Learning to Reason Assisted by Automated Reasoning

Wolfgang Windsteiger[0000−0002−7449−8388]

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz (JKU)
Altenbergerstraße 69, 4040 Linz, Austria
`Wolfgang.Windsteiger@risc.jku.at`
`https://risc.jku.at/m/wolfgang-windsteiger/`

Abstract. We report on using logic software in a novel course-format for an undergraduate logic course for students in computer science or artificial intelligence. Although being designed as the students' basic introduction to the field of logic, the course features a novel structure and it adds some modern content, such as SAT and SMT solving, to the traditional and established topics, such as propositional logic and first order predicate logic. The novel course design is characterized by, among others, the integration of existing logic software into the teaching of logic. In this paper we focus on the module on first-order predicate logic and the use of the *Theorema* system as a proof-tutor for the students. We report on statistical evaluation of data collected over two consecutive years of teaching this course. On the one hand, we asked for feedback of students on how helpful they felt the software support was. On the other hand, we evaluated their results in the exams during the course and their development over the entire teaching period. The performance in exams is then correlated with students' own perception of the helpfulness of software.

Keywords: Theorema · Automated Theorem Proving · Teaching Logic

1 Introduction

The new curriculum (2013) for a bachelor study in computer science at JKU Linz assigned a prominent role to a new modern logic-course [2,7], whose novelties concentrate on two different aspects:

- we wanted students to see *logic in action* through the use of logic software in teaching and
- we wanted to present *modern topics* (such as e.g. automated or interactive theorem proving and satisfiability modulo theories) to the students early in their studies in addition to classical content such as propositional and first order predicate logic.

The course is designed consisting of four modules taught by different lecturers, who also use different software in their respective modules. The course starts with the first module on propositional logic (SAT) and finishes with quantifier-free first-order logic with theories in the fourth module (SMT). First-order predicate logic (FO) occupies the two modules (FOA and FOB) in the middle with FOA covering the language itself and its pragmatics, i.e., how first-order logic appears in everyday life of a computer scientist, e.g., when giving an algorithm specification. In this report we concentrate on module FOB on *proving in first-order predicate logic* and discuss how we use the *Theorema* system [6,8] to support the teaching of reasoning in first-order predicate logic. The FOB module consists of *three units* made up from

– a *lecture part*, where theory is presented,
– an *exercise session* based on weekly exercise sheets with examples,
– a *mandatory quiz*, which is a short exam of 15 minutes duration, and
– an *optional bonus exercise* based on using some logic software.

Moreover, there is one *optional lab exercise*, which is an extended bonus exercise, for the entire module.[1]

The lecture introduces the concept of proof trees for representing proofs in mathematics and computer science, a concrete set of inference rules for first-order predicate logic, and a proof search procedure that applies rules iteratively. The exercise sheets consist of concrete proof problems, starting with abstract inference rules in order to concentrate on the proof search, over formal proving with quantifiers, until finally ending with the informal natural language presentation of formal proofs. In the frame of the bonus exercises students are asked to generate automated proofs for selected exercises from the current exercise sheet, i.e., they should try to let *Theorema* do proofs that they should be able to do manually at the same time. In order to be admitted to submit solutions to a bonus exercise, students must answer a short questionnaire on their own perception of the usefulness of the software and its influence on their personal proving capabilities. In the lab exercise, in contrast, they should submit a hand-written proof of some simple statement plus an automatically generated proof of the same statement by *Theorema*. Both variants of the proof have to be compared in a short oral presentation.

We collect and evaluate statistical data regarding both the students' personal experience as well as their performance. We are interested in how students perceive the interaction with the *Theorema* system from their point of view and whether their exposure to the system shows a measurable influence on their own proving capabilities in the quizzes. In addition, we investigate how the performance of different groups of students develops over time. We present and compare data collected over two consecutive classes taught in winter 2020 and in winter 2021.

[1] Note that, in the statistical evaluation presented later, we neglect the final lab exercise at the end of the module, because there is no item following the lab exercise, in which we could measure some influence of doing the lab exercise or not.

	Week 0	Week 1	Week 2	Week 3	Week 4	Week 5
FOB1	FOB1L FOB1E* FOB1B*	FOB1E†	FOB1B† FOB1Q			L
FOB2		FOB2L FOB2E* FOB2B*	FOB2E†	FOB2B† FOB2Q		A
FOB3			FOB3L FOB3E* FOB3B*	FOB3E†	FOB3B† FOB3Q	B

Fig. 1. Nested Module Schedule. * marks publication and † marks deadline.

In Section 2 we present the detailed components of the course and their interplay (Section 2.1), a short introduction to the *Theorema* system (Section 2.2), and the way how logic software is applied in the various parts (Section 2.3). The main part is Section 3, where we discuss the statistics. The results presented in this work extend preliminary results presented earlier, see [9,10], by the follow-up statistics of the second year and the respective comparison of the development over the years.

