Research report - Proof assistants for teaching: a survey

Frédéric Tran Minh, Laure Gonnord, Julien Narboux

Research report
Proof assistants for teaching: a survey

Frédéric Tran Minh[1], Laure Gonnord[1], and Julien Narboux[2]

[1] Grenoble INP - LCIS
[2] ICube - UMR 7357- University of Strasbourg

Abstract

In parallel to the ever-growing usage of mechanized proofs in diverse areas of mathematics and computer science, proof assistants are used more and more for education. In this talk, we will provide a survey of the different works related to the use of proof assistants for teaching. This includes works where the authors report on their experiments using proof assistants to teach logic, mathematics or computer science, as well as designs or adaptations of proof assistants for teaching. We provide an overview of both tutoring systems that have been designed for teaching proof and proving, or general-purpose proof assistants that have been adapted for education, adding user interfaces and/or dedicated input or output languages.

Introduction

The story of proof assistants originates in 1968 with Automath developed by De Bruijn [24]. Two categories of software can be identified.

Automatic Theorem Provers (ATP) take a statement as input (often expressed in a language supporting First Order Logic) and act as a black box to determine if the statement is universally true or if a counter-example can be produced. CVC-5, Z3, Vampire, Superposition are some famous examples.

On the other hand, Interactive Theorem Provers (ITP) check for validity formal proofs designed and written by a human user in a dedicated programming language, as expressive as possible (i.e. supporting high order logic (HOL) or dependent type theory (DDT)). In ITPs, some automation is provided to help the user fill in parts of the proof considered trivial by a human being. The proof state (current goal and hypothesis) is displayed at any point of the proof.

Since the 1960s, numerous Interactive Theorem Provers, also called Proof Assistants, have been developed. Among the most famous still widely used nowadays are Mizar [76], Agda [74, 83], Coq [37, 32], HOL-Light [49] Isabelle [82] and Lean [40]. All these systems mainly differ in the size of their kernels, the underlying foundations and the expressiveness of their language[1].

These ITPs have been successfully used in mathematics to formally verify proofs containing computer code (famous examples are proofs of the four-color theorem by Appel and Haken in 1976, formally verified by Gonthier in 2008 [48] and the Kepler conjecture - Hales theorem, proved by Hales in 1998 following Toth in 1953 and formally verified by the FlySpeck project in 2006 in Isabelle [52, 53]), and also in software verification (an example is the C compiler CompCert verified by Coq in 2016 [67]).

The growing role of formalization in mathematics and computer science (the most famous example being the formalization and proof of the four-color theorem [48]) has at the same time

[1] As the detailed description of each of these proof assistants is out of the scope of this paper, the reader can refer [47, 78, 113] for a survey about proof assistants.

stimulated the use of proof assistants in education, especially for teaching mathematics, logic, and computer science. In this talk, we will provide a survey of the different works related to the use of proof assistants for teaching.

Section 1 exposes a variety of teaching experiences using proof assistants, from undergraduate mathematics to quasi-expert manipulation of program proofs. Section 2 describes adaptations that have been made to adapt the tools themselves, and also their user interfaces, to the special needs of education. We dedicate Section 3 to natural language interfaces.

1 Teaching experiments using proof assistants

From the 70's, with the first reported experiences using Mizar, to now, the literature is full of experiences that somehow reflect the existing status of proof assistants: their expressiveness, of course, and also the publicity that often comes from their usage in research developments. In this section, we report a variety of teaching experiences using teaching assistants, at different levels from undergraduate to master, and targeting various topics, from logic to more advanced formal verification of programs.

1.1 Teaching logics and meta-theory

The first experience of teaching (propositional) logic using proof assistants dates back to 1975, with the Mizar software. Matuszewski, Rudnicki and Trybulec developed an environment based on Mizar dedicated to teaching logic [106]. This experiment, conducted at the Warsaw University, is described in the more recent paper [76].

Similarly, other proof assistants have been widely used in the last 40 years as computer artifacts to support teaching logic-related topics such as propositional logic, induction, or set theory. The Coq Wiki [101, 102] reports a list of such experiences with Coq.

Teaching regarding logical and predicate logic can rely on relatively low additional machinery as proof assistants expose natively their underlying logic. Exercising natural deduction, sequent-calculus is thus relatively straightforward. Other authors such as Villadsen and its co-authors have proposed using proof assistants with a deep embedding of some calculus, allowing formalization of meta-theory to teach both logic and automated deduction [107, 44, 108].

1.2 From logics to mathematical proofs

Mathematical proofs are not all captured by the manipulation of logical artifacts, and the previous teaching experiences, although beneficial since they enable the formal (and quasi-syntactic) manipulation of proofs, quickly show their limit as they do not capture all subtleties of even undergraduate mathematics. Many experiences thus relate courses and even sequences of courses elaborating on a continuum of concepts from propositional logics to analysis via set theory, functions and relations, number theory, calculations and induction.

Following the initial experiment at University of Warsaw cited in Section 1.1, Retel and Zalewska relate [90] that different flavors of Mizar were then used, mostly at the University of Warsaw, but also at other institutions, to also teach axiomatic geometry (1985), introduction to mathematics (1987), topology (c.a. 1990). In the same paper, the authors describe their own teaching experience with 1st-year students in Bialystok learning formalization of mathematics, logic, and set theory with Mizar.

Blanc, Giacometti, Hirschowitz and Pottier proposed in 2007 a software learning environment based on Coq [17] to teach undergraduate mathematics.

In [58], the authors report on four different experiments teaching mathematics in Orsay, Montpellier, Paris, and Strasbourg using Coq, Edukera and Lean.

From 2014 to 2018, Avigad, Lewis, and Van Doorn taught an undergraduate mathematics course where proofs in natural language, symbolic logic and natural deduction, and the language of the Lean3 theorem prover were presented as 3 distinct languages to avoid confusion between all these activities. However, they "emphasize that the latter two components are carried out in service of the first." [8]. The course covered the following topics: propositional logic, elementary set theory, relations and functions, natural numbers and induction, elementary number theory, finite combinatorics, construction of real numbers, and theory of infinite, but only the first four chapters were worked out using the Lean3 prover. At the Sorbonne University, a front-end to Lean (Deaduction [58]) was also used as an executable support to teach 1st-year undergraduate maths (algebra (sets, functions), analysis (limits, continuity)). [105] reports the development of a 1st-year undergraduate mathematics course at Université Grenoble-Alpes in 2022, complementing the main math class, where students used in parallel Edukera [93] and Lean3 to explore the mathematics formal language, propositional logic, natural numbers, functions, and an overview of real analysis (numerical sequences and limits). Notably, the originality of this course is the dual navigation between two different tools.

