相继式演算与霍尔逻辑的证明树构建器

Joomy Korkut普林斯顿大学，美国新泽西州普林斯顿
joomy@cs.princeton.edu

我们开发了一款基于网络的教学型证明助手——**证明树构建器**，它能够让用户从相继式演算和简单命令式语言的霍尔逻辑中的初始目标出发，向上应用规则。我们为该工具配备了基本的证明自动化功能和 Z3 集成。普林斯顿大学自动推理课程的学生使用了我们的工具，并认为它直观易用。**证明树构建器** 可通过 https://proof-tree-builder.github.io 在线访问。

1 引言

在几乎所有形式下，生成证明树都是一项繁琐的任务。学生在纸上书写证明树时，若要回退，只能使用橡皮擦，这会拖慢进度。学生在 LaTeX 中输入证明树时，必须满足编译器的要求，此时正确使用推理规则反而成了次要问题。理想情况下，程序应确保使用正确的规则，并以规范的 LaTeX 格式呈现证明树，让学生专注于证明树的实质内容。

为解决这一问题，我们设计了一个基于浏览器的图形化证明助手——**证明树构建器**。用户可以指定证明目标、选择下一步要应用的证明规则，并手动提供应用该规则所需的信息（例如替换项、前置条件和后置条件），从而构造证明。如果无法应用某条规则，该工具会显示警告信息。因此，用户可以通过不断应用证明规则，完成给定证明目标的证明。经验丰富的用户可以使用*自动化模式*，该模式会隐藏无关规则，并配备一个基础的自动命题证明器；初学者则可以使用*学习模式*，在这种模式下，用户需要更审慎地选择规则。

证明树构建器 支持一阶逻辑的相继式演算证明 [27]，以及针对包含顺序语句、条件语句、循环语句和赋值语句的简单命令式语言的霍尔逻辑证明 [14]。普林斯顿大学研究生级自动推理课程的学生使用了我们的工具，认为它直观易用，而且该工具并未降低他们的理解能力。

2 操作演示

证明树构建器 界面顶部有一个带按钮的菜单，左侧栏列出了当前活动的证明，每项都有相应按钮，可将证明树输出为 LaTeX、删除证明树，或将其保存为文件。其余白色区域是工作区，证明会显示在这里。每棵证明树都可以在工作区内拖放。用户还可以放大或缩小视图，以聚焦于较大证明树的特定部分；用户体验类似于使用矢量编辑软件。

假设用户想在学习模式下证明 $\vdash p \Rightarrow q \Rightarrow (p \wedge q)$。他们会先点击左上角的“添加 LK 目标”按钮，然后输入目标，如图 1 所示。

J. Marcos、W. Neuper 和 P. Quaresma（编）：教育软件定理证明组件 2022（ThEdu'22），EPTCS 375，2023 年，第 54–62 页，
doi:10.4204/EPTCS.375.5

© J.Korkut 本作品采用知识共享署名许可协议授权。

解析器同时支持 Unicode 字符及其 ASCII 表示形式，例如 ∧ 和 /\、&&。用户输入时，可以在文本框下方看到解析结果。此功能有助于用户再次确认运算符优先级和绑定变量作用域。如果出现解析错误，文本框下方会以红色显示错误信息。

