



International Journal of Mathematical Education in Science and Technology

ISSN: (Print) (Online) Journal homepage: <https://www.tandfonline.com/loi/tmes20>

Interactive theorem provers for university mathematics: an exploratory study of students' perceptions

Paola Iannone & Athina Thoma

To cite this article: Paola Iannone & Athina Thoma (2023): Interactive theorem provers for university mathematics: an exploratory study of students' perceptions, International Journal of Mathematical Education in Science and Technology, DOI: [10.1080/0020739X.2023.2178981](https://doi.org/10.1080/0020739X.2023.2178981)

To link to this article: <https://doi.org/10.1080/0020739X.2023.2178981>



© 2023 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group



Published online: 30 Mar 2023.



Submit your article to this journal [↗](#)



Article views: 1943




View related articles [↗](#)



View Crossmark data [↗](#)

Interactive theorem provers for university mathematics: an exploratory study of students' perceptions

Paola Iannone ^a and Athina Thoma^{b,c}

^aDepartment of Mathematics Education, Loughborough University, Loughborough, UK; ^bDepartment of Mathematics, Imperial College London, London, UK; ^cSouthampton Education School, University of Southampton, Southampton, UK

ABSTRACT

Programming is becoming increasingly common in mathematics degrees as it is a desirable skill for new graduates. However, research shows that its use is mostly restricted to computational or modelling tasks. This paper reports a study on students' perceptions of and difficulties with Lean, an interactive theorem prover introduced as part of a transition to proof first-year module. The data consist of first-year university mathematics students' questionnaire responses ($n = 99$) and sections of 37 semi-structured interviews with students from the same cohort. Findings highlight students' difficulties with syntactic and strategic knowledge, in line with similar research on programming, and how conceptual knowledge is discussed in terms of conceptual-mathematics and conceptual-programming. Moreover, some students share how the experience with Lean changed their perception of mathematics by contributing to the epistemological shift from school to university, which is necessary for students to be successful. However, many students failed to engage with Lean due to its difficult syntax and because they perceived programming to be disconnected from the activity of proving and not worth the time investment needed to become proficient with this tool. We conclude with some reflections on the implications of this study for university teaching and suggestions for future research.

ARTICLE HISTORY

Received 25 March 2022

KEYWORDS

University mathematics; programming; interactive theorem provers; students' perceptions

1. Introduction

Programming is becoming one of the most important skills for knowledge transfer in mathematics. In a recent review commissioned by the Engineering and Physical Sciences Research Council in the United Kingdom (Bond, 2018) one of the recommendations for knowledge transfer is that:

All mathematics students should acquire a working knowledge of at least one programming language. (p. 13)

Indeed, at least in the United Kingdom, the presence of computational modules¹ in mathematics at university is increasing steadily (Iannone & Simpson, 2022). However, there is

CONTACT Paola Iannone  p.iannone@lboro.ac.uk

This article has been corrected with minor changes. These changes do not impact the academic content of the article.

© 2023 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group

This is an Open Access article distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited, and is not altered, transformed, or built upon in any way. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

still very little research on how the use of programming in mathematics degrees impacts students' learning of mathematics at the university level. Lockwood and Mørken (2021) in a recent editorial, call for more research in this field, stressing the importance of investigating the effect that the use of programming has on students' engagement with mathematical practices such as proving.

This paper reports findings of an investigation of students' perceptions of the introduction of an interactive theorem prover² in a first-year pure mathematics module and how this experience impacted their engagement with proof, their beliefs about the role of proof and the nature of mathematics. We first review the relevant literature on programming in university mathematics, we then briefly introduce the literature on interactive theorem provers and focus on Lean, which was used in the study. Then, we discuss our data collection methods and the context of our study. We present our study's findings and conclude with some implications for teaching undergraduate mathematics.

2. Programming in university mathematics

Undergraduate mathematics students, during their degree, engage with several programming packages such as SPSS, Excel, R, MATLAB and Python. To clarify what we mean by programming we adopt a definition of programming as an activity that requires students to write linked lines of code, debug, implement logical loops and use some variables. According to this definition, therefore, using SPSS or Excel does not constitute programming, while using Python or MATLAB does. A study on the teaching of programming in mathematics degrees in the UK (Sangwin & O'Toole, 2017) found that while computational applications of programming are relatively widespread in UK universities, the use of interactive theorem provers in pure mathematics is not. Moreover, a survey of Canadian mathematicians reported that 43% of the 302 participants used computer programming in their research, while only 18% included programming in their teaching (Buteau et al., 2014). Given the importance placed on programming skills, the findings of these studies seem to indicate the need for more varied use of programming in mathematics degrees and highlight the discontinuity between mathematicians' practices and their teaching – a gap often reported in mathematics education (Artigue, 2016).

The literature on the impact of programming on learning mathematics at the university level so far has focused either on the description of interventions using programming in teaching or on the impact of the use of programming on activities such as combinatorial thinking, modelling and proof-writing. The former comprises articles published in professional journals such as the *MSOR Connections* in the UK and describes the implementation of some aspects of programming in teaching. One typical example of this work is the paper by Lynch (2020) where the author describes how programming has been embedded in the mathematics curriculum at Manchester Metropolitan University (UK). Such papers offer many examples of implementations and practical suggestions for programming tasks and are mainly aimed at practitioners. The second strand of literature comprises a small number of papers that document the impact of programming on university students' mathematics learning. Three examples are the study by Lockwood and De Chenne (2019) in the context of combinatorics, by Buteau et al. (2020) on mathematical modelling tasks and by Thoma and Iannone (2022) on proof production. These studies

explore how asking students to use programming can contribute to their learning and to the consolidation of productive mathematical habits.

One area that has not been investigated in the mathematics education literature regards the difficulties that students encounter when starting to learn programming within pure mathematics, and what impact this experience has on their views of mathematics, or in other words, on their mathematics epistemologies. For a review of the literature on students' difficulties with programming, we therefore turn to the computer science education literature.

2.1. Students' difficulties with programming

In a comprehensive review of the literature, Qian and Lehman (2017) highlighted three types of difficulties students encounter when programming:

- Syntactic difficulties – when students cannot follow the syntax of the programme. Cases include the misuse of commas and semicolons, failing to define variables before using them and many more. These issues are thought to be the easiest to overcome and are indicated to be the most common difficulties that students have with programming.
- Conceptual difficulties – when students fail to grasp the basic mechanisms of the programme they are using. Such difficulties include not being able to use variables appropriately, for example thinking that the variables can hold more than one value or being unable to use loops.
- Strategic difficulties – these include difficulties in planning, writing and debugging a code, and generally include what some of the literature calls expert-level knowledge in computing.

Together with these main categories of difficulties the authors also propose other factors that contribute to difficulties students experience when programming. These include the complexity of the programming task, the confusion between natural language and technical (computing) language and previous mathematics knowledge. Qian and Lehman (2017) conclude that most of the difficulties that students encounter when learning programming originate from previous knowledge and previous experience, therefore, encouraging the study of programming difficulties in context.

In our study, we investigate programming difficulties in context both by considering the nature of the programme used, therefore contextualizing the investigation to a specific programming type, and the mathematical nature of the tasks for which the programme was used, contextualizing the investigation to pure mathematics and proof. Our study reports both on the difficulties students perceive they had when programming and how the students perceived that their attempts to use the interactive theorem prover impacted their views of mathematics. Below we provide an overview of the literature on proof and programming.

