Using Isabelle in Two Courses on Logic and Automated Reasoning

Jørgen Villadsen(✉) and Frederik Krogsdal Jacobsen

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

Abstract. We present our experiences teaching two courses on formal methods and detail the contents of the courses and their positioning in the curriculum. The first course is a bachelor course on logical systems and logic programming, with a focus on classical first-order logic and automatic theorem proving. The second course is a master course on automated reasoning, with a focus on classical higher-order logic and interactive theorem proving. The proof assistant Isabelle is used in both courses, using Isabelle/Pure as well as Isabelle/HOL. We describe our online tools to be used with Isabelle/HOL, in particular the Sequent Calculus Verifier (SeCaV) and the Natural Deduction Assistant (NaDeA). We also describe our innovative Students' Proof Assistant which is formally verified in Isabelle/HOL and integrated in Isabelle/jEdit using Isabelle/ML.

Keywords: Logic · Automated reasoning · Proof assistant Isabelle

1 Introduction

We present our experiences teaching two courses on formal methods at the Technical University of Denmark (DTU):

- BSc Course: DTU Course 02156 Logical Systems and Logic Programming https://kurser.dtu.dk/course/02156
- MSc Course: DTU Course 02256 Automated Reasoning https://kurser.dtu.dk/course/02256

Both courses are taught in English. Figure 1 shows the objectives and content of the two courses. The objectives need to be approved by the study board and are not expected to change much from year to year. The above links also include some official statistics like evaluations and grades but mostly in Danish. Both courses count for 5 ECTS points, which corresponds to approximately 2 h of lectures and 2 h of group exercise sessions per week, plus individual study and assignment work (expected around 9 h per week in total), for 13 weeks (summing up to 140 h with exam preparations).

J. F. Ferreira et al. (Eds.): FMTea 2021, LNCS 13122, pp. 117–132, 2021.
https://doi.org/10.1007/978-3-030-91550-6_9

(a) Objectives and content of the course Logical Systems and Logic Programming.

General course objectives
The aim of the course is to give the students an introduction to some of the basic declarative formalisms from formal computer science and logic that can be used for describing, analysing and constructing IT systems. It will cover theoretical insight as well as practical skills in relevant high-level programming languages.

Learning objectives
A student who has met the objectives of the course will be able to:

- relate different kinds of proof systems
- construct formal proofs in elementary logics
- exploit selected classical and non-classical logics
- use the backtracking algorithm for simple problem solving
- analyze the effect of a declarative program
- establish a functional design for a given problem, so that the main concepts of the problem are directly traceable in the design
- master logical approaches to programming in terms of defining recursive predicates
- communicate solutions to problems in a clear and precise manner

Content
The course covers logic programming (in particular Prolog as a rapid prototyping tool), elementary logics (including propositional and first-order logic), proof systems (deductive systems and/or refutation systems), and problem solving techniques (for instance the backtracking algorithm).

(b) Objectives and content of the course Automated Reasoning.

General course objectives
Reasoning is the ability to make logical inferences. The aim of the course is to give the students an introduction to automatic and interactive computer systems for reasoning about mathematical theorems as well as properties of IT systems. It will cover theoretical insight as well as practical skills in relevant proof assistants.

Learning objectives
A student who has met the objectives of the course will be able to:

- explain the basic concepts introduced in the course
- express mathematical theorems and properties of IT systems formally
- master the natural deduction proof system
- relate first-order logic, higher-order logic and type theory
- construct formal proofs in the procedural style and in the declarative style
- use automatic and interactive computer systems for automated reasoning
- evaluate the trustworthiness of proof assistants and related tools
- communicate solutions to problems in a clear and precise manner

Content
The natural deduction proof system, first-order logic, higher-order logic and type theory. Formal proofs in the procedural style and in the declarative style using automatic and interactive provers. The Isabelle proof assistant in artificial intelligence and computer science.

Fig. 1. Objectives and course content of the two courses.

The first course is on logical systems and logical programming, and is intended for final-year BSc students (over the years interested students have successfully taken it already at the start of the second year of their bachelor). The course has been given more or less in the same format since 2006 with an increasing number of students, currently around 80 students per year.

The second course is on automated reasoning, and is intended for MSc students (interested students have successfully taken it already during the final year of their bachelor). The course was given for the first time in 2020 and has around 40 students per year.

The main changes due to COVID-19 were online lessons using Zoom and online home exams instead of physical at DTU. We did not make any other changes to the courses and we will not elaborate on the COVID-19 situation in the present paper.

Both of the courses use the proof assistant Isabelle [30] to showcase verified proof systems and provers, which we have implemented in Isabelle. This allows us to discuss common proof methods for e.g. soundness and completeness and allows students to experiment with larger proofs without losing track of what is going on. We also use Isabelle for assignments and exam questions concerning these proof systems. This allows students to get immediate feedback from the proof assistant, and allows us to easily check if the submitted proofs are correct. Recent research indicates that quick formative evaluation has a large impact on learning when teaching introductory computer science [16]. We use Isabelle since it is the proof assistant that we know best and because Isabelle is a generic proof assistant which allows us to use both Isabelle/HOL and Isabelle/Pure as detailed in later sections.

