A Lean-based Language for Teaching Proof in High School

Frédéric Tran Minh, Laure Gonnord, Julien Narboux

A Lean-based Language for Teaching Proof in High School

Frédéric Tran Minh[0009−0003−9221−5585]1, Laure Gonnord[0000−0002−8013−1611]1, and Julien Narboux[0000−0003−3527−7184]2

[1] Grenoble INP - LCIS, UGA
{frederic.tran-minh,laure.gonnord}@lcis.grenoble-inp.fr
[2] IRIF - CNRS - Université Paris Cité
narboux@irif.fr

Abstract. Now pervasive in many mathematics and computer science research domains, proof assistants have recently gained importance in education, mostly during the college years. In this article, we propose a new learning environment as a layer above the Lean proof assistant, specifically targeting high-school level proofs. Drawing inspiration from coherent logic and adaptations of proof assistants for teaching, such as Lean-Verbose and Coq Waterproof, along with our own experience using proof assistants in higher education, we designed Yalep, a declarative controlled natural language, with a minimal number of syntactic constructions, which favors forward-chaining of facts. Yalep provides convenience for type theory hiding, and functions defined on type subsets. This paper presents the design choices and implementation of these features.

1 Introduction

Proof assistants are software that help users construct formal proofs and verify their correctness. Proof assistants are used more and more for teaching proof and proving[1] in higher education, but their use in high school is still limited. In [37], we presented a survey of the use of proof assistants for teaching, and we showed that most of the work is done in the context of higher education. Very few works are done in the context of high school. TPE [32], EasyProve [25], and Edukera [28] have been partly designed for high school. As these tools provide mainly a point-and-click user interface, the design of the input language was not crucial. Sylvie Boldo *et al.* also designed a set of exercises for high-school students about divisibility using the Coq/Rocq proof assistant and tested it with three high-school students [5].

[1] Mathematics didactics distinguishes between proof and proving [34]. Proving is the process, proof is the concept. Yalep aims to support the teaching of both: what is a proof (which, as opposed to a simple convincing argument, can be checked mechanically), and also how to produce a proof (by training students). For the sake of concision, in the paper we wrote “teaching proof” to mean “teaching proof and proving”.

In this paper, we present the prototype of a tool called Yalep[2] for teaching proof in high school using a proof assistant. Our work is inspired by Lean-Verbose [24], Coq Waterproof [43] and Isabelle Proof Buddy [18], which are layers above a proof assistant for use in higher education. Other ingredients for the design include our personal empirical experience with proof assistants in research and/or higher education, as well as coherent logic. The main goal of Yalep is to provide an environment for learning proof using a minimal language that is close to the language used in high school.

The context of the design of Yalep is the French educational system, in which all coauthors of this paper are involved. We chose to address high-school students because we all encountered difficulties in our undergraduate courses when it came to writing proofs.[3] Addressing high-school proofs requires us to develop a system that is easy to use, targeting students and teachers without a background in computer science or logic. Despite the fact that a few design choices may be specific to the French context, we strongly believe that many ideas presented in the paper could be used in other countries where proof is part of the curriculum.

In Section 2, we draw on the particularities of the high-school teaching context to set out the requirements for our tool. In Section 3, we describe the main characteristics and the design of our language. We then dive into more implementation details. Section 4 describes how Yalep renders the type-theoretical foundations of the proof assistant as invisible as possible. Section 5 focuses on how it represents functions whose domain and co-domain are type subsets. Section 6 presents a didactic progression to validate our tool.

2 Requirements Geared Towards High School

Our ambition for Yalep is to provide an ecosystem that helps high-school students understand the notion of mathematical proofs. This section provides a specification for Yalep, along with the limitations of existing approaches.

Language design Usually, and notably in the French educational system, high-school students are not familiar with logic and formal proofs, and very few of them are familiar with programming languages. Wemmenhove [42] thus points out that the steep learning curve of the syntax of an educative proving tool is one major difficulty for students (and also math teachers). Conversely, existing approaches using point-and-click user interfaces (such as Edukera), despite their "intuitive" interface, have shown their limits in terms of efficiency in the long term: some skills required for paper-and-pencil proofs may in fact be hidden in the graphical interface [3,19]. We aim to design a language "natural

[2] Yet Another Learning Environment for Proof.

[3] Moreover, Villani (field medallist) and Torossian wrote in 2018 a report about teaching mathematics (https://www.education.gouv.fr/21-mesures-pour-l-enseignement-des-mathematiques-3242) which advocates to give more room to proof in high school.

enough" so it can be read by high-school students without prior training or executing the proof script in a proof assistant, and as uncluttered as possible without generating ambiguity, so students can use it while focusing on proof rather than syntax.

We thus propose the following language requirements:

RL1 A declarative style for the proofs resembling what we could expect a high-school student to write.

RL2 A restricted formal textual language: very few keywords, no synonyms (different keywords referring to the same proof action, like obtain/fix for eliminating existential).

RL3 No overloaded keywords (a same keyword referring to separate proof actions depending on the context, like "let $x := 0$" / "let $x \in \mathbb{R}$")

RL4 Syntactic sugar to hide the mathematical notations that are out of the scope of high-school curriculum: for instance, avoid the $\forall$ symbol.

Expressivity and automation With a focus on high-school curriculum, the topics covered by our proposal have to include at least: elementary number theory, elementary numeric and literal computations, numeric functions (composition, variations), numeric sequences, limits, and derivatives. In addition, our solution should enable some (restricted) kind of automation, to transparently enable the computer check to "skip" parts of the proofs that would not have been justified by students.

This variety of topics gives the following requirements:

RA1 The technological choices should make it possible to consider extensions to many secondary education subjects without too much development;

RA2 All proof steps for which the teacher considers that no justification is needed should be accepted silently;

RA3 All proof steps for which the teacher expects a proof should be rejected when submitted without a sub-proof;

RA4 The level of justification required should be configurable by the teacher.

Assistance The tool should assist the student in the process of writing a proof, more specifically, the learning environment should:

RH1 display the current proof state, making explicit the known statements and the current goal at a given point of a proof;

RH2 provide constant feedback to the student about the correctness of the proof steps;

RH3 provide relevant explanations in case of errors;

RH4 help the student with the formal "rhetorical" part of the proof, providing help with proof by cases/induction;

RH5 provide help for the "creative" part of the proof.

2.1 The choice of Lean

Proof assistants, unlike (fully automatic) provers, offer facilities to adapt the automation level to topics or users' skills. Their interaction features also enable a constant display of the proof state (requirement RH1).

Among the available proof assistants, we chose Lean, which provides the following relevant features:

- A particularly flexible parser and delaborator (pretty printer), which is convenient to implement a controlled natural language (referred to as CNL in the rest of the paper) [24];
- A large and unified mathematical library for having solid foundations without reinventing the wheel;
- Lean4Web[4] runs Lean in a web browser, enabling easy usage in class;
- A Lean module enables to interact with JavaScript/React widgets hosted in a Visual Studio extension [26], which we exploit for some graphical display and point-and-click interaction.

The work described in this paper could also be carried out using Rocq. We could quite easily adapt Coq Waterproof to fit our design goals, and use Jscoq for the web interface, as it also allows some interactivity [13]. Jscoq runs the proof assistant on the client side, which allows dealing with many students with a modest server.

3 Yalep, a Tiny Controlled Language for Forward Reasoning

Our proposition, Yalep, constructed as a layer over Lean, aims at describing a set of statements followed by their proof.[5] In order to give a taste of it, we depict in Figure 1 a simple running example for the current section.

3.1 A "pseudo-natural" declarative language with very few variants

A proof exercise in Yalep is composed of a statement (under the form of Assumptions followed by a Conclusion), and its Proof. The syntax for proofs is inspired by coherent logic [33], a subset of first-order logic that is as expressive as first-order logic and has good meta-theoretical properties.

A proof (that can be hierarchical) is composed of a list of *facts*, delimited using the connector ⬥. It ends with the □ connector. Proofs are mandatorily expressed using *forward reasoning*.

Assumptions and proof statements are not mandatorily labeled. This brings the user scripts closer to traditional pen-and-paper practice, where assumptions

[4] https://github.com/leanprover-community/lean4web

[5] Its grammar, intentionally restricted, is available in BNF format in the project repository hosted at https://gricad-gitlab.univ-grenoble-alpes.fr/yalep/Yalep as well as other examples in English and in French.

[6] In the Yalep language, `integer` qualifies an element of $\mathbb{Z}$.

```
Theorem product_even "5. Proving (for all ...)"
  Conclusion: for all integer n, for all integer b, if n is even then n*b is even
Proof
  let n be an integer
  let b be an integer
  assume n is even
  ◆ there exists an integer k such that n = 2*k
  obtain such k
  ◆ n*b = 2*(k*b)
  ◆ there exists an integer q such that n*b = 2*q
  ◆ n*b is even
□

Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (forall integer n, n is even or n is odd)
    Conclusion: for all integer n, n*(n+1) is even
Proof
    let n be an integer
    ◆ n is even or n is odd
    ◆ if n is even then n*(n+1) is even
      proof
        assume n is even
        ◆ n*(n+1) is even by product_even
      □
    ◆ if n is odd then n*(n+1) is even
      proof
        assume n is odd
        ◆ n+1 is even
        ◆ n*(n+1) is even by product_even
      □
    ◆ n*(n+1) is even
□
```

Fig. 1: A taste of Yalep proofs.[6]

are generally only named when they represent key facts or as abbreviations to avoid copying long statements (Requirement RL1).

To meet the requirements of the French high-school curriculum, statements and facts are expressed in plain English (or French) using verbal formulations that correspond to the usual symbols (e.g., logical symbols $\wedge, \vee, \forall, \exists, \Longrightarrow, \Longleftrightarrow$, and number set notations $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$) to which pupils are not yet supposed to be exposed, depending on their grade level (RL4). Our tool also extends the relative quantification (like $\forall x > 0, \ldots$) already available out of the box in proof assistants, to tactic constructs (such as `let x>0;let n be an integer`, etc.) (RL1).

Predicates can be written in sentential form, such as `n is odd` in Figure 1.

Additionally, to fulfill requirement RL2, and unlike other CNL, we chose to provide, if possible, a unique syntactical construct for each proof action. As an example, our language does not allow writing "let $x \in \mathbb{R}$ such that $x + 2 > 0$", since the same semantics can be achieved using `let x∈ℝ, assume x+2>0`.

In our language, we do not allow imperative commands such as "Unfold the definition of ..." or "Let's rewrite the goal" that are often used in proof assistants. As an example, in Figure 1, the fact `◆ n is even or n is odd` has to be stated instead of saying "apply assumption 1 to n" as possible in Lean Verbose/Coq Waterproof.

In the following sections, we will explain the underlying choices behind this syntax and the automation.

3.2 A language based on forward chaining

Backward reasoning consists of proof steps (or tactics in a proof assistant) modifying the goal or creating new goals, whereas forward chaining corresponds to proof steps acting on the assumptions, or deducing new facts from assumptions in the context.

In usual mathematical practice, as well as in proof assistant scripts, steps of both kinds are freely mixed. Occurrences of backward reasoning are usually explicitly introduced, for instance, with "since ... it suffices to prove that ...". However, since proof beginners tend to confuse necessary and sufficient conditions, some teachers try to stick to forward reasoning, at least as far as the written trace of the proof is concerned. In the proof search phase, however, backward chaining is frequently used orally.

This analysis led us to propose a rather constrained language that "syntactically avoids" backward reasoning in final proofs; whereas it enables "proof holes" to mimic the "bottom to top" blackboard usage, frequently used by teachers. As an example, case disjunction is not triggered by an a priori announcement like "Let's discuss whether n is even or n is odd". Instead, the user has to state (and possibly prove) three facts: `◆ P ⟹ R` ; `◆ Q ⟹ R` and `◆ R`.

Mixing backward and forward Our language discards as many backward tactics as possible. However, two of them must remain to maintain expressivity, namely `let` and `assume`, corresponding respectively to the introduction rules of the connectors $\forall$ and $\Longrightarrow$.[7] Since they introduce a new object or hypothesis into the context, and consequently modify the goal, they have no forward equivalent.[8]

Discussion From the pedagogical perspective, favoring forward chaining has the following advantages:[9]

- It restricts the number of commands (RL2): instead of learning a list of tactics (one per backward chaining command), students only learn the sole forward chaining command, `◆`, that inserts a new fact in the proof context.
- Students have to explicitly announce new facts (RL1): for instance, to prove R by using the assumption P or Q, they should explicitly announce (and prove) `◆ P ⟹ R` and `◆ Q ⟹ R` (like a pen and paper proof). The counterpart here is that the proof assistant does not help/assist the user in the task of discovering new goals.
- Forward facts can be written in any order, provided that they are provable in the context at the moment they are needed. The drawback here is that it may favor "unstructured proofs". In backward automated proofs, only "opened" sub-goals can be proven.

