A Proof Tree Builder for Sequent Calculus and Hoare Logic

Joomy Korkut

Princeton University, USA
joomy@cs.princeton.edu

Abstract

We have developed a web-based graphical proof assistant, the **Proof Tree Builder**, that lets you apply rules upwards from the initial goal in sequent calculus and Hoare logic for a simple imperative language. We equipped our tool with a basic proof automation feature and Z3 integration. Students in the automated reasoning course at Princeton University used our tool and found it intuitive. The **Proof Tree Builder** is available online at https://proof-tree-builder.github.io.

1 Introduction

Producing proof trees is a tedious task in almost any form. When a student writes a proof tree on paper, they have no way to backtrack without using an eraser and slowing down the process. When a student types up a proof tree in LaTeX, they must satisfy the compiler, at which point using the right inference rules becomes a secondary concern. Ideally, a program should make sure that the right rules are used and render the proof trees in proper LaTeX, while the student focuses on the substance of the proof trees.

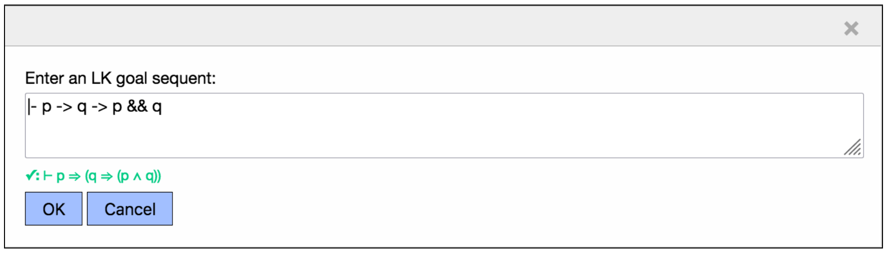
As a solution to this problem, we have designed a browser-based tool, the **Proof Tree Builder**, which allows the user to construct proofs by specifying a proof goal, choosing the proof rule that should be applied next, and manually providing the necessary information for the rule to be applied (e.g. substitution terms, pre- and post- conditions). If a rule cannot be applied, the tool shows a warning message. The user is thus able to achieve a complete proof for the given proof goal by continuously applying proof rules. Experienced users can use the *automation mode*, which hides irrelevant rules and has a basic automated propositional prover, while beginners can use the *learning mode*, in which the user has to be more deliberate in the rules they pick.

The **Proof Tree Builder** supports sequent calculus proofs [10] for first-order logic and Hoare logic proofs [11] for a simple imperative language with sequencing, conditional, loop, and assignment statements. Students in the graduate-level automated reasoning course at Princeton University used our tool and found it intuitive, and our tool did not decrease their comprehension.

2 A walk-through

The **Proof Tree Builder** interface has a top menu with buttons, and a left bar that has the list of active proofs, each of which has buttons to output the tree as LaTeX, delete the tree, or save the tree as a file. The remaining white area is a workspace where proofs appear. Each proof tree can be dragged and dropped within the workspace. The user can also zoom in and out to focus on certain parts of bigger proof trees; the user experience is similar to that of a vector editing software.

Suppose the user wants to prove $\vdash p \Rightarrow q \Rightarrow (p \wedge q)$ in the learning mode. They would start by clicking the Add LK Goal button on the top left and then their goal, as seen in figure 1. The parser supports both Unicode characters such as $\wedge$ and ASCII representations of them such as `/\` and `&&`. As the user types, they can see how it is parsed below the text box. This feature helps the user double check the operator precedence and binder scopes. If there is a parse error, the user sees it below the text box in red.



Enter an LK goal sequent:

```
|- p -> q -> p && q
```

✔: ⊢ p ⇒ (q ⇒ (p ∧ q))

OK Cancel

Figure 1: Entering a goal in the **Proof Tree Builder**.

