软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007656] [CSTR: 32375.14.jos.007656]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

引理库驱动的线性数据结构算法定理自动证明*

王昌晶, 万亮亮, 刘艳娇, 龙海建, 左正康

(江西师范大学 人工智能学院, 江西 南昌 330022)

通信作者: 左正康, E-mail: zuo803@jxnu.edu.cn

摘　要: 尽管大语言模型 (large language model, LLM) 为算法定理自动证明领域提供了一条新的探索路径, 但现有方法未能充分提升自动定理证明器在证明中间命题时的能力. 当前在线性数据结构算法形式化定理证明方面所需的标签数据非常稀少, 将大语言模型用于预测时, 尤其具有挑战性. 为弥补这一不足, 构建一种新的面向线性数据结构算法的引理库驱动证明方法 LDPM, 融合了大语言模型与形式化技术. 该方法通过构建一个由已被证明的新引理组成的引理库, 通过从引理库中调取相关引理, 可有效强化自动定理证明器对中间命题的证明效能. 引理库的构建遵循三阶段递进式策略: 首先通过联合请求机制生成多个候选引理陈述, 以丰富库内资源; 接着借助迭代证明策略逐个证明候选引理, 逐步提升引理库的可靠性; 最终引入引理反思机制, 借助 LLM 对引理进行修正与优化, 从而实现引理库的动态扩展与持续完善. 通过这一方法, 线性数据结构算法定理证明的性能得以显著提升, 与最先进的方法比较, 证明成功率从 44.00% 提高到 82.00%. 消融实验进一步表明, 引理库在提升自动定理证明器证明中间命题的能力上发挥了关键作用, 使线性数据结构算法定理证明的成功率提高了 38.00%.

关键词: 大语言模型; 形式化技术; 定理证明; 引理库; Isabelle/HOL

中图法分类号: TP18

中文引用格式: 王昌晶, 万亮亮, 刘艳娇, 龙海建, 左正康. 引理库驱动的线性数据结构算法定理自动证明. 软件学报. http://www.jos.org.cn/1000-9825/7656.htm

英文引用格式: Wang CJ, Wan LL, Liu YJ, Long HJ, Zuo ZK. Lemma-library-driven Automated Theorem Proving for Linear Data Structure Algorithms. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7656.htm

Lemma-library-driven Automated Theorem Proving for Linear Data Structure Algorithms

WANG Chang-Jing, WAN Liang-Liang, LIU Yan-Jiao, LONG Hai-Jian, ZUO Zheng-Kang

(School of Artificial Intelligence, Jiangxi Normal University, Nanchang 330022, China)

Abstract: Although large language models (LLMs) provide a new exploration path in the field of automated algorithmic theorem proving, existing methods fail to sufficiently improve the capability of automated theorem provers in proving intermediate propositions. Currently, labeled data for formal theorem proving in linear data structure algorithms is very scarce, which presents significant challenges when large language models are used for prediction. To address this issue, this study proposes a novel lemma-library-driven proof method, LDPM, for automated theorem proving of linear data structure algorithms, which integrates large language models with formalization techniques. The method constructs a lemma library composed of newly proven lemmas. By retrieving relevant lemmas from this library, the capability of the automated theorem prover in proving intermediate propositions can be effectively enhanced. The construction of the lemma library follows a three-stage progressive strategy. First, a joint request mechanism is adopted to generate multiple candidate lemma statements, thus enriching the library resources. Second, an iterative proof strategy is employed to prove the candidate lemmas one by one, gradually improving the reliability of the lemma library. Finally, a lemma reflection mechanism is introduced, leveraging LLMs to revise and optimize lemmas, achieving the dynamic expansion and continuous refinement of the lemma library. Through this method, the performance of theorem proving for linear data structure algorithms is significantly improved. Compared with the state-of-the-art methods, the success

* 基金项目: 国家自然科学基金 (62462037, 62462036); 江西省主要学科学术与技术带头人培养项目 (20232BCJ22013); 江西省自然科学基金重点项目 (20242BAB26017)
收稿时间: 2024-12-06; 修改时间: 2025-05-06, 2025-11-05; 采用时间: 2026-02-26; jos 在线出版时间: 2026-07-08

rate of theorem proving is increased from 44.00% to 82.00%. Ablation experiments further show that the lemma library plays a key role in improving the capability of automated theorem provers in proving intermediate propositions, increasing the success rate of theorem proving for linear data structure algorithms by 38.00%.

Key words: large language model (LLM); formalization technique; theorem proving; lemma library; Isabelle/HOL

定理的形式化验证是确保定理正确性的关键途径之一. 在数据结构的定理证明中, 通常包括建模、性质描述和验证[1–3]. 其中, 建模是对数据结构操作函数 (如增、删、改、查等操作函数) 进行形式化定义; 性质描述则是对操作函数的功能属性进行形式化定义; 而验证则是对定理进行形式化证明的过程. 在定理验证过程中, 用户通常通过交互式定理证明器 (如 Coq[4]、Isabelle[5]、Lean[6]和 Metamath[7]) 引导证明步骤, 逐步推进, 直到定理验证成功. 但实际情况中, 定理的形式化验证过程往往极为复杂且耗时, 需要大量的人工干预和精力. 因此, 推动自动定理证明的实现已成为一项迫切的研究需求. 自动定理证明不仅能够有效降低人工参与所带来的成本, 还能大幅提升定理验证的效率, 使得更复杂的定理证明任务变得可行, 为形式化验证领域带来更广泛的应用前景.

机器学习在交互式定理证明领域的研究范式可归纳为以下 3 类: 第 1 类为基于模板的交互式证明生成方法[8,9], 该方法针对当前证明状态预测并执行相应的证明策略, 从而生成新的证明状态, 并循环此过程直至证明目标达成. 第 2 类方法基于序列到序列模型进行证明生成[10,11], 其核心思想围绕中间命题构建证明过程, 具备接近自然语言的表达形式. 在此范式下, 用户仅需提出中间命题或猜想, 无需关注具体的策略实现, 后续证明策略由自动定理证明器[12]自动合成. 第 3 类方法聚焦于定理筛选[13,14], 即在已构建的定理或引理库中, 选取对当前证明目标具有潜在帮助的定理或引理, 以辅助后续证明过程.

最近, 通过大语言模型 (large language model, LLM) 生成中间命题/猜想 (第 2 类)[15,16]和构造相关定理/引理库 (第 3 类)[17]显现出极大的应用潜力. 但是在形式化定理证明方面所需的标签数据非常稀少, 将大语言模型用于预测时, 尤其具有挑战性, 这是由于在数据稀缺、容易拟合过度的情况下, 性能与模型大小的比例关系很快就会被打破[18]. 尤其是在线性数据结构算法定理这一领域, 堆栈、队列和双端队列 (deques) 等经典线性数据结构在安全攸关关键应用中得到广泛应用. 在《信息技术安全评估通用标准》的最高评估保证级别 (EAL) 中, 此类数据结构必须经过形式化规范, 并验证符合其规范[19].

现有研究通过不同的策略增强了 LLM 生成形式化证明草图的能力[15,17,20], 从而获得易于进行形式化验证的证明草图, 随后利用交互式定理证明器 (如 Isabelle) 内置的自动定理证明器 Sledgehammer[12]来证明草图中的中间命题. 然而, 上述方法在处理线性数据结构定理的证明任务时, 对于自动定理证明器在中间命题层面的证明能力尚未形成有效提升. Sledgehammer 在处理中间命题时采用基于预定义定理/引理库的前提选择策略, 以筛选相关引理并合成所需的证明策略. 然而, 随着线性数据结构问题复杂性的增加, Sledgehammer 在处理中间命题时存在引理选择范围不足的问题. 而 DSP[15]方法在提升 LLM 生成形式化证明草图能力方面取得了显著进展, 但未能充分增强自动定理证明器在验证中间命题时的能力, 这在更复杂的证明任务中可能导致局限. 现有方法实验结果表明, 无论是单独使用 Sledgehammer (验证成功率为 36.00%), 还是采用现有方法 DSP[15] (验证成功率为 44.00%), 其证明能力均因中间引理的不足而受到限制. 相较本文提出的方法 (验证成功率为 82.00%), 上述两种方法分别有 46.00% 和 38.00% 的证明失败是因为缺少中间引理库.

为弥补这一不足, 本研究构建了一种新的引理库驱动的线性数据结构算法定理自动证明方法, 融合了大语言模型与形式化验证技术. 该方法通过构建一个由已被证明的新引理组成的引理库, 通过从引理库中调取相关引理, 可有效强化自动定理证明器对中间命题的证明效能. 本文紧密围绕线性数据结构的特性开展研究, 这主要体现在: 1) 核心引理库的构建. 本文方法引理库重点覆盖了线性结构 (如栈、有界栈、先进先出队列、优先级队列和子列表) 的典型操作 (如迭代、插入、删除) 所涉及的归纳原理和状态不变量. 2) 生成策略的针对性. 本文方法集成的中间引理生成策略在处理线性的数据访问模式时最为有效.

构建引理库面临两个主要挑战: 1) 获取多样性的引理陈述, 以丰富引理库的内容; 2) 形式化验证引理陈述, 以保证引理库可靠可信. 为应对上述挑战, 引理库驱动的证明方法是一个思维链过程, 包括以下步骤: 首先借助 LLM

产生定理的非形式化证明, 以此为基础构造其对应的形式化证明草图; 随后, 采用联合请求机制, 分别以非形式化证明与形式化证明草图为引导, 促使 LLM 生成多样化的候选引理陈述, 以扩充引理库; 为保障引理的逻辑严谨性, 本文引入迭代证明策略验证这些候选引理以提升引理库的可靠性; 在此基础上, 进一步引入引理反思机制, 针对 LLM 在引理生成过程中可能存在的偏差进行修正, 以优化引理陈述的准确性; 最终, 依托构建完成的引理库, 调用自动定理证明器验证证明草图中的各个中间命题, 从而保障定理证明的完整性与正确性.

综上所述, 本文的主要创新贡献可归纳为以下 3 个方面.

(1) 提出一种面向线性数据结构的引理库驱动式定理证明方法. 此方法将 LLM 与形式化验证技术相结合, 促使 LLM 生成与目标定理及其子目标均具有相关性的引理陈述, 并通过迭代式构建过程形成全新的自定义引理库. 该库的引入有效提升了现有自动定理证明器在线性数据结构算法定理中间命题证明中的推理能力.

(2) 针对线性数据结构引理库构建过程中面临的主要挑战, 设计三阶段递进式策略: 首先通过联合请求机制生成多个候选引理陈述, 以丰富库内资源; 接着借助迭代证明策略逐个证明候选引理, 逐步提升引理库的可靠性; 最终引入引理反思机制, 借助 LLM 对引理进行修正与优化, 从而实现引理库的动态扩展与持续完善.

(3) 在栈、有界栈、先进先出队列、优先级队列及子列表这多种典型线性数据结构上对本文方法进行了系统评估. 与最先进的工作比较, 结果显示本文方法将线性数据结构算法定理证明的成功率从 44.00% 提高至 82.00%, 消融实验进一步验证了引理库在增强现有自动定理证明器处理此类定理中间命题时发挥了关键作用.

1 相关工作

本文相关工作主要聚焦于基于形式化证明草图生成的定理证明、基于预定义引理筛选的定理证明或基于自定义引理库构建的定理证明.

