Proof assistants for undergraduate mathematics education: elements of an a priori analysis

Evmorfia-Iro Bartzia, Emmanuel Beffara, Antoine Meyer, Julien Narboux

Proof assistants for undergraduate mathematics education: elements of an a priori analysis

Evmorfia-Iro Bartzia[a] and Emmanuel Beffara[b] and Antoine Meyer[c] and Julien Narboux[d]

[a]IMAG, Université de Montpellier, CNRS, Montpellier, France;
[b]Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France;
[c]LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454, Marne-la-Vallée, France;
[d]UMR 7357 CNRS, University of Strasbourg, France

ARTICLE HISTORY
Compiled April 29, 2023

ABSTRACT
This paper presents an *a priori* analysis of the use of six different interactive proof assistants for education, based on the resolution of a typical undergraduate exercise on abstract functions. It proposes to analyze these tools according to three main categories of aspects: language and interaction mode, automation and user assistance, and proof structure and visualization. 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; logic; reasoning; proof; university; computer assisted theorem proving

1. Introduction

The terms *proof assistant* (PA), or *interactive theorem prover* (ITP), refer to a category of software tools designed to let a user interactively construct and verify the correctness of formal mathematical proofs. In general a PA does not directly enable the user to *discover* a proof, but rather helps them formalize an existing proof or proof sketch, which can then be *checked* automatically to ensure its correctness. In particular, one should not confuse PAs with *automated* theorem provers, whose function is to attempt to automatically discover proofs[1].

Some PAs are designed to work in a specific domain such as geometry, logic or the analysis of computer programs, while others are general-purpose. Additionally, PAs used in the classroom can be sorted roughly in two categories: systems built by the community of educators, and systems built by mathematicians, logicians or computer scientists specializing in mathematics formalization and interactive theorem proving. Systems developed by educators are sometimes referred to as 'tutors', because they integrate features specifically designed to provide hints and learning-oriented feedback.

CONTACT Email: evmorfia-iro.bartzia@umontpellier.fr; emmanuel.beffara@univ-grenoble-alpes.fr; antoine.meyer@univ-eiffel.fr; narboux@unistra.fr

[1]Of course, both types of tools may indeed be related. For instance, some PAs may use automated provers in specific proof steps.

PAs have been used for years in education in a variety of contexts, for instance to teach mathematical logic or elements of theoretical computer science. The well-know *Software Foundations* electronic book series (Pierce et al., 2022, and following volumes) presents a wide range of 'one hundred percent formalized and machine-checked' texts related to software reliability, written using Coq, which are designed for 'advanced undergraduates to PhD students and researchers' and are used in several computer science departments worldwide. There are many other such examples. Increasingly, PAs are also used by teachers who are not researchers in the field of interactive theorem proving, to teach classical mathematical topics or even proof and proving in general. In the context of French academia, a joint publication by university teachers (Kerjean et al., 2022) provides assorted recent examples of PA usage in first year courses.

In this context, it is not surprising that investigating the use of technology in general (and PAs in particular) for the teaching and learning of proof and proving has become an active topic in mathematics education research. For instance, the 19th ICMI Study distinguished six major themes relative to this topic (Hanna & de Villiers, 2012). Knobelsdorf, Frede, Böhne, and Kreitz (2017) present a preliminary study, based on socio-cultural cognition theories, investigating whether 'an interactive theorem prover like Coq can be used to help undergraduate computer science students learn mathematical proving within the field of theory of computation'. Symmetrically, the interactive theorem proving community has shown interest in the use of PAs 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[2].

Recently, another volume featuring contributions from researchers of both communities focused specifically on the use of software tools for computer-assisted proof in education (Hanna, Reid, & de Villiers, 2019). In the introductory chapter, it is stated 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 ends with the following statement:

> We already have some sense that proof assistants greatly diminish the need for verification and justification, but we know almost nothing of their 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. (Hanna et al., 2019, p. 9)

implying that much research is still required in order to gain further insight on the convergence of both fields. Hanna and Yan (2022a) and Thoma and Iannone (2022) made recent contributions in this direction in the case of the Lean PA.

In this paper, we focus on the potential use of PAs for teaching proof and proving itself, at the transition between high school and university. The mathematical content we address may therefore be more involved than the elementary and abstract proofs studied in a beginners' logic course (for instance basic set theory or propositional logic), but less than in a general-purpose undergraduate analysis or algebra course. Also note that in this work we choose to consider PAs as possible teaching tools and not as professional tools. A PA's underlying proof theory, the structure and size of its mathematical libraries and its efficiency for professional research are therefore not directly relevant. We will instead focus on the way each tool may or may not enable the development of skills related to proof and proving. To summarize, the questions which motivate this work can be phrased as follows:

[2] `https://www.uc.pt/en/congressos/thedu`, retrieved on 20/07/2022.

- 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 designed an analysis grid to capture characteristics of PAs likely to have an impact on teaching and learning, and used it to analyze the resolution of a single typical exercise about functions in a selection of six different PAs (Coq, Lean, Isabelle, Edukera, D∀∃duction and Lurch). Our aim is to help distinguish aspects of PAs which may facilitate or hinder student's learning of the knowledge and skills involved in proof and proving, as a preliminary to future research.

The article is structured as follows. In Section 2 we present the theoretical background used for this study. In Sections 3 and 4 we present our case study, detailing our methodology, the choice of an exercise and the PAs we decided to analyze. In Section 5 we present several analysis criteria (or *aspects*) which we apply to our data. Finally section 6 presents several hypotheses, based on our analysis, regarding the possible impacts of each PA in relation to its various aspects.

2. Theoretical context

This work draws on several theoretical references, some related to the integration of tools in mathematical education, others to the specific area of proof and proving. We briefly summarize a few selected references which helped us formulate our research questions and clarify our observations regarding the possible impact of PAs for teaching.

2.1. Instrumental approach

We rely on concepts from the *instrumental approach* (Rabardel, 1995). Loosely quoting Trouche (2005), this approach makes a distinction between *artifacts*, 'bare' technical (material or conceptual) objects yet unrelated to a subject, *tools*, technical objects which are integrated (or may become integrated) by a subject in their work, and *instruments*, mixed entities consisting of a tool and its specific usage modes as constructed by a given subject. *Instrumental genesis* is the process by which subjects transform artifacts into instruments. This complex process depends both on the background and knowledge of the subject and on the constraints and characteristics of the artifact.

Understanding instrumental genesis requires to distinguish two dual components, *instrumentalization*, which refers to the way a subject appropriates and parameterizes the artifact, and *instrumentation* which refers to the emergence and evolution of schemes relevant to the realization of a task or type of tasks. Among other aspects, it also requires to articulate social schemes, which are built into an artifact and adopted by a subject's sociocultural environment, and individual schemes. To summarize, the instrumental approach investigates the conditions under which artifacts may become instruments, in particular in the context of mathematics education, possibly with the help of teachers.

We will refer to the above concepts in this work to guide our reflection on the characteristics of PAs which may or may not have an impact on the teaching and learning of proof. In particular, since several PAs were originally designed for professional use, they likely carry strong social schemes in their respective user communities.

2.2. Teaching and learning proof

In order to distinguish proof from other forms of argumentation such as heuristic or explanation, Duval and Egret (1993) present two distinctions regarding mathematical statements: first, that of *content* vs. *status*, and second that of *truth value* vs. *epistemic value*. A statement's status does not depend on its content but only on the context in which it is introduced. It can be either theoretical (definition, theorem, conjecture, etc.) or operational (premise, conclusion, etc.), the latter depending on the former. The epistemic value of a statement can be either *semantic*, expressing an intuitive degree of trust based on its content (from 'absurd' to 'obvious'), or *theoretical* ('necessary', when it appears as the conclusion of an inference step). Each epistemic value is also associated with a truth value ('true', 'false' or 'indeterminate'), depending on a discipline's culture and conventions. However, mathematics should only associate the value 'true' to the epistemic value 'necessary' (and not for instance to 'obvious').

According to the authors, explicitly taking into account each statement's status and considering its theoretical rather than semantic epistemic values is one of the main differences between proof and argumentation, and a possible source of difficulty for learners. PAs take this distinction to an extreme by only allowing strict inference steps, and in some cases explicitly displaying the status of each statement at every step, without ever mentioning semantic epistemic values or truth values.

Hanna (2000) also reflects on the role of proof in mathematics education and its relationship with other activities such as heuristics, exploration and visualization. She argues that 'even in the eyes of practising mathematicians rigorous proof, however it is defined, is secondary in importance to understanding' (p. 6). She then lists several possible functions of proof, stating that 'every student just entering the world of mathematics [must] start with the fundamental functions: *verification* and *explanation*' (our emphasis), adding that '[i]n the educational domain, then, it is only natural to view proof first and foremost as explanation, and in consequence to value most highly those proofs which best help to explain' (p. 8).

It could be argued that PAs may tend to go against this point of view, sometimes placing verification foremost at the cost of explanatory power. We will have to question this effect of PAs when analyzing their use for teaching. Among the other functions of proof cited by Hanna, professional PAs such as Isabelle, Coq or Lean, also heavily promote that of *systematization*: 'the organisation of various results into a deductive system of axioms, major concepts and theorems' and *communication*: 'the transmission of mathematical knowledge' (Hanna, 2000, p. 8), as witnessed by the extremely large collections of formally proved mathematical facts collaboratively developed by their communities of users.

In J. Selden and Selden (2010), the authors distinguish several aspects of the structure of proofs and relate them to students' abilities (or lack thereof). First, they discuss the *hierarchical structure* of a proof, which decomposes a proof into nested sub-proofs and sub-constructions. They also propose the notion of *construction path*, which is a kind of *a posteriori* linear reconstruction of an idealized proof. Finally, they distinguish the *formal-rhetorical* parts of a proof from its *problem-centered* parts.

According to the authors, the formal-rhetorical parts of a proof, which imply formally manipulating statements, analyzing (and acting on) their logical structure, 'folding' or 'unfolding' definitions, mostly involve *behavioural knowledge* (which they relate to Vergnaud's *concepts-in-action* and *theorems-in-action*): knowing *what* to do and *when*, in particular knowing how to construct the next step in a proof. This mainly has to do with invoking *action schemes* and does not always require deep understanding or insight.

On the contrary, problem-centered parts may require *conceptual knowledge* (knowing *why*), mathematical intuition and the mental availability of mathematical resources (for instance knowing which external theorem to invoke and when). These two types of proof elements are closely connected, for instance when 'constructing the formal-rhetorical part of a proof can be very helpful in revealing the "real problem" to be solved in the rest of the proof' (J. Selden & Selden, 2010, p. 344).

Finally, we mention a survey by A. Selden (2012) on the transition from secondary to tertiary education. Section 2 reports on students' documented proving difficulties, related for instance to mathematical language in reasoning: confusion with everyday language, difficulties with the order, scope and proper use of quantifiers, or with the formulation of negation. Other difficulties concern the nature and structure of proofs themselves, for instance the erroneous perception that proofs are always linear and 'top-down', or the distinction between inductive and deductive proofs. The section mentions several other important proof-related skills, such as understanding and manipulating definitions or other formal statements (in particular the one to be proved), reusing previous definitions and theorems, knowing how to read and check proofs. We will keep these in mind when analyzing possible impacts of PAs, especially in light of the elements from J. Selden and Selden (2010), as some of these tasks may be facilitated or altered.

3. Methodology

As stated above, our main contribution is an analysis grid for PAs used in education. We simultaneously developed and instantiated this grid by analyzing the resolution of a single exercise in six different PAs. We detail our research methodology below.

Our first task was to select PAs with varied characteristics, not to determine the overall best for teaching but to provide an *a priori* analysis of their potential impact. Hence, we chose to include tools which may not be stable nor well maintained but present interesting specificities. We retained three well-established professional PAs (Coq, Lean and Isabelle) and three more or less experimental PAs designed for teaching (Lurch, Edukera and D∀∃duction). Our selection is presented in Section 4.2.

Concurrently, we chose a single exercise to analyze in all PAs. Since we wished to focus on the process of proof and not on other mathematical difficulties it had to concern elementary objects, without being so trivial that its proof would appear obvious to students. Finally, we preferred an ordinary exercise likely to appear in an undergraduate introductory course. We opted for a simple exercise on the direct and inverse image of abstract functions, which we detail in Section 4.1.

We then constructed a first draft of a solution in all six tools. Based on initial observations and on relevant literature (summarized in the previous section), we designed an analysis grid consisting of three categories of aspects. To this end, we kept in mind general concepts related to instrumental genesis as well as proof-specific criteria (regarding proof structure, the status and epistemic value of statements, the possible functions of proof and the known required skills or difficulties of students). We also attempted to capture the main differences between each of the selected tools.

Once our grid was established, we carefully revisited each resolution of the exercise in full detail, with one experimenter playing the role of an idealized student[3] and building

[3]For practical reasons we did not ask actual students to perform the resolution: identifying voluntaries proficient enough with each tool seemed very time-consuming, and did not appear relevant for such a preliminary study. This may however constitute a possible next step for our research.

the proof interactively in each tool[4], and one or several observers recording proof steps, tool interactions, potential errors etc. We collectively analyzed the obtained resolution traces with respect to this grid. Details of the analysis grid as well as observations regarding the aspects of each PA are presented in Section 5.

Finally, by analyzing the data collected on this example, we summarized a few possible impacts of PAs on the learning of proof, trying to associate relevant aspects from the grid. We present these observations in Section 6, with the intent to help formulate useful questions and hypotheses for future research.

4. Case study: solving an exercise in six PAs

4.1. Exercise selection

As previously mentioned, we chose to analyze a typical exercise about sets, relations and functions, as commonly found in introductory courses about reasoning and proof. It is available (or easy to formalize) in all studied PAs, with minor variants.