The BSc course additionally revolves heavily around the Prolog programming language, on which we spend around half of the time. Students thus learn to couple logical programming with logic, and we showcase many interesting programs related to the rest of the course content. The MSc course focuses more on functional programming within the Isabelle proof assistant, and how this can be coupled to formal methods and proofs about programs.

To enable this use of Isabelle and Prolog, we need students to hit the ground running so they can use the implementations of the logical concepts from the beginning. We recall the following quote from Donald Knuth:

> *When certain concepts of TEX are introduced informally, general rules will be stated; afterwards you will find that the rules aren't strictly true. In general, the later chapters contain more reliable information than the earlier ones do. The author feels that this technique of deliberate lying will actually make it easier for you to learn the ideas. Once you understand a simple but false rule, it will not be hard to supplement that rule with its exceptions.*

For Isabelle and Prolog, we throw the students into the deep end and return later to explain how everything actually fits together. Unification, for example, is treated informally until late in the course where students have the logical

background to understand how it works and need the details of it in order to master the resolution calculus for first-order logic [3].

On the other hand, we never cut corners about logic itself. With the proof assistant Isabelle/HOL we can create canonical reference documents for logics and their metatheory. The formal language of Isabelle/HOL, namely higher-order logic, is precise and unambiguous. This means every proof can be mechanically checked, and that it is impossible to cheat and omit any details.

We summarize our main points:

1. We use Isabelle in both courses, including the editor Isabelle/jEdit and the Isabelle/ML facilities.
2. By exploring formally verified proof systems and provers, we use formal methods on the field of formal methods itself.
3. In the advanced course we in addition use Isabelle/Pure, showing the generic Isabelle logical framework and forcing students to manage without the automation of Isabelle/HOL.
4. We rely on group exercise sessions with competent teaching assistants and peer assistance in combination with the Isabelle proof assistant and our own tools.
5. We have individual assignments, as often and as early as possible, with a quick feedback loop from the teaching assistants.

In the next section, we discuss related work. In Sect. 3 we detail the position of our courses within the context of the rest of our computer science and software engineering program. Next we describe the BSc course in Sect. 4, followed by a description of the MSc course in Sect. 5. Finally we describe our overall experiences and ideas for future work in Sect. 6 and conclude in Sect. 7.

2 Related Work

Our two courses are based on a number of tools for teaching logic developed in recent years [10–15,21,36–40]. In the present paper we elaborate, for the first time, on the courses and detail our experiences.

We are not aware of any textbooks for teaching logic using the Isabelle proof assistant, but textbooks on formalizing a number of other computer science topics exist, like the book on programming language semantics [24,29] or functional algorithms [26–28]. These books show that the proof assistant Isabelle/HOL can be used for teaching semantics, algorithms and data structures. There are also impressive books for the proof assistant Coq [33] and the proof assistant Lean [1] but we are not aware of approaches to teaching logic and automated reasoning where the proof systems and provers are formalized in a proof assistant. We envision a textbook around our tools, but are currently relying on a number of unpublished smaller notes and tutorials to teach students how to use them.

Bella [2] presents a teaching methodology for the so-called Inductive Method to verified security protocols and notes the following step:

> *But the first and foremost step is to convince the learners that they already somewhat used formal methods, although for other applications, for example in the domains of Physics and Mathematics. The argument will convey as few technicalities as possible, in an attempt to promote the general message that formal methods are not extraterrestrial even for students who are not theorists.*

We attempt to promote a similar message to the students following our courses.

Harrison [19], Blanchette [5] and Reis [34] discuss aspects of formalizing the metatheory of proof systems and provers. In contrast to our work they do not consider the use of such formalizations as central components and tools in logic and automated reasoning courses.

3 Curricular Overview

The first of our courses is the BSc course meant for final-year students, while our second course is the MSc course. We would like to briefly explain the positioning of our courses within the overall computer science and engineering curriculum at the Technical University of Denmark (DTU). The curriculum at DTU is organized in a half-year semester structure, but after the first year students are free to organize their own study plan and have many electives which can be used for any course offered at the institution.

While the MSc course is intended to be followed after our BSc course, our students have very varied backgrounds because many MSc students have BSc degrees from other institutions. The backgrounds of the students following the BSc course are also varied because the course is followed by many exchange students, BEng students, and General Engineering students. Figure 2 shows the context of our courses in the overall computer science and software engineering program.