At Fordham University, Heather MacBeth ran a 1st-year undergraduate mathematics course entitled "The Mechanics of Proof", [73] as a sequence of worked examples and exercises. Each statement is proved first with a natural language style, and then in Lean4. The course starts with very elementary "proofs by calculation", and progressively introduces logic connectives (first with familiar examples, with numbers), and then natural numbers and induction, divisibility and number theory, functions, sets and relations.

These individual experiences all rely on the fact that proof assistants provide a way to check student proofs, with various possible adaptations that we shall develop later (Section 2). Nevertheless, there are relatively few analyses of the impact of such new student skills on the concrete ability to understand and write classical "pen and paper" proofs. One such notable work is [18], in which the authors report a precise pedagogical method introducing three intermediate levels of formality to transition from a Coq proof to a textbook proof advocating that "the difficulty and the value of transferring the skill of proving in a proof assistant to the skill of textbook proving is extremely underestimated.". In [9], Bartzia et. al. studied the potential impact of different features of proof assistants. Thoma et. al. [104] report on the effect on proof production and proof writing after using the Lean proof assistant. They conclude on clear progress but not for all the problems related to the development of textbook proofs: they conclude that at least for computer science students the formalism is not the main difficulty in learning proof, and that starting undergraduate studies with the use of "a suitable proof assistant" is valuable.

1.3 Proofs assistants and program verification

Proof assistant machinery beyond propositional logics is also taught "for itself": as some examples, the Coq wiki [102] mentions: a Theoretical Foundations of Theorem Proving Systems, by Paweł Urzyczyn, in 2006 at Warsaw University; and also a class on Automated Theorem Proving, conducted by Andrew W. Appel, in 2007 at Princeton University. There are also classes about Coq in Strasbourg and many other universities.

More interestingly, the increasing usage of proof assistants in the formal verification research community had a strong impact on the evolution of graduate programs in computer science. In this particular domain, the proofs are all done by induction on the syntax of the programming

languages which consist of many cases that are easier to deal with using a proof assistant, and the research specialists have made their graduate courses evolve in the same way as their research activities. As few examples, the proof assistant Coq has for example been used at the ENSTA french engineering school, for a course on software safety; and at ENIEE-CNAM for a master-level class on formal methods and specification languages.

Proof assistants are here used as a way to automatize proofs checks, or semantics specification, but not for themselves, as advocated by Tobias Nipkow that uses HOL/Isabelle (and the Isabelle/Isar high level language [111] for proof scripts) as a support of its graduate course about semantics and Hoare's logic [81].

Similarly, [43] reports and compares a few experiences teaching formal verification with various proof assistants such as PVS (the Prototype Verification System). Bertot has also given a course about programming language semantics using Coq [14].

In 2005, Delahaye reported on an experience of teaching an introduction to programming languages semantics course with Coq to fifth-year students [41]. The course was associated with a workshop about the design of certified software. The author asserts the students benefited from the practical nature of the course, finding it easier to start programming while receiving instant feedback than to specify and prove theorems on paper without feedback.

Perhaps the best-known material in this line of program verification and foundations of programming languages is the series of 6 volumes entitled Software Foundations [86], initiated by Benjamin Pierce, which served as a basis for a software verification course he gave at University of Pennsylvania [85], and many courses after him over the world.

Finally, Coq and other proof assistants are at the heart of the courses and seminars on program reasoning, coordinated by Xavier Leroy in 2020, at Collège de France [66].

2 Specializing proof assistants for education

The variety of teaching experiences, in terms of purpose, types of students, and the studied area, has unsurprisingly led to the observation of the necessity of a specialized proof environment. The teachers face the following difficulties: the necessity to teach a new specific language as a premise of the course itself; the proof steps granularity, which often is too fine grain for the desired effect; the action of transferring to full "pen and paper" proof; and finally, the question of feedback.

In this section, we review some proof assistants specifically dedicated to education (Section 2.1); some interface adaptations to state-of-the-art proof assistants (Section 2.2); and then enumerate some tutoring fonctionalities based on the improvement of feedback added to existing proof assistants (Section 2.3).

2.1 Some proof assistants dedicated to education

Teachers, or didacticians, have designed proof assistants specialized to their educational purpose. A special focus is placed on the graphical user interfaces (GUIs), on tutoring, and on simplified language interfaces. As a special case, the proof assistants dedicated to geometry form a huge part of this category.

Logics In the textbook "A logical approach to discrete math", by Gries and David [51], the authors develop a mechanized simple language to express the solutions of exercises in logic. Realizing that students needed more feedback, Wolfram Kahl developed in 2011 CalcCheck,

a proof-checker able to verify if LaTeX-formatted sheets expressing proofs in this language are correct [55].

Jape [19], developed by Surin and Bornat at the University of Oxford in 1996, is a proof editor in first-order logic integrating a graphical user interface, designed to be easy to use by beginners. The interface displays lists of definitions and conjectures, it can display proofs in a boxed Fitch-style.

PhoX is a proof assistant with High-order logic (*à la* HOL/Isabelle) since 2018 by Raffali specifically to teach logic to 3rd-year and 4th-year master students [89]. PhoX provides a graphical user interface through XEmacs, thanks to its support by ProofGeneral [7], over a limited number of tactics, designed to ease usability.

There also exists some educational point-and-click environments aimed at solving logic puzzles, in propositional and first-order logic: among them, we can cite *The incredible proof machine* by Breitner [20], the *Logic Puzzle* proof game of Lerner [65], QED by Terence Tao [98] or Carnap by Leach-Krous [63].

Undergraduate mathematical proofs PhoX [89]'s environment also comes with some analysis topics (limits continuity, intermediate value theorem), topology (properties of closures, of connected sets), and algebra and number theory (symmetric group, rings, prime numbers), thought for undergraduate students.