[7] In fact, these two constructions are the same in proof assistants implementing the type-theory paradigm of "propositions as types".

[8] This ability to introduce $\forall$ and $\Rightarrow$ and to create sub-proofs distinguishes our language from flat two-column proofs [16].

[9] As an illustrative comparison, we provide, in the project documentation, two variants of an inductive proof that every natural number is either even or odd.

– Forward tactics do not modify the main goal; instead, they make an accumulation of facts in the current context. Each fact can be justified by a "sub-proof". This is the analogue of sub-goal creation in backward chaining.

3.3 Implicit proof actions

Yalep also provides additional features so to fulfill requirement RA2:
– It implicitly introduces or eliminates some of the logic connectors.
– Connectors ◎ and ◆ allow to structure proofs and trigger parametrized automation.

Implicit versus explicit connectors Table 1 depicts usual logic connectors' usage in Yalep. The indication "(silent)" means that:
– if the connector appears in the goal, the presence of the premises in the context is enough to state a new fact introducing the connector.
– if the connector appears in a hypothesis, one can state a new fact following from the elimination of the connector, without justifying anything.

Table 1: Implicit or explicit introduction or elimination of connectors in Yalep

Connector	Appears in goal	Appears in assumption
`P and Q`	(silent)	(silent)
`P or Q`	(silent)	(silent)
$P \Longrightarrow Q$	`assume P`	(silent)
$\forall x \in E, P(x)$	`let x ∈ E`	(silent)
$\exists x \in E, P(x)$	(silent)	`obtain such x`
$P \Longleftrightarrow Q$	(silent)	(silent)

As on paper proofs, dealing with conjunction is completely implicit, and introduction of existential is implicit ($\exists x \in E, P(x)$ needs no justification as long as the context contains $a \in E$ and a proof of $P(a)$). For disjunction, usually, on paper, the introduction rule is implicit, but the elimination of disjunction (reasoning by cases) often is explicit. We decided to make both implicit, which contributes to reducing the language.[10]

Finally, similarly to coherent logic [33], we allow the implicit combination of universal quantifier and implication. For example, if a fact of the form $\forall \overline{x}, H_1(\overline{x}) \wedge \ldots \wedge H_n(\overline{x}) \Rightarrow Q(\overline{x})$ is known and facts $H_1(\overline{a}), \ldots, H_n(\overline{a})$ are in the context, one can conclude $Q(\overline{a})$ without any explicit justification.

The introductions of the universal quantifier and implication need to remain explicit, as they are usually explicit in paper proofs, and rendering the introduction of the universal quantifier implicit would require automatic naming.

[10] More precisely, case disjunction is implicit but not automatic; if `P or Q` lies in the context, it is up to the user to decide whether to create or not and prove new facts `◆ P ⟹ R` and `◆ Q ⟹ R` so that subsequent fact `◆ R` can be asserted without further explicit justification.

One could decide to introduce implicitly x when the goal is $\forall x, P(x)$ and x is available. But that would introduce confusion between the name of the bound variable and the name of the arbitrary variable, and the system would assist the student too much.

The rule for eliminating the existential quantifier needs to be explicit, because it introduces a new object. Precisely, it is often a pedagogical objective to expect from the student a proper naming of the newly introduced free variable, not necessarily named after the bounded variable, especially if a same existential definition is used twice and they should choose two different names (as in "assume that m and n are even. Obtain k and k' such that. . . ").

Solving facts As we give priority to forward chaining, the syntactic construction to create a new fact (⬥) has a central role. The user can then solve this subgoal in three different ways:

- either opening a whole nested sub-proof by the construct (`proof` ... □).
- either by using one of the two provided types of explicit short justifications, those that relate to a statement or a fact (we use the keyword `since` as in `since x > 0`) and those that relate to a statement or a fact name (we use the keyword `by` as in `by triangle inequality`).
- either silently, with no justification at all, by relying on the automation.

We provide a syntactic construction (◎) that is similar to Lean (`calc`) or Lean Verbose (`Calc`) or Waterproof (`&`) tactics. It enables chaining several relational propositions by transitivity.[11]

Unlike Lean Verbose (that uses the current computation to solve the current goal), our (◎) keeps the pure forward chaining style, and introduces a new fact independently of the current goal. It even feeds the context with intermediate calculations. For example:

```
◎ x ⩽ y
_   < z
```

will add three new facts to the context: $x \leqslant y$, $y < z$ and $x < z$.

Discussion Let us firstly remark that all the (silent) in Table 1 demonstrate the minimal need for additional syntax (Requirement RL2).

Making elimination implicit also enables a step towards coherent logic [33], in which proofs are of coarser granularity than those in, for instance, natural deduction.[12] As a simple example, $A \wedge B \wedge C$ is automatically split onto $\{A, B, C\}$ as soon as it enters the context, which saves the double elimination of $\wedge$.

Some teachers may find the implicit elimination of connectors $\forall$, $\Longrightarrow$, $\Longleftrightarrow$, or $\vee$ too permissive (RA3). This advocates for a (yet to be developed) "à la carte" justification, according to some teacher-based parametrization (RA4), because it depends on the contextual "base of knowledge".

[11] The term is somewhat mathematically abusive since we allow combining different relations, e.g. $x \leqslant y \wedge y < z \Longrightarrow x < z$.

[12] This approach tending to enlarge proof granularity [29] is similar to assertion level proofs [17], implemented in Tutch [1] or Omega [31,2].

3.4 User interface, interaction and automation

Proof automation In this section, we consider the choices toward the objective of requirement RA2. The design choice is to rely on Lean powerful tactics, instead of relying on a general-purpose prover (like Lurch [8], Naproche). We think that this choice will facilitate the fulfillment of requirement RA3, and produce relevant feedback.

Our tool provides different kinds of automation that are triggered silently, which include:

- Elimination of connectors such as `and`, `or`, $\Longrightarrow$ (already discussed in the previous section).
- Termination of unfinished proofs (from the proof assistant point of view, but that should "look finished" for the user). It is generally triggered by the end-of-proof or end-of-sub-proof token □ , but can be called by other tactics. The implementation of the tactic itself is an ordered collection of trial and error attempts, using connector introduction, basic properties of operators (commutativity, associativity) or "trivial" lemmas (such as $x > 0 \Rightarrow x \geqslant 0$), and the Lean tactic for solving tautologies.[13]
- Justification of numerical facts: whenever a stated fact is left unjustified, it triggers native Lean tactics performing numerical computations and applying lemmas from the math Lean library targeting the high-school topics.
- Proof of legality of a function application: it derives facts from assumptions and membership class inference. (see Section 5)

Some other automation tactics are triggered by some idioms of our language. For instance, on Figure 1, `n is odd` is elaborated into a library definition, and `by product_even` has the effect of trying to prove the statement using the provided predicate. Other similar behaviors are linked to `since` or `by induction`.

User interface and facilitation The user already benefits from the native real-time display of the "proof state" (current assumptions and goal) provided by the Lean Proof assistant, customized thanks to the configurability of the pretty printer.

Considering that in the research phase of a proof, a student is frequently led to conduct backward reasoning, either orally or in draft form, we propose a point-and-click user-friendly way to trigger "backward-style" actions, which insert text into the proof script in our forward style.

The integration of interface buttons via JavaScript widgets hosted in a Visual Code extension communicating with the Lean Server is made possible by the Lean ProofWidgets layer [26]. For instance:

- when selecting the goal, the system displays a button to insert an appropriate "tactic" introducing the main connector.
- when selecting an assumption in the proof state, a button is displayed to insert a tactic eliminating the main connector of the assumption.

[13] This implies that Yalep is not suitable for teaching the fine-grained rules of logic. Many other tools are available for this purpose [14,12,27,38,36,23,20,35,22,40,39,41], see [37] for further details.

In our example, starting with the incomplete proof on the left of Figure 2, clicking on `n is even or n is odd` will expand the hole `?` with the proof displayed on the right-hand side.

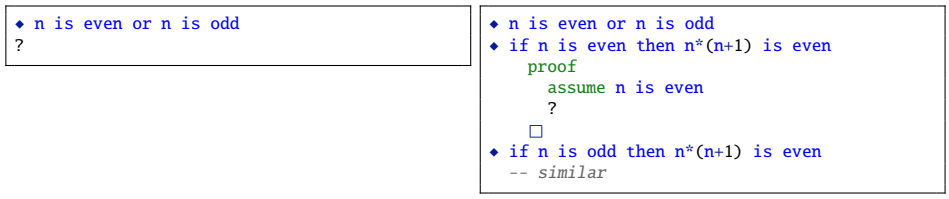


Fig. 2: Backward reasoning using proof widgets.

4 Hiding Types from High-school Students

In type theory — which constitutes the basis of proof assistants such as Rocq or Lean — each object x comes with a type T, and the judgement $x : T$ is not a proposition; in contrast, usual educational presentation of mathematics relies on set theory, where all objects are un-typed sets, and when x and T are two sets, the proposition $x \in T$ can be proved or disproved. In contrast, Yalep allows to state and prove $\sqrt{2} \notin \mathbb{Q}$.[14] Without any additional interface, students would have to grasp the subtle nuance between $x : \mathbb{R}$ and $x \in \mathbb{R}^+$; the tight seal between types would lead to weird considerations when mixing different types of numbers, e.g., $((2 : \mathbb{N}) > 0)$ could be rejected when $((2 : \mathbb{R}) > 0)$ is expected ; functions defined between types do not model accurately usual high-school functions defined between sets as they would have to be "type-partial" (see next section).

Yalep avoids any type manipulation. On the cosmetic side, we exploit the Lean ability to implicitly infer any type ascription, either in statements ($\forall x \in E, P(x)$ elaborates as $\forall x : T, x \in E \implies P(x)$), in tactics (`let x ∈ E` will elaborate as `let x:T \ assume x ∈ E`) or in theorem statements (`Theorem test (...) Assumptions: (x ∈ E) (...)` will elaborate as `theorem test (x:T) (h : x ∈ E) (...)`). In each case, the type ascription $x : T$ does not appear explicitly, replaced by a set-based predicate ($x \in E$).

On the theoretical side, one step towards hiding types is to consider several types as subsets of a unique type. The special case of number types is essential. Proof assistants like Lean, Rocq, usually represent sets of numbers ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$) as "hermetic" types, resulting from their construction. T. Portet and Y. Bertot presented a formalization of numbers where the only number type to be considered is the `Real` type [4]. In their proposition, $\mathbb{N}$ is considered as an inductive predicate over `Real`, which allows to keep native inductive proofs and to recover recursive sequence definitions by means of an additional layer. Since the stability of operations over $\mathbb{N}$ becomes propositional ($(x + y \in \mathbb{N})$ is a statement

[14] Two different proofs of this statement are available in the project repository.

to be proved, whereas the judgement $(x + y : \text{Nat})$ was definitional), automatic proof of such statements is triggered by a type class inference mechanism.

Yalep implements some of the above ideas, taking into account the requirements for high-school proofs. We keep the idea of a unique number type, of which $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ are subsets, and the solution of using class inference to facilitate automation triggering. However, we do not define $\mathbb{N}$ inductively. Rather, for any types T and E, provided there exists an injective mapping $c : E \to T$, we define the set of objects of type E with ${}^sE := \{x : T \mid \exists y : E, x = c(y)\}$. For numbers, we denote our reference type by `Number` (presently an alias of `Real` but it may evolve to the type `Complex` in the future).

This simpler representation has the advantage of applying directly and in the same manner to $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$, with the adaptation of the library theorems specific to number type being reduced to a matter of coercions. However, inductive proofs and recursive functions are not native in Yalep and require intermediate lemmas. In addition, we are not aiming at preserving computability, whereas it is an objective of Portet and Bertot's work.

5 Partial Functions in Yalep

In proof assistants based on type theory (Lean, Rocq, ...), functions are total over a type (let's say "type-total"): they must be defined at any point of the domain type. In mathematics education, functions are total over a set (let's say "set-total"). A set being a predicate on a type, these functions would have to be represented in type theory as "type-partial".

Various approaches have been proposed to handle this situation.

Mathlib for Lean or Math Components for Rocq give a default value (e.g., 0) to functions where they are not defined, and subject associated theorems to assumptions that guarantee that function arguments belong to their domain. For example, in Lean's Mathlib, the inverse of $(0 : \mathbb{Q})$ is defined to be 0, but the predicate $a \cdot a^{-1} = 1$ will only hold if $a \neq 0$. Other ways of dealing with this issue include attaching a proof obligation to each relation involving terms that may be undefined [32] (implemented in TPE), or simulating a logic of partial terms with partial setoids and formalizing "directed quasi equalities", then automating their proofs [9].[15] In this last paper, Sacerdoti-Coen and Zoli specify that they do not consider the Rocq type theoretic approach (also possible in Lean) to encode the function domain into the type because it makes visible that a proof term is an argument of the function. We propose exploring this approach anyway, but hiding the proof term in suitable syntactic sugar with automation.