Exercise 4.1. Given three sets A, B and C such that $C \subseteq A$ and a function $f : A \to B$, show that:

(1) $C \subseteq f^{-1}(f(C))$.
(2) If f is injective then $f^{-1}(f(C)) \subseteq C$.
(3) If f is injective then $f^{-1}(f(C)) = C$.

We chose this exercise because it involves relatively simple mathematical concepts, little calculation, and the required proofs are of manageable size, yet not trivial for most undergraduate students. They rely on the concepts of set, set inclusion and set equality, function, direct and inverse image of a set by a function, and injectivity. Understanding these concepts' definitions and using them adequately requires a certain proficiency with universal and existential quantification, logical implication, and possibly set comprehension.

Note that in this exercise the same notation is used to refer to distinct notions and its intended meaning has to be inferred from the context. For instance, $f(C)$ can only be distinguished from $f(x)$ by knowing that C denotes a subset of A and x an element of A. Similarly, if D is a subset of B, $f^{-1}(D)$ is always well-defined and denotes the inverse image of D by f, whereas the *function* f^{-1} (the inverse of f) may not exist at all in general. In our experience, such *overloaded* notation is a common source of difficulty for first-year students in exercises such as this. In a way, this kind of mathematical notation requires students to be aware of the *type* of objects denoted by each symbol.

Figure 1 provides a possible 'hand-written' proof of each of the exercise's statements for future reference. Its style is voluntarily more detailed and formal than what is expected from students, in part to make it easier to compare to PA-produced proofs.

Given the nature of the statements, most other proofs are likely to use essentially the same arguments, although experienced mathematicians may formulate or organize them in different ways. For instance for question 1, instead of finishing the proof by writing 'By definition of set inclusion, we have $C \subseteq f^{-1}(f(C))$', some writers may prefer to *start* it with : 'By definition of set inclusion it is sufficient to prove that $\forall x, x \in C \implies x \in f^{-1}(f(C))$', or to omit this step altogether. Additionally, it should be noted that in this text many proof elements, such as definitions, remain implicit.

[4]For lack of space, details of the resolution in each PA were moved to Appendix A.

(1) Let $x \in C$.
By definition of $f(C)$, we have $f(x) \in f(C)$.
By definition of $f^{-1}(f(C))$, we have $x \in f^{-1}(f(C))$.
Therefore by definition of set inclusion, we have $C \subseteq f^{-1}(f(C))$. □

(2) Assume f is injective.
Let $x \in f^{-1}(f(C))$.
By definition of $f^{-1}(f(C))$, we have $f(x) \in f(C)$
By definition of $f(C)$, there exists $x' \in C$ such that $f(x') = f(x)$.
By injectivity of f, we deduce $x' = x$.
Since $x' \in C$, we also have $x \in C$.
Therefore by definition of set inclusion, we have $f^{-1}(f(C)) \subseteq C$. □

(3) Assume f is injective.
From (1) we know that $C \subseteq f^{-1}(f(C))$.
From (2) we know that $f^{-1}(f(C)) \subseteq C$.
Therefore by definition of set equality, $f^{-1}(f(C)) = C$. □

Figure 1.: Hand-written proof for exercise 4.1.

Each statement is assumed to hold the operational status of *conclusion* to a rewriting or deduction step applied to previous statements (*premises* for that step). Hypotheses are sometimes not recalled, especially when they are the very previous statement.

Similarly, the way quantifiers and logical constructs are handled follows more or less explicit conventions. For instance, the phrase 'Let $x \in C$' actually corresponds to two different 'actions': the introduction of an object (x), and of a hypothesis concerning it (we assume that $x \in C$). The phrase 'there exists $x' \in C$ such that $f(x') = f(x)$' carries the meaning that such an x' exists *and* introduces a new object also called x' for which this property holds. A more careful phrasing could be: 'there exists an element in C whose image equals $f(x)$, let us call x' one such element'[5].

4.2. Proof assistant selection

We now briefly describe the six PAs we chose to analyze in this work. Details of possible resolutions of exercise 4.1 in each PA are available under Appendix A.

4.2.1. Coq

Coq is a free and open-source PA created in the 1980s in French academia (The Coq Development Team, 2022). It was used to prove mathematical theorems such as the Feit-Thompson theorem and the four-colour theorem, and to prove the correctness of large computer programs such as the CompCert certified compiler (Leroy, 2009). Coq has also been used for years as a teaching tool in graduate or undergraduate curricula[6].

A possible way to construct a proof in Coq consists in writing a sequence of commands or *tactics* operating on the current *proof state*, as one may write a script in a imperative programming language. A proof state consists in a list of *goals*, the statements one has

[5] One may distinguish 'there exists x' as a quantifier (acting as a binder for x) and '[we know that] there exists some x [so we pick one]' as an element of proof discourse (introducing x, as 'let x' does). These two statuses are analogous to the distinction between 'A implies B' (which states an implication) and 'A, hence B' (which states a use of *modus ponens* with the previous implication as premise).

[6] See `https://github.com/coq/coq/wiki/Universities-teaching-Coq`.

to prove, each with its *context*. A context is an ordered list of typed variables, some denoting more or less complex objects, others denoting statements whose status in the proof is that of hypotheses[7]. Tactics operate on the state, usually by transforming the current goal or some of the variables and hypotheses in the current goal's context. They range from very simple transformations to elaborate, composite and possibly automated strategies. A proof is considered complete once all goals are 'closed' (usually by matching them with known axioms or hypotheses), meaning that there is nothing left to prove.

Coq includes an interactive development environment allowing a programmer to navigate through proof scripts and visualize the effect of each tactic on the proof state. In this work, we mostly used Coq in this 'imperative' mode, using elementary tactics rather than Coq's more advanced automation features or lower-level syntactic constructs.

4.2.2. Lean

Lean is an open-source project[8] initiated by Leonardo de Moura at Microsoft Research in 2013. Its shares a large part of its theoretical foundations with Coq. The language offers similar proof-writing possibilities, in particular tactics-based imperative proof scripts. Lean also relies on a very extensive user-maintained library of mathematical definitions, theorems and tactics called mathlib (The mathlib community, 2020).

Lean has been gaining momentum in the mathematics and computer science education community (see for instance Hanna & Yan, 2022a, 2022b; Thoma & Iannone, 2022). It is now used as a teaching tool in several institutions worldwide. Using Lean's meta-programming framework, which allows advanced users to write their own tactics, teachers have been creating their own Lean-derived proof environments, for instance the Lean-verbose tactics library by Massot[9] which we discuss below, or the D∀∃duction environment created by Le Roux, which we describe in more detail in Section 4.2.6.

As mentioned above, there exist several proof-writing styles in Lean. Lean (together with mathlib) proposes an LCF-style language based on tactics. Similarly to Coq, it also allows writing proofs directly as functional programs using a succinct and expressive syntax. We will instead focus on the 'declarative' proofs promoted by Lean-verbose.

Originally, Massot created a set of Lean tactics to be used in the context of a first-year mathematics and computer-science undergraduate course on logic and proof. The course alternated exercises on abstract logic and first-term analysis notions and theorems. Massot justifies using Lean in this course to his students by stating: 'The most visible reason for [using Lean in this course] is to check the strict application of logical rules. But the most important reason is to provide in real-time the current local context and current goal at each reasoning step.'[10]

Massot's Lean tactics (originally in French, later published online under the name lean-verbose) mimic the typical natural-language phrasings he expects his students to use when writing a proof on paper. In Massot's own words: 'This project provides tactics for Lean in a very controlled natural language. [...] The goal is not to make Lean code easier to write, the goal is to make Lean code easier to transfer to a traditional paper proof.'[9] In some sense, Lean-verbose pursues a goal similar to that of the first declarative PA languages such as Mizar and Isar (cf. Section 5.1.2).

---

[7] In fact, there is no fundamental difference in Coq's underlying theory between objects and statements, but it may help to think of them that way.

[8] `https://leanprover.github.io/about/`

[9] `https://github.com/PatrickMassot/lean-verbose`

[10] `https://www.imo.universite-paris-saclay.fr/~pmassot/mdd154/`, retrieved 28/06/2022, our translation.

4.2.3. Isabelle

Isabelle is a free and open-source PA developed at University of Cambridge and Technische Universität München (Nipkow et al., 2002). Like Coq, it was used to formalize several mathematical topics and prove relevant theorems about them, and to show the correctness of complex computer programs[11]. For instance, Isabelle was used to prove the correctness of a general-purpose operating system kernel (Klein et al., 2009).

One of Isabelle's original aspects is that it is in fact a general deduction framework in which different logical systems can be implemented. Most current Isabelle developments are performed using higher-order logic (Isabelle/HOL), but other systems may be used (Isabelle/FO for first-order logic, Isabelle/ZF for Zermelo-Fraenkel set theory, etc.)[12].

Isabelle allows proofs to be written in two different styles, a procedural style using tactics (similar to the ones used in Coq and Lean), and a declarative style supported by its Isar language, which consists in writing structured mathematics-like proofs by listing and connecting statements along with their justifications. In this work, for contrast with previously studied tools, we focus on the declarative style offered by Isabelle/Isar. According to its reference manual[13], Isar stands for *Intelligible semi-automated reasoning*. Distancing itself from the tactics-based style, the manual additionally states:

> Drawing from both the traditions of informal mathematical proof texts and high-level programming languages, Isar offers a versatile environment for structured formal proof documents. Thus properly written Isar proofs become accessible to a broader audience than unstructured tactic scripts (which typically only provide operational information for the machine). Writing human-readable proof texts certainly requires some additional efforts by the writer to achieve a good presentation, both of formal and informal parts of the text. On the other hand, human-readable formal texts gain some value in their own right, independently of the mechanic proof-checking process.

4.2.4. Lurch

Lurch is a free and open-source word processor built on the mathematical objects representation standard OpenMath[14], which can check the steps of a mathematical proof (Carter & Monks, 2017). Lurch is designed for student use and was experimented for teaching in 2008 and 2013. To our knowledge, it is no longer maintained. Its user interface is inspired by that of a standard 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. The operational status of each statement as well as the validity or invalidity of a deduction step are color-coded, and variable and hypothesis scopes are visualized using scope delimiters.

Lurch's main feature is the unconstrained mix of formal mathematical statements with illustrative or explanatory text, complete with text styling and rich layout. This allows users to write proofs close to ordinary hand-written mathematical text, with the added benefit of computer-verified validity of deduction steps.

---

[11] `https://isabelle.in.tum.de/community/Projects`

[12] This also likely has a didactic impact as the system provides two different types of logical quantifiers: quantifiers belonging to the core Isabelle deduction system or 'meta-logic', and quantifiers provided by the hosted logical system, which may be confusing to students in the process of acquiring these notions.

[13] `https://isabelle.in.tum.de/doc/isar-ref.pdf`

[14] `https://openmath.org/about/`

4.2.5. Edukera

Edukera is a closed-source web-based graphical PA using Coq as internal mathematical engine (Rognier & Duhamel, 2016). At the time of writing, it is no longer actively developed. It was specifically designed to help teach proof and proving as well as high school and undergraduate mathematical content, including algebra and basic analysis. It is therefore unsuitable for professional or academic use.

Edukera's originality is to combine a point-and-click interface with a presentation of the whole proof in a style mimicking human-written text. Even though Coq is used in the background, it is not visible to the user. The system instead focuses on familiar mathematical notation, natural language, logic, and the application of deduction rules and lemmas in the relevant theory. Students therefore do not need to learn an ad-hoc syntax or to understand Coq's internal theory in order to be able to solve exercises. Edukera also includes several tutoring features such as hands-on tutorials, detailed explanations of certain concepts and occasional historical notes.

Edukera offers a library of a few hundred thematically organized exercises. Teachers using Edukera may compose their own exercise sheets by selecting among existing exercises, but it cannot be considered an extensible environment. The exercise library covers elementary topics in formalization, logic, set theory[15], calculus and analysis.

4.2.6. D∀∃duction

D∀∃duction[16] is a recent free and open-source graphical PA based on Lean and created mainly by Frédéric Le Roux. It was designed for teaching, and is under active development. Similarly to Edukera, it provides a purely point-and-click user interface.

Exercise sheets essentially consist of annotated Lean files defining the required objects and theorems and listing exercises. Since exercise sheets are open-source (just as the rest of the software) and follow essentially standard Lean syntax, this tool may be considered user-extensible, at least for expert users: it is in principle possible for a teacher to design their own exercises, including their choice of notations, lemmas, axioms or theorems.

D∀∃duction's originality is to combine Coq and Lean's tactics-based style (in particular regarding interaction with proof and visualization of the proof state, clearly highlighting each object's theoretical and operational status) with a purely point-and-click interface similar to that of Edukera. Users may select one or several statements to operate upon, which may be hypotheses or a goal, and perform one of several available actions: introduction or elimination of a logical connector, application of a certain proof strategy (reasoning by cases, *reductio ad absurdum*…), creation of objects or intermediate goals, unfolding of a definition or application of a provided theorem.

D∀∃duction does not attempt to build a full proof text similar to what a human could write, but offers various interactive proof visualization features, such as a proof tree view and a summary of the proof outline, which can be opened to the side and are updated dynamically as the proof progresses. Finally, the tool proposes a few settings regarding the automation of certain deduction steps, such as the automatic introduction of quantifiers or implications, or the implicit use of definitions.

---

[15] Our Exercise 4.1 features in the 'Functions' chapter of the 'Sets' collection, numbers 4 and 24.

[16] `https://github.com/dEAduction/dEAduction`, retrieved 15/09/2022.