2 The Use of Software in the Teaching of Logic

2.1 Modules, Quizzes, Bonus, Lab

The logic-course is composed of *mandatory* and *elective components* that contribute in different ways to the final grade for the whole course. Lecture and exercise follow the flipped-classroom paradigm, where students need to study the theoretical parts on their own and then come to the class and ask questions, discuss problems, and do examples. What was previously the lecture is now done by the students autonomously with the help of lecture notes, lecture slides, and video recordings done by the lecturers[2]. The former exercise class is now the classroom part where people are physically present.

The FOB module consists of *three units* (FOB1–FOB3) and the grading of the module is based mainly on the mandatory *quizzes* (FOB1Q–FOB3Q) after each unit. Students can enhance their scores through voluntary *bonus exercises* (FOB1B–FOB3B) after each unit and a voluntary *lab exercise* after the entire module. Through publication dates and deadlines for the respective items we enforce a nested sequence of lecture/exercise/quiz/bonus through the module as depicted in Fig. 1. For example, unit FOB2 starts in week 1 with making available the lecture material (FOB2L) for the students and the publication of

[2] Lecture recordings are almost mainstream nowadays, but we switched to flipped-classroom with videos already one year before the pandemic.

the exercise sheet (FOB2E) and the bonus exercise (FOB2B), respectively. In week 2 we discuss the exercises, and the respective quiz (FOB2Q) is in week 3 giving students one week time to practice and to digest the presentation of the exercises. Fig. 1 shows how the units overlap, e.g., in week 2 they finish unit 1 with submitting the bonus FOB1B and doing the quiz FOB1Q, then there is the exercise for unit 2, and after this exercise class they can already start to devote some time to unit 3 due to the publication of FOB3L, FOB3E, and FOB3B.

A bonus exercise spans an entire unit, its publication together with the respective exercise sheet marks the beginning of the unit and its deadline together with the quiz marks the end of the unit. We designed the units like this because the bonus exercise can then serve two purposes: if the bonus is done before the exercise class, the software can be a *tutor*, if it is done after the exercises the software can serve as *checker*, see also Section 2.3

2.2 Interactive Automated Theorem Proving in Theorema

The *Theorema* system aims to be a computer assistant for the working mathematician. Support should be given not only for proving but also for defining and executing algorithms, structuring knowledge, etc. The Theorema project started in the 1990s with a first prototype implementation in Mathematica 3, a major re-design and a re-implementation based on new features available since Mathematica 7 is now called *Theorema 2.0*. Since the system has been presented to this community several times, we refer to some overview articles [4,3,5,6] and only mention some peculiarities that are relevant for the didactical frame, in which the system is used in the logic course.

Working with Theorema 2.0 means to write mathematical content into a *Theorema notebook* and then perform some action on the content using the *Theorema commander*. By mathematical content we mean *informal parts* (basically everything that can be written into a Mathematica notebook, such as text, tables, figures, graphics, etc.) intermixed with *Theorema-specific formal elements* such as definitions or theorems. A Theorema notebook is just a Mathematica notebook using a special stylesheet in order to support special behavior of Theorema-specific items. The Theorema commander is a click-based user interface that supports certain manipulations on formal mathematical content, e.g., proving a formula based on some knowledge base. Every Mathematica user can use the features just described by loading the *Theorema 2.0* add-on package into a Mathematica session. The *Theorema* package is open source and freely available under GNU GPL 3 or higher.

Suppose the task is to prove statement A using knowledge represented by formulas K. The user would first type the statement A into a formula-cell in the notebook, usually inside a named Theorem-, Lemma-, or Proposition-environment. The same would be done for all formulas in K, where knowledge could go into Definition-environments or other theorems, lemmas, or propositions. We want to emphasize that formulas can be used as knowledge even if they have not yet been proven. All environments may carry names as well as each formula inside an environment can carry a separate label as name. Then one would switch to the Theorema commander and choose the Prove-activity, which guides one through the process of setting up the automated prover. The proof goal is defined by simply selecting the notebook cell containing the goal formula A. Next is the setup of the knowledge base, which is achieved through the *knowledge browser* that shows an outline of each open notebook displaying only formal mathematical content while preserving the sectional structure of the notebooks. Sectional groupings can be collapsed in order to gain an overview over the whole document. For the formal entities, the commander does not display the formulas in detail, it rather shows the formulas' labels only and presents the entire formula as a tool-tip when hovering the mouse over the label. Each formula is accompanied with a checkbox that, when checked, includes the corresponding formula into the knowledge base K.