3. Proof and programming

Proof is one of the stumbling blocks for students in the transition from school to university mathematics, and indeed Gueudet et al. (2016) speak about a cognitive and epistemological crisis that students face in this transition. Understanding proof and its requirements is

therefore one of the important steps students must take to become part of the community of mathematicians (Dawkins & Weber, 2017). Among the issues that prevent this shift are students' difficulties with the technical language of mathematics (Lee & Smith III, 2009), with understanding the roles and necessity of proof (Hanna, 1990) and many others. It is beyond the scope of this paper to comprehensively review the literature on proof in university mathematics, but here we note that many interventions have been trialled to alleviate this situation, some of them including the use of programming languages such as ISETL (Dubinsky, 1995). However, these interventions have not lasted much beyond the time of the studies of which they were the focus. Even programmes such as ISETL, which has a wide set of resources for its use and that was shown to be effective in supporting students' learning of pure mathematics, are not in use – at least in the UK (Sangwin & O'Toole, 2017). We hypothesize, following Dawkins and Weber (2017), that this is because such interventions did not align with mathematicians' practices and therefore were not sustained beyond the initial interest of some motivated individuals who adopted them. The case of interactive theorem provers could be different as their use is gaining momentum in pure mathematics research. Investigating cases of teaching involving interactive theorem provers could help ascertain their pedagogical potential.

3.1. Interactive theorem provers

Interactive theorem provers are different from computational programmes such as MATLAB in that they have a symbolic library of mathematical objects that can be used to check and (potentially) write mathematics proofs. Such programmes include COQ (<https://coq.inria.fr>) and Lean (<https://leanprover.github.io/>). Python (<https://www.python.org>), which is becoming very popular not only in mathematics degrees but in engineering and biochemistry too, does have a symbolic mathematical library but this is not well developed and, to the best of our knowledge, is not used by pure mathematicians. Already Crowe and Zand (2001) mentioned interactive theorem provers as an emerging tool that could support logic teaching, but no example was offered of their use or the drawbacks or affordances of its introduction in first-year undergraduate mathematics teaching. More recently Hanna and Xiaoheng (2021) offer the theoretical underpinnings of why interactive theorem provers could be successfully used in teaching pure mathematics at the university level. They discuss the need for approaches to teaching pure mathematics that incorporate new technologies, therefore taking into account the demands of the modern workplace.

Interactive theorem provers have been in use in pure mathematics and computer science research since the 1960s with de Bruijn's Automath prover (Bruijn's, 1980), but they have just now started to make their way into the teaching of undergraduate mathematics and the first reports of their use are encouraging. Avigad (2019) reports the experience of a first-year module in logic that included Lean programming in the curriculum and observes how the use of Lean supported the students in the transition to the rigour of mathematics and helped them appreciate both the need for proof and the need for technical language (of mathematics and computing). Buzzard (2020) reports his experience of using Lean both as a research tool and for teaching a pure mathematics module in the first year of undergraduate mathematics. In educational research, Thoma and Iannone (2022) analyse written proofs by students who had or had not engaged with Lean programming

and show how learning to use Lean can help students overcome some of the difficulties of their first encounter with proof, even when writing proofs with pen and paper. The analysis on aspects of students' proof production included using the mathematical technical language, being able to divide a proof into goals and subgoals and generally writing proofs in a style that aligns with what mathematicians expect (for a discussion of mathematicians' expectations of proof-writing see Lew & Mejía-Ramos, 2019).

Proof and proof-writing have been long recognized as one of the stumbling blocks for students starting a university degree, both for writing and producing proofs (Moore, 1994), and for accepting the values conveyed by the place that proof has for the mathematics community (Dawkins & Weber, 2017). It therefore appears important to investigate the impact of the use of a programme designed to code proofs on students' interaction with proof. In the following section, we discuss Lean in some detail as this was the interactive theorem prover used in this study.

3.2. Lean

Lean is a programming language designed for research in mathematics that is now attracting increasing interest as a tool for teaching pure mathematics to undergraduate students (Avigad, 2019). The aim of Lean, an open-source theorem prover, is to bring interactive and automated reasoning together and build a theorem prover with powerful automation and an automated reasoning tool that can check and produce proofs (although the latter aim is not realized yet).

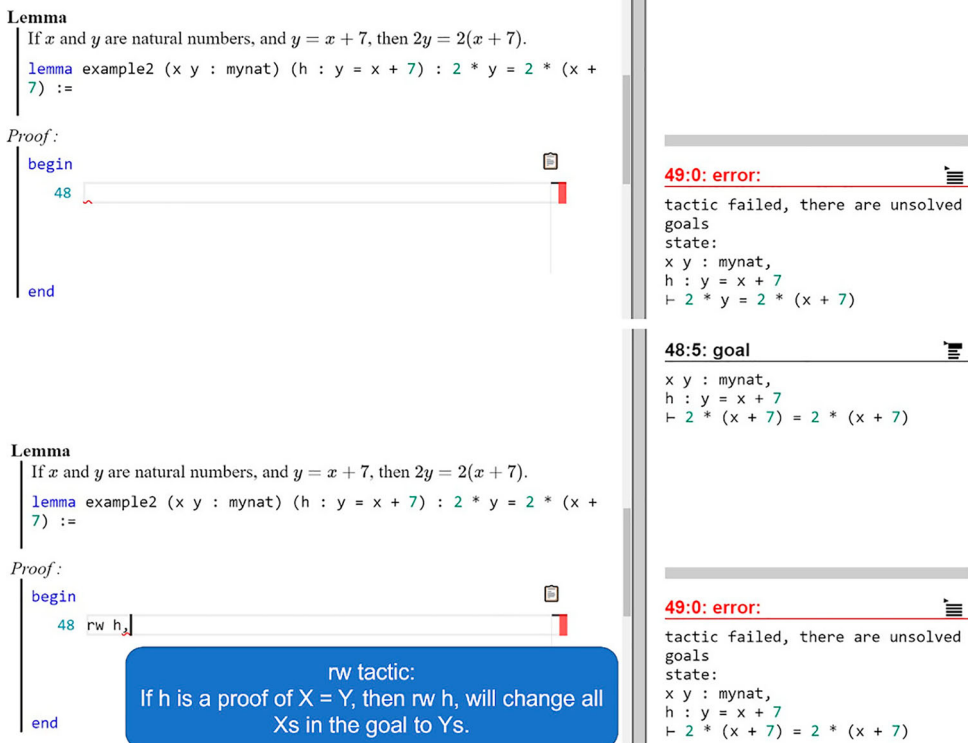
The Lean project started in 2013 and currently both a downloadable and an online version of the software are available. Lean requires the user to clarify the goal of a proof, to break the goal of a proof in smaller sub-goals, to make sure that mathematical symbols are used consistently. The code is checked, and the user receives instant feedback regarding symbol and logical consistency of the statements.

Lean is currently one of the few programming languages used in pure mathematics research and shows potential in terms of possible support in pure mathematics teaching, particularly transition to proof and logic modules. Figures 1 and 2³ show Lean's interface. The user writes on the left-hand side of the screen between the *begin* and *end*. Lean shows the goals on the right-hand side and each time a line of code is added the goal changes. If the code is incorrect an error message appears to help the user identify where the written code is incorrect (Figure 2).

4. Aims of the study

The literature on students' difficulties in programming framed our research, however, students need to understand mathematics conceptually to be able to handle (and eventually contribute to) Lean's symbolic library or programme proofs in Lean. Therefore, the current study not only focuses on students' perceptions of the use of one interactive theorem prover but also includes an investigation of how becoming familiar with an interactive theorem prover may impact students' views of mathematics. To conduct this investigation, we asked the following research questions:

RQ1 What are the difficulties that students report they encountered when using Lean while studying a transition to proof module?



Lemma
If x and y are natural numbers, and $y = x + 7$, then $2y = 2(x + 7)$.

```
lemma example2 (x y : mynat) (h : y = x + 7) : 2 * y = 2 * (x + 7) :=
```

Proof:

```
begin
```

48

```
end
```

48:0: goal

```
x y : mynat,
h : y = x + 7
⊢ 2 * y = 2 * (x + 7)
```

49:0: error:

```
tactic failed, there are unsolved goals
state:
x y : mynat,
h : y = x + 7
⊢ 2 * y = 2 * (x + 7)
```

Lemma
If x and y are natural numbers, and $y = x + 7$, then $2y = 2(x + 7)$.

```
lemma example2 (x y : mynat) (h : y = x + 7) : 2 * y = 2 * (x + 7) :=
```

Proof:

```
begin
```

48 rw h₂

```
end
```

48:5: goal

```
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

49:0: error:

```
tactic failed, there are unsolved goals
state:
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

rw tactic:
If h is a proof of $X = Y$, then $\text{rw } h$, will change all X s in the goal to Y s.

