A Proof Tree Builder
for Sequent Calculus and Hoare Logic

Joomy Korkut

Princeton University
Princeton, New Jersey, USA

`joomy@cs.princeton.edu`

We have developed a web-based pedagogical 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 graphical proof assistant, 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 [27] for first-order logic and Hoare logic proofs [14] 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 entering their goal, as seen in Figure 1. The

parser supports both Unicode characters such as ∧ 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.
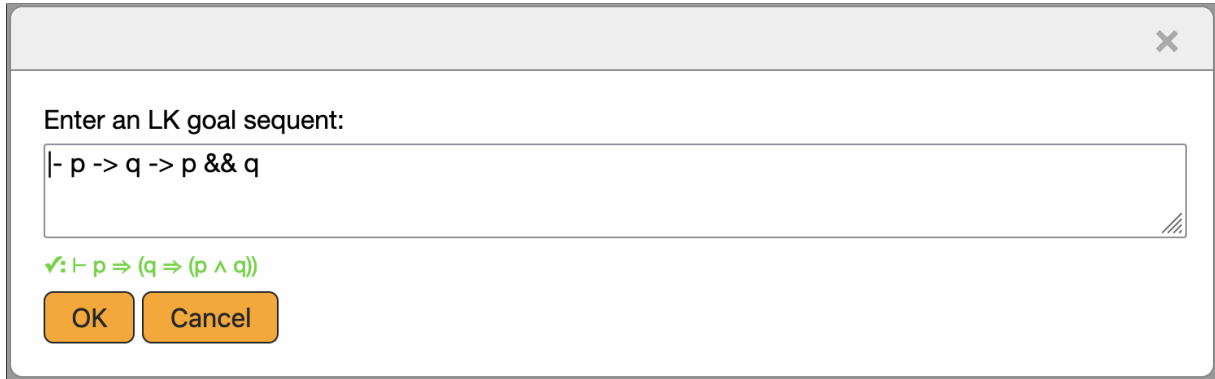

Figure 1: Entering a goal in the **Proof Tree Builder**.

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.
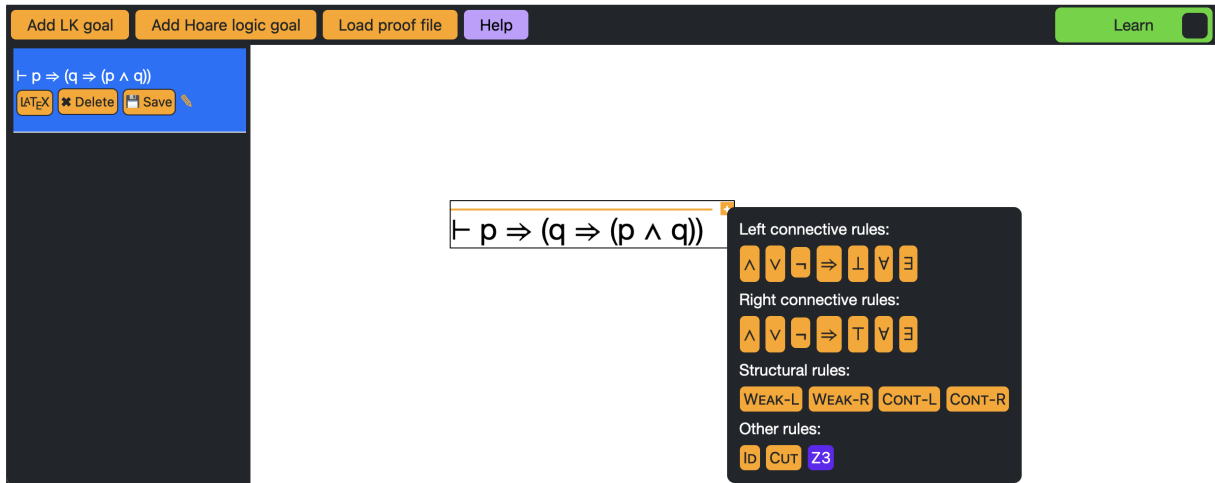

Figure 2: Proof rule options popup menu after clicking the ➕ button.