The next step in the Prove-activity is the *setup of the prover*. A prover in *Theorema 2.0* consists of a collection of individual inference rules that are applied by a proof search engine in a certain order guided by rule-priorities. The system knows some predefined collections of rules including reasonable priorities that lead to "nice" proofs in many cases. However, the Theorema commander allows one to activate or deactivate every single inference rule, and the pre-assigned priorities can be changed as well. Once all settings are given the prove-task can be submitted. Usually, it is a good strategy to first run the prover with default settings and a limited search depth and search time (both configurable in the prover setup). In case the proof would not succeed, the failing proof can be inspected and checked whether certain settings might be changed in order to prevent the prover from running into an undesired path. Otherwise, search depth and search time can be increased in order to allow the prover to terminate successfully. When the prover stops it writes a link to the automatically generated proof into the notebook just below the environment containing formula A. When clicking the link, a nicely formatted version of the proof explained in natural language is displayed in a separate window, which also offers several options to simplify the proof.

During proof generation and when a proof window is open, the Theorema commander shows a *tree visualization* of the proof search. In a successful proof all nodes belonging to a successful branch are colored green, nodes in failing branches are red, and pending nodes are blue. Pending nodes are proof situations that can still be handled by one of the available rules. Simplification of a successful proof essentially removes all failing branches and pending nodes resulting in an all-green proof tree that in fact corresponds to a formal proof tree as taught in our logic-course. Click-navigation connects the Theorema commander and the proof window, i.e., clicking a cell in the proof window highlights the corresponding node in the tree view, whereas clicking a node in the tree scrolls the proof window to the textual description of the respective proof step. Moreover, when hovering the mouse over a node in the tree, the name of the rule applied in that node is displayed as a tool-tip.

Proving with *Theorema* is an experience of *interactive automated theorem proving*. Although finally a proof is generated in a fully automated fashion, there is a lot of user interaction along the way from stating a theorem until accepting its proof. Sometimes this interaction amounts to adjustments in the prover setup, such as activating or deactivating rules, rearranging rules through refining priorities, or adjusting search depth or search time. It is also common that a proof fails because of missing knowledge. In this case, auxiliary lemmata have to be formulated and passed to the prover and, ideally, also these lemmata are proved then. In any case, all these actions are *inter*actions because they are a reaction to the inspection of the failing proof and an insight regarding the reason for failure. The experienced user can learn a lot from failing proofs even but, admittedly, for beginner students this aspect is probably less pronounced.

2.3 How Software is Embedded into the Course

Our main didactic hypothesis is that for doing (correct and complete) proofs it is beneficial to first get acquainted with the *rules of formal proving* based on the formal language of first-order predicate logic. This contrasts the approach often seen in mathematics, where proving is taught as an "art" that is just demonstrated by many examples without actually telling the rules of the game. Students see many special tricks for particularly interesting special cases but are often insecure how to do the uninteresting routine cases. Therefore, we explain a set of "practical" proof rules for FO, where we neglect aspects of completeness and minimality and concentrate on convenience of the calculus for generating "nice proofs". We then view proofs as proof trees, where each connection from one node to its children must be justified by one of given rules. Finally, we discuss how a proof tree can be presented in natural language intermixed with mathematical formulas in different levels of detail depending on the context with the aim to de-mystify the "art of proving" that they often come across in their mathematics courses.

This view of proving is, in fact, the key philosophy behind the *Theorema* system since its beginnings in the early 1990s. It is based on a natural-deduction-like set of inference rules inspired by expert human provers and a pattern-based proof search procedure that aims at finding proofs that are similar to how well-educated human mathematicians would do the proofs. Proof trees are generated in form of proof objects that are on the one hand the basic datastructure guiding the proof search and on the other hand allow nice natural language presentations of proofs. One of the most important guiding principles in *Theorema* is natural style both in input and output. That said, *Theorema* seems a perfect fit for being taken as software support in our logic course. Unlike in other modules, where logic software *employs methods* taught in the course to solve concrete "practical problems", whose size alone would discourage solution attempts without machine support, we employ the *Theorema* software in module FOB to *help students to practice methods* that they should be able to apply even without the help of a computer.

The main idea of using *Theorema* in the frame of the FOB module is that students do the same proofs twice, once by hand and once with *Theorema*. We use the examples from the weekly exercises, which students are required to do by hand, and let them generate Theorema-proofs in the frame of the respective bonus exercises. From the sequence of tasks within one unit shown in Fig. 1, Section 2.1, one sees that students can either do the proofs *manually first* and then *check* whether *Theorema* does them the same way, or they generate an *automated proof first* and then do the manual proof just following the Theorema-model. Both scenarios offer some learning benefit for the students.

- The first approach will typically be chosen by students who are able to do the proofs by hand. Instead of waiting for feedback until the exercise session, they can compare their own proofs to the system-generated versions. They can either feel enforced or they can revise their own versions after comparing to the system proofs.
- The second approach may be a way for students who feel unable to do the proofs by hand. They can let *Theorema* do the proofs, then study the steps the system applied, and finally imitate the system proof in their own formulation (in the best case).