● 基于形式化证明草图生成的定理证明. 在自动形式化定理证明这一前沿领域中, 机器学习技术已成为研究热点. 然而, 这类方法在生成形式化证明的过程中, 对形式化训练数据具有高度依赖性, 而此类数据在实际中往往较为匮乏; 相比之下, 非形式化证明语料则来源丰富、获取门槛较低. 受此现象启发, 早期工作如 Jiang 等人[15]提出的 DSP 方法, 采用“非形式化证明引导形式化草图生成, 再由定理证明器验证”的流水线. 随后, Zhao 等人[20] 提出一种以子目标为导向的演示学习框架, 有效提升了大语言模型用于形式化定理证明的效果. First 等人[16]更进一步, 将完整证明一次性生成, 并引入了证明修复模型作为辅助手段, 以提高定理证明的成功率. 尽管上述工作均聚焦于利用 LLM 构造形式化证明, 但 LLM 在生成长序列证明时表现欠佳. 为应对这一挑战, Wang 等人[17]引入了 LEGO-Prover, 作为一种新颖的定理证明方法, 它通过模块化地构造证明, 并使用一个不断增长的技能库来不断提高 LLM 的形式化能力. 尽管上述研究在提升 LLM 生成形式化证明草图能力方面取得了显著进展, 然而它们未能充分增强自动定理证明器在验证中间命题时的能力, 这在更复杂的证明任务中可能导致局限. 本文请求大语言模型自动生成辅助定理证明的引理, 并形式化验证这些生成的引理. 通过集成这些经过验证的引理, 旨在降低定理证明的复杂度, 从而增强自动定理证明器在验证中间命题时的能力.

● 基于预定义引理筛选的定理证明. Claessen 等人[21]首次提出 Hipspec 的核心思想, 通过理论探索 (theory exploration) 自动合成定理/引理. 如今, 自动推理工具 (hammer) 在形式化方法、程序验证中已成为不可或缺的助手, Czajka 等人[22]在 Coq 中设计的 Coqhammer 与 Paulson 等人[12]在 Isabelle 设计的 Sledghammer, 均是通过自动化合成来辅助定理证明. 基于语义合成方法是将定理/引理分解为子目标, 利用 theory exploration 来自动查找所需引理并验证, 其引理生成的方法是基于交互式定理证明器引导人工来构造及证明, 存在一定难度. 本文聚焦于该领域中的核心挑战: 如何为证明复杂定理生成所需的引理. 提出了基于大语言模型的方法和用于构建引理库的 3 个策略, 生成了经过验证的相关引理, 显著提升了证明复杂定理的正确性和效率.

Alemi 等人[23] 使用了循环神经网络 (LSTM/GRU) 与卷积神经网络 (CNN) 来提取特征用于定理/引理筛选, 从而完全避免手工设计特征. 随后为能够更好地提取命题语法结构信息, Paliwal 等人[24]将图神经网络 (graph neural network, GNN) 应用于这一任务. 最近, Mikuła 等人[14]介绍了 Magnushammer, 使用 Transformer 模型以及对比学

习 (contrastive learning) 作为解决定理/引理筛选问题的一种新方法. 上述基于机器学习的工作充分展示了定理筛选对证明成功率的巨大影响, 然而依赖于从一个系统预定义定理/引理库中进行筛选. 随着问题的复杂化, 这种依赖于预设引理库的方式可能难以应对更大规模、更为复杂的定理证明任务. 为增强自动定理证明器在合成证明策略时的效能, 本文构建了一个全新的自定义引理库, 该库的引理更加多样化、可信性高, 能够有效扩展其可用的引理资源.

● 基于自定义引理库构建的定理证明. 在处理规模更大、复杂性更高的定理证明任务时, 不仅需要在加载的系统预定义定理/引理库中启发式地筛选相关定理/引理, 还需构造自定义的引理库. Mündler 等人[25]展示了在交互定理证明 Isabelle/HOL 中数据结构 B+树的命令式实现的验证. Toth 等人[26]验证了实时双端队列, 其中所有操作都需要常数时间. Miao 等人[27]提出了一种兼容的经过验证的 TEE 架构, 其中验证了 386 个引理/定理. 虽然上述工作成功验证了程序的正确性, 但其实现依赖于人工构建引理进行形式化证明, 这种方式不仅高度依赖专家知识, 也伴随着显著的人力资源消耗. Wang 等人[17]通过启发 LLM 生成并构造与定理相关自定义引理库, 并证明这些引理. 尽管他们借助引理增强了 LLM 生成定理形式化证明草图的能力, 但其生成的引理仅限于与定理相似的引理, 未能利用引理来充分提升自动定理证明器对中间命题的证明能力. 本研究以定理的非形式化证明及其形式化证明草图为依据, 驱动 LLM 生成与定理整体及其中间命题相关联的引理, 充分展现了 LLM 在引理生成与形式化验证方面的潜力. 通过迭代方式证明引理, 进而建立起一个全新的自定义引理库.

2 LDPM 方法

引理库驱动证明方法 (lemma-library-driven proof method, LDPM) 结合了大语言模型 (如 ChatGPT[28,29]) 与交互式定理证明器 (如 Isabelle), 旨在提升自动定理证明器的证明能力. 大语言模型在接收特定输入后, 已表现出对学习模式与任务处理的强大适应力, 这一特性在研究中被定义为上下文学习 (in-context learning, ICL)[30]. 最近的研究[15–17,20]成功利用 ICL 自动生成了形式化证明草图和相关候选引理. 然而, 作为概率模型, 大语言模型 (如 ChatGPT) 并不能保证生成结果的真实性和可靠性. 另一方面, 交互式定理证明器通过用户输入的策略和定理名称, 逐步构造证明步骤并引导中间证明目标. 其小内核架构确保了每一步证明的可靠性. 然而, 交互式定理证明器的不足在于学习曲线陡峭且验证效率较低. 为此, 将大语言模型与交互式定理证明器相结合, 利用前者的任务处理能力与后者的高可靠性, 成为提升自动定理证明效率的一种可行方案.

LDPM 正是基于这一思路, 通过大语言模型生成初步的非形式化证明、形式化草图以及与之关联的候选引理陈述, 并对这些候选引理进行逐个证明以建立可靠的自定义引理库. 对于证明过程中失败的引理, 则调用 LLM 生成修正后的陈述以提升其可用性. 最后, 依托系统预定义定理/引理库和构建的自定义引理库, 交互式定理证明器对形式化证明草图所包含的中间命题进行验证, 从而实现定理证明的自动化和高效化. 这种方法不仅增强了引理库的多样性和可靠性, 还充分提升了自动定理证明器对中间命题的证明能力. 优化了整体的证明效率.

如图 1(a) 所示, 引理库驱动证明方法 (LDPM) 包含 5 个核心模块: 第 1 个模块是大语言模型 (LLM): 使用图 1(b) 中 LLM 请求格式的规则, 在不同阶段生成和获取预期的结果, 这包括生成非形式化证明、形式化证明草图和相关候选引理陈述. 在图 1(b) 中, 请求 LLM 的输入内容包括 4 个部分 (LLM 当前的角色、所处理任务的描述、多个示例和待解决的问题), 通过这 4 个部分的提示, LLM 能更准确地理解任务生成预期的结果. 第 2 个模块是自动定理证明器: 依托 Isabelle 中的自动证明工具 Sledgehammer, 按照图 1(c) Sledgehammer 请求格式的要求, 对 LLM 生成的形式化证明草图中的中间命题/猜想进行自动证明. 在图 1(c) 中, 自定义引理库构建完成后, 需在调用 Sledgehammer 之前将其加载到 Isabelle 的证明环境中, 借助自定义引理库辅助 Sledgehammer 证明中间命题, 获得可靠可信的定理形式化证明. 第 3 个模块是未证明引理矢量存储: 用于存储由 LLM 生成但尚未验证的候选引理陈述, 以便在进入正式验证流程前可被灵活调用与修改. 第 4 个模块是已证明引理矢量存储: 其所收录的引理均已完成验证, 包含引理陈述和形式化证明, 这一模块确保所有已被验证的引理能够用于后续证明过程. 第 5 个模块是自定义引理库: 在 Isabelle 中, 构建新的自定义引理库, 区别于其系统预定义定理/引理库, 包含经过验证的新引理, 为后续的形式化证明提供可靠的支持和扩展.

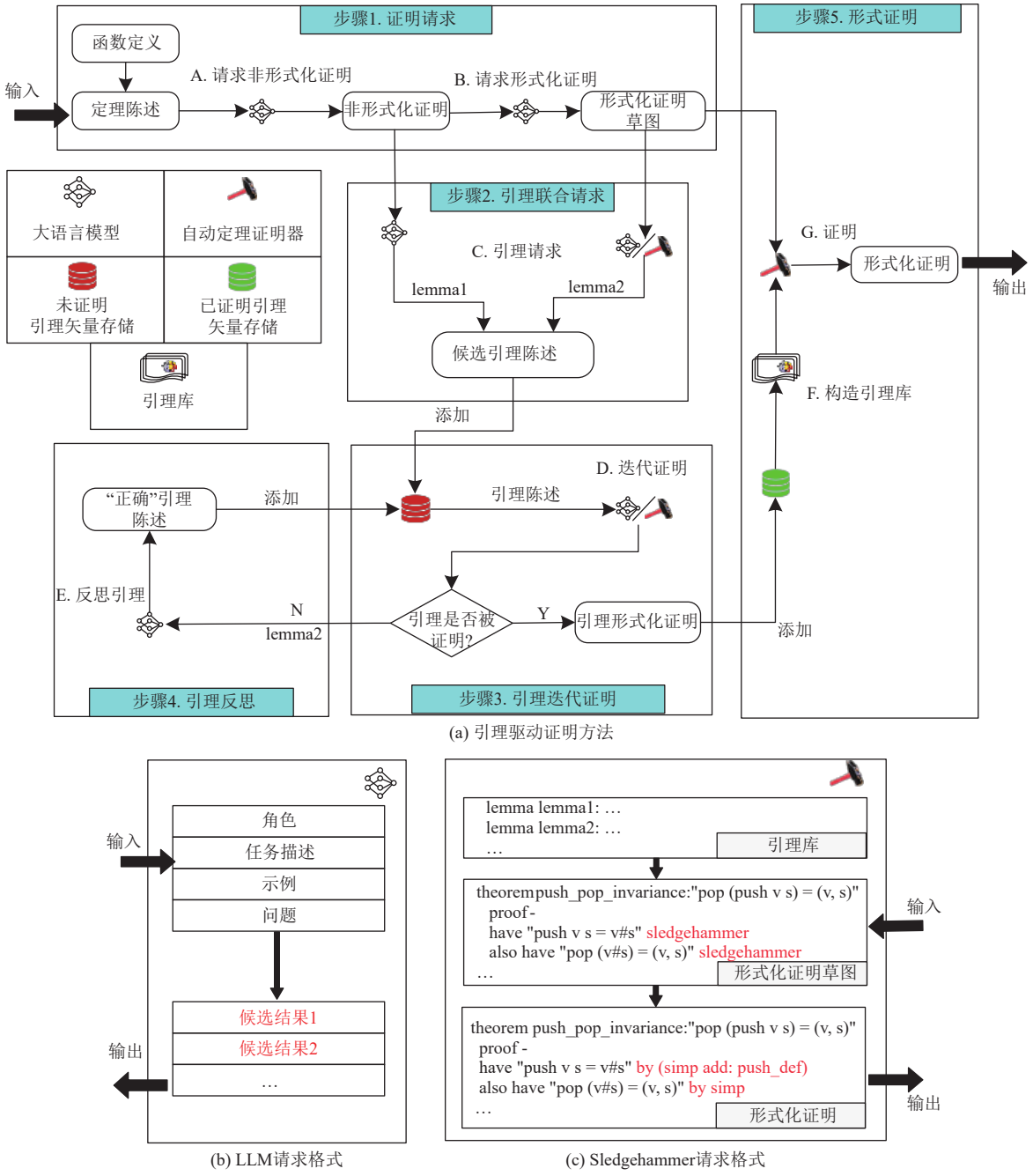

图 1 LDPM 技术路线概述

如图 1(a) 所示, LDPM (引理库驱动证明方法) 包含 5 个处理步骤: 步骤 1. 证明请求: 该步骤分为两部分, 非形式证明请求和形式证明草图请求. 首先, 根据定理陈述及其函数定义, 向 LLM 请求生成定理的非形式证明. 随后, 再请求 LLM 将非形式证明生成为形式证明草图, 为后续的自动定理证明奠定基础. 步骤 2. 引理联合请求: 在此步骤中, 采用两种不同的引导方式, 使 LLM 能够生成覆盖范围更广的候选引理陈述. 一是基于非形式证明的引理请求, 利用定理完整的非形式证明, 向 LLM 请求生成相关的候选引理 (Lemmas1), 这些引理有助于证明整个定理. 二

是基于形式化证明草图的引理请求机制, 如果现有自动定理证明器 (如 Sledgehammer) 证明不了草图中的中间命题, 系统将请求 LLM 生成针对性的引理 (Lemmas2), 以协助完成对该命题的验证. 步骤 3. 引理迭代证明: 在此步骤中, 将第 2 步生成的候选引理陈述添加到未证明引理的矢量存储中. 然后, 通过迭代的方式逐个证明这些引理陈述, 具体细节在第 2.3 节中讨论. 步骤 4. 引理反思: 如果在第 3 步中未能成功证明 lemma2, 就通过 LLM 的反思能力, 修改 lemma2, 生成更“正确”的引理陈述. 新的引理陈述将再次被添加至未证明引理矢量存储中, 并尝试继续证明. 步骤 5. 形式证明: 第 3 步中证明成功引理被收录至已证明引理矢量存储, 并借助这些引理在 Isabelle 中建立出全新的自定义引理库. 在此基础上, 运用该引理库, 借助现有自动定理证明器 (如 Sledgehammer) 对形式化证明草图内的各中间命题进行验证, 以保障最终形式化证明的准确无误. 通过这 5 个步骤, LDPM 实现了大语言模型与自动定理证明器的深度融合, 有效提升了数据结构算法定理证明的自动化和准确性.

