PROOFBUDDY
Acquiring Proof Competence with Friendly Assistance

Nadine Karsten
Technische Universität Berlin
Berlin, Germany
n.karsten@tu-berlin.de

Frederik Krogsdal Jacobsen
Technical University of Denmark
Kongens Lyngby, Denmark
fkjac@dtu.dk

Uwe Nestmann
Technische Universität Berlin
Berlin, Germany
uwe.nestmann@tu-berlin.de

Jørgen Villadsen
Technical University of Denmark
Kongens Lyngby, Denmark
jovi@dtu.dk

Proof competence, i.e. the ability to write and check (mathematical) proofs, is an important skill in Computer Science, but for many students it represents a difficult challenge. The main issues are the correct use of formal language and the ascertainment whether proofs, especially the students' own ones, are complete and correct. Proof competence should already be taught in introductory courses. To improve on this endeavour, we introduce PROOFBUDDY, a web-based tool using the Isabelle proof assistant. The approach allows beginning Computer Science students to understand proofs as programs using pattern matching over inductive types. Using a web-based tool has two advantages: (1) Students do not have to install Isabelle themselves; (2) Instructors may be provided with data to support the individual learning processes. We plan to test the approach in several courses at the Technical University of Berlin (TUB) and the Technical University of Denmark (DTU).

1 Introduction

The curriculum of a Bachelor study programme in Computer Science typically includes programming skills, technical understanding about computers, theoretical knowledge and math. Especially the latter two cause difficulties for students. Theoretical courses cover formal languages, automata, logic, complexity and computability, which all have in common that they build on mathematical structures and proofs about them. Hence Computer Science students have to write and verify mathematical and logical proofs in several theoretical courses. This requires *proof competence*, which includes the following four specific competencies [5]: (1) *professional* competence describes the contextual knowledge about the proposition that has to be proved; (2) writing down a proof in sufficiently formal language is called *representation* competence; (3) *communication* competence is then understood as arguing about the procedure and solution of a proof; (4) finally, *methodological* competence summarizes three aspects [9]: proof scheme, proof structure and chain of conclusions. All of these competencies should be taught in introductory courses in Theoretical Computer Science.

The teaching of proof competence with the above-mentioned facets is usually not an explicit part of the curriculum. As a result, students work through proofs that the teacher presents during lectures and try to replicate [22]. Most students fail this way. In [7], Frede and Knobelsdorf analyzed the homework and the written exams of an introductory course at TU Berlin. The result was that students make the most mistakes in exercises with proofs, regardless of the final grade. The main challenges in writing proofs are the usage of formal language while writing proofs, and the ascertainment whether a proof is complete and correct [11–13]. Therefore, the failure rate in introductory courses in Theoretical Computer