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 by 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 three buttons, ▪, ✖, and ⊘, next to the rule label. These buttons can be used to undo proof rule applications. The first button, ▪, unapplies the proof rule,
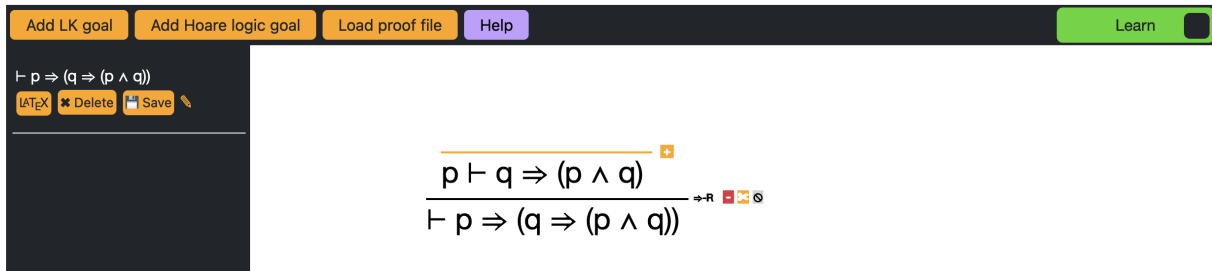
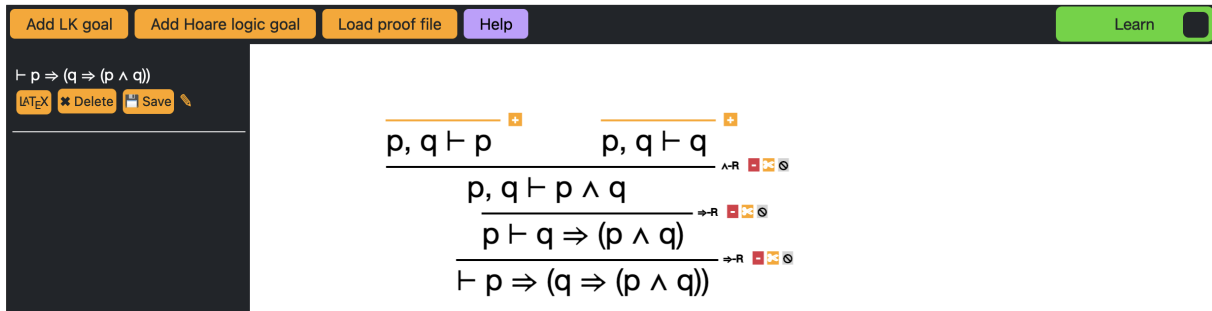Figure 3: Incomplete proof tree after one rule application.

Figure 4: Two incomplete premise proof trees.

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 incomplete. A detached proof tree can be dragged and dropped onto the incomplete end of another proof tree to reattach. The third button, ⊘, hides the premises of the proof rule and can be used to fold/unfold parts of a proof while working on large proofs.