Course numbers and ECTS points are as follows for the BSc courses:

- 01017 Discrete Mathematics (5 ECTS)
- 02101 Introductory Programming (5 ECTS)
- 02105 Algorithms and Data Structures 1 (5 ECTS)
- 02110 Algorithms and Data Structures 2 (5 ECTS)
- 02141 Computer Science Modelling (10 ECTS)
- 02156 Logical Systems and Logic Programming (5 ECTS)
- 02157 Functional Programming (5 ECTS)
- 02180 Introduction to Artificial Intelligence (5 ECTS)
- 02450 Introduction to Machine Learning and Data Mining (5 ECTS)

We have here omitted the traditional BSc courses in Computer Engineering and Software Engineering as they play only a minor role in this context.

The MSc courses are organized in study lines which are optional to follow, but nevertheless guide the study planning for the students. Except for a single Innovation in Engineering course we do not have any mandatory MSc courses, though

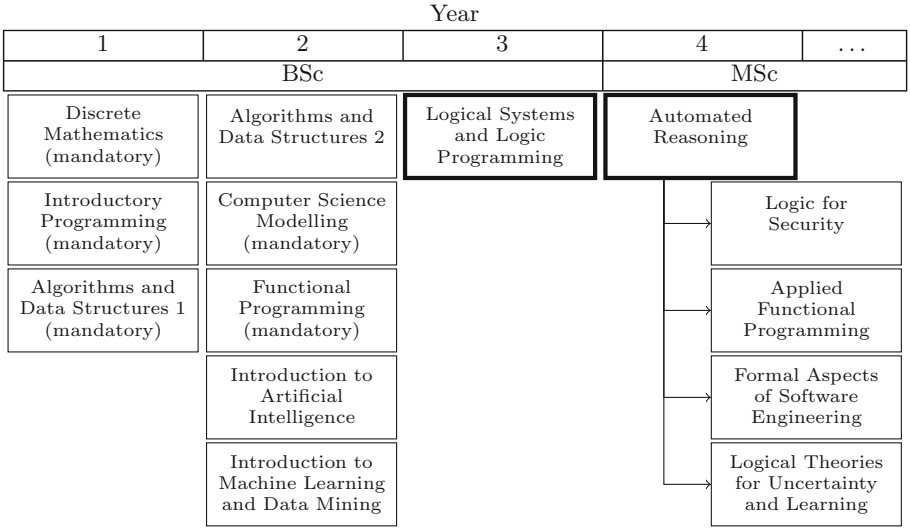

Fig. 2. Suggested course progression surrounding our courses.

students must of course primarily take courses related to computer science. The selected MSc courses to be taken after our courses on logic and automated reasoning are on the following study lines (we also have study lines in Computer Security and Digital Systems, but the former is more practically oriented compared to Safe and Secure by Design and the latter is more Electrical Engineering oriented):

- Study Line: Artificial Intelligence and Algorithms
  02256 Automated Reasoning (5 ECTS)
  02287 Logical Theories for Uncertainty and Learning (5 ECTS)
- Study Line: Embedded and Distributed Systems
  02257 Applied Functional Programming (5 ECTS)
- Study Line: Safe and Secure by Design
  02244 Logic for Security (7.5 ECTS)
- Study Line: Software Engineering
  02263 Formal Aspects of Software Engineering (5 ECTS)

For the BSc course, we recommend that students have previous programming experience as well as knowledge of discrete mathematics and at least basic knowledge of algorithms and data structures. Functional programming is an advantage due to our use of systems implemented in Isabelle/HOL. These prerequisites are obtained in mandatory first and second year courses by most of the students following our BSc course.

However, a significant number of the students following our BSc course are either exchange students, come from the General Engineering program at DTU, or are BEng students. For exchange students, the structure of the curriculum of

their home institution may diverge from ours, which means that they sometimes have quite different backgrounds. Students from the General Engineering program have an interdisciplinary study plan, which means that they may not have all of the recommended prerequisites. Finally, BEng students have a curriculum which differs significantly from that of the BSc students, and are generally more focused on practical applications.

For the MSc course, we recommend that students have followed our BSc course and have experience with functional programming and basic algorithms in artificial intelligence. Students coming from the BSc program at DTU will mostly have these prerequisites, but a large amount of students on our MSc programmes come from other institutions. This means that we generally need to assume that students will not have all of the recommended prerequisites, and especially that they have not followed our BSc course.

Our courses provide skills that are useful in a number of MSc courses at DTU. A firm grasp of logic is of course useful for courses such as Logic for Security and Logical Theories for Uncertainty and Learning. Familiarity with formal methods and logic is useful for a course on Formal Aspects of Software Engineering. Several topics covered in our courses can provide interesting project ideas to implement for a course on Applied Functional Programming or for a BSc or MSc thesis. At DTU, it is also quite common to organize special elective courses based on student interest in a specific topic, and we have done so based on advanced topics related to our courses several times.