Submitted to:
TFPIE 2023

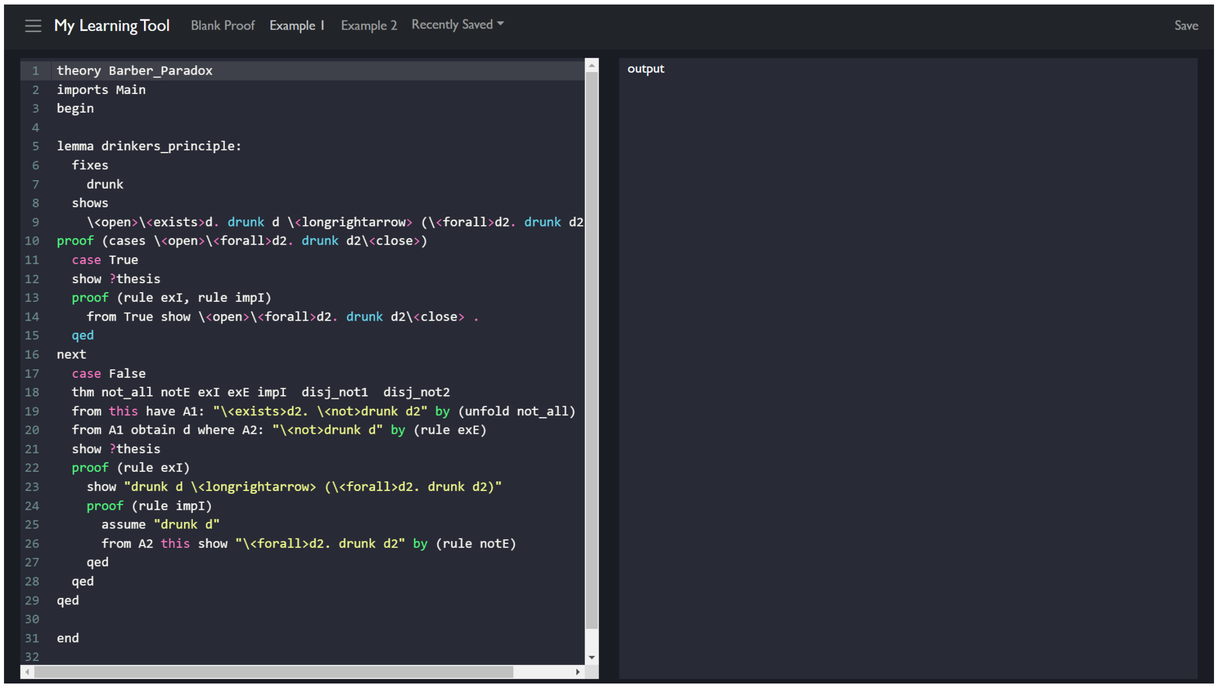
Science courses is high. We claim that this failure rate is due to the fact that students lack sufficient individual feedback for their proofs. In classical lectures, teachers usually just present given proofs without discussing the process of constructing them starting from a target proposition [22] and with small gaps [15]. In exercise groups, teachers certainly talk about the concepts of the course and especially proofs. However, usually, there is not enough time for students to attempt proofs on their own. The first time that students actually try to develop and write proofs is as part of some homework. Up to this point, though, students will not have had any feedback on their own conceptions. There may be teachers' office hours, where students can ask questions and can ask for feedback, but there cannot really be feedback provided at this point on the usage of formal language, on completeness and correctness of proof candidates, because this will in most cases only come *after* the homework was handed in. Unfortunately, and quite normally, students get their annotated homework back only days or even weeks after having it handed in. This is simply too late for students to reconstruct their failures and usually there is no possibility to correct the proofs and submit them a second time. As a consequence, students cannot learn from their mistakes.

2 Proof Assistants and their Friendliness

In order to reduce the time for students between writing a proof and getting feedback on it—without requiring large amounts of human resources—we suggest to use software to evaluate their proof work. *Theorem provers* have been developed to find and verify proofs. They may be fully automatic or depend on interaction with human provers. *Automatic* theorem provers try to solve theorems by themselves; their proofs are mostly not intended to be read and understood by humans and they are limited in what propositions they may find proofs for. *Interactive* theorem provers (also known as proof *assistants*) require the collaboration with humans; they offer book-keeping support to check for completeness of user-provided proofs, but they usually only offer limited support to find proofs automatically. Otherwise, the main advantage is that each proof step provided by the user is checked for correctness and can be understood as part of a proofscript. Isabelle [18, 19], Coq [2] and Lean [16] are some of the best known proof assistants. Since proof assistants are generally based on definitions in a functional programming language, the proofs encountered in a typical Computer Science program are especially suited for evaluation by a proof assistant, as detailed below. In the following, we focus on Isabelle/HOL (Higher-Order Logic), the language of which can broadly be considered as a combination of functional programming and logic.