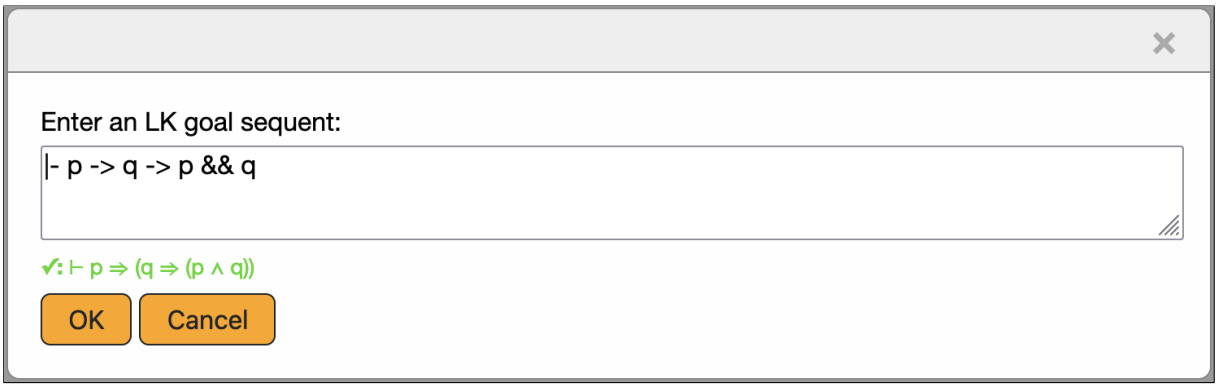

图 1：在**证明树构建器**中输入一个目标。

用户添加目标后，会在工作区中看到一棵未完成的证明树，其标志是目标上方的橙色线条，以及橙色线条右侧的按钮。用户点击该按钮后，会出现一个包含所有证明规则选项的弹出菜单，如图 2 所示。

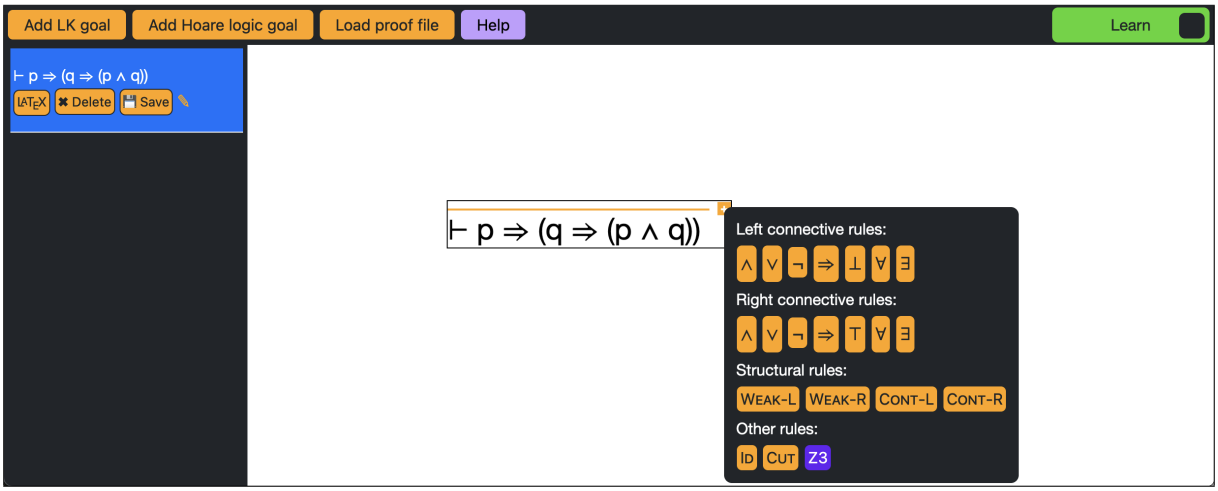

图 2：点击按钮后显示的证明规则选项弹出菜单。

对于图中的目标，用户需要应用 $\Rightarrow_R$ 规则，因此可以点击右连接词规则下方的 $\Rightarrow$ 按钮，进入图 3 所示的界面。接着，再应用一次 $\Rightarrow_R$ 规则和一次 $\wedge_R$ 规则即可继续证明。完成后一步后，该规则的两个前提都会变成未完成的证明树，如图 4 所示。用户应分别完成这些未完成的证明树。随后，在两个未完成的子树中应用 ID 规则，即可完成该证明。