Both of our courses consist of a mix of lectures, live demonstrations of programs and proofs in Isabelle, and exercise sessions. During exercise sessions, students are free to discuss the problems within groups, and teaching assistants are available to provide help and formative evaluation during the sessions. Since many exercise sessions concern systems implemented in Isabelle, students can get immediate feedback on their proofs, and may ask teaching assistants for more detailed feedback and help if this is not sufficient. To aid student independence, we have for some of our systems developed tools which can provide more detailed formative evaluation of student work than Isabelle. Solutions are provided after all exercise sessions so students can compare their own proofs with ours. This is in contrast to the assignments where only feedback is provided.

Both courses additionally have several individual assignments, which we grade and provide feedback on quickly. These assignments count for part of the overall grade of the courses, with the rest of the grade coming from the exam.

4 BSc Course: Logical Systems and Logic Programming

The first of our courses is the BSc course on Logical Systems and Logic Programming. The course is essentially split in two concurrently running parts. One part of the course covers logic programming in Prolog, while the other part concerns formal logic. The course is based primarily on the textbook Mathematical Logic for Computer Science by Ben-Ari [3], and we cover most of the book in the course.

The course learning objectives can be seen in Fig. 1a and the week-by-week plan of the course can be seen in Table 1.

Table 1. Course plan for the course on Logical Systems and Logic Programming.

Week	Main topics	Assignment
1	Tutorial on Logic Programming	
2	Introduction (Prolog Note)	
3	Propositional Logic: Formulas, Models, Tableaux	
4	Propositional Logic: Deductive Systems	X
5	Propositional Logic: Sequent Calculus Verifier—Isabelle	
6	Propositional Logic: Resolution	X
7	First-Order Logic: Formulas, Models, Tableaux	
8	First-Order Logic: Deductive Systems	X
9	First-Order Logic: Sequent Calculus Verifier—Isabelle	
10	First-Order Logic: Terms and Normal Forms	X
11	First-Order Logic: Resolution	
12	First-Order Logic: Logic Programming	
13	First-Order Logic: Undecidability and Model Theory	X

We start by introducing the basic features of Prolog through a number of examples and exercises. We continue to introduce more Prolog features throughout the course, and use Prolog to show how to implement many of the concepts in logical systems.

After the short introduction to Prolog, we begin covering propositional logic. Following Ben-Ari's book, we cover formulas, semantics, models, and semantic tableaux. This also allows us to discuss the issues of soundness and completeness. Next, we cover deductive systems in the styles of Hilbert and Gentzen, and show how to prove completeness by relating systems to existing systems that are known to be complete.

Having done this, we take an excursion into formal methods by introducing the Isabelle proof assistant. We use our Sequent Calculus Verifier (SeCaV) [11, 12, 15], which is implemented in Isabelle/HOL, to teach students how to write and formally verify proofs. This allows students to experiment with their proofs while getting immediate feedback on their correctness. For this first introduction, we use a version of SeCaV which is restricted to propositional logic. Since SeCaV is implemented within Isabelle/HOL, this also exposes students to the basics of proofs in the Isar proof language of Isabelle.

To conclude the sessions on propositional logic we introduce resolution, including Prolog programs that implement each step of a proof by resolution. This allows students to experiment with resolution proofs while also exposing them to non-trivial Prolog programs.

Next, we go through essentially the same topics as before, but now for first-order logic. We again use Prolog programs to explain concepts such as Skolemization and include a Prolog program for resolution in first-order logic.

At this point, we again digress to explore the full version of our Sequent Calculus Verifier, which is a deductive system for first-order logic. The system allows us to explain concepts such as de Bruijn indices and substitution of bound variables with simple implementations. Additionally, we showcase the formal proofs of soundness and completeness for the system. This allows us to explain these proofs in much detail while exposing students to more advanced usage of Isabelle. The implementation of SeCaV in Isabelle/HOL is also a good example for students, since it includes fully elaborated and concrete implementations of e.g., syntax, semantics, and proof rules.

Having done this, we include a number of exercises on implementing logical concepts in Prolog, including the implementation of a SAT solver. We briefly introduce concepts such as higher-order programming and constraint programming in Prolog. We also "close the loop" by finally explaining the relation between logic programming in Prolog and first-order logic. At this point the students have been sufficiently exposed to both to understand this quite quickly.

The final lecture is spent discussing some simple results in model theory and the concept of undecidability.

Throughout the course, students must hand in assignments concerning the various topics of the course. The first assignment is a mix of pen-and-paper formal proofs and Prolog programming exercises, while later assignments also include formal proof exercises in the Sequent Calculus Verifier. These assignments contribute to the final grade of the course. The rest of the grade is determined by a written final exam, which also includes a mix of pen-and-paper formal proofs and Prolog programming exercises.

5 MSc Course: Automated Reasoning

The second of our courses is the MSc course on Automated Reasoning. The course is essentially split in two concurrently running parts. One part of the course covers proving and programming in Isabelle [22,25], while the other part concerns formal logic [10–15,21,36–40]. The course learning objectives can be seen in Fig. 1b and the week-by-week plan of the course can be seen in Table 2.