For Computer Science students in particular, Isabelle/HOL has several advantages. (1) For students who are already competent functional programmers, definitions (i.e. functions) in the language of Isabelle/HOL have a familiar look. For the others, the possibility of using the usual logical and mathematical symbols in definitions can make the entry into functional programming smoother. (2) Proofs by induction or case analysis in Isabelle/HOL closely resemble the notion of pattern matching in functional languages. (3) Proofs about logical formulas can be interpreted as proofs about the types of functions, and the type system of Isabelle/HOL makes this correspondence quite clear. (4) In Isabelle/HOL, users have the possibility to write proofs in the so-called "Isar-style", using a mixture of forward and backward reasoning [23]. This style allows users to formulate assumptions and (sub)goals explicitly. In our opinion, this way of writing proofs makes it easier for beginners to follow and construct proofs, as they do resemble detailed pen-and-paper proofs quite closely. (5) The students can be motivated to defeat the machine [17]. Each lemma can be seen as a level which has to be completed such that there is no subgoal left.

Unfortunately, the use of Isabelle/HOL also has several disadvantages. First, students must be able to install Isabelle on their computer to use it. Second, while Isabelle/HOL is very powerful, the interface and language used is not very intuitive and friendly to beginners. The functional programming language of Isabelle/HOL allows for a number of constructions, such as the use of common logical and mathematical



```
theory Barber_Paradox
imports Main
begin

lemma drinkers_principle:
  fixes
    drunk
  shows
    \<open>\<exists>d. drunk d \<longrightarrow> (\<forall>d2. drunk d2
proof (cases \<open>\<forall>d2. drunk d2\<close>)
  case True
  show ?thesis
  proof (rule exI, rule impI)
    from True show \<open>\<forall>d2. drunk d2\<close> .
  qed
next
  case False
  thm not_all notE exI exE impI  disj_not1  disj_not2
  from this have A1: "\<exists>d2. \<not>drunk d2" by (unfold not_all)
  from A1 obtain d where A2: "\<not>drunk d" by (rule exE)
  show ?thesis
  proof (rule exI)
    show "drunk d \<longrightarrow> (\<forall>d2. drunk d2)"
    proof (rule impI)
      assume "drunk d"
      from A2 this show "\<forall>d2. drunk d2" by (rule notE)
    qed
  qed
qed

end
```

Figure 1: Screenshot of the interface of a prototype of PROOFBUDDY.

notation, which are unusual in the overall landscape of functional programming languages. Additionally, students may have a hard time understanding how to state and begin proofs in the formal Isar language.

To solve these problems, we introduce PROOFBUDDY, a browser-based tool interfacing an Isabelle instance running on a web server. A screenshot of the tool can be seen in Figure 1; on the left-hand side, it shows the Isar-style proof of the "drinkers principle". On the right-hand side, there will be the output from Isabelle; none is displayed when the proof is correct. Therefore the web framework communicates with the Isabelle server. This allows us to add more—and also more friendly—feedback to the output, depending on the failure or the behavior of the user. Our tool permits students to use Isabelle without having to install the Isabelle application locally. Additionally, defining our own interface enables us to conduct experiments with subsets of both the functional programming language and the proof language of Isabelle/HOL. This makes it possible to, for example, introduce features one at a time, ensuring that students can only use the features that they are supposed to learn in a given session.

The major benefit, however, of using a web-based approach is the ability to monitor the progress and behavior of each student as they learn to program and prove in Isabelle/HOL. Moreover, exercises can be designed e.g. by restricting the language, and monitored for completion. This also opens the possibility of forcing students to complete exercises in a certain order, and to tailor exercises to the pace and needs of the individual student. Finally, the approach may be used for assignments or exams, where it is desired to restrict students to solve problems, e.g. proving a lemma without using induction.

3 Teaching Approach

We plan to test PROOFBUDDY in an existing MSc-level course on automated reasoning at DTU in the spring of 2023. This course is for advanced Computer Science students, and the main topic of the course is learning Isabelle. We thus expect students to already be somewhat proficient at pen-and-paper proofs, and the main purpose of using the tool in this course is not to teach the students basic proofs, but to determine how students use the tool. As discussed above, introducing the students to Isabelle through a web-based tool instead of immediately giving them their own Isabelle installation makes it possible to introduce concepts in a restricted setting, and to determine which concepts seem to be difficult for the students to learn by monitoring their behavior and progress while using the tool. By first testing out ProofBuddy with MSc-level students, who should not find our approach too difficult, we expect to be able to improve the tool and our exercises before using the approach with BSc-level students.