Considering that students are usually supposed to verify that a term is defined *before* writing it, and that set-theoretic presentation of a function is a triple $f := (D, F, G)$ encoding the domain D, the co-domain F, and the graph $G \subseteq D \times F$, we aim at obtaining a Lean formalization of a type-partial function

[15] A synthetic presentation can also be found in C. Paulin's lecture https://www.lri.fr/~paulin/LASER/coq-slides4.pdf

$f \in F^D$, that, associated with suitable automation and syntactic sugar, satisfies the following conditions:

PF1 The internal representation of f contains its domain and co-domain, and should expose them whenever it is required;
PF2 To write the term $f(x)$ the context must contain a proof of $x \in D$, should it be proven silently or by the user;
PF3 The internal representation of f contains a proof of $\forall x \in D, f(x) \in F$;
PF4 Usual notations for function application and definition are preserved (RL4).

The rest of the section is devoted to the design of our library for partial functions.

5.1 A first proposition: functions between subtypes associated to sets

Consider a set D and a set F of elements of respective types α and β, and that we want to model a map from D to F. The sub-type associated to `D:Set α`, denoted $\uparrow D$, is the type $\sum_{x:\alpha}(x \in D)$ of dependent pairs $\langle$`x:α,hx: x ∈ D`$\rangle$. As $\uparrow D$ is a type, as well as $\uparrow F$, we can consider the type-total function $f : \uparrow D \to \uparrow F$ which embeds all the required information. This representation satisfies PF1, PF2 and PF3 and also, on the developer side:

(a) eases the definition of a function $f \in F^D$ such that $\forall x \in D, f(x) = \ldots$ (provided a proof that $\forall x \in D, f(x) \in F$). Example:
`def f:↑[2;+∞[→↑[3;+∞[:=fun ⟨x,(hx:x⩾2)⟩=>⟨x^2,((by nlinarith):(x^2 ⩾ 3))⟩` creates the square function f from $[2;+\infty[$ to $[3;+\infty[$;
(b) eases the definition of a function application expression $f(x)$, provided a proof that $x \in D$ is given.
Example: `(f ⟨3,(by norm_num:(2:Real)⩽3)⟩).val` would mean $f(3)$;
(c) allows $f(x)$ to be definitionaly equal to its expression (e.g. x^2) (their equality does not require any proof). Example: Lean would automatically unify the previous expression with 3^2.

Let's now examine (a) and (b) from the student side: the examples above show that the underlying formalization has to be encapsulated to satisfy PF4. Regarding (b): each time the user wants to apply f to some $x : \alpha$, she has to express `f ⟨x,hx: x ∈ D⟩`, that is, prove that $x \in D$, which is tedious in practice (even if it has been proved once, it must be repeated each time); additionally the notation is heavy. Thus, we combine this formalization with:

– A tactic that automates the proof of $x \in D$. It calls the Lean tactic `assumption`, so that proving the fact once is enough for all occurrences of $f(x)$. It will also attempt to perform basic numerical calculations, as well as class inferences, to determine membership in a subset of numbers.
– Some syntactic sugar to hide the original notation. We take advantage of the fact that, while proof assistants use the `f x` notation for function application, the usual mathematical notation is $f(x)$. So, parsing `f(x)` will discharge the

proof goal $hx : x \in D$ and then expand to `f ⟨x,hx⟩`. [16] Note that to avoid parsing conflicts, we chose to use here a special Unicode pair of parentheses.

Regarding (a), to define a new function, the user has to prove that $f(x) \in F$ provided that $x \in D$. To avoid this, we provide a tactic automating this task and some syntactic sugar to create a new function while triggering the tactic. In Yalep, our previous function then becomes:

```
let's define the function f from [2;+∞[ to [3;+∞[ that maps x to x^2
```

5.2 A second formalization for partial functions

The previous formalization still has a few drawbacks:

- `f : ↑D → ↑F` is a type ascription. Type manipulation is exposed to the user.
- Given `f : ↑D → ↑F` and `g : ↑D' → ↑F'`, f and g have distinct types, thus homogeneous equality $f = g$ does not typecheck, even if $D = D'$ and $F = F'$ propositionally (it does not make sense to prove or disprove it).[17] For example, suppose that for $f : D \to F$ we denote by f' the derivative of f, whose domain is $D'_f := \{x \in D \mid f \text{ is derivable at } x\}$. Then it is not possible to define the set of functions $\{f : D \to F \mid D'_f = D \text{ and } f = f'\}$.

To tackle this, we bundle the function domain, co-domain, and a type-total function into a structure (`Map α β`). Any function from a set of elements of type α to a set of elements of type β would have type (`Map α β`).

```
structure Map (α β : Type)  [i:Inhabited β]  where
  func : α  → β
  domain : Set α
  codomain : Set β
  prop : ∀ x ∈ domain, func x ∈ codomain
  prop_out : ∀ x:α,  x ∉ domain →  func x = i.default
```

For instance, two functions $g : [1,2] \times \mathbb{Q}^+ \to \mathbb{Z}$ and $h : \mathbb{N} \times \mathbb{Q}^- \to [0;+\infty[$ would have the same type `Map (Number × Number) Number`.

The main drawback of this structure is the absence of automatic proof obligations when applying a function. We restore these proof obligations by defining two mutually reciprocal functions σ and ζ, allowing to switch from one representation to another. Then, applying and defining a function within the second representation reduces to the same activities as within the first one:

- *Applying a function:* σ maps `(u : Map α β)` to $[\sigma(u) : \uparrow(u.domain) \to \uparrow(u.codomain)]$ well-defined by: $\forall \langle x,h\rangle \in \uparrow(u.domain), \sigma(u)\langle x,h\rangle = \langle \text{func}(u)(x), \text{prop}(u)(h)\rangle$.

[16] `f ⟨x,hx⟩` being a pair `⟨y:β,hy: y ∈ F⟩`, the macro $f(x)$ also silently applies the projection `Subtype.val := fun ⟨y,hy⟩ => y` when needed, to avoid students manipulating directly objects of sub-type $\uparrow F$.

[17] It might have been worth considering heterogeneous equality here, but we did not investigate further since it might have led to considering two different equalities, and run into complications about congruence closure [30].

Applying u to x means applying $\sigma(u)$ to $\langle x, h : x \in D\rangle$, thus the operation generates the obligation to prove $x \in D$. The user oriented macro $u(\cdot)$ expands just like $\sigma(u)(\cdot)$.

– *Defining a function:* Conversely, ζ maps a function $f : \uparrow D \to \uparrow F$ to a structure $\zeta(f)$ of type `Map α β`, defined by:
  - $\text{domain}(\zeta(f)) = D$ and $\text{codomain}(\zeta(f)) = F$
  - $\forall x, \text{func}(\zeta(f))(x) = \begin{cases} \text{Subtype.val}\,(f\langle x, h_x\rangle)\ [18] & \text{if } (h_x : x \in D) \\ b \text{ (an arbitrary fixed value of type } \beta) & \text{if } (h_x : x \notin D) \end{cases}$
  - proofs of $\text{prop}(\zeta(f))$ and $\text{prop\_out}(\zeta(f))$ are straightforward.

Like in the first formalization, a specific syntax allows to define a new application $u := \zeta(f)$ where f is built as described in §5.1. But two technical difficulties arise: first, `if x ∈ D...` blocks automatic unification and forces a call to the tactic `apply dif_pos` judiciously placed in the appropriate tactics.[19] Second, the statement `u∈F^D`, which means `u.domain=D and u.codomain=F`, is propositional (rather than definitional): once again, a tactic has to be triggered to use it when needed.

In our solution, a macro providing an interface for function definition also automatically generates, for each defined function, two lemmas that help tactics to unify `u(x)` with its expression, and `u.domain` and `u.codomain` with concrete values to which they may be propositionally equal.

For instance, the last example of section 5.1 generates:[18]

– `f.def:` `∀x∈[2;+∞[, ((sigma f) ⟨x, ...⟩).val = x^2`
– `f.domains:` `f ∈ [3;+∞[^[2;+∞[`

5.3 Consequences

Defining function properties Each definition of a property of a function may require its own automation. Once again, we hide the automation in the syntax.

For example, defining the syntax "f is increasing on D'" requires proving that $D' \subseteq D$. If not, in the term `∀x∈D', ∀y∈D', x≤y⟹f(x)≤f(y)`, the automation associated to `f(x)` will not be able to deduce that $x \in D$ from $x \in D'$.

The need of contextual connectors (and , ⟹) As pointed out by Wiedijk [45], when elaborating $P \implies Q$ or (P and Q), the automation will be stuck if elaborating Q requires a proof of P. For example, to be able to elaborate statements like $\forall x,\ x \neq 0 \longrightarrow f(x) \neq 0$ or $\forall x \in \mathbb{R}, x^2 + 1 \neq 0$ and $f\left(x^2 + 1\right) > 0$, the automation hidden in $f(\cdot)$ should access respectively proofs that $x \neq 0$ and $x^2 + 1 \neq 0$. For that reason, we redefine the implication $P \implies Q$ as $\forall (h : P),\ Q$ and the conjunction (P and Q) to mean $\exists (h : P),\ Q$.

[18] `Subtype.val` is the projection `fun ⟨y,hy⟩ => y`. When `z:↑F`, `z.val` is syntactic sugar for `Subtype.val z`.

[19] The decision procedure generated by `if x ∈ D` would demand that proposition `x ∈ D` is decidable, which is not the case in general, except in classical logic, at the cost of marking definition `noncomputable`.

6 Validation

During the design of Yalep, we also developed a pedagogical progression intended for tenth-graders. Its goal is to introduce the main proof structures gradually, without emphasizing formal logic. Our running example 1 is part of this progression: it illustrates two exercises designed respectively to present the introduction of quantifier ∀ and the elimination of connector `or` (proof by cases). The whole progression culminates in the classical proof of irrationality of $\sqrt{2}$ (as integer parity and this particular proof are parts of the French high-school curriculum). This proof is the classical proof used as an example and will allow the reader to compare with other proof assistants [31,32,44]. To demonstrate the readability and accessibility of Yalep, we also reproduced the so-called informal proof proposed by Barendregt in the first chapter of [44], which is based on the fact that there is no infinite descending sequence of natural numbers.[20]

In June 2025, the tool was tested during a 6-hour workshop at IRIF[21] spread over a week, with tenth-grade students. Students engaged enthusiastically in the proof tasks and the goals of supporting an introduction to proof course at the high-school level, while leaving plenty of autonomy were achieved. We felt that students really grasped the main ideas about proof, but were often hampered by their weaknesses in literal arithmetic. However, the main technical limitations we encountered were the current inability to configure the scope of the automation – some incoherently ordered proof steps have sometimes been validated simply because the tool could prove them to be true – and the lack of optimization leading to poor responsiveness for an interactive tool on ageing machines.

7 Related work

TPE [32] is a complete environment geared towards teaching proof. Proof steps are checked with Otter. Since it only provides a "point-and-click" end-user interaction mode, there is no need for defining and parsing an input language.

As for minimality, the syntax of Tutch [1] is close to Yalep's since the tool was conceived with the same objective in mind. However, the tool does not seem to handle any other theory than natural numbers, logic, and lists.

Lean Verbose [24] and Coq Waterproof [43], respectively based on proof assistants Lean and Rocq, provide other controlled natural languages (CNL) with comprehensive libraries. They both propose teacher configuration and help commands. Coq Waterproof also presents a proof/HTML mixed document interface. Both adapt the language to make the transfer to paper proof effective, but none of them tries to restrict the language to a minimal set of constructs.

[20] The detailed progression as well as all mentioned proof scripts — solutions to exercises in the progression, the standard proof and the Barendregt proof of irrationality of $\sqrt{2}$ — are available in the project repository.