应用证明规则后，用户会在规则标签旁看到三个按钮：、、和。这些按钮可用于撤销证明规则应用。第一个按钮用于撤销该证明规则，

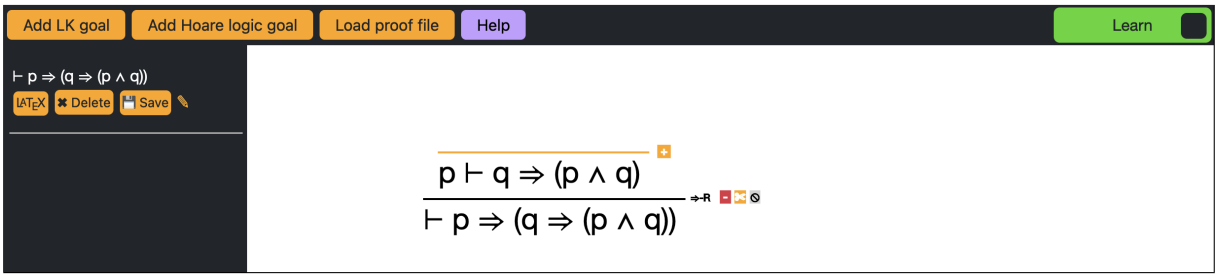

图 3：应用一条规则后的未完成证明树。

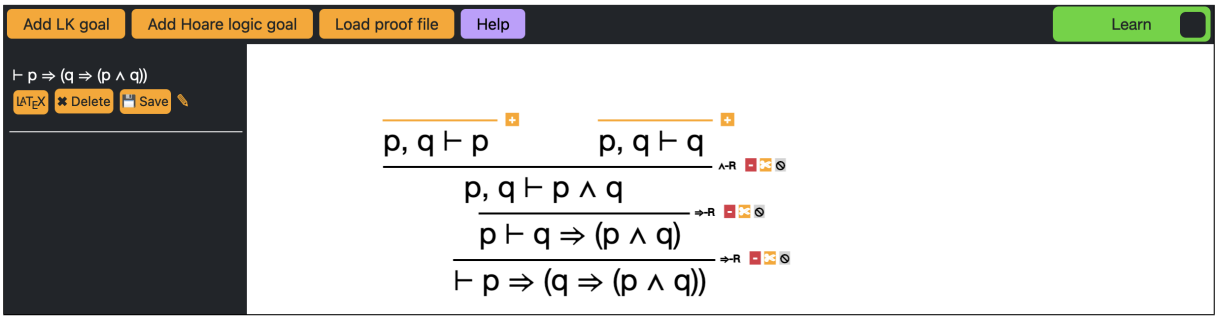

图 4：两个未完成的前提证明树。

以及其上方的所有规则。第二个按钮可将证明树一分为二：它会将上方的子树分离出来，形成一棵独立的证明树，从而使原证明树变得不完整。可以将分离出的证明树拖放到另一棵证明树的不完整末端，以重新连接。第三个按钮可隐藏该证明规则的前提，在处理大型证明时，可用于折叠或展开证明的部分内容。

对于包含量词的一阶逻辑相继式，点击量词对应的规则后，系统会根据规则提示用户输入一个新鲜变量或一个项，如图 5 所示。

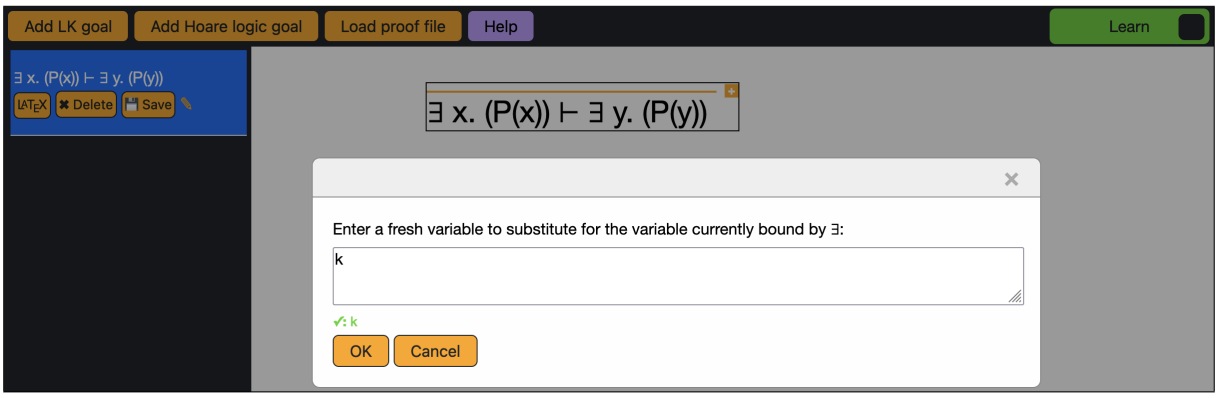

图 5 ：应用相继式中的 $\exists_L$ 规则时，提示用户输入一个新鲜变量 .