For first-order logic sequents that contain quantifiers, clicking on the rules for the quantifiers will show the user a prompt asking for a fresh variable or a term, depending on the rule, as seen in Figure 5.
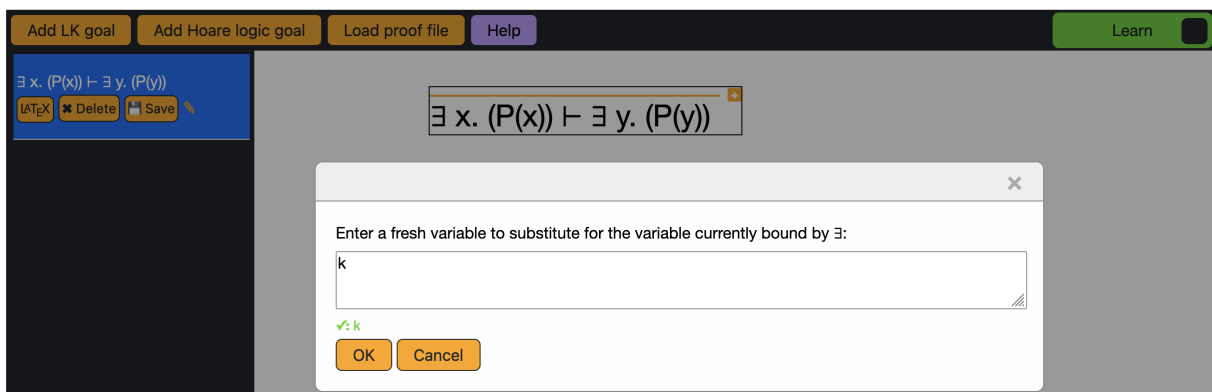
Figure 5: A prompt asking the user for a fresh variable when applying the $\exists_L$ rule in a sequent.

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 connectives for both sides of a sequent, the structural rules depending on the number of formulas in each side of the sequent, the cut rule, and the automation buttons, as seen in Figure 6. For a Hoare logic proof, the relevant buttons are the rule for the immediate command and the logical rules.
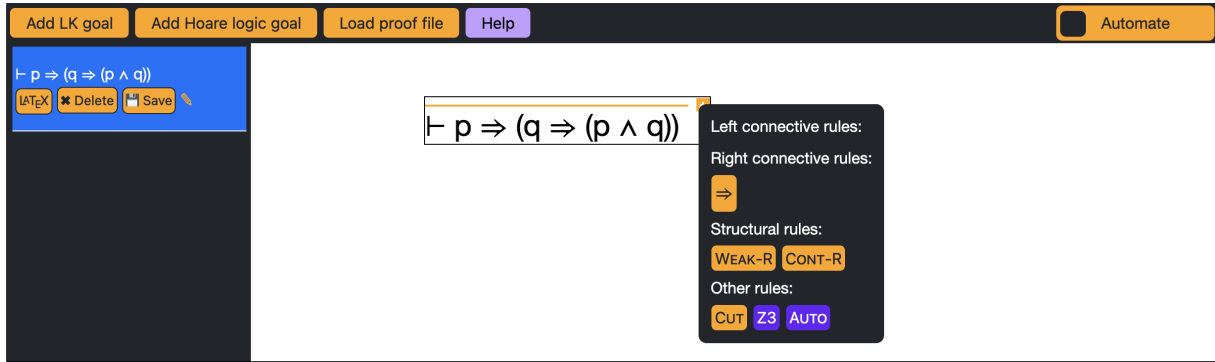

Figure 6: The automation mode hides the buttons for the sequent calculus proof rules that are impossible to apply.

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 6, 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 [8] 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 8. subsection 3.2 discusses this feature in more detail.

Figure 7 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.
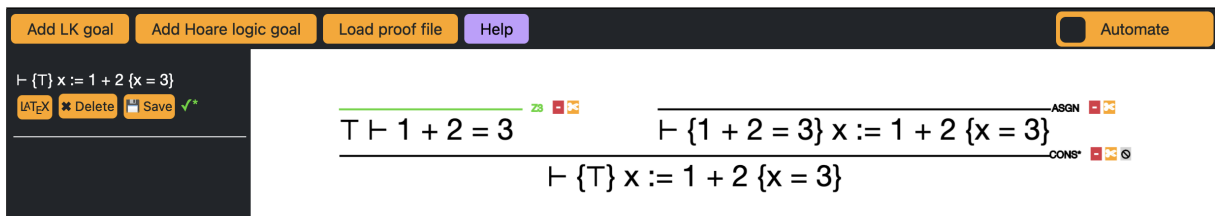