Lurch is described by its authors, Carter and Monks, as a word processor enabling to typeset maths [31], equipped with verification features designed for student use. In a graphical environment, the user enters a mathematical text in natural language, but selects specific expressions to mark them as "meaningful". Lurch verifies the logical coherence of meaningful expressions and provides feedback, and ignores the rest of the text. Three levels of input constraints and automation are available: formal proofs, semi-formal proofs, and expository proofs. The teacher can prepare a dedicated rule set (logic system, axioms). The system can act as a tutor, grading, coaching and providing hints. Two experiments using Lurch with students were conducted [30].

Diproche ("DIdactical PROof CHEcking") checks student solutions to proving exercises written in a controlled natural language [29] (The authors present it as an educational variant of Naproche [61]). Its language is adapted to specific areas common in introductory courses to proof method, and supports exercises in propositional logic, set theory, functions and relations, number theory, axiomatic geometry, and elementary group theory. An ATP is used to provide feedback to check inference steps, detect typical error patterns and provide counterexamples to false claims. The set theory number theory modules of the system were used in 2020/2021 in a first-semester course at the University of Flensburg.

Edukera has been developed in 2016 by Rognier and Duhamel [93]. It appears as a web interface: the user interacts with the software by pointing and clicking, there is no text-based language to typeset. A database of 900 exercises is provided, including 200 formalization exercises. In formalization exercises, the student is proposed a sentence in French or English (like "Someone read all books by Victor Hugo"), and is asked to formalize it using quantifiers, variables and given predicates; to do so, the user has to click on connectives in a prefix order. The resulting answer is dynamically displayed. In proof exercises, a statement is to be proven. To show the goal, one must select suited rules to apply (introduction, elimination, apply lemmas, ...); the current state of the proof can be displayed in Fitch style or Gentzen sequent calculus style.

The special case of geometry Several e-learning environments have also emerged to support proof learning in the context of synthetic geometry in secondary education. They differ from general-purpose proof assistants for two reasons. First, the feedback and error messages are designed for use in the classroom. Second, the systems are often not of general purpose from a logical point of view: there are proper proofs that can not be formalized in these systems (using analytic geometry, using complex numbers, ...), the user can not introduce its lemmas or definitions. These geometric systems do not deal with degenerated cases and hence can present proofs without case distinctions as a list of modus-ponens steps along with figures displayed with embedded dynamic geometry software. They usually do not allow reasoning by contradiction. There are many such software such as AgentGeom [34], Baghera [110], Chypre [12], Cabri Euclide [70], Geometrix [50], Geometry Tutor [4], Geometry Explanation Tutor [2], Mentoniezh [88],QED-Tutrix [64], Turing [91]. They differ in many aspects such as the feedback they provide to students' errors and how they can provide hints by computing the distance from a pre-defined list of solutions or using key steps provided by the teachers, the way proofs are stored behind the scenes, if they separate the exploration and problem-solving phase of the exercises, ... For a survey (in French) comparing these systems from the didactic point of view see [100].

2.2 Development of user interfaces on top of proof assistants

User interfaces make sense particularly for teaching purposes, as they partly respond to essential challenges: interactivity and visualization are essential to understand the structure of proofs, and provide feedbacks; and graphical helpers could help to manipulate the underlying sometimes complex language.

As programming languages, proof assistants are "naturally" equipped with graphical user interfaces and development environments. In addition to editing facilities, the ability to update the proof state was their minimal interactivity requirement. Individual proof assistants often have their own graphical user interface, like CoqIDE for Coq, XIsabelle [27] or Isabelle/jEdit for Isabelle. Web interfaces (jsCoq [45] for Coq, Clide [71] for Isabelle, The Lean 4 web editor [80] for Lean) have the advantage to facilitate maintenance or adoption to large classes. Other web interfaces include Alfie [103], CoqWeb [17], Waterproof [87], ProofWeb [112], Trylogic [99], PeaCoq [92].

Among all these interfaces, the Emacs-based front-end ProofGeneral is the most generic front-end for interactive assistants; which supports more than 10 proof assistants, including Coq, Isabelle, PhoX, HOL Light and others. The advent of Visual Studio also enabled the diffusion of provers plugins, with enhanced capabilities. As plugins become more pervasive, some APIs (Lean Proofwidgets [79]) have been designed for the participative evolution of interfaces for the most common provers.

An additional and required feature for education is the capability to provide a visualization of the proof itself: displaying proof trees, Fitch or Genzen style while manipulating these proof techniques. Our oldest reference is CPT [94], in which "Goal Tree display" was designed with ergonomy in mind when proceeding forward and backward proofs. Various proof displays are available for instance in Proofweb, Deaduction [58] and Paperproof [57]. Paperproof is a Visual Studio Code extension for Lean that displays a boxed proof tree-like representation of the current active - possibly incomplete - proof. With a clever use of nested boxes, colors and curved arrows, it manages to render backward chaining as well as forward chaining and variable scopes. On the proof presentation side, JsCoq and Waterproof provide a "mixed document" feature that allows the formal proof source code to be embedded into formatted HTML or

LaTeX content.

Bertot, Thery and Kahn conceived in 1994 a "proof by pointing" algorithm [16] which associates the backward application of an inference rule and a residual sub-expression to each possible user selection of a sub-expression of a goal sequent. The implementation results in a graphical interface that enables the user to direct her proof by selecting sub-expressions, the tactic corresponding to the inferred rule being automatically applied subsequently to the mouse selection The software has been implemented in Coq and for other proof assistants (Isabelle, HOL). These principles were later embedded in a wider multi-process interface first called CtCoq [13], implemented in Lisp ; PCoq (2001-2003) is a reimplementation in Java [3].

This concept of "proof by pointing" has been used in a variety of proof assistants, used or not for educational purposes: CtCoq, Papuq [33], CoqWeb, Deaduction, Waterproof, ProveEasy [26], for instance, proposes such a feature.

Among proofs, interfaces have been developed to help visualize the objects "under proof" themselves. Geoview, an extension of PCoq designed for geometry [15] and GeoProof [77] for instance, displays geometric figures. ProofWidgets' supplied examples include graphical presentations like commutative diagrams, Venn diagrams, geometry diagrams, animated plots of parametric functions, and graph representations.