我们上面给出的演示展示了证明树构建器的**学习模式**。用户可以点击右上角的开关切换到自动化模式。在自动化

模式下，点击应用新规则的按钮时，只会显示相关按钮。对于相继式演算证明，相关按钮包括相继式两侧最外层联结词对应的规则、根据相继式每一侧公式数量而定的结构规则、切规则以及自动化按钮，如图6所示。对于霍尔逻辑证明，相关按钮包括对应于当前命令的规则和逻辑规则。

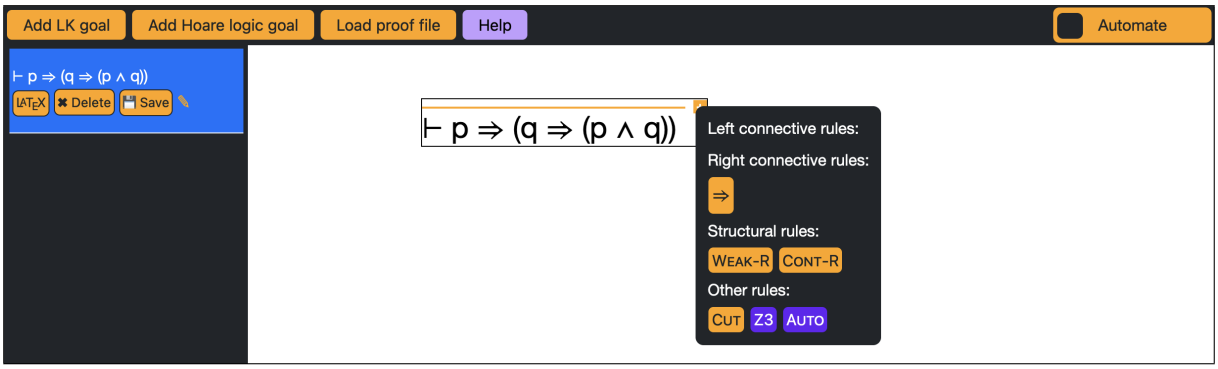

图6：自动化模式会隐藏不可能应用的相继式演算证明规则按钮。

在自动化模式中，有一个学习模式下不会出现的新按钮：AUTO按钮。它会执行一种基本的证明搜索。对于图6中的目标，AUTO按钮实际上可以生成整棵证明树。第3.1小节将讨论其能力范围。

另有一个名为 Z3 的按钮，可提供另一种自动化功能。它会对目标运行 Z3 定理证明器 [8]，如果 Z3 证明了其有效性，我们的工具就会解除该目标，并在目标上方画一条绿色线。如果 Z3 能找到反模型，我们的工具就会像图 8 所示那样将其展示给用户。第3.2小节将更详细地讨论这一功能。

图7展示了一个简单的霍尔逻辑证明，其中应用了后果规则。后果规则的第一个前提是霍尔逻辑证明树中的一个相继式演算目标。然而，我们的相继式演算规则无法解释数值上的函数和关系，因此用户必须使用 Z3 伪公理来检查数值等式目标。