Category	Criteria
Language and interaction mode	Type of user input Imperative vs declarative style Object naming and referencing Use of mathematical notations
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 visualization	Global vs local viewpoint on proof Possibility to create new definitions and lemmas Status of statements

Table 1.: Categories of analysis criteria.

5. 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 category includes several criteria summarized in Table 1. As mentioned before, other important practical factors are left out of this study, such as type of license, availability, ease of installation, integration with learning management systems, etc.

5.1. Language and interaction mode

The first category we consider relates to the nature of interactions between user and PA. 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[17].

5.1.1. Type of user input

Interaction between the user and the PA generally includes both click-based and text-based modalities, to varying degrees. In D∀∃duction and Edukera, most interactions are point-and-click (through menus, buttons, drag-and-drop), textual input being required on few occasions (for instance when introducing an existential witness). In Coq, Lean and Isabelle, the user types in tactics or proof text, following a strict syntax; the user interface or development environment (IDE) may provide assistance through text completion and tactic suggestion. In Lurch, the user experience is similar to that of 'literate programming' where code is mixed with text; by default, natural-language text is ignored and carries no semantics for the PA, while 'meaningful expressions', whose syntax resembles that of standard mathematics, are combined following annotations by the user to form deduction steps, which are then formally checked by the software.

[17]In professional PAs, user experience may also depend on third-party development tools and interfaces.

5.1.2. Imperative vs declarative style

The input languages of PAs usually exhibit a mix of imperative or declarative features. In a mostly imperative language the user describes a proof as a sequence of changes to be performed on the proof state using a predefined set of orders or *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. This proof style is sometimes referred to as LCF-style for 'Logic of Computable Functions', in reference to a pioneering system developed by Robin Milner in 1972 (see Gordon (2000) for a history of LCF-style reasoning tools). In contrast, in a declarative or 'mathematics-like' language such as pioneer Mizar (Rudnicki, 1992), later followed by the Isar language used in Isabelle (Wenzel & Wiedijk, 2002), one provides assertions along with their justification similarly to a natural-language proof. Statements are written explicitly, using a syntax resembling mathematical language. Regardless of the style used, proofs are then validated algorithmically by the underlying deduction system.

To make the distinction clearer, let us discuss in some detail the use of a definition (here that of '$x \in f(C)$') in the course of a proof in each of the tools we studied.

Coq and Lean are often used in an imperative style: the user enters tactics which perform transformations on the current proof state. For instance, in a Coq solution to Question 1 of exercise 4.1, to prove that $f(x) \in f(C)$ (written `im f C (f x)` here) the user may run the command `unfold im` which instructs the prover to 'unfold' the definition of `im`, yielding as a new goal $\exists x_0(x_0 \in C \wedge f(x) = f(x_0))$, as illustrated in Figure 2. Note that the user does not state the obtained goal manually, but simply what transformation to apply to the current proof state.

Initial state	Tactic	Obtained state

`f : A -> B`
`C : Ens`
`x : A`
`H : C x`
`------------`
`im f C (f x)`
	`unfold im`	`f : A -> B`
		`C : Ens`
		`x : A`
		`H : C x`
		`--------------------------------`
		`exists x0 : A, C x0 /\ f x = f x0`

Figure 2.: Effect of a Coq tactic on the current proof state.

In Edukera and D∀∃duction, the user simply clicks the 'def' button while the statement is selected and the definition of the image of a set is unfolded automatically, with the same effect as `unfold im` in Coq.

In Isabelle/Isar the language is mostly declarative: at every step the user writes down a claim, then justifies it by invoking an available definition or theorem. Assuming the hypothesis $x \in C$ (labelled `x_in_C`) is available in the current context, the user may type: `have "f x ∈ f ` C" using x_in_C by (rule imageI)`. This attempts to prove $f(x) \in f(C)$ using hypothesis `x_in_C` and the definition of the image of a set (`imageI`). A significant difference with the previous approach is that users have to write successive goals themselves. In Lurch, definitions are also managed in a declarative style: the user has to manually type in and justify statements using definitions or theorems.

A fundamental specificity of tools used in an imperative style is that the user builds the proof step by step by acting on the state. In most of them (except Edukera), one may argue that the main final product is simply a *script* allowing to reconstruct the whole proof, while declarative style yields a more or less complete proof text.

5.1.3. Object naming and referencing

In Coq, Lean and Isabelle, users can name hypotheses and objects whenever they are introduced. For instance the Coq tactic `intros x H` applied when the current goal is `forall x, C x -> pre f (im f C) x` attributes the name `x` to the generic element to be considered in the proof of inclusion and the name `H` to the statement `C x` meaning that `x` is in set `C`. Any other names could have been chosen instead, and one could also let the PA pick names automatically. 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 both Edukera and D∀∃duction, variables and hypotheses are automatically assigned names. In Lurch one can choose custom labels for statements.

Even though each tool differs in presentation choices and interaction styles, being able to refer to objects by name is essential to the structuration of a proof. Note that naming objects is mandatory for writing statements, while naming intermediate facts and assumptions may sometimes be optional (using tactics like `assumption` or clicking on hypotheses) since statements are usually assimilated to their name.

5.1.4. Use of mathematical notations

Mathematical notations are essential for readability, both in pen-and-paper proofs and in PAs. In our case, the required notations are the logical connectives, the relation symbols $\in$ for membership and $\subseteq$ for set inclusion and the notations for the image and preimage of a set by a function, respectively $f(C)$ and $f^{-1}(C)$. All studied PAs provide notations for these objects, which are used either for output (displaying proof states to the user), for input (composing statements or objects) or both.

In Coq, standard libraries mimic symbols using common characters, like `->` for implication or `/\` for conjunction. In Lean, established practice involves the use of Unicode symbols for $\to$, $\wedge$, $\in$ and $\subseteq$. In Edukera and D∀∃duction, usual mathematical symbols are displayed and the point-and-click interface handles input.

Note that the notations for image and preimage of a set, which are common in mathematical practice, are ambiguous. The reader is expected to be able to distinguish in a given context the application of some function f on an element x of its domain (written $f(x)$) from the image under f of a subset C of its domain (noted $f(C)$), using previous binding information on variables x and C in the proof text.

The way this ambiguity is handled varies between tools. Professional PAs relying on a strict syntax tend to use distinct notations, while some teaching-oriented PAs follow mathematical practice. In Edukera for example, since the notation is used only as an element of display and the appropriate definition is automatically chosen internally by the system, students are not explicitly reminded of this distinction.

Concept	Math notation	Lean	Isabelle
Direct image	$f(C)$	`f ' C`	`f ` C`
Reverse image	$f^{-1}(C)$	`f ⁻¹' C`	`f -` C`

From experience, ambiguity in notations is a common source of confusion for students and the effects of one approach (using explicit but non-standard notations for disambiguation) or the other (following the common but possibly ambiguous practice) is hard to predict. We can expect that PAs using distinct notations help students realize that distinct concepts are at play, but may not train them to recognize which notion is referred to in a hand-written statement and handling the disambiguation process.

5.2. Automation and user assistance

This category refers to all features providing some kind of assistance in the selection of a usable rule in a given context, in the syntactic manipulation of statements (in particular regarding type checking, substitution and pattern matching), in the more or less automatic application of rules, etc. Other features include the organized presentation of available rules and theorems, automatic scope management, and contextual hints or feedback. According to testimonies from PA designers and teachers, finding a good balance in the level of automation is a challenge, especially in an educational context where proof clarity may sometimes be considered more important than efficiency or succinctness. In extreme cases, certain advanced tactics may be able to solve some non-trivial goals automatically, without students getting any insight into the main ideas of the proof. Conversely, some non-elementary proof steps which could be considered correct in a pen and paper proof may require several elementary steps in a formal proof.

5.2.1. Mathematical libraries

In a PA, every notion and fact is formally built from first principles, in a spirit of *systematization*. This implies that proving any mathematical statement usually involves using many definitions and theorems, directly or indirectly. Most PAs thus provide libraries of common notions and facts to build on. In professional PAs, definitions and theorems are referred to by a name. Libraries may contain thousands of facts, so these systems often provide additional assistance such as textual or pattern-based search (e.g. Coq's `Search` command), automatic completion, online help, lemma suggestion (e.g. the `suggest` tactic from Lean's `mathlib` library), etc.

Teaching-oriented PAs like Edukera and D∀∃duction have much smaller libraries and simply list the predefined lemmas and definitions available in any exercise, sorted by topic. Each exercise can therefore be seen as a 'closed micro-world' or 'sandbox'. In Lurch, users may specify which parts of the standard library they wish to use.

5.2.2. 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.

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 successively until one succeeds. Automation consists in selecting the appropriate instance of the rule in the current context.

In Coq, Lean, D∀∃duction 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 these systems, pattern matching and substitution are performed automatically. In Lean, Coq and Edukera's math mode, generic commands are available to eliminate or introduce logical connectors and quantifiers. Only when ambiguity occurs is the user required to add further input. In other tools, the user generally has to determine the outermost logical connective themselves. Besides, professional PAs provide varying degrees of automation, as mentioned below.

5.2.3. Scope management

According to previous research on proof (see for instance A. Selden, 2012) keeping track of the scopes of variables and hypotheses is a source of difficulty for students. It sometimes leads to confusion between free and bound variables, or to circular arguments. In professional PAs and also in D∀∃duction, scope management is fully automatic and available variables and hypotheses are gathered in corresponding areas of the interface thanks to the notion of 'proof state'. In Edukera, unproven statements are clearly distinguished from proved ones, scopes are outlined and indented and can be selected when introducing new variables or hypotheses. Isabelle and Lurch also have syntactic or visual hints to indicate scopes, but more work is left to the user.

5.2.4. 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. In Coq, 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, Lean 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.

5.2.5. Type of feedback

Most modern systems provide users with feedback on the current proof state, on each statement's theoretical status or its operational status, in the terminology of Duval and Egret (1993). Feedback varies from basic to very rich and is provided in various ways, either textually or using visual hints, by highlighting elements of the syntax of the PA's language, or by separating statements between disjoint areas of the user interface. These features may play an important role in an educational context.

In Coq, Lean and Isabelle the standard interface highlights the part of the proof script that was successfully checked so far, displays the current active goal with its context and indicates other open goals (see figure 3 for an example in Coq's default interface). A dedicated area shows error messages when a rule does not apply, when an expression is not well-typed, or of course when a syntax error occurs.

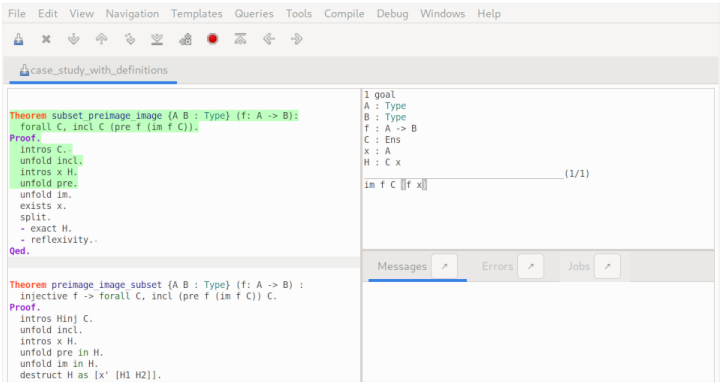


Figure 3.: CoqIDE showing a partially checked proof script for Question 1

Feedback in Lurch is very rich: there is a colour code to indicate the status of each statement (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.

5.3. Proof structure and proof state visualization

This final category concerns the aspects of a PA related to how proofs are perceived and handled. There are two main design choices: in some PAs, the whole proof text is visible at once, and users complete it by inserting new assertions. Work may be done progressively and non-linearly 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 possibility to decompose a long proof by writing down definitions and separately proving intermediate lemmas which can then be reused.

5.3.1. Global vs local viewpoint on proof

In Coq, Lean or D∀∃duction, the user may visualize the sequence of invoked tactics and navigate through them to view the proof state at each point. The proof as a whole is implicit and is never displayed entirely. Moreover, the origin of each statement appearing in the context (hypothesis of the theorem to be proven, previously proved fact, hypothesis in a proof by cases, by induction or by contradiction) is not displayed. In these tools, it is also natural to treat the goals in the order in which they are generated. One may say the viewpoint on proof is local, with much information hidden.

On the contrary, in Edukera or Lurch (like 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 at each point. Due to their declarative style and since proof texts in these two PAs rather closely imitate usual mathematical language, they offer a more global viewpoint on proofs without resorting to back-and-forth navigation through the proof script. Isar (Isabelle's language) and Lean-verbose combine both aspects by presenting 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[18].

5.3.2. Possibility to create new definitions and lemmas

This criterion has to do with extensibility of the current theory by the user. D∀∃duction and Edukera do not allow the creation of new definitions or theorems by end users. Work is confined to a 'deductive island' imposed by the system. In Edukera, teachers can compose exercise sheets but only from existing exercises; the development of new theories is not accessible to end users. In D∀∃duction expert teachers users may create new worksheets including new definitions and theorems, within the limits of the interface's capabilities. In Coq, Lean, Isabelle, or Lurch, users are free to restructure their proofs by introducing new lemmas or concepts. It is of course the main purpose of professional tools to be able to define new objects and to verify their properties.

5.3.3. Status of statements

As already stated, one may distinguish the theoretical and operational status of statements, which may vary in the course of a proof: a statement may be the conclusion of a

[18]This is indeed what Massot stresses to his students, Cf. Section 4.2.2.

deduction step and the premise of the following one. The status of statements is rather clear in all PAs (except Edukera where admitted lemmas/axioms, and proved lemmas are not distinguished). In D∀∃duction, hypotheses 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 status of statements is displayed using 'bubbles'.