Some extensions have been specially designed to target teaching activities. Some are rather technical yet important: ProofWeb, Trylogic, ProofLab/AProS [96] as a few, provide access to exercise database that can be enriched by the teacher; and/or link to existing classical material to understand a particular proof technique. Deaduction's database contains exercises in logic, set theory, functions and relations, and basic analysis, as needed for a first-year undergraduate course. The Theorem Proving Environment (Epgy project) [97], ProofWeb, and the Trylogic interface to Moodle provide the ability for the teacher to replay student proofs or track student progression. Finally, PROOFBUDDY [56] collects "fine-grain" data about the way students interact with the Isabelle proof assistant.

Other adaptations include "next step guidance" (some of the software cited in "proof by pointing", as well as ETPS [5], CPT); eventually based on "proof planning" technique: X-Barnacle/CLAM [69]; Ωmega Tutor (dialog project) [95]; Tutch (Tutorial Proof Checker) [1].

Theory adaptations, simplifications Papuq is an extension of CoqIde designed for teaching logic and set theory to first-year students at Warsaw University. The authors first propose to expose only the aspects of Coq type theory that are relevant to teach to students, and slightly adapt its vocabulary. They discuss the presentation of sets as predicates, partial functions, relations and functions as primitive objects and functional relations. They call these adaptations "Naïve Type Theory" after Constable [36] and Kozubek [60].

The formalization and manipulation of functions over a subset of a type universe (typically: a subset of $\mathbb{R}$) or undefined terms (e.g. : undefined division result, limit, derivative, supremum) with rough proof assistants is known to be an additional burden irrelevant to the study of undergraduate mathematics. Several authors tried to tackle this difficulty in a teaching context. Sommer and Nuckols proposed [97] to handle functions and relations defined on a partial domain (e.g. $x \mapsto 1/x$) by generating side conditions either automatically justified or returned to the user as "proof obligations" to be proved, the management of the proof obligations being integrated into the internal system of inference rules. The method has been implemented in the Theorem Prover Environment (Epgy project). Coen and Zoli adopt a theoretical modelization of undefined terms by elements of a partial setoid over an asymmetric relation of "quasi equality" [35]. They implemented their idea in the Matita theorem prover [6].

The CoqWeb interface for Coq was designed to target teaching activities. It features proof

by pointing and clicking like CtCoq, but does not, on purpose, infer the right tactic from a pointed expression, since "it was too much an invitation to click without relating it to any concept". Instead, the student can choose from a list of possible moves given in informal natural language, each of which, when clicked, triggers a specific Coq tactic. The original available Coq tactic library has been restricted to the minimum necessary for educational purposes, and some of them have been adapted to provide modes: in student-guided mode, a hint is provided for the next move; in teacher mode, expert teachers can declare definitions, axioms and exercises.

Other tools (Waterproof, ProveEasy, TPE/Epgy, WinKE [39], ETPS) also provide a teacher mode to provide additional (simplifier) rules and axioms. Finally, Ωmega Tutor, TPE/Epgy and Tutch provide automation to adapt proof step granularity. proof step granularity.

2.3 Improving the feedback

A major advantage of proof assistants is that they provide immediate feedback on the validity of a proof step. Error messages returned by the assistant are generally not usable by students: the feedback is therefore binary. A crucial issue if proof assistants are to be used in teaching, is that they should be able to provide feedback that sheds light on the error made. Below are some responses to this problem.

The proof tutor Tutch of Abel, Chang and Pfenning consists of a proof language and a command line proof checker program which can annotate proof steps; in case of erroneous or insufficiently justified steps, the program provides error messages, and assumes them to continue checking to the end of the input. Terms of the proof language are structured by bracket-surrounded frames corresponding to the scope of introduced symbols or hypotheses. Several flavors (from proof-term style to assertion level via declarative style) of the language are available, depending on the teaching activity purpose.

Inspired by the results of the Dialog Project (2001) [11], Dietrich, Benzmüller and Schiller integrated a tutoring module to the Ωmega proof assistant [10, 95]. The system invites the student user to a dialog in natural language where she submits her next proof step. The system replies to the student proposition along three criteria: correctness, proof granularity, or relevance. In case of too coarse-grained step, the user is prompted for an additional justification.

The idea of introducing "buggy rules" into a problem solver to model bugs in algorithms or fallacies in students' reasoning seems to originate in the late 1970s with Brown, Burton and Larkin [22, 23]. The authors conceived a system for synthesizing a model of pupil misconceptions in school basic arithmetics, and applied it by designing for student teachers a game simulating the possible errors of a pupil [21]. Following the same idea, Farrell et al. designed a tutor to teach how to program in Lisp [42]. This work later inspired Zinn in the 2000s when developing Slopert aimed at diagnosing errors in symbolic differentiation. Recently, Diproche named anti-ATP [28] the idea of trying to verify students' erroneous proof steps with an ATP with false rules or axioms as input. Different types of false rules (logical fallacies, axioms of erroneous distributivity, commutativity, monotonicity, or rules of false analogy), based on teaching experience, lead to the prediction of different possible types of mistakes, thus providing precise feedback to the students.

3 Proof output or input in natural language

Proof assistants have often been criticized for the fact that their language, which is essentially a computer language, is too far away from the vernacular of mathematics, and that it requires an excessive depth of detail in the justifications, masking the key ideas of proofs. These peculiarities

put off both the mathematical community, which was reluctant to commit to formalization, and the teaching community, which saw the use of proof assistants as ineffective in transmitting the language habits of mathematicians.

Based on these observations, many authors have sought to bring the language of proof assistants closer to the natural language of mathematics, in two main ways :

- Using controlled natural languages (CNL) [62] as input. These are languages that have the rigor and precision of a programming language, but whose vocabulary, syntactic structure, and tolerance for ellipses and missing steps allow them to resemble natural language; their interpretation generally requires a high level of automation.
- Conversely, converters have been developed to convert a proof script in the native language of a proof assistant into a "human-readable" language emulating the vernacular language. The presentation of the output can be formatted, for example in LaTeX. The structured nature of the proof term and the use of interactive formats (html, javascript) allows a level of detail to be selected by the user.