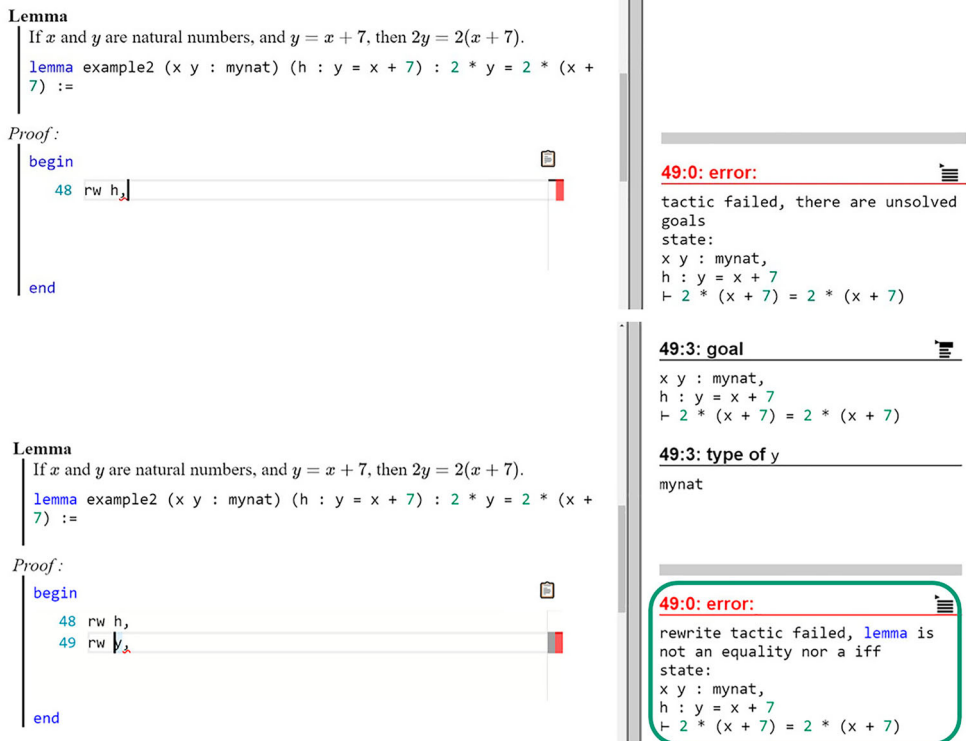
Figure 1. Multiple screenshots showing an example of breakdown in goals and subgoals in Lean – with correct use of tactics with our explanatory note on the `rw` tactic. Note: Tactic mode and term modes are the modes most used to code proofs in Lean. The Lean manual (<https://Leanprover.github.io/reference/tactics.html>) describes a tactic as ‘a sequence of instructions bracketed by keywords beginning and end’.

RQ2 What was the effect of students’ encounter with Lean on their views of mathematics?

5. Methodology, methods and analysis

5.1. The study of students’ perceptions in educational research

The perspective through which we frame the study is that of the investigation of students’ perceptions of Lean and how those perceptions impacted on their views of mathematics. This approach is not new, Entwistle and Ramsden (1983) and later Marton and Säljö (1997), in discussing the relationship between approaches to learning and differences in outcomes of learning, found that the differences in outcomes of learning depend on the differences in the way in which students engage to achieve such learning. They found two ways of engaging that may be adopted by the students and described those as approaches to learning. Entwistle and Ramsden (1983) named these approaches as deep learning and surface learning. They characterize *deep learning* as learning with the intention of understanding, which



Lemma
If x and y are natural numbers, and $y = x + 7$, then $2y = 2(x + 7)$.

```
lemma example2 (x y : mynat) (h : y = x + 7) : 2 * y = 2 * (x + 7) :=
```

Proof:

```
begin
  48 rw h,
end
```

48:5: goal

```
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

49:0: error:

```
tactic failed, there are unsolved goals
state:
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

49:3: goal

```
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

49:3: type of y
mynat

49:0: error:

```
rewrite tactic failed, lemma is not an equality nor a iff
state:
x y : mynat,
h : y = x + 7
⊢ 2 * (x + 7) = 2 * (x + 7)
```

Figure 2. Multiple screenshots showing an example of breakdown in goals and subgoals in Lean – with incorrect use of tactics and error message, with our addition of the box around the error note.

in mathematics is akin to relational understanding (Skemp, 1978) while *surface learning* is learning for memorizing, passively accepting ideas and information, which in mathematics is akin to procedural understanding (Skemp, 1978). In subsequent studies, Marton and Säljö (1997) found that a key influence on the way in which students engage with learning are the perceived demands of the learning task. This is to say that the way in which a student perceives the task – its difficulties, what questioning is valuable for learning, what are the required outcomes of that learning – is a big influence on their engagement with learning. From the research outlined here, it appears that students' self-report on aspects of their learning experience offers invaluable insight into their engagement with learning. It is within this line of research that our study and its research tools were designed.

5.2. Methods

The data reported in this paper are part of a larger dataset collected in one university in the South of England in the academic year 2018–2019. The mathematics department of this university ranks consistently in the top five in the research assessment in the UK and students need to achieve the highest marks in mathematics at the end of the school

examinations to be admitted to this university. The cohort that took part in the study consisted of about a third of international (i.e. non-UK) students, which is the norm for this university. The students that took part in the project were enrolled in a transition to proof module in the Autumn term of their first year. The module was taught traditionally with a mixture of lectures and seminars and the Professor who taught the module held voluntary workshops on the use of Lean⁴ every Thursday evening during the teaching period and used Lean in their research. During these workshops, the students and the Professor worked on coding some of the modules' proofs in Lean and wrote code for mathematics objects aiming to expand the Lean library. The module materials, lectures and exercise sheets were designed so that they could be solved by using Lean. It is important to note, that at the time of data collection, very few learning resources existed to support the teaching of Lean, apart from the Lean manual (<https://Leanprover.github.io/reference/>) which the Professor thought was difficult to follow.

There were 300 students enrolled in this cohort and of those only 18 took up the voluntary workshops on a regular basis, although many more attended once or twice during the semester. The Professor also discussed the use of Lean during some of the compulsory lectures. In those instances, he showed how some of the proofs included in the syllabus could be programmed in Lean by displaying the coding of the proofs on the screen overhead and at times coding live in class. In this way, students could see how this interactive theorem prover worked and what was required to be able to use it.

The data analysed for this paper concern students' difficulties with and perceptions of Lean programming. We collected two data types. The first data type was responses to a questionnaire which was administered to the whole cohort near the end of the teaching period. The aim of the questionnaire was to ascertain students' level of engagement with Lean and reasons for engagement or disengagement ($n = 99$, for the text of the questionnaire see the Appendix) and was anonymous. The questionnaire included biographical questions, two closed questions about the familiarity of the students with Lean and their reasons for engaging (or not) with this programme, and several open questions regarding students' experiences with Lean.

The second data type consisted of the first 10 minutes (on average) of semi-structured interviews with 37 volunteering students during the last two months of the teaching period. These interview sections investigated students' perceptions of the use of Lean in more detail (see Figure 3 for a breakdown of the participants in the different data collection periods). The remaining parts of the interviews asked students to go through some pen and paper proof tasks and the analysis of these data is not part of this study. The interviews were carried out by the second author of this paper. The relevant sections of the interviews were transcribed for ease of analysis. Of the students that participated in the interviews, seven were Lean users and regularly participated in the Lean workshops.

5.3. Analysis

The questionnaire contained ten questions, three closed and seven open-ended. In this study, we focused on the analysis of seven questions (Q1, Q5 – Q10 – see the Appendix for the full questionnaire. Responses to Q10 were considered if the response mentioned Lean or programming in relation to pure mathematics). For the three closed questions (Q1, Q5

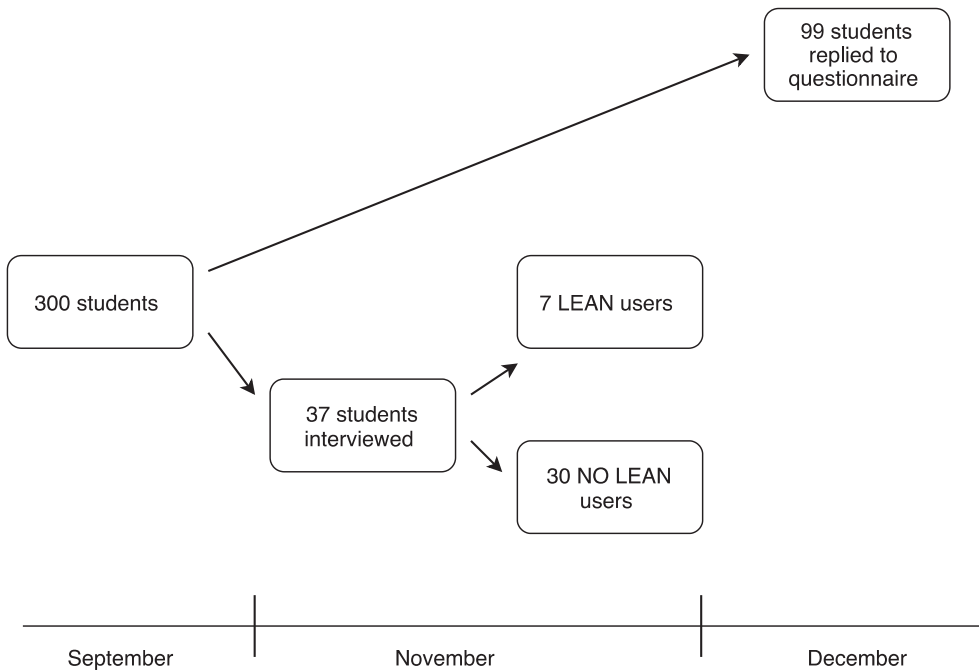


Figure 3. Break down of participants and data collection.

ad Q9), numbers of responses are reported. Students' responses to the open-ended questions (Q6, Q7, Q8, Q10) and the interview transcripts were analysed following thematic analysis in two stages:

- (1) **Stage 1:** the first cycle of analysis followed loosely the categories of difficulties related to programming found in Qian and Lehman (2017). To those, we added categories for students' affective response to the use of Lean, and Lean's impact on students' perceptions of mathematics.
- (2) **Stage 2:** the second cycle of analysis was carried out within each of the categories found in the first cycle to further refine the categories.

In the first cycle of analysis, we observed that most of the difficulties with using Lean could be coded according to Qian and Lehman's (2017) framework as difficulties in conceptual, syntactic and strategic knowledge. To those, we added a category for affect (when students expressed emotions related to the use of Lean) and a category for epistemology (when students reflected on the nature of mathematics). This first cycle of coding was akin to structural coding (Saldaña, 2021) in that the data was broken down into segments and then grouped into overarching themes. The overarching themes originating from this first cycle of coding are reported in Table 2 together with sample statements.

In the second cycle of coding the utterances in each of the themes in Table 2 were coded again line by line. During this part of the coding exercise, the broad categories were refined, and sub-themes were obtained. A small part of the first cycle of coding was carried out by

Table 1. Data reporting the reasons students did not engage with Lean – for this question students were allowed to choose more than one reason for non-engagement.

Reasons for not engaging with Lean	Number of students
Time constraints	49
Difficulty with the programming language	23
Not interesting enough	14
Other	9
Not applicable – I am still using Lean	13

the two authors independently and then compared to achieve a shared understanding of code meanings (as in triangulation analysis, Mok & Clarke, 2015).

6. Results

6.1. Closed questions

The gender breakdown of the respondents to the questionnaire reported 71 male students, 27 female students and one student who preferred not to say.⁵ This gender distribution is not surprising as it mirrors the gender distribution of mathematics degrees in the UK, especially in research-intensive universities like the one where the data were collected. When students were asked to state their familiarity with Lean the responses confirmed the very low uptake of the voluntary sessions offered by the Professor. Only eight students replied that they were extremely or moderately familiar with Lean while 43 students admitted to not being familiar at all with the software. We recall here that the Professor had shown proofs written in Lean and had done some live coding during lectures. They also discussed the use of this interactive theorem prover in class with the students. Therefore, students who admitted to not being familiar at all with Lean would have seen its use and some coding during lectures. Finally, the last closed questions asked students the reasons why they did not engage with Lean. The results are reported in Table 1.

We observe that the most common reason for not engaging was the lack of time. As Lean workshops were offered as an extra-curricular activity, many students did not feel that they could spare the time in an already very busy schedule. This is reflected in the analysis of the qualitative comments as this quote shows:

Yeah. Basically, I've just been trying to keep up with problem sheets. If I did have spare time, I would look into Lean. But right now, I feel like just keep on top of what being given is more important. So yeah, as soon as if I did ever catch up, I would probably check out Lean if it was still an option. But as of right now I'm just getting further behind. (Nick⁶ – non-Lean user)

6.2. Qualitative analysis

The definitions of the codes in the first level of coding and examples of utterances are given in Table 2. In what follows the main themes are discussed in turn to present students' experiences with Lean.

Table 2. List of first-level coding with definitions and examples adapted from Qian and Lehman (2017). The numbers next to the utterances indicate students' allocated numbers as given in the spreadsheet of the collected data.

Codes and their definition	Examples
Affect Emotional response to using LEAN.	It's a cool language. If I have [the] chance and time in the future, I'd like to learn it. [83]
Conceptual General and specific knowledge about how LEAN works, constructs and operations.	Seeing how dependent type theory can be used as a foundation of all of maths. [85]
Syntactic General and specific knowledge about the vocabulary and syntax of LEAN.	It's hard to remember all the command in LEAN. It's really different from normal maths. [56]
Strategic Applying programming – use of tactics.	Proofs for the equivalence relation lectures could be transliterated easily into LEAN, but the early sheets involving sets and polynomials were difficult, even with tactics such as <i>norm_num</i> . [64]