[21] a French computer science research lab in Paris

Figure 11 shows a comparison between Lean Verbose and Yalep based on a proof of the squeeze theorem.

```
Example "The squeeze theorem. (Lean Verbose)"
 Given: (u v w:ℕ→ℝ) (ℓ:ℝ)
 Assume: (hu:u converges to ℓ)
         (hw:w converges to ℓ)
         (h :∀n,u n⩽v n) (h':∀n,v n⩽ w n)
 Conclusion: v converges to ℓ
Proof:
 Let's prove that ∀ε>0, ∃N,∀n⩾N,|v n-ℓ|⩽ε
 Fix ε>0
 Since u converges to ℓ and ε>0 we get n₁
  such that h₁:∀n⩾n₁,|u n-ℓ|⩽ε
 Since w converges to ℓ and ε > 0 we get n₂
  such that h₂:∀n⩾n₂,|w n-ℓ|⩽ε
 Let's prove that max n₁ n₂ works:
  ∀n⩾max n₁ n₂,|v n-ℓ|⩽ε
 Fix n⩾max n₁ n₂
 Since n⩾max n₁ n₂ we get
  (hn₁: n⩾n₁) and (hn₂: n⩾n₂)
 Since ∀n⩾n₁, |u n-ℓ|⩽ε and n⩾n₁ we get
  (hn₁ℓ:-ε⩽u n-ℓ) and (hn₁d:u n-ℓ⩽ε)
 Since ∀n⩾n₂,|w n-ℓ|⩽ε and n⩾n₂ we get
  (hn₂l:-ε⩽w n-ℓ) and (hn₂d:w n-ℓ⩽ε)
 Let's prove that |v n-ℓ|⩽ε
 Let's first prove that -ε⩽v n-ℓ
 Calc -ε⩽u n-ℓ by assumption
      _  ⩽v n-ℓ since u n⩽v n
 Let's now prove that v n-ℓ⩽ε
 Calc v n-ℓ⩽w n-ℓ since v n⩽w n
      _     ⩽ε     by assumption
 QED
```

```
Theorem squeeze "The squeeze theorem (Yalep)"
 Assumptions:
  (u∈ℝ^ℕ)(v∈ℝ^ℕ)(w∈ℝ^ℕ)(ℓ∈ℝ)
  (u converges to ℓ)(w converges to ℓ)
  (for all natural n,u(n)⩽v(n))
  (for all natural n,v(n)⩽w(n))
  Conclusion: (v converges to ℓ)
Proof
 let ε > 0
 ◆ there exists a natural n₁ such that
   for all natural n,if n⩾n₁ then |u(n)-ℓ|⩽ε
 obtain such n₁
 ◆ there exists a natural n₂ such that
   for all natural n,if n⩾n₂ then |w(n)-ℓ|⩽ε
 obtain such n₂
 define n₀:=max n₁ n₂
 ◆ for all natural n,if n⩾n₀ then |v(n)-ℓ|⩽ε
  proof
   let n be a natural
   assume n⩾ n₀
   ◆ |u(n)-ℓ|⩽ε since n⩾n₁
   ◆ |w(n)-ℓ|⩽ε since n⩾n₂
   ◎ -ε⩽u(n)-ℓ
       _ ⩽v(n)-ℓ
   ◎ v(n)-ℓ⩽w(n)-ℓ
         _
         ⩽ ε
   ◆ |v(n)-ℓ|⩽ε
  □
□
```

Fig. 3: Comparison of LeanVerbose and Yalep for the squeeze theorem. The proof in Lean Verbose is extracted from [24]. In the header, when data are defined by a type ascription in Lean Verbose (`ℓ:ℝ`), the same is obtained by a predicate in Yalep (`ℓ∈ℝ`)(for that reason it appears in the section 'Assumptions'). The type ascription (`ℓ:Number`) exists in the underlying Lean representation but is implicitly inferred. Lean Verbose expects a precise justification of each fact, expressed in a rich syntax, whereas Yalep allows it to be omitted. In Lean Verbose, some specific syntax is dedicated to backward introduction of existential quantifier and conjunction, as a few (respectively `Let's prove that ... works` and `Let's first prove ...`)

Diproche [7] also proposes a CNL geared towards education. The most advanced student error diagnosis technology we know of is "antiATP" technology.[22] As the verification engine is specially tailored to enable better control over feedback, Diproche does not rely on an existing and maintained comprehensive library. As a consequence, available topics, currently limited to elementary number theory, set theory, and geometry, may be difficult to extend. On the interaction side, Diproche does not display the proof state.

Proof Buddy [18] uses the declarative language of Isar but is geared towards learning logic, leaving the rules of meta-logic apparent.

[22] Carl calls the principle of "buggy rules" antiATP, which consists of adding students' frequent errors as lemmas and checking whether automation solves the goal. See also [6,37] for details.

While proposing a non-ambiguous syntax with minimal redundancy, Yalep does not aim at recognizing proofs nor statements expressed in natural language nor distinguishing ambiguities that arise from such a language, like in [46]. As ambiguities of natural language are additional hurdles to learning proof [10,11,15], we propose to separate the learning of logic from the acculturation to mathematical style. Our goal is hence very different from that of Naproche [21], or even Diproche, which uses the same syntax for the introduction of variables/assumptions and the elimination of the existential quantifier.

Finally, to our knowledge, none of the cited tools, based or not on type theory, claim to make proofs involving functions defined on subsets of $\mathbb{R}$ accessible to high-school students.[23]

Other comparisons of Yalep with Lean Verbose, Coq Waterproof, and Diproche are available in the Yalep repository.

8 Conclusion and Future Work

We have presented Yalep, a Lean-based prototype of a pedagogical environment for high-school students to learn the concept of proof using a proof assistant. Yalep's input language for proofs is inspired by coherent logic, and designed to favor forward chaining of facts.

In addition to text-based proof writing, Yalep provides point-and-click helpers and controlled automation. It also offers a formalization of the notion of function (from sets to sets), and an ambient type `Number` that prevents explicit type coercion. All these features were developed with ease of use for high-school students in mind, as well as similarity to pen-and-paper proofs, and were tested by tenth-grade students.

Yalep has several limitations: automation can be slow, especially when it fails to prove the statement proposed by the student; and it is sometimes too powerful in automatically proving statements that the teacher would require a justification for. Moreover, our formalization of numbers is not geared toward computations.

In the future, we plan to work on improving the automation to make it more configurable by the teacher. Our next goal is to improve the feedback given to the students (including error messages) beyond the success/failure diagnostic. We will also evaluate the pedagogical impact of our tool (using students/teachers surveys, pre- and post-tests, analysis of student productions), in collaboration with maths teachers and experts in math education.

Acknowledgments This work is partially funded by the ANR Project APPAM ANR-23-CE38-0009-01. Yalep front-end is inspired by Lean Verbose and even relies on some pieces of its code, specifically: parts of its parser for theorem headers and the 'Calc' environment, and some helper functions.

[23] At the time of submitting the paper, Massot informed us that he tweaked Lean Verbose to make type number coercions transparent to the student and provided a convincing proof that $\sqrt{2} \notin \mathbb{Q}$. Due to time constraints, we could not investigate deeply.

References

1. Abel, A., Chang, B.Y.E., Pfenning, F.: Human-Readable Machine-Verifiable Proofs for Teaching Constructive Logic. In: Workshop Proof Transformation and Presentation and Proof Complexities (PTP'01) (2001), https://www.cs.cmu.edu/~fp/papers/ptp01.pdf
2. Autexier, S.: The CoRe Calculus. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Nieuwenhuis, R. (eds.) Automated Deduction – CADE-20, vol. 3632, pp. 84–98. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11532231_7
3. Bartzia, E., Meyer, A., Narboux, J.: Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis. In: Trigueros, M. (ed.) INDRUM 2022: Fourth conference of the International Network for Didactic Research in University Mathematics. pp. 253–262. Reinhard Hochmuth, Hanovre, Germany (Oct 2022), https://hal.science/hal-03648357
4. Bertot, Y., Portet, T.: Chassez le naturel dans la formalisation des mathématiques. In: 36es Journées Francophones des Langages Applicatifs (JFLA 2025). Roiffé, France (Jan 2025), https://inria.hal.science/hal-04757635
5. Boldo, S., Clément, F., Hamelin, D., Mayero, M., Rousselin, P.: Teaching divisibility and binomials with coq. In: Proceedings of 13th International Workshop on Theorem proving components for Educational software (ThEdu'24). vol. 419, p. 124–139. Open Publishing Association (May 2025). https://doi.org/10.4204/eptcs.419.8, http://dx.doi.org/10.4204/EPTCS.419.8
6. Carl, M.: Using Automated Theorem Provers for Mistake Diagnosis in the Didactics of Mathematics. CoRR **abs/2002.05083**(arXiv:2002.05083) (feb 2020). https://doi.org/10.48550/arxiv.2002.05083, https://arxiv.org/abs/2002.05083
7. Carl, M., Lorenzen, H., Schmitz, M.: Natural Language Proof Checking in Introduction to Proof Classes – First Experiences with Diproche. In: Proceedings of the International Workshop on Theorem Proving Components for Educational Software (Th'Edu) 2021. vol. 354, pp. 59–70 (Feb 2022). https://doi.org/10.4204/EPTCS.354.5
8. Carter, N.C., Monks, K.G.: Lurch: A Word Processor that Can Grade Students' Proofs. In: Lange, C., Aspinall, D., Carette, J., Davenport, J., Kohlhase, A., Kohlhase, M., Libbrecht, P., Quaresma, P., Rabe, F., Sojka, P., Whiteside, I., Windsteiger, W. (eds.) Joint Proceedings of the MathUI, OpenMath, PLMMS and ThEdu Workshops and Work in Progress at the Conference on Intelligent Computer Mathematics 2013. No. 1010 in CEUR Workshop Proceedings, Aachen (2013), http://ceur-ws.org/Vol-1010/paper-04.pdf
9. Coen, C.S., Zoli, E.: A Note on Formalising Undefined Terms in Real Analysis. In: Proceedings of International Workshop on Proof Assistants and Types in Education (PATE07) (2007), https://www.cs.unibo.it/~sacerdot/PAPERS/pate07.pdf
10. Durand-Guerrier, V., Njomgang-Ngansop, J.: Questions de logique et de langage à la transition secondaire - supérieur. L'exemple de la négation. In: Kuzniak Alain & Sokhna Mohamed (ed.) Actes du Colloque Espace Mathématique Francophone. pp. 1033–1047. No. numéro spécial, Dakar, Senegal (Apr 2009), https://hal.science/hal-00804096
11. Edmonds-Wathen, C., Trinick, T., Durand-Guerrier, V.: Impact of Differing Grammatical Structures in Mathematics Teaching and Learning, pp. 23–46. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-14511-2_2, https://doi.org/10.1007/978-3-319-14511-2_2

12. Ehle, A., Hundeshagen, N., Lange, M.: The Sequent Calculus Trainer with Automated Reasoning - Helping Students to Find Proofs. In: Proceedings of 6th International Workshop on Theorem proving components for Educational software (ThEdu'17). vol. 267, pp. 19–37 (Mar 2018). https://doi.org/10.4204/EPTCS.267.2, http://arxiv.org/abs/1803.01467v1
13. Gallego Arias, E.J., Pin, B., Jouvelot, P.: jsCoq: Towards Hybrid Theorem Proving Interfaces. In: Proceedings of 5th International Workshop on Theorem proving components for Educational software (ThEdu'16). vol. 239, pp. 15–27 (Jan 2017). https://doi.org/10.4204/EPTCS.239.2
14. Gasquet, O., Schwarzentruber, F., Strecker, M.: Panda: A Proof Assistant in Natural Deduction for All. A Gentzen Style Proof Assistant for Undergraduate Students. In: Blackburn, P., van Ditmarsch, H., Manzano, M., Soler-Toscano, F. (eds.) Tools for Teaching Logic - Third International Congress, TICTTL 2011, Salamanca, Spain, June 1-4, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6680, pp. 85–92. Springer (2011). https://doi.org/10.1007/978-3-642-21350-2_11
15. Hache, C., Mesnil, Z.: Outils logiques pour analyser les formulations des preuves dans des manuels de lycée. In: Séminaire national de didactique des mathématiques de l'ARDM. Paris, France (Nov 2019), https://hal.science/hal-03257750
16. Herbst, P.G.: Establishing a custom of proving in american school geometry: Evolution of the two-column proof in the early twentieth century. Educational Studies in Mathematics **49**(3), 283–312 (2002). https://doi.org/10.1023/A:1020264906740
17. Huang, X.: Reconstruction Proofs at the Assertion Level. In: Proceedings of the 12th International Conference on Automated Deduction. p. 738–752. CADE-12, Springer-Verlag, Berlin, Heidelberg (1994)
18. Karsten, N., Jacobsen, F.K., Eiken, K.J., Nestmann, U., Villadsen, J.: ProofBuddy: A Proof Assistant for Learning and Monitoring. In: Proceedings Twelfth International Workshop on Trends in Functional Programming in Education. vol. 382, pp. 1–21 (Aug 2023). https://doi.org/10.4204/EPTCS.382.1
19. Kerjean, M., Leroux, F., Massot, P., Mayero, M., Mesnil, Z., Modeste, S., Narboux, J., Rousselin, P.: Utilisation des assistants de preuves pour l'enseignement en L1 - Retours d'expériences. Gazette Société Mathématique de France (Aug 2022), https://hal.science/hal-03979238
20. Korkut, J.: A Proof Tree Builder for Sequent Calculus and Hoare Logic. In: Proceedings ThEdu'22. vol. 375, pp. 54–62 (2023). https://doi.org/10.4204/EPTCS.375.5, http://arxiv.org/abs/2303.05865v1
21. Kühlwein, D., Cramer, M., Koepke, P., Schröder, B.: The Naproche System. In: Calculemus Emerging Trends 2009 (joint with CICM'09) (july 2009)
22. Leach-Krouse, G.: Carnap: An Open Framework for Formal Reasoning in the Browser. In: Proceedings 6th International Workshop on Theorem proving components for Educational software, ThEdu@CADE 2017. vol. 267, pp. 70–88 (Mar 2018). https://doi.org/10.4204/EPTCS.267.5
23. Machın, B., Sierra, L.: Yoda: A Simple Tool for Natural Deduction (2011), http://logicae.usal.es/TICTTL/actas/MachinSierra.pdf
24. Massot, P.: Teaching Mathematics Using Lean and Controlled Natural Language. In: Bertot, Y., Kutsia, T., Norrish, M. (eds.) 15th International Conference on Interactive Theorem Proving (ITP 2024). Leibniz International Proceedings in Informatics (LIPIcs), vol. 309, pp. 27:1–27:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2024). https://doi.org/10.4230/LIPIcs.ITP.2024.27
25. Materzok, M.: Easyprove: A Tool for Teaching Precise Reasoning. In: Proceedings of the Fourth International Conference on Tools for Teaching Logic (TTL2015). M. An-

