Theorem Provers as a Learning Tool in Theory of Computation

Maria Knobelsdorf, Christiane Frede
Universität Hamburg
Fachbereich Informatik (Department of Informatics)
Vogt-Kölln-Straße 30
22527 Hamburg, Germany
{knobelsdorf, frede}@informatik.uni-hamburg.de

Sebastian Böhne, Christoph Kreitz
Universität Potsdam
Fachbereich Informatik (Department of Informatics)
August-Bebel-Straße 89
14882 Potsdam, Germany
{boehne, kreitz}@uni-potsdam.de

ABSTRACT

This paper presents first results of an evaluation study investigating whether an interactive theorem prover like Coq can be used to help undergraduate computer science (CS) students learn mathematical proving within the field of theory of computation. Set within an educational design research approach and building on cognitive apprenticeship and socio cultures cognition theories, we have collected empirical, mainly qualitative observational data focusing on students' activities with Coq in an introductory course specifically created for that matter. Our results strengthen the assumption that a theorem prover like Coq, indeed, can be beneficial in mediating undergraduate students' activities in learning formal proving. In comparison to pen & paper proofs, students were profiting strongly from the system's immediate feedback and scaffolding. These results encourage the idea to extend the scientifically dominated use of theorem provers like Coq to pedagogical use cases in undergraduate CS education.

CS CONCEPTS

• **Social and professional topics~Computer science education** • **Applied computing~Interactive learning environments** • Applied computing

KEYWORDS

Computer science education; theorem prover; students; theory of computation; distributed cognition theory; qualitative research; observational study; proofs; logic; data structures.

1 INTRODUCTION

The theory of computation is an important field of undergraduate computer science (CS) education ([30], p. 55-60) and in consequence, CS majors are required to take introductory courses covering topics like automata theory, computability, or complexity [55]. The corresponding concepts, theories, and algorithms are strongly mathematical in nature, both because of the formalized descriptions used for the discourse and because of a strong focus on mathematical proofs for the presented theories and approaches. By introducing idealized mathematical models of the computer and discussing methods for designing and analyzing them, students are supposed to develop the ability of thinking abstractly about computational processes and models. Introductory courses to theory of computation traditionally are organized around weekly lectures that present the course topics using blackboard or slides. The lectures are accompanied by weekly homework assignments based on the current lecture topics, which students are expected to solve individually or in small groups and submit in writing for reviewing and grading by tutors. Tutorial and supervised exercise sessions usually round off this approach, which is also very common in introductory undergraduate courses to mathematics or physics, see also [41]. [36]. There are obviously computing departments and teachers that work with other pedagogies, but in many universities all over the world, especially in Germany, this is the predominant approach.

Many undergraduate CS students have difficulties in theory courses with considerable dropout and poor performance in final exams, see for example [11][25][47]. So far suggested alternative pedagogical approaches and tools in this field address mainly a lack of interest and engagement as well as inabilities to understand theoretical concepts, both potential reasons assumed causing students failure in theory of computation [12][23][54] [48][20][27]. Empirical insights on student difficulties with theory of computation are rare. But recent studies including our own investigations, indicate that students rather seem to have specific difficulties with mastering mathematical language and creating formal proofs in corresponding assignments of theory of computation [35][43][47][33]. Also in mathematics education, students' difficulties with formal proofs and reasoning have been investigated outlining the complexity this domain-specific proficiency has in mathematics education [1][4][60]. We argue that within the domain of theory of computation, educational designs and pedagogical approaches are required that focus specifically on guiding and supporting students how to create formal proofs as well as formally reason within a proof. For that reason, we have started investigating whether and how an

ICER '17, August 18-20, 2017, Tacoma, WA, USA

ACM ISBN 978-1-4503-4968-0/17/08…$15.00
http://dx.doi.org/10.1145/3105726.3106184

interactive theorem prover like Coq [7] might be a suitable learning tool for that purpose. Theorem provers support creating and checking formal proofs while operating on an explicit and high level of formalization. We belief that this makes them suitable to scaffold students' learning and investigate relevant questions related to formal proving practices and how students develop domain-specific competencies [49] within theory of computation.

Set within an educational design research [38] approach, we have collected empirical, mainly qualitative observational data focusing on students' activities with Coq in an introductory course to logic and data structures specifically created for that matter. Educational design research (which also goes by the name design-based research) has its roots in design research, which focuses on designing and investigating instructional interventions and learning environments in the settings for which they are intended, with the dual purpose of developing and refining theories of learning as well as testing and revising the design itself. A central feature of design research is a reliance on iterations of intervention, analysis, and redesign in four interrelated phases to allow researchers to test, revise, and refine conjectures. This paper, then, is neither representing just an empirical evaluation study, nor just an experience report. It is explicitly an educational design research paper, with the purpose of presenting a first iteration of theoretical considerations combined with empirical insights most relevant to the development of a student-oriented pedagogy in the field of theory of computation.

The remainder of this paper is structured as follows: In the next section, we will start with a brief overview of the theoretical background our pedagogical considerations and research focus are based on. In section 3, we will then introduce the theorem prover Coq and discuss its potential for being turned into a learning tool while comparing it with related work. In section 4, we will present research questions that framed this first iteration of educational design research and continue with describing the course design and methods used for its evaluation. The results of this evaluation in section 5 contain a category system capturing students' activities with Coq observed during the course, additional analysis and observations, and a discussion with further interpretations. The paper concludes in section 5.

2 THEORETICAL BACKGROUND

We build on an understanding of teaching and learning shaped by socio-cultural cognition theories [58] focusing mainly on cognitive apprenticeship [13][14] and distributed cognition theory [28][44][32]. These approaches strongly relate to Vygotskian developmental psychology [59] understanding learning activities of students to be embedded within a socio-cultural environment and mediated by material and representational systems (e.g., symbols, inscriptions, visuals), which are often metaphorically referred to as tools or mediational means [61]. Cognitive processes are regarded to be, not just individual and mental, but also situated within and constituted through a specific community of practice [13][58] and a related cognitive system as the interplay between one or several human beings, their activities, and tools used in these activities [28][29]. For educational research this has the important implication that we have to consider that the unit that we are studying just as the pedagogy that we are creating is considering "people in action using tools of some kind" ([51], p. 147) embedded within a community of practice.