In the frame of the lab exercise, we try to guide the students towards the second approach. They are asked to first do a proof with *Theorema* and then, after understanding the Theorema proof, formulate a proof in their own words. They need to give a short oral presentation comparing the two versions of the proof. It was interesting to observe that many students decided in their hand written versions to deviate from the path that the Theorema proof had marked. In the majority of the cases the argument was that they felt their own version was easier to understand. However, the majority of these cases were logically wrong.

Due to didactical considerations but also organizational reasons, the use of software is on a voluntary basis only. This leads to (at least) two motivations for students to spend time on the software. One group of software users are the curious students that are interested in doing a bit more than required, they want to get as much as possible out of the course. The other group of users are those that need the bonus points in order to pass the course. We have no recordings of students' motives to do the bonus or lab exercises. Hence, we assume the statistics presented in the next section cover both groups equally.

3 Results

Fig. 2 shows the distribution of content over the FOB module. Examples to be solved with *Theorema* in the frame of the bonus exercises FOBnB are always examples taken from the corresponding exercise sheet FOBnE. E.g., in FOB3E students have to prove that a function $f\colon A \to B$ is surjective if its restriction $f|_C\colon C \to B$ for $C \subseteq A$ is surjective. The usual definition of surjectivity, i.e.,

$$\underset{y\in B}{\forall}\ \underset{x\in A}{\exists}\ f(x) = y$$

and the implicit definition of $f|_C$ to be the unique function $g\colon C \to B$ with $\underset{x\in C}{\forall}\ g(x) = f(x)$ are given, the task is to apply the definitions correctly and handle the (alternating) quantifiers appropriately depending on whether they appear

	FOBnE/FOBnQ	FOBnB
FOB1	pattern-based proof search procedure with hypothetical inference rules, first-order proofs without quantifiers	first-order proofs without quantifiers from FOB1E
FOB2	first-order proofs with quantifiers	first-order proofs with quantifiers from FOB2E
FOB3	first-order proofs with quantifiers and informal natural language presentation referring to concrete mathematical concepts introduced by definitions; induction proofs	concrete mathematical proofs from FOB3E

Fig. 2. Module FOB unit contents.

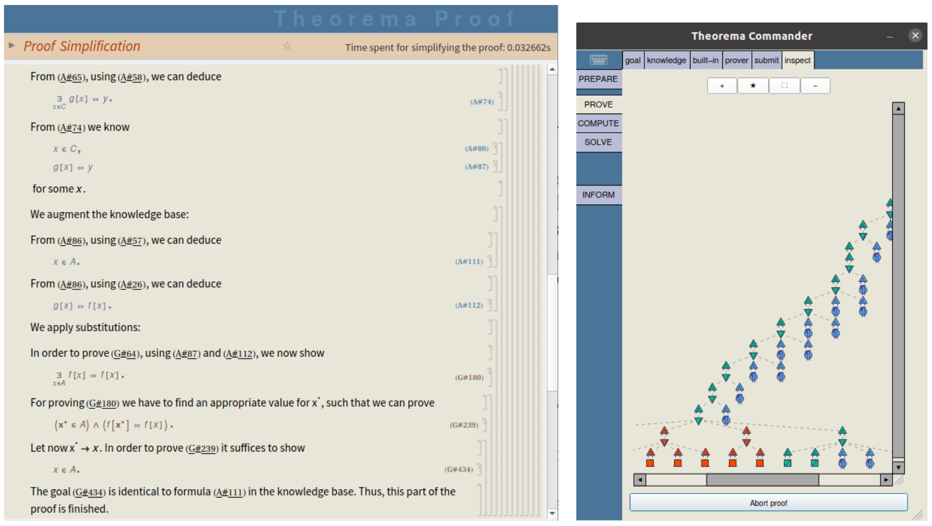
in the goal or in the knowledge. Consequently, FOB3B consists of a Theorema notebook containing the theorem and all necessary definitions in Theorema language. In addition, there are some hints concerning prover configuration because with standard settings *Theorema* would explore too many failing branches before it finds the successful proof. The task in FOB3B is then to generate a proof with *Theorema*. Fig. 3 shows part of an automated proof as generated by the *Theorema* system and the corresponding proof tree. The screenshot of the proof window does not show the entire proof, the intention is to only give a flavor of the style *how* automated proofs appear for students, namely with natural language explanation of each step and nicely formatted mathematical expressions. Moreover, both the proof as well as the proof tree are hyperlinked, meaning that a mouse-click on a formula in the proof will highlight the corresponding node in the tree, whereas clicking a node in the tree would highlight the corresponding cells in the proof notebook. We claim that this style of proof presentation is feasible for tutoring purposes for undergraduate students.