We start by exploring our formally verified micro provers for propositional logic [37,38], which allow us to explain how provers can be implemented in e.g., Haskell, Isabelle/ML and Standard ML and how to prove correctness in Isabelle.

The Natural Deduction Assistant (NaDeA) [39] is a browser application for classical first-order logic with constants and functions. The syntax, the semantics and the inductive definition of the natural deduction proof system along with the soundness and completeness proofs are verified in Isabelle/HOL. Finished NaDeA proofs are automatically translated into the corresponding Isabelle-embedded proofs.

We have developed teaching materials about Isabelle/Pure [41], showing the generic Isabelle logical framework in order to ensure that students understand

Table 2. Course plan for the course on Automated Reasoning.

Week	Main topics	Assignment
1–2	Prerequisites, micro provers, getting started with Isabelle	X
3–4	Natural Deduction Assistant (NaDeA)	X
5–6	Isabelle/Pure for Intuitionistic and Classical First-Order Logic	X
7–8	Isabelle/Pure for Intuitionistic and Classical Higher-Order Logic	X
9–10	Axiomatic Propositional, First-Order and Higher-Order Logic	X
11–12	Students' Proof Assistant (SPA)	
13	Reserve/buffer lecture	X

what is going on at a lower level when they use the automation of Isabelle/HOL, and the learning outcome is tested in assignments using Isabelle/Pure.

We briefly describe our route from axiomatic propositional logic [7] to first-order logic with equality in our Students' Proof Assistant (SPA) [36] running inside Isabelle/HOL with a formally verified LCF-style prover kernel [31] and declarative proofs [41,42].

The students can experiment in Isabelle/HOL with our formalized soundness and completeness theorems for several axiomatic systems [10], including the following well-known axioms in addition to the rule modus ponens:

Wajsberg 1937
$$p \Rightarrow (q \Rightarrow p)$$
$$(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$$
$$((p \Rightarrow q) \Rightarrow p) \Rightarrow p$$
$$\bot \Rightarrow p$$

Wajsberg 1939
$$p \Rightarrow (q \Rightarrow p)$$
$$(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$$
$$((p \Rightarrow \bot) \Rightarrow \bot) \Rightarrow p$$

Łukasiewicz 1948
$$((p \Rightarrow q) \Rightarrow r) \Rightarrow ((r \Rightarrow p) \Rightarrow (s \Rightarrow p))$$
$$\bot \Rightarrow p$$

We extend the Wajsberg 1939 axiomatic system for propositional logic to first-order logic with equality [20]:

$\vdash q$ if $\vdash p \Rightarrow q$ and $\vdash p$ (modus ponens rule)
$\vdash \forall x.p$ if $\vdash p$ (generalization rule)

$\vdash p \Rightarrow (q \Rightarrow p)$
$\vdash (p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$
$\vdash ((p \Rightarrow \bot) \Rightarrow \bot) \Rightarrow p$
$\vdash (\forall x.p \Rightarrow q) \Rightarrow (\forall x.p) \Rightarrow (\forall x.q)$
$\vdash p \Rightarrow (\forall x.p)$ provided $x \notin \mathrm{FV}(p)$

$$\vdash (\exists x.x = t) \text{ provided } x \notin \text{FVT}(t)$$
$$\vdash t = t$$
$$\vdash s_1 = t_1 \Rightarrow \cdots \Rightarrow (s_n = t_n \Rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n))$$
$$\vdash s_1 = t_1 \Rightarrow \cdots \Rightarrow (s_n = t_n \Rightarrow P(s_1, \ldots, s_n) = P(t_1, \ldots, t_n))$$

$$\vdash (p \Leftrightarrow q) \Rightarrow (p \Rightarrow q)$$
$$\vdash (p \Leftrightarrow q) \Rightarrow (q \Rightarrow p)$$
$$\vdash (p \Rightarrow q) \Rightarrow ((q \Rightarrow p) \Rightarrow (p \Leftrightarrow q))$$
$$\vdash \top \Leftrightarrow (\bot \Rightarrow \bot)$$
$$\vdash \neg p \Leftrightarrow (p \Rightarrow \bot)$$
$$\vdash (p \wedge q) \Leftrightarrow ((p \Rightarrow (q \Rightarrow \bot)) \Rightarrow \bot)$$
$$\vdash (p \vee q) \Leftrightarrow \neg(\neg p \wedge \neg q)$$
$$\vdash (\exists x.p) \Leftrightarrow \neg(\forall x.\neg p)$$

Here FV is the set of free variables in a formula and FVT is the set of free variables in a term. Note that the axiomatic system is substitutionless as it uses equality in a clever way to avoid the complications of substitution [20,36].

