Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis

Evmorfia Bartzia, Antoine Meyer, Julien Narboux

Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis

Evmorfia Iro Bartzia[1], Antoine Meyer[2], and Julien Narboux[3]

[1]IMAG, Université de Montpellier, CNRS, Montpellier, France; [2]LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454 Marne-la-Vallée, France; [3]UMR 7357 CNRS, University of Strasbourg, France

This paper presents an a-priori analysis of the use of five different interactive proof assistants for education based on the resolution of a typical undergraduate exercise on abstract functions. It proposes to analyse these tools according to three main categories of aspects: (1) language and interaction mode, (2) automation and user assistance, (3) proof structure and visualisation. We argue that this analysis may help formulate and clarify further research questions on the possible impact of such tools on the development of reasoning and proving skills.

Keywords: Teaching and learning of logic, reasoning and proof, Digital and other resources in university mathematics education, Transition to, across and from university mathematics, Novel approaches to teaching, Computer assisted theorem proving.

INTRODUCTION

Investigating the use of technology for the teaching and learning of proof and proving is an active topic in both communities of education research and computer-assisted theorem proving. The topic of proof has been garnering interest in the mathematics education research community for years. The 19th ICMI Study focused on six major themes relative to the teaching and learning of proof and proving. It led to the publication of a study volume (Hanna & de Villiers, 2012) with contributions from specialists of the field providing insight on these themes. The interactive theorem proving community has also shown interest in the use of proof assistants for teaching since at least 2007 and the workshop on Proof Assistants and Types in Education (Geuvers & Courtieu, 2007), followed since 2011 by the ThEDU workshop series.

Recently Hanna and de Villiers, together with Reid, coordinated another volume specifically focusing on the use of software tools for computer-assisted proof in education (Hanna et al., 2019), featuring contributions from researchers in both communities. In the introductory chapter they state that the book's goal is "to begin a dialogue between mathematics educators and researchers actively developing automatic theorem provers and related tools". The chapter concludes with the statement that "we know almost nothing of [proof assistants'] potential contribution to other roles of proof, such as explanation, communication, discovery, and systematization, or how they now may become more relevant as pedagogical motivation for the learning of proof in the classroom", implying that much research is still required in order to gain further insight on the convergence of both fields.

Proof assistants (henceforth PAs) are quite broadly used to teach logic, proof of computer programs and, increasingly, classical mathematical topics by teachers who are not researchers in the field of interactive theorem proving. In this paper we focus on the potential use of these tools for teaching proof and proving itself at the transition between high school and university. In this work we consider proof assistants as possible teaching tools and not as professional tools. The tool's underlying proof theory and the structure and size of its mathematical libraries are therefore not directly relevant. We will instead focus on the way each tool enables the development of skills related to proof and proving.

The questions which motivate this work can be phrased as follows:

- What are the possible effects of using PAs on students' learning of proof?
- What characteristics of PAs are likely to strengthen or hinder these effects?

In order to start addressing these questions we chose to analyse the resolution of a single typical exercise about functions using a selection of five different PAs (Coq, Isabelle, Edukera, dEAduction and Lurch, introduced briefly below). We solved this exercise using each PA in turn, with one experimenter building the proof interactively and two observers. Based on initial observations we designed an analysis grid to capture some of the tools' characteristics likely to have an impact on teaching and learning. We then revisited each resolution of the exercise and analysed it with respect to this grid. Our aim is to help distinguish aspects of each PA which may facilitate or hinder student's learning of the various skills involved in proof and proving (Selden, 2012) as a preliminary step to future research.

We first briefly introduce the concept of PA. We then present our case study before describing our analysis grid. We finish by raising additional questions regarding the possible impacts of each PA on the teaching and learning of proof and proving.

PROOF ASSISTANTS IN EDUCATION

The term *proof assistant*, or *interactive theorem prover*, refers to a software tool allowing a user to interactively construct a formal mathematical proof. Some systems are designed to work in a specific domain such as geometry, logic or the analysis of computer programs, while others are general-purpose. Additionally, proof assistants used in the classroom can be sorted roughly in two categories: some are built by the community of educators and others are designed by specialists of interactive theorem proving for research or other professional purposes.