tonia Huertas, João Marcos, María Manzano, Sophie Pinchinat, François Schwarzentruber, Rennes, France (June 2015), https://arxiv.org/abs/1507.03675
26. Nawrocki, W., Ayers, E.W., Ebner, G.: An Extensible User Interface for Lean 4. In: Naumowicz, A., Thiemann, R. (eds.) 14th International Conference on Interactive Theorem Proving (ITP 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 268, pp. 24:1–24:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.ITP.2023.24, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2023.24
27. Perháč, J., Novotný, S., Chodarev, S., Kristensen, J.T., Tveito, L., Shturmov, O., Thomsen, M.K.: OnlineProver: Experience with a Visualisation Tool for Teaching Formal Proofs. In: Proceedings of 13th International Workshop on Theorem proving components for Educational software (ThEdu'24) (2024). https://doi.org/10.5324/nikt.6205
28. Rognier, B., Duhamel, G.: Présentation de la plateforme Edukera. In: Signoles, J. (ed.) Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016). Saint-Malo, France (Jan 2016), https://hal.science/hal-01333606
29. Schiller, M.R.G.: Granularity Analysis for Mathematical Proofs. Topics in Cognitive Science **5**(2), 251–269 (Apr 2013). https://doi.org/10.1111/tops.12012
30. Selsam, D., Moura, L.: Congruence Closure in Intensional Type Theory. In: Proceedings of the 8th International Joint Conference on Automated Reasoning - Volume 9706. p. 99–115. Springer-Verlag, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-319-40229-1_8, https://doi.org/10.1007/978-3-319-40229-1_8
31. Siekmann, J., Benzmüller, C., Fiedler, A., Meier, A., Normann, I., Pollet, M.: Proof Development in OMEGA: The Irrationality of Square Root of 2. pp. 271–314 (Jan 2003)
32. Sommer, R., Nuckols, G.: A Proof Environment for Teaching Mathematics. Journal of Automated Reasoning **32**, 227–258 (2004). https://doi.org/10.1023/B:JARS.0000044825.55318.95
33. Stojanovic, S., Narboux, J., Bezem, M., Janicic, P.: A Vernacular for Coherent Logic. In: Lecture Notes in Computer Science. Lecture Notes in Computer Science, vol. 8543, p. 16. Springer, Coimbra, Portugal (Jul 2014). https://doi.org/10.1007/978-3-319-08434-3_28, https://inria.hal.science/hal-00983975
34. Stylianides, G.J., Stylianides, A.J., Moutsios-Rentzos, A.: Proof and proving in school and university mathematics education research: a systematic review. ZDM – Mathematics Education **56**(1), 47–59 (Feb 2024). https://doi.org/10.1007/s11858-023-01518-y, https://link.springer.com/10.1007/s11858-023-01518-y
35. Tao, T.: QED - an interactive textbook. https://teorth.github.io/QED/ (2018)
36. Thompson, D., Seligman, J.: Teaching natural deduction in the right order with natural deduction planner. IfCoLog Journal of Logics and their Applications **4**(1), 193–219 (January 2017)
37. Tran Minh, F., Gonnord, L., Narboux, J.: Proof Assistants for Teaching: a Survey. In: Proceedings of 13th International Workshop on Theorem proving components for Educational software (ThEdu'24). vol. 419, p. 1–27. Open Publishing Association (May 2025). https://doi.org/10.4204/eptcs.419.1, http://dx.doi.org/10.4204/EPTCS.419.1
38. Vasconcelos, D.R.: ANITA: Analytic Tableau Proof Assistant. Electronic Proceedings in Theoretical Computer Science **375**, 38–53 (2023). https://doi.org/10.4204/EPTCS.375.4, http://arxiv.org/abs/2303.05864v1, in Post-Proceedings ThEdu'22
39. Villadsen, J., Jacobsen, F.K.: Using Isabelle in Two Courses on Logic and Automated Reasoning. In: Ferreira, J.F., Mendes, A., Menghi, C. (eds.) Formal Methods Teaching. pp. 117–132. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-91550-6-9

40. Villadsen, J., Jensen, A.B., Schlichtkrull, A.: NaDeA: A Natural Deduction Assistant with a Formalization in Isabelle. In: Proceedings of the Fourth International Conference on Tools for Teaching Logic (TTL2015) (Jul 2015). https://doi.org/10.48550/arXiv.1507.04002
41. Villadsen, J.: Minimal Sequent Calculus for Teaching First-Order Logic: Lessons Learned. In: Proceedings of 13th International Workshop on Theorem proving components for Educational software (ThEdu'24). vol. 419, p. 75–89. Open Publishing Association (May 2025). https://doi.org/10.4204/eptcs.419.5, http://dx.doi.org/10.4204/EPTCS.419.5
42. Wemmenhove, A.: Waterproof: Transforming a proof assistant into an educational tool. Ph.D. thesis, Mathematics and Computer Science (Mar 2025)
43. Wemmenhove, J., Arends, D., Beurskens, T., Bhaid, M., McCarren, S., Moraal, J., Rivera Garrido, D., Tuin, D., Vassallo, M., Wils, P., Portegies, J.: Waterproof: Educational Software for Learning How to Write Mathematical Proofs. In: Proceedings ThEdu'23. vol. 400, p. 96–119. Open Publishing Association (Apr 2024). https://doi.org/10.4204/eptcs.400.7
44. Wiedijk, F.: The Seventeen Provers of the World. Springer, Berlin, Heidelberg (2006)
45. Wiedijk, F., Zwanenburg, J.: First Order Logic with Domain Conditions. In: Goos, G., Hartmanis, J., Van Leeuwen, J., Basin, D., Wolff, B. (eds.) Theorem Proving in Higher Order Logics, vol. 2758, pp. 221–237. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/10930755_15
46. Xie, L., Hui, Z., Cao, Q.: A Natural Formalized Proof Language. In: Chin, W.N., Xu, Z. (eds.) Theoretical Aspects of Software Engineering. pp. 446–464. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-64626-3_26

Appendix A The syntax in BNF

Figure 4 describes the syntax of Yalep's proof language in BNF form. The root symbol is `proof`. `?` is a placeholder for a goal which remains to be proved. The grammar for terms is just the one of Lean, extended with domain specific notations (`n is odd` for `odd n`, `Absurd` for `False`, and for $\wedge, \ldots$).

⟨*proof*⟩ ::= 'Proof'
  ⟨*tactic*⟩* ⟨*finalTactic*⟩?
  □

⟨*tactic*⟩ ::= 'let' ⟨*identRelativeBinder*⟩
  | 'let' ⟨*ident*⟩ 'be' ⟨*numberType*⟩
  | 'assume' ⟨*prop*⟩
  | 'obtain such' ⟨*ident*⟩
  | ◆ ⟨*assumptionName*⟩? ⟨*prop*⟩
    ⟨*justification*⟩?
  | ◎ (⟨*calcStep*⟩)*
  | 'define' ⟨*ident*⟩ ':=' ⟨*term*⟩

⟨*term*⟩ ::= ⟨*prop*⟩
  | ⟨*proof_term*⟩
  | ... terms of other types...

⟨*calcStep*⟩ ::= ⟨*prop*⟩ ⟨*justification*⟩?

⟨*assumptionName*⟩ ::= ⟨*ident*⟩ ':'

⟨*numberType*⟩ ::= 'a number'
  | 'a' ('real' | 'rational' | 'natural')
    'number'?
  | 'an integer'?

⟨*identType*⟩ ::= ⟨*numberType*⟩ ⟨*ident*⟩
  | ⟨*ident*⟩ ('of type' | ':') ⟨*type*⟩
  | ⟨*ident*⟩ ⟨*identRelativeBinder*⟩

⟨*identRelativeBinder*⟩ ::= ⟨*ident*⟩ ∈ ⟨*term*⟩
  | ⟨*ident*⟩ > ⟨*term*⟩
  | etc

⟨*justification*⟩ ::= 'since' ⟨*prop*⟩
  | 'since [' ⟨*prop*⟩(,⟨*prop*⟩)* ']'
  | 'by' ⟨*ident*⟩
  | 'by [' ⟨*ident*⟩(,⟨*ident*⟩)* ']'
  | 'by induction'
  | 'by cases'
  | 'by cases :' ⟨*prop*⟩ 'or' ⟨*prop*⟩
  | 'proof' ⟨*proofBody*⟩ □

⟨*finalTactic*⟩ ::= 'witness' ⟨*term*⟩ 'works'
  | '?'

Fig. 4: BNF grammar

Appendix B A proposed high-school progression

B.1 A pedagogical progression

Table 2 presents a pedagogical progression for high-school students. We do not want to teach the reasoning rules using abstract propositional logic, but rather with concrete mathematical statements. We decide to focus on the topic of parity, which is a classical topic in high-school mathematics and allows introducing different kind of reasoning: proof by cases, by contradiction, by induction. Whereas in the first exercises all assumptions are already given in the context, we choose to introduce the concept of universal quantification and implication from steps 4 and 5.[24].

Table 2: A pedagogical progression to introduce the main proof connectors to 10h-grade students on the topic of parity.

n°	proof structure to learn	example of statement to prove
1	Introduce new facts ; understand what is the goal in a proof	Prove that $81 = 2 * 40 + 1$
2	Introduce $\exists$. Understand the concept of definition.	Prove that 81 is odd.
3	Eliminate $\exists$. Understand the concept of assumption, and free variable.	Given an odd integer n, prove that $n + 1$ is even. Given an odd integer n, prove that n^2 is odd.
4	Introduce $\Longrightarrow$ (in the form "if P then Q").	Given an integer n, prove that "if n si even then n^2 is even".
5	Introduce $\forall$ (in the form "for all n ,...").	Prove that "for all integer n, for all integer b, if n is even then $n * b$ is even".
6	Introduction of or .	Prove that "0 is even or 0 is odd".
7	Elimination of or : proof by cases. Reusing a previously proved theorem.	Prove that "for all integer n, $n(n + 1)$ is even". (postponing the proof than that every integer is even or odd)
8	Proof of negation.	Prove that 1 is not even. Prove that no integer is both odd and even.
9	Proof by induction.	Prove by induction that for all natural n, we have $n \leqslant n^2$. Prove by induction that every integer is either odd or even. (hint: state a first theorem for every natural)
10	Introduction of $\Longleftrightarrow$ (in the form "... if and only if ...").	Prove that for all integer n, n is odd if and only if n is not even.
11	Proof by contrapositive.	Prove that for all integer n, if n^2 is even then n is even.
12	A proof with several steps reusing the previous ones.	Prove that $\sqrt{2}$ is irrational (provided that every rational r is written in the form p/q where p is integer, q is positive integer and p and q are not both even).