所有涉及请求大语言模型的动作, 包括请求非形式化证明、形式化证明草图、候选引理和反思引理等, 每个请求包含了输入内容和输出内容. 更多细节见源代码文档, 具体可参阅 https://github.com/fuhengxing/LDPM.

2.1 证明请求

证明请求包括非形式化证明请求和形式化证明草图请求. 证明请求与 Jiang 等人[15]和 Wang 等人[17]的设置相似.

非形式化证明请求: 如源代码文档图 A1 所示, 首先在 Isabelle 中定义了数据结构的若干操作函数 (如增、删、改、查等). 接着, 使用一个定理描述了这些操作函数的某种属性. 随后, 采用 ICL 机制引导 LLM 生成该定理的非形式化证明. 为使其生成内容在结构上与 Isabelle 形式化证明的规范保持一致, 本研究规定非形式化证明须以递进式推理结构呈现, 从而复现 Isabelle 中形式化证明的典型表达范式.

形式化证明草图请求: 如源代码文档图 A2、图 A3 所示, 凭借已获得的非形式证明引导 LLM 构建定理的形式化草图, 该草图勾勒出初步的形式化推理结构, 其中所包含的中间命题/猜想还没有得到验证. 在此阶段, 对于这些待证明的中间部分, 我们暂以“Sledgehammer”作为占位符, 标识其证明策略有待后续由该自动化工具填充.

Sledgehammer 是一种强大的自动推理工具, 其合成证明策略极其依赖系统预定义定理/引理库, 借助启发式方法从中筛选相关定理/引理, 并调用外部自动定理证明器合成证明策略. 然而, 当面对复杂命题时, 单靠系统预定义定理/引理库, Sledgehammer 往往力不从心. 因此, 构建一个全新的自定义引理库, 不同于现有的定理/引理库, 成为关键问题. 而这一构建过程面临两大问题: 构造引理和证明引理.

2.2 引理联合请求

引理请求的目的是获取多样化的引理陈述, 以丰富引理库. 如图 1(a) 所示, 本文分别以非形式化证明和形式化证明草图为切入点引导 LLM 生成丰富的候选引理陈述, 这两种引理请求的方式分别是基于非形式化证明请求候选引理陈述和基于形式化证明草图请求候选引理陈述. LLM 请求生成候选引理时, 其输出包含解决该问题需要何种类型的引理、该引理的形式化陈述.

基于定理的非形式化证明请求候选引理陈述: 如源代码文档图 A4、图 A5 所示, 基于非形式化证明的推理步骤, 引导 LLM 构造出多个候选引理陈述, 这些引理对于解决目标定理可能具有潜在价值. 利用非形式化推理的引导作用, LLM 能够在缺乏严格形式化表达的前提下, 推理出能够辅助定理证明的引理. 这种方式侧重于从更高层次的推理逻辑中提取引理.

基于定理的形式化证明草图请求候选引理陈述: 如源代码文档图 A6、图 A7 所示, 当调用 Sledgehammer 证明形式化证明草图中的中间命题失败时, 依据形式化证明草图中的推理路径及其失败的中间命题, 引导 LLM 生成具有针对性的引理陈述. LLM 在这种情况下会尝试生成针对性的引理陈述, 帮助解决当前命题. 这种引理请求主要针对那些自动证明工具无法处理的难点, 提供补充.

这两种引理请求的方式有各自的特点, 第 1 次引理请求 (基于非形式化证明) 关注整个非形式化证明, 生成与定理整体推理相关的引理. 这些引理往往从全局角度出发, 涵盖更广泛的证明步骤. 第 2 次引理请求 (基于形式化证明草图) 更加精确, 集中在形式化证明草图中某个具体命题的解决. 它作为第 1 次引理请求的补充, 目的是生成一些容易被第 1 次引理请求忽略的引理. 实际上, 第 2 次引理请求的出现是为了弥补非形式化证明和形式化证明之间的差距. 这种差距源于非形式化证明的抽象性和形式化证明的严谨性之间的区别. 非形式化推理可能会忽略

某些细节, 而这些细节在形式化推理中尤为重要.

与 LEGO-Prover[17]方法不同, 本文除了使用基于定理的非形式化草图请求候选引理外, 还通过基于定理的形式化证明草图请求候选引理陈述的方式, 可以丰富引理种类.

采用联合请求策略构造引理时, 请求大语言模型加入线性数据结构操作信息, 以优先级队列为例, 其核心操作函数包括 *priority* 与 *min*. 其中, *priority* 函数依据给定的关键值查询该元素在队列中所对应的优先级; *min* 函数则用于获取当前队列头部元素的关键值. 利用上述函数有利于大语言模型理解定理 *min_priority_leq_all*(“*the* (*priority q* (*min q*)) ⩽ *the* (*priority q v*)”) 的含义, 此定理表示队列首元素的优先级最小. 大语言模型分别分析操作函数执行过程和优先级队列约束条件“*distinct* (*map f st xs*)”和“*sorted* (*map snd xs*)” (其中“*xs*”是优先级队列对应的列表, 函数 *distinct* 约束优先级队列元素关键值不重复, 函数 *sorted* 约束优先级队列中元素优先级有序), 理解目标定理的含义及证明细节. 构造引理 *distinct_fst_alist_of* 和 *sorted_alist_of*, 分别约束当前优先级队列元素值不重复和对应优先级有序排列. 此外, 根据优先级队列的操作函数, 可以将定理化简为一个中间命题“*the* (*map_of* (*alist_of q*)(*min q*)) ⩽ *the* (*map_of*(*alist_of q*) *v*)”. 根据上述优先级队列的相关信息, 有利于大语言模型构造针对线性数据结构算法定理的潜在引理, 如引理 *min_priority_leq_all_aux*, 并构造 4 个约束条件, 明确引理的使用范围, 有针对性的辅助验证中间命题. 更多细节见第 4.4.1 节线性数据结构.

2.3 引理迭代证明

本节提出的迭代证明方法 (iterative proof method, IPM) 是第 2 节 LDPM 方法的一个步骤, 它或者调用直接证明方法 (direct proof method, DPM) 来直接证明引理, 提高引理证明的效率, 或者递归调用 LDPM 来进行迭代证明引理. 这是由于证明一个定理可能需要证明引理, 而证明引理中间可能又需要证明子引理, 因此需要迭代证明引理. 整个函数调用关系是: LDPM (作用于定理) 调用 IPM, IPM 调用 DPM 或递归调用 LDPM (作用于引理).

引理库只能存储完成形式化验证并证明成功的引理, 确保这些引理在后续的定理或引理证明中能够被安全使用. 本文通过迭代方式对这些引理进行证明, 从而建立起一个全新的自定义引理库.

2.3.1 直接证明方法 (DPM)

如图 2 所示, 直接证明方法 (DPM) 与图 1 引理库驱动证明方法 (LDPM) 技术路线类似, 但不需要步骤 2–4. 以证明引理 *alist_of_push* (向优先队列中插入新元素) 为例, 具体步骤如下.

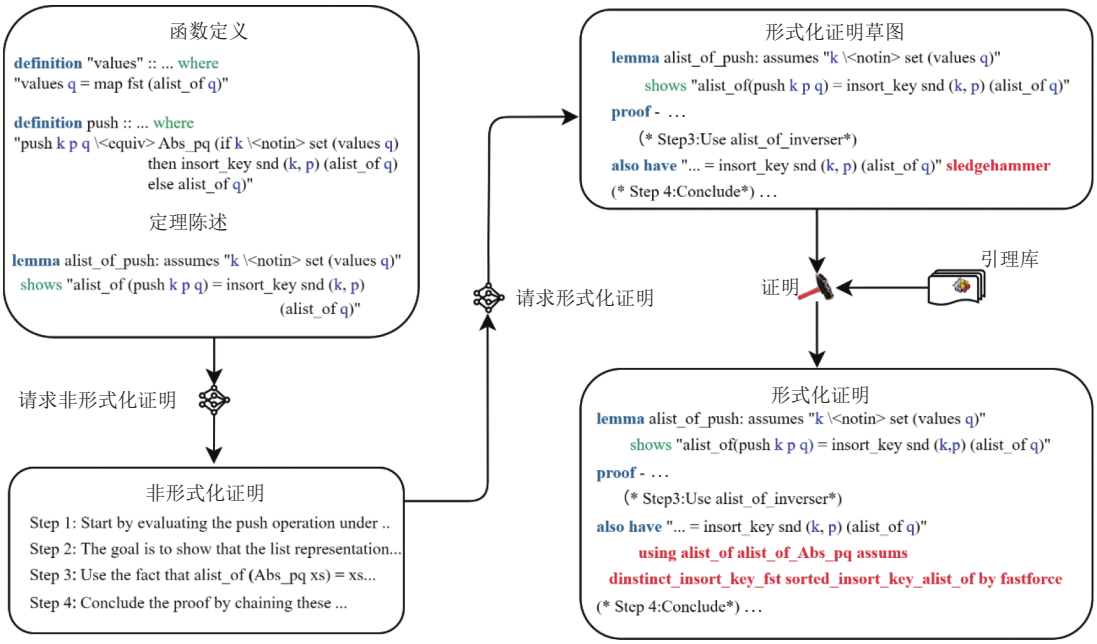

图 2 直接证明方法 (DPM)

非形式化证明请求: 首先根据定理及其函数定义, 提示 LLM 生成引理 *alist_of_push* 的非形式化证明.

形式化证明草图请求: 凭借已获得的非形式证明引导 LLM 构建定理的形式化草图, 该草图勾勒出证明的总体结构, 其中所包含的中间命题或猜想尚未得到验证.

自动验证阶段: 在这一阶段, 会在 Isabelle 中加载系统预定义和自定义引理库 (来自第 2.5.1 节), 并请求 Sledgehammer 对形式化证明草图中的中间命题/猜想进行证明. Sledgehammer 会根据中间命题/猜想从引理库中选择相关的引理, 例如引理 *distinct_insort_key_fst* (确保优先队列执行插入操作后键的唯一性) 和引理 *sorted_insort_key_alist_of* (确保优先队列执行插入操作后仍然保持有序), 并调用外部自动定理证明器合成证明策略, 最终通过 Isabelle 的小内核进行验证, 确保整个形式证明的可信度.

与本节引理库驱动证明方法 (LDPM) 不同, DPM 并不参与新引理的构造与验证, 其核心任务是在已有自定义引理库的支持下, 完成对形式化证明草图的验证. DPM 的设计目的是快速验证 LDPM 中构造的引理陈述, 提升整体证明效率.

尽管 DPM 可以借助系统预定义定理/引理库提高定理证明的效率, 但面对更大规模更为复杂的定理证明任务, 还需构造自定义的引理库. 引理库驱动证明方法 (LDPM) 则建立了一个全新的自定义引理库, 以增强 Sledgehammer 合成证明策略的能力, 从而提升定理证明的效率.