The input languages of PAs are usually classified into two categories: *imperative* and *declarative* languages. In an imperative language the user orders changes to be performed on the *proof state* (the current set of declared variables and constants, assumed hypotheses, and goals) using a predefined set of orders (also called *tactics*). Each tactic consists in one or several deduction rules to be applied, or other manipulations of the proof state. Most tactics do not contain explicit mathematical statements. In a declarative language one provides assertions along with their

justification, in a way similar to a natural-language proof. The statements are therefore written explicitly, using a syntax resembling mathematical language.

In simple cases the validity of each proof step is ensured by matching some of the available statements with the premise of a given deduction rule, substituting variables accordingly in the rule's conclusion, and verifying that each involved expression is well-typed. This may be complemented with other automation techniques, for instance to help searching for applicable rules, to perform automatic computations in restricted domains (arithmetic, algebra…), to assist in syntactic manipulations, etc.

In the terminology of Duval & Egret (1993), most PAs clearly distinguish the theoretical status (hypothesis, axiom, definition, theorem, conjecture) and operational status (premise, conclusion, external rule, goal) of each statement. This is done using visual hints, the syntax of the PA's language, or by separating statements between disjoint areas of the user interface. This may be an important feature in an educational context, since this distinction is known to be a source of difficulty for students.

We will revisit these characteristics below, when we detail the aspects which constitute our analysis grid, and illustrate them on our selection of proof assistants.

CASE STUDY: ANALYSING AN EXERCISE IN FIVE PROOF ASSISTANTS

The exercise we chose for this work is a typical elementary proof about sets, relations and functions commonly found in introductory courses about reasoning and proof, in both Mathematics and Computer Science curricula, and available or formalisable in all studied PA. The exercise text reads as follows, with minor variants:

1 Given $f : A \to B$ and $C \subseteq A$, show that $C \subseteq f^{-1}(f(C))$.

2 Given $f : A \to B$ and $C \subseteq A$, show that if f is injective then $f^{-1}(f(C)) \subseteq C$.

We chose this exercise because it involves few and fundamental mathematical concepts and little calculation. The required proofs are of a manageable size, yet not trivial for students. They involve the concepts of set, function, subset relation, direct and inverse image and injectivity. The definitions of these concepts require universal and existential quantifiers and implication, which students tackling the proof are required to be able to manipulate.

We now briefly describe the five PAs we chose to analyse in this work. Coq and Isabelle are professional systems which are also used for teaching. Lurch, Edukera and dEAduction were designed specifically for teaching.

Coq and *Isabelle* are free and open-source proof assistants. Coq was created in the 1980s in French academia (Coq Team, 2022). Isabelle was developed at University of Cambridge and Technische Universität München (Nipkow et al., 2002). Both have been used successfully to prove mathematical theorems such as the Feith-Thompson theorem, the four-colour theorem or the Kepler conjecture, and to prove the correctness of large-scale computer programs. They have also been experimented for several years as teaching tools in graduate or undergraduate curricula on various

topics. A difference between the two lies in the kind of user interaction and language they offer. In this work, we only use Coq in imperative mode, and Isabelle in declarative mode using its Isar language.

Lurch is a free and open-source word processor built on OpenMath, that can check the steps of a mathematical proof (Carter & Monks, 2013). Lurch is designed for student use and was experimented for teaching in 2008 and 2013. To our knowledge, it is no longer maintained, but was kept under consideration due to its originality with respect to other PA. Its user interface is inspired by that of a word processor, proof checking being presented similarly to spell-checking: one can write text freely, then mark some mathematical expressions as meaningful and check their validity.

Edukera is a closed-source web-based graphical proof assistant loosely based on Coq (Rognier & Duhamel, 2016). It is no longer maintained but was kept in our study for the same reasons as Lurch. It was designed to help teach proof and proving as well as classical high school mathematics content including algebra and basic analysis. Its originality is to combine a point-and-click interface with a presentation of the whole proof mimicking human-written text.