[24] All proof scripts for these exercises are available at : https://gricad-gitlab.univ-grenoble-alpes.fr/yalep/Yalep/-/blob/main/ExamplesEnglish/01_HighSchoolProgression_v1.lean

B.2 Solution to the exercises

Figures 5 to 8 depict solutions to the exercises of the previous section. These examples demonstrate in our opinion the readability of our proof language.

```
Theorem "1. Facts"
  Conclusion: 81 = 2*40 + 1
Proof
  ◆ 81 = 2*40 + 1
□
```

```
Theorem "2. Proving 'there exists' "
  Conclusion: 81 is odd
Proof
  ◆ 81 = 2*40 + 1
  ◆ there exists an integer k such that 81 = 2*k + 1
    proof
      witness 40 works
    □
  ◆ 81 is odd
□
```

```
Theorem successor_odd "3a. Using 'there exists' "
  Assumptions: (n is integer)  (n is odd)
  Conclusion: n+1 is even
Proof
  ◆ there exists an integer k such that n = 2*k+1
  obtain such k
  ◆ n+1 = 2*(k+1)
  ◆ there exists an integer q such that n+1 = 2*q
  ◆ n+1 is even
□
```

```
Theorem square_odd "3b. Using 'there exists' "
  Assumptions: (n is integer) (n is odd)
  Conclusion: n^2 is odd
Proof
  ◆ there exists an integer k such that n = 2*k+1
  obtain such k
  ◆ n^2 = 2*(2*k^2+2*k) + 1
  ◆ there exists an integer q such that n^2 = 2*q + 1
  ◆ n^2 is odd
□
```

```
Theorem "4. Proving (if...then...)"
  Assumptions: (n is integer)
  Conclusion: if n is even then n^2 is even
Proof
  assume n is even
  ◆ there exists an integer p such that  n = 2*p
  obtain such p
  ◆ n^2 = 2*(2*p^2)
  ◆ there exists an integer q such that n^2 = 2*q
  ◆ n^2 is even
□
```

```
Theorem product_even "5. Proving (for all ...)"
  Assumptions:
  Conclusion: for all integer n, for all integer b, if n is
        even then n*b is even
Proof
  let n be an integer
  let b be an integer
  assume n is even
  ◆ there exists an integer k such that n = 2*k
  obtain such k
  ◆ n*b = 2*(k*b)
  ◆ there exists an integer q such that n*b = 2*q
  ◆ n*b is even
□
```

```
Theorem zero_even_or_odd "6. Proving (... or ... )"
  Assumptions:
  Conclusion: 0 is even or 0 is odd
Proof
  ◆ 0 = 2*0
  ◆ there exists an integer q such that 0 = 2*q
  ◆ 0 is even
  ◆ 0 is even or 0 is odd
□
```

Fig. 5: Pedagogical progression : proof scripts for exercises 1-6

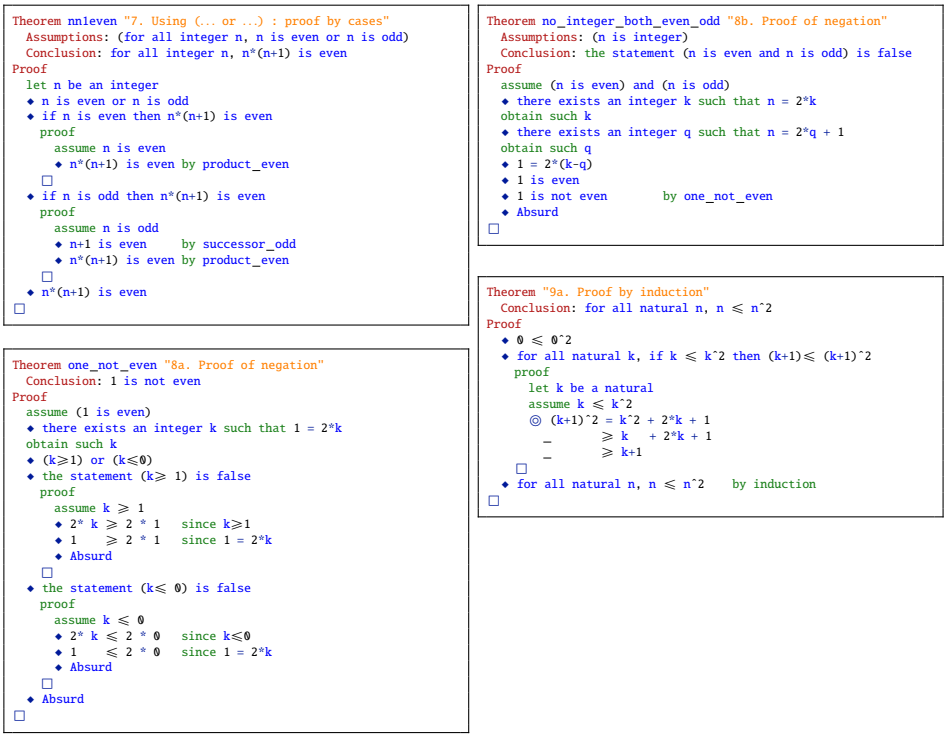
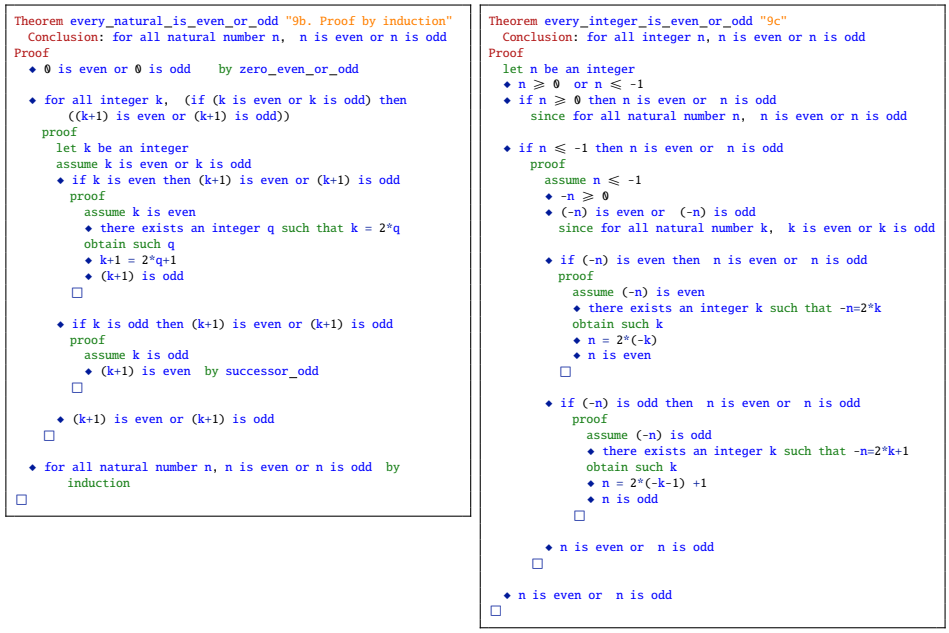


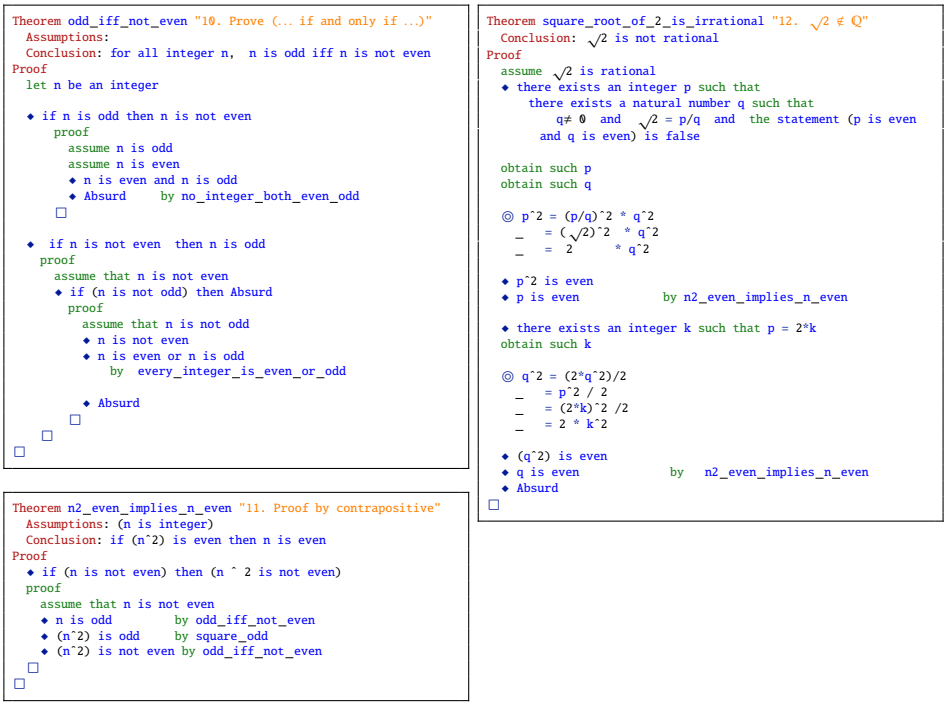
Fig. 6: Pedagogical progression : proof scripts for exercises 7-9a



```
Theorem every_natural_is_even_or_odd "9b. Proof by induction"
  Conclusion: for all natural number n,  n is even or n is odd
Proof
  ◆ 0 is even or 0 is odd    by zero_even_or_odd

  ◆ for all integer k,  (if (k is even or k is odd) then
       ((k+1) is even or (k+1) is odd))
    proof
      let k be an integer
      assume k is even or k is odd
      ◆ if k is even then (k+1) is even or (k+1) is odd
        proof
          assume k is even
          ◆ there exists an integer q such that k = 2*q
          obtain such q
          ◆ k+1 = 2*q+1
          ◆ (k+1) is odd
        □

      ◆ if k is odd then (k+1) is even or (k+1) is odd
        proof
          assume k is odd
          ◆ (k+1) is even  by successor_odd
        □

      ◆ (k+1) is even or (k+1) is odd
    □

  ◆ for all natural number n, n is even or n is odd  by
      induction
□
```

```
Theorem every_integer_is_even_or_odd "9c"
  Conclusion: for all integer n, n is even or n is odd
Proof
  let n be an integer
  ◆ n ⩾ 0  or n ⩽ -1
  ◆ if n ⩾ 0 then n is even or  n is odd
      since for all natural number n,  n is even or n is odd

  ◆ if n ⩽ -1 then n is even or  n is odd
      proof
        assume n ⩽ -1
        ◆ -n ⩾ 0
        ◆ (-n) is even or  (-n) is odd
          since for all natural number k,  k is even or k is odd

        ◆ if (-n) is even then  n is even or  n is odd
          proof
            assume (-n) is even
            ◆ there exists an integer k such that -n=2*k
            obtain such k
            ◆ n = 2*(-k)
            ◆ n is even
          □

        ◆ if (-n) is odd then  n is even or  n is odd
            proof
              assume (-n) is odd
              ◆ there exists an integer k such that -n=2*k+1
              obtain such k
              ◆ n = 2*(-k-1) +1
              ◆ n is odd
            □

        ◆ n is even or  n is odd
      □

  ◆ n is even or  n is odd
□
```

Fig. 7: Pedagogical progression : proof scripts for exercises 9b-9c



```
Theorem odd_iff_not_even "10. Prove (... if and only if ...)"
  Assumptions:
  Conclusion: for all integer n,  n is odd iff n is not even
Proof
  let n be an integer

  ◆ if n is odd then n is not even
      proof
        assume n is odd
        assume n is even
        ◆ n is even and n is odd
        ◆ Absurd      by no_integer_both_even_odd
      □

  ◆  if n is not even  then n is odd
    proof
      assume that n is not even
      ◆ if (n is not odd) then Absurd
        proof
          assume that n is not odd
          ◆ n is not even
          ◆ n is even or n is odd
              by  every_integer_is_even_or_odd

          ◆ Absurd
        □
    □
□
```

```
Theorem n2_even_implies_n_even "11. Proof by contrapositive"
  Assumptions: (n is integer)
  Conclusion: if (n^2) is even then n is even
Proof
  ◆ if (n is not even) then (n ^ 2 is not even)
  proof
    assume that n is not even
    ◆ n is odd          by odd_iff_not_even
    ◆ (n^2) is odd      by square_odd
    ◆ (n^2) is not even by odd_iff_not_even
  □
□
```

```
Theorem square_root_of_2_is_irrational "12.  √2 ∉ ℚ"
  Conclusion:  √2 is not rational
Proof
  assume  √2 is rational
  ◆ there exists an integer p such that
      there exists a natural number q such that
          q≠ 0  and   √2 = p/q  and  the statement (p is even
        and q is even) is false

  obtain such p
  obtain such q

  ◎ p^2 = (p/q)^2 * q^2
    _   = ( √2)^2  * q^2
    _   =  2       * q^2

  ◆ p^2 is even
  ◆ p is even            by n2_even_implies_n_even

  ◆ there exists an integer k such that p = 2*k
  obtain such k

  ◎ q^2 = (2*q^2)/2
    _   = p^2 / 2
    _   = (2*k)^2 /2
    _   = 2 * k^2

  ◆ (q^2) is even
  ◆ q is even             by   n2_even_implies_n_even
  ◆ Absurd
□
```