Speaking of theory of computation as a domain-specific community of academic research practice [34], its culture and tools have strong roots in mathematics [57]. Here, tools used specifically by members of this community involve mathematical formal inscriptions, i.e., symbols, visuals and notations created with pen & paper and on black-/whiteboard, designed to conceive theoretical objects and to mediate mental and collaborative processes of human agents in writing and in speech that lead to these objects. In studying theory of computation, undergraduate CS students have to master using these tools when working on their weekly assignments. This tool mastering can be regarded as an enculturation process within the community of practice of theory of computation and it is strongly shaped by the educational and pedagogical design used in theory of computation courses (see also section 1). For that matter, students can observe the teacher using the tools during lecture (model-based learning) and also learn from textbooks and by interacting through verbal speech and writing with their peer students and tutors. As has been investigated and outlined before, it is extremely challenging for novice students to uncover domain-specific competencies on their own when the pedagogical approach of a course is solely focusing on presenting factual knowledge [14][53] and without sufficient distinction between knowledge as the item of inquiry and competencies that are needed to handle it ([6],p. 295ff).

Speaking of educational and pedagogical approaches that address students' domain-specific competency development, *cognitive apprenticeship* has been suggested with concepts called *modelling*, *coaching*, *scaffolding*, or *fading* [13][14]. These techniques in many ways are based on the concept of *zone of proximal development (ZPD)* referring to a student's potential of mastering new tasks (i.e. developing a new competency) with assistance and guidance or in collaboration with external agents [59][62][45]. Based on the insights we have about students difficulties with theory of computation assignments [35][43], we hypothesize that pen & paper as a mediational mean and learning tool embodies not enough knowledge to scaffold students' ZPD and will always require additional scaffolding from tutors, textbooks, or other sources. Theorem prover on the other hand embody knowledge of formal proofs. For that matter, we have started investigating whether and how an interactive theorem prover like Coq [7] might be turned into a suitable tool that is scaffolding students learning within the discussed context and in the next section, we will introduce Coq and extend on this argument.

3 THEOREM PROVERS IN EDUCATION

Theorem provers have been developed since the late 1960's, when Nicolas de Bruijn pioneered the field with his Automath proof checker. Automath's modern descendants – systems such as Coq, Nuprl, Isabelle, HOL, or Agda [7][2][15][42][9]– have evolved to a state that they are increasingly used for formalization and verification tasks in frontier applications in mathematics and

computer science, see for example [24][21]. Most existing typical proofs created for machine verification not only require users to have a deep understanding of the underlying logic and of the system that was used for the formalization but also have little resemblance to proofs presented in textbooks and university lectures. Therefore, when introduced in education, theorem provers are generally used in advanced coursework in mathematics and theory of computation for the same purpose addressing graduate students with a profound background in mathematics and functional programming [3][15][26]. Only a few, though promising examples of using theorem proving in introductory courses can be found in the literature. One line of this related work focuses on teaching students how to use proof automation and the relevant programming, but is not teaching formal proving as such [50][46]. The other line of work is using theorem provers as underlying system for providing (intelligent) tutoring systems that support the development of less detailed proofs [5][22][52]. Systems like the EPGY Theorem Proving Environment [56] allows step-wise proof development similar to pen and paper proofs. But, the approach is not based on a theorem prover with an expressive formal language and is thus difficult to extend to more complex fields in theory of computation like automata theory. In [22], the theorem prover Coq has been extended by a visualization tool in order to teach high-school students geometry. While this approach is based on the same theorem prover as ours, it focuses on a bottom-up construction of proofs, which is better for the presentation of proofs but makes it more difficult to find them. Also here, an extension to more complex fields of theory of computation seems not possible.

In the next subsections, we will discuss how the theorem prover Coq can be used as a learning tool, starting with a short introduction of Coq.

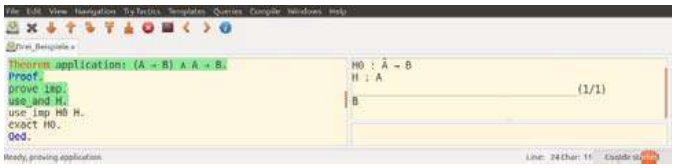

Figure 1: The Coq IDE, showing in the left-hand window a proof as sequence of applied proof steps. Steps already evaluated are marked green. The current result of executing a step can be seen in the right-hand window, while error-messages are displayed in the lower right-hand window.

3.1 The theorem prover Coq

The theorem prover Coq has been developed in the 1980's by researchers at the French "Institut national de recherche en informatique et en automatique (INRIA)" [7]. The system allows formalizing mathematical assertions and developing related formal proofs that are evaluated by the system. Like many theorem provers, Coq implements a higher-order type theory and thus theorems in Coq are understood as types and the proofs for theorems as elements of the respective type. Almost all theorems include implications and/or universal quantifiers. Underlying proofs of such theorems are functions (typed λ-calculus) so that such proof assistants are closely interwoven to functional programming.

Coq offers an IDE comparable to a typical IDE for programming (see Figure 1). In the main left-hand window, a user can generate a proof step-by-step, applying so-called *tactics* similar to invocating commands. A tactic is comparable to a procedure or subroutine in a program written in a functional programming language. Being called in a proof, a tactic splits an assertion into sub-assertions following "backward reasoning" (from conclusion to premises). In the evaluation, every step of the proof is evaluated for its formal correctness comparable to compiling and running a program code. In the Coq IDE, the user can manually step through the proof and see every sub-goal of the proof displayed in the right-hand window, indicating the next step in the proof. This way, Coq provides direct feedback through the entire process of creating a proof, especially which steps the user still has to perform before completing and whether a proof step is successfully applicable. Thus, creating proofs with Coq has many similarities to write and debug computer programs.

