Teaching Algorithms and Data Structures with a Proof Assistant (Invited Talk)

Tobias Nipkow
Technical University of Munich
Germany
http://www.in.tum.de/~nipkow

Abstract

We report on a new course *Verified Functional Data Structures and Algorithms* taught at the Technical University of Munich. The course first introduces students to interactive theorem proving with the Isabelle proof assistant. Then it covers a range of standard data structures, in particular search trees and priority queues: it is shown how to express these data structures functionally and how to reason about their correctness and running time in Isabelle.

CCS Concepts: • **Software and its engineering** → **Software verification**; **Formal software verification**; Functional languages; • **Applied computing** → **Education**.

Keywords: Data structures, verification, teaching, Isabelle

ACM Reference Format:
Tobias Nipkow. 2021. Teaching Algorithms and Data Structures with a Proof Assistant (Invited Talk). In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '21), January 18–19, 2021, Virtual, Denmark.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3437992.3439910

1 Introduction

Teaching a course on programming language semantics and formalizing some of its meta-theory in the proof assistant Isabelle in the 1990s led me to fantasize about a *Mechanized Semantics Textbook* [7, 8] and the possibility to have the students of a semanics course do their own proofs in the proof assistant. Roughly ten years later, two incarnations of this concept emerged at UPenn [16] and TU Munich [9] which lead to the two mechanized semantics textbooks *Software Foundations*[1] in Coq [2] and *Concrete Semantics* [13] in Isabelle [14].

This article is about a course (and a related book [11]) on verified functional data structures and algorithms taught annually since 2017 at TUM. Like *Concrete Semantics*, it is entirely based on Isabelle, but now the subject matter is algorithms instead of semantics. Of course correctness of algorithms, but also their running time complexity (including amortized analysis). The running time of a function f is expressed as a separate function T_f whose definition is directly derived from the definition of f but which counts computation steps. Alternatively one can also compute f and T_f simultaneously in a monadic framework [6, 10].

The next two sections present the actual contents of the course and the related book. The final section discusses generic aims and principles underlying the course.

2 Verified Textbook Algorithms

A recent survey [12] has underlined that the time is ripe for a textbook on verified algorithms: the authors surveyed which algorithms in the famous textbook by Cormen *et al.* [4] had been verified in an (automatic or interactive) theorem prover in the literature. It turns out that the only major omissions in the core of the book (excluding *Selected Topics*) are hashing including probabilities, Fibonacci heaps and van Emde Boas trees.

However, many of the algorithms in [4] are of an imperative nature. A second source of inspiration for us was Okasaki's influential book *Purely Functional Data Structures* [15].

3 The Course

3.1 Format

The course is open to both bachelor's and master's students. There is one 1.5 hours lecture and one 1.5 hours lab session every week for 14 weeks. There is one homework sheet per week. The course is worth 5 ECTS points, with 30 ECTS

CPP '21, January 18–19, 2021, Virtual, Denmark

ACM ISBN 978-1-4503-8299-1/21/01.
https://doi.org/10.1145/3437992.3439910

[1] Meanwhile a series of four volumes, the second of which is dedicated to programming languge semantics [17], the third to functional algorithms [1]

points being the average workload per semester. The students must have some background in discrete maths (including an introduction to proofs), algorithms and functional programming.[2]

The whole course utilizes the proof assistant Isabelle: in the lectures, the lab sessions and the homework. The final grade is a rougly equal combination of homework and a final exam. The 2020 exam was online, also using Isabelle, previous exams were paper-based.