3.1 Translating formal proof into natural language

Ωmega , implemented in Lisp by Benzmüller [10] is based on a concept of "proof planner" which consists in identifying different levels of granularity in the proof, using forward and backward state-space search. It supports ATP integration (Otter) and embeds a computer algebra system (CAS) to help extract proof plans. It relies on Proverb [54] to output proofs in a natural language at a user-selectable level of abstraction.

Theorema is an environment been developed since 1995 by Buchberger and then by Windsteiger [25, 115] on top of Mathematica (and in the language of Mathematica scripts), which aims at associating theorem proving with computing and solving facilities. It generates proofs for statements expressed in untyped higher-order logic, in a style that imitates natural language, but seemingly still supports only a few theories (eg predicate logic, induction on natural numbers).

In 2013, Ganesalingam and Gowers wrote a program that proves elementary undergraduate-level statements trying to produce a "human-style output" [46]; to do so they try to analyze and reproduce typical stereotypes mathematicians apply when faced with these kinds of problems. The outputs of the program were mixed with human solutions written by an undergraduate student and a PhD student, and submitted to the internet community to guess which of them were auto-generated: according to the authors, the results were encouraging in that "the program did reasonably well at fooling people that it was human".

3.2 Controlled Natural Languages (CNL)

The first step in this direction was the use of a declarative style [114] in place of procedural style as input in interactive theorem provers. While the latter consists of a sequence of imperative orders constructing the proof term (the intermediate statements can generally be omitted or are considered as simple type ascriptions; they can be computed and dynamically displayed by the software), the former expects the user to enter a list of statements, like in a traditional mathematical text, the proof terms justifying these statements being essentially automatically computed, even if the user can provide hints.

Mizar [76] is probably the first famous interactive theorem prover to accept a declarative style script. The Isabelle community followed when Wenzel developed the Isar extension in 2007 [111].

In the 1960s, Glushkov carried out research aimed at designing a system featuring a low-level inference engine complemented by a high-level reasoner, and parsing a powerful input language close to the natural mathematical language. These objectives stimulated the work of several research teams in Kiev in a program called "Evidence Algorithm" [72]. By 1990, a language (TL - Theory Language) and a parser and prover (SAD - System for Automated Deduction) were developed in a Russian version. This work was later taken up by Andriy Paskevych in Paris during his PhD thesis [84]. TL evolved to ForTheL (Formula Theory Language), SAD was rewritten in English, the inference engine based on external ATPs, and theoretical advances enabled the development of the reasoner. Independently, Bonn, Schröder, Koepke, Kühlwein and Cramer were pursuing similar goals - a controlled natural language associated with an adapted prover [61, 38]. The Naproche system was developed during Cramer's PhD thesis in 2013 and later merged with SAD, enriching the language ForTheL and extending the parser to support LaTeX input [59]. The program has been integrated into Isabelle IDE and benefits from the user interface and the access to the ATPs but do not share any logical connection with Isabelle prover [68].

On top of Ωmega discussed above, Autexier and Wagner developed PlatΩ, a "mediator between text editors and proof assistance systems" [109] which allows the user to enter her statements in a CNL named PL (proof language). PlatΩ converts the input into a format understandable by Ωmega ; in return, hints or completed proofs from Ωmega are converted by PlatΩ into PL to be displayed by the editor (Texemacs).

Lean Verbose is a controlled natural language in development since 2021 by Patrick Massot, implemented as a set of user-defined tactics in Lean4 [58, 75]. According to the author, using this tactic language is not easier to learn than the core Lean language but using it improves students' performance while transitioning to handwritten proofs.

With Waterproof [87] (2018), Jim Portegies and Jelle Wemmenhove imagined a custom tactic language on top of Coq that on the one hand requires the user to respect additional constraints (like type ascription, signposting, or distinguish "take" from "assume"); but on the other hand, it relaxes the requirements on trivial steps justification. Note that a hint tactic is provided to suggest the introduction of the main connector of the goal and error messages have been refined. The software has been tested for four years at Eindhoven University of Technology with first-year undergraduate students.

Conclusion

We have presented an overview of the use and design of proof assistants for teaching. Proof assistants have been used for a long time for teaching, and many tools have been designed. In some fields such as foundations of software and logic, some teaching material has been produced and shared, and the use of proof assistants is widespread. However, the evaluation of the impact of these tools from a didactic point of view and their use for teaching mathematics is still at its debut. A growing community of researchers is addressing these issues, and we are optimistic that using proof assistants for both teaching and research will be commonplace by the middle of the 21^{th} century.

Acknowledgments This work is partially funded by the ANR Project APPAM ANR-23-CE38-0009-01.

References

[1] Andreas Abel, Bor-Yuh Evan Chang, and Frank Pfenning. Human-Readable Machine-Verifiable Proofs for Teaching Constructive Logic. 2001.

[2] Vincent Aleven, Octav Popescu, and Kenneth R Koedinger. Towards Tutorial Dialog to Support Self- Explanation: Adding Natural Language Understanding to a Cognitive Tutor. 2001.

[3] Ahmed Amerkad, Yves Bertot, Loïc Pottier, and Laurence Rideau. Mathematics and Proof Presentation in Pcoq. In *Workshop Proof Transformation and Presentation and Proof Complexities in Connection with IJCAR 2001*, Siena, June 2001.

[4] John R. Anderson, C. F. Boyle, and Gregg Yost. The geometry Tutor. In *IJCAI Proceedings*, pages 1–7, 1985.

[5] Peter B. Andrews, Chad E. Brown, Frank Pfenning, Matthew Bishop, Sunil Issar, and Hongwei Xi. ETPS: A System to Help Students Write Formal Proofs. *Journal of Automated Reasoning*, 32(1):75–92, 2004.

[6] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. The Matita Interactive Theorem Prover. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, volume 6803, pages 64–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[7] David Aspinall. Proof General: A Generic Tool for Proof Development. In Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Susanne Graf, and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785, pages 38–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[8] Jeremy Avigad. Learning Logic and Proof with an Interactive Theorem Prover. In Gila Hanna, David A. Reid, and Michael de Villiers, editors, *Proof Technology in Mathematics Research and Teaching*, volume 14, pages 277–290. Springer International Publishing, Cham, 2019.