6. Possible impacts on the learning of proving skills

As their name suggests, PAs relieve the user of some of the tasks associated with proving. While this may be desirable in a professional setting in order to save time, to reduce proof size or complexity or to leverage powerful automation techniques, it might become a hindrance when the goal is precisely to let students practise some of these basic tasks. Conversely, automating certain challenging tasks or assisting the user in performing them may in some cases be beneficial for teaching. By helping students circumvent possible difficulties (such as memorizing definitions, writing down correct formal statements, applying substitutions, managing variable and hypothesis scopes, keeping track of the status of each statement, etc.), PAs may allow them to focus on arguably 'higher-level' tasks such as reasoning and problem-solving. Based on our analysis, we formulate a few hypotheses on the possible effects of the use of PAs in teaching.

6.1. Memorization and formulation

When asking students to solve an exercise on functions on paper, 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 their formal definition.

When using a PA where details of definitions and properties are always at hand, one may postulate that memorization of formal statements is not required in order to 'solve' the exercise, and therefore that students are not exercising this particular skill. However, they may be required to read, understand and use definitions and theorems at the appropriate time. Contrary to pen-and-paper proofs, feedback in case a definition or theorem does not apply to the selected goal or hypothesis is immediate, possibly helping students understand their applicability conditions. It might thus be the case that being repeatedly presented with formal definitions and properties and putting them to use may actually help students memorize them.

A less optimistic hypothesis would suggest that students becoming proficient with the formalism of PAs may successfully use definitions and statements without suitably understanding them.

6.2. Manipulation of formal statements

It has been remarked that performing substitution is one of the many difficulties of the proving activity (Mckee, Savic, Selden, & Selden, 2010, p. 212). As we observed, the six PAs we studied differ in the way they automate the manipulation of formal statements. In four cases (D∀∃duction, Edukera, Lean and Coq), it may be possible to achieve a complete proof without needing to write a single mathematical statement. Lean,

Coq and Edukera automatically identify the outermost operator in a mathematical term. While this does not entirely exempt the user from thinking about statements and anticipating which rules may be used next, it is not up to them to actually figure out which substitution makes a statement match the premise of a rule, or how to apply that substitution to the rule's consequent in order to compute the effect of the rule's application.

This is not the case in Isabelle and Lurch, where the user has to explicitly write 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 the last step in imperative PAs in order to clearly see its effect. This may provide another way to practise skills related to formula manipulation, by reading and controlling 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 regarding substitutions). These feedbacks are of course unavailable in a pen-and-paper proof.

6.3. Perception of proof structure

As already mentioned, PAs strongly assist users in the realization of a number of proof-related tasks, which may be attributed to the formal-rhetorical parts of proofs: automatic management of scopes and contexts, availability of definitions, automatic substitutions, identification of outermost operators, management of statements' operational status and of open goals, and general bookkeeping. This is especially true in imperative-style PAs, but even declarative PAs assist in performing these tasks via syntactic control and immediate feedback for instance.

We hypothesize that in proofs mostly consisting of formal-rhetorical steps, in which the problem-centered part is therefore small, users of imperative PAs (provided they are proficient with each tool's syntax and interface) may feel as though they are 'pushing symbols around until it works', possibly not encountering any difficulty at all or not understanding why the proof went through. In Exercise 4.1, question 1 might be entirely solved this way by somewhat experienced users, but it is less likely that the problem-centered part for question 2 would disappear completely.

More generally, some imperative-style tools may act as 'blinders', allowing one to entirely focus on the current proof state and current goal, keeping other parts of the proof out of mind. This 'tunnel' effect may also make a trial-and-error exploration strategy more viable than in declarative-style PAs. Edukera stands out as a special case in that the whole text of the proof remains visible, even though user input is mostly imperative and syntactic manipulations largely automated. One could say that Edukera interactively builds a construction path in the sense of J. Selden and Selden (2010).

Not knowing 'how to start' is another frequent hurdle when learning proof (A. Selden, 2012). It would be interesting to determine whether the assistance of PAs, and the possible instrumentation process they enable, allow students to gain autonomy in the way they tackle proof, identifying recurring first steps and developing habits in the formal-rhetorical parts of proofs, 'unpacking' the logical connectives in the current goal until all objects and hypotheses are at hand and only the 'core' statement remains.

To sum up, one may hypothesize that PAs, in particular imperative ones, have a very strong effect on the treatment of formal-rhetorical parts of proofs, and may help in 'revealing the real problem', in the words of J. Selden and Selden (2010).

7. Conclusion

One obvious advantage of PAs for teaching is the capacity to automatically verify proofs, either produced by teachers or by students. Apart from this, what are the reasons for a teacher to start using a PA as part of their course? As we saw, design choices in PAs entail different actions from the user. Certain concepts (for instance that of substitution) are always present but under various forms. Freeing students from certain tasks may enable them to concentrate on deeper ideas, and may contribute in helping become better provers outside of the PA. Or, it may deprive them of an opportunity to practise the critical skills associated with these tasks. Figuring out the effect of each PA on learning and investigating the very partial list of hypotheses from the previous section would of course require further research, for instance analyzing the evolution of student's proficiency with various tasks before and after using different PAs. Do PAs have an effect on possible syntactic and semantic difficulties encountered when working on proof? Do they favour the development of higher-level skills such as problem-solving, writing a proof on paper, or summarizing a proof verbally? Do they lead to a better understanding of the process of proof itself? How much do these effects depend on students' backgrounds, in particular in computer programming?

In the growing community of teachers using PAs, one may notice several distinct tendencies (see for instance Kerjean et al., 2022, for testimonies in France). For some educators, who tend to opt for teaching-oriented PAs, the main goal of using PAs is to ultimately 'let the tool disappear', and to teach students to write pen-and-paper proofs respecting a specific set of writing conventions. Others, who tend to adopt professional PAs, consider that a valid tactics-based proof is sufficient. Finally, some educators (notably in the Lean community), testify that 'teaching Lean and math in the same course is too much'. We believe our preliminary analysis may prove useful in order to further investigate the choices and practices of teachers using PAs, and possibly to contribute to the design of teaching resources incorporating PAs.

Finally, a question of epistemological interest which we have begun to investigate is that of the impact of the various theoretical models underlying each PA, for instance the theory of dependent types for Coq and Lean (including the well-known Curry-Howard correspondence between statements and types and between proofs and programs). Is it useful, or even required, for students or teachers to understand some of these concepts in order to use a PA suitably? What are the possible consequences of these various 'visions of mathematics' on the learning of proof and of other mathematical content?

References

Carter, N. C., & Monks, K. G. (2017). A Web-Based Toolkit for Mathematical Word Processing Applications with Semantics. In *International Conference on Intelligent Computer Mathematics* (pp. 272–291).

Duval, R., & Egret, M.-A. (1993, July). 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)*. Paris, France.

Gordon, M. (2000, July). From LCF to HOL: a short history. In *Proof, language, and interaction: essays in honour of Robin Milner* (pp. 169–185). Cambridge, MA, USA: MIT Press.

Hanna, G. (2000). Proof, Explanation and Exploration: An Overview. *Educational Studies in Mathematics*, *44*(1/2), 5–23.

Hanna, G., & de Villiers, M. (Eds.). (2012). *Proof and Proving in Mathematics Education* (Vol. 15). Dordrecht: Springer Netherlands.

Hanna, G., Reid, D. A., & de Villiers, M. (Eds.). (2019). *Proof Technology in Mathematics Research and Teaching* (Vol. 14). Cham: Springer International Publishing.

Hanna, G., & Yan, X. (2022a, July). Opening a discussion on teaching proof with automated theorem provers (pre-print). *For the Learning of Mathematics*, *41*, 42–46.

Hanna, G., & Yan, X. (2022b, June). Teaching with Computer-Based Proof Assistants: Perspectives from Instructors of Mathematics. , *54*.

Kerjean, M., Le Roux, F., Massot, P., Mayero, M., Mesnil, Z., Modeste, S., … Rousselin, P. (2022, October). Utilisation des assistants de preuves pour l'enseignement en L1: Retours d'expériences. *Gazette de la SMF*, *174*.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., … Winwood, S. (2009, October). seL4: formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (pp. 207–220). New York, NY, USA: Association for Computing Machinery.

Knobelsdorf, M., Frede, C., Böhne, S., & Kreitz, C. (2017, August). Theorem Provers as a Learning Tool in Theory of Computation. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 83–92). New York, NY, USA: Association for Computing Machinery.

Leroy, X. (2009, July). Formal verification of a realistic compiler. *Communications of the ACM*, *52*(7), 107–115.

Mckee, K., Savic, M., Selden, J., & Selden, A. (2010). *Making Actions in the Proving Process Explicit, Visible, and "Reflectable".*

Nipkow, T., Wenzel, M., Paulson, L. C., Goos, G., Hartmanis, J., & van Leeuwen, J. (Eds.). (2002). *Isabelle/HOL – A Proof Assistant for Higher-Order Logic* (Vol. 2283). Berlin, Heidelberg: Springer.

Pierce, B. C., de Amorim, A. A., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., … Yorgey, B. (2022). *Logical foundations* (Vol. 1; B. C. Pierce, Ed.). Electronic textbook. Retrieved from `https://softwarefoundations.cis.upenn.edu/`

Rabardel, P. (1995). *Les hommes et les technologies; approche cognitive des instruments contemporains.* Armand Colin.

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

Rudnicki, P. (1992). An Overview of the Mizar Project. In *Proc. of the 1992 Workshop on Types for Proofs and Programs.* Chalmers University of Technology.

Selden, A. (2012). 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). Dordrecht: Springer Netherlands.

Selden, J., & Selden, A. (2010, September). Teaching Proving by Coordinating Aspects of Proofs with Students' Abilities. In D. A. Stylianou, M. L. Blanton, & E. J. Knuth (Eds.), *Teaching and Learning Proof Across the Grades* (Zeroth ed., pp. 339–354). Routledge.

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

The mathlib community. (2020, January). The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 367–381).

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.

Trouche, L. (2005). Construction et conduite des instruments dans les apprentissages mathématiques: Nécessité des orchestrations. *Recherches en Didactique des Mathematiques*, *25*(1), 91–138.

Wenzel, M., & Wiedijk, F. (2002, September). A Comparison of Mizar and Isar. *Journal of Automated Reasoning*, *29*(3), 389–411.

Appendix A. Detailed exercise resolution

This appendix describes in more detail possible resolutions of exercise 4.1 in each of the six PAs we chose to analyze.

A.1. Coq

In Coq's syntax, a colon is used after a variable (or more generally after any *expression*) to indicate its *type*. For instance, if variable `x` denotes an object of type `A` we write `x : A`. For hypotheses, the same syntax introduces the statement corresponding to each hypothesis. For instance `(H : C x)` means that the name `H` refers to a proof of the statement `C x` (whose meaning is explained below). Note that the fact that types and statements are introduced with same syntax is not a coincidence. As we will mention again below, there is a strong correspondence between both concepts.

Figure A1 presents preliminary definitions used to model the mathematical objects involved in the proof. Type `Ens` represents sets using Coq's predefined functions: in this setting a set of elements of type `A` is represented by its characteristic property, a map from `A` to the type `Prop` of propositions (meaning that each value of type `A` is mapped to `True` if it belongs to the set and `False` otherwise). Membership of some `x` of type `A` in some set `C` is therefore written `C x`. From this initial modelling choice, we are able to define set inclusion (`incl`), the image of a set by a function (`im`), its pre-image (`pre`) and the injectivity property for function `f`, called `injective`.

```
1  Definition Ens {A : Type} := A -> Prop.
2
3  Definition incl {A: Type} (C D: Ens) :=
4    forall (x: A), C x -> D x.
5
6  Definition im {A B: Type} (f: A -> B) (C: Ens): Ens :=
7    fun y => exists x, C x /\ y = f x.
8
9  Definition pre {A B: Type} (f: A -> B) (D: Ens): Ens :=
10   fun x => D (f x).
11
12 Definition injective {A B: Type} (f: A -> B) :=
13   forall x x', f x = f x' -> x = x'.
```

Figure A1.: Proof script in Coq : preliminary definitions

Figure A2 shows a possible proof script for question 1[19]. It proceeds by 'backward reasoning', transforming the current goal 'in reverse' by successively introducing quantified objects and hypotheses and unfolding definitions of symbols. In the last steps of the proof, the only remaining goals are `C x`, which matches an existing hypothesis (`H`) and can thus be proved directly (line 14), and `f x = f x` which is proved by reflexivity of Coq's built-in equality (line 15). Note that by Coq's inner mechanism, all unfolding steps are actually optional, which allows one to write a shorter proof. However, unfolding definitions may allow a user to visualize intermediate states and hidden quantifiers, and thus follow the proof's logic more easily when interacting with the software.

[19]We do not wish to explain notations in full detail here. Let us simply mention that all text surrounded by `(* ... *)` is considered as commentary and is not evaluated by Coq. We use it here to describe the effect of

```
1  Theorem subset_preimage_image {A B: Type} (f: A -> B):
2    forall C, incl C (pre f (im f C)).
3  Proof.              (* Context: A, B, f : A -> B               *)
4                      (* Goal: forall C, incl C (pre f (im f C)) *)
5    intros C.         (* New in context: C : Ens                 *)
6                      (* Goal: incl C (pre f (im f C))           *)
7    unfold incl.      (* Goal: forall x, C x -> pre f (im f C) x *)
8    intros x H.       (* New in context: x : A, H : C x          *)
9                      (* Goal: pre f (im f C) x                  *)
10   unfold pre.       (* Goal: im f C (f x).                     *)
11   unfold im.        (* Goal: exists x0, C x0 /\ f x = f x0     *)
12   exists x.         (* Goal: C x /\ f x = f x                  *)
13   split.            (* Two goals: C x and f x = f x            *)
14   - apply H.        (* Goal C x is closed                      *)
15   - reflexivity.    (* Goal f x = f x is closed                *)
16 Qed.
```

Figure A2.: Proof script in Coq for question 1. Text between `(*...*)` is not part of the script, we use it to indicate the evolution of the goals as indicated by Coq.