We try to capture two different aspects of using software in teaching logic: the *personal impression of students* after doing proofs with *Theorema* is collected using a questionnaire, while the *performance of students in the quizzes* is correlated to their participation and their performance in the bonus exercises.

The case study has been done in two consecutive years (winter 2020 and winter 2021) with identical content and only slight organizational changes. Most notably, in 2021 the submission deadline for the bonus exercises was right *before* the corresponding quiz with the consequence that the bonus exercise with *Theorema* had to be completed before doing the quiz. In 2020 the deadline was a little later so that it was possible in principle to do the bonus after the quiz.

3.1 Personal Impression of Students

Before being admitted to submit the solution of a bonus exercise, students were required to answer at most two standard questions about their experiences with


Theorema Proof

Proof Simplification — Time spent for simplifying the proof: 0.032662s

From (A#65), using (A#58), we can deduce

$\underset{x \in C}{\exists}\, g[x] == y.$ (A#74)

From (A#74) we know

$x \in C,$ (A#86)

$g[x] == y$ (A#87)

for some x.

We augment the knowledge base:

From (A#86), using (A#57), we can deduce

$x \in A.$ (A#111)

From (A#86), using (A#26), we can deduce

$g[x] == f[x].$ (A#112)

We apply substitutions:

In order to prove (G#64), using (A#87) and (A#112), we now show

$\underset{x \in A}{\exists}\, f[x] == f[x].$ (G#180)

For proving (G#180) we have to find an appropriate value for x^*, such that we can prove

$(x^* \in A) \wedge (f[x^*] == f[x]).$ (G#239)

Let now $x^* \to x$. In order to prove (G#239) it suffices to show

$x \in A.$ (G#434)

The goal (G#434) is identical to formula (A#111) in the knowledge base. Thus, this part of the proof is finished.

Fig. 3. Final part of a proof generated by *Theorema* (left) and proof tree visualization in the Theorema commander (right).

Theorema depending on whether they were successful in generating an automated proof with the system (Group A) or not (Group B). In each group they had to select the answer that fitted best to their personal situation regarding the relation between automated proving with software support and doing (the same) proofs by hand for the exercises. The possible answers are shown in Fig. 4 and 5, and according to their answers we formed categories A.1–A.9 and B.10–B.16. In both years we had 200–300 self-assessments for each of the bonus exercises and the ratio A:B was a constant ≈2:1. With these high numbers we consider the results to be non-random.

First we analyze the relative sizes of the categories in group A and compare the two classes winter 2020 and winter 2021, see Fig. 6. In both instances the top four are A.1, A.4, A.5, and A.8 and they separate quite clearly from the remaining five. Quite encouraging are A.1[3], which ranks first in FOB3B in 2020 and, in 2021, first in average over all three bonus exercises and first overall (groups A and B together). We also highlight A.8[4] particularly in 2021, where this category shows a monotonic increase over time to finally rank clearly first in FOB3B with a huge margin to the second-biggest group. Category A.5[5] displays an interesting development in 2021 from rank 1 with 23% in FOB2B dropping to rank 6 with only 7% in FOB3B.

[3] Not able to do the proofs by hand but feel capable after using *Theorema*.
[4] Hard time doing the proofs by hand but feel improvement through using *Theorema*.
[5] No problems doing the proofs by hand but will do proofs differently after having used *Theorema*.

A.1 I did not try or was not able to do the examples by hand, but now I think would be able to do them.

A.2 I did not try or was not able to do the examples by hand. I think I would still not be able to do such proofs.

A.3 I had no problems doing the proofs by hand. However, they are different from the Theorema proofs and I'm confused now whether my proofs are wrong.

A.4 I had no problems doing the proofs by hand. However, they are slightly different from the Theorema proofs because Theorema uses certain rules that I did not know. Still, I think my proofs are fine.

A.5 I had no problems doing the proofs by hand. However, they are slightly different from the Theorema proofs and in the future I would do my proofs differently.

A.6 I had no problems doing the proofs by hand. After doing the proofs with Theorema I realized that at least one of my original proofs was wrong.

A.7 I had a hard time doing the proofs by hand. However, I think when doing the next proof by hand, it will be equally difficult, doing the proof with Theorema did not help me for improving my own skills.

A.8 I had a hard time doing the proofs by hand. After doing the proof with Theorema I understand much better how all of this works. I feel that my own skills improved by using Theorema.

A.9 I don't see any connection between the examples from the exercises and the Bonus Exercise with Theorema

Fig. 4. Possible answers for Group A.

B.10 I did not try or was not able to do these examples by hand. I wanted to see how Theorema does the proofs, but I failed to produce a complete proof.

B.11 I did not try or was not able to do these examples by hand. Theorema is much too complicated for me to use it for such exercises.

B.12 I had no problems doing the proofs by hand. Unfortunately, I failed to produce a complete proof with Theorema. It would have been interesting to compare.