2.3.2 迭代证明方法

本文提出的迭代证明方法, 通过协同使用直接证明方法 (DPM) 与引理库驱动证明方法 (LDPM), 实现了对引理的多阶段验证. 该组合策略能够针对不同引理的证明需求灵活调整验证路径, 从而最大化验证效率. 具体而言, 验证过程首先尝试使用 DPM, 其验证工作主要依赖系统预定义引理库提供的支持. 若 DPM 在某一定理的证明中失败, 则系统转而采用 LDPM 继续验证. LDPM 通过大语言模型生成新的引理, 并在此基础上构建自定义引理库, 以应对复杂性较高的证明任务. 无论经由 DPM 还是 LDPM 成功验证的引理, 都会被添加到自定义引理库中. 随着引理库的不断扩展, 后续的定理和引理证明会变得更加高效.

算法 1 调用 DPM 来直接证明引理, 提高引理证明的效率, 或者递归调用 LDPM 来进行迭代证明引理, 从而在证明效率与复杂问题处理之间取得平衡.

算法 1. 迭代证明方法 (IPM).

输入: 未被证明的引理 *lemma*, 循环边界 *l*1 和 *l*2, 大语言模型 *LLM*, 形式化验证环境 *isabelle*, 已被证明引理矢量存储 *VS*, 引理迭代深度 *depth*;
输出: 更新后的矢量存储 *VS*.

1. *i* = 0;
2. *j* = 0;
3. *success* = false;
4. *formal* = None; //表示引理的形式化证明
5. **while** not success and *i* < *l*1 **do**{
6. *formal* = DPM(*lemma*, *LLM*, *isabelle*, *success*); // DPM 验证
7. *i*++;}
8. **if** *success* **then**
9. *VS*.add(*lemma*, *formal*);
10. return *VS*
11. **if** *depth* > 0 **then** // 检查引理迭代深度
12. **while** not *success* and *j* < *l*2 **do**{
13. *formal* = LDPM(*lemma*, *LLM*, *isabelle*, *success*, *depth*–1); // LDPM 验证, 引理迭代深度减 1

14. *j*++;}
15. **if** *success* **then**
16. *VS*.add(*lemma*, *formal*); //添加引理形式化证明到矢量存储中
17. return *VS*

该算法的执行步骤如下. 定义变量 (第 1–4 行): 定义循环边界 *i* 和 *j*, 分别代表 DPM 和 LDPM 的尝试次数. 变量 *success* 表示引理是否被验证成功, 初始为 false. 变量 *formal* 初始为 None, 用于表示引理的形式化证明. DPM 证明尝试 (第 5–7 行): 在一个循环中反复尝试使用 DPM 证明引理. 通过 *success* 来判断引理是否被证明成功, 并判断 DPM 尝试次数 *i* 是否超过设定的循环边界 *l*1. 如果引理证明成功或超过最大循环边界, 则退出循环; 否则继续调用 DPM. 检查引理是否证明成功 (第 8–10 行): 如果引理最终被成功证明, 则将其形式化证明结果添加到已证明引理的矢量存储中, 并返回新的引理矢量存储. 判断引理迭代深度 (第 11 行): 如果引理的迭代深度 *depth* 小于或等于 0, 则不再执行 LDPM, 即使 DPM 没有证明引理也不会进入 LDPM 阶段. LDPM 证明尝试 (第 12–14 行): 如果 *depth* 大于 0, 并且 DPM 未能成功证明引理, 则进入 LDPM 阶段. 在 LDPM 中, 请求 LLM 生成子引理, 并通过这些子引理辅助证明引理. 同时, 减少引理的迭代深度 *depth*. 继续循环判断引理是否被证明, 并检查 LDPM 的循环边界 *j* 是否超出循环边界 *l*2. 成功证明后的处理 (第 15、16 行): 如果引理最终被成功证明, 则将其形式化证明结果添加到已证明引理的矢量存储中. 最后 (第 17 行): 返回包含新证明引理的矢量存储, 供后续定理证明使用.

采用迭代证明策略证明引理时, 请求大语言模型利用前面迭代相关信息, 例如之前已构建的引理库, 此引理库是面向线性数据结构的, 依赖引理库逐个证明候选引理, 不断增强引理库可信性. 例如在证明引理 *alist_of_push* (“$k \notin set\ (values\ q) \Rightarrow alist\_of\ (push\ k\ p\ q) = insort\_key\ snd\ (k,p)(alist\_of\ q)$”), 表示当 *k* 不在优先级队列中时, 使用两种方式将新元素插入到优先级队列, 两种方式的效果等价. 一种是通过函数 *push* 插入新元素到优先队列中, 另一种是通过函数 *insorted_key* 将新元素插入到优先级队列对应的列表中) 时, 使用函数 *alist_of* 化简 *push*, 即使用规则 *Abs_pq_inverse*(“$?y \in \{xs.distinct\ (map\ f\ st\ xs) \wedge sorted\ (map\ snd\ xs)\} \Rightarrow alist\_of\ (Abs\_pq\ ?y) = ?y$”). 但要使用规则 *Abs_pq_inverse*, 必须满足两个约束条件 *distinct* (*map fst xs*) 和 *sorted* (*map snd xs*), 分别约束优先级队列关键值不重复和元素有序排列, 也就必须要当前优先级队列满足这两个约束条件. 首先会从自定义引理库中检索这两个约束条件是否被证明, 若已证明, 则可直接使用规则 *Abs_pq_inverse* 证明引理 *alist_of_push*; 若未证明, 则请求模型构造这两个约束条件来约束当前优先级队列, 使其能适用规则 *Abs_pq_inverse*. 例如大语言模型会构造引理 *distinct_insort_key_fst*(“$distinct\ (map\ f\ st\ (insort\_key\ snd\ (k,p)\ (alist\_of\ q)))$”) 和引理 *sorted_insort_key_alist_of*(“$sorted\ (map\ snd\ (insort\_key\ snd\ (k,p)\ (alist\_of\ q)))$”). 引理 *distinct_insort_key_fst* 表示元素插入优先级队列后关键值依然不重复, 引理 *sorted_insort_key_alist_of* 表示元素插入优先级队列后元素依然有序排列. 证明这两个引理, 并添加到引理库中, 引理 *alist_of_push* 可以适用规则 *Abs_pq_inverse*. 引理 *alist_of_push* 通过迭代的方式得以证明.

2.4 引理反思

引理反思的机制旨在通过 LLM 的自反思改进生成的引理. 在构造引理时, LLM 容易陷入推理的“思维漩涡”, 产生幻觉或错误结论[31]. 为了解决这一问题, 引理反思基于 LLM 的自我评估或外部反馈来调整和改进输出, 从而提升生成引理的质量.

在 LDPM 的第 2 步引理联合请求中, 在定理的形式化证明草图中, 若存在某个中间命题未能被直接证明, 则请求 LLM 生成与该命题相关的候选引理陈述 (Lemmas2), 目的是辅助该命题的证明. 在 LDPM 的第 4 步引理反思中, 基于引理 lemma2 的证明失败, LLM 通过反思来分析引理为何无法被证明, 并提出新的引理, 如源代码文档图 A8、图 A9 所示. 通过 LLM 自我评估, LLM 生成一个经过调整的新引理, 这个新引理被设计成更加适合当前的证明环境, 旨在解决未能被验证的中间命题. 新的引理会再次进入验证阶段.

在执行引理反思策略的过程中, 需引导大语言模型结合线性数据结构的相关信息进行分析, 例如在反思引理 *alist_of_push*(“$alist\_of\ (push\ k\ p\ q) = insort\_key\ snd\ (k,p)(alist\_of\ q)$”) 时, 在之前大模型构造引理时, 发现可以通过函数 $alist\_of$ 来化简函数 $push$: $alist\_of\ (push\ k\ p\ q) = insort\_key\ snd\ (k,p)(alist\_of\ q)$. 通过分析线性数据结构

操作函数 *alist_of*、*push* 和 *insort_key* 等 (其中, 函数 *alist_of* 表示将优先队列 q 转换成列表形式. 函数 *push* 检查判断待插入元素的关键字是否存在于优先队列中, 若不存在则插入. 函数 *insort_key* 将元素插入到列表中), 大语言模型理解引理陈述的含义, 分析引理陈述的不足 (比如证明此引理陈述缺少前提条件). 发现元素 k 需要满足约束条件: $k \notin set\ (values\ q)$, 即元素 k 不在优先级队列 q 中. 当 k 满足上述条件时, 引理陈述 $alist\_of\_push$ 才成立. 因此大语言模型通过对定理的证明过程分析, 修正原来的引理陈述并给出新的引理陈述 $alist\_of\_push$, 此时引理陈述 $alist\_of\_push$ 满足约束条件 $k \notin set\ (values\ q)$.

2.5 形式化验证

2.5.1 自定义引理库构建

本文采用矢量存储技术辅助构建引理库, 利用 ChromaDB 作为矢量存储数据库, 帮助管理和检索引理数据. 该矢量存储数据库的每个存储单元由 3 个核心要素组成: 文档、元数据以及嵌入向量. 其中, 文档通常对应引理的形式化陈述或其完整的证明内容; 元数据则记录了该引理关联的数据结构信息; 嵌入向量是由嵌入模型对引理陈述或用户查询实施编码映射后所生成的数值表示. 当系统获得查询请求时, 会调用相同的嵌入模型将查询内容转换为嵌入向量, 进而以余弦相似度为度量标准, 在已有存储中检索与该查询向量最匹配的文档, 最终返回对应的引理信息.

为对处于不同验证阶段的引理进行系统化管理, 本文构建了两类矢量存储结构: 其一为未证明引理矢量存储, 主要收录通过引理请求获得的候选引理陈述. 所有存入的引理陈述均经过语法检查, 以确保其形式规范性; 元数据部分记录了该引理关联的数据结构信息, 便于之后索引并调用. 在图 1(a) 所示的 LDPM 流程中, 第 3 步将验证未证明引理矢量存储中的引理. 其二为已证明引理矢量存储, 用于收录通过形式化验证的引理, 涵盖引理陈述与完整的形式化证明, 也使用元数据记录该引理关联的数据结构信息. 为进一步提升引理的复用效率, 本研究在 Isabelle 环境中依据数据结构类别分别构造专属的引理库. 在证明过程中, 属于同一数据结构的定理及其衍生引理共享该库资源, 使得已验证的引理能够在同类证明中反复使用, 大幅降低了重复构建的开销. 当一条引理成功完成形式化验证后会存入已证明引理矢量存储, 并同步更新对应数据结构的引理库内容. 随着定理/引理的证明不断演进, 不仅能保证引理库的及时更新, 还能够利用已验证的引理加速后续定理的证明过程, 提高整个定理验证的效率.

自定义引理库的构建是一个渐进的过程: 初始状态下引理库内容较为有限甚至为空; 随着定理证明的逐步推进, 每一条新获证明的引理都会被纳入其中, 最终实现引理库的不断积累与扩展.

2.5.2 形式化证明草图验证

如图 1(a) 所示, 形式化验证是引理库驱动证明方法的最后一步, 其目的是确保由 LLM 生成定理的形式化证明草图的正确性. LLM 生成的形式化证明草图通常包含多个未被证明的中间命题. 在此步骤中, 将已经构建的自定义引理库加载到 Isabelle 的证明环境中, 利用 Isabelle 中的自动定理证明工具 Sledgehammer, 逐步合成这些中间命题的证明策略. Sledgehammer 通过整合系统预定义定理/引理库与自定义引理库中的引理资源, 生成适配的证明策略, 从而对形式化证明草图所包含的各个中间命题进行验证.

3 实 验

3.1 数据集和评估

本研究构建了 5 类线性数据结构算法定理, 共计 50 条, 覆盖了多种线性数据结构类型及其核心操作, 具体如下.