We have developed a new BSc-level course at TUB, where we focus on teaching how to create and properly write down proofs. The course is planned for summer 2023. Referring to our hypotheses, the concept of the course is to enable substantially more feedback for students. Thus, we use ProofBuddy in addition to the help of the teacher and fellow students to provide the feedback much more directly (in real time) and also more individually. The idea is to start with propositional logic and first-order logic, because these constitute the foundations of proofs [22]. Afterwards, we introduce inductive data structures and prove properties about them. We want to avoid that students just learn to use the tool without actually understanding the proofs that they develop with it. Therefore, the whole conception of the course aims to strengthen the mutual transformation between formal proofs—as developed with ProofBuddy—and traditional pen-and-paper proofs, in both directions. Confronted with these transformations, students are supposed to also learn that there are different degrees of formality to prove propositions.

4 Related Work

The idea to use proof assistants for teaching to give the students more direct feedback is not entirely new. Many courses that employ this idea address students in higher semesters and assume proof competence and functional programming skills [17,24]. In [13], Knobelsdorf et al. report on the use of Coq [2] to enhance proof competence, with the following result: While students after the course were able to prove theorems with Coq, their ability to write pen-and-paper proofs was not significantly better than before the course. Therefore, Böhne and Kreitz try in [4] to improve pen-and-paper proofs by requiring to add comments in the formal Coq proof; nevertheless the improvement of writing pen-and-paper proofs was only minimal. There are some approaches to integrate a proof assistant in first-year courses. Lean [16] was used by Avigad [1] in a logic course in 2015 to combine mathematical language and natural language in proofs. Lean was also used for the Natural Number Game [6], which is a web application designed to teach formal proofs in a proof assistant using basic arithmetic theorems. In [3], Kreitz, Knobelsdorf and Böhne consider a first-semester Bachelor course, where Coq is used in the lectures and the exercises to support the handling of formalisms. This course was not realized. At TUB, we integrated Isabelle [18,19] in a second semester Bachelor course in 2021. Isabelle was used to introduce propositional and first-order logic step by step. Using Isabelle for the exercises and homework was optional. The students had fun in proving but it was time-consuming for them to step into Isabelle.

There are many other tools with the purpose to teach how to prove. To name just a few: Jape [25,26], ProofWeb [10], SPA [21], SeCaV [8], and NaDeA [27–29]. The web application Clide [14,20] is based on Isabelle. Clide was implemented already in 2013 and consists of an editor integrated within a web interface, and an Isabelle backend for checking proofs. Clide was not focused on teaching, however, but focused on allowing collaborative writing and editing of proofs. The Clide application seems to no longer be available.

5 Concluding Remarks

We introduce ProofBuddy, a web-based tool for monitoring and guiding the use of Isabelle/HOL by students. Such an approach allows instructors to gain insight into how students try to write proofs; it also enables us to carry out didactic research on student behaviour. The use of Isabelle/HOL allows students to write proofs about functional programs, and many concepts in, e.g., logic can naturally be encoded as functional programs. The approach thus also allows us to study how students write and prove properties about inductive definitions and use pattern matching. We expect that the results of such studies could be used to design guided exercise series which support the learning progression of individual students.

References

[1] Jeremy Avigad (2019): *Learning Logic and Proof with an Interactive Theorem Prover.* In Gila Hanna, David A. Reid & Michael de Villiers, editors: *Proof Technology in Mathematics Research and Teaching*, Springer International Publishing, Cham, pp. 277–290, doi:10.1007/978-3-030-28483-1_13.

[2] Yves Bertot & Pierre Castéran (2004): *Interactive theorem proving and program development. Coq'Art: The Calculus of inductive constructions.* Springer Berlin, Heidelberg, doi:10.1007/978-3-662-07964-5.

[3] Sebastian Böhne, Maria Knobelsdorf & Christoph Kreitz (2016): *Mathematisches Argumentieren und Beweisen mit dem Theorembeweiser Coq.* In A. Schwill & U. Liuckey, editors: *HDI 2016 – 7. Fachtagung zur Hochschuldidaktik der Informatik*, *Commentarii informaticae didacticae* 10, Universitätsverlag Potsdam, pp. 69–80. (in German).