3.2 Contents

The first 5 weeks of the course are dedicated to an introduction to Isabelle. The remaining 9 lectures are dedicated to data structures and algorithms. They are typically concerned with sorting, search trees, priority queues and amortized analysis. The topics are selected from chapters of a book in the making [11] with the following contents:

- Sorting: insertion sort, quicksort,
  top-down and bottom-up mergesorts
- Selection in linear time
- Binary trees
- Search trees:
  2-3 trees, red-black trees, AVL trees , Braun trees, tries,
  join-based $\cup$, $\cap$ and $\setminus$
- Huffman's algorithm
- Priority queues:
  leftist heaps, Braun trees, binomial heaps
- Dynamic programming
- Amortized analysis:
  splay trees, skew heaps, pairing heaps

Although suitable for teaching, the book is also meant as the definitive reference for precise correctness and complexity proofs: for some of the functional data structures in the book, e.g. Braun trees, precise proofs were missing in the literature (before the authors' own recent work that is the basis of the book).

This is a *live* book that is meant to grow. New chapters are already in preparation, e.g. about the Hood-Melville queue, and the list of authors will grow. *You* could become a co-author yourself!

4 Aims and Principles

There are the obvious concrete aims: teach the algorithms themselves, their functional formulation and their correctness and running time proofs. Equally important are certain abstract aims that we discuss below.

Algorithms are Logic

Algorithms are often taught with little emphasis on formalities and correctness proofs. An important aim of this course is to instill into the student's mind the understanding that one can reason about the correctness and complexity of algorithms expressed in a functional style just as easily as about numbers. Not just in principle, but with the help of a proof assistant also in practice.

The remaining aims and principles are independent of the subject matter and apply to any course utilizing an interactive theorem prover.

No More LSD Trip Proofs!

Computer science students frequently lack the training to write coherent proofs. Next to the algorithms, the central aim of the course is to teach the students the art of precise logical proofs. I believe that this goal is best reached if the following two principles are followed.

Teach Proofs, Not Proof Scripts

Most theorem provers provide a scripting language for writing proofs, typically as sequences of commands. Such proofs are for machines, not for humans. They do not show what property is being proved at each point. They are like assembly language programs. They do not convey ideas. I believe that proper proofs are best taught in some high-level structured proof language as offered, for example, by Mizar [5] and Isabelle [18].

Isabelle's structured proof language *Isar* is used for most of the course. It is close to the informal language of mathematics and allows a smooth transition in the presentation style during the course: from Isar proofs on the machine to more traditional proofs on the blackboard (see below).

Teach Proofs, Not Logic

Mathematicians have been doing this successfully for a long time. To be provocative: proof systems like natural deduction belong in logic courses, where the fine structure of logic is studied. But constructing large proofs by single step inferences in some proof system is a straightjacket. Application-oriented courses (remember: the course topic is not logic!) should reason *modulo logic*: if the student believes that A together with B implies C, she should be able to just write

`from` A `and` B `conclude` C `by` *hammer*

where *hammer* is some suitable proof method of the underlying proof assistant. Isar allows exactly that, and Isabelle offers a number of automatic hammers for this purpose, in particular the connection to powerful external automatic provers [3]. The motto is: *Do not let logic dominate your thinking, let automatic provers take care of logic.*

It has to be acknowledged that using a theorem prover does not guarantee the intended learning outcome. Proof automation prevents frustration but also permits students to create proofs without understanding the underlying logical principles, e.g. induction.

[2] At TUM, these three courses are a mandatory part of the undergraduate curriculum.

Teach the Subject Matter, Not the Proof Assistant

I believe that teaching a course with a proof assistant is most convincing for the students if it focusses on the subject matter (here: algorithms and data structures) rather than the proof assistant. It is inevitable that in the beginning the proof assistant needs to be introduced (in our case 5 out of 14 weeks) and that for most of the course the students will struggle with it. But the more it can be deemphasized in the later lectures on the subject matter, the better. This leads to another principle:

Do Not Let the Proof Assistant Dominate Your Presentation

In the beginning of the course, when introducing the proof assistant, it is essential to demonstrate the interaction with the proof assistant in class for long periods.