Amongst Pelletier's problems [32] for automated reasoning is problem 34, which is also known as Andrews's Challenge. The proof is not obvious at first glance since it relies on the fact that bi-implication is both commutative and associative [36]:

$$((\exists x.\forall y.P(x) \Leftrightarrow P(y)) \Leftrightarrow ((\exists x.Q(x)) \Leftrightarrow (\forall y.Q(y)))) \Leftrightarrow$$
$$((\exists x.\forall y.Q(x) \Leftrightarrow Q(y)) \Leftrightarrow ((\exists x.P(x)) \Leftrightarrow (\forall y.P(y))))$$

Comparing the declarative proofs in Isabelle/HOL and SPA is a good exercise for the students.

In addition to our tools for teaching logic we cover the following online papers:

1. M. Ben-Ari (2020): A Short Introduction to Set Theory [4]
2. W. M. Farmer (2008): The Seven Virtues of Simple Type Theory [6]
3. T. C. Hales (2008): Formal Proof [17]
4. T. Nipkow (2021): Programming and Proving in Isabelle/HOL [25]
5. L. C. Paulson (2018): Computational Logic: Its Origins and Applications[31]

The paper by Farmer provides a concise definition of higher-order logic and the tutorial by Nipkow provides a substantial set of exercises which the students must solve.

6 Discussion and Future Work

As mentioned, our BSc course uses our Sequent Calculus Verifier (SeCaV), which is embedded in Isabelle/HOL, for several exercise sessions and assignments. While the system is designed to be quite simple to use and understand, we have experienced that some students have a hard time writing proofs in the system. Additionally, the embedding in Isabelle/HOL is not able to give very helpful

error messages if a proof is wrong. To alleviate these issues, we have recently developed an online tool called the SeCaV Unshortener [11], which allows users to write proofs in a simpler syntax, which is then automatically translated into the embedding in Isabelle. Additionally, the tool is able to warn users about mistakes in their proofs by explicitly telling users why e.g. a proof rule cannot be applied. Recent research indicates that this kind of feedback impacts learning in computer science significantly, and is sufficient to allow students to move forward in most cases [18].

We also use SeCaV in our MSc course but only as self-study concerning the course prerequisites and selected parts of the papers [11,12,15] in the first weeks of the course.

We would like to integrate even more algorithms and proofs into Isabelle. Work is currently ongoing on an Isabelle implementation and proof of correctness of a tool for converting formulas to conjunctive normal form.

Michaelis and Nipkow [23] formalized a number of proof systems for propositional logic in Isabelle/HOL: resolution, natural deduction, sequent calculus and an axiomatic system. We would like to extend this line of work to first-order logic and higher-order logic.

We find that one of the main issues in both our 2020 and 2021 course on automated reasoning and formally verified functional programming is the course prerequisites. Functional programming is a prerequisite but we do not require a specific language and it is not possible to exclude any students. This is a real problem and in general we need to use the first part of the course to teach some of the prerequisites. Another prerequisite is mathematical logic—syntax, semantics and proof systems—and we use the micro provers to teach logic, functional programming and the basics of a proof assistant, in particular Isabelle, in a way that is challenging to almost all students. It is not for beginners and some students will most likely quit the course in the first month. In 2021, after the first month, 37 students were active and almost everyone submitted the first assignment. We have no clear solution to the issues concerning the course prerequisites but for 2022 we plan to offer a series of online sessions for self-study in mathematical logic and functional programming.

7 Conclusion

We have presented our detailed experiences teaching two courses on formal methods. The first course is the bachelor course on logical systems and logic programming, which has a focus on classical first-order logic and automatic theorem proving. We have additionally described how we use Prolog and Isabelle to introduce students to logic and formal methods.

The second course is the master course on automated reasoning, which has a focus on classical higher-order logic and interactive theorem proving. The proof assistant Isabelle is used more heavily in this course, and we use Isabelle/Pure as well as Isabelle/HOL. We have also described our online tools to be used with Isabelle/HOL, in particular the Sequent Calculus Verifier (SeCaV) and

the Natural Deduction Assistant (NaDeA). In addition, we have described our innovative Students' Proof Assistant which is formally verified in Isabelle/HOL and integrated in Isabelle/jEdit using Isabelle/ML.

We have described how our courses fit into the overall computer science and engineering curriculum, and what issues and challenges we experience that students often face. We have suggested some future work on the courses by which we hope to improve student learning outcomes.

Our teaching philosophy is related to the IsaFoL (Isabelle Formalization of Logic) project [5] which aims at developing formalizations in Isabelle/HOL of logics, proof systems, and automatic/interactive provers. Notable work in the same line includes the soundness and completeness of epistemic [8] and hybrid [9] logic and an ordered resolution prover for first-order logic [35]. These formalizations can serve as starting point for a student project to formalize the soundness and completeness of various other proof systems and provers.

We would like to formalize even more topics within basic logic such that students can explore concrete and executable definitions of various topics such as Skolemization while also seeing formal proofs of their correctness. Our overall conclusion is that using formal methods, in particular the proof assistant Isabelle, as a central tool for teaching logic and formal methods is possible as we have demonstrated since our first use of the Natural Deduction Assistant (NaDeA) and the Sequent Calculus Verifier (SeCaV) in 2014 and 2019, respectively.

