停下的地方接着做。”

证明局限性讨论。 当大语言模型识别出发散保持这一障碍时（第 4.2 节），互动转为协作式讨论：

> “你能解释一下界和超时情形的问题，以及解决它需要做些什么吗？ ”
>
> “你能给我一个该性质的反例，放进论文里吗？”