Epistemologies Utterances about the nature of mathematics and proof in particular.	Yes, doing Lean makes me constantly see maths from a different perspective, thus giving me extra insight into what I'm trying to prove. [1] Definitely, seeing the way maths is built from the bottom up is very beneficial to actually understanding content [11]

6.2.1. Affect

In this category are utterances that concern students' emotional responses to Lean. Emotional responses were, as expected, both positive and negative. For the positive emotions, fun and enjoyment were most frequently mentioned, both in the questionnaire and in the interviews. Fun was mostly linked to programming and for some this feeling of fun was enough motivation to carry on engaging with Lean:

Definitely, it forces you to think more about how you're proving something. And it's fun, so more likely to do it. [97]

Yes. Programming is fun. Maths is fun. Combining the two is even more fun. Fun is good. Fun is life. [3]

The feeling of enjoyment was mostly linked to the enjoyment of the achievement that could be obtained by using Lean, to the discovery of new ways of doing mathematics and to the newly found interest for mathematical rigour:

I enjoyed seeing properly rigorous proofs for the first time. [12]

I enjoyed the Professor, and the art of some of the foundations of maths. [68]

However negative emotions such as helplessness, being unpleasantly challenged, perceptions of Lean being too hard to learn and being overwhelmed by the challenge were by far the most common in our sample. Such feelings were especially evident in the interviews:

I struggled a lot. I think it's very different to not only thinking about normal proofs. I think it's very different to any other programming language. So essentially using any knowledge of any other programming language is irrelevant. And when I say I didn't had a bad experience of ... , I think I was just put off with just how new it is. (Nataly – non-Lean user)

Other students simply did not enjoy learning about Lean and did not think that Lean added anything to their learning:

All very complicated and confusing as I have no idea how to understand Lean. [10]

These feelings of helplessness, challenge and difficulty, especially the feeling that this programming language was very different from any other programming language the students had seen, stopped many students from engaging. Also, the fact that using Lean was not a compulsory part of the module may have demotivated students who were already very busy and were struggling to keep pace with the demand of their degree.

6.2.2. *Syntactic knowledge*

This umbrella theme contains utterances where students raised concerns in terms of the syntactic knowledge needed to use Lean. Students discussed issues with the vocabulary and syntax of Lean which hindered their use of the software at great length, both during the interviews and in the questionnaire answers. We note that the difficulty with the programming language was also the second most selected reason of disengagement with Lean as reported in the summary of the results from the closed questions (Table 1). Indeed, 43 utterances across the 99 students in the open comments refer to explicit difficulties with syntax. Students who expressed their problems with the syntax, discussed the ‘intuition’ needed to be able to understand the Lean code and the time investment needed to learn the syntax:

Syntax, as a non-Leaner requires a lot of intuition about the code to understand what’s happening. [16]

Syntax took time to get used to. [88]

For some students, writing code in Lean to prove a statement was very different from writing the same proof on pen and paper and this posed a real challenge. They could not connect what they were trying to do in Lean to the way in which proofs are written on paper. In their responses, some students drew parallels between writing in Lean and writing in a different language.

Actually, it doesn’t [is not helpful], because using Lean is just like talking math in French, I totally have no idea about what it says. [43]

It’s hard to remember all the commands in Lean. It’s really different from normal maths. [71]

Finally, student [89] shared their experience in terms of the syntax and discusses potential pedagogical interventions to help support learning the syntax and becoming more familiar with the vocabulary in a particular proof:

Once I understood the local syntax (by which I mean the functions specifically required for that lecture) it was quite straight forward. Perhaps spending a minute listing out the functions that will be used before the proof would be useful. [89]

In summary, utterances in this category point towards general perceptions of difficulties for the language syntax and at time frustration with not being able to use Lean due to the syntax barrier.

6.2.3. *Conceptual knowledge*

In this category are utterances that concern both conceptual knowledge of mathematics and links between programming and mathematics.

Many students (for example we found 41 utterances directly related to rigour and proof among the answers to Question 6), regardless of their familiarity with Lean coding, discussed how Lean examines the rigour of the proof. Comments such as:

I liked verifying the proofs that we saw in lectures. [73]

You cannot fall in the trap of trying to explain a proof to a person. You have to use rigorous water-tight arguments. [74]

are common across the questionnaire replies indicating that many students at this stage had developed an intuitive concern about rigour in mathematics. During the interviews, the students discussed this issue in more detail. In the following extract, we observe how the student depicted Lean as a knowledgeable other to whom they must communicate mathematics. More specifically, the student observed that writing in Lean requires rigorous proofs and statements. Also, the student added that unclear or inaccurate statements will not be accepted by Lean.

Well first of all, Lean doesn't take any handwaving [laughs] right. So that... It teaches you something about rigour, and I think that was the most important point about it was the concept of rigour that... that was, I think, the most beneficial part of what Lean has given to me. The concept of rigour. (Luke – Lean user)

However, the same student later reflected on his concept of rigour.

There's something that I realised. So, Lean made me really aware of the idea of rigour right. But then I also noticed when you're faced with a mathematical problem initially, rigour is not the most important point [...] I realised rigour was always a retrospective thing 'cause I noticed when I was doing some of the exercises in that book trying to rigourise everything straight away made me not... like I couldn't work out the solution as quickly as if I just intuitively thought about it first and I said this is probably the solution. (Luke – Lean user)

In the quote above, the student realized that rigour in proofs is important, especially if the proof is to be written in Lean code, but in retrospect. They reflected that initially, the proof must be thought of intuitively and then the writing could be made rigorous, conveying the importance of mathematical intuition and conjecturing when writing proofs.

One of the features of Lean is the need to break down the proof into goals and subgoals for programming it. Some students mentioned this feature of the smaller proof steps as helpful in writing rigorous proofs, while others discussed how this helps in finding the next step in the proving process and shows the necessity of proof even for trivial statements:

Found it to be an excellent way to keep track of progress through a proof, especially long ones; Lean tells you what parts of the proof have already been completed and what goals remain. [98]

In an interview, a Lean user discussed in more details the structure and the steps of the proof and how in Lean these steps must be explicit and complete for the proof to continue. Like other students in our study, this student drew a comparison between the Lean proofs and the pen and paper proofs. In this utterance, the student discussed how different is the proving process in Lean and pen and paper, especially considering the goals of the proof. In terms of his Lean proving, since the software provides the steps, he did not focus on the whole proof but just on each of the parts that Lean required him to prove (see also Figure 1 for Lean's interface).

So... I don't know if it's good or bad, but like sometimes I don't see the whole picture of the proof, I just look at the screen on the side and now you have to prove this. [...] And so when I do pen and paper, it's more like I'm looking at the whole thing. But like in Lean, it seems to work even faster because like the computer, like I don't have to write up everything I know exactly what I need to prove, like the computer does that for me, at least in tactic mode. (Larry – Lean user)

Many students remarked on the way in which mathematics is built in Lean starting from the foundations. One student expressed in their response how interesting was to see that mathematics can be constructed from Dependent Type Theory.⁷

Seeing how Dependent Type Theory can be used as a foundation of all of maths [...] Partially, certainly the ideas behind it have been very beneficial. Using Lean would improve my understanding of Dependent Type Theory, on the other hand I now understand Dependent Type Theory enough to help with my proofs and I'm not sure how much using Lean further would improve my understanding – once you know it, you know it. [99]

Similarly, another student discussed how viewing mathematics being built from first principle in the lectures, assisted them with understanding the content of mathematics:

Definitely, seeing the way maths is built from the bottom up is very beneficial to actually understanding content. [56]

Linked to the idea of constructing mathematics from first principles is the comment of a Lean user who was interviewed. The student was previously discussing how in the pen and paper proofs sometimes he struggled as he did not recall the definitions exactly, but also noticed that is not the case when he is proving in Lean.

I guess you could say, like Lean helps with that if you're doing maths in Lean, because it literally lists all the definitions. So, it's like I'm not unconsciously assuming something about a definition that's not true or something like that. But yeah, otherwise that... The other proofs that I can't solve when I do remember the definitions correctly, it's just I'm not seeing some trick I need to do. (Larry – Lean user)