During the second part of the course I gradually move to conventional presentations based on slides and the blackboard, although I never completely abandon Isabelle. I believe that slides (with animations) and the blackboard are better suited to explain many concepts and proofs than an Isabelle demo is. I rely on Isabelle's LaTeX generation facility that supports prettyprinting of definitions and theorems from Isabelle theories on slides without having to type them in a second time (and getting them wrong).

When moving to the blackboard for developing proofs, I initially stick closely to Isar to phrase these proofs. As the students become more comfortable with Isar, I begin taking more and more liberties on the blackboard, moving towards informal proofs. The aim is to strengthen the students' ability to bridge the gap between formal and informal proofs. While the lecture presentations gradually move away from Isabelle, the homework still is Isabelle-based and the students have to convert their intuition into Isabelle proofs.

Acknowledgments

Peter Lammich and Mohammad Abdulaziz co-taught the course with me for the past four years and commented on this extended abstract. This research is supported by DFG Koselleck grant NI 491/16-1.

References

[1] Andrew W. Appel. *Verified Functional Algorithms*, volume 3 of *Software Foundations*. Electronic textbook, 2020. http://softwarefoundations.cis.upenn.edu.

[2] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Texts in Theoretical Computer Science. Springer, 2004.

[3] Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. Automatic proof and disproof in Isabelle/HOL. In C. Tinelli and V. Sofronie-Stokkermans, editors, *Frontiers of Combining Systems (FroCoS 2011)*, volume 6989 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2011.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[5] Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. Mizar in a nutshell. *Journal of Formalized Reasoning*, 3(2):153–245, 2010. URL: https://jfr.unibo.it/article/view/1980.

[6] Jay A. McCarthy, Burke Fetscher, Max S. New, Daniel Feltey, and Robert Bruce Findler. A Coq library for internal verification of running-times. In Oleg Kiselyov and Andy King, editors, *Functional and Logic Programming, FLOPS 2016*, volume 9613, pages 144–162. Springer, 2016. URL: https://doi.org/10.1007/978-3-319-29604-3_10.

[7] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. In Vijay Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 1996. URL: https://doi.org/10.1007/3-540-62034-6_48.

[8] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. *Formal Aspects Comput.*, 10(2):171–186, 1998. URL: https://doi.org/10.1007/s001650050009.

[9] Tobias Nipkow. Teaching semantics with a proof assistant: No more LSD trip proofs. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation, VMCAI 2012*, volume 7148 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-27940-9_3.

[10] Tobias Nipkow. Verified root-balanced trees. In Bor-Yuh Evan Chang, editor, *Programming Languages and Systems, APLAS 2017*, volume 10695, pages 255–272. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-71237-6_13.

[11] Tobias Nipkow, Jasmin Blanchette, Manuel Eberl, Peter Lammich, Christian Sternagel, Bohua Zhan, et al. Verified functional data structures and algorithms. To appear, 2021.

[12] Tobias Nipkow, Manuel Eberl, and Maximilian P. L. Haslbeck. Verified textbook algorithms - A biased survey. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis, ATVA 2020*, volume 12302 of *Lecture Notes in Computer Science*, pages 25–53. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-59152-6_2.

[13] Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. http://concrete-semantics.org.

[14] Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[15] Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

[16] Benjamin C. Pierce. Lambda, the ultimate TA: using a proof assistant to teach programming language foundations. In Graham Hutton and Andrew P. Tolmach, editors, *Proc. 14th ACM SIGPLAN international conference on Functional programming, ICFP 2009*, pages 121–122. ACM, 2009. URL: https://doi.org/10.1145/1596550.1596552.

[17] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Catalin Hritcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. *Programming Language Foundations*, volume 2 of *Software Foundations*. Electronic textbook, 2020. http://softwarefoundations.cis.upenn.edu.

[18] Markus Wenzel. *Isabelle/Isar — A Versatile Environment for Human-Readable Formal Proof Documents*. PhD thesis, Institut für Informatik, Technische Universität München, 2002.