[4] Sebastian Böhne & Christoph Kreitz (2018): *Learning how to Prove: From the Coq Proof Assistant to Textbook Style.* *Electronic Proceedings in Theoretical Computer Science* 267, p. 1–18, doi:10.4204/eptcs.267.1. Available at `http://dx.doi.org/10.4204/EPTCS.267.1`.

[5] Esther Brunner (2014): *Mathematisches Argumentieren, Begründen und Beweisen.* Springer Spektrum Berlin, Heidelberg, doi:10.1007/978-3-642-41864-8. (in German).

[6] Kevin Buzzard & Mohammed Pedramfar (2021): *The Natural Number Game.* Available at `https://www.ma.imperial.ac.uk/%7Ebuzzard/xena/natural_number_game/`.

[7] Christiane Frede & Maria Knobelsdorf (2018): *Explorative Datenanalyse der Studierendenperformance in der Theoretischen Informatik.* In N. Bergner, R. Röpke, U. Schroeder & D. Krömker, editors: *Hochschuldidaktik der Informatik - HDI 2018 - 8. Fachtagung des GI-Fachbereichs und Ausbilding/Didaktik der Informatik, Frankfurt, Germany, September 12-13, 2018*, Universitätsverlag Potsdam, pp. 135 – 149. (in German).

[8] Asta Halkjær From, Frederik Krogsdal Jacobsen & Jørgen Villadsen (2022): *SeCaV: A Sequent Calculus Verifier in Isabelle/HOL.* In Mauricio Ayala-Rincon & Eduardo Bonelli, editors: Proceedings 16th *Logical and Semantic Frameworks with Applications*, Buenos Aires, Argentina (Online), 23rd - 24th July, 2021, *Electronic Proceedings in Theoretical Computer Science* 357, Open Publishing Association, pp. 38–55, doi:10.4204/EPTCS.357.4.

[9] Aiso Heinze & Kristina Reiss (2003): *Reasoning and Proof: Methodological Knowledge as a Component of Proof Competence.* In Maria Alessandra Mariotti, editor: *Proceedings of the Third Conference of the European Society for Research in Mathematics Education*, pp. 1–10. Thematic Working Group 4, paper 5.

[10] Maxim Hendriks, Cezary Kaliszyk, Femke van Raamsdonk & Freek Wiedijk (2010): *Teaching logic using a state-of-the-art proof assistant.* *Acta Didactica Napocensia* 3(2), pp. 35–48.

[11] Felix Kiehn, Christiane Frede & Maria Knobelsdorf (2017): *Was macht Theoretische Informatik so schwierig? Ergebnisse einer qualitativen Einzelfallstudie.* In Maximilian Eibl & Martin Gaedke, editors: *INFORMATIK 2017*, Gesellschaft für Informatik, Bonn, pp. 267–278, doi:10.18420/in2017_20. (in German).

[12] Maria Knobelsdorf & Christiane Frede (2016): *Analyzing Student Practices in Theory of Computation in Light of Distributed Cognition Theory.* In: *Proceedings of the 2016 ACM Conference on International Computing*

Education Research, ICER '16, Association for Computing Machinery, New York, NY, USA, p. 73–81, doi:10.1145/2960310.2960331. Available at `https://doi.org/10.1145/2960310.2960331`.

[13] Maria Knobelsdorf, Christiane Frede, Sebastian Böhne & Christoph Kreitz (2017): *Theorem Provers as a Learning Tool in Theory of Computation*. In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, Association for Computing Machinery, New York, NY, USA, p. 83–92, doi:10.1145/3105726.3106184. Available at `https://doi.org/10.1145/3105726.3106184`.

[14] Christoph Lüth & Martin Ring (2013): *A Web Interface for Isabelle: The Next Generation*. In Jacques Carette, editor: *Conferences on Intelligent Computer Mathematics CICM 2013*, *Lecture Notes in Artificial Intelligence* 7961, Springer, pp. 326–329.