3.2 Turning Coq into a learning tool

Coq has been developed with the purpose to support experts in formalization and verification tasks. Using the full capabilities of Coq requires proficiency in type theory and functional programming. Also in terms of usability, the Coq IDE was not designed for novice students' needs. We will give three examples that emphasize potential issues for novice students: 1) While the proof is built through backward reasoning (see previous section), writing the proof down happens from top to bottom of the left-hand window which intuitively might be understood by students as forward reasoning and confuse them. 2) The right-hand window shows only the subgoals created last but not previously accomplished steps, which might cause students to get lost. 3) Predefined tactics for a proof are not displayed anywhere in the IDE (comparable to block elements in Scratch) and students have to remember that they exist.

Nevertheless, Coq has several advantages that make it a potential learning tool. Because of its interactive nature it provides immediate, individual feedback especially if a step is correct and whether a proof is finished or requires additional steps and can therefore be regarded as formative assessment monitoring students learning. This advantage strongly contrasts the common practice, where students create proofs with pen & paper and receive feedback from their tutors usually one week later, which is summative assessment. In addition, working with Coq and creating proofs in a programming-debugging sort of manner certainly represents activities for CS students that are more close to the computing culture and therefore could be additionally motivating and engaging.

Another advantage of Coq is its explicit and high level of formalization, which suits particularly well to scaffold students' learning. This sounds paradoxical at first, since students already struggle with the formalisms in pen and paper proofs. Developing a proof is creating arguments in a logical sequence of steps using deductive reasoning. However, there are different degrees of formalization and precision and we believe that this is not made

explicit and clear enough for students. Building on this argument, we hypothesize that students' problems are not so much related with the formalistic elements themselves but the mixture between formal and informal aspects. Coq, by contrast, explicitly defines one level of formalization providing orientation within the chain of formal reasoning.

In order to be used by students without any prerequisites in functional programming and type theory, we suggest using an *information hiding* principle. Within this pedagogically motivated approach, students are provided with a proof assignment to be created in Coq and corresponding predefined tactics that function as building blocks of the proof (the idea roughly corresponds to BlueJ and Scratch where specific elements of programming languages are hidden and visual representations of them are used instead). As an example, consider the proof of the following statement from logic: $(A \longrightarrow B) \wedge A \longrightarrow B$ in Coq. For proving the implication in Coq (see also Figure 1 above), $(A \longrightarrow B) \wedge A$ has to be assumed and it has to be shown that B follows. This is realized by a backward reasoning tactic that we call *prove_imp*. The conjunction is then decomposed by another predefined tactic called *use_and*. An implication like $A \longrightarrow B$ can be used if we know its premise, too. Since this is the case here we can apply *use_imp* to both hypotheses to obtain B, with which we can conclude the proof.'. With these predefined tactics, students can focus on the reasoning without having to understand type theory, implementation of tactics, or other internals of Coq.

In the long run, we intend to create assignments with corresponding tactics that cover what is regarded to be the basic topics from theory of computation (e.g. automata theory, Turing machines). The latter is obviously not trivial as it requires formalizing the corresponding domain and developing tactics that can serve as adequate building blocks for novices to be used in their proofs. Hence, this is a long-term objective, which we start investigating by creating and evaluating a first educational design, which will be presented in more detail in the next section.

4 COURSE SETUP AND EVALUATION

In the previous section, we have outlined a potential approach how Coq could be modified in order to act as a learning tool to help novice students learning to create formal proofs within theory of computation. In a first step, we were interested whether undergraduate students with no prerequisites in functional programming and type theory are able to master creating formal proofs with Coq if being provided with assignments and predefined tactics as building blocks as suggested in Sec. 3.2. In particular, we focused on the following **research questions**:

1. What kind of problems and issues do students run into when working with Coq, especially usability issues?
2. Do students enjoy working with Coq leading to more satisfaction in creating formal proofs than using pen & paper?
3. Is Coq significantly better supporting students in creating formal proofs due to its scaffolding quality in comparison to pen & paper proof assignments?

In order to start investigating these research questions, we have created a two weeks, full time, elective course, which took place in early October 2016 at Universität Hamburg, Germany, addressing CS undergraduate students in their second or higher year as a major. As course content, we chose propositional logic for the first three days and predicate logic for the next two days of the course, as well as data structures (i.e., lists, natural numbers, and binary trees), which covered the last five days of the course. Also, by focusing on logic and rather simple data structures, creating predefined tactics that serve as building blocks for student proofs required far less time than, e.g., formalizing automata theory. In the next subsections, we introduce the course setup and its implementation in more detail and describe what kind of data we collected during the course and how we analyzed it afterwards.

4.1 Course Design and Participants

Following our theoretical considerations, the pedagogy of our course was implementing the cognitive apprenticeship approach [14] focusing on scaffolding and fading [45]. However, we decided to keep also the traditional framework of lecture & assignments because we intend our approach to be used in future within this traditional teaching setting. Here, we relied on experiences of previously implementing cognitive apprenticeship within the traditional framework of an introductory course to theory of computation [36].

We used lecture elements in order to introduce the course content and demonstrate relevant competencies. Co-author Böhne of this paper gave all lectures during the entire course using slides and a blackboard. Aligned with the lecture content, students received assignments, which they were supposed to work on individually or in small groups. These sessions took place in a seminar room and in a computer lab and were supervised by co-author Böhne and an additional tutor (a graduate CS student). They both walked around constantly through the room, answering questions and offering help. At the end of each exercise session, they also presented the most significant questions and key points of the assignments that had occurred before in individual discussions with students. Each day started with a one-hour lecture and was followed by a two-hour exercise session with assignments to be worked on. After a lunch break, afternoon started with another one-hour lecture and was again followed by a two-hour session. As students required more time than expected to work on their assignments, we replaced three lecture sessions by additional exercise time. The fifth and tenth day were the last days for the appropriate topic and not introducing any new content but providing additional time for unsolved assignments from the days before.