栈 (stack) 部分共涵盖 5 个定理, 这些定理形式化地刻画了元素在进栈与出栈操作中所遵循的行为关系. 栈的操作遵循后进先出 (LIFO) 原则. 例如, 定理 *push_pop_identity* 通过逻辑等式 $\forall v\ s.\ pop\ (push\ v\ s) = (v, s)$ 形式化地验证了 *push* 和 *pop* 操作的协同工作符合栈的预期行为: 推入一个元素后立即弹出, 应得到原栈和推入的元素.

有界栈 (bounded stack) 部分共涵盖 8 条定理, 与基础栈结构相比, 这些定理在涵盖其标准操作的同时, 额外引入了对栈容量的明确限制. 例如, 定理 *bounded_stack_push_pop_inverse*("$\neg isfull\ s \Longrightarrow pop\ (push\ x\ s) = (Some\ x, s)$") 表示了在有界栈栈未满的情况下, *push* 操作后立即执行 *pop* 操作会正确返回被推入的元素和原始的栈状态.

先进先出队列 (FIFO queue) 部分共涵盖 4 个定理, 用于刻画元素在入队与出队操作中所遵循的行为逻辑.

先进先出队列的操作严格遵循先进先出原则, 即元素的出队顺序与入队顺序保持一致. 例如, 定理 *deq2list_ind* (“*deq2list q xs = rev q@ xs*”) 表示将队列 q 中的所有元素依次出队并添加到列表 xs 的前面, 最终结果等于将队列 q 的内部列表直接与 xs 连接.

优先级队列 (priority queue) 部分共涵盖 7 个定理. 该数据结构以列表形式组织, 列表中每个元素均由键及其对应的优先级共同构成一个有序对. 元素的出队次序由优先级数值决定, 数值越小则出队次序越靠前. 例如, 定理 *min_priority_leq_all*(“$[\![\neg is\_empty\ q; v \in set\ |q|]\!] \Rightarrow the\ (the\ priority\ q\ (min\ q)) \leqslant the\ (priority\ q\ v)$”) 表示对于所有非空的优先级队列 q, 以及队列中存在的任意元素 v, 队列中优先级最小的元素的优先级不大于 v 对应的优先级.

子列表 (sublist) 部分共涵盖 26 个定理, 主要描述列表间的前缀关系、后缀关系和子序列的性质. 前缀关系是列表的一种偏序关系. 例如, 定理 *prefix_snoc*(“$prefix\ xs\ (ys@[y] \leftrightarrow xs = ys@[y] \vee prefix\ xs\ ys$”) 表示列表 xs 是 ys 追加元素 y 后的新列表的前缀, 当且仅当 xs 等于 $ys@[y]$ 或者 xs 已经是 ys 的前缀.

每组数据结构包括定理的形式化陈述以及与之相关的数据结构的定义, 所有这些内容都在 Isabelle 中进行表达. 源代码文档图 B1 展示了优先队列数据结构操作定义和相关定理形式化陈述.

本研究旨在以自动化方式生成定理的形式化证明, 同时保证所得证明的正确无误. 证明的正确性主要依据两个标准进行判定: 1) 证明文本必须完全摒弃“sorry”和“oops”这类用于临时跳过证明过程的关键字, 杜绝不完整的推导. 2) 每个证明都必须通过 Isabelle 定理证明器的完整校验, 且验证过程零错误, 以此作为证明正确的最终依据. 本系统采用 Isabelle-client API[32]作为核心接口从而实现与 Isabelle 的高效程序化交互, 完成形式化证明的提交、执行与结果获取.

3.2 基　线

● Sledgehammer. 本文选取 Isabelle 2022 环境中的 Sledgehammer 作为基准证明工具, 用于对线性数据结构算法定理进行自动证明尝试. Sledgehammer 的参数配置沿用其在 Isabelle 2022 中的默认设定: 每个证明请求的超时时间设定为 30 s, 并调用 6 个外部自动定理证明器协同工作, 即 CVC4[33]、veriT[34]、Z3[35]、SPASS[36]、Vampire[37] 和 Zipperposition[38].

● DSP. 该方法由 Jiang 等人[15]设计, 其操作流程可分为 3 个环节: 首先, 通过一定方式获得定理的非形式化证明; 其次, 以此为基础驱动语言模型生成相应的形式化证明草图; 最后, 采用自动定理证明器对草图内容进行验证. Jiang 等人将 DSP 方法应用于中学数学定理的形式化验证, 实际测试表明该方法能够有效生成证明. 本研究在其工作基础上, 完整复现并调整了该方法的流程, 以 ChatGPT 为核心生成模型, 进而将其扩展至线性数据结构相关算法定理的证明任务中.

● DSP$^+$. 作为 DSP 方法的扩展, DSP$^+$被纳入本研究的基准方法集. 与原始 DSP 一致, DSP$^+$同样借助 ChatGPT 生成定理的非形式化证明与形式化证明草图, 但其关键改进在于增加了基于非形式化证明的引理请求机制. 该机制旨在通过引理辅助定理证明, 并在草图验证过程中调用这些引理, 从而增强定理的可证明性.

3.3 实验设置

本研究的方法流程中, 以 ChatGPT 为大语言模型, 设定参数 *temperature* (用于控制大语言模型生成文本随机性的超参数) T=0.7. 每个定理经历 25 次非形式化证明生成, 每次非形式化证明进一步产生 2 个形式化证明草图, 故每个定理需完成对 50 个形式化证明草图的验证. 只要有一个形式化证明草图验证通过, 则表示定理证明成功. 在引理生成环节, 每轮请求中 ChatGPT 输出 5 条候选引理. 候选引理进入未证明引理矢量存储须满足两个条件: 该候选引理与定理之间嵌入向量的余弦相似度大于 0.2, 且与所有已证明引理嵌入向量的余弦相似度大于 0.08. 每次调用 ChatGPT 时, 输入内容遵循图 1(b) 的格式, 包括任务描述、1–3 个示例和待解定理, 示例个数依据输入文本长度动态决定. 同时, 我们从提供的示例集中选取与当前定理相近的示例, 并保证所采示例不涉及正在证明的定理. 在引理迭代证明时, 直接证明方法 DPM 循环边界 $l1$=1, 即定理证明时无需调用引理; 引理库驱动证明方法 LDPM 循环边界 $l2$=5, 即定理证明时需要调用引理, 大模型会生成 5 个候选引理, 引理迭代深度 *depth*=2, 即定理需要调用引理, 引理又可以调用子引理.

3.4 主要结果

表 1 结果显示, DSP 方法在线性数据结构算法定理的证明中取得了 44.00% 的成功率, 相比 Sledgehammer 基准方法提升 8.00%, 并多完成了 4 个定理的证明. 该增益主要得益于大语言模型对复杂定理的化简与推导能力, 通过将复杂定理合理划分为多个相对简单的子命题, 使得证明难度降低. 随后, DSP$^+$方法将成功率提升至 62.00%, 相比 DSP 方法实现了 18.00% 的增长, 并多证明了 9 个定理. 其改进策略体现为借助新引理对定理证明提供辅助支持, 有效缓解了大语言模型在生成长篇幅形式化证明时易出错的局限, 使证明过程趋于简化和模块化.

表 1 不同证明方法在线性数据结构算法定理上的验证成功率

分类	方法	stack	bounded stack	FIFO queue	priority queue	sublist	成功率 (%)
基线	Sledgehammer	5	1	1	2	9	36.00
	DSP[15]	5	3	2	2	10	44.00
	DSP+引理 (DSP$^+$)	5	6	2	2	16	62.00
	LDPM	5	8	2	6	20	82.00
消融	–联合请求	5	8	2	3	14	64.00 (−18.00)
	–迭代证明	5	6	2	2	16	62.00 (−20.00)
	–引理反思	5	8	2	4	14	66.00 (−16.00)

LDPM 在本研究涉及的线性数据结构定理验证中达到了 82.00% 的成功率, 较 DSP$^+$方法成功率提高 20.00%, 额外证明 10 个定理. 其核心优势在于其引入了引理库驱动的证明方法, 设置了一个线性数据结构算法定理共享的自定义引理库, 并收录所有证明成功的引理. 借助该库, 定理之间的内在联系得以强化, 信息得以复用, 进而大幅提高了定理证明的成功率.

表 2 结果显示, 基准方法 Sledgehammer 的在线性数据结构算法定理上的验证总时间仅为 90.00 s, 因为它无需对定理陈述的形式化证明草图进行证明, 而是对定理陈述直接使用 Sledgehammer 工具进行自动化验证. 同时, 该方法未使用大语言模型, 因此无需计入 LLM 调用时间. DSP$^+$方法的总验证时间为 6 027.63 s, 约为 DSP 方法总验证时间 (1 848.76 s) 的 3 倍. 这是因为在引入新引理辅助定理证明的过程中, 需要额外的 LLM 调用时间和 Isabelle 验证时间, 从而增加了总体时间成本. LDPM 方法的总验证时间为 4 520.72 s, 约为 DSP$^+$方法的 0.75 倍. 这是由于 LDPM 设置了一个线性数据结构算法定理共享的自定义引理库, 并保存所有证明成功的引理, 从而减少了 LLM 调用时间和 Isabelle 验证时间.

表 2 不同证明方法在线性数据结构算法定理上的时间成本

基线	类型					时间成本 (s)		
	stack	bounded stack	FIFO queue	priority queue	sublist	LLM调用时间	Isabelle验证时间	验证总时间
Sledgehammer	5	1	1	2	9	—	90.00	90.00
DSP[15]	5	3	2	2	10	214.86	1 633.90	1 848.76
DSP+引理 (DSP$^+$)	5	6	2	2	16	1 700.07	4 327.56	6 027.63
LDPM	5	8	2	6	20	1 275.05	3 247.67	4 520.72

综合表 1 和表 2 的结果可知, LDPM 方法在本研究所涉及的线性数据结构定理验证任务中, 与最先进的方法比较, 验证成功率从 44.00% 提高到 82.00%. 尽管验证总时间约为最先进方法的 2.5 倍, 但在自动定理证明中, 验证成功率是首要评价指标, 在准确性得到显著提升的前提下, 这个时间开销是可接受的.

4 分 析

4.1 消融研究

引理库的持续扩展显著增强了现有自动定理证明器 (如 Sledgehammer 所整合的 CVC4、VeriT、Z3、SPASS、

Vampire 和 Zipperposition 这 6 个外部证明器) 的定理证明能力. 为了展示自定义引理库对这些自动定理证明器证明定理的影响, 本文进行了消融实验, 分别给出了取消不同策略的实验效果, 并比较系统在整个数据集上的性能差异, 见表 1 和图 3. 通过与基线 DSP$^+$ 的成功率对比可以看出, 这 3 种策略在有界栈 (bounded stack)、优先队列 (priority queue) 和子列表 (sublist) 上分别多证明了 2 条、4 条和 4 条定理, 结果具有显著差异. 在取消引理联合请求策略的情况下, LDPM 的验证通过率下降至 64.00%. 在取消引理迭代证明策略的情况下, LDPM 的验证通过率下降至 62.00%. 在取消引理反思策略的情况下, LDPM 的验证通过率下降至 66.00%. 由此看出引理迭代证明策略作用最大, 引理反思策略作用最小.