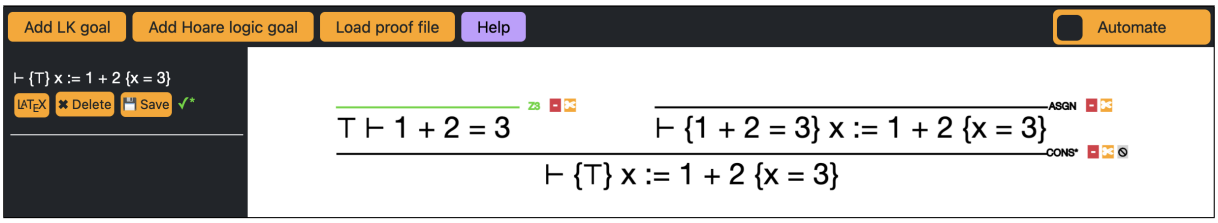

图7：在一个简单的霍尔逻辑示例中应用 Z3 伪公理。

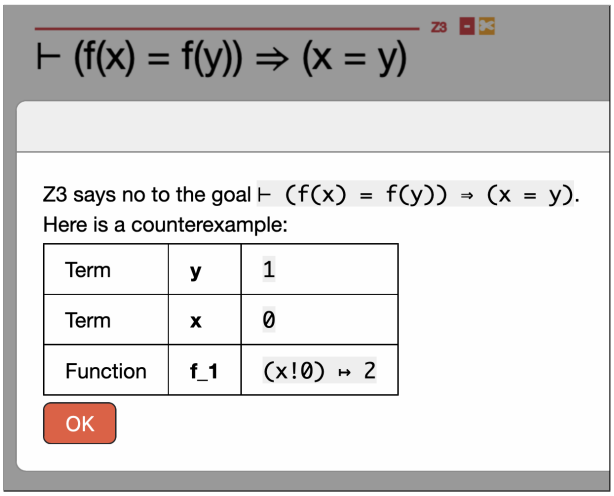

图 8：Z3 为给定目标生成反模型。

3 设计与实现

3.1 自动化

自动化按钮并不十分智能；它只是按特定顺序应用可用规则。当某条规则可以应用于相继式中的多个公式时，它就会卡住。例如，像$p \Rightarrow q$，$p \Rightarrow r$，$p \vdash q$这样的命题相继式本应很容易证明，但我们的证明器需要知道用户想要将 $\Rightarrow_L$ 规则应用于哪个公式。这个选择可能会导致不同的证明树，因此我们的自动化系统不会替用户做出选择，而是直接停止。

这一限制可以视为一种教学选择：如果学生能够自动完成更多证明步骤，他们可能会更频繁地选择这种方式，最终学到的内容反而更少。不过，实现更多自动化功能可以为用户提供提示，因此也有其价值。我们可以在未来选择实现一些关于这类技术的现有研究成果 [10, 29, 30] 。

3.2 Z3 集成

我们自动推理课程的家庭作业 [17] 包括为命题和一阶相继式编写证明树，以及为处理整数算术的程序编写霍尔三元组。对于相继式，证明树很容易构造；但涉及整数算术的霍尔三元组则需要等式和算术运算的证明规则。我们的定理证明器确实支持项中的整数和算术运算符，但没有用于算术函数求值或等式关系的证明规则。我们不想让这些目标未经检查。因此，我们为 Z3 工具添加了一条新的伪公理 。应用这条规则时，它会用 SMT-LIB 语言对当前相继式取否定编码， [3],运行 Z3 定理证明器 [8] 并寻找反模型。如果不存在反模型，该目标就是有效的。

Z3 伪公理并不是证明规则，它只是在检查这些相继式在某个理论下是否有效之后，让我们能够对这些证明“略过不提”的一种方式。我们会明确告知用户，这条规则并非形式化的证明规则；我们是有意模糊可证性与有效性之间的区别。

我们之所以做出这一教学选择，是因为加入所有关于等式和算术运算的规则（如果扩展到其他数据类型，还要加入其他运算的规则），会使证明助手的

界面变得杂乱，也会模糊这些家庭作业的重点。与其陷入等式和算术运算的细节，我们更愿意把大部分时间用于讲授霍尔逻辑的基础知识。

当学生不确定自己尝试证明的目标是否有效时，Z3 伪公理也很有用。应用 Z3 伪公理后，他们可以判断目标是否有效；如果无效，还能得到一个反模型。如果目标有效，他们可以随时撤销该规则，并像往常一样继续证明相继式。

3.3 实现细节

我们的工具使用 JavaScript 编写，由于仅包含静态文件，因此完全在客户端运行。它使用 HTML5 画布作为工作区，并借助 Fabric.js [32]库处理图形。解析器由 PEG.js [21], 这一类 Yacc 风格的解析器生成器根据我们的语法生成。我们的工具通过 Emscripten [7]加载一个编译为 WebAssembly 的 Z3 版本。随后，Z3 由 Web Workers 运行，从而实现前文所述的 Z3 集成。

对工具正确性的形式化保证不在本项目的讨论范围内。在我们的实现中，每条规则都对应一个类，这些类的构造函数会检查规则应用是否正确。阅读源代码即可查看这些检查。

保存证明文件时会生成用于创建证明规则类实例的 JavaScript 代码。将证明文件载入**证明树构建器**后，系统会执行这些代码，从而重新运行正确性检查。因此，篡改证明文件以得出错误结论，与在工具中直接得出错误结论一样不可能。

4 相关工作

Logitext [31] 是基于网页的图形化证明助手的先驱之一。它是一款用于相继式演算的证明助手，用户通过点击相继式中的公式来应用规则。对于新用户而言，这种界面可能令人困惑，因为点击会以难以理解的方式改变证明状态。用户学会如何使用系统后，操作也可能变得过于简单，因为他们只需不断点击任意公式，直到完成证明。相比之下，我们的系统要求用户在每个规则步骤中明确选择一条证明规则。