For every day, an assignment sheet was offered with approx. 15-20 assignments requiring between 5-20 minutes each. In sum, we offered 221 assignments (70 covering propositional logic, 48 predicate logic and 103 data structures). Among these 221 assignments, 51 assignments were not about proving but about understanding logic (24) and data structures (10), getting familiar with Coq without creating proofs (4) and defining functions in

Coq without proving (13). The remaining 170 assignments covered proofs that were supposed to be created by using:

- only pen and paper (39 first week, 1 second week)
- first pen and paper than the same assignment with Coq (18 first and 1 second week)
- only Coq (23 first week, 45 second week)
- first Coq directly and then using Coq's text editor for "pen & paper"-alike proofs with natural language and math symbols (18 first week, 25 second week).

This mixture of proof assignments (going back and forth between pen & paper and Coq) was supposed to provide us with insights relevant for our research questions, especially how students perform in working on the same assignment while using different tools. In addition, during the first week students were also asked to comment most of their Coq proofs within their "proof code" just like they are used to do with programming. This step was inserted because of the problems appearing while the students should go back from Coq to pen and paper. Two weeks after the end of the course, we offered a final exam that students could attend in order to get credit points for the course.

Regarding drop out, 21 undergraduate CS majors registered for the course and on average, 16 (11 male and 5 female students) attended all morning and afternoon sessions of the course. Most of these students just finished their first or second year as CS major. Almost all students had attended an introductory course of mathematics and theory of computation (including propositional and predicate logic) in their first year and passed it with grades between B and C. In addition, all students attended introductory programming courses in Java in their first year. Being asked in a survey at the beginning of the course, students reported to have taken our course because they were interested to use Coq and wanted to improve their ability of creating formal proofs and be better prepared for further courses in theory of computation.

4.2 Data Collection and Analysis

Within the entire course, we conducted three surveys during lecture time using pre-defined measuring instruments as well as self-defined items. The first survey focused on students' prior knowledge and motivation (gathered on first day), students' self-perceived first impressions of Coq usability and working with Coq (third day), as well as a final feedback on working with Coq, the course concept, and self-perceived knowledge gain (last day). For space reasons, we will not report on these data further but wanted to mention its existence for conclusive reasons.

Because we were interested in how students worked with Coq in a regular classroom environment and what kind of problems and challenges they faced, we approached the research field following the *Natural Inquiry* paradigm ([37], p. 36ff) collecting observational data during all assignment sessions, which took place in the computer lab. Here, we focused on collecting students' questions and problems they faced during their individual working sessions with the assignments. Because most of the students were working in groups of two or three (13 out of 16), we had natural think-aloud sessions that could easily be observed without interfering with students' activities. Especially, we focused on observing students discussing intensively a problem or asking for help and capturing these observations by taking notes. Within this first iteration, we wanted to get a general broad picture of the working progress of all students instead of explicitly observing one group intensively, e.g. using videos. Because of the group size and the majority of students working in groups, it was possible to capture most discussions and create first insights into how students manage to work with Coq. Obviously, our data is not representing exactly 100% of all problems that students ran into and only those that were discussed aloud.

Creating notes, we started with writing down students' discussions while observing them. After capturing several such discussions, it was possible to create summarizing first codes to capture a situation comparable to open coding. Here, we distinguished between problems students had solved on their own and problems where the tutor had to help. In addition, we noticed that students were discussing different problems that we organized around first forms of categories. Moreover, at the end of every day, everyone present during these sessions exchanged their individual observations they had made during the day. All these written notes and the first codes capturing student problems were further coded with qualitative content analysis [39] and the coding software MaxQDA using a summarizing technique. Because the notes were already summarized during the exercise sessions, a student discussion of a problem was defined as the smallest data unit. While using MaxQDA, new categories appeared or were changed, resulting in a category system, which is presented next.

5 RESULTS

As described in the previous subsection, our qualitative data consists of directly observed situations in which students were dealing with problems. In the data collection process, we distinguished each observed situation in which students were discussing a problem: there were problems students were able to solve alone or with the help of other students, (we will call these problems *S-problems*) and problems that students were not able to solve alone or with the help of other students. In that case, the teacher and tutor were called for help (we will call this *T-problems*). Because this definition is built on the way a problem was solved in the end, situations where students had a long discussion about a current problem but asked for help in the end will be considered as a T-problem instead of S-problem. We make this particular distinction between S- and T-problems because it provides insight to which extent Coq reduced T-problems due to its scaffolding qualities.

5.1 Category System

Coding and interpreting all observed student activities, we derived a category system that will be presented next. Each category describes a specific kind of problem students ran into when working on their assignments and each problem of a category was distinguished as S- or T-problem. However, the presented categories only covers problems that were related to the course content and pedagogy and especially to the research

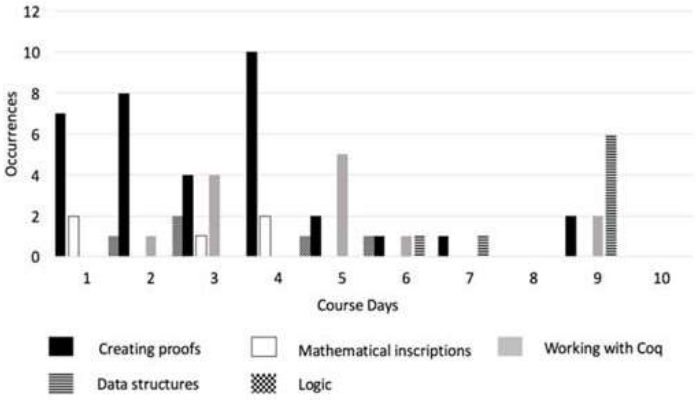

Figure 3: Occurrences of T-problems

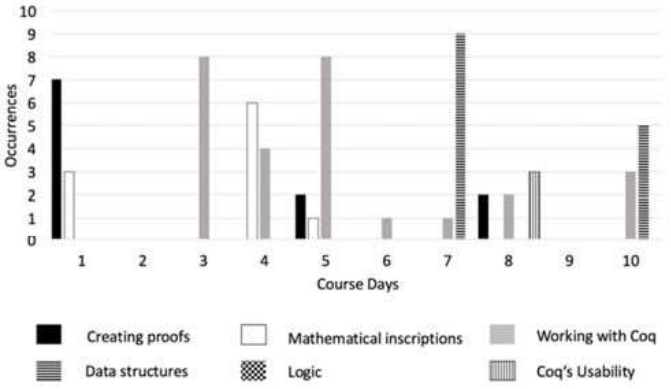

Figure 2: Occurrences of S-problems

questions. We also observed and coded technical issues regarding hardware infrastructure, e.g. login problems, or system failures with Coq that were caused by the predefined tactics or the Coq version used. As these problems were not relevant for our research questions, we will not consider them further here