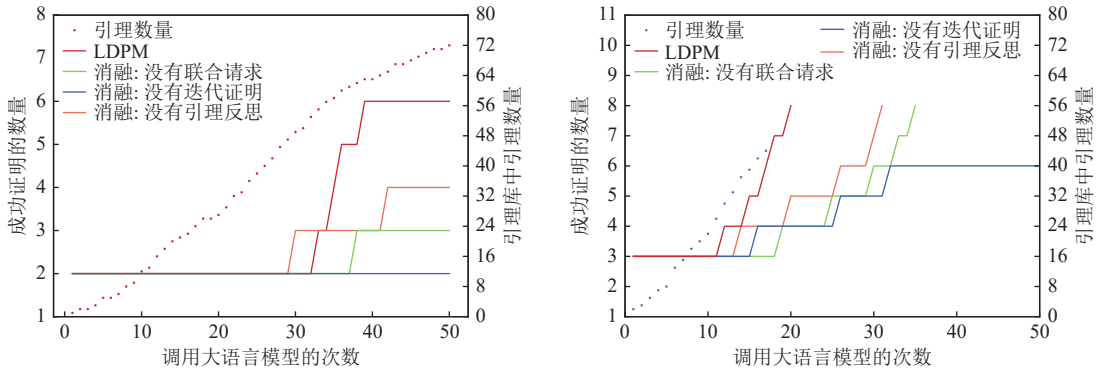

图 3 LDPM 证明定理数量的消融实验

如图 3 所示, 起初引理库对定理证明没有显著差异, 但随着调用大语言模型的次数增加, 库中的引理数量不断增长 (散点), 这种积累逐渐对定理证明表现出重要影响. 例如: 图 3(a) 优先级队列定理证明中, 在有引理库的情况下, 使用 LDPM 方法优先级队列相关的定理得到了很好的解决, 完成了 6 个定理证明. 然而, 在取消引理联合请求策略的情况下, 验证了 3 个定理. 在取消引理迭代证明策略的情况下, 只验证了 2 个定理. 在取消引理反思策略的情况下, 验证了 4 个定理. 图 3(b) 有界栈定理证明中, 引理库的扩充有效提高了定理的证明成功率, 并且在调用 LLM 的次数明显减少的情况下, LDPM 方法完成了 8 个定理证明. 在取消引理联合请求策略的情况下, 验证了 8 个定理. 在取消引理迭代证明策略的情况下, 只验证了 6 个定理. 在取消引理反思策略的情况下, 验证了 8 个定理. 相比之下, 这 3 种情况所需的 LLM 调用次数明显多于 LDPM 方法.

这一实验结果进一步验证了本文的假设: 自定义引理库有效辅助现有自动定理证明器证明定理, 并随着引理库的不断扩展, 大大加速定理的证明过程. 这表明构建和扩展引理库是提高定理证明成功率的关键因素, 尤其在处理线性数据结构算法定理时效果尤为显著.

4.2 引理库对自动定理证明器证明能力的影响

通过从引理库中筛选相关的引理, 可以扩展自动定理证明器对证明子目标的搜索路径, 以更好地合成证明策略, 从而提高定理证明的效率. 如图 4 所示, 首先基于定理 *min_priority_leq_all* (确保优先队列中队首元素的优先级最小) 非形式化证明和形式草图证明请求 LLM 生成相关引理陈述 *distinct_fst_alist_of* (当优先队列非空时, 将其转化成键值对列表, 且所有键值对的第 1 个元素是唯一的, 不存在重复)、*min_priority_leq_all_aux* (在一个按键值对的优先级有序排列且键唯一的键值对列表中, 头部元素的优先级小于键值对列表中任意其他元素的优先级) 和 *sorted_alist_of* (在生成的键值对列表中, 对所有键值对的第 2 个元素按顺序排列). 接着, 逐一证明这些引理, 确保其正确性, 并将证明成功的引理添加到已证明引理矢量存储中, 刷新引理库以加载最新的引理. 最后, 借助已更新的引理库, Sledgehammer 能够筛选相关引理, 从而合成中间命题 (定理 *min_priority_leq_all* 形式化证明中的中间命题 h4) 的证明策略, 以确保定理的严格正确性.

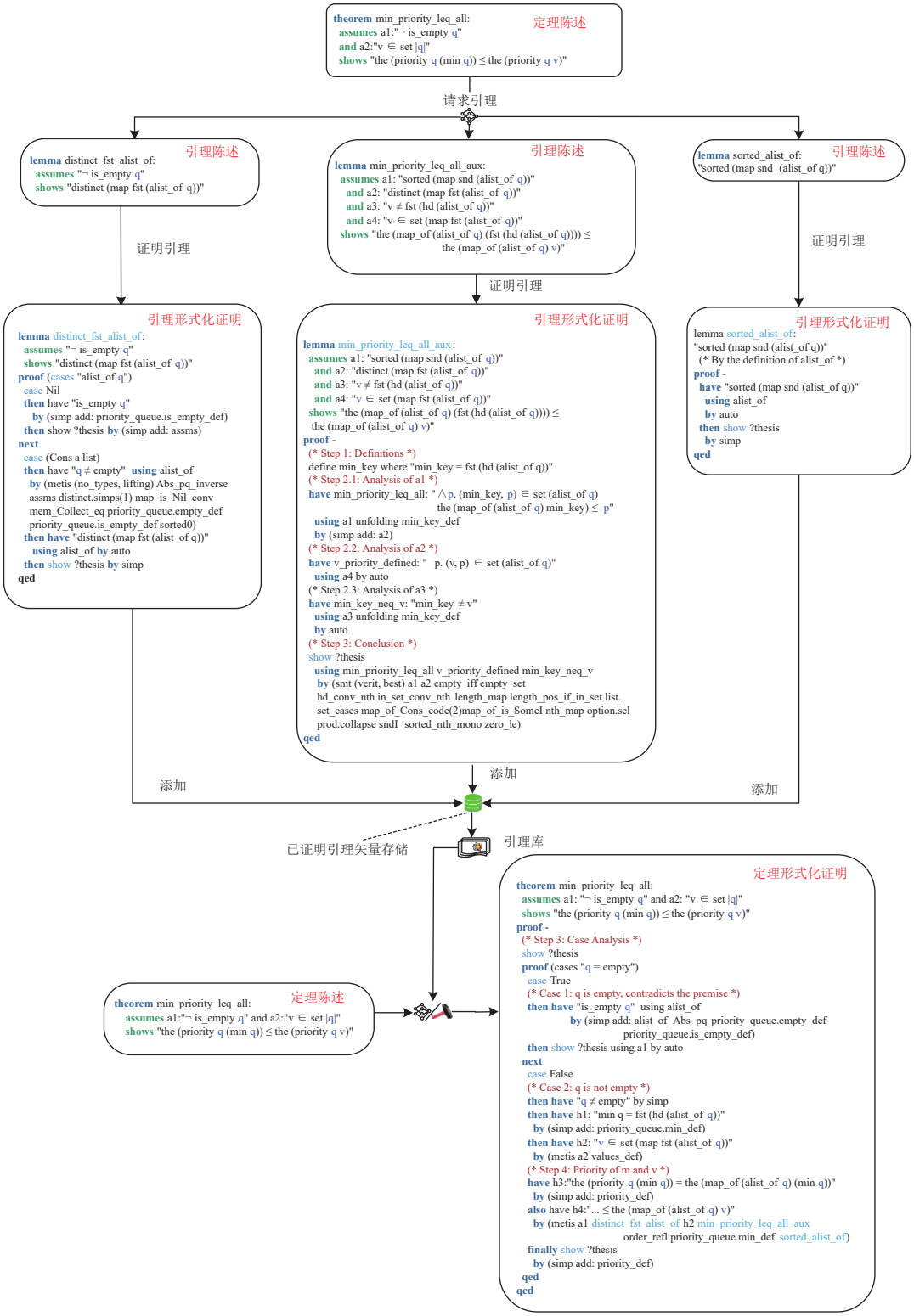

图 4　定理 *min_priority_leq_all* 证明

4.3 3 个策略对引理库扩展的影响

图 5 展示了引理库中不同子集的引理数量及其有效性. 图 5(a) 显示了针对数据结构优先级队列构建的引理库子集, 图 5(b) 则展示了针对数据结构有界栈构建的引理库子集. 每个引理库被分为 3 个子集: 第一, 基于定理的非形式化证明请求的引理 (Lemmas1); 第二, 基于定理的形式化证明草图请求的引理 (Lemmas2); 第三, 通过引理反思获取的引理 (Reflection lemmas). 蓝色柱状对应各子集中的引理总数; 黄色柱状则代表在定理证明过程中实际被调用的引理数量, 反映了对定理证明的有效性; 绿色柱状表示在引理证明过程中引用了该子集中引理的数量, 反映了对引理证明的有效性. 这些数据展示了引理库中通过不同方式构造的子集对线性数据结构算法定理证明的支持程度.

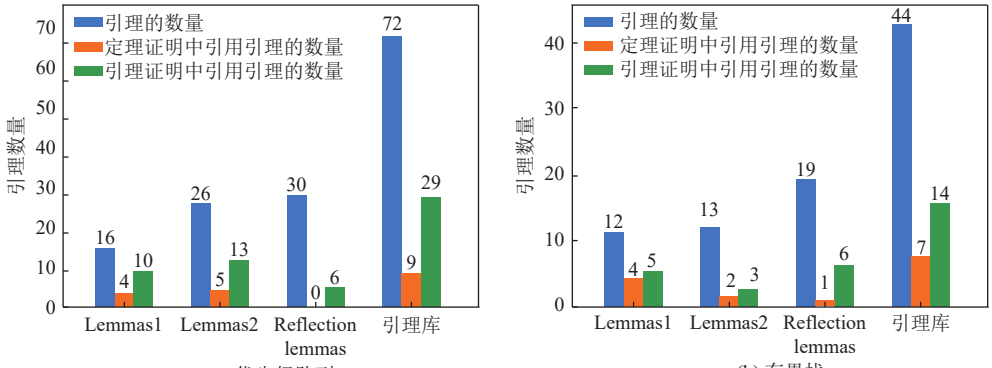

图 5 引理库中不同子集数量及其有效性的比较

● 引理请求. 引理的构造依托于联合请求策略, 具体分为两条途径: 其一, 基于定理的非形式化证明, 引导大语言模型生成引理 (Lemmas1); 其二, 基于定理的形式化证明草图, 引导大语言模型生成引理 (Lemmas2). 以图 5(a) 所示优先级队列为例, 最终通过验证的引理总数达 72 个, 其中 16 个来源于 Lemmas1, 26 个来源于 Lemmas2. 两类引理合计占比 58.33%, 有效丰富了引理库的内容, 增强了其对后续定理证明的支撑能力. 图 5(b) 则显示, 这两种方式构造且被验证的引理在数据结构有界栈中占 56.82% (引理库中引理的数量达到 44 个, 其中 12 个来自 Lemmas1, 13 个来自 Lemmas2), 同样有效地增强了其引理库的内容.

● 引理证明. 采用迭代方式证明这些引理, 保证引理库扩展后的可信性. 如图 5(a) 的绿色柱状所示, 在引理库中, 约 40.3% 的引理在证明时引用了其他引理, 这显著促进了引理的证明. 从有效性指标来看, Lemmas1 子集表现最佳, 共计 10 条引理被后续证明所采用, 有效性达 62.5%. Lemmas2 子集虽然被引用的引理数量最多, 但其有效性稍逊, 为 50.0%. 而 Reflection lemmas 子集则在引用规模与有效性两个维度上均位列最末. 在图 5(b) 中, 子集 Lemmas1 同样展现出最高的有效性, 该现象进一步凸显了 Lemmas1 子集中引理的独特价值, 其内在原因可能与引理的构造路径密切相关: Lemmas1 依托定理的非形式化证明引导大语言模型生成引理, 从而从全局证明视角中归纳出具有较强普适性的引理; 而 Lemmas2 与 Reflection lemmas 则聚焦于形式化草图中的特定中间命题, 所生成的引理虽具有较强的针对性, 但通用性相对不足. 综上所述, 引理库中的引理体现了数据结构的通用性和针对性.

● 引理反思. 如图 5(a) 所示, 在数据结构优先级队列上, 通过引理反思获取了 30 个引理, 有效的扩展了引理库. 如图 5(b) 所示, 在数据结构有界栈上, 通过引理反思获取了 19 个引理, 同样有效的扩展了引理库. 尽管在图 5 中, 子集 Reflection lemmas 的引理数量显著高于子集 Lemmas1 和 Lemmas2, 但其对定理证明的有效性却极低. 具体而言, 图 5(a) 中的黄色柱状显示, 子集 Reflection lemmas 的引理在定理证明中并未被引用, 数量为 0; 而图 5(b) 则显示, 该子集在定理证明中的有效性仅为 5.3%, 为所有子集中最低. 这表明, Reflection lemmas 在促进定理证明方面的作用有限. 这一现象可能归因于 LLM 在缺乏丰富反馈信息的情况下, 其反思能力受到限制. 子集 Reflection

lemmas 在引理证明过程中展现出一定的效用.