序列演算训练器 [11, 12]是卡塞尔大学开发的一款工具，其功能与**证明树构建器**相似。开发我们的工具时，我们并不知道他们的工具，但后来惊喜地发现两者有不少相似之处。不过，它们之间也存在一些显著差异。他们的工具使用 Java 编写，需要安装；而我们的工具使用 JavaScript 编写，可直接在浏览器中运行，无需安装。他们的工具只支持一阶逻辑和命题逻辑的相继式演算，而我们的工具还支持霍尔逻辑。两种工具都使用 Z3 检查有效性，但如图 8 所示，我们的工具还会向用户展示无效目标的反模型。在这些方面，我们认为自己的工具改进了他们的工作。序列演算训练器要求用户在相继式中选择一个公式或子公式，然后按下规则按钮。相比之下，我们的工具采用了另一种应用规则的用户界面。我们的工具在未完成的证明树旁设置了按钮，其设计灵感来自 ALF 证明编辑器 [20],Epigram [22] 和 Agda [24]的定理证明“空洞驱动”开发风。用户按下按钮并点击证明后，无需选择规则要应用于相继式中的哪个公式；只有在存在歧义时，系统才会提示用户选择所指的公式。如果该工具的用户在相继式中的某个公式内选择子公式

并尝试应用规则，就会看到一条警告消息，说明规则只能应用于顶层公式。他们的工具在微观层面上具有教学价值，因为它要求用户更深入地理解规则的运作方式；而我们的工具在宏观层面上更具教学价值，因为它能让学生足够快速地推进证明，从而证明有趣的定理。

相继式演算计算器 [15] 是另一个与我们的工具类似的工具。该工具的用户界面要求用户先选择要应用的规则，再在规则的空白处填入与其应用方式相匹配的公式和项，然后将规则拖到树中要应用该规则的部分。该工具也支持一阶逻辑，而我们的工具还支持用于简单命令式语言的霍尔逻辑。我们的工具在撰写这篇学士论文的一年前就已开始开发；开发期间，我们并不知道他们的工具。他们基于填空的做法是我们也希望加入自己工具的一项功能，尤其适用于霍尔逻辑中应用后果规则时选择中间公式等步骤，或一阶逻辑中应用量词规则时选择新鲜变量或项。

也有用于构建其他演算的证明树的教学工具。Panda [13] 是一个自然演绎证明助手，支持通过正向或反向推理构建证明树，但不提供自动化功能；它的图形界面与序列演算训练器类似。“Click and coL- LecT” [5] 是一个用于线性逻辑证明的工具，其界面与 Logitext 几乎相同。Sequoia [26] 是一种支持多种相继式演算风格逻辑系统的工具。使用 Sequoia 的学生可以在树中选择一个尚未证明的相继式，再从侧边栏中选择要应用的规则。

此外，还有一系列用于不同演算的证明工具，支持树形以外的各种证明风格。NaDeA [28] 是另一款自然演绎证明助手；它允许用户以结构化列表的形式书写证明 [16]。它的句法、语义和证明系统都在 Isabelle 证明助手中完成了形式化 [23], ，因此学生可以了解“幕后”的运作，深入探究定义的每个细节，并在 Isabelle 中验证最终证明。AXolotl [6] 是一款教学工具，可处理希尔伯特风格、相继式风格和自然演绎证明。该工具允许针对特定问题限制可用规则，从而帮助学生专注于与问题相关的规则。我们的工具在自动化模式下会隐藏联结词与相继式中公式不匹配的规则按钮，但无法根据规则与整个证明的相关性来隐藏规则。Carnap [18] 是一个框架，允许用户定义自己的逻辑系统并在其中书写证明，但不支持证明树。Holbert [25] 是一款教学型证明助手，允许用户定义自己的逻辑系统，并使用这些系统构建证明树。对于课程内容不限于相继式演算和霍尔逻辑的情况，这类工具会更加实用。不过，这类工具的通用性可能导致学生收到的错误消息不够有帮助。