B.13 I had no problems doing the proofs by hand. I'm not interested how an automated proof looks, I have done them by hand anyway.

B.14 I had a hard time doing the proofs by hand. Unfortunately, I failed to produce a complete proof with Theorema. It would have been interesting to compare.

B.15 I had a hard time doing the proofs by hand. I'm not interested how an automated proof looks, I have done them by hand anyway.

B.16 I don't see any connection between the examples from the exercises and the Bonus Exercise with Theorema.

Fig. 5. Possible answers for Group B.

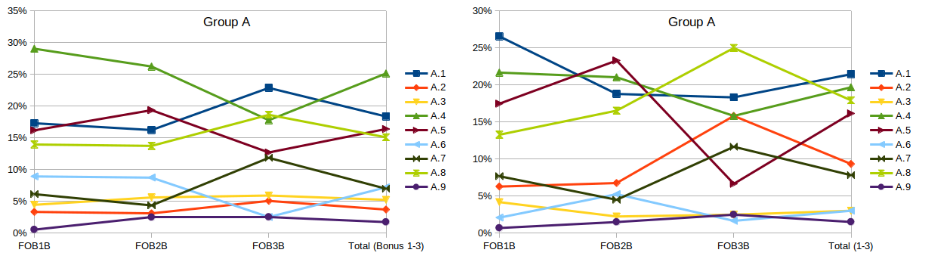


Fig. 6. Development of the group sizes (relative) in self-assessment from FOB1B to FOB3B in group A in winter 2020 (left) and winter 2021 (right).

The situation is less pronounced in group B, which are generally the less interesting cases, since these are the students who failed to generate automated proofs with *Theorema*, see Fig. 7. We are happy that those who consider *Theorema* too complicated (B.11) are a rather small group, and that, at least in 2021, those that were interested in *Theorema* at least in principle (though unsuccessful, categories B.10, B.12, and B.14) dominate those that show no interest at all (B.13 and B.15). Common to both instances is the significant drop of B.12[6] from FOB2B to FOB3B. We think that this is mainly due to fact that less people had no problems doing the proofs by hand, so the fraction of them that also had problems with the software shrunk as a consequence of that.

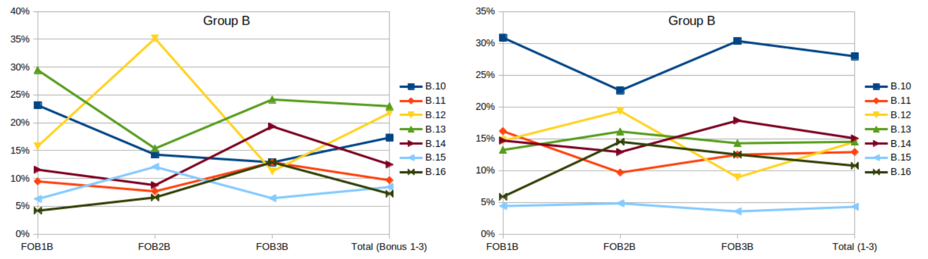


Fig. 7. Development of the group sizes (relative) in self-assessment from FOB1B to FOB3B in group B in winter 2020 (left) and winter 2021 (right).

3.2 Performance in the Quizzes

In addition to the students' *opinion* about the effects of software support on their own proving skills presented in the previous section we now analyze their

[6] No problems doing the proofs by hand but unable with *Theorema* although keen.

performance in the quizzes. The numbers do not differ much between the two instances of the course, the concrete numbers shown below are from winter 2021. In each quiz FOBnQ we compare the average points scored in the following groups:

All: all students in FOBnQ.
FOBnB: those students in FOBnQ who did bonus exercise FOBnB successfully.
FOB*B: those students in FOBnQ who did FOB1B–FOBnB successfully.
FOB0B: those students who did no bonus exercise successfully.

We not only record the average scores and standard deviations but add a statistical assessment in form of a *(two-sided) Student's T-Test* [1] comparing the sample values of different groups. The Student's T-Test gives a probability (p-value) that the given sample data occur under the hypothesis of equal mean values of underlying distributions. We use a variant of the test that does not assume equal variances of the underlying distributions. When we compare two samples with different averages, then a low p-value means that there is a low probability that the different averages occur although the underlying distribution have equal mean value, i.e., the mean values are equal and the different sample averages just occur by chance. Typically, in statistics, one calls a difference *statistically significant* if $p < 0.05$, so the smaller the numbers in the tables below, the more significant are the different averages in the samples compared. For example, the analysis of quiz FOB1Q (see Table 1) tells us that the average score 4.74 of the 107 students that also did bonus FOB1B is significantly better ($p = 5.65 \times 10^{-6}$) than the 4.36 scored by the 187 students that neglected FOB1B, and still significantly better ($p = 0.0003$) than the 4.50 scored overall by the 294 participants[7]. The under-average performance of those not having done the bonus is statistically not significant ($p = 0.0943$).