When the user adds a goal, they will see an incomplete proof tree in the workspace, as denoted by the orange line over the goal and the ⊞ button on the right side of the orange line. When the user clicks on the ⊞ button, a popup menu with all the proof rule options appear, as shown in figure 2.

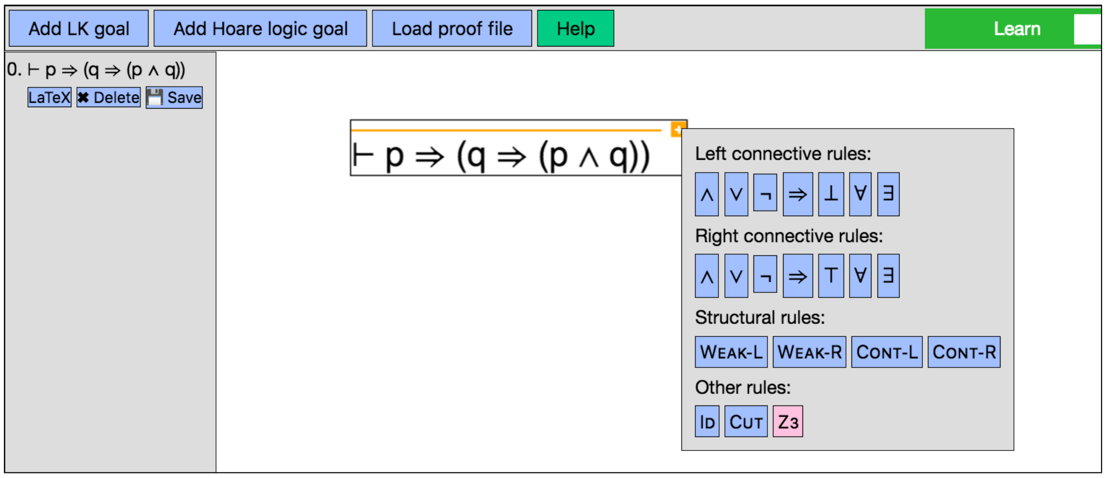


Figure 2: Proof rule options popup menu after clicking the ⊞ button.

For the goal in the figure, the user needs to apply the $\Rightarrow_R$ rule, so they can click on the $\Rightarrow$ button under Right connective rules and reach the screen in figure 3. The proof then proceeds with one more application of the $\Rightarrow_R$ rule and an application of the $\wedge_R$ rule. After the latter step, both premises of the rule will turn into incomplete proof trees, as seen in figure 4. These incomplete proof trees should be completed separately the user. This proof can then be finished by applying the ID rule in both incomplete subtrees.

After a proof rule has been applied, the user sees two buttons, ■ and ⤨, next to the rule label. These buttons can be used to undo proof rule applications. The first button, ■, unapplies the proof rule, and all the rules above. The second button, ⤨, divides a proof tree into two by detaching the subtree above into a separate proof tree, which makes the original proof tree

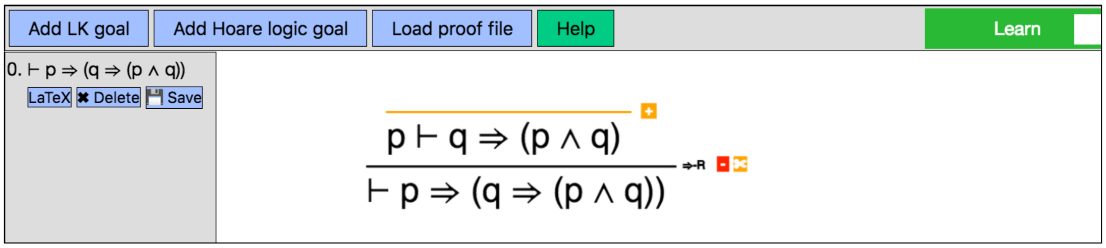


Figure 3: Incomplete proof tree after one rule application.