Coding and interpreting all observed student activities, we derived a category system that will be presented next. Each category describes a specific kind of problem students ran into when working on their assignments and each problem of a category was distinguished as S- or T-problem. However, the presented categories only covers problems that were related to the course content and pedagogy and especially to the research questions. We also observed and coded technical issues regarding hardware infrastructure, e.g. login problems, or system failures with Coq that were caused by the predefined tactics or the Coq version used. As these problems were not relevant for our research questions, we will not consider them further here.

Creating proofs

Problems students had with creating a proof. These included recognizing and understanding assumptions as well as how to start a proof and decide whether it is complete and correct.

Mathematical inscriptions

Problems students faced with mathematical inscriptions especially how to write a step or argument down in a formal way.

Coq's Usability

Problems students had with the Coq IDE (e.g. using menu functions, understanding meaning of displayed elements).

Working with Coq

Problems students faced with Coq (e.g. choosing the correct tactic or variables) including typing errors while copy-pasting an assignment from the sheet into the IDE.

Data Structures

Problems occurring about data structures like understanding the concept of a list or a tree.

Logic

Problems students faced with the aspects of logic like syntax and semantics of definitions or using quantifiers.

In the following closer examination of these categories, we will provide quantitative information about these categories' occurrences over the days with further specification of the single problems. As mentioned in section 4, the course content covered during lectures and assignments was organized as follows:

- Day 1-3: propositional logic
- Day 3-5: predicate logic
- Day 6-10: data structures

5.2 T-Problems

In total, we observed 66 situations in which students were asking the teacher or tutor for help (51 in the first week and 15 in the second week). Relating these numbers to the categories (see also Figure 3), the T-problems occurred as follows:

- Creating proofs (35 times)
- Mathematical inscriptions (5 times)
- Coq's usability (0 times)
- Working with Coq (13 times)
- Data structures (8 times)
- Logic (5 times)

Creating proofs: During the first two days, when working with pen and paper, the students had problems recognizing the assumption and whether a proof was complete or whether there were any missing steps (e.g. "How many steps do I need?", "How do I proceed next?", "Have I shown everything?"). Another frequent problem students had was to decide if their proof was complete. In addition, they were confusing premises with conclusions and frequently asked what was given and what they were supposed to prove. After starting to work with Coq in the afternoon of day two, these problems stayed mostly the same, but their number was decreasing. In the afternoon of day three, the course topic changed from propositional logic to predicate logic and the students were ought to create pen & paper proofs again. The occurring high number of problems were mostly the same as during the first two days during pen & paper proofs from propositional logics. After returning to Coq in the afternoon of day four, T-problems decreased.

Mathematical inscriptions: T-problems regarding mathematical inscriptions occurred in just few cases during week

one and were related to proof assignments with pen and paper and focusing on questions about formal correctness and the meaning of symbols and definitions.

Working with Coq: After introducing Coq on day two and three, the T-problems of working with Coq during the first week were about using variables and universal quantifiers in the correct way or loading data into Coq as well as whether they had to write every single step down. Students also made typing errors frequently and transferred the assignment incorrectly from the assignment sheet to Coq. In the second week, the only T-problems that occurred while working with Coq were few cases on day nine about using specific tactics.

Coq's usability: There were no T-problems regarding Coq's usability.

Data structures: About 50% of all T-problems occurring in the second week were problems with the content-related parts of the course, i.e. data structures. More precisely, the students had problems with understanding and working with binary-trees and recursion.

Logic: The T-problems regarding logic were about quantifiers and syntax but mostly about basic comprehension regarding predicate logic.

Further remarks: On day eight and ten no T-problems of the regarding categories occurred. A reason could be that the lecturer reminded the students on day eight to use what they had already covered in the lecture before. Another reason could be that the students were using day ten to finish the assignments they had started the days before. But, this was also the case on day 5.

5.3 S-Problems

In total, we observed 65 situations in which students were discussing problems and solving them on their own (39 in the first week and 26 in the second week). Relating these numbers to the categories (see also Figure 2), the S-problems occurred as follows:

- Creating proofs (11 times)
- Mathematical inscriptions (10 times)
- Coq's usability (3 times)
- Working with Coq (27 times)
- Data structures (14 times)
- Logic (0 times)

Creating proofs: On the first day, the S-problems were similar to the T-problems and students talked about the given assumption and how they were not able to get an idea for the solution while solving a proof with paper & pen. From day two to four, there were no problems with creating proofs that students could be able to solve by their own. On day five, there occurred S-problems with obtaining the correct assumption again and how to use previously created proofs. On the eighth day, two student groups discussed problems about how a proof should be formulated in natural language.

Mathematical inscriptions: S-problems of this category occurred on day four, when the students discussed how to write down a proof in natural language and in the correct way.

Working with Coq: The S-problems of this category from the third day on featured problems about working with Coq after using

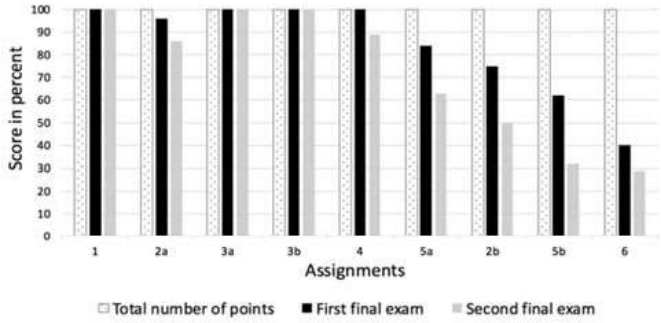

Figure 4: Achieved points in final exam

it for the first time and were similar to T-problems. Students also had problems with typing errors while transferring the assignment sheet to Coq. But in this case instead of the occurrences of T-problems, the students recognized the mistakes without help from the teacher or tutor. Furthermore, there were discussions about choosing the correct tactics.