还有一些项目专注于通过拖放方式构建证明 [4, 9, 19]。与我们的工具不同，这些系统侧重于游戏化设计和吸引人的图形界面。

5 结论

证明树构建器 已用于普林斯顿大学 [17]的自动推理课程。学生可以选择使用**证明树构建器**，也可以手写证明树。所有人都使用了**证明树构建器**，而且几乎所有人在这些题目上都得了满分。没有错误的证明，但有些学生错误地将有效公式判定为无效，并试图构造反模型。学生没有在课程讨论页面上询问如何使用该工具，教授认为这表明我们的工具易于使用。起初，教授担心该工具最终会降低学生的理解能力，因为学生可能会“狂按按钮”，而不是理解演算；但期中考试相关题目的成绩与上一年相近，

这表明学生的理解能力没有受到影响。教授表示，他计划明年再次使用**证明树构建器** 。我们尚未进一步开展关于该工具有效性的用户研究。

《**证明树构建器**》源于我们在 LaTeX 中或手工在纸上编写证明树时遇到的种种挫折。为了解决这些问题，我们提供了直观的界面，帮助用户构建相继式演算和霍尔逻辑的证明树，并学习这些证明系统。我们还为系统添加了自动化功能和 Z3 支持，用于确认目标的有效性，并为无效目标生成反例。

6 致谢

我要感谢 Anastasiya Kravchuk-Kirilyuk 和 John Li 对项目开发所作的贡献，感谢 Anna Blech 和 Andrew W. Appel 对论文提出的意见，还要感谢 Zachary Kincaid 在其课程中使用我们的工具。

参考文献

[1] Ian Arawjo、Cheng-Yao Wang、Andrew C Myers、Erik Andersen 和 François Guimbretière（2017）：通过游戏化语义教学编程。载于：*2017* 年 *ACM* 人机交互大会论文集，第 4911–4923 页，doi:10.1145/3025453.3025711。[2] Michaël Armand、Germain Faure、Benjamin Grégoire、Chantal Keller、Laurent Théry 和 Benjamin Werner（2011）：通过证明见证将 *SAT/SMT* 求解器模块化集成到 *Coq* 中。载于：认证程序与证明国际会议，Springer，第 135–150 页，doi:10.1007/978-3-642-25379-9_12。[3] Clark Barrett、Pascal Fontaine 和 Cesare Tinelli（2016）：模理论可满足性库（*SMT-LIB*）。网址：http://www.smt-lib.org。[4] Joachim Breitner（2016）：使用 *Incredible Proof Machine* 进行可视化定理证明。载于：交互式定理证明国际会议，Springer，第 123–139 页，doi:10.1007/978-3-319-43144-4_8。[5] Etienne Callies 和 Olivier Laurent（2021）：*Click and coLLecT*：交互式线性逻辑证明器。载于：第五届线性逻辑与应用趋势国际研讨会（*TLLA 2021*）。网址：https: //hal-lirmm.ccsd.cnrs.fr/lirmm-03271501。[6] David M Cerna、Rafael PD Kiesel 和 Alexandra Dzhiganskaya（2020）：用于自主学习形式推理的移动应用程序。doi:10.48550/arXiv.2002.12553。[7] Emscripten贡献者（2022）：构建为 *WebAssembly*。网址：https://web.archive.org/web/20221030175022/https://emscripten.org/docs/compiling/WebAssembly.html。[8] Leonardo De Moura 和 Nikolaj Bjørner（2008）：*Z3*：高效的 *SMT* 求解器。载于：系统构造与分析工具和算法国际会议，doi:10.1007/978-3-540-78800-3_24。[9] Pablo Donato、Pierre-Yves Strub 和 Benjamin Werner（2022）：拖放式证明策略。载于：第 *11* 届 *ACM SIGPLAN*认证程序与证明国际会议论文集，第 197–209 页，doi:10.1145/3497775.3503692。[10] Arno Ehle（2017）：带等号的一阶逻辑相继式演算中的证明搜索。硕士论文，卡塞尔大学。[11] Arno Ehle、Norbert Hundeshagen 和 Martin Lange（2015）：相继式演算训练器——帮助学生正确构造证明。*arXiv* 预印本 *arXiv:1507.03666*，doi:10.48550/arXiv.1507.03666。[12] Arno Ehle、Norbert Hundeshagen 和 Martin Lange（2018）：具备自动推理功能的相继式演算训练器——帮助学生寻找证明。*arXiv* 预印本 *arXiv:1803.01467*，doi:10.48550/arXiv.1803.01467。