Acknowledgements. We thank Asta Halkjær From for comments on drafts. We thank the three anonymous reviewers whose comments and suggestions helped improve the paper.

References

1. Baanen, A., Bentkamp, A., Blanchette, J., Limperg, J., Hölzl, J.: The Hitchhiker's Guide to Logical Verification (2020). https://github.com/blanchette/logical_verification_2020
2. Bella, G.: You already used formal methods but did not know it. In: Dongol, B., Petre, L., Smith, G. (eds.) FMTea 2019. LNCS, vol. 11758, pp. 228–243. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32441-4_15
3. Ben-Ari, M.: Mathematical Logic for Computer Science. Springer, London (2012)
4. Ben-Ari, M.: A Short Introduction to Set Theory (2020). https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/books/set.pdf
5. Blanchette, J.C.: Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In: Mahboubi, A., Myreen, M.O. (eds.) Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, 14–15 January 2019, pp. 1–13. ACM (2019)
6. Farmer, W.M.: The seven virtues of simple type theory. J. Appl. Log. **6**(3), 267–286 (2008). https://doi.org/10.1016/j.jal.2007.11.001
7. From, A.H.: Formalizing Henkin-style completeness of an axiomatic system for propositional logic. In: Proceedings of the Web Summer School in Logic, Language and Information (WeSSLLI) and the European Summer School in Logic, Language

and Information (ESSLLI) Virtual Student Session, pp. 1–12 (2020). Preliminary paper, accepted for Springer post-proceedings
8. From, A.H.: Epistemic logic: completeness of modal logics. Archive of Formal Proofs, October 2018. https://devel.isa-afp.org/entries/Epistemic_Logic.html, Formal proof development
9. From, A.H.: Formalizing a Seligman-style tableau system for hybrid logic. Archive of Formal Proofs, December 2019. https://isa-afp.org/entries/Hybrid_Logic.html, Formal proof development
10. From, A.H., Eschen, A.M., Villadsen, J.: Formalizing axiomatic systems for propositional logic in Isabelle/HOL. In: Kamareddine, F., Sacerdoti Coen, C. (eds.) CICM 2021. LNCS (LNAI), vol. 12833, pp. 32–46. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81097-9_3
11. From, A.H., Jacobsen, F.K., Villadsen, J.: SeCaV: a sequent calculus verifier in Isabelle/HOL. In: 16th International Workshop on Logical and Semantic Frameworks with Applications (LSFA 2021) – Presentation Only/Online Papers, pp. 1–16 (2021). https://mat.unb.br/lsfa2021/pages/lsfa2021_proceedings/LSFA_2021_paper_5.pdf
12. From, A.H., Jensen, A.B., Schlichtkrull, A., Villadsen, J.: Teaching a formalized logical calculus. Electron. Proc. Theor. Comput. Sci. **313**, 73–92 (2020). https://doi.org/10.4204/EPTCS.313.5
13. From, A.H., Lund, S.T., Villadsen, J.: A case study in computer-assisted meta-reasoning. In: González, S.R., Machado, J.M., González-Briones, A., Wikarek, J., Loukanova, R., Katranas, G., Casado-Vara, R. (eds.) DCAI 2021. LNNS, vol. 332, pp. 53–63. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-86887-1_5
14. From, A.H., Villadsen, J.: Teaching automated reasoning and formally verified functional programming in Agda and Isabelle/HOL. In: 10th International Workshop on Trends in Functional Programming in Education (TFPIE 2021) – Presentation Only/Online Papers, pp. 1–20 (2021). https://wiki.tfpie.science.ru.nl/TFPIE2021
15. From, A.H., Villadsen, J., Blackburn, P.: Isabelle/HOL as a meta-language for teaching logic. Electron. Proc. Theor. Comput. Sci. **328**, 18–34 (2020). https://doi.org/10.4204/eptcs.328.2
16. Grover, S.: Toward a framework for formative assessment of conceptual learning in K-12 computer science classrooms. In: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE 2021, pp. 31–37 (2021). https://doi.org/10.1145/3408877.3432460
17. Hales, T.C.: Formal proof. Not. Am. Math. Soc. **55**, 1370–1380 (2008)
18. Hao, Q., et al.: Towards understanding the effective design of automated formative feedback for programming assignments. Comput. Sci. Educ. 1–23 (2021). https://doi.org/10.1080/08993408.2020.1860408
19. Harrison, J.: Formalizing basic first order model theory. In: Grundy, J., Newey, M. (eds.) TPHOLs 1998. LNCS, vol. 1479, pp. 153–170. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055135
20. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, Cambridge (2009)
21. Jensen, A.B., Larsen, J.B., Schlichtkrull, A., Villadsen, J.: Programming and verifying a declarative first-order prover in Isabelle/HOL. AI Commun. **31**(3), 281–299 (2018)
22. Krauss, A.: Defining Recursive Functions in Isabelle/HOL (2021). https://isabelle.in.tum.de/doc/functions.pdf