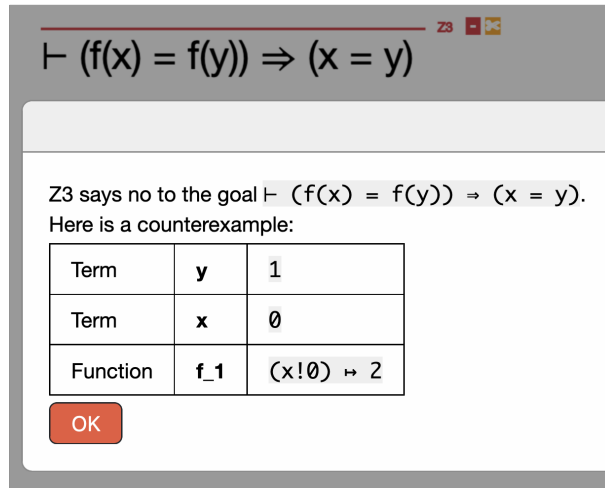Figure 7: Applying the Z3 pseudo-axiom in a simple Hoare logic example.

Figure 8: Z3 generating a countermodel for the given goal.

3   On design and implementation

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 [10, 29, 30] in the future.

3.2   Z3 integration

The homework assignments in our automated reasoning course [17] 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 require 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 for Z3. When applied, this rule encodes negation of the the sequent at hand in the SMT-LIB language [3], runs the Z3 theorem prover [8] and looks for countermodels. If there are no countermodels, the goal is valid.

   The Z3 pseudo-axiom 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 with 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 pseudo-axiom 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 pseudo-axiom, they can find out if the goal is valid, and get a countermodel if it is not. If it is valid, they can always unapply the rule and continue to prove the sequent as usual.

3.3   Implementation details

Our tool is written in JavaScript; it consists of static files and therefore runs solely on the client side. It uses an HTML5 canvas for the workspace, powered by the Fabric.js [32] library to handle graphics. The parser is generated from our grammar by PEG.js [21], a Yacc-style parser generator. Our tool loads a version of Z3 compiled to WebAssembly via Emscripten [7]. Z3 is then run by web workers, which makes the Z3 integration described earlier possible.

Formal guarantees about the correctness of our tool are not in the scope of this project. In our implementation, each rule is a class and the constructor of these classes check whether the rule application is correct. These checks can be inspected by reading the source code.

Saving a proof file creates JavaScript code that creates instances of the proof rule classes. When a proof file is loaded into **Proof Tree Builder**, it evaluates this code and therefore reruns the correctness checks, which makes tampering with the proof files to get incorrect conclusions no more possible than getting an incorrect conclusion from within the tool.

4   Related Work

Logitext [31] 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 [11, 12] 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 8. 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 [20], Epigram [22] and Agda [24]. 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 a warning message that explains that rules can only be applied to top-level formulas. Their tool is pedagogically 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.

The Sequent Calculus Calculator [15] is another tool similar to ours. Their tool's user interface involves picking a rule to apply, filling the holes in the rule with formulas and terms that match how the user will apply the rule, and then dragging the rule on the part of the tree onto the part of the tree the rule should be applied. Their tool also handles first-order logic, but ours also handles Hoare logic for a simple imperative language. Our tool was first developed a year before this bachelor's thesis was written; we were not aware of their tool while developing ours. Their approach based on filling holes is a feature we would like to add to our tool as well, especially for steps like picking the middle formulas in a consequence rule application in Hoare logic, or for picking fresh variables or terms in quantifier rule applications in first-order logic.

There are pedagogical tools for building proof trees for other calculi as well. Panda [13] 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. "Click and coL-LecT" [5] is a tool for linear logic proofs, with almost the same interface as Logitext. Sequoia [26] is a tool that allows various logic systems in sequent calculus style. Students using Sequoia pick an unproved sequent in the tree and pick a rule from the side bar to apply.