Figure A3 shows a possible proof script for the second question. It starts similarly to the proof for question 1, first introducing all relevant objects and hypotheses. However, from line 14 on, the proof proceeds by transforming hypotheses, and not the goal itself which remains the same until the very end of the proof, where one hypothesis directly matches the goal (line 25). This may be referred to as a 'forward reasoning' strategy, which may be considered closer to the natural order in which a human would reason, or at least write a proof[20]. As previously, invocations of the **unfold** tactic are not actually required and can be performed automatically by Coq.

```
1  Theorem preimage_image_subset {A B: Type} (f: A -> B) :
2    injective f -> forall C, incl (pre f (im f C)) C.
3  Proof.
4    intros Hinj C.              (* Ctx: Hinj : injective f, C : Ens        *)
5                                (* Goal: incl (pre f (im f C)) C           *)
6    unfold incl.                (* Goal: forall x, pre f (im f C) x -> C x *)
7    intros x H.                 (* Ctx: x : A, H : pre f (im f C) x        *)
8                                (* Goal: C x                               *)
9    unfold pre in H.            (* Ctx: H : im f C (f x)                   *)
10   unfold im in H.             (* Ctx: H : exists x0, C x0 /\ f x = f x0  *)
11   destruct H as [x' [H1 H2]]. (* Ctx: x' : A, H1 : C x', H2 : f x = f x' *)
12   apply Hinj in H2.           (* Ctx: H2 : x = x'                        *)
13   rewrite <- H2 in H1.        (* Ctx: H1 : C x                           *)
14   apply H1.
15 Qed.
```

Figure A3.: Proof script in Coq for question 2. Comments marked `Ctx:` indicate changes in the context (new or changed assumptions).

---

each tactic on the proof state. Notation `{A : Type}` introduces implicit arguments, which can usually be omitted and inferred from the context.

[20]Some backward reasoning steps are nevertheless common in hand-written proofs, for instance using the phrasing 'it suffices to show that'.

Note that Coq's standard library contains several definitions which we could have used to solve our exercise, notably basic definitions of sets, the inclusion and equality relations, the image of a set by a function, and function injectivity. However, these definitions employ techniques which would be too long to describe in this text.

To complete the description of this resolution using Coq, we point out another way to express the proofs as functional-style *programs*. This follows from an extended version of the well-known Curry-Howard correspondence, which establishes formal analogies between logical statements and *types*. According to this correspondence, the proof of a statement can then be expressed as a program which, given evidence for the validity of statement's hypotheses, returns evidence for the validity of the statement itself. This principle also underlies the Lean PA. We leave a more detailed discussion of this topic for future work.

A.2. Lean verbose

Figure A4 shows Lean code for Exercise 4.1 using lean-verbose tactics exclusively. Note that lean-verbose allows to introduce a variable in the context and an assumption about in a single step (line 8). The use of the definition of inverse image is implicit on line 9. Lean verbose allows backward reasoning steps using the tactic 'it suffices to prove that'. Line 23 we can remark some imperative steps, applying a substitution to the current context without stating what is the effect of this substitution.

```
1  import verbose_tactics
2  section exercise
3  variables {α β : Type}
4
5  theorem subset_preimage_image (f : α → β) (s : set α) :
6    s ⊆ f ⁻¹' (f ' s) :=
7  begin
8    Fix x ∈ s,
9    Let's prove that exists x', x' ∈ s ∧ f x' = f x,
10   Let's prove that x works  -- Nothing else to do !
11 end
12
13 theorem preimage_image_subset (f : α → β) (s : set α) :
14   function.injective f → f ⁻¹' (f ' s) ⊆ s :=
15 begin
16   Assume hf : function.injective f,
17   Fix x ∈ f ⁻¹' (f ' s),  -- introduces (x_mem : x ∈ f ⁻¹' (f ' s))
18   We reformulate x_mem to (exists x', x' ∈ s ∧ f x' = f x),  -- optional
19   By x_mem we obtain y such that (hys : y ∈ s) (hyx : f y = f x),
20   Fact h : y = x,
21     By hf it suffices to prove that f y = f x,
22     We conclude by hyx,
23   We replace ←h,
24   We conclude by hys
25 end
26
27 end exercise
```

Figure A4.: Solution to Exercise 4.1 using lean-verbose.

A.3. Isabelle

Figure A5 shows a proof script for questions 1 and 2 of Exercise 4.1. One may notice the use of common English connectives frequently used in mathematical proofs such as 'fix', 'assume', 'then', 'show', etc. which are here endowed with specific semantics. The keyword **by** introduces so-called proof methods, which provide a justification to each proof step. The most common method in this example is `rule`, which together with a label indicates the use of a previously proven deduction rule or lemma. Other available methods may involve substitutions or automated simplifiers or provers. Figure A6 shows the definitions of each predefined lemma used in this example (including the implicitly-used `subsetI` lemma).

```
lemma subset_preimage_image:
  "C ⊆ f -` (f ` C)"
proof
  fix x
  assume "x ∈ C"
  then have "f x ∈ f ` C" by (rule imageI)
  then show "x ∈ f -` f ` C" by (rule vimageI2)
qed

lemma preimage_image_subset:
  "inj f ⟹ f -` (f ` C) ⊆ C"
proof
  assume hf: "inj f"
  fix x
  assume "x ∈ f -` f ` C"
  then have "f x ∈ f ` C" by (rule vimageD)
  then obtain y
    where hxy: "f x = f y" and hy: "y ∈ C"
    by (rule imageE)
  from hf and hxy have "x = y" by (rule injD)
  with hy show "x ∈ C" by hypsubst
qed
```

Figure A5.: Isabelle/Isar proof script for Exercise 4.1. Symbol ⟹ denotes implication in Isabelle's meta-logic.

```
lemma subsetI [intro!]: "(⋀x. x ∈ A ⟹ x ∈ B) ⟹ A ⊆ B"
lemma imageI: "x ∈ A ⟹ f x ∈ f ` A"
lemma imageE [elim!]:
  assumes "b ∈ (λx. f x) ` A" obtains x where "b = f x" and "x ∈ A"
lemma vimageI2: "f a ∈ A ⟹ a ∈ f -` A"
lemma vimageD: "a ∈ f -` A ⟹ f a ∈ A"
lemma injD: "inj f ⟹ f x = f y ⟹ x = y"
```

Figure A6.: A selection of Isabelle/Isar predefined lemmas. Symbol ⋀ denotes universal quantification in Isabelle's meta-logic.

A.4. Lurch

Figure A7 shows a possible proof text for questions 1 of Exercise 4.1, with validation hints activated. Green dots correspond to the conclusions of valid deduction steps. Yellow dots indicate so-called 'undischarged' hypotheses, i.e. statements which are used as premises in deduction steps but are not themselves proven. Placing the cursor on various parts of the last line of this proof, we can highlight the nature (in red) and operational status or *role* (in blue) of various parts of the sentence, as can be seen on Figure A8. Here, we distinguish meaningful statements, rule premises, and 'reasons', i.e. the invoked rule for a deduction step. Various features of the Lurch editor allow a user to pick a role and a scope for any text fragment.

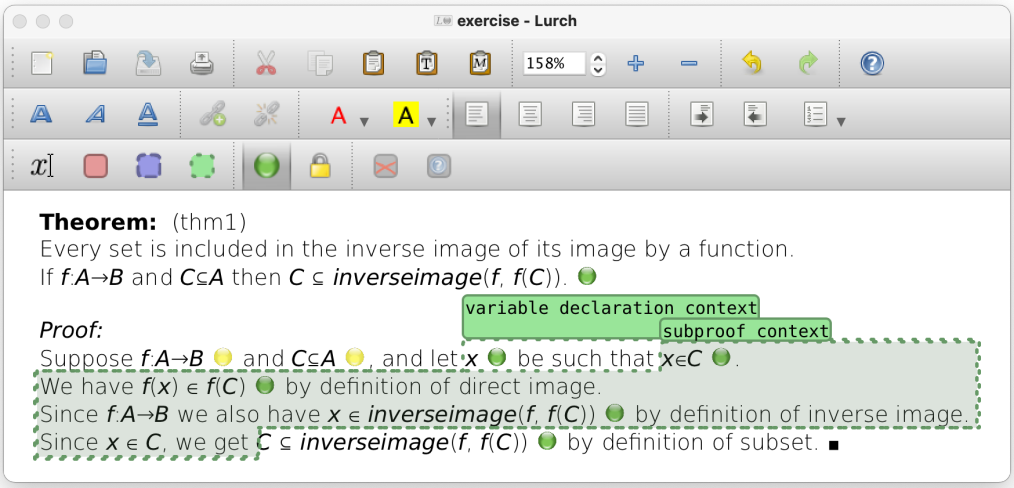
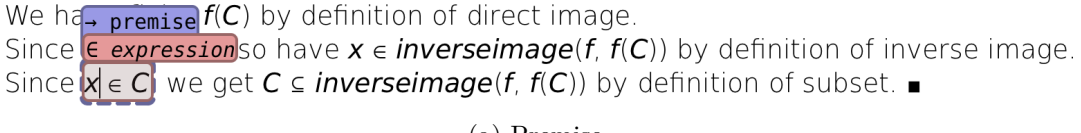
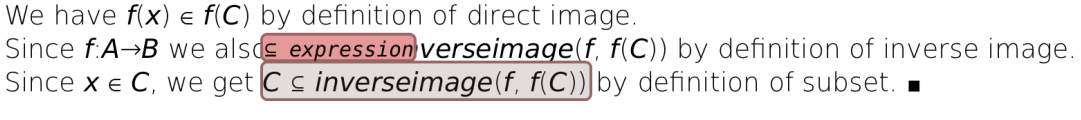


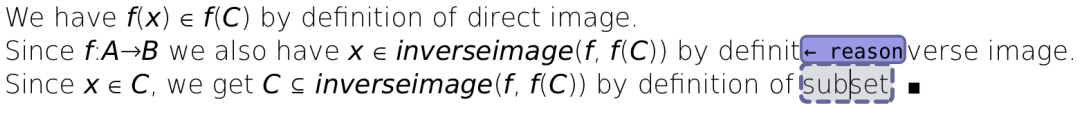
Figure A7.: Lurch proof text for question 1 of Exercise 4.1. Nested green-framed areas indicate variable and hypothesis scopes.



(a) Premise.



(b) Statement.



(c) Reason.

Figure A8.: Roles and nature of statements in a Lurch proof step.

When validation is activated, Lurch verifies that each statement or label cited as a premise is indeed available in the current scope, and that the rule invoked as a 'reason'

is indeed valid. This is done by checking that such a rule actually exists, then searching for a suitable substitution of free variables in the rule's body and matching each premise of the rule to some premise invoked in the text. Figure A9 shows the detailed feedback provided by Lurch when inspecting the above reasoning step. Notice that the obtained substitution is very explicitly detailed.

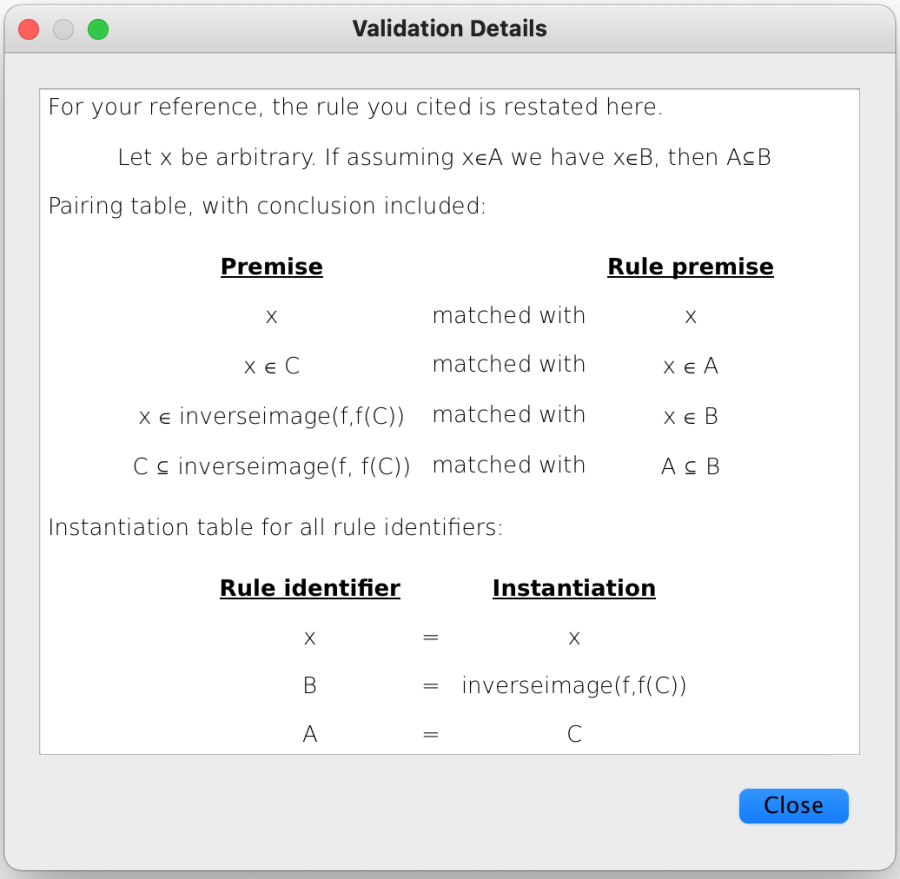


Figure A9.: Validation details for a Lurch proof step.