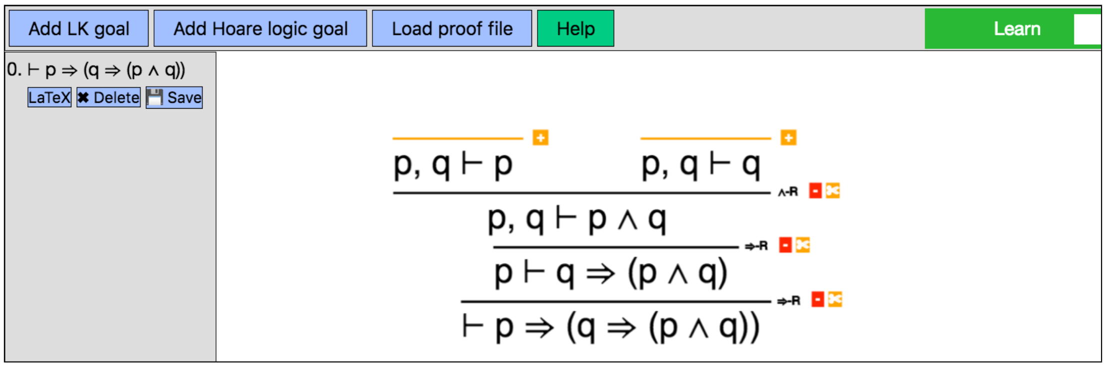


Figure 4: Two incomplete premise proof trees.

incomplete. A detached proof tree can be dragged and dropped onto the incomplete end of another proof tree to reattach.

The demonstration we have given above shows the *learning mode* of the **Proof Tree Builder**. The user can switch to the *automation mode* by clicking the switch at the top right corner. In the automation mode, clicking on the + button to apply a new rule will only show the relevant buttons. For a sequent calculus proof, the relevant buttons are the rules for the outermost conjunctives for both sides of a sequent, the structural rules, the cut rules, and the automation buttons, as seen in figure 5. For a Hoare logic proof, the relevant buttons are the rule for the immediate command and the logical rules.

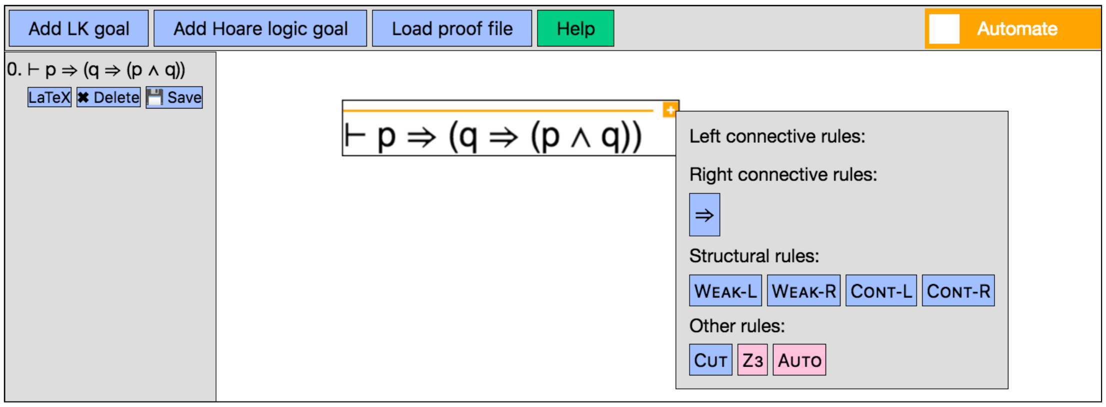
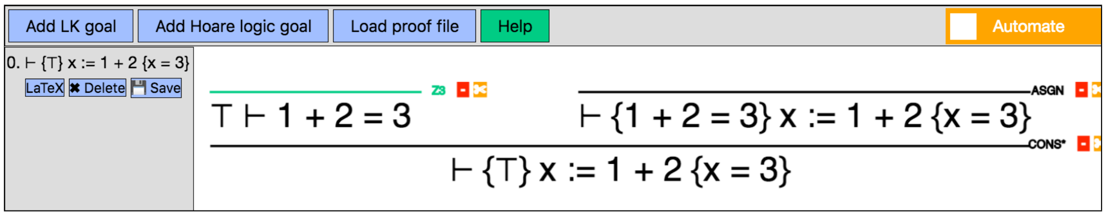


Figure 5: The automation mode hides the buttons for the sequent calculus proof rules that are impossible to apply.