There is also a collection of tools for proofs in various calculi, in various proof styles other than trees. NaDeA [28] is another natural deduction proof assistant; it lets the users write structured list style [16] proofs. Its syntax, semantics, and proof system are all formalized in the Isabelle proof assistant [23], so students can see "behind the scenes", dig into the definitions all the way down, and verify the resulting proof in Isabelle. AXolotl [6] is an educational tool that can handle Hilbert style, sequent style and natural deduction proofs. Their tool allows restricting the allowed rules for a given problem, hence helping the student focus on the rules that are relevant to the problem. Our tool's automation mode hides the buttons for the rules whose connectives do not match the formulas in the sequent, but it cannot hide rules based on their relevance to the entire proof. Carnap [18] is a framework that allows the users to define their own logic systems and write proofs in them, but not proof trees. Holbert [25] is a pedagogical proof assistant that allows users to define their own logic systems and also build proof trees using them. For a course that is not limited to sequent calculus and Hoare logic, such tools would be more useful. However, the generality of the such tools may cause the students to get less helpful error messages.

There are also projects that focus on a drag-and-drop model of building proofs [4, 9, 19]. 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 [17]. 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. We have not done a further user study on the effectiveness of our tool.

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

References

[1] Ian Arawjo, Cheng-Yao Wang, Andrew C Myers, Erik Andersen & François Guimbretière (2017): *Teaching programming with gamified semantics*. In: *Proceedings of the 2017 CHI conference on human factors in computing systems*, pp. 4911–4923, doi:10.1145/3025453.3025711.

[2] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry & Benjamin Werner (2011): *A modular integration of SAT/SMT solvers to Coq through proof witnesses*. In: *International Conference on Certified Programs and Proofs*, Springer, pp. 135–150, doi:10.1007/978-3-642-25379-9_12.

[3] Clark Barrett, Pascal Fontaine & Cesare Tinelli (2016): *The Satisfiability Modulo Theories Library (SMT-LIB)*. Available at `http://www.smt-lib.org`.

[4] Joachim Breitner (2016): *Visual theorem proving with the Incredible Proof Machine*. In: *International Conference on Interactive Theorem Proving*, Springer, pp. 123–139, doi:10.1007/978-3-319-43144-4_8.

[5] Etienne Callies & Olivier Laurent (2021): *Click and coLLecT An Interactive Linear Logic Prover*. In: *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*. Available at `https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271501`.

[6] David M Cerna, Rafael PD Kiesel & Alexandra Dzhiganskaya (2020): *A mobile application for self-guided study of formal reasoning*. doi:10.48550/arXiv.2002.12553.

[7] Emscripten Contributors (2022): *Building to WebAssembly*. Available at `https://web.archive.org/web/20221030175022/https://emscripten.org/docs/compiling/WebAssembly.html`.

[8] Leonardo De Moura & Nikolaj Bjørner (2008): *Z3: An efficient SMT solver*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, doi:10.1007/978-3-540-78800-3_24.

[9] Pablo Donato, Pierre-Yves Strub & Benjamin Werner (2022): *A drag-and-drop proof tactic*. In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 197–209, doi:10.1145/3497775.3503692.

[10] Arno Ehle (2017): *Proof Search in the Sequent Calculus for First-Order Logic with Equality*. Master's thesis, Universität Kassel.

[11] Arno Ehle, Norbert Hundeshagen & Martin Lange (2015): *The sequent calculus trainer-helping students to correctly construct proofs*. arXiv preprint arXiv:1507.03666, doi:10.48550/arXiv.1507.03666.

[12] Arno Ehle, Norbert Hundeshagen & Martin Lange (2018): *The sequent calculus trainer with automated reasoning-helping students to find proofs*. arXiv preprint arXiv:1803.01467, doi:10.48550/arXiv.1803.01467.

[13] Olivier Gasquet, François Schwarzentruber & Martin Strecker (2011): *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*, Springer, pp. 85–92, doi:10.1007/978-3-642-21350-2_11.