Figure A10 shows a possible proof text for questions 2 of Exercise 4.1. Figure A11 shows a list of user-written 'if-then rule' definitions used to solve the exercise, in a display mode where meaningful expressions are made visible using red brackets. Blue brackets are used to indicate parts of the text with a specific role (labels, in this case).

A.5. Edukera

Figures A13 and A14 show the state of the Edukera 'worksheet' throughout one possible resolution sequence of question 1 of Exercise 4.1. The worksheet is divided in two areas. On the left-hand side is a dynamically-numbered list of declarations and statements,

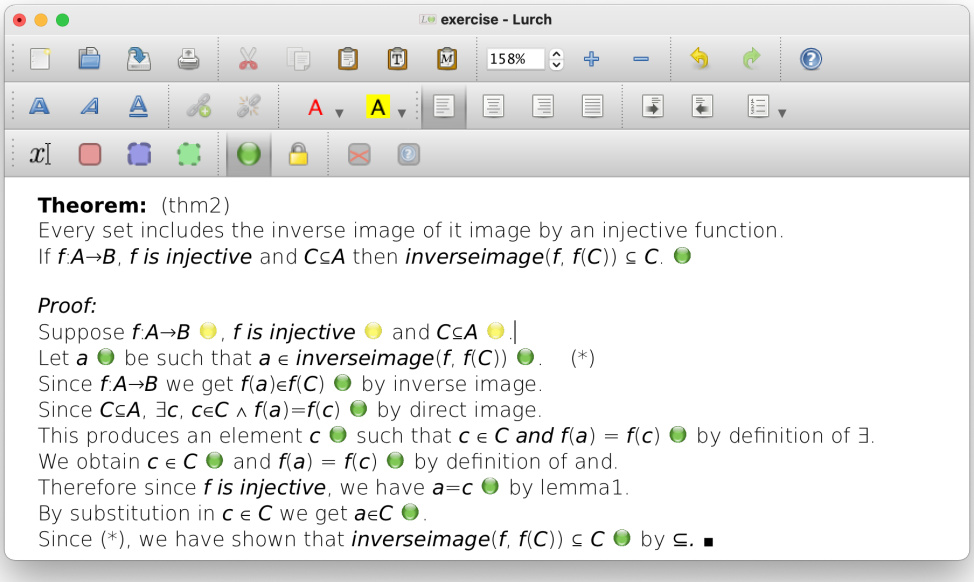


Figure A10.: Lurch proof text for question 2 of Exercise 4.1. The star (*) marker is defined as a label for an intermediate statement, and referred to in the last line of the proof.

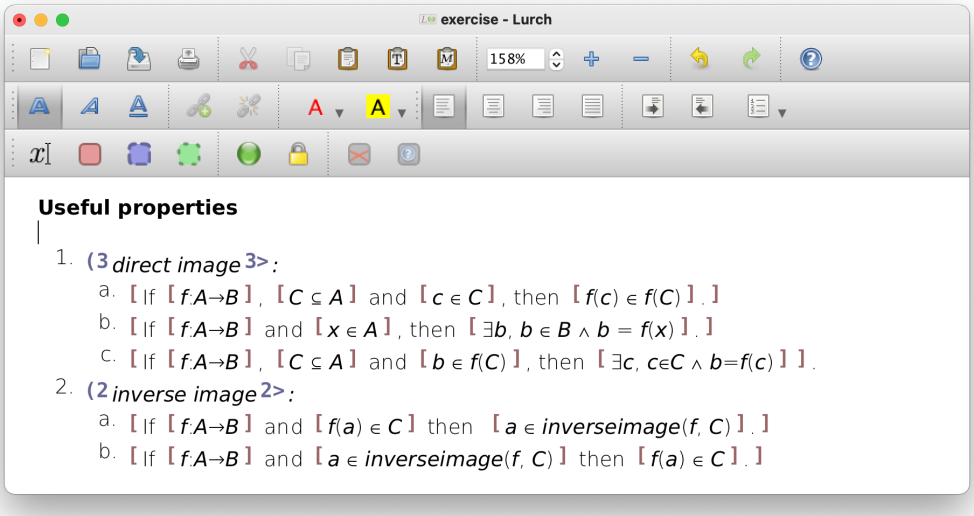


Figure A11.: User-defined if-then rules used in the above proofs. Blue brackets define labels for each rule.

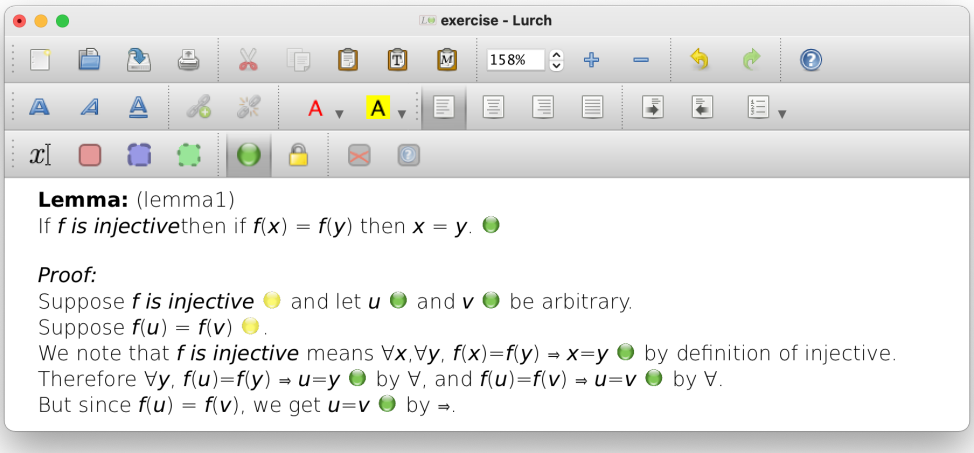
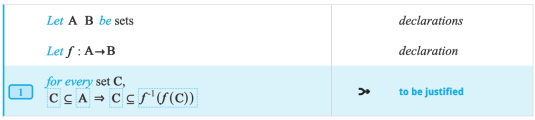
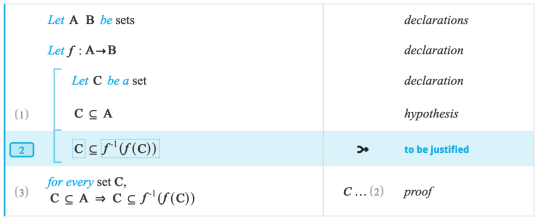


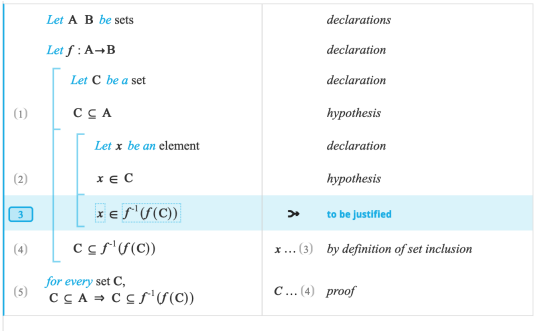
Figure A12.: User-defined lemma used in Figure A10, together with its proof.



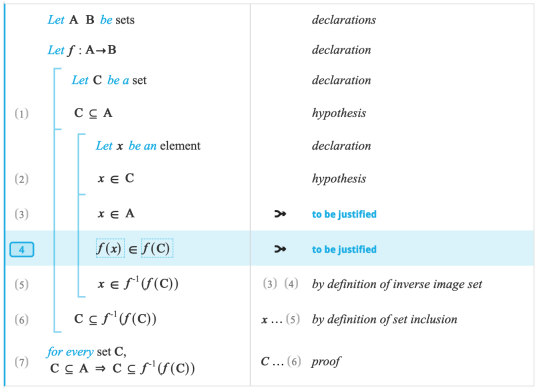
(a) Initial state.



(b) Introduced hypothesis $C \subseteq A$.

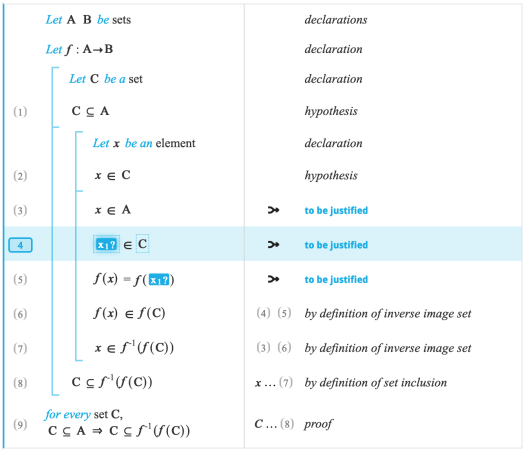


(c) Unfolded $\subseteq$ in $C \subseteq f^{-1}(f(C))$.

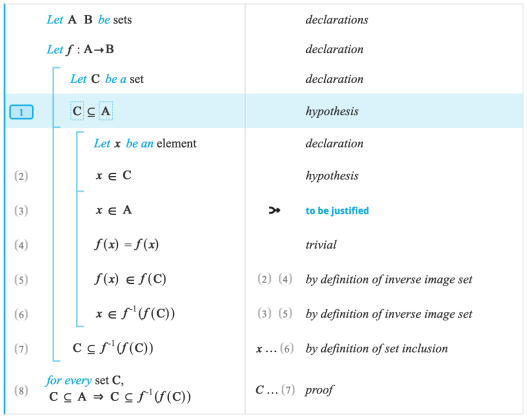


(d) Unfolded f^{-1} in $f^{-1}(f(C))$.

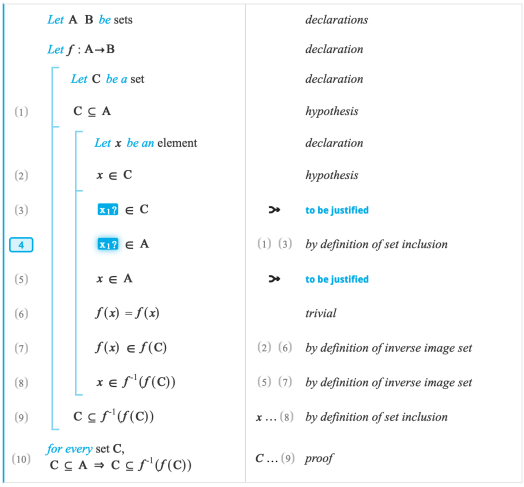
Figure A13.: Edukera resolution of question 1 in Exercise 4.1 (first part).



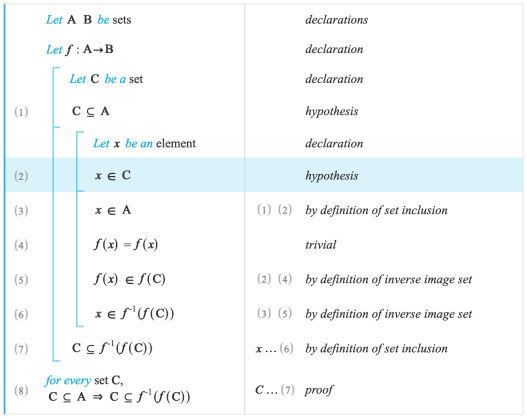
(e) Unfolded f in $f(C)$.



(f) Identified meta-variable x_1.



(g) Unfolded $\subseteq$ in $C \subseteq A$.

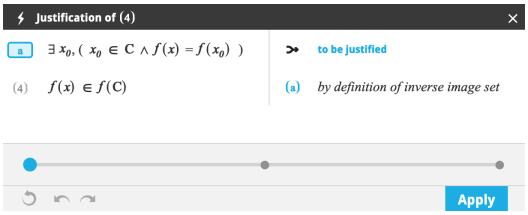


(h) Final state: identified goals.

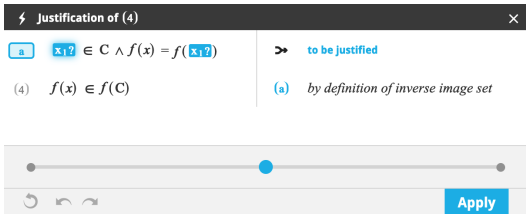
Figure A14.: Edukera resolution of question 1 in Exercise 4.1 (continued).

each on a separate line. Light blue brackets delineate variable or hypothesis scopes. The right-hand side shows the functional status of each line (in this case: declaration, hypothesis, or conclusion).

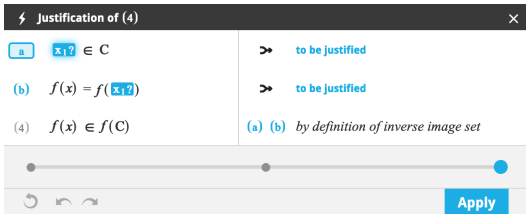
Moreover, in the right-hand side, the theoretical status of the statement is also shown: a statement is either justified or is a conjecture (as indicated by the text 'to be justified'), which in Coq terminology would be called a goal. When the statement is justified, the reason used for its justification is displayed, including the labels (statement numbers) of the corresponding premises and the type of justification used (definition, lemma, logic rule, etc.). Next to open goals, a double-tailed arrow (≫) allows the user to attempt unification of the goal with a previous statement appearing in the worksheet.



(a) No automation.

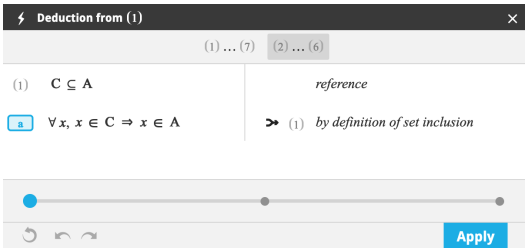


(b) Automatic treatment of existential quantifier.