In the automation mode, there is a new button, the Auto button, that does not appear in the learning mode. This button performs a primitive proof search. For the goal in figure 5, the Auto button can actually generate the entire proof tree. Subsection 3.1 discusses the extent of its capabilities.

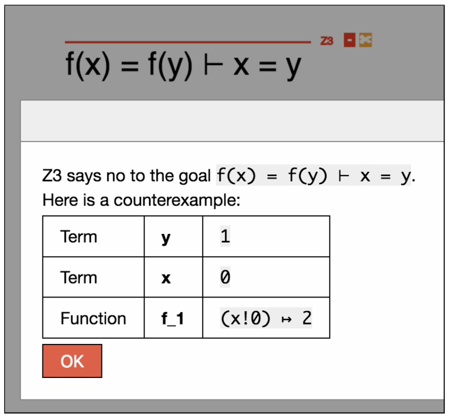
There is another button called Z3, which provides a different kind of automation. It runs the Z3 theorem prover [4] on the goal, and if Z3 proves validity, our tool discharges the goal and draws a green line over the goal. If Z3 can find a countermodel, our tool shows it to the user, as seen in figure 7. Subsection 3.2 discusses this feature in more detail.

Figure 6 shows a simple Hoare logic proof with a consequence rule application. The first premise of the consequence rule is a sequent calculus goal inside a Hoare logic proof tree. However, our sequent calculus rules cannot interpret functions and relations on numbers, so the user has to use the Z3 pseudo-axiom to check the numeric equality goal.



$$\dfrac{\dfrac{}{T \vdash 1+2=3}\,\text{Z3} \qquad \dfrac{}{\vdash \{1+2=3\}\ x := 1+2\ \{x=3\}}\,\text{ASGN}}{\vdash \{T\}\ x := 1+2\ \{x=3\}}\,\text{CONS*}$$

Figure 6: Applying the Z3 pseudo-rule in a simple Hoare logic example.



f(x) = f(y) ⊢ x = y

Z3 says no to the goal `f(x) = f(y) ⊢ x = y`.
Here is a counterexample:

Term	**y**	`1`
Term	**x**	`0`
Function	**f_1**	`(x!0) ↦ 2`

Figure 7: Z3 generating a countermodel for the given goal.

3 Design choices

3.1 Automation

The automation button is not very clever; it merely applies available rules in a specific order. It gets stuck when a rule can be applied to multiple formulas in a sequent. For example, a propositional sequent such as $p \Rightarrow q, p \Rightarrow r, p \vdash q$ should be easily provable, but our prover needs to know which of the formulas the user would like to apply the $\Rightarrow_L$ rule to. This choice can lead to different proof trees, so our automation system does not make that choice, it just stops instead.

This limitation can be justified as a pedagogical choice: if the students can automate more of the proof, they may take that option more often and end up learning less. There is value, however, in implementing more automation features as a way to provide hints to the user. We may choose to implement some of the existing work on such techniques [21, 20, 6] in the future.

3.2 Z3 integration

The homework assignments in our automated reasoning course [13] include writing proof trees for propositional and first-order sequents, and Hoare triples for programs that handle integer arithmetic. For sequents, proof trees are easily constructable, but Hoare triples involving integer arithmetic requires proof rules for equality and arithmetic operations. Our theorem prover does have integers and arithmetic operators in terms, but it does not have proof rules for arithmetic function evaluation or the equality relation. We do not want to leave these goals unchecked. Therefore we added a new *pseudo-axiom* to our tool, called the Z3 rule. When applied, this rule encodes negation of the the sequent at hand in the SMT-LIB language [1], runs the Z3 theorem prover [4] and looks for countermodels. If there are no countermodels, the goal is valid.