Coq's usability: On day eight, the students were confused that Coq removes the brackets if a subgoal is executed. This was the only situation in which students had usability difficulties.

Data structures: In the second week, most S-problems of this category involved problems again about understanding the covered data structures. Most observed problems were regarding lists, which were introduced on day seven. Almost all problems with binary-trees on day nine resulted in T-problems and we could not observe situations in which the students solved problems about this topic on their own.

5.4 Students' Results of Final Exam

Fourteen students participated in the final exam and all of them passed it (the exam was offered twice for those who could not attend the first one). The grades obtained resembles a typical bell curve with the peak between B and C indicating that the exam assignments had an appropriate level of difficulty. The final exam had the four following assignment types: 1) Working with and in Coq (1, 2a, 3a, 3b, 4, 5a); 2) giving line-by-line comments for a proof in Coq (2b); 3) solving a proof with pen & paper that was solved in Coq before (5b); and 4) solving a proof with pen & paper that was not solved in Coq before (6) (see Figure 4 that is sorted by assignment types). The first and second assignment covered proofs about propositional and predicate logic, the third assignment was defining a data type while assignment four to six were about proofs about data structures.

Students scored best with proofs to be created with Coq. The scores were slightly worse for assignment types 2 and 3 if the students were suggested to write line-by-line comments for their code or had to write a proof in natural language that they had already solved in Coq. In detail, the students had problems being precise in their choice of words especially in assignment type 2 and problems with proof structuring in assignments type 3. The worst scores appeared with assignments of type 4, when students had to write the proof down without solving it in Coq first. Here, just like during exercise sessions, students' results indicated problems with identifying the

assumption of a proof and using it correctly in the chain of reasoning. Also, they were using wrong mathematical notations and seemed to be uncertain or confused about what information they already had and what they had to prove. In several cases, they only delivered an idea of what they were supposed to prove. However, this was mainly related to the assignment about data structures, which was the topic with which the students struggled most during the two-weeks block course.

5.5 Discussion

In this section, we will discuss our research questions stated in the beginning of section 4 while relying on the results presented in the previous subsections.

Research Question (RQ) 1: When starting with Coq, students required surprisingly little training to get used to work with the theorem prover. We were expecting students to be challenged by the way Coq displays the backward reasoning of a tactic in the right-hand upper window of the IDE (see also section 3.2). But, students seemed quickly understand this part. Except for one problem with the usage of brackets, we did not observe any serious usability issues with Coq. This indicates that provided presentations and explanations of how Coq works during lecture seemed to be sufficient. Also, we hypothesize that students were probably able to transfer their familiarity with programming IDEs (from previous programming courses attended at our department they are familiar at least with BlueJ and Eclipse) to the usage of Coq's IDE.

RQ 2: While the students were solving their first assignments with pen & paper on day one, it became obvious that they only had limited understanding of propositional and predicate logic and how to develop a formal proof within these topics (e.g. repeatedly, they tried using truth tables for developing a proof). This was the case although they all had attended an introductory course in theory of computation that covered propositional and predicate logic and most of our students had attended this course just three months earlier. The fact that creating proofs problems appeared with assignments about propositional logic and then again with predicate logic showed that the students could not easily transfer their understanding. With the pen & paper proofs it became quite obvious how difficult it was for the students to start a proof and decide when they had accomplished it. Instead, working with Coq seemed to lead to more satisfaction: The low attrition rate and overall positive feedback in the surveys regarding their interaction with Coq, indicates that students were very motivated to work with the tool and within the created setting.

RQ 3: Comparing T- and S-problems in both diagrams (see Figure 3 and Figure 2) we have reason to believe that working with Coq indeed seemed to help scaffolding students' activities in developing a formal proof. We conclude this from the strong decreasing amount of T-problems of category *creating proofs* during first week once Coq was introduced as well as remaining S-problems indicating that students were strongly working on their assignments but not relying anymore on the teacher or tutor. The more the course advanced, the less students ran into problems that were related to *creating proofs* or *working with Coq*. Also, the more skilled students became with using Coq, the less it was necessary or requested by the students to discuss all solutions in plenary by the end of each exercise session. Instead, students were able to discuss most questions and problems with each other. While Coq's scaffolding qualities seemed to support students strongly, the tool's strictness did not seem to limit students' activities. On the contrary, comparing their proof activities on the first day when using only pen & paper with days three and four while using Coq, the limitation of potential approaches enabled students to master at least one way of proof development instead of getting lost by to many options. Supporting our hypothesis of Coq's scaffolding ability, another observation is that through the entire course as well as during final exam students constantly performed better when using Coq in comparison to using pen & paper first or after using Coq (see corresponding assignment 5b and 6 in the final exam in Figure 4). This also corresponds to observations during exercise sessions where students displayed a dislike to start proofs with pen & paper as well as to switch back from Coq to pen & paper alike proofs.

One possible interpretation of students performing well in Coq but still struggling with pen & paper could be that they did not really master Coq proofs but got a solution by trial and error only. Although this strategy would have been possible for provided logic assignments, it would not have worked with data structures. Also, as we observed students working with Coq and in their questions they asked, this explanation seems to be rather unlikely. Instead, we rather believe that within our course design we did not pay enough attention to how Coq as a scaffolding element is dismantled carefully. This is also supported by situated cognition theory: Activities learned and practiced with a specific tool are not easily transferable to activities with another tool, especially when the tools embody different knowledge ([58], p. 10). In the next iteration of improving and testing our course design, we will pay particular attention to this aspect investigating further how Coq is mediating students understanding of formal proofs and what needs to be provided in order to create better transfer between tools.