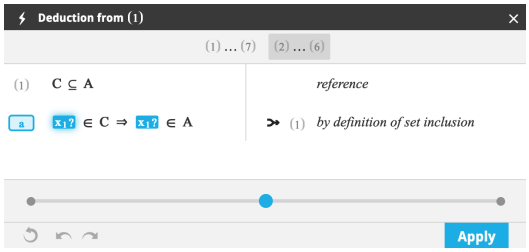


(c) Automatic treatment of existential quantifier and conjunction.

Figure A15.: Details of Edukera justification step: unfolding of set image.



(a) No automation.



(b) Automatic treatment of universal quantifier.

(c) Automatic treatment of universal quantifier and implication.

Figure A16.: Details of Edukera deduction step : unfolding of inclusion.

Each new state in the sequence shown is obtained from the previous state by selecting a statement in the worksheet (highlighted in blue in the screenshots) and clicking the appropriate buttons of the user interface. No text is input manually by the user during this sequence. Instead, in some cases Edukera displays modal boxes offering options for the application of a rule. For instance, Figure A15 shows three possible ways to unfold the definition of the image of a set[21] in order to justify the goal $f(x) \in f(C)$ (Cf. Fig. A13d). Note that this is a *backward* reasoning step, i.e. the modification of an open goal. The user may select among three possible automation levels using the slider at the bottom of the modal window, each level corresponding to the automatic treatment of a logic operator.

In contrast, Figure A16 shows a modal window displayed when performing a (forward) deduction step from hypothesis $C \subseteq A$, unfolding the definition of set inclusion (Cf. Fig. A14f). When performing forward deductions, the user may have to select a scope in which to insert new statements. In this context, scopes are represented by intervals of statement numbers (as can be seen at the top of each window in Fig. A16). In the current case, only scope (2) ... (6) is suitable in order to finish the proof, because variable x does not exist in the outermost scope.

Note that Edukera automatically introduces meta-variables in places where some unknown expression should be, in particular when justifying an existentially-quantified statement (Cf. Fig. A15) or deducing from a universally-quantified one (Cf. Fig. A16). Meta-variables can be considered as place-holders for unknown expressions meant to be filled in later in the construction of a proof. Such variables are named automatically when they are introduced (x_1 in the present case). In order to complete the proof, the user *has* to instantiate these variables, either by typing them in (directly or using the provided visual keyboard) or by identifying (indeed *unifying*) the statement where they occur with another statement. This may be done either from the worksheet itself, or directly from the modal windows A15 and A16.

We conclude this overview with a completed worksheet for question 3 of Exercise 4.1, shown in Fig. A17, including the detailed proof for question 2. Note that line (11) in this worksheet is justified by a direct call to a lemma established in the Edukera exercise for question 1.

A.6. D∀∃duction

Questions 1 and 2 of our exercise 4.1 are available as two entries in the 'sets and maps' exercise file (in French, Cf Fig. A18). Resolution steps are provided for Question 1 (Cf. Figs. A19 and A20). Figure A23a shows the dynamic proof tree view and Fig. A23b the proof outline for question 1.

[21] There is a typo in Edukera, the justification should indeed be relative to the image of a set, and not the inverse image.

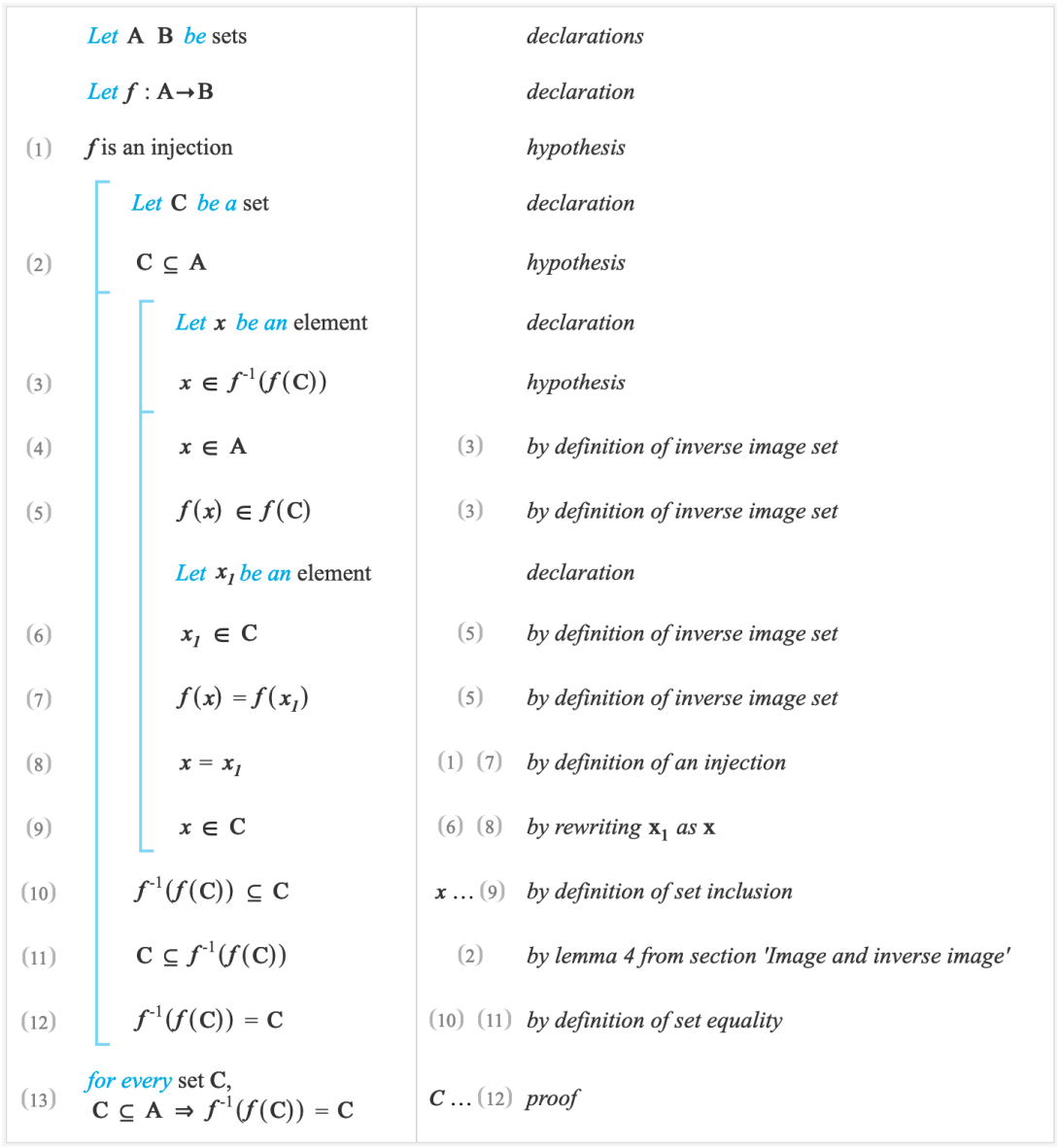


Figure A17.: Completed Edukera worksheet for Exercise 4.1 question 3.

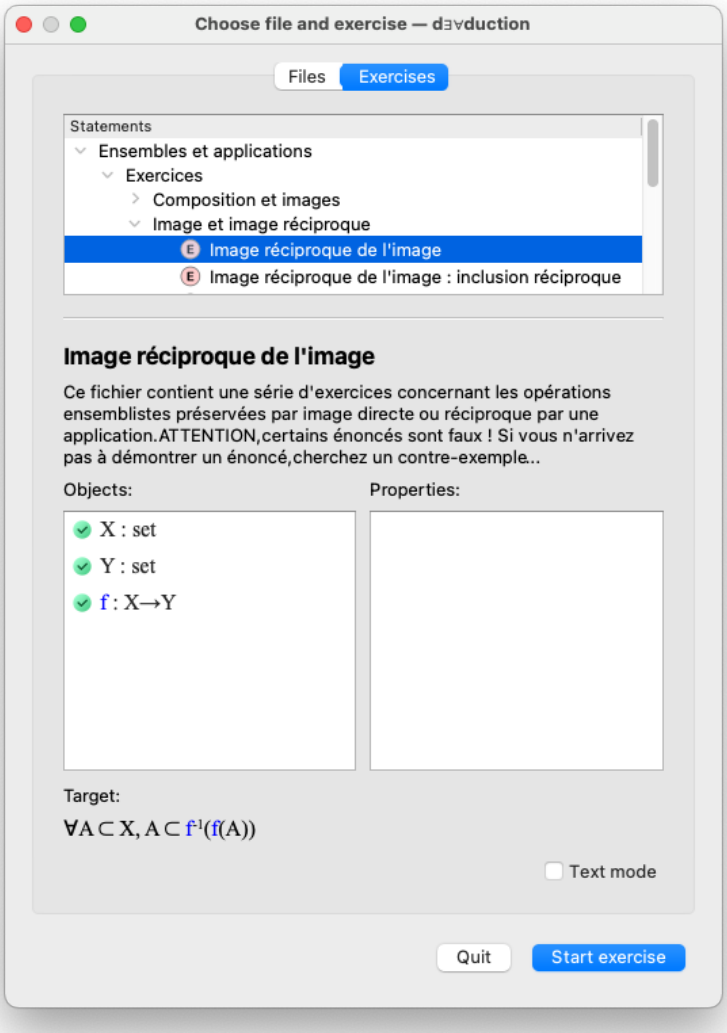
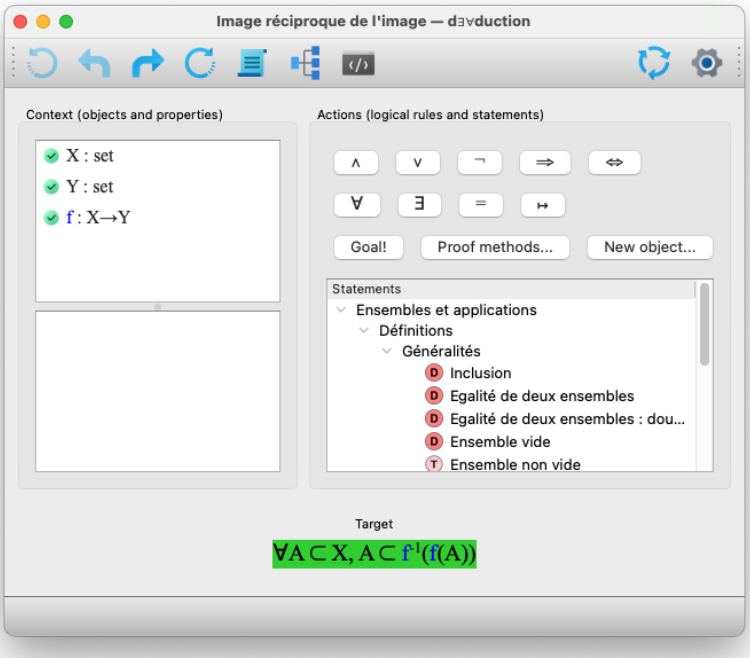
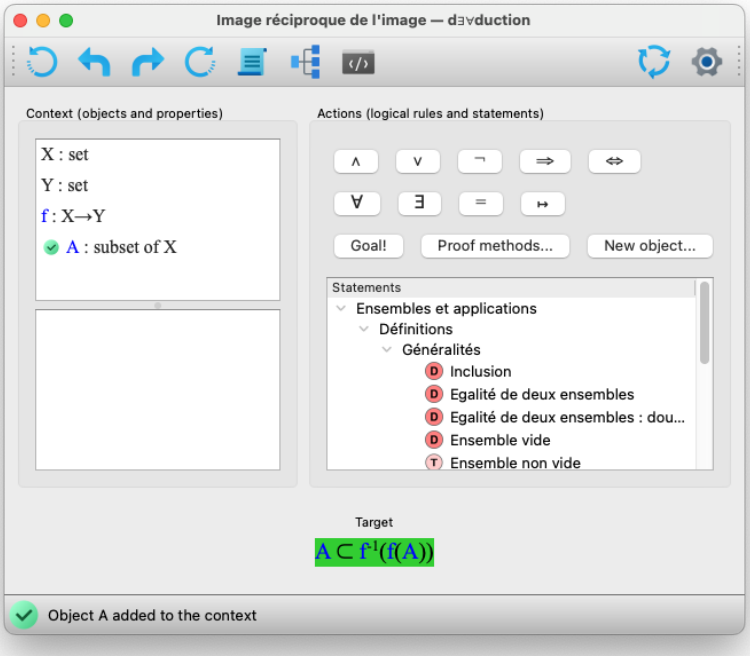


Figure A18.: Exercise selection dialog in D∀∃duction.

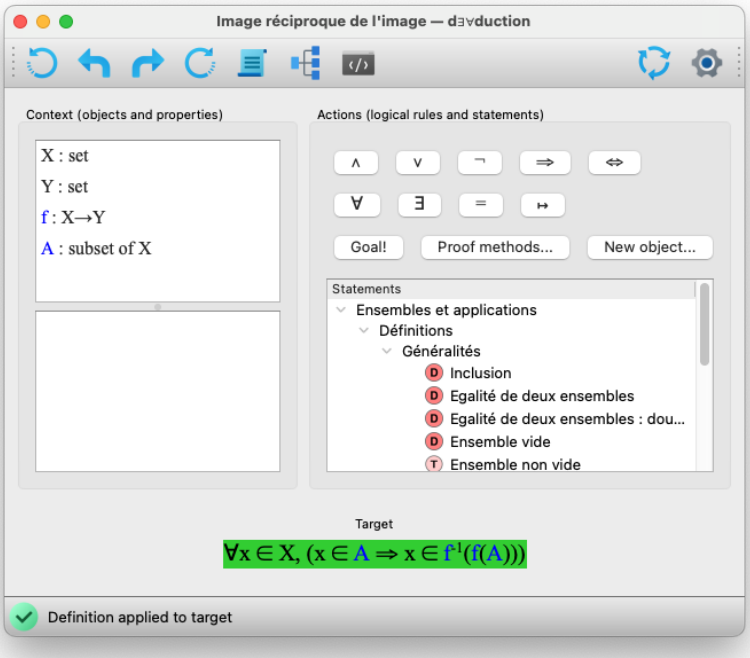


(a) Initial state.