4.4 案例研究

为更直观地说明引理库驱动证明方法的操作过程, 本节引入了两个具体案例. 第 4.4.1 节将展示该方法高效地证明线性数据结构——优先队列算法定理 (*min_priority_leq_all*). 第 4.4.2 节将该方法扩展到证明非线性数据结构——二叉树重要定理 (*size*1*_height*), 进一步探索本文方法的潜力.

本节案例将会使用到辅助定理证明器 Isabelle, 现将该语言的核心关键词简要说明如下.

theorem: 声明待证明的定理, 明确需要验证的命题.

lemma: 声明辅助性的引理, 用于支撑主定理的证明, 作用是拆分复杂证明为多个小目标.

proof: 启动证明过程, 后续紧跟证明策略或分情况讨论.

case: 分情况证明的分支标记.

have: 证明过程中临时推导的中间结论, 用于逐步构建证明链.

then show: 基于前一步的 have 结论, 推导并证明当前分支的目标 (?thesis 代表当前需证的子目标).

next: 切换到分情况证明的下一个分支.

from: 引用已有的假设、引理或归纳假设.

using: 结合多个假设、引理或自动推理工具完成推导.

by: 调用证明策略 (如 simp 简化器、auto 自动推理、metis 定理证明器、fastforce 强力推理) 完成当前结论的证明.

assumes: 声明定理或引理的前提条件, 明确证明的假设边界.

shows: 声明定理或引理的结论, 即需要证明的目标.

qed: 结束证明过程, 标记当前定理或引理的证明完成.

4.4.1 线性数据结构

本研究首先通过联合请求机制生成与待证定理关联的候选引理陈述, 接着借助迭代证明策略对这些引理进行验证, 并将验证通过的引理收录至引理库中. 在对定理的形式化证明草图进行验证时, 系统从引理库中检索与当前中间命题相匹配的引理, 以此为自动定理证明器提供支撑, 辅助其合成有效的证明策略, 最终实现对中间命题的证明, 并确保定理整体形式化证明的正确性. 如图 4 所示, 定理 *min_priority_leq_all*(“*the* (*priority q* (*min q*)) ⩽ *the* (*priority q v*)”) 表示在优先队列中, 位于队首的元素的优先级具有最小值. 其中, *priority* 函数的作用是根据给定的关键值查询其对应的优先级, 而 *min* 函数则用于获取当前队首元素的关键值. 其具体的证明流程如下所示.

为证目标定理 *min_priority_leq_all*, 首先请求大模型生成定理非形式化证明, 大模型会根据优先级队列的操作函数 (如函数 *prioprity* 和 *min*), 逐步化简并证明定理 *min_priority_leq_all*, 整个证明过程被逐层拆解, 构建出逻辑严谨的推导路径. 随后, 以该非形式化证明为基础, 引导大模型构建出定理的形式化证明草图. 该形式化证明草图以逻辑形式语言为载体, 遵循自然语言证明所呈现的推理脉络, 逐步构建出完整的形式化推导体系.

其次, 大模型构造相关引理辅助验证定理. 根据定理 *min_priority_leq_all* 的非形式化证明及其形式化证明草图, 通过联合请求机制引导大模型生成相关的引理. 在此过程中, 大模型对该定理证明可能涉及的引理进行了分析. 大模型根据优先级队列满足的约束条件: “*distinct* (*map fst xs*)”和“*sorted* (*map snd xs*)”, 其中“*xs*”是优先级队列对应的列表, 函数 *distinct* 约束优先级队列元素关键值不重复, 函数 sorted 约束优先级队列中元素优先级有序. 通过对定理进行细致分析, 当前定理 *min_priority_leq_all* 中, 优先级队列对应的列表“*xs*”等于“*alist_of q*”, 函数 *alist_of* 表示将优先队列 *q* 转换成列表形式. 由此, 大模型提炼出两个核心的新引理: *distinct_fst_alist_of*、和 *sorted_alist_of*. 分别约束当前优先级队列元素值不重复和对应优先级有序排列. 形式化草图中一个关键中间命题 (“*the* (*map_of* (*alist_of q*)(*min q*)) ⩽ *the* (*map_of* (*alist_of q*) *v*)”) 无法直接被 Sledgehammer 验证, 该中间命题的核心含义在于: 具有最小优先级的元素, 其优先级数值严格小于队列中所有其他元素的优先级. 为完成对该命题的证明, 大模型基于优先级队列的固有约束条件, 提炼出 4 项前置约束, 并以此为基础构建了辅助引理 *min_priority_leq_*

all_aux, 充分融合了优先级队列的约束信息. 值得注意的是, 引理 *distinct_fst_alist_of* 和 *sorted_alist_of* 分别源自对定理的不同非形式化证明请求, 而引理 *min_priority_leq_all_aux* 则是基于定理的形式化证明草图所生成的辅助引理.

然后证明上述引理. 大模型将引理分解为中间命题, 逐个验证其正确性. 以引理 *distinct_fst_alist_of*、*min_priority_leq_all_aux* 和 *sorted_alist_of* 为例, 这引理属于优先级队列, 证明时要满足优先级队列的约束要求, 例如"*alist_of* ?$x \in \{xs.distinct\ (map\ fst\ xs) \wedge sorted\ (map\ snd\ xs)\}$". 当所有引理均完成形式化验证并成功入库之后, 系统进入对定理 *min_priority_leq_all* 形式化证明草图的验证阶段. 此时, Sledgehammer 会从引理库中检索与当前中间命题相匹配的引理, 并基于这些引理整合生成相应的证明策略. 该机制使自动定理证明器可以充分调用引理库中存储的引理, 在证明过程中动态构建合适的证明策略. 以某一中间命题的验证为例, 其证明策略可表达为"by(metis a1 distinct_fst_alist_of h2 min_priority_leq_all_aux order_refl priority_queue.min_def sorted_alist_of)". 该策略不仅整合了由大语言模型生成并经严格验证的引理 *distinct_fst_alist_of*、*min_priority_leq_all_aux* 与 *sorted_alist_of*, 同时也调用了系统预定义的定理或引理, 如 *order_refl*、*priority_queue* 与 *min_def*. 通过对这些引理的有机组合, 证明策略得以有效验证草图中的全部中间命题, 从而确保定理 *min_priority_leq_all* 的最终正确性.

4.4.2 扩展性案例

为进一步探索本文方法的适用性, 我们将其推广至非线性数据结构算法定理的证明任务中. 对于需要引理支持的非线性数据结构定理, 设计合适的引理, 通过逐步推导的方式实现对定理正确性的验证. 定理 *size1_height* ("*size1* $t \leqslant 2$^*height*(t :: *'a tree*)") 描述了二叉树中节点总数与树高的关系, 其核心结论为: 对于任意二叉树 *t*, 节点总数不超过 2^{*height t*}−1 (其中 *height t* 表示树的高度). 该定理的详细证明步骤如下.

首先, 如图 6 所示, 大语言模型基于二叉树的递归结构, 分别对 *t* = *leaf* 和 *t* = *Node l a r* 两种情形进行证明, 逐步展开定理 *size1_height* 的完整证明. 随后, 大语言模型将该非形式化证明转换为形式化证明草图, 并尝试通过 Sledgehammer 对草图内的中间命题进行自动化验证. 然而, 在此过程中出现了一个中间命题 (如: "2^*height l*+2^*height r* $\leqslant 2 * 2$^(*max* (*height l*)(*height r*))"). 由于 *height l* 和 *height r* 之间的大小关系无法直接确定, Sledgehammer 无法直接完成证明.

Step 1: Prove for the binary tree t using structural induction…
Step 2: Base case (the leaf node Leaf). .. size1 Leaf = 1. Similarly, …2 ^ height Leaf = 2 ^ 0 = 1.Since both size1 Leaf and 2 ^ height Leaf are equal to 1….
Step 3: Inductive case (the internal node Node l x r) .
Step 4: we can conclude that size1 (Node l x r) = size1 l + size1 r.…we get height (Node l x r) = max (height l) (height r) + 1.
Step 5: Then, according to the rules of exponentiation, from the above formula, through simplification, we can deduce that 2 ^ height (Node l x r) = 2 * 2 ^ (max (height l) (height r)).
Step 6: Application of the inductive hypothesis: we know that size1 l ≤ 2 ^ height l. we know that size1 r ≤ 2 ^ height r…we can obtain that size1 l + size1 r ≤ 2 ^ height l + 2 ^ height r.
Step 7: Further derivation of the inequality:For the inequality 2 ^ height l + 2 ^ height r ≤ 2 * 2 ^ max (height l) (height r)...
Step 8: Draw a conclusion:we can conclude that size1 (Node l x r) ≤ 2 ^ height (Node l x r), and the proof of the inductive case is completed.

```
theorm size1_height: "size1 t ≤ 2 ^ height (t::'a tree)"
proof (induction t)
  case Leaf
  have "size1 Leaf = 1" by simp
  have "2 ^ height Leaf  = 1" by simp
  then show ?case by simp
next
  case (Node l x r)
  have "size1 (Node l x r) = size1 l + size1 r" by simp
  have "height (Node l x r) = max (height l) (height r) +1" by simp
  have "2 ^ height (Node l x r) = 2 * 2 ^ (max (height l) (height r))"
by simp
  from Node.IH(1) have "size1 l ≤ 2 ^ height l" .
  from Node.IH(2) have "size1 r ≤ 2 ^ height r" .
  have "size1 l + size1 r ≤ 2 ^ height l + 2 ^ height r"
    using Node.IH(1,2) by auto
  also have "2^ height l + 2 ^ height r ≤ 2 * 2 ^ max (height l)
(height r)"
    by (metis power_Suc power_sum)
  then show ?case
    using calculation by fastforce
qed
```

图 6 二叉树定理 *size1_height* 证明

为攻克这一证明难题, 借助大语言模型专门为该中间命题生成辅助引理 *power_sum* (如: "2^a + 2^$b \leqslant$ 2^(*Suc* (*max a b*))"). 引理 *power_sum* 与前述证明受阻的中间命题在功能层面具有显著的契合性, 二者具有紧密的对

应关系, 是攻克该证明难题的关键桥梁. 该引理本身亦被进一步拆分和归纳为多个中间命题, 使得其证明过程转化为对多个子命题的依次证明, 从而降低了整体证明的复杂性.

最后, 借助引理 *power_sum*, 定理 *size*1*_height* 的自动验证得以最终完成. 正是凭借该引理, 此前难以直接证明的中间命题获得了必要的理论依据, 从而在引理的支撑下被成功推进.

5 局限性

本文聚焦线性数据结构, 我们将从两方面来阐述其原因.

一方面, 在形式化定理证明方面由于缺少训练集, 尤其在数据结构定理证明领域, 无法进行大规模实验. 但我们团队在线性数据结构进行深入研究, 积累了一定数据量的训练集, 因此采取线性数据结构作为研究对象.

另一方面, 本文方法在初始设计与验证过程中, 是紧密围绕线性数据结构的特性进行的. 这主要体现在: 1) 核心引理库的构建. 本文方法引理库重点覆盖了线性结构 (如栈、有界栈、先进先出队列、优先级队列和子列表) 的典型操作 (如迭代、插入、删除) 所涉及的归纳原理和状态不变量. 对于非线性结构中更复杂的递归模式 (如多路递归、嵌套递归) 和复杂的状态不变量 (如平衡因子), 当前引理库的覆盖度有限. 2) 生成策略的针对性. 本文方法集成的中间引理生成策略在处理线性、顺序性的数据访问模式时最为有效. 当面对复杂的非线性结构 (如平衡二叉树的旋转操作、图的遍历) 时, 证明搜索空间会呈指数级增长, 现有策略可能无法高效地生成关键的中间引理, 从而需要人工干预或策略调整.

因此, 本文方法存在局限性, 尤其是在处理复杂的非线性数据结构算法 (如维护红黑树或 AVL 树的平衡性) 时, 需要对引理库和生成策略进行针对性的增强. 未来的工作将致力于将此框架扩展至更复杂的非线性领域.