Fig. 8: Pedagogical progression : proof scripts for exercises 10-12

B.3 An example : a high-school proof with functions

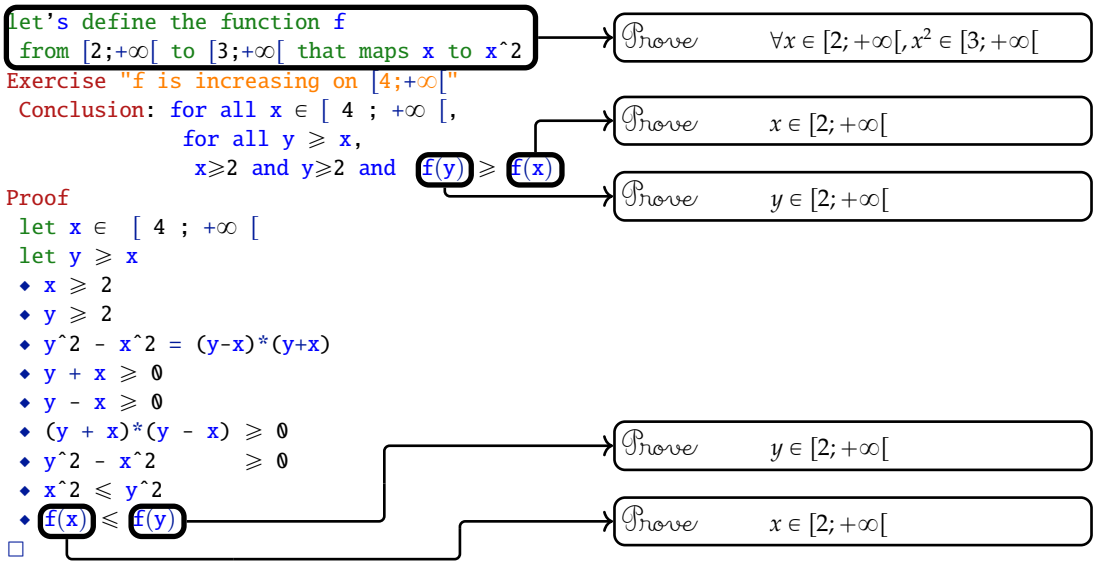


Fig. 9: Proof that the square function restricted to $[2;+\infty[\to [3;+\infty[$ is increasing on $[4;+\infty[$. Notice that the student is required to know the definition of "increasing", the system does not provide assistance for unfolding definitions. Equality between two polynomial expressions do not require justification. Yalep deals automatically with the types and the fact that x is in the domain of f, and the use of the equality is implicit.

Appendix C A comparison with Coq Waterproof, LeanVerbose and Diproche

In this section, we compare Yalep with other proof assistants and controlled natural languages. We first show a side by side comparison of Yalep with Coq Waterproof for the proof that the sum of two continuous functions is continuous 10. Then we compare Yalep with LeanVerbose for the proof of the squeeze theorem 11. Finally, we compare Yalep with Diproche using the example provided in the paper about Diproche 13.

Coq Waterproof

```
Theorem sum_continuous (f g : ℝ → ℝ) (a : ℝ)
 (_ : f is _continuous_ in a) (_ : g is _continuous_ in a) :
 (fun x : ℝ => f(x) + g(x)) is _continuous_ in a.
Proof.
 We need to show that
   (∀ ε > 0, ∃ δ > 0, ∀ x ∈ ℝ,
     0 < |x - a| < δ ⇒ |(f(x) + g(x)) - (f(a) + g(a))| < ε).
 Take ε > 0.
 Since (f is continuous in a) it holds that
   (∀ ε₁ > 0, ∃ δ₁ > 0, ∀ x ∈ ℝ,
     0 < |x - a| < δ₁ ⇒ |f(x) - f(a)| < ε₁ ) (i).
 Use ε₁ := (ε/2) in (i).
 Indeed, (ε/2 > 0).
 It holds that
   (∃ δ₁ > 0, ∀ x ∈ ℝ, 0 < |x - a| < δ₁ ⇒ |f(x) - f(a)| < ε
       /2).
 Obtain such a δ₁ .
 -- Similarly obtain δ₂ from continuity of g.
 Choose δ := min(δ₁,δ₂).
 Indeed, (δ > 0).
 We need to show that
   (∀ x ∈ ℝ,
     0 < |x - a| < δ ⇒ |(f(x) + g(x)) - (f(a) + g(a))| < ε).
 Take x ∈ ℝ.
 Assume that (0 < |x - a| < δ).
 It holds that (& 0 < |x - a| < δ ⩽ δ₁).
 It holds that (& 0 < |x - a| < δ ⩽ δ₂).
 Since (0 < |x - a| < δ₁ ) it holds that (|f(x) - f(a)| < ε/2).
 Since (0 < |x - a| < δ₂ ) it holds that (|g(x) - g(a)| < ε/2).
 We conclude that (& |(f(x) + g(x)) - (f(a) + g(a))|
                     = |f(x) - f(a) + g(x) - g(a)|
                     ⩽ (|f(x) - f(a)| + |g(x) - g(a)|)
                     < ε/2 + ε/2
                     = ε ).
Qed.
```

Yalep

```
Theorem sum_continuous "sum of continuous functions"
 Assumptions: (D ⊆ ℝ) (F ⊆ ℝ)
              (f ∈ F ^ D) (g ∈ F ^ D)  (a ∈ D)
              (f is continuous at a) (g is continuous at a)
 Conclusion: f+g is continuous at a
Proof
 let ε > 0
 ◆ ε/2 > 0
 ◆ ∃δ>0, ∀ x∈D, |x - a| ⩽ δ  ⟹ |f(x) - f(a)| ⩽ ε/2
 obtain such δ₁
 ◆ ∃δ>0, ∀ x∈D, |x - a| ⩽ δ  ⟹ |g(x) - g(a)| ⩽ ε/2
 obtain such δ₂
 define δ := min δ₁ δ₂
 ◆ δ > 0
 ◆ ∀ x∈D, |x - a| ⩽ δ  ⟹ |(f+g)(x) - (f+g)(a)| ⩽ ε
       proof
         let x ∈ D
         assume |x - a| ⩽ δ
         ◎ |x - a| ⩽ δ
           _       ⩽ δ₁
         ◆ |f(x) - f(a)| ⩽ ε/2

         ◎ |x - a| ⩽ δ
           _       ⩽ δ₂
         ◆ |g(x) - g(a)| ⩽ ε/2

         ◎ |(f+g)(x)-(f+g)(a)|=|(f(x)+g(x))-(f(a)+g(a))|
           _                 = |(f(x)-f(a))+(g(x)-g(a))|
           _                 ⩽ |f(x)-f(a)|+|g(x)-g(a)|
                                 by triangle inequality
           _                 ⩽ ε/2 + ε/2
           _                 = ε
       □
□
```

Fig. 10: Comparison of Coq Waterproof and Yalep for the sum of continuous functions. The proof in Coq Waterproof is taken from [42]. Note that Yalep handles functions from D to F whereas Coq Waterproof deals with functions from $\mathbb{R}$ to $\mathbb{R}$. In Coq Waterproof, the proof is structured by signposting (`We need to show` ...), sometimes optional, sometimes mandatory when the goal changes. In Yalep, the proof is structured by facts and possible sub-proofs. Note that Coq Waterproof's `Choose` and Yalep's `define` both assign a new variable, but `Choose` also introduces existential quantification which is automatic in Yalep.

Lean Verbose

```
Example "The squeeze theorem. (Lean Verbose)"
 Given: (u v w:ℕ→ℝ) (ℓ:ℝ)
 Assume: (hu:u converges to ℓ)
         (hw:w converges to ℓ)
         (h :∀n,u n≤v n) (h':∀n,v n≤ w n)
 Conclusion: v converges to ℓ
Proof:
 Let's prove that ∀ε>0, ∃N,∀n≥N,|v n-ℓ|≤ε
 Fix ε>0
 Since u converges to ℓ and ε>0 we get n₁
  such that h₁:∀n≥n₁,|u n-ℓ|≤ε
 Since w converges to ℓ and ε > 0 we get n₂
  such that h₂:∀n≥n₂,|w n-ℓ|≤ε
 Let's prove that max n₁ n₂ works:
  ∀n≥max n₁ n₂,|v n-ℓ|≤ε
 Fix n≥max n₁ n₂
 Since n≥max n₁ n₂ we get
  (hn₁: n≥n₁) and (hn₂: n≥n₂)
 Since ∀n≥n₁, |u n-ℓ|≤ε and n≥n₁ we get
  (hn₁ℓ:-ε≤u n-ℓ) and (hn₁d:u n-ℓ≤ε)
 Since ∀n≥n₂,|w n-ℓ|≤ε and n≥n₂ we get
  (hn₂l:-ε≤w n-ℓ) and (hn₂d:w n-ℓ≤ε)
 Let's prove that |v n-ℓ|≤ε
 Let's first prove that -ε≤v n-ℓ
 Calc -ε≤u n-ℓ by assumption
     _   ≤v n-ℓ since u n≤v n
 Let's now prove that v n-ℓ≤ε
 Calc v n-ℓ≤w n-ℓ since v n≤w n
      _     ≤ε      by assumption
 QED
```

Yalep

```
Theorem squeeze "The squeeze theorem (Yalep)"
 Assumptions:
  (u∈ℝ^ℕ)(v∈ℝ^ℕ)(w∈ℝ^ℕ)(ℓ∈ℝ)
  (u converges to ℓ)(w converges to ℓ)
  (for all natural n,u(n)≤v(n))
  (for all natural n,v(n)≤w(n))
  Conclusion: (v converges to ℓ)
Proof
 let ε > 0
 ◆ there exists a natural n₁ such that
   for all natural n,if n≥n₁ then |u(n)-ℓ|≤ε
 obtain such n₁
 ◆ there exists a natural n₂ such that
   for all natural n,if n≥n₂ then |w(n)-ℓ|≤ε
 obtain such n₂
 define n₀:=max n₁ n₂
 ◆ for all natural n,if n≥n₀ then |v(n)-ℓ|≤ε
  proof
   let n be a natural
   assume n≥ n₀
   ◆ |u(n)-ℓ|≤ε since n≥n₁
   ◆ |w(n)-ℓ|≤ε since n≥n₂
   ◎ -ε≤u(n)-ℓ
     _ ≤v(n)-ℓ
   ◎ v(n)-ℓ≤w(n)-ℓ
     _     ≤ ε
   ◆ |v(n)-ℓ|≤ε
  □
□
```

Fig. 11: Comparison of LeanVerbose and Yalep for the squeeze theorem. The proof in Lean Verbose is extracted from [24]. In the header, when data are defined by a type ascription in Lean Verbose (`ℓ:ℝ`), the same is obtained by a predicate in Yalep (`ℓ∈ℝ`)(for that reason it appears in the section 'Assumptions'). The type ascription (`ℓ:Number`) exists in the underlying Lean representation but is implicitly inferred. Lean Verbose expects a precise justification of each fact, expressed in a rich syntax, whereas Yalep allows it to be omitted. In Lean Verbose, some specific syntax is dedicated to backward introduction of existential quantifier and conjunction, as a few (respectively `Let's prove that ... works` and `Let's first prove ...`). Note that a statement `∀n≥n₁,|u(n)-ℓ|≤ε` would not be accepted in Yalep since `u(n)` needs an assumption `n∈ℕ` to elaborate. As opposite, in Lean Verbose the type ascription `n:Nat` is inferred from the term `u n` .