[9] Evmorfia Bartzia, Antoine Meyer, and Julien Narboux. Proof assistants for undergraduate mathematics and computer science education: Elements of a priori analysis. *INDRUM 2022: Fourth conference of the International Network for Didactic Research in University Mathematics*, October 2022.

[10] Christoph Benzmuller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, Wolf Schaarschmidt, Jorg Siekmann, and Volker Sorge. Ωmega: Towards a Mathematical Assistant. In *Proceedings of the 14th Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 252–255, Townsville, Australia, 1997. W. McCune.

[11] Christoph Benzmüller, Armin Fiedler, Malte Gabsdil, Helmut Horacek, Ivana Kruijff-Korbayová, Manfred Pinkal, Jörg H. Siekmann, Dimitra Tsovaltzi, Bao Quoc Vo, and Magdalena Wolska. Tutorial dialogs on mathematical proofs. In *International Joint Conference on Artificial Intelligence*, 2003.

[12] Philippe Bernat. CHYPRE: Un logiciel d'aide au raisonnement. Technical Report 10, IREM, 1993.

[13] Yves Bertot. The CtCoq System: Design and Architecture. *Formal Aspects of Computing*, 11(3):225–243, September 1999.

[14] Yves Bertot. Semantics for programming languages with Coq encodings. March 2015.

[15] Yves Bertot, Frédérique Guilhot, and Loïc Pottier. Visualizing Geometrical Statements with GeoView. *Electronic Notes in Theoretical Computer Science*, 103:49–65, November 2004.

[16] Yves Bertot, Gilles Kahn, and Laurent Théry. Proof by pointing. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, pages 141–160, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[17] Jérémy Blanc, André Hirschowitz, Loïc Pottier, and Giacometti. Teaching proofs for freshmen with Coqweb. *Proceedings PATE07*, pages 93–107, 2007.

[18] Sebastian Böhne and Christoph Kreitz. Learning how to Prove: From the Coq Proof Assistant to Textbook Style. *Electronic Proceedings in Theoretical Computer Science*, 267:1–18, March 2018.

[19] Richard Bornat and Bernard Sufrin. Jape's quiet interface. 1996.

[20] Joachim Breitner. Visual Theorem Proving with the Incredible Proof Machine. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving*, volume 9807, pages 123–139. Springer International Publishing, Karlsruhe Institute of Technology, 2016.

[21] John Seely Brown and Richard R. Burton. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills*. *Cognitive Science*, 2(2):155–192, April 1978.

[22] John Seely Brown, Richard R. Burton, and Kathy M. Larkin. Representing and using procedural bugs for educational purposes. In *Proceedings of the 1977 Annual Conference on - ACM '77*, pages 247–255, Not Known, 1977. ACM Press.

[23] John Seely Brown and Kurt VanLehn. Repair Theory: A Generative Theory of Bugs in Procedural Skills. *Cognitive Science*, 4(4):379–426, October 1980.

[24] N.G. Bruijn, de. *Automath : A Language for Mathematics*. EUT Report. WSK, Dept. of Mathematics and Computing Science. Technische Hogeschool Eindhoven, 1968.

[25] Bruno Buchberger, Adrian Craciun, Tudor Jebelean, Laura Kovács, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *J. Appl. Log.*, 4(4):470–504, 2006.

[26] Rod Burstall. ProveEasy: Helping people learn to do proofs. *Electr. Notes Theor. Comput. Sci.*, 31:16–32, December 2000.

[27] A Cant and M A Ozols. *Xlsabelle: A Graphical User Interface to the Isabelle Theorem Prover*. 1995.

[28] Merlin Carl. Using Automated Theorem Provers for Mistake Diagnosis in the Didactics of Mathematics, February 2020.

[29] Merlin Carl, Hinrich Lorenzen, and Michael Schmitz. Natural Language Proof Checking in Introduction to Proof Classes – First Experiences with Diproche. *Electronic Proceedings in Theoretical Computer Science*, 354:59–70, February 2022.

[30] Nathan Carter and Kenneth Monks. From Formal to Expository: Using the Proof-Checking Word Processor Lurch to Teach Proof Writing. January 2016.

[31] Nathan C Carter and Kenneth G Monks. Lurch: A Word Processor that Can Grade Students' Proofs. July 2013.

[32] Pierre Castéran and Yves Bertot. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[33] Jacek Chrzaszcz and Jakub Sakowicz. Papuq: A Coq assistant ? In *PATE'07*, 2007.

[34] Pedro Cobo, Josep M. Fortuny, Eloi Puertas, and Philippe R. Richard. AgentGeom: A multiagent system for pedagogical support in geometric proof problems. *International Journal of Computers for Mathematical Learning*, 12(1):57–79, April 2007.

[35] Claudio Sacerdoti Coen and Enrico Zoli. A Note on Formalising Undefined Terms in Real Analysis. In *PATE'07 International Workshop on Proof Assistants and Types in Education - June 25th, 2007 - The 2007 Federated Conference on Rewriting, Deduction and Programming*, 2007.

[36] Robert L. Constable. Naïve Computational Type Theory. In Helmut Schwichtenberg and Ralf Steinbrüggen, editors, *Proof and System-Reliability*, pages 213–259. Springer Netherlands, Dordrecht, 2002.

[37] Thierry Coquand. *Une Théorie Des Constructions*. PhD thesis, Université Paris 7, January 1985.

[38] Marcos Cramer. *Proof-Checking Mathematical Texts in Controlled Natural Language*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, October 2013.

[39] Marcello D'Agostino and Ulrich Endriss. WinKE: A Proof Assistant for Teaching Logic. 1998.

[40] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[41] David Delahaye, Mathieu Jaume, and Virgile Prevosto. Coq, un outil pour l'enseignement. Une expérience avec les étudiants du DESS Développement de logiciels srs. *Technique et Science Informatiques*, 24:1139–1160, November 2005.

[42] Robert G Farrell, John R Anderson, and Brian J Reiser. 1984 - An Interactive Computer-Based Tutor for LISP. 1984.