DEAduction [2] is a recent free and open-source graphical interface to the LEAN proof assistant created by Frédéric Le Roux. It was specifically designed for teaching, and is under active development. It provides a purely point-and-click user interface.

By lack of space we cannot provide here a full account of the proofs of the exercise in each PA, but we will give additional details during the presentation. Interested readers may download proof files for this exercise in each studied PA online [1].

ASPECTS OF PROOF AND PROVING IN PROOF ASSISTANTS

In this section we describe the three main categories of aspects of PA we retained in our analysis, each including several criteria which are summarised in Table 1. Other factors of practical importance are left out of this study, such as type of license, availability, ease of installation, integration with learning management systems, etc.

Language and interaction mode	type of user input, imperative or declarative style, object naming, possibility of writing ill-formed statements
Automation and user assistance	mathematical libraries, rule selection and application, scope management, rule chaining and automated computation, type of feedback
Proof structure and proof state visualisation	global or local viewpoint on proof, status of statements, possibility to create new definitions and lemmas

Table 1: Categories of aspects of proof assistants and related analysis criteria

Language and interaction mode

The first category we consider relates to the nature of interactions between user and proof assistant. We focus in particular on the tools' linguistic, semiotic and visual characteristics. This includes the syntax and semantics of the input language, if any, the textual, graphical or mixed output language displayed by the PA, and more generally any kind of visual hints which carry proof-related meaning.

Type of user input. Interactions between the user and a PA generally include both mouse-based and text-based modalities, to varying degrees. In dEAduction and Edukera most interaction is mouse-based (through menus, buttons, drag-and-drop), textual input being only rarely required (for instance when introducing an existential witness). In Coq and Isabelle the user respectively types in tactics or proof text, both obeying a strict syntax. In Lurch the user experience is similar to that of "literate programming" where code is mixed with explanatory text. By default natural-language text is ignored and carries no semantics. "Meaningful expressions", whose syntax resembles that of standard mathematics, are then combined with one another to form deduction steps, which are then formally checked by the software.

Imperative or declarative style. Coq is an example of an imperative language. The user types in *tactics* which perform transformations of the current proof state. In Question 1 of the analysed exercise, to prove that $f(x) \in f(C)$ the user runs the command `unfold im`, which instructs the prover to unfold the definition of an element being in $f(C)$, yielding as new goal $\exists x_0 \left(x_0 \in C \wedge f(x) = f(x_0) \right)$. In Isabelle the language is declarative: at every step the user has to declare what will be proved, i.e. she has to state how the goal will be transformed after she applies the next proof step. Assuming the hypothesis $x \in C$, referred to by label `Hx`, is available in the current scope, the user may type: `have "f x f ` C" using Hx by (rule imageI)`. This line attempts to prove $f(x) \in f(C)$ using the hypothesis $x \in C$ and the definition of the image of a set (`imageI`). Deduction steps in Lurch have a similar structure. In Edukera the user simply clicks the "def" button while the goal is selected and the definition of the image of a set is unfolded automatically. In dEAduction the user has to select the appropriate definition from a predefined list.

Object naming and referencing. In Coq and Isabelle the user can choose the names of hypotheses and objects when they are introduced. In Edukera, each line of the proof is automatically numbered, and is referenced whenever it is used as a premise in a deduction step. In dEAduction variables and hypotheses are automatically assigned fresh names (i.e. not bound in the current context). In Lurch one can choose custom labels for statements. Even though each tool offers different presentation choices and interaction styles, being able to refer to objects by name is essential to the structuration of a proof.

Automation and user assistance

Automation refers to all features facilitating the selection of a usable rule in a given context, the syntactic manipulation of statements (in particular regarding type checking, substitution and pattern matching), the chained application of rules, etc. Other features include the organised presentation of available rules and theorems, automatic scope management, and contextual hints or feedback. According to some PA designers' and teachers' testimonies, finding a good balance in the level of automation is a challenge, especially in an educational context where efficiently completing a proof may not be the main goal.