The Z3 rule is not a proof rule, it is merely a way for us to "hand-wave" these proofs after checking if these sequents are valid modulo a theory. We make it clear to the user that this rule is not a formal proof rule; we are flouting the distinction between provability and validity on purpose.

We made this pedagogical choice because adding all the rules for equality and arithmetic operations (and other operations if we extend other data types) would have cluttered the proof assistant's interface and obscured the focus of these homework assignments. We would rather spend most of our time teaching the essentials of Hoare logic than get bogged down in details of equality and arithmetic operations.

The Z3 rule is also useful when the student is not sure if the goal they are trying to prove is valid in the first place. By applying the Z3 rule, they can find out if the goal is valid, and get a countermodel if it is not.

4 Related Work

Logitext [22] is one of the pioneers in web-based graphical proof assistants. It is a proof assistant for sequent calculus, where the user applies rules by clicking on a formula in the sequent. This interface can be confusing for new users, since their clicks change the proof state in cryptic ways. It can also be too easy once the user learns how to use the system, since they can keep clicking on random formulas until the proof is finished. Our system, in comparison, requires the user to explicitly pick a proof rule in individual rule steps.

The Sequent Calculus Trainer [7, 8] is a tool developed at the University of Kassel, and it has similar features to the **Proof Tree Builder**. We were not aware of their tool when we were developing ours, but we were pleasantly surprised to find similarities. However, there are some significant differences. Their tool is written in Java and requires installation, while our tool is written in JavaScript and runs in the browser, with no need for installation. Their tool only handles sequent calculus for first-order and propositional logic, while our tool handles Hoare logic as well. Both tools use Z3 to check for validity but our tool also presents countermodels for invalid goals to the user, as seen in figure 7. In these respects, we believe our tool improves on their work.

The Sequent Calculus Trainer requires the user to pick a formula or subformula in the sequent and then to press a rule button. In contrast, our tool has adapted a different user interface to apply rules. Our tool has the ⊞ button next to incomplete proof trees, inspired by the hole-driven development style of theorem proving of the ALF proof editor [15], Epigram [16] and Agda [18]. Once the user presses ⊞ and clicks on a proof, they do not have to pick which formula in the sequent the rule applies to, unless it is ambiguous, in which case they receive a prompt to pick which formula they mean. If a user of their tool picks a subformula inside a formula in the sequent and tries to apply a rule, they see warning message that explains that rules can only be applied to top-level formulas. Their tool is pedogically useful in the small, because it requires the user to have a better understanding of how the rules work, but ours is more pedagogically useful in the large, because it allows students to move fast enough to prove interesting theorems.

There are pedagogical tools for other calculi as well. Panda [9] is a proof assistant for natural deduction that allows building proof trees by backward or forward reasoning, with no automation, but with a similar graphical interface to the Sequent Calculus Trainer. NaDeA [19] is another natural deduction proof assistant; it lets the users write structured list style [12] proofs. Its syntax, semantics, and proof system are all formalized in the Isabelle proof assistant [17], so students can see "behind the scenes", dig into the definitions all the way down, and verify the resulting proof in Isabelle. "Click and coLLecT" [3] is a tool for linear logic proofs, with almost the same interface as Logitext.

There are also projects that focus on a drag-and-drop model of building proofs [14, 2, 5]. Unlike our tool, these systems focus on gamification and appealing graphics.