[14] Charles Antony Richard Hoare (1969): *An axiomatic basis for computer programming*. Communications of the ACM 12(10), pp. 576–580, doi:10.1145/363235.363259.

[15] Matteo Kamm & Mike Marti (2019): *The Sequent Calculus Calculator*. Bachelor's thesis, Hochschule für Technik Rapperswil. Available at `https://eprints.ost.ch/id/eprint/798/1/FS2019-BA-EP-Marti-Kamm-TheSequentCalculusCalculator.pdf`.

[16] Richard W. Kaye (2015): *Proofs as structured lists and proof trees*. Available at `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: Automated Reasoning about Software, Fall 2021*. Available at `http://web.archive.org/web/20211217155727/https://www.cs.princeton.edu/courses/archive/fall21/cos516/`.

[18] Graham Leach-Krouse (2018): *Carnap: an open framework for formal reasoning in the browser*. arXiv preprint arXiv:1803.03092, doi:10.48550/arXiv.1803.03092.

[19] Sorin Lerner, Stephen R Foster & William G Griswold (2015): *Polymorphic blocks: Formalism-inspired UI for structured connectors*. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 3063–3072, doi:10.1145/2702123.2702302.

[20] Lena Magnusson & Bengt Nordström (1993): *The ALF proof editor and its proof engine*. In: *International Workshop on Types for Proofs and Programs*, Springer, pp. 213–237, doi:10.1007/3-540-58085-9_78.

[21] David Majda & Futago za Ryuu (2022): *PEG.js - Parser Generator for JavaScript*. Available at `https://web.archive.org/web/20221007224930/https://pegjs.org/`.

[22] Conor McBride & James McKinna (2004): *The view from the left*. Journal of Functional Programming 14(1), pp. 69–111, doi:10.1017/S0956796803004829.

[23] Tobias Nipkow, Markus Wenzel & Lawrence C Paulson (2002): *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, doi:10.1007/3-540-45949-9.

[24] Ulf Norell (2007): *Towards a practical programming language based on dependent type theory*. Ph.D. thesis, Chalmers University of Technology. Available at `https://www.cse.chalmers.se/~ulfn/papers/thesis.pdf`.

[25] Liam O'Connor & Rayhana Amjad (2022): *Holbert: Reading, Writing, Proving and Learning in the Browser*. arXiv preprint arXiv:2210.11411, doi:10.48550/arXiv.2210.11411.

[26] Giselle Reis, Zan Naeem & Mohammed Hashim (2020): *Sequoia: a playground for logicians*. In: *International Joint Conference on Automated Reasoning*, Springer, pp. 480–488, doi:10.1007/978-3-030-51054-1_-32.

[27] M. E. Szabo, editor (1969): *The Collected Papers of Gerhard Gentzen*. Studies in Logic and The Foundations of Mathematics, North-Holland Publishing Company, Amsterdam.

[28] Jørgen Villadsen, Asta Halkjær From & Anders Schlichtkrull (2019): *Natural Deduction Assistant (NaDeA)*. In: *Theorem Proving Components for Educational Software*, doi:10.48550/arXiv.1904.00618.

[29] A. A. Voronkov (1990): *A proof-search method for the first order logic*. In: *COLOG-88*, Springer, pp. 327–338, doi:10.1007/3-540-52335-9_63.

[30] Hao Wang (1960): *Toward mechanical mathematics*. IBM Journal of Research and Development 4(1), pp. 2–22, doi:10.1147/rd.41.0002.

[31] Edward Z. Yang (2012): *Logitext*. Available at `https://github.com/ezyang/logitext`.

[32] Juriy Zaytsev, Stefan Kienzle & Andrea Bogazzi (2022): *Fabric.js JavaScript Canvas Library*. Available at `https://web.archive.org/web/20221112201025/http://fabricjs.com/`.