Mathematical libraries. Contrary to traditional proofs, most PAs provide libraries of definitions and theorems, and make their formal definitions easy to access. PAs may also provide additional assistance such as contextual search, automatic completion, online help, etc. Professional PAs like Coq and Isabelle provide thousands of proven mathematical facts. Edukera and dEAduction simply list predefined lemmas and definitions, sorted by topic, not all of which are available in every exercise.

Rule selection and application. One of the main actions when building a proof in a PA consists in performing a reasoning step by applying a theorem or a logical rule, or by substituting a symbol by its definition. Each PA provides a different level of assistance and automation for these tasks, mainly regarding the way a rule or statement is instantiated when it is used (i.e. its variables substituted by terms), or the way a given rule, theorem or definition is selected with respect to the current context. In Isabelle and Lurch, the user writes instantiated mathematical expressions, and explicitly invokes a rule by its name. The tool then checks that this instantiation is correct, and if so applies the rule. In Lurch, multiple rules may share the same name, in which case all matching rules are tried in order until one succeeds. In Coq, dEAduction and Edukera, commands to unfold a definition or apply a theorem are provided, either by invoking them by name or by selecting them from a list. In all three systems, pattern matching and substitution are performed automatically. For logical deduction rules, a varying degree of automation is offered. In some of the tools (Coq and Edukera in maths mode), generic commands are available to eliminate or introduce logical connectors and quantifiers. Only when ambiguity occurs is the user required to add input. In other tools, the user generally has to determine the outermost logical connective themself.

Scope management. According to teacher testimony and previous research on proof, keeping track of the scope of each variable or hypothesis is a source of difficulty for students, which sometimes leads to confusion between free and bound variables, or to circular arguments. In Coq and dEAduction, scope management is fully automatic and available variables and hypotheses are neatly gathered in corresponding areas of the interface. In Edukera, unproven statements are clearly distinguished from proved ones, scopes are visually materialised and can be selected when introducing new

variables or hypotheses. Isabelle and Lurch also have syntactic or visual means to indicate scopes, but more work is left to the user to maintain them.

Rule chaining and automated computation. Some PAs offer possibilities for implicit or explicit "chaining" of rule applications. For instance, when applying a universally quantified theorem, Edukera offers to perform the introduction of the universal quantifier and introduction of implication in a single step. Coq also supports implicit chaining of rules: for example, a single invocation of the `apply` command to deduce $x=x'$ from the hypothesis $f(x)=f(x')$ using the injectivity of f successively unfolds the definition of injectivity, eliminates two universal quantifiers and one implication, and performs the associated pattern matching and substitution steps. Other tactics in Coq or Isabelle may perform further automatic transformations. Finally, in specific mathematical areas such as basic arithmetic or linear algebra, PAs may give access to fully automatic solvers, for instance when checking simple equations.

Type of feedback. Feedback varies from basic to very rich. In Coq and Isabelle little feedback is given, apart from error messages when a rule does not apply or when an expression is not well-typed. On the contrary, feedback in Lurch is very rich: there is a colour code to indicate the status of each statement (undischarged hypothesis, valid or invalid conclusion) and visual hints to highlight the scopes of hypotheses. Moreover, very complete feedback on rule application is provided, including a list of selected premises and an explicit substitution of variables.

Proof structure and proof state visualisation

This final category concerns the aspects of a PA related to how proofs are perceived and manipulated. There are two main design choices: in some PA, the whole proof text is visible at once, and users complete it by inserting new assertions. Work may be done progressively on several parts of the proof. In others, only the current goal and the current proof state is prominently displayed. Other aspects related to proof structure concern the users' possibility to decompose a long proof by writing down and separately proving intermediate definitions theorems which can then be reused.