5 Conclusion

The **Proof Tree Builder** has been used in the automated reasoning course at Princeton University [13]. The students were given the options to either use the **Proof Tree Builder** or write their proof trees by hand. Everyone used the **Proof Tree Builder** and almost everyone got full points on the problems. There were no incorrect proofs, but some students incorrectly identified valid formulas as invalid and tried to produce a countermodel. The students did not ask any questions on the course discussion page about how to use the tool, which the professor thought was a data point in favor of our tool being easy to use. At first, the professor wondered whether the tool would ultimately decrease comprehension, since students can "button-mash" instead of understanding the calculus, but the results for relevant questions on the midterm exam were similar to the previous year's scores, indicating that student comprehension was intact. The professor reports that he plans to use the **Proof Tree Builder** again next year.

The **Proof Tree Builder** was born out of our frustrations writing proof trees in LaTeX or on paper by hand. We resolved these frustrations by providing an intuitive interface to build proof trees for sequent calculus and Hoare logic and to learn these proof systems. We enhanced our system with automation features and we added Z3 support to confirm the validity of goals and to generate counterexamples for invalid goals.

6 Acknowledgments

I want to thank Anastasiya Kravchuk-Kirilyuk and John Li for their contributions to the development of the project, Anna Blech and Andrew W. Appel for their comments on the paper, and Zachary Kincaid for using our tool in his course.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[2] Joachim Breitner. Visual theorem proving with the incredible proof machine. In *International Conference on Interactive Theorem Proving*, pages 123–139. Springer, 2016.

[3] Etienne Callies and Olivier Laurent. Click and coLLecT An Interactive Linear Logic Prover. In *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, 2021.

[4] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[5] Pablo Donato, Pierre-Yves Strub, and Benjamin Werner. A drag-and-drop proof tactic. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 197–209, 2022.

[6] Arno Ehle. Proof search in the sequent calculus for first-order logic with equality. Master's thesis, Universität Kassel, 2017.

[7] Arno Ehle, Norbert Hundeshagen, and Martin Lange. The sequent calculus trainer-helping students to correctly construct proofs. *arXiv preprint arXiv:1507.03666*, 2015.

[8] Arno Ehle, Norbert Hundeshagen, and Martin Lange. The sequent calculus trainer with automated reasoning-helping students to find proofs. *arXiv preprint arXiv:1803.01467*, 2018.

[9] Olivier Gasquet, François Schwarzentruber, and Martin Strecker. Panda: A proof assistant in natural deduction for all. a gentzen style proof assistant for undergraduate students. In *International Congress on Tools for Teaching Logic*, pages 85–92. Springer, 2011.

[10] Gerhard Gentzen. Untersuchungen über das logische schließen. I. *Mathematische Zeitschrift*, 39(1):176–210, 1935.

[11] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[12] Richard W. Kaye. Proofs as structured lists and proof trees. `http://web.archive.org/web/20210903173847/https://web.mat.bham.ac.uk/R.W.Kaye/logic/prooftrees.html`, 2015.

[13] Zachary Kincaid. COS 516/ECE 516: Automated Reasoning about Software, Fall 2021. `http://web.archive.org/web/20211217155727/https://www.cs.princeton.edu/courses/archive/fall21/cos516/`, 2021.

[14] Sorin Lerner, Stephen R Foster, and William G Griswold. Polymorphic blocks: Formalism-inspired ui for structured connectors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3063–3072, 2015.

[15] Lena Magnusson and Bengt Nordström. The ALF proof editor and its proof engine. In *International Workshop on Types for Proofs and Programs*, pages 213–237. Springer, 1993.

[16] Conor McBride and James McKinna. The view from the left. *Journal of Functional Programming*, 14(1):69–111, 2004.

[17] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[18] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

[19] Jørgen Villadsen, Asta Halkjær From, and Anders Schlichtkrull. Natural Deduction Assistant (NaDeA). In *Theorem Proving Components for Educational Software*, 2019.

[20] AA Voronkov. A proof-search method for the first order logic. In *COLOG-88*, pages 327–338. Springer, 1990.

[21] Hao Wang. Toward mechanical mathematics. *IBM Journal of Research and Development*, 4(1):2–22, 1960.

[22] Edward Z. Yang. Logitext. `https://github.com/ezyang/logitext`, 2012.