Table 1. Results of FOB1Q (max. 5 points) with p-values for equal means. Values in parentheses show the size of the groups and column '$\mu \pm \sigma$' contains the average scores (μ) in the group samples together with their standard deviations (σ).

	$\mu \pm \sigma$	All	FOB0B
All (294)	4.50 ± 0.81	—	—
FOB0B (187)	4.36 ± 0.93	0.0943	—
FOB1B (107)	4.74 ± 0.49	0.0003	5.65×10^{-6}

Table 2 shows the corresponding figures for quiz FOB2Q, which we consider the most difficult quiz because it is the first one with quantifier rules and it is only about quantifier proving whereas in FOB3Q it is quantifier proving as a

[7] Note that the statistical test gives much more confidence in different mean values than only comparing observed averages and taking into account the standard deviations or variances in the samples.

repetition plus induction proofs. The situation is similar to the one above: those who do the bonus perform best, followed by the overall average, and the no-bonus-group is the weakest. The different averages are statistically significant, since all p-values are far below 5%. Among the bonus students, we even distinguish between those who did *only FOB2B* and those who did FOB1B–FOB2B. The latter are marginally better scoring 3.87 against 3.79, but this difference cannot be confirmed statistically with $p = 0.6353$, meaning that it can be by chance with a probability of over 60%.

Table 2. Results of FOB2Q (max. 5 points) with p-values for equal means. Values in parentheses show the size of the groups and column '$\mu \pm \sigma$' contains the average scores (μ) in the group samples together with their standard deviations (σ).

	$\mu \pm \sigma$	All	FOB0B	FOB2B
All (290)	3.30 ± 1.29	—	—	—
FOB0B (166)	2.99 ± 1.24	0.0102	—	—
FOB2B (109)	3.79 ± 1.21	0.0006	2.41×10^{-7}	—
FOB*B (91)	3.87 ± 1.20	0.0002	8.43×10^{-8}	0.6353

Finally we discuss our observations in quiz FOB3Q, see Table 3. Again, those who do the bonus perform best, followed by the overall average, and the no-bonus-group is the weakest, but the averages differ much less. The only message that is supported by statistics is that those who did bonus exercises perform better than those who do not. As in FOB2Q, the difference between those who did only the last bonus to those who did all bonus exercises (3.58 vs. 3.68) is not supported by statistics with $p = 0.5620$.

Table 3. Results of FOB3Q (max. 5 points) with p-values for equal means. Values in parentheses show the size of the groups and column '$\mu \pm \sigma$' contains the average scores (μ) in the group samples together with their standard deviations (σ).

	$\mu \pm \sigma$	All	FOB0B	FOB3B
All (282)	3.46 ± 1.05	—	—	—
FOB0B (147)	3.30 ± 1.04	0.1329	—	—
FOB3B (97)	3.58 ± 1.07	0.3560	0.0474	—
FOB*B (64)	3.68 ± 1.10	0.1529	0.0215	0.5620

We also compared the performance of the groups A.1–B.16, see Section 3.1, against each other. We do not go into further detail because only a few of the different averages could be confirmed as statistically significant in winter 2020 and, unfortunately, even less in winter 2021.

A final observation we want to share is given in Fig. 8, which shows the development of the performance over time split up into the groups A.1–B.16 from Section 3.1. The striking development of group A.3 is less telling that one might expect because of the small size of that group (4%, 2%, and 3%), hence we neglect A.3 in the further analysis. One can see that subgroups in group A are a bit closer to each other than those in group B. There is a falling tendency from FOB1Q to FOB2Q, regardless of what happened in the bonus, and this is certainly due to the difficulty of FOB2Q, see above. Between FOB2Q and FOB3Q we see more diversity, some groups improve whereas some decline. Among the declining groups, A.2 and B.11 correspond to each other, it contains those that are not able to do the proofs by hand and that see no improvement through software support. Also declining are A.6 and B.13, both having no problems doing the proofs by hand. Those that used *Theorema* successfully at least detected that their hand-proofs are wrong. However, they seem to not have been drawing the right conclusions because still their performance got worse. Among the improving groups, A.7 is a strange phenomenon, because they improve in group A from last position (neglecting A.3) in FOB2Q to second in FOBQ3, although their self-assessment says they had a hard time with hand-proofs and the software did not help them. On the other hand, A.8 claimed to have the feeling their skills had improved through using *Theorema*, but their performance stays constant. Another nice facet is the development of B.14 and B.15 from FOB2Q to FOBQ3. Starting from the same level in FOB2Q, the ones that show interest in the software (although finally failing, B.14) improve in FOB3Q, while those that confess not being interested in the automated proofs show no improvement.