6 结 论

本文提出了引理库驱动证明方法 (LDPM), 旨在通过构建一个不断扩展的引理库, 增强现有自动定理证明器在处理定理的形式化证明草图中中间命题的能力. 通过融合 LLM 与形式化验证技术, 借助定理的非形式化证明和形式化证明草图启发 LLM 生成多个候选引理, 并且对候选引理进行迭代式证明, 从而不断提高引理库的可靠性. 此外, 利用 LLM 反思未被证明的引理, 借助 LLM 生成更“正确”的引理陈述, 进一步扩展引理库的内容. 与以往方法集中于构造易证明的形式化草图不同, LDPM 特别关注提升自动定理证明器在中间命题处理上的能力. 实验测试表明, LDPM 确实可以提高数据结构算法定理的证明成功率. 本文的消融研究和详细分析显示了 LDPM 中每项策略的有效性.

尽管 LDPM 在实验验证中表现出显著优势, 但其在以下几个方面仍有进一步提升的空间: 首先, 在引理筛选方面, Sledgehammer 当前所采用的筛选策略较为固定, 其机械式的引理选择过程可能使部分关键引理被遗漏, 导致定理难以成功证明. 针对这一问题, 未来可引入启发式筛选机制, 引导 Sledgehammer 进行更为精准的引理筛选, 以提升证明策略生成的效率与准确性. 其次, 在模型生成质量方面, 当前由 ChatGPT 输出的候选引理陈述及形式化草图在领域适配性上仍有提升空间. 针对这一现状, 对 LLM (如 ChatGPT) 进行面向特定领域的微调, 有望显著提升其在该任务中的表现能力. 此外, 在引理反思环节中, 当前输入至 LLM 的反馈信息相对有限, 一定程度上制约了其反思机制的有效发挥, 若能在反思任务中引入更丰富、更具结构化的反馈信号, 将有助于激发模型进行更深层次的自我优化, 从而提升其在引理构造与迭代过程中的综合性能.

References

[1] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33–61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[2] Ding HR, Wang ZG, Fu M, Chen HB. Formal methods and system software: Practice and suggestions. Science and Technology Foresight, 2023, 2(1): 33–45 (in Chinese with English abstract). [doi: 10.3981/j.issn.2097-0781.2023.01.003]

[3] Wang SL, Zhan BH, Sheng HH, Wu H, Yi SC, Wang LT, Jin XY, Xue B, Li JH, Xiang SQ, Xiang Z, Mao BF. Survey on requirements

classification and formalization of trustworthy systems. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2367–2410 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]

[4] Barras B, Boutin S, Cornes C, Courant J, Filliâtre Jc, Giménez E, Herbelin H, Huet G, Muñoz C, Murthy C, Parent C, Paulin-Mohring C, Saïbi A, Werner B. The Coq proof assistant reference manual: Version 6.1. 1997. https://inria.hal.science/inria-00069968/document

[5] Nipkow T, Wenzel M, Paulson LC. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Berlin, Heidelberg: Springer, 2002. [doi: 10.1007/3-540-45949-9]

[6] de Moura L, Kong S, Avigad J, van Doorn F, von Raumer J. The Lean theorem prover (system description). In: Proc. of the 25th Int'l Conf. on Automated Deduction. Berlin: Springer, 2015. 378–388. [doi: 10.1007/978-3-319-21401-6_26]

[7] Megill N, Wheeler DA. Metamath: A Computer Language for Mathematical Proofs. Morrisville, North Carolina: Lulu Press, 2019.

[8] Nagashima Y, Kumar R. A proof strategy language and proof script generation for Isabelle/HOL. In: Proc. of the 26th Int'l Conf. on Automated Deduction. Gothenburg: Springer, 2017. 528–545. [doi: 10.1007/978-3-319-63046-5_32]

[9] Gauthier T, Kaliszyk C, Urban J, Kumar R, Norrish M. Learning to prove with tactics. Journal of Automated Reasoning, 2021, 65(2): 257–286. [doi: 10.1007/s10817-020-09580-x]

[10] Jiang AQ, Li WD, Tworkowski S, Czechowski K, Odrzygóźdź T, Miłoś P, Wu YH, Jamnik M. Thor: Wielding hammers to integrate language models and automated theorem provers. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 608.

[11] Wu YH, Jiang AQ, Li WD, Rabe MN, Staats C, Jamnik M, Szegedy C. Autoformalization with large language models. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 2344.

[12] Paulson LC, Blanchette JC. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Proc. of the 8th Int'l Workshop on the Implementation of Logics. Yogyakarta: IWIL, 2011. 1–11.

[13] Kühlwein D, van Laarhoven T, Tsivtsivadze E, Urban J, Heskes T. Overview and evaluation of premise selection techniques for large theory mathematics. In: Proc. of the 6th Int'l Joint Conf. on Automated Reasoning. Manchester: Springer, 2012. 378–392. [doi: 10.1007/978-3-642-31365-3_30]

[14] Mikuła M, Tworkowski S, Antoniak S, Piotrowski B, Jiang AQ, Zhou JP, Szegedy C, Kuciński Ł, Miłoś P, Wu YH. Magnushammer: A Transformer-based approach to premise selection. In: Proc. of the 12th Int'l Conf. on Learning Representations. Vienna: OpenReview.net, 2024.

[15] Jiang AQ, Welleck S, Zhou JP, Lacroix T, Liu JC, Li WD, Jamnik M, Lample G, Wu YH. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: OpenReview.net, 2023.

[16] First E, Rabe MN, Ringer T, Brun Y. Baldur: Whole-proof generation and repair with large language models. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 1229–1241. [doi: 10.1145/3611643.3616243]

[17] Wang HM, Xin HJ, Zheng CY, Liu ZY, Cao QX, Huang YY, Xiong J, Shi H, Xie E, Yin J, Li ZG, Liang XD. LEGO-Prover: Neural theorem proving with growing libraries. In: Proc. of the 12th Int'l Conf. on Learning Representations. Vienna: OpenReview.net, 2024.

[18] Han JM, Rute J, Wu YH, Ayers E, Polu S. Proof artifact co-training for theorem proving with language models. In: Proc. of the 10th Int'l Conf. on Learning Representations. OpenReview.net, 2022.

[19] Hardin DS, Hardin SS. Efficient, formally verifiable data structures using ACL2 single-threaded objects for high-assurance systems. In: Proc. of the 8th Int'l Workshop on the ACL2 Theorem Prover and its Applications. Boston: ACM, 2009. 100–105. [doi: 10.1145/1637837.1637853]

[20] Zhao XL, Li WD, Kong LP. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. arXiv:2305.16366, 2023.

[21] Claessen K, Johansson M, Rosén D, Smallbone N. Automating inductive proofs using theory exploration. In: Proc. of the 24th Int'l Conf. on Automated Deduction. Lake Placid: Springer, 2013. 392–406. [doi: 10.1007/978-3-642-38574-2_27]

[22] Czajka Ł, Kaliszyk C. Hammer for Coq: Automation for dependent type theory. Journal of Automated Reasoning, 2018, 61(1): 423–453. [doi: 10.1007/s10817-018-9458-4]

[23] Alemi AA, Chollet F, Een N, Irving G, Szegedy C, Urban J. DeepMath-deep sequence models for premise selection. In: Proc. of the 30th Int'l Conf. on Neural Information Processing Systems. Barcelona: Curran Associates Inc., 2016. 2243–2251.

[24] Paliwal A, Loos S, Rabe M, Bansal K, Szegedy C. Graph representations for higher-order logic and theorem proving. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI Press, 2020. 2967–2974. [doi: 10.1609/aaai.v34i03.5689]

[25] Mündler N, Nipkow T. A verified implementation of B^+-trees in Isabelle/HOL. In: Proc. of the 19th Colloquium on Theoretical Aspects of Computing. Tbilisi: Springer, 2022. 324–341. [doi: 10.1007/978-3-031-17715-6_21]

[26] Toth B, Nipkow T. Real-time double-ended queue verified (proof pearl). In: Proc. of the 14th Int'l Conf. on Interactive Theorem Proving. Białystok: ITP, 2023. 29:1–29:18. [doi: 10.4230/LIPIcs.ITP.2023.29]

[27] Miao XH, Chang R, Zhao JH, Zhao YW, Cao S, Wei T, Jiang L, Ren K. CVTEE: A compatible verified TEE architecture with enhanced security. IEEE Trans. on Dependable and Secure Computing, 2023, 20(1): 377–391. [doi: 10.1109/TDSC.2021.3133576]

[28] OpenAI Team. ChatGPT: Optimizing language models for dialogue. 2022. https://blog.cloudhq.net/openais-chatgpt-optimizing-language-models-for-dialogue/

[29] OpenAI. GPT-4 technical report. arXiv:2303.08774, 2023.

[30] Dong QX, Li L, Dai DM, Zheng C, Ma JY, Li R, Xia HM, Xu JJ, Wu ZY, Chang BB, Sun X, Li L, Sui ZF. A survey on in-context learning. In: Proc. of the 2024 Conf. on Empirical Methods in Natural Language Processing. Miami: ACL, 2024. 1107–1128. [doi: 10.18653/v1/2024.emnlp-main.64]

[31] Zhang WQ, Shen YL, Wu KJ, Peng QY, Wang J, Zhuang YT, Lu WM. Self-contrast: Better reflection through inconsistent solving perspectives. In: Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Bangkok: ACL, 2024. 3602–3622. [doi: 10.18653/v1/2024.acl-long.197]

[32] Koepke P, Lorenzen A, Shminke B. CICM'22 system entries. In: Buzzard K, Kutsia T, eds. Intelligent Computer Mathematics. Cham: Springer, 2022. 344–348. [doi: 10.1007/978-3-031-16681-5_24]

[33] Barrett C, Conway CL, Deters M, Hadarean L, Jovanović D, King T, Reynolds A, Tinelli C. CVC4. In: Gopalakrishnan G, Qadeer S, eds. Computer Aided Verification. Berlin, Heidelberg: Springer, 2021. 171–177. [doi: 10.1007/978-3-642-22110-1_14]

[34] Bouton T, de Oliveira DCB, Déharbe D, Fontaine P. veriT: An open, trustable and efficient SMT-solver. In: Schmidt RA, ed. Automated Deduction—CADE-22. Berlin, Heidelberg: Springer, 2009. 151–156. [doi: 10.1007/978-3-642-02959-2_12]

[35] de Moura L, Bjørner N. Z3: An efficient SMT solver. In: Ramakrishnan CR, Rehof J, eds. Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer, 2008. 337–340. [doi: 10.1007/978-3-540-78800-3_24]

[36] Weidenbach C, Dimova D, Fietzke A, Kumar R, Suda M, Wischnewski P. SPASS version 3.5. In: Schmidt RA, ed. Automated Deduction—CADE-22. Berlin, Heidelberg: Springer, 2009. 140–145. [doi: 10.1007/978-3-642-02959-2_10]

[37] Riazanov A, Voronkov A. The design and implementation of Vampire. AI Communications, 2002, 15(2–3): 91–110.

[38] Cruanes S. Logtk: A Logic ToolKit for automated reasoning and its implementation. In: Proc. of the 4th Workshop on Practical Aspects of Automated Reasoning. Vienna: PAAR@IJCAR 2014, 2014. 39–49. [doi: 10.29007/4z1m]

附中文参考文献

[1] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33–61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[2] 丁浩然, 王肇国, 付明, 陈海波. 形式化方法与系统软件: 实践与发展建议. 前瞻科技, 2023, 2(1): 33–45. [doi: 10.3981/j.issn.2097-0781.2023.01.003]

[3] 王淑灵, 詹博华, 盛欢欢, 吴昊, 易士程, 王令泰, 金翔宇, 薛白, 李静辉, 向霜晴, 向展, 毛碧飞. 可信系统性质的分类和形式化研究综述. 软件学报, 2022, 33(7): 2367–2410. http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]

作者简介

王昌晶, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为高可信软件, 智能化软件.

万亮亮, 硕士, 主要研究领域为定理证明, 智能化软件.

刘艳娇, 硕士, 主要研究领域为定理证明, 形式化方法.

龙海建, 硕士, 主要研究领域为定理证明, 形式化方法.

左正康, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为形式化方法, 智能化软件.