Students also commented about the relationship (or lack of) between writing proofs in pen and paper or writing proofs in Lean. We found at least eight students, among those who gave detailed answers, who were not able to see a connection between coding in Lean and writing a pen and paper proof and for this reason they dismissed Lean as a useful tool and consider it as related only to computer science:

I don't think it would be beneficial [using Lean]. I would rather spend my time doing maths problems by myself, instead of trying to write a code. Lean is useful if you are interested in computer science. [41]

Others linked proving in Lean and proving on pen and paper. In the quote below, a Lean user reflected on the proof of a theorem related to the greatest common divisor which he initially solved in pen and paper and then wrote in Lean. Coding his proof in Lean made this student realize that there were some issues with the pen and paper proof in terms of assuming statements that were not needed:

I actually realised when I wrote it on paper, I was assuming some extra things that I didn't. [...] So, on paper I did it for positive integers when in Lean I just did it for all the integers. Yeah. So, that was cool. (Luke – Lean user)

Another student came to a similar realization. He commented on how he changed his proof-writing practices because of using Lean and how he now considers the special cases that he might have missed in the past. Since Lean code is very detailed and accounts for each possible case, he now uses this practice in his pen and paper proofs too and can understand when he makes this oversight.

When I'm coming up with proofs, thanks to Lean, I see the, like the special cases that I'm not accounting for, like I really break everything down thanks to doing Lean like. Yeah, I see the, like, special cases. Like if I do some assumption, then I see like exactly what does this assumption hold for. Like, or when I do assume a prime is odd like before, like I could easily make the mistake that like not go through the case where the prime is two then it's even. So now I spot that mistake easily. (Larry – Lean user)

The same student also commented on the differences between pen and paper proofs and proofs in Lean in terms of the steps that are needed to be proven and written about in each case, as we can see from the quote below:

Although that's why it exists. Isn't it? Because the computer is better than us at checking. But then I suppose using Lean is the best way of finding the error when Lean doesn't accept your proof. So maybe it isn't. Maybe it doesn't make us better at checking proofs ourselves no. Maybe it doesn't make us better at identifying false proofs, but it does make us better at going through the proof and tweaking it so that's right. I think, I think that that's what it does. (Larry – Lean user)

Another student stated that since starting the module he now considers how they would code a proof into Lean while writing in pen and paper. Note however how the student expressed how he is not yet able to do more complicated proofs in Lean because of a lack of familiarity with the syntax needed.

For pen and paper proofs. Hm.. Let's see. Yeah, I try and think about how I can if I'm writing a proof how I would type that into Lean, but I don't think I know because we're doing quite complex things in math already and I just don't know how the syntax or any of that would work in Lean. So I'm not sure how much of a help that is yet. (Luke – Lean user)

Finally, during the interviews, students referred to their Professor's discussions, taking place during the lectures, of the differences between mathematics and computer science and how they found these remarks interesting and contributing to their understanding of the role of proof in mathematics:

And then also during the lectures, the Professor [...] how to say [...] philosophical remarks that he places here and there about foundations of maths. And he references like type theory a lot... and the difference between like maths proofs and computer science proofs and how they actually sometimes are the same. And also their differences. Yeah, that's very, very thought-provoking, which is good. (Luke – Lean user)

Summarizing this section, some students seemed to be receptive to moving between programming and mathematics and to consider different types of proofs. This appears to contribute to their transition into the rigour of university mathematics. For most students however the difficulties inherent to the complexity of Lean prevented them from actively engaging with the language, which they perceived did not contribute to their mathematics learning and was not to be related to doing mathematics and writing proofs in pen and paper.

6.2.4. Strategic knowledge

There are relatively few utterances linked to strategic knowledge in our data, and they are different between Lean users and non-Lean users. For Lean users, those are mostly related to the use of tactics. Students found it difficult to understand how and when to apply some specific tactics to solve some mathematics problems. The discussion of specific tactics such as the tactic *intro H'* which introduces a hypothesis in the proof and is usually one of the first steps in the proof writing stage, the tactics *linarith*, which is used when solving goals (parts of proof) with linear arithmetic, and *simp* (simplifier), which is used to simplify the main goal using hypotheses and lemmas introduced earlier, dominates this theme:

Introducing H' [38]

Knowing some of the shortcuts like *linarith* and the *simp*. [6]

Other students reported difficulties relate more closely to problem-solving skills in using Lean, as in the extract of the interview below:

But if I use it myself, I can only, I only do what he [the Professor] did but I don't know how to change variables, change statement like that. (Ned – Non-Lean user)

We hypothesize that the relatively little attention to strategic knowledge is linked to the fact that not many students engaged actively with Lean. While it could be easy to appreciate syntactic logic difficulties only by following the Professor programming during the lectures, it may have been necessary to engage actively in Lean programming to appreciate the subtleties connected to strategic knowledge and the use of tactics.

6.2.5. Epistemology – the nature of mathematics

Perhaps the most surprising of the themes emerging from the analysis of the qualitative data is the link that some students seemed to establish between their experiences of Lean and their perceptions of the nature of mathematics. Of the 52 utterances given as answers to Question 8 which mention proof directly, 16 explicitly referred to the newly acquired preoccupation with rigour, and of the necessity for proof. Therefore, at least for some students, the Lean experience allowed them to review their ideas about what mathematics is and helped them moving towards a view of mathematics more relevant to university studies:

It made me re-evaluate my view on maths. [63]

INCREDIBLY [*emphasis in the text*] rigorous building up of mathematical knowledge means my understanding has never been deeper, problem sheets were actually really fun. [4]

More specifically, the first important change is related to the perception of necessity of proof. Some students stated how learning about Lean helped them understand how even simple or obvious facts need to be proved (at least when they started to study more formal mathematics) before they can be accepted:

I enjoyed finding out where the maths I take for granted comes from. I enjoyed learning how to prove things and how much of the maths I do needs proof. [69]

Learn how to prove intuitive things rigorously. [92]

Such impact resonates also in the interviews. Liam (a Lean user) summarized these ideas as follows:

[...] my idea of what proof is and rigour specifically, especially with Lean. [...] But it's really changed since I got here. (Liam – Lean user)

These data seem to indicate that the Lean experience helped at least some of the students to overcome some of the epistemological difficulties with pure mathematics that characterize the transition to university studies.

7. Discussion

The aim of this study is to investigate students' perceptions of and difficulties with the interactive theorem prover Lean in a first-year transition to proof module. The focus on students' perceptions is motivated by the fact that those perceptions play a big role in students' engagement with learning and with the learning resources in a conceptual way (Entwistle & Ramsden, 1983). Because of the nature of the interactive theorem prover used, the investigation focused on students' difficulties with Lean and the impact that using Lean may have on students' perception of mathematics and of proof. Although there are many studies of students' difficulties with programming (see Qian & Lehman, 2017), we believe that the case of interactive theorem provers deserves attention for the educational impact that using such programmes may have on the learning of pure mathematics. The nature of the introduction of Lean in this module – where Lean was a voluntary part of the curriculum, but examples of live coding and of coded proofs were also discussed in the compulsory lectures – allowed students to familiarize themselves with Lean and its functions even if they decided they could not fully engage in the coding activity.

We posed two research questions for this study and below we report our findings. The first research question [RQ1] concerned the difficulties that students encountered when engaging with Lean, and what benefits/drawbacks they perceived this engagement had on their ability to write proofs. The classification by Qian and Lehman (2017) in syntactic, conceptual and strategic difficulties, also discussed in the work of McGill and Volet (1997), was reflected in our data regarding students' difficulties with programming – with some noticeable differences due to the nature of Lean. The focus on students' perceptions also brought to the fore affective reactions to engagement with Lean and reflections on the role and nature of mathematics. Students found that the Lean syntax was difficult to learn, and was a stumbling block for engagement for some, even for those who had previous programming experience. This perceived difficulty gave rise to negative emotions of helplessness and prevented many students to pursue the learning of Lean. Also, the feeling of helplessness that many students experienced reflects the perceived lack of adequate resources available at the time, such as a simple coding manual or a variety of worked-out examples. Moreover, most students felt they did not have the time to engage with the software in an already busy schedule and did not have the motivation to take up a non-compulsory part of the module. These findings regarding difficulties with syntactic knowledge are not new; already in a study describing the inclusion of MATLAB in first-year university mathematics, Tonkes et al. (2005) report that learning the syntax was the first hurdle students had to overcome to engage with the programme.