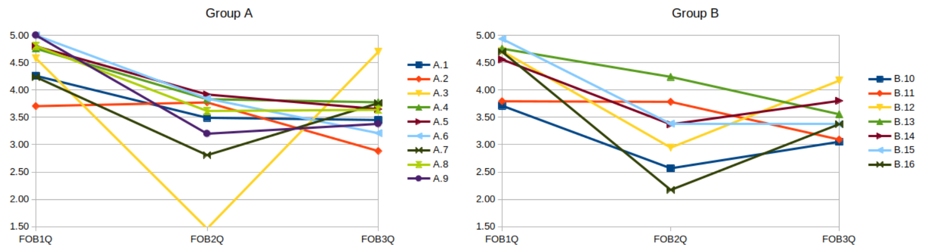


Fig. 8. Development of the performance from FOB1Q to FOB3Q in winter 2021 in group A (left) and group B (right).

4 Conclusion

We report on a big case study using the *Theorema* system as a proof tutor in a big logic course for almost 400 first semester students of computer science or artificial intelligence. The case study consists of a *self-assessment* of students after using the software and a *statistical evaluation* of test results based on groups defined through the answers in the self-assessment. Our didactical hypothesis is that students can improve their own proving skills through working with a natural-style automated theorem prover. Some statistics support this claim, e.g., in general, students that worked with the software performed better than average while the others show results under average. However, there are also surprising results that do not exactly match our expectations, e.g., students, who reported that the automated proofs were of no help, improved, while those, who claimed better understanding through working with the software, did not improve. Our expectation was just the other way round. In this context, it is important to recall that statistical tests can only reveal *correlations* but *no causalities*. This has to be emphasized in particular in our scenario where the groups are not assigned randomly but students actively join a group or not. Now, if one group performs better than another, this can be because students are better *because* of being in that group or because the *better students* (more talented, more interested, or more motivated) *chose* that group. A superior setup would be to divide the entire class into a group that does the bonus exercises and compare them to those that do no bonus exercises, but this is not feasible in our logic course because of the voluntary character of the software-related parts.

In future versions of the case study we have to take measures that answers in the self-assessment become more reliable so that students see no benefit in checking answers that the teachers might like better than others. Moreover, we will also try to investigate the development of individual students during the course like, e.g., follow the paths through which of the categories A.1–B.16 individual students travel from FOB1Q to FOB3Q. In any case, we are happy that the *Theorema* system can be applied in a reasonable way in education with a big group of first-semester students.

References

1. Student's T-Test. `https://en.wikipedia.org/wiki/Student's_t-test`, accessed: 2022-05-23
2. Biere, A., Schreiner, W., Seidl, M., Windsteiger, W.: Logic for Computer Science (2020), course in the first year in the bachelor program for computer science at Johannes Kepler University Linz (JKU), taught since 2013
3. Buchberger, B.: Theorema: A Proving System Based on Mathematica. The Mathematica Journal **8**(2), 247–252 (2001)
4. Buchberger, B., Craciun, A., Jebelean, T., Kovacs, L., Kutsia, T., Nakagawa, K., Piroi, F., Popov, N., Robu, J., Rosenkranz, M., Windsteiger, W.: Theorema: Towards Computer-Aided Mathematical Theory Exploration. Journal of Applied Logic **4**(4), 470–504 (2006). `https://doi.org/http://dx.doi.org/10.1016/j.jal.2005.10.006`
5. Buchberger, B., Dupre, C., Jebelean, T., Kriftner, F., Nakagawa, K., Vasaru, D., Windsteiger, W.: The Theorema Project: A Progress Report. In: Kerber, M., Kohlhase, M. (eds.) Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Compu-

tation and Mechanized Reasoning). pp. 98–113. St. Andrews, Scotland, Copyright: A.K. Peters, Natick, Massachusetts (6-7 August 2000)

6. Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-Assisted Natural-Style Mathematics. JFR **9**(1), 149–185 (2016), `http://dx.doi.org/10.6092/issn.1972-5787/4568`
7. Cerna, D.M., Seidl, M., Schreiner, W., Windsteiger, W., Biere, A.: Computational Logic in the First Semester of Computer Science: An Experience Report. In: Springer (ed.) CSEDU 2020. pp. 1–8 (2020)
8. Windsteiger, W.: Theorema 2.0: A Brief Tutorial. In: Jebelean, T., Zaharie, D. (eds.) Proceedings of SYNASC 2017. pp. 1–3. IEEE Explore (2017)
9. Windsteiger, W.: Automated Theorem Proving in the Classroom. In: Janicic, P. (ed.) Proceedings Automated Deduction in Geometry (ADG 2021). Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 352, pp. 54–63 (2021), `http://dx.doi.org/10.4204/EPTCS.352.6`, extended abstract
10. Windsteiger, W.: Automated Theorem Proving in the Classroom. RISC Report Series 21-15, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria (August 2021), extended version of keynote talk at ADG 2021 conference