[43] Ingo Feinerer and Gernot Salzer. A comparison of tools for teaching formal software verification. *Formal Aspects of Computing*, 21(3):293–301, May 2009.

[44] Asta Halkjær From, Jørgen Villadsen, and Patrick Blackburn. Isabelle/HOL as a Meta-Language for Teaching Logic. *Electronic Proceedings in Theoretical Computer Science*, 328:18–34, October 2020.

[45] Emilio Jesús Gallego Arias, Benoît Pin, and Pierre Jouvelot. jsCoq: Towards Hybrid Theorem Proving Interfaces. *Electronic Proceedings in Theoretical Computer Science*, 239:15–27, January 2017.

[46] M. Ganesalingam and W. T. Gowers. A fully automatic problem solver with human-style output, September 2013.

[47] H. Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 34(1):3–25, February 2009.

[48] Georges Gonthier. Formal Proof—The Four- Color Theorem. 55(11), 2008.

[49] Mike Gordon. From LCF to HOL: A Short History. In Gordon Plotkin, Colin P. Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*, pages 169–186. The MIT Press, May 2000.

[50] Jacques Gressier. Géométrix. http://revue.sesamath.net/spip.php?article42, 2006.

[51] David Gries and Fred B. Schneider. A Logical Approach to Discrete Math. In *Texts and Monographs in Computer Science*, 1993.

[52] Thomas Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, November 2005.

[53] Thomas Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture, January 2015.

[54] Xiaorong Huang and Armin Fiedler. Presenting Machine-Found Proofs. 1104, January 2000.

[55] Wolfram Kahl. The Teaching Tool CalcCheck A Proof-Checker for Gries and Schneider's "Logical Approach to Discrete Math". In *Certified Programs and Proofs*, 2011.

[56] Nadine Karsten, Frederik Krogsdal Jacobsen, Kim Jana Eiken, Uwe Nestmann, and Jørgen Villadsen. ProofBuddy: A Proof Assistant for Learning and Monitoring. *Electronic Proceedings in Theoretical Computer Science*, 382:1–21, August 2023.

[57] Evgenia Karunus and Anton Kovsharov. Lean Paperproof. https://github.com/Paper-Proof/paperproof, 2024.

[58] Marie Kerjean, Frédéric Leroux, Patrick Massot, Michaela Mayero, Zoé Mesnil, Simon Modeste, Julien Narboux, and Pierre Rousselin. Utilisation des assistants de preuves pour l'enseignement en L1 - Retours d'expériences. *Gazette SMF*, August 2022.

[59] Peter Koepke. Textbook Mathematics in the Naproche-SAD System. In *Joint Proceedings of the FMM and LML Workshops*, 2019.

[60] Agnieszka Kozubek and Paweł Urzyczyn. In the Search of a Naive Type Theory. In Marino Miculan, Ivan Scagnetto, and Furio Honsell, editors, *Types for Proofs and Programs*, volume 4941 of *PATE'07*, pages 110–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[61] Daniel Kühlwein, Marcos Cramer, Peter Koepke, and Bernhard Schröder. The Naproche System.

2009.

[62] Tobias Kuhn. A survey and classification of controlled natural languages. *Comput. Linguist.*, 40(1):121–170, March 2014.

[63] Graham Leach-Krouse. Carnap: An Open Framework for Formal Reasoning in the Browser. *Electronic Proceedings in Theoretical Computer Science*, 267:70–88, March 2018.

[64] Nicolas Leduc. *QED-Tutrix : Système Tutoriel Intelligent Pour l'accompagnement Des Élèves En Situation de Résolution de Problèmes de Démonstration En Géométrie Plane*. PhD thesis, École Polytechnique de Montréal, December 2016.

[65] Sorin Lerner, Stephen R. Foster, and William G. Griswold. Polymorphic Blocks: Formalism-Inspired UI for Structured Connectors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3063–3072, Seoul Republic of Korea, April 2015. ACM.

[66] Xavier Leroy. Sémantiques mécanisées, quand la machine raisonne sur ses langages. Cours et Séminaires, 2020.

[67] Xavier Leroy, Sandrine Blazy, Daniel Kästner, Bernhard Schommer, Markus Pister, and Christian Ferdinand. CompCert - A Formally Verified Optimizing Compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress*, Toulouse, France, January 2016. SEE.

[68] Adrian Lon, Peter Koepke, Anton Lorenzen, Adrian Marti, Marcel Schütz, and Makarius Wenzel. The Isabelle/Naproche Natural Language Proof Assistant. In *CADE 28*, pages 614–624, July 2021.

[69] Helen Lowe. The Use of Theorem Provers in the Teaching and Practice of Formal Methods. In *Proceedings of the 1st Irish Workshop on Formal Methods*, 1997.

[70] Vanda Luengo. Cabri-Euclide : un micromonde de preuve intégrant la réfutation Principes didactiques et informatiques Réalisation. 1998.

[71] Christoph Lüth and Martin Ring. A Web Interface for Isabelle: The Next Generation. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Intelligent Computer Mathematics*, volume 7961, pages 326–329. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[72] Alexander Lyaletski and Konstantin Verchinine. Evidence Algorithm and System for Automated Deduction: A Retrospective View, May 2010.

[73] Heather Macbeth. The Mechanics of Proof. https://hrmacbeth.github.io/math2001/index.html, 2023.

[74] Lena Magnusson and Bengt Nordström. The Alf proof editor and its proof engine. In Gerhard Goos, Juris Hartmanis, Henk Barendregt, and Tobias Nipkow, editors, *Types for Proofs and Programs*, volume 806, pages 213–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[75] Patrick Massot. Verbose Lean 4. https://github.com/PatrickMassot/verbose-lean4, 2021.

[76] Roman Matuszewski and Piotr Rudnicki. MIZAR: The first 30 years. *Mechanized Mathematics and Its Applications*, 4:3–24, March 2005.

[77] Julien Narboux. A Graphical User Interface for Formal Proofs in Geometry. *Journal of Automated Reasoning*, 39(2):161–180, August 2007.

[78] M. Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and M. Ikram Ullah Lali. A Survey on Theorem Provers in Formal Methods, December 2019.