We did not find many utterances that related to difficulties concerning strategic knowledge and the ones we found related mainly to the use of tactics in Lean. We hypothesize that the absence of many references to strategic knowledge is due to the very low number of

students who took up the voluntary Lean workshops. Strategic knowledge comes into play when actively planning and debugging programmes and it may not be immediately visible to those who did not actively engage, but only saw Lean used in lectures. Therefore, our data do not warrant a finding of what difficulties related to strategic knowledge students may encounter.

The novel findings of our study concern how students referred to conceptual knowledge (in programming) in relation to conceptual knowledge in mathematics, therefore discussing the affordances and drawbacks of this programme in context, as recommended by Qian and Lehman (2017). In our study, the context is the students' first encounter with the requirements for proof in university mathematics. Some students reported a clear impact that learning about proof automation had on their perceptions of what a proof is and of the role that proof has in mathematics. Amongst the areas of impact mentioned in the data there are the recognition of the necessity for proof, the necessity for rigour in mathematics and the realization that mathematics can be built from its first principles. These are well-known stumbling blocks for mathematics students in their first year. Many studies report how students struggle to justify the need to prove statements they are familiar with and see as obvious (e.g. Almeida, 2000), and how students find the rigour required for university mathematics writing difficult to conceptualize and implement in their own mathematics writing (Selden & Selden, 2008).

Moreover, our analysis highlighted that some students in our sample saw how certain characteristics of Lean may be beneficial for their proof-writing with pen and paper. They made a direct link between these two modes of proving, indicating that habits acquired while writing proofs in Lean could also be transferred to writing proofs on pen and paper. One such example is the way in which proofs can be structured. Selden (2011) reports how undergraduate students struggle to organize proofs because they cannot see what the intermediate goals are to eventually prove the main result. Some of the students in our study stated how learning the division in goals and subgoals necessary to programme in Lean helped them acquiring this habit also when writing proofs on pen and paper. Indeed, the study by Thoma and Iannone (2022) offers some empirical evidence that this transfer of habits acquired for Lean programming to proofs written on pen and paper may indeed happen. Lastly, some students emphasized how the distinction between technical and natural language can be clarified using an interactive theorem prover, as Avigad (2019) had also hypothesized.

However, many students did not see the link between pen and paper proofs and Lean proofs and thought that the difficulties that were brought by using the new programme got in the way of their learning mathematics. Those students, who also experienced negative emotions related to Lean, were unsure of the benefits of engaging with the interactive theorem prover. They considered mathematical language completely distinct from programming language, with one student even equating learning to programme a proof in Lean to learning mathematics in a foreign language. This lack of appreciation of the role of the interactive theorem prover paired with the difficulties with the Lean syntax, therefore, prevented many students from engaging with the programme and this may represent a stumbling block for the adoption of this tool. We hypothesize that these negative responses to the use of Lean were motivated by two factors. The first regards the module organization in that Lean was not a compulsory part of instruction and the Lean learning material available at the time of the study was not well developed or linked to the curriculum directly.

The second factor may be linked to the general use of programming in mathematics. As we have seen programming in university (but also in schools in the UK) is mostly taught in the context of computational methods and mathematical modelling (Sangwin & O'Toole, 2017). It may be the case that the students in our sample did not make the link between programming and proving because they were unfamiliar with the nature of interactive theorem provers. Writing a proof is a very distinct activity from mathematical modelling or computational mathematics and students – some of whom were versed in other programming languages – just did not see the link between programming and proof.

With respect to the second research question [RQ2] we observe some impact of the experience of becoming familiar with Lean on students' views of the nature of mathematics, even for students who only saw Lean used in the lectures. Many studies on the transition from school to university mathematics report that students experience an epistemological crisis during their first year at university (e.g. Gueudet et al., 2016). During this transition, students are asked to move from thinking of mathematics as a set of calculations and procedures to thinking of mathematics as an abstract subject where (at least in pure mathematics) proof is the main mechanism to accept or reject a statement⁸ and where proof is obtained by a chain of deductive logical reasoning. For some of the students in our sample, the need for rigour is brought to the fore by the programming experience and is linked to the fact that a mathematics statement can be accepted only if it has been proved rigorously, in this case, if Lean does not return an error message. This is not just a formal requirement for the students, but our data suggest that for some students this becomes an epistemological one. The utterances by the students who engaged actively with Lean may indicate how they now see the requirement for formal proof of all mathematics statements for them to be accepted as true, even those which seem self-evident, and how this requirement for correct deductive reasoning is a fundamental part of mathematics. Indeed, some students reported that their perception of mathematics changed when faced with the requirements of an interactive theorem prover. They also added that they now appreciate the role of proof and abstraction in advanced mathematics, indicating that their experience with Lean may have facilitated this epistemological shift. Therefore, using an interactive theorem prover at the start of a mathematics degree could also support students during the transition to advanced mathematics and help alleviate the epistemological crisis that the students experience in their first year at university which often causes students to drop out from mathematics degrees (Gueudet, 2008).

Summarizing the findings of our study, when students held positive attitudes towards Lean and engaged with the programming language – even if not extensively – we detected an impact on their epistemologies of mathematics and on their perceptions of rigour and necessity of proof. However, our analysis shows that most students were discouraged from using Lean due to its difficult syntax, the lack of teaching material, and the absence of a clear link between coding in Lean and learning mathematics.

8. Concluding remarks and future research

In this paper, we outlined the findings of an exploratory study on students' perceptions of the use of an interactive theorem prover in a first-year mathematics module. The findings show that interactive theorem provers may be useful to alleviate some of the difficulties related to the role and construction of proof that students encounter when joining

a mathematics degree. Introducing interactive theorem provers also gives students much sought-after programming experience. Our data shows however that for some students, who were not able to make the link between a programming language and the activity of proof writing, or that found learning Lean too demanding, the experience brought frustration and ultimately disengagement.

This study has all the limitations of an exploratory study. Engagement with Lean was not a compulsory requirement of the module, although all students were exposed to Lean during the lectures, and this meant that only a few students engaged with it in a consistent way. Another limitation is that the university where the study took place is very high ranking in the UK and the students who access it achieved the highest marks at the end of their school. The findings of the study therefore can be considered in this context, and future studies could investigate whether these findings can be supported also when participants are students who are not enrolled in high-ranking research universities. We can also not ignore the impact that the Professor had on the students' perceptions, his enthusiasm and above all his experience of using Lean in his own research which were reflected in his teaching. Larger studies where an interactive theorem prover is a compulsory part of instruction would help refine and better support our initial findings.