Global vs local viewpoint on proof. In Coq or dEAduction, the user may visualise the sequence of invoked tactics and navigate through them to view the evolution of the proof state at each point. The proof as a whole is left implicit, it is never displayed entirely [3]. Moreover, the origin of each statement in the context (hypothesis of the theorem to be proven, previously proved fact, hypothesis in a proof by cases or by contradiction) is not displayed. In both tools, it is also natural to treat the goals in the order in which they are generated by the system. One may say the viewpoint on proof is local, with much information hidden. On the contrary, in Edukera or Lurch (or in a pen-and-paper proof), the proof state is implicit: it is composed of the list of open statements combined with the list of hypotheses which are assumed to hold in the scope of each open statement. Due to their declarative style and since proof texts in these two PAs rather closely imitates usual mathematical language, they offer a more global viewpoint on proofs without resorting to back-and-forth navigation through

proof lines. Isar (Isabelle's language) combines both aspects by allowing both a complete, more or less human-readable proof text, and the ability to display the current proof state at each line of the proof.

Possibility to create new definitions and lemmas. DEAduction and Edukera do not allow the user to create new definitions or theorems, the user is on a "deductive island" imposed by the system. In Edukera, teachers can compose their own exercise sheets but they cannot create new exercises. Developing new theories is not possible for end users. Using Coq, Isabelle, or Lurch, the user is free to restructure her proof by introducing new lemmas or concepts.

Status of statements. As already stated, one may distinguish the *theoretical status* of a statement (axiom, lemma, hypothesis, conjecture, etc.) and its *operational status* (premise, conclusion, external statement) which may vary in the course of a proof: a statement may be the conclusion of a deduction step and the premise of another one. The status of statements is rather clear in all PAs (except Edukera where admitted lemmas/axioms, and proved lemmas are not distinguished). In DEAduction hypotheses of the exercises and other elements of the context are displayed in separate frames. Moreover, hypotheses used at least once as premises are greyed out. In Isabelle, local hypotheses introduced to prove universally quantified implications are syntactically distinguished. In Lurch, the validity of each step is displayed using a colour code. The operational status of statements is displayed using "bubbles".

POSSIBLE IMPACTS ON THE TEACHING AND LEARNING OF PROOF

As their name suggests, proof assistants relieve the user of some of the tasks usually associated with proving. While this may be desirable in a professional setting, it might become a hindrance when the goal is precisely to let students practice some of these tasks. Based on our analysis, we formulate a few hypotheses on the possible effects of the use of PAs in teaching regarding various possible teaching goals.

Possible effects on memorisation and formulation. When asking students to solve an exercise on functions, a possible prerequisite or desired learning outcome is that students intuitively understand relevant definitions (in our case those of set and function, set inclusion, direct and inverse image, and the notion of injectivity) and be able to state (and use) their formal definition. When using a PA where details of definitions and properties are always at hand, one may postulate that memorisation of formal statements is not required to "solve" the exercise. Rather, students may be required to read, understand and appropriately make use of them. However, it might be the case that being repeatedly presented with definitions and properties and putting them to use may actually help memorise them.

Possible effects on manipulation of formal statements. It has been remarked that performing substitution is one of the many difficulties of the proving activity. As we observed, the five PAs we studied differ in the way they automate the manipulation of formal statements. In three cases (dEAduction, Edukera and Coq), it may be possible to achieve a complete proof without actually having to write a single

mathematical statement. Coq and Edukera automatically identify the outermost operator in a mathematical term. While this does not completely exempt the user from thinking about statements and anticipating which rules may be used next, it is not up to the user to actually figure out which substitution makes a statement match a given pattern, or how to apply it to another statement in order to use a rule. This is not the case in Isabelle and Lurch, where the user explicitly writes down mathematical terms, and the system simply checks if they are correct. In all cases however, *a posteriori* control and validation is possible, for example by replaying a step in imperative PAs. This may provide another way to practise skills related to formula manipulation, by reading and control rather than by writing. A related possible effect is that PAs may forbid certain incorrect manipulations, produce correct but unexpected outcomes or provide additional feedback (Lurch in particular provides rich and explicit feedback on substitutions). These retroactions are of course unavailable in a pen-and-paper proof.