[79] Wojciech Nawrocki, Edward W. Ayers, and Gabriel Ebner. An Extensible User Interface for Lean 4. pages 20 pages, 1531949 bytes, 2023.

[80] Wojciech Nawrocki, Jon Eugster, and Bentkamp. The Lean 4 web editor. https://live.lean-lang.org/, 2022.

[81] Tobias Nipkow. Teaching Semantics with a Proof Assistant: No More LSD Trip Proofs. In

Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *VMCAI'12: Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 24–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[82] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[83] Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 G\"{o}teborg, Sweden, September 2007.

[84] Andrei Paskevich. *Méthodes de formalisation des connaissances et des raisonnements mathématiques : aspects appliqués et théoriques*. Informatique, UNIVERSITÉ PARIS XII, Paris, December 2007.

[85] Benjamin Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. *ACM SIGPLAN Notices*, 44:121–122, August 2009.

[86] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2017.

[87] Jim Portegies, Jelle Wemmenhove, Thijs Beurskens, Sean McCarren, Jan Moraal, and David Tuin. Waterproof: Educational software for learning how to write mathematical proofs, 2022.

[88] Dominique Py. Geometry problem solving with Mentoniezh. *Computers & Education*, 20(1):141–146, February 1993.

[89] C. Raffalli and R. David. Computer Assisted Teaching in Mathematics. *Workshop on 35 years of Automath*, 2002.

[90] Krzysztof Retel and Anna Zalewska. Mizar as a Tool for Teaching Mathematics. *Mechanized Mathematics and Its Applications 4 (1)*, pages 35–42, January 2005.

[91] P. R. Richard and J. M. Fortuny. Amélioration des compétences argumentatives à l¿aide d¿un système tutoriel en classe de mathématique au secondaire. *Annales de didactique et de sciences cognitives - Revue internationale de didactique des mathématiques*, 12:83–216, January 2007.

[92] Valentin Robert. Introducing PeaCoq. https://ptival.github.io/2015/06/04/introducing-peacoq/, 2014.

[93] Benoit Rognier and Guillaume Duhamel. Présentation de la plateforme edukera. In Julien Signoles, editor, *Vingt-Septièmes Journées Francophones Des Langages Applicatifs (JFLA 2016)*, Saint-Malo, France, January 2016.

[94] Richard Scheines and Wilfried Sieg. Computer Environments for Proof Construction. *Interactive Learning Environments*, 4(2):159–169, January 1994.

[95] Marvin Schiller, Dominik Dietrich, and Christoph Benzmüller. Proof step analysis for proof tutoring - a learning approach to granularity. *Teaching Mathematics and Computer Science*, 6(2):325–343, 2008.

[96] W. Sieg. The AProS Project: Strategic Thinking & Computational Logic. *Logic Journal of IGPL*, 15(4):359–368, July 2007.

[97] Richard Sommer and Gregory Nuckols. A Proof Environment for Teaching Mathematics. *Journal of Automated Reasoning*, 32:227–258, 2004.

[98] Terence Tao. QED - an interactive textbook. https://teorth.github.io/QED/, 2018.

[99] Patrick Terrematte and João Marcos. TryLogic tutorial: An approach to learning Logic by proving and refuting. July 2015.

[100] Michele Tessier-Baillargeon, Philippe R. Richard, Nicolas Leduc, and Michel Gagnon. Étude comparative de systèmes tutoriels pour l'exercice de la démonstration en géométrie. *Annales de didactique et de sciences cognitives*, (22):93–117, July 2017.

[101] The Cocorico Coq wiki. Coq in the classroom. http://coq.inria.fr/cocorico/

CoqInTheClassroom, 2021.

[102] The Cocorico Coq wiki. Universities teaching Coq. https://github.com/coq/coq/wiki/Universities-teaching-Coq, 2024.

[103] The Programming Logic Group in Göteborg. Implementation of Proof Editors. https://www.cse.chalmers.se/research/group/logic/implementation.mhtml.

[104] Athina Thoma and Paola Iannone. Learning about Proof with the Theorem Prover LEAN: The Abundant Numbers Task. *International Journal of Research in Undergraduate Mathematics Education*, 8, April 2022.

[105] Frédéric Tran Minh. Use of two proof assistants in an introduction to proof course : An experiment. In *ThEdu'23*, Roma, 2023.

[106] Andrzej Trybulec. On a System of Computer-Aided Instruction of Logic. *Bulletin of the Section of Logic*, 12(4):214–218, 1983.

[107] Jørgen Villadsen, Asta Halkjær From, and Anders Schlichtkrull. Natural deduction and the Isabelle proof assistant. *arXiv preprint arXiv:1803.01473*, 2018.

[108] Jørgen Villadsen and Frederik Krogsdal Jacobsen. Using Isabelle in Two Courses on Logic and Automated Reasoning. In João F. Ferreira, Alexandra Mendes, and Claudio Menghi, editors, *Formal Methods Teaching*, pages 117–132, Cham, 2021. Springer International Publishing.

[109] Marc Wagner, Serge Autexier, and Christoph Benzmüller. PlatΩ: A Mediator between Text-Editors and Proof Assistance Systems. *Electronic Notes in Theoretical Computer Science*, 174(2):87–107, May 2007.

[110] N. Webber, C., L. Bergia, S. Pesty, and Nicolas Balacheff. The Baghera project: A multi-agent architecture for human learning. *Proceedings of the Workshop Multi-Agent Architectures for Distributed Learning Environments*, pages 12–17, 2001.

[111] Makarius Wenzel. Isabelle/Isar — a generic framework for human-readable proof documents. January 2007.

[112] Cezary Kaliszyk Freek Wiedijk. Teaching logic using a state-of-the-art proof assistant. In *PATE'07*, 2007.

[113] Freek Wiedijk. Comparing Mathematical Provers. In Andrea Asperti, Bruno Buchberger, and James Harold Davenport, editors, *Mathematical Knowledge Management*, volume 2594, pages 188–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[114] Freek Wiedijk. A Synthesis of the Procedural and Declarative Styles of Interactive Theorem Proving. *Log. Methods Comput. Sci.*, 8(1), 2012.

[115] Wolfgang Windsteiger. Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System. *Electronic Proceedings in Theoretical Computer Science*, 118:72–82, July 2013.