6 CONCLUSION

The presented educational design research approach suggests the theorem prover Coq as a new learning tool for introductory courses to theory of computation. Theorem provers are designed and used for scientific and industrial purposes. Through creating specific assignments with predefined tactics, it was possible to transfer Coq into a learning tool and make it available to students in an introductory course on logic and data structures. Our evaluation shows that, contrary to prevailing assumptions, the students managed well to work with Coq. Our results strengthen the assumption that with our approach, indeed, Coq can be beneficial in scaffolding undergraduate students' activities of formal proofs. Especially the ability of creating a formal proof step-by-step and distinguishing between single elements of a proof, students were profiting strongly from the program's immediate feedback and scaffolding quality in comparison to pen and paper alike proofs. Open questions for future investigations will concern the nature of student understanding of formal proofs as it is mediated by Coq in comparison to pen and paper.

REFERENCES

[1] P. Anapa and S. Hatice. 2010. Investigation of undergraduate students' perceptions of mathematical proof. Procedia-Social and Behavioral Sciences 2.2: 2700-2706. S.

[2] S. Allen, M. Bickford., R. Constable, R. Eaton, C. Kreitz,, L. Lorigo, and E. Moran. 2006. Innovations in Computational Type Theory using Nuprl. Journal of Applied Logic, 4, 4, 428-469.

[3] P. Andrews, C. Brown, F. Pfenning, M. Bishop, S. Issar and H. Xi. 2004. Etps: A system to help students write formal proofs. Journal of Automated Reasoning, 75-92.

[4] D. Almeida. 2000. A survey of mathematics undergraduates' interaction with proof: some implications for mathematics education. International Journal of Mathematical Education in Science and Technology 31.6: 869-890.

[5] S. Autexier, D. Dietrich and M. Schiller. 2012. Towards an Intelligent Tutor for Mathematical Proofs. In Proceedings THedu'11: 1-28.

[6] C. Bereiter. 1997. Situated cognition and how to overcome it. In Situated cognition: Social, semiotic, and psychological perspectives, Kirshner, D. and Whitson, J. A., Eds. NJ: Erlbaum, Hillsdale, 281-300.

[7] Y. Bertot and P. Casteran. 2004. Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions, Springer.

[8] W. Billingsley and P. Robinson 2007. Student Proof Exercises Using MathsTiles and Isabelle/HOL in an Intelligent Book. Journal of Automated Reasoning, Volume 39(2): 181-281.

[9] Bove, P. Dybjer and U. Norell. 2009. A Brief Overview of Agda - A Functional Language with Dependent Types. In TPHOLs 2009, LNCS 5674, Springer, 73-78.

[10] E. Brady. 2013. Idris, a General Purpose Dependently Typed Programming Language: Design and Implementation. Journal of Functional Programming, 23(5):552-593.

[11] W. Brookes, W. (2004, January). Computing theory with relevance. In Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30. Australian Computer Society, Inc., 9-13.

[12] C. Chesñevar, M. González and A. Maguitman. 2004. Didactic strategies for promoting significant learning in formal languages and automata theory. In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education. ITiCSE '04. ACM, 7-11.

[13] J.S. Brown, A. Collins and P. Duguid. 1989. Situated cognition and the culture of learning. *Educational Researcher*, 18, 32–42.

[14] A. Collins, J. S. Brown and A. Holum. 1991. Cognitive apprenticeship: Making thinking visible. American Educator, 6(11):38-46.

[15] R. Constable et.al. 1986. Implementing Mathematics with the Nuprl Development System, Prentice-Hall.

[16] Q. Cutts, S. Esper, M. Fecho, S. Foster and B. Simon. 2012. The Abstraction Transition Taxonomy: Developing Desired Learning Outcomes through the Lens of Situated Cognition. In Proceedings of the ninth annual International Conference on International Computing Education Research. ICER'12. ACM, 63-70.

[17] E. Deitrick, B. Shapiro, M. Ahrens, R. Fiebrink, P. Lehrman and S. Farooq. 2015. Using Distributed Cognition Theory to Analyze Collaborative Computer Science Learning. In Proceedings of the eleventh annual International Conference on Computing Education Research. ICER'15. ACM, 51-60.

[18] D. Delahaye, M. Jaume and V. Prevosto. 2005. Coq, un outil pour l'enseignement. Technique et Science Informatiques, 24, 9, 1139-1160.

[19] V. Durand-Guerrier, P. Boero, N. Douek, S. S. Epp, and D. Tanguay. 2012. Examining the Role of Logic in Teaching Proof. In Proof and Proving in Mathematics Education, Volume 15 of the series New ICMI Study Series, 369-389.

[20] C. García-Osorio, I. Mediavilla-Sáiz, J. Jimeno-Visitación and N. García-Pedrajas. 2008. Teaching push-down automata and turing machines. *ACM SIGCSE Bulletin*, 40, 3.

[21] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot and I. Pasca. 2013. A machine-checked proof of the odd order theorem. In International Conference on Interactive Theorem Proving. Springer Berlin Heidelberg. 163-179..

[22] F. Guilhot. 2005. Formalisation en Coq et visualisation d'un cours de géométrie pour le lycée. TSI, 24, 1113-1138

[23] H. Habiballa and T. Kmet. 2004. Theoretical branches in teaching computer science. International Journal of Mathematical Education in Science and Technology. 35, 6, 829-841.

[24] T. Hales, M. Adams, G. Bauer, D.T. Dang, J. Harrison, L.T. Hoang and T. Q. Nguyen. 2015. A formal proof of the Kepler conjecture. arXiv preprint arXiv:1501.02155.

[25] M. Hamilton, J. Harland and L. Padgham. 2003Experiences in teaching computing theory via aspects of problem-based learning. In Proceedings of the fifth Australasian conference on Computing education, Volume 20, Australian Computer Society, 207-211.

[26] M. Henz and A. Hobor. 2011. Teaching Experience: Logic and Formal Methods with Coq. In Certified Programs and Proofs, Lecture Notes in Computer Science, Vol. 7086, Springer, 199-215.

[27] M- Hielscher and C. Wagenknecht. 2006. AtoCC: learning environment for teaching theory of automata and formal languages. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. ITICSE '06. ACM, 306-306.