Possible effects on the perception of proof structure. In our experience, users of imperative-style proof assistants such as dEAduction, Edukera and Coq may feel as though they are "pushing symbols around until it works", possibly not understanding why the proof went through. This may be strengthened by the fact that these tools automatically manage scopes and contexts, including the identification of each statement's operational status. Even though replaying previous steps is possible, these tools may act as "blinders", allowing one to entirely focus on the current proof state, possibly "forgetting" about other parts of the proof. This "tunnel" effect may even be strengthened by the fact that these tools automate several aspects of proving, which makes a trial-and-error exploration strategy more viable than in declarative-style PAs such as Lurch and Isabelle. Edukera stands out as a special case in that the whole text of the proof remains visible throughout, even though user input is mostly imperative and syntactic manipulations largely automated.

As we can see, different design choices in each PA entail different actions on the part of the user. Certain concepts (for instance that of substitution) intervene in all cases but quite differently, and may require different levels of proficiency from the user. One may argue that freeing students from certain tasks (writing syntactically correct statements, keeping track of variables and hypotheses' status and scope, memorising definitions and theorems, recalling what remains to be proven) may enable them to concentrate on the deeper ideas involved in a proof, and may contribute in overcoming these difficulties outside of the PA by mere "habituation". Conversely, one may object that acquiring these skills is indeed one of the intended goals of these activities, and that it is therefore essential to have students practise them and not rely on a tool's facilities. For a discussion of the possible effects of using a PA on the acquisition of proving skills, see for example (Thoma & Iannone, 2021).

Figuring out the actual effect of each PA on learning would of course require further research. It would be interesting to try and analyse student's proficiency with various proof-related competencies when using different types of PAs. Do PAs have an effect

on known syntactic and semantic difficulties that students typically encounter when working on proof? Do they favour the development of higher-level competencies such as writing a full and correct proof on paper, or summarising the main arguments of a proof verbally? How much do these effects depend on students' backgrounds?

NOTES

1. See https://github.com/jnarboux/PA_a_priori_analysis for screenshots and source files in all PAs.

2. Deaduction website, including source code: https://github.com/dEAduction/dEAduction

3. Readers unfamiliar with PAs may consult the following site for examples of proof scripts along with corresponding proof states: https://plv.csail.mit.edu/blog/alectryon.html#alectryon.

REFERENCES

Carter, N. C., & Monks, K. G. (2013). Lurch: A word processor built on OpenMath that can check mathematical reasoning. *Joint Proceedings of the MathUI, OpenMath, PLMMS and ThEdu Workshops and Work in Progress at CICM co-located with Conferences on Intelligent Computer Mathematics (CICM 2013).*

Duval, R., & Egret, M.-A. (1993). Introduction à la démonstration et apprentissage du raisonnement déductif. *Repères-IREM*, 12, 114–140.

Geuvers, H., & Courtieu, P. (2007). *Proceedings of International Workshop on Proof Assistants and Types in Education (PATE07)*.

Hanna, G., & de Villiers, M. (Eds.). (2012). *Proof and Proving in Mathematics Education*. Springer.

Hanna, G., Reid, D. A., & de Villiers, M. (Eds.). (2019). *Proof Technology in Mathematics Research and Teaching*. Springer.

Leron, U. (1983). Structuring Mathematical Proofs. *The American Mathematical Monthly*, 90(3), 174–185.

Nipkow, T., Paulson, L. C., & Wenzel, M. (2002). *Isabelle HOL: a Proof Assistant for Higher-Order Logic*. Springer.

Rognier, B., & Duhamel, G. (2016). Présentation de la plateforme edukera. In J. Signoles (Ed.), *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*.

Selden, A. (2012). Transitions and Proof and Proving at Tertiary Level. In G. Hanna & M. de Villiers (Eds.), *Proof and Proving in Mathematics Education* (pp. 391–420). Springer.

The Coq Development Team. (2022). *The Coq Proof Assistant (8.15)*. Zenodo.

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*, 64–93.