23. Michaelis, J., Nipkow, T.: Formalized proof systems for propositional logic. In: Abel, A., Forsberg, F.N., Kaposi, A. (eds.) 23rd International Conference on Types for Proofs and Programs, TYPES 2017, Budapest, Hungary, 29 May–1 June 2017. LIPIcs, vol. 104, pp. 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
24. Nipkow, T.: Teaching semantics with a proof assistant: no more LSD trip proofs. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 24–38. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_3
25. Nipkow, T.: Programming and Proving in Isabelle/HOL (Tutorial) (2021). https://isabelle.in.tum.de/doc/prog-prove.pdf
26. Nipkow, T.: Teaching algorithms and data structures with a proof assistant (invited talk). In: Hritcu, C., Popescu, A. (eds.) 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, CPP 2021, Denmark, 17–19 January 2021, pp. 1–3. ACM (2021). https://doi.org/10.1145/3437992.3439910
27. Nipkow, T., et al.: Functional Algorithms, Verified! (2021). https://functional-algorithms-verified.org/
28. Nipkow, T., Eberl, M., Haslbeck, M.P.L.: Verified textbook algorithms. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 25–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_2
29. Nipkow, T., Klein, G.: Concrete Semantics - With Isabelle/HOL. Springer, Heidelberg (2014)
30. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
31. Paulson, L.C.: Computational logic: its origins and applications. Proc. R. Soc. A. **474**(2210), 20170872 (2018). https://doi.org/10.1098/rspa.2017.0872
32. Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs, September 2016. http://isa-afp.org/entries/SuperCalc.shtml, Formal proof development
33. Pierce, B.C., et al.: Software Foundations – 6 Online Textbooks (2021). https://softwarefoundations.cis.upenn.edu/
34. Reis, G.: Facilitating meta-theory reasoning (invited paper). In: Pimentel, E., Tassi, E. (eds.) Proceedings Sixteenth Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, Pittsburgh, USA, 16th July 2021. Electronic Proceedings in Theoretical Computer Science, vol. 337, pp. 1–12. Open Publishing Association (2021). https://doi.org/10.4204/EPTCS.337.1
35. Schlichtkrull, A., Blanchette, J., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. J. Autom. Reason. **64**(7), 1169–1195 (2020)
36. Schlichtkrull, A., Villadsen, J., From, A.H.: Students' Proof Assistant (SPA). In: Quaresma, P., Neuper, W. (eds.) Proceedings 7th International Workshop on Theorem Proving Components for Educational Software, ThEdu@FLoC 2018, Oxford, United Kingdom, 18 July 2018. Electronic Proceedings in Theoretical Computer Science, vol. 290, pp. 1–13. Open Publishing Association (2018). https://doi.org/10.4204/EPTCS.290.1
37. Villadsen, J.: A micro prover for teaching automated reasoning. In: Seventh Workshop on Practical Aspects of Automated Reasoning (PAAR 2020) – Presentation Only/Online Papers, pp. 1–12 (2020). https://www.eprover.org/EVENTS/PAAR-2020.html

38. Villadsen, J.: Tautology checkers in Isabelle and Haskell. In: Calimeri, F., Perri, S., Zumpano, E. (eds.) Proceedings of the 35th Edition of the Italian Conference on Computational Logic (CILC 2020), Rende, Italy, 13–15 October 2020. CEUR Workshop Proceedings, vol. 2710, pp. 327–341. CEUR-WS.org (2020). http://ceur-ws.org/Vol-2710/paper-21.pdf
39. Villadsen, J., From, A.H., Schlichtkrull, A.: Natural Deduction Assistant (NaDeA). In: Quaresma, P., Neuper, W. (eds.) Proceedings 7th International Workshop on Theorem Proving Components for Educational Software, THedu@FLoC 2018, Oxford, United Kingdom, 18 July 2018. EPTCS, vol. 290, pp. 14–29 (2018). https://doi.org/10.4204/EPTCS.290.2
40. Villadsen, J., Schlichtkrull, A., From, A.H.: A verified simple prover for first-order logic. In: Konev, B., Urban, J., Rümmer, P. (eds.) Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning (PAAR 2018) co-located with Federated Logic Conference 2018 (FLoC 2018), Oxford, UK, 19 July 2018. CEUR Workshop Proceedings, vol. 2162, pp. 88–104. CEUR-WS.org (2018). http://ceur-ws.org/Vol-2162/paper-08.pdf
41. Wenzel, M.: The Isabelle/Isar Reference Manual (2021). https://isabelle.in.tum.de/doc/isar-ref.pdf
42. Wenzel, M.: Isar—a generic interpretative approach to readable formal proof documents. In: Bertot, Y., Dowek, G., Théry, L., Hirschowitz, A., Paulin, C. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 167–183. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48256-3_12