[15] Robert C Moore, Martha Byrne, Sarah Hanusch & Timothy Fukawa-Connelly (2016): *When we Grade Students' Proofs, Do They Understand our Feedback?* In Tim Fukawa-Connelly, Nicole Engelke Infante, Megan Wawro & Stacy Brown, editors: *Proceedings of the 19th Annual Conference on Research in Undergraduate Mathematics Education*, Special Interest Group of the Mathematics Association of America, pp. 310–324.

[16] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Doorn & Jakob Raumer (2015): *The Lean Theorem Prover (System Description)*. In: *Automated Deduction - CADE-25*, *Lecture Notes in Computer Science* 9195, pp. 378–388, doi:10.1007/978-3-319-21401-6_26.

[17] Tobias Nipkow (2012): *Teaching Semantics with a Proof Assistant: No more LSD Trip Proofs*. In V. Kuncak & A. Rybalchenko, editors: *Verification, Model Checking, and Abstract Interpretation (VMCAI 2012)*, *LNCS* 7148, Springer, pp. 24–38.

[18] Tobias Nipkow (2021): *Programming and Proving in Isabelle/HOL*. Available at `https://isabelle.in.tum.de/dist/Isabelle2021/doc/prog-prove.pdf`.

[19] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. *LNCS* 2283, Springer.

[20] Martin Ring & Christoph Lüth (2014): *Real-time collaborative Scala development with Clide*. In Heather Miller & Philipp Haller, editors: *Proceedings of the Fifth Annual Scala Workshop*, ACM, pp. 63–66.

[21] Anders Schlichtkrull, Jørgen Villadsen & Asta Halkjær From (2018): *Students' Proof Assistant (SPA)*. In Pedro Quaresma & Walther Neuper, editors: *Proceedings 7th International Workshop on Theorem proving components for Educational software, ThEdu@FLoC 2018, Oxford, United Kingdom, 18 july 2018*, *EPTCS* 290, pp. 1–13, doi:10.4204/EPTCS.290.1. Available at `https://doi.org/10.4204/EPTCS.290.1`.

[22] John Selden & Annie Selden (2009): *Teaching Proving by Coordinating Aspects of Proofs with Students' Abilities*. In: *Teaching and Learning Proof Across the Grades*, chapter 20, Routledge, pp. 339–354, doi:10.4324/9780203882009.

[23] Daniel Solow (2014): *How to Read and Do Proofs*, 6th edition. John Wiley & Sons, Inc.

[24] Alexander Steen, Max Wisniewski & Christoph Benzmüller (2016): *Einsatz von Theorembeweisern in der Lehre*. *Commentarii informaticae didacticae* 10, pp. 81–92. (in German).

[25] Bernard Sufrin & Richard Bornat (1996): *User interfaces for generic proof assistants part I: Interpreting gestures*. *Proc. of User Interfaces for Theorem Provers (UITP-06), York, UK*.

[26] Bernard Sufrin & Richard Bornat (1998): *User interfaces for generic proof assistants part II: Displaying proofs*. *User Interfaces* 98, p. 147.

[27] Jørgen Villadsen, Andreas Halkjær From & Anders Schlichtkrull (2018): *Natural Deduction and the Isabelle Proof Assistant*. *Electronic Proceedings in Theoretical Computer Science* 267, p. 140–155, doi:10.4204/eptcs.267.9. Available at `http://dx.doi.org/10.4204/EPTCS.267.9`.

[28] Jørgen Villadsen, Andreas Halkjær From & Anders Schlichtkrull (2019): *Natural Deduction Assistant (NaDeA)*. *Electronic Proceedings in Theoretical Computer Science* 290, p. 14–29, doi:10.4204/eptcs.290.2. Available at `http://dx.doi.org/10.4204/EPTCS.290.2`.

[29] Jørgen Villadsen, Alexander Jensen & Anders Schlichtkrull (2015): *NaDeA: A Natural Deduction Assistant with a Formalization in Isabelle*. *Journal of Applied Logics* 4(1), pp. 55–82.