[13] Olivier Gasquet、François Schwarzentruber 和 Martin Strecker（2011）：Panda：面向所有人的自然演绎证明助手。面向本科生的 Gentzen 风格证明助手。载于：国际逻辑教学工具大会，Springer，第 85–92 页，doi:10.1007/978-3-642-21350-2_11。[14] Charles Antony Richard Hoare（1969）：计算机编程的公理化基础。《ACM 通讯》12(10)，第 576–580 页，doi:10.1145/363235.363259。[15]Matteo Kamm 和 Mike Marti（2019）：相继式演算计算器。学士论文，拉珀斯维尔应用科技大学。网址：https://eprints.ost.ch/id/eprint/798/1/FS2019-BA-EP-Marti-Kamm-TheSequentCalculusCalculator.pdf。[16] Richard W. Kaye（2015）：作为结构化列表和证明树的证明。网址：http://web.archive.org/web/20210903173847/https://web.mat.bham.ac.uk/R.W.Kaye/logic/prooftrees.html。[17] Zachary Kincaid（2021）：COS 516/ECE 516：软件自动推理，2021 年秋季。网址：http://web.archive.org/web/20211217155727/https://www.cs.princeton.edu/courses/archive/fall21/cos516/。[18] Graham Leach-Krouse（2018）：Carnap：一个用于在浏览器中进行形式推理的开放框架。arXiv 预印本 arXiv:1803.03092，doi:10.48550/arXiv.1803.03092。[19] Sorin Lerner、Stephen R Foster 和 William G Griswold（2015）：多态块：受形式主义启发、用于结构化连接器的用户界面。载于：第 33 届 ACM 人机交互年度大会论文集，第 3063–3072 页，doi:10.1145/2702123.2702302。[20] Lena Magnusson 和 Bengt Nordström（1993）：ALF 证明编辑器及其证明引擎。载于：证明与程序类型国际研讨会，Springer，第 213–237 页，doi:10.1007/3-540-58085-9_78。[21] David Majda 和 Futago za Ryuu（2022）：PEG.js——JavaScript 解析器生成器。网址：https: //web.archive.org/web/20221007224930/https://pegjs.org/。[22] Conor McBride 和 James McKinna（2004）：来自左侧的视角。《函数式编程杂志》14(1)，第 69–111 页，doi:10.1017/S0956796803004829。[23] Tobias Nipkow、Markus Wenzel 和 Lawrence C Paulson（2002）：Isabelle/HOL：高阶逻辑证明助手。Springer，doi:10.1007/3-540-45949-9。[24] Ulf Norell（2007）：迈向一种基于依赖类型论的实用编程语言。博士论文，查尔姆斯理工大学。网址：https://www.cse.chalmers.se/~ulfn/papers/ thesis.pdf。[25] Liam O'Connor 和 Rayhana Amjad（2022）：Holbert：在浏览器中阅读、写作、证明和学习。arXiv 预印本 arXiv:2210.11411，doi:10.48550/arXiv.2210.11411。[26] Giselle Reis、Zan Naeem 和 Mohammed Hashim（2020）：Sequoia：逻辑学家的试验场。载于：自动推理国际联合会议，Springer，第 480–488 页，doi:10.1007/978-3-030-51054-1_-32。[27] M. E. Szabo，编（1969）：Gerhard Gentzen 论文集。逻辑与数学基础研究，North-Holland 出版公司，阿姆斯特丹。[28] Jørgen Villadsen、Asta Halkjær From 和 Anders Schlichtkrull（2019）：自然演绎助手（NaDeA）。载于：教育软件定理证明组件，doi:10.48550/arXiv.1904.00618。[29] A. A. Voronkov（1990）：一阶逻辑的证明搜索方法。载于：COLOG-88，Springer，第 327–338 页，doi:10.1007/3-540-52335-9_63。[30] Hao Wang（1960）：迈向机械化数学。《IBM 研究与开发杂志》4(1)，第 2–22 页，doi:10.1147/rd.41.0002。[31] Ed ward Z. Yang（2012）：Logitext。网址：https://github.com/ezyang/logitext。[32] Juriy Zaytsev、Stefan Kienzle 和 Andrea Bogazzi（2022）：Fabric.js JavaScript 画布库。网址：https://web.archive.org/web/20221112201025/http://fabricjs.com/。