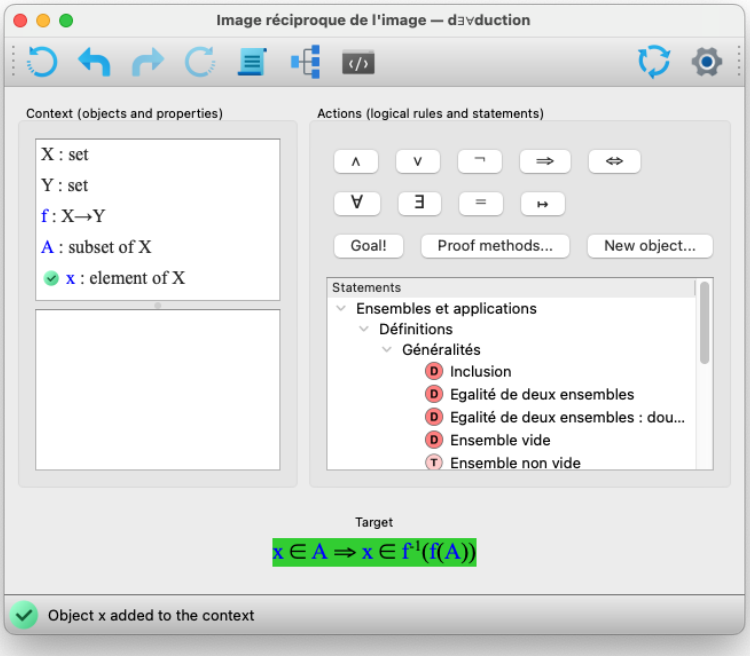


(b) Introduced hypothesis $A \subseteq X$.

Figure A19.: D∀∃duction resolution of question 1 in Exercise 4.1 (first part).

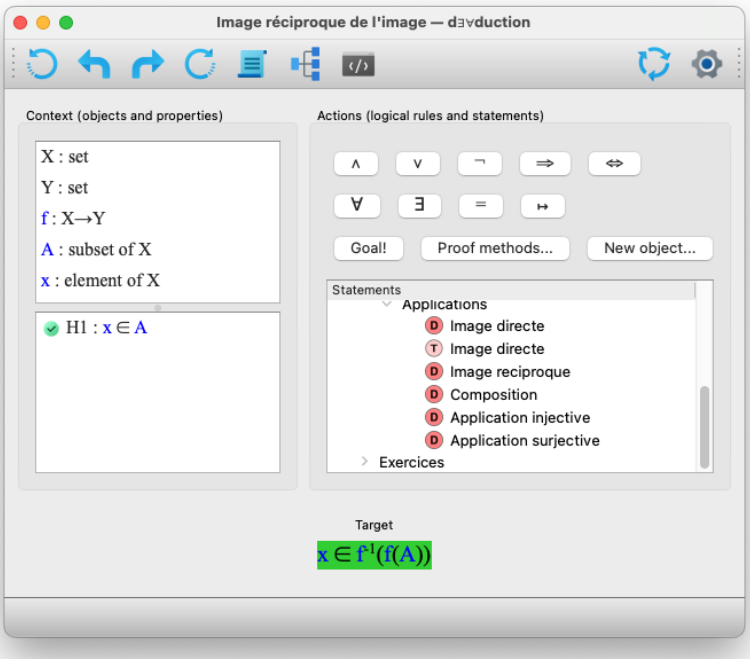


(a) Unfolded $\subseteq$ in the goal.

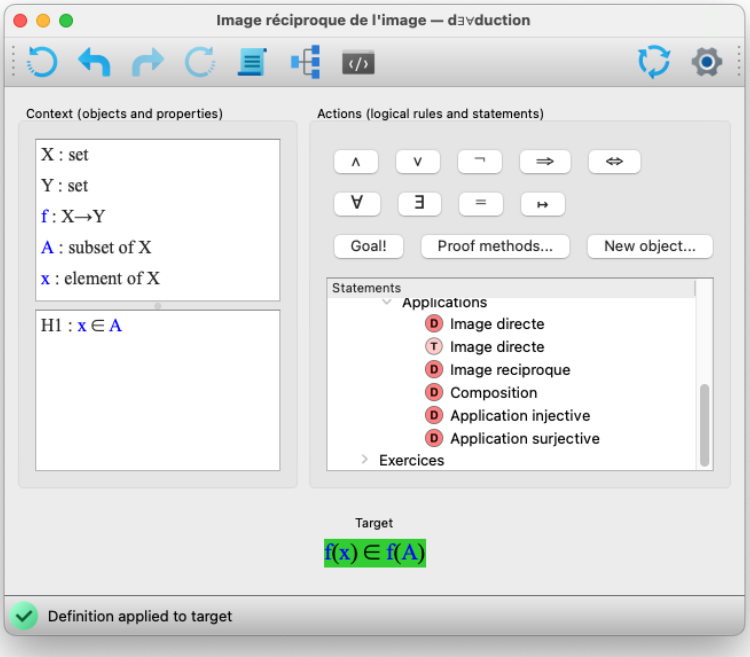


(b) Introduced $\forall$, new object x.

Figure A20.: D$\forall\exists$duction resolution of question 1 in Exercise 4.1 (second part).

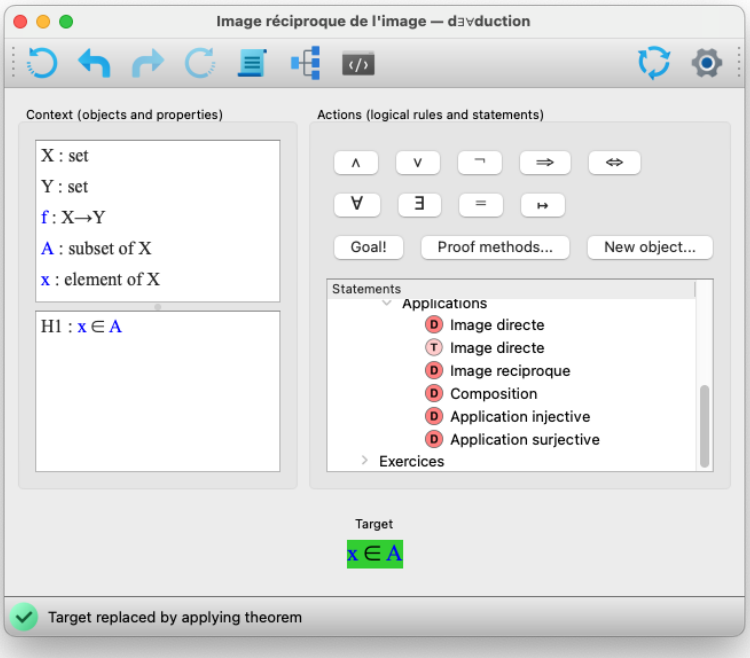


(a) Introduced $\Rightarrow$, new hypothesis $x \in A$.

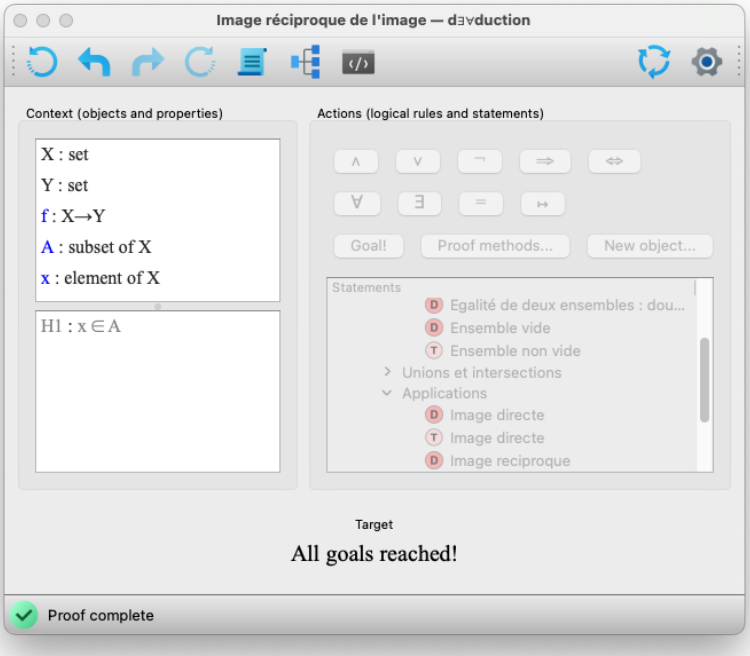


(b) Unfolded f^{-1} in the goal.

Figure A21.: D∀∃duction resolution of question 1 in Exercise 4.1 (third part).

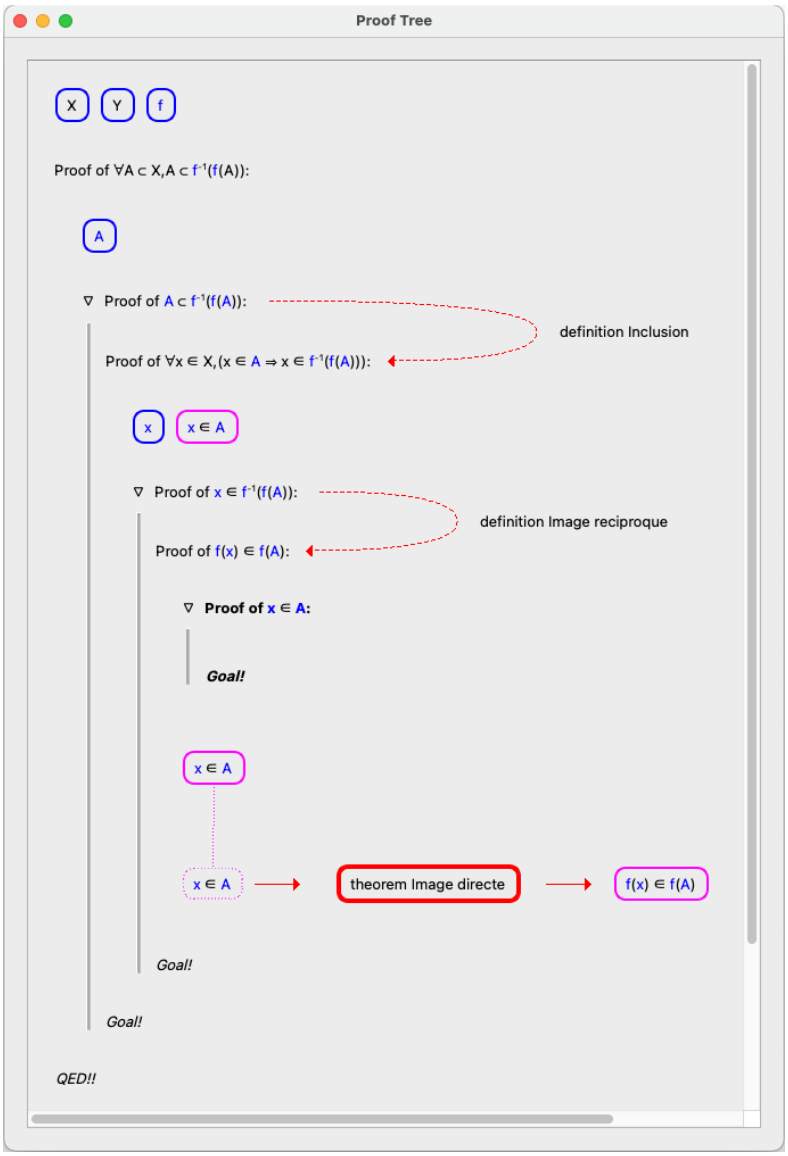


(a) Applied theorem $x \in A \Rightarrow f(x) \in f(A)$.

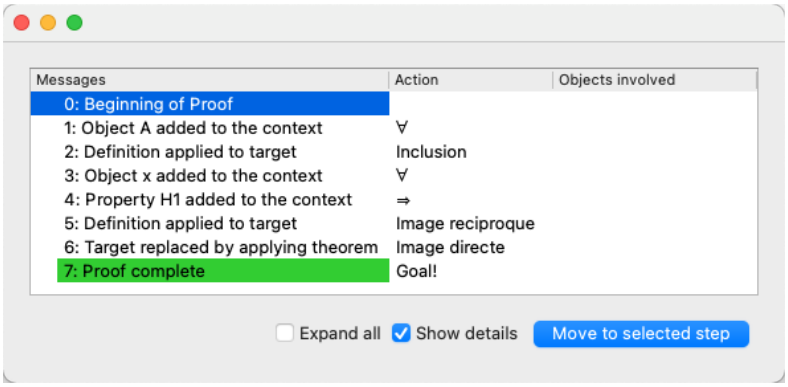


(b) Unfolded $\subseteq$ in the goal.

Figure A22.: D∀∃duction resolution of question 1 in Exercise 4.1 (end).



(a) Proof tree.



(b) Proof steps outline.

Figure A23.: Proof visualization for question 1 of exercise 4.1 in D∀∃duction.