Our data also show the importance of adequate teaching material which is paramount to the success of any intervention. Many of the students' difficulties with the syntactic and strategic knowledge could be alleviated by appropriate learning resources, which were not well developed at the time of the study given the novelty of the use of an interactive theorem prover in undergraduate teaching. As the use of interactive theorem provers in teaching mathematics becomes more widespread, steps are made to produce teaching material that can help students overcome the first difficulty with syntactic knowledge. The Natural Number Game,⁹ for example, takes the students through the construction of the natural numbers – a mathematics topic familiar to them – while learning Lean. Finally, we believe that future research should focus on the investigation of the reasoning skills that students employ when programming proofs with an interactive theorem prover and how these relate to reasoning skills related to proof production.

Notes

1. University degrees in the UK are typically modularized. To progress from one year to the next, each student needs to accrue a certain number of credits by completing appropriate modules. Each module has between 3 and 4 contact hours per week, depending on the stage at which it is offered, and can be compulsory or optional. The number of weeks in a module vary between universities – in our study the module was 11 weeks long.
2. Interactive theorem provers are sometimes also called 'automated theorem provers'. The first naming stresses the interactive nature of the proof writing, while the latter the capacity of the software to eventually produce proofs. Given the stage of the development of Lean and the use the students made of the tool we have adopted the first name.
3. The screenshots are taken from the Natural Number Game (https://www.ma.imperial.ac.uk/buzzard/xena/natural_number_game/).
4. The version used in this module was lean v3.
5. We note that no field in the questionnaire was mandatory so response numbers may not always add to 99.

6. To distinguish between the quotes coming from the interviews and the questionnaire responses we use different labels. We use pseudonyms when the quote is coming from the interview data and numbers, 1–99, when we report utterances from the questionnaire.
7. The mathematics in Lean is underpinned by Type Theory and not by Set Theory as the mathematics generally taught at university.
8. Of course, we exclude axioms from this consideration.
9. https://www.ma.imperial.ac.uk/buzzard/xena/natural_number_game/.
10. Module code, redacted to preserve confidentiality.

Acknowledgements

We would like to thank the Professor and the students who participated in our study for giving up some of their time to take part in the research.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by Imperial College London.

ORCID

Paola Iannone  <http://orcid.org/0000-0001-7904-5380>

References

- Almeida, D. (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. <https://doi.org/10.1080/00207390050203360>
- Artigue, M. (2016). Mathematics education research at university level: Achievements and challenges. In V. Durand-Guerrier, R. Hochmuth, E. Nardi, & C. Winsløw (Eds.), *Research and development in university mathematics education* (pp. 11–27). Taylor and Francis.
- Avigad, J. (2019). Learning logic and proof with an interactive theorem prover. In G. Hanna, D. Reid, & M. de Villiers (Eds.), *Proof technology in mathematics research and teaching* (pp. 277–290). Springer.
- Bond, P. (2018). The era of mathematics. *UK Research and Innovation*. <https://www.eu-maths-in.eu/wp-content/uploads/2018/05/EraOfMathematicsReport.pdf>
- Bruijn, d. N. G. (1980). A survey of the project Automath. In J. P. Seldin, & J. R. Hindley (Eds.), *To H. B. Curry: Essays on combinatory logic, lambda calculus and formalism* (pp. 579–606). Academic Press.
- Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2020). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*, 51(7), 1020–1041. <https://doi.org/10.1080/0020739X.2019.1648892>
- Buteau, C., Jarvis, D. H., & Lavicza, Z. (2014). On the integration of computer algebra systems (CAS) by Canadian mathematicians: Results of a national survey. *Canadian Journal of Science, Mathematics and Technology Education*, 14(1), 35–57. <https://doi.org/10.1080/14926156.2014.874614>
- Buzzard, K. (2020). Proving theorems with computers. *Notices of the American Mathematical Society*, 67(11), 1791–1799. <https://doi.org/10.1090/noti2177>
- Crowe, D., & Zand, H. (2001). Computers and undergraduate mathematics 2: On the desktop. *Computers & Education*, 37(3–4), 317–344. [https://doi.org/10.1016/S0360-1315\(01\)00057-4](https://doi.org/10.1016/S0360-1315(01)00057-4)

- Dawkins, P. C., & Weber, K. (2017). Values and norms of proof for mathematicians and students. *Educational Studies in Mathematics*, 95(2), 123–142. <https://doi.org/10.1007/s10649-016-9740-5>
- Dubinsky, E. (1995). ISETL: A programming language for learning mathematics. *Communications on Pure and Applied Mathematics*, 48(9), 1027–1051. <https://doi.org/10.1002/cpa.3160480905>
- Entwistle, N., & Ramsden, P. (1983). *Understanding student learning*. Croom Helm.
- Gueudet, G. (2008). Investigating the secondary–tertiary transition. *Educational Studies in Mathematics*, 67(3), 237–254. <https://doi.org/10.1007/s10649-007-9100-6>
- Gueudet, G., Bosch, M., DiSessa, A. A., Kwon, O. N., & Verschaffel, L. (2016). *Transitions in mathematics education*. Springer Nature.
- Hanna, G. (1990). Some pedagogical aspects of proof. *Interchange*, 21(1), 6–13. <https://doi.org/10.1007/BF01809605>
- Hanna, G., & Yan, X. (2021). Opening a discussion on teaching proof with automated theorem provers. *For the Learning of Mathematics*, 41(3), 42–46.
- Iannone, P., & Simpson, A. (2022). How we assess mathematics degrees: The summative assessment diet a decade on. *Teaching Mathematics and its Applications: An International Journal of the IMA*, 41(1), 22–31. <https://doi.org/10.1093/teamat/hrab007>
- Lee, K., & Smith III, J. P. (2009). Cognitive and linguistic challenges in understanding proving. In *Proceedings of the ICMI Study 19 Conference: Proof and Proving in Mathematics Education*, May 2008, 2, 21–26.
- Lew, K., & Mejía-Ramos, J. P. (2019). Linguistic conventions of mathematical proof writing at the undergraduate level: Mathematicians’ and students’ perspectives. *Journal for Research in Mathematics Education*, 50(2), 121–155. <https://doi.org/10.5951/jresmetheduc.50.2.0121>
- Lockwood, E., & De Chenne, A. (2019). Enriching students’ combinatorial reasoning through the use of loops and conditional statements in python. *International Journal of Research in Undergraduate Mathematics Education*, 6, 303–346. <https://doi.org/10.1007/s40753-019-00108-2>
- Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in rume. *International Journal of Research in Undergraduate Mathematics Education*, 7, 404–416. <https://doi.org/10.1007/s40753-020-00129-2>
- Lynch, S. (2020). Programming in the mathematics curriculum at Manchester Metropolitan University. *MSOR Connections*, 18(2), 5–12. <https://doi.org/10.21100/msor.v18i2.1105>
- Marton, F., & Säljö, R. (1997). Approaches to learning. In F. Marton, D. Hounsell, & N. Entwistle (Eds.), *The experience of learning. Implications for teaching and studying in higher education* (pp. 36–55). Scottish Academic Press.
- McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing students’ knowledge of programming. *Journal of Research on Computing in Education*, 29(3), 276–297. <https://doi.org/10.1080/08886504.1997.10782199>
- Mok, I. A. C., & Clarke, D. J. (2015). The contemporary importance of triangulation in a post-positivist world: Examples from the learner’s perspective study. In A. Bikner-Ahsbabs, C. Knipping, & N. Presmeg (Eds.), *Approaches to qualitative research in mathematics education. Advances in mathematics education* (pp. 403–425). Springer. https://doi.org/10.1007/978-94-017-9181-6_15
- Moore, R. C. (1994). Making the transition to formal proof. *Educational Studies in Mathematics*, 27(3), 249–266. <https://doi.org/10.1007/BF01273731>
- Qian, Y., & Lehman, J. (2017). Students’ misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1), 1–24. <https://doi.org/10.1145/3077618>
- Saldaña, J. (2021). *The coding manual for qualitative researchers*. Sage.
- Sangwin, C. J., & O’Toole, C. (2017). Computer programming in the UK undergraduate mathematics curriculum. *International Journal of Mathematical Education in Science and Technology*, 48(8), 1133–1152. <https://doi.org/10.1080/0020739X.2017.1315186>
- Selden, A. (2011). Transitions and proof and proving at tertiary level. In G. Hanna, & M. De Villiers (Eds.), *Proof and proving in mathematics education: The 19th ICMI study* (pp. 391–420). Springer.
- Selden, A., & Selden, J. (2008). Overcoming students’ difficulties in learning to understand and construct proofs. In M. P. Carlson, & C. Rasmussen (Eds.), *Making the connection: Research and teaching in undergraduate mathematics* (pp. 95–110). (No. 73). Mathematics Association of America.

- Skemp, R. (1978). Relational understanding and instrumental understanding. *The Arithmetic Teacher*, 26(3), 9–15. <https://doi.org/10.5951/AT.26.3.0009>
- Thoma, A., & Iannone, P. (2022). Learning about proof with the theorem prover Lean: The abundant numbers task. *International Journal of Research in Undergraduate Mathematics Education*, 8(1), 64–93. <https://doi.org/10.1007/s40753-021-00140-1>
- Tonkes, E., Loch, B. I., & Stace, A. (2005). An innovative learning model for computation in first year mathematics. *International Journal of Mathematical Education in Science and Technology*, 36(7), 751–759. <https://doi.org/10.1080/00207390500271677>

Appendix

Below are the questions included in the questionnaire.

- (1) Please tell us your gender (drop-down menu: Female, Male, Prefer not to say)
- (2) Reflecting on the whole term, please write two aspects of the¹⁰ course that you enjoyed, if any.
- (3) Reflecting on the whole term, please write two aspects of the course that you found challenging if any.
- (4) What changes would you recommend for the course, if any?
- (5) How would you describe your familiarity with Lean? (Drop-down menu: Not at all familiar, Slightly familiar, Somewhat familiar, Moderately familiar, Extremely familiar)
- (6) What did you find interesting (or not) about the use of Lean in the lectures?
- (7) What did you find challenging (or not) about the use of Lean in the lectures?
- (8) Do you think that using Lean would be beneficial (or not) for you and why?
- (9) If you have attempted to use Lean but then stopped, what was the main reason for this? (Drop-down menu: Time constraints, Difficulty with the programming language, Not interesting enough, Not applicable – I am still using Lean, Other. For the choice of other there was the possibility of writing in a blank field).
- (10) Do you have any further comments that you would like to make?