[28] J. Hollan, E. Hutchins and D Kirsh. 2000. Distributed cognition: toward a new foundation for human-computer interaction research. ACM Transactions on Computer-Human Interaction (TOCHI), 7(2), 174-196.

[29] E. Hutchins. 1995. Cognition in the Wild. Bradford: MIT Press.

[30] IEEE Computer Society and ACM. 2013. Computer Science Curricula. DOI: 10.1145/2534860

[31] INRIA. The Coq Webpage with a collection of teaching practices: https://coq.inria.fr/cocorico/CoqInTheClassroom

[32] I. Karasavvidis. 2002. Distributed Cognition and Educational Practice, In Journal of Interactive Learning Research, 13 (1/2), 11-29.

[33] F. Kiehn, C. Frede and M. Knobelsdorf. 2017. Was macht Unentscheidbakreit und Turinmaschinen so schwierig? Ergebnisse einer qualitativen Einzelfallstudie. HDI 2017. 7,5. HDI-Workshop des GI-Fachbereichs Informatik und Ausbildung / Didaktik der Informatik

[34] M. Knobelsdorf. 2015. The Theory Behind Theory - Computer Science Education Research Through the Lenses of Situated Learning. In Proceedings of the 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP), Lecture Notes in Computer Science, Volume 9378, Springer, 21-21.

[35] M. Knobelsdorf and C. Frede. 2016. Analyzing Student Practices in Theory of Computation in Light of Distributed Cognition Theory. In Proceedings of the 12th ICER Conference, ACM, 73-81.

[36] M. Knobelsdorf, C. Kreitz, C., and S. Böhne. 2014. Teaching theoretical computer science using a cognitive apprenticeship approach. In *Proceedings of the 45th ACM technical symposium on Computer science education*. SIGCSE '45. ACM, New York, 67-72.

[37] Y. Lincoln, and E. Guba. 1985. Naturalistic Inquiry. Sage Publications. Newbury Park, California.

[38] S. McKenney and C. Reeves. 2012. Conducting educational design research. New York, Routledge.

[39] P. Mayring. 2000. Qualitative Content Analysis. Forum: Qualitative Social Research [Online Journal], 1, 2, Art. 20.

[40] R. Nederpelt and H. Geuvers. 2014. Type Theory and Formal Proof: An Introduction, Cambridge University Press.

[41] J. Nespor. 1994. Knowledge in Motion. Taylor & Francis ISBN 978-0-7507-0270-6.

[42] T. Nipkow, L. Paulson and M. Wenzel. 2002. Isabelle/HOL A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, Vol. 2283, Springer.

[43] M. Parker and C. Lewis. 2014. What makes big-O analysis difficult: understanding how students understand runtime analysis. Journal of Computing Science in Colleges, 29, 4, 164-174.

[44] R. Pea. 1993. Practices of distributed intelligence and designs for education. In Distributed cognitions: Psychological and educational considerations. G. Salomon, Ed. Cambridge: Cambridge University Press, 47-87.

[45] R. Pea. 2004. The Social and Technological Dimensions of Scaffolding and Related Theoretical Concepts for Learning, Education, and Human Activity. Journal of the Learning Sciences, 13(3), 423-451.

[46] B. Pierce et al. 2017. Software Foundations. https://softwarefoundations.cis.upenn.edu/current/index.html

[47] N. Pillay. 2009. Learning Difficulties Experienced by Students in a Course on Formal Languages and Automata Theory. SIGCSE Bulletin. 41, 4, 48-52.

[48] S. H. Rodger, B. Bressler, T. Finley, S. and Reading, S. 2006. Turning automata theory into a hands-on course. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. SIGCSE '06. ACM, 379-383.

[49] S. Rychen, L. H. Salganik. 2003. A holistic model of competence. In Key Competencies for a Successful Life and a Well-Functioning Society Rychen, S., Salganik, L. H. Ed, Hogrefe & Huber, Seatlle, 41-62.

[50] J. Sakowicz and J. Chrząszcz. 2007. Papuq, a Coq assistant. In Proceedings of PATE'07 conference, Elsevier, 79-96.

[51] R. Säljö. 1998. Learning as the use of tools: a sociocultural perspective on the human-technology link. *In Learning with computers*. Littleton, K. and Light, P., Ed. Routledge, New York, 144-161.

[52] M. Schiller, D. Dietrich and C. Benzmüller. 2008. Proof step analysis for proof tutoring – a learning approach to granularity. Teaching Mathematics and Computer Science 6.2: 325-343.

[53] A. H. Schoenfeld. 1994. Reflections on doing and teaching mathematics. In *Mathematical thinking and problem solving*, Schoenfeld, A. H., Ed. Routledge, 53-70.

[54] S. Sigman. 2007. Engaging students in formal language theory and theory of computation. SIGSCE Bulletin. 39, 1, 450-453

[55] M. Sipser. 2012. Introduction to the Theory of Computation Cengage Learning, Boston, USA.

[56] R. Sommer and G. Nuckols. 2004. A Proof Environment for Teaching Mathematics. Journal of Automated Reasoning, Volume 32(3): 227-258.

[57] M. Tedre and E. Sutinen. 2008. Three traditions of computing: what educators should know. Computer Science Education, 18, 3, 153-170.

[58] J. Tenenberg and M. Knobelsdorf. 2013. Out of our minds: a review of sociocultural cognition theory. Computer Science Education, 24, 1, 1-24.

[59] L. S. Vygotsky. 1978. Mind in Society: The Development of Higher Psychological Processes: Harvard University Press.

[60] K. Weber. 2001. Student difficulty in constructing proofs: The need for strategic knowledge. Educational studies in mathematics 48.1: 101-119.

[61] J.W. Wertsch. 1993. Voices of the Mind: Sociocultural Approach to Mediated Action. Harvard University Press.

[62] D. Wood, J. Bruner and G. Ross. 1976. The role of tutoring in problem solving. Journal of Child Psychology and Child Psychiatry, 17, 89–100