Lean Verbose

```
Lemme sqrt_2_is_irrational "√2 ∉ ℚ"
 Given:
 Assume:
 Conclusion: √2 ∉ ℚ
Proof:
 Assume that h : √2 ∈ ℚ
 Let's prove it's contradictory
 Since √2 ∈ ℚ we get p : ℤ and q : ℤ such that
  f01: q≠0, f02: √2=p/q and f03: ¬(p is even ∧ q is even)
 Fact f04 : p^2 = (2 : ℝ) * q^2  by
  Calc
   p^2 = (p^2/q^2) * q^2 since q ≠ 0
   _   = (p/q)^2 * q^2   by computation
   _   = (√2)^2 * q^2    since p/q = √2
   _   = 2 * q^2         by computation
 Since p^2 = 2 * q^2 we get f05 :  p^2 is even
 Since p^2 is even we get f06 : p is even
 Since p is even we get k tel que f07 : p = 2 * k
 Fact f08 : (2 * k)^2 = 2 * q^2 by
  Calc (2 * k)^2 = p^2      since p = 2 * k
       _         = 2 * q^2 by assumption
 Since (2 * k)^2 = 2 * q^2 we get f09 : 2 * k^2 = q^2
 Since q^2 = 2 * k^2 we get f10 : q^2 is even
 Since q^2 is even we get f11 : q is even
 Since ¬(p is even ∧ q is even), p is even et q is even we
       conclude that False
QED
```

Yalep

```
Theorem square_root_of_2_is_irrational "12. √2 ∉ ℚ"
  Conclusion: √2 is not rational
Proof
    assume √2 is rational
    ◆ there exists an integer p such that
        there exists a natural number q such that
            q≠ 0  and  √2 = p/q  and  the statement (p is
        even and q is even) is false

    obtain such p
    obtain such q

    ◎ p^2 = (p/q)^2 * q^2
      _   = (√2)^2  * q^2
      _   =  2      * q^2

    ◆ p^2 is even
    ◆ p is even            by n2_even_implies_n_even

    ◆ there exists an integer k such that p = 2*k
    obtain such k

    ◎ q^2 = (2*q^2)/2
      _   = p^2 / 2
      _   = (2*k)^2 /2
      _   = 2 * k^2

    ◆ (q^2) is even
    ◆ q is even             by   n2_even_implies_n_even
    ◆ Absurd

□
```

Fig. 12: Comparison of Lean Verbose and Yalep for the irrationality of $\sqrt{2}$. The Lean Verbose proof has been kindly written by Massot – the developer of Lean Verbose – for the purpose of this comparison. In Lean Verbose, each fact or computation must be carefully justified whereas Yalep allows some unjustified facts. The last version of Lean Verbose (since July 2025) allows to manipulate integer, rational or real numbers while discharging the student from explicit ascriptions and coercions (most of the time; is this example there remains an explicit type ascription `(2:ℝ)`). However, the internal mechanism is very different from Yalep's : while Yalep considers all numbers as real numbers – integer or rational are predicates over numbers – Lean Verbose keeps the original Lean number types ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$) and hides coercions into tactics. For example, `(p:ℝ)^2=2*(q:ℝ)^2` and `(p:ℤ)^2=2*(q:ℤ)^2` are two distinct statements, but the latter follows from the former by application of an `apply_mod_cast` tactic triggered by the `Since` syntax.

Diproche

```
Let x be an integer.
Prove: If x is even, then 2−3x is even.
  Proof:
    Let x be even.
    Then, there is an integer k such that x=2k.
    Let k be an integer with x=2k.
    Then we have 2−3x=2−3(2k)=2(1−3k).
    Hence 2−3x is even.
  qed.
```

Yalep

```
Theorem "Diproche like"
  Assumptions: (x is integer)
  Conclusion: if x is even then 2-3*x is even
Proof
  assume x is even
  ◆ there exists an integer k such that x=2*k
  obtain such k
  ◎ 2-3*x = 2-3*(2*k)
    _     = 2*(1-3*k)
  ◆ 2-3*x is even
□
```

Fig. 13: Comparison of Diproche and Yalep for a simple exercise. The proof in Diproche is taken from [7]. In Yalep, the user does not have to choose between 'Then' and 'Hence' nor understand possible shades of meaning between both. In Diproche, multiplication is implicit; and the same word 'let' is used for introducing a symbol, an assumption, or eliminate the existential quantifier, whereas in Yalep we have chosen to keep three different words (let, assume, obtain).

Appendix D Backward vs Forward Proofs

Tables 3 and 4 depict two versions of a proof by induction that every natural number is either even or odd, using backward chaining or not. The first version, in Table 3, is written with extensive backward chaining in a pseudo language very similar to Lean Verbose.[25]

In the second version depicted in Table 4, written in Yalep with mainly forward chaining[26], the numbers on the left refer to the corresponding tactic number in version 1. On a blackboard, we could write the proof version 2 in the chronological order described by the numbers. For example, we could start to write the conclusion step 1 at the bottom of the blackboard, and then write step 2 (0 is even or 0 is odd) at the top of the blackboard, and then write step 3 (0 is even) a few lines under, etc.

When `k` is a natural number, `P (k)` will denote " `k is even or k is odd` " and the goal will be $\vdash$ `∀ n ∈ ℕ, P (n)` .

*

[25] We have taken liberties with the language of LeanVerbose to ease the comparison with the other version.

[26] For the sake of conciseness, and as the purpose of this comparison is to focus on forward and backward reasoning, we used the symbols $\Longrightarrow$, $\forall$, $\exists$, even it would have been appropriate for high school students to use their literal versions (if . . . then . . . , for all, there exists) as Yalep allows.

Table 3: First version: with extensive backward chaining

n°	Proof step	Bwd Fwd	tactic	new goal(s)
1	`Let's prove it by induction`	B	induction	2 new goals : 1. ⊢ `P (0)` 2. ⊢ `∀ k ∈ ℕ, P (k) ⟹ P (k+1)`
quasi silent start of 1st goal (2nd goal pending)				
2	`Let's prove that 0 is even or 0 is odd`		recalls current goal (optional)	
3	`Let's prove that 0 is even`	B	Or introduction (left)	⊢ `0 is even`
4	`We unfold even`	B	unfold	⊢ `∃ k ∈ ℕ, 0 = 2*k`
5	`Let's prove that 0 works`	B	existential introduction (witness 0)	⊢ `0 = 2*0`
6	`We compute`		solves current goal	
silent switch to next goal, since the previous has been solved				
7	`Let's prove that ∀k∈ℕ, P (k) ⟹ P (k+1)`		recalls current goal (optional)	
8	`Let k be an integer`	B	∀ introduction	⊢ `P (k) ⟹ P (k+1)`
9	`Assume that k is even or k is odd`	B	⟹ introduction	⊢ `P(k+1)`
10	`Let's discuss whether k is odd or even`	B	Or elimination	2 new goals : 1. ⊢ `k is even ⟹ P (k+1)` 2. ⊢ `k is odd ⟹ P (k+1)`
quasi silent start of 1st goal (2nd goal pending) : k is even ⟹ P(k+1)				
11	`Assume that k is even`			⊢ `P(k+1)`
12	`By assumption, Choose an integer q such that k = 2*q`	F	creates a new variable and a new assumption	
13	`Let's prove that k+1 is odd`	B	Or introduction (right)	⊢ `k+1 is odd`
14	`Let's prove that q works`	B	introduces existential quantifier	⊢ `k+1 = 2*q+1`
15	`We compute`	B	rewrites goal	⊢ `2*q+1 = 2*q+1` (solved by reflexivity)
silent switch to next goal, since the previous has been solved : current goal : k is odd ⟹ P(k+1)				
16	`Assume that k is odd`			⊢ `P(k+1)`
17	`By assumption, choose an integer q such that k = 2*q+1`	F	creates a new variable and a new assumption	
18	`Let's prove that k+1 is even`	B	Or introduction (left)	⊢ `k+1 is even`
19	`Let's prove that q+1 works`	B	introduces existential quantifier	⊢ `k+1 = 2*(q+1)`
20	`We compute`	B	rewrites goal	⊢ `2*q+1+1 = 2*(q+1)` (solved by computation)

Table 4: Second version: Yalep with mainly forward chaining

no	Proof step	Bwd Fwd	tactic	new goal?
	sets explicitly a new temporary goal (until closing token □) (mandatory announce)			
2	◆ 0 is even or 0 is odd	F	creates a new assumption	
	proof		sub-proof of temporary goal : P(0)	
5	◆ 0 = 2*0	F	creates a new assumption	
4	◆ ∃ k ∈ ℕ, 0 = 2*k	F	introduces existential quantifier from previous facts	
3	◆ 0 is even	F	folds definition from previous fact	
	◆ 0 is even or 0 is odd	F	Or introduction (left)	
	□		closes current goal	
	sets explicitly a new temporary goal (until closing token □) (mandatory announce)			
7	◆ ∀ k ∈ ℕ, P (k) ⟹ P (k+1)	F	creates a new assumption	
	proof		sub-proof of temporary goal : $\forall k \in \mathbb{N}, P(k) \Rightarrow P(k+1)$	
8	let k be an integer	B	∀ introduction	⊢ P (k) ⟹ P (k+1)
9	assume k is even or k is odd	B	⟹ introduction	⊢ P(k+1)
	sets explicitly a new temporary goal (until closing token □) (mandatory announce)			
	◆ k is even ⟹ P (k+1)	F	creates a new assumption	
	proof		sub-proof of temporary goal : k even $\Rightarrow P(k+1)$	
11	assume k is even	B	⟹ introduction	⊢ P(k+1)
	there exists an integer q such that k=2*q	F		
12	obtain such q	F	∃ elimination	
14	◆ k+1 = 2*q+1	F	new assumption deduced from above	
13	◆ (k+1) is odd	F	∃ introduction (+ fold definition)	
	◆ (k+1) is even or (k+1) is odd	F	Or introduction (right)	
	□		closes current goal	
	sets explicitly a new temporary goal (until closing token □) (mandatory announce)			
	◆ k is odd ⟹ P (k+1)	F	creates a new assumption	
	proof		sub-proof of temporary goal : k odd $\Rightarrow P(k+1)$	
16	assume k is odd	B	⟹ introduction	⊢ P(k+1)
	there exists an integer q such that k=2*q+1	F		
17	obtain such q	F	∃ elimination	
19	◆ k+1 = 2*(q+1)	F	new assumption deduced from above	
18	◆ (k+1) is even	F	∃ introduction (+ fold definition)	
	◆ (k+1) is even or (k+1) is odd	F	Or introduction (left)	
	□		closes current goal	
10	◆ (k+1) is even or (k+1) is odd by cases : k is even or k is odd	F	or elimination	
	□		closes current goal	
1	◆ for all natural number n, n is even or n is odd by induction	F	induction	

Appendix E Another proof that $\sqrt{2} \notin \mathbb{Q}$

The proof script depicted in Figure 14 is a transcription in Yalep of the proof proposed by Henk Barendregt that $\sqrt{2}$ is irrational, published in [44] as the "informal proof" given in the first chapter before the 17 versions in proof assistants.

```
let's define the function P from ℕ to ℙ that maps m to
       (∃n∈ℕ, m^2=2*n^2 and m>0)

Theorem claim "Claim"
  Assumptions: (m∈ℕ)
  Conclusion: P(m)⟹∃m'∈ℕ, m'<m and P(m')
Proof
  assume that P(m)
  ◆ ∃n∈ℕ, m^2=2*n^2 and m>0
  obtain such n
  ◆ m^2 is even
  ◆ m is even  by n2_even_imp_n_even
  ◆ ∃ k ∈ ℤ, m = 2*k
  obtain such k
  ◆ k ⩾ 0
  ◆ k ∈ ℕ
  ◎ 2*n^2 = m^2
    _       = 4*k^2
  ◆ n^2 = 2*k^2
  ◆ m^2 > 0          since m>0
  ◆ n^2 > 0
  ◆ n > 0
  ◆ ∃k∈ℕ, n^2=2*k^2
  ◆ P(n)
  ◎ m^2 = n^2 + n^2
    _   > n^2
  ◆ m > n
  witness n works
□
```

```
Theorem claim2 "Claim2"
  Conclusion: for all natural m , the statement P(m) is false
Proof
  define A := {m∈ ℕ | P(m)}
  ◆ A ⊆ ℕ
  let m be a natural
  assume P(m)
  ◆ m ∈ A
  ◆ A ≠ ∅
  ◆ ∃ a ∈ ℕ, a is the least element of A
  obtain such a
  ◆ a ∈ A and ∀x∈A, a ⩽ x
  ◆ ∃ a' ∈ ℕ, a'< a and  P(a')  by claim
  obtain such a'
  ◆ a' ∈ A
  ◎ a'< a
    _ ⩽ a'
  ◆ Absurd
□
```

```
Theorem lemma1 "Lemma 1"
  Assumptions: (m∈ℕ) (n∈ℕ)
  Conclusion: m^2 = 2*n^2 ⟹ m=0 and n=0
Proof
  assume that m^2 = 2 * n^2
  ◆ the statement m ≠ 0 is false
    proof
      assume that m ≠ 0
      ◆ m ⩾ 0
      ◆ m > 0
      ◆ P(m)
      ◆ Absurd    by claim2
    □
  ◆ m=0
  ◆ n=0
□
```

```
Theorem sqrt_2_is_irrational "√2 ∉ ℚ"
  Conclusion: √2 is not rational
Proof
    assume that √2 is rational

    ◆ there exists an integer p such that
        there exists an integer q such that
            q≠0 and √2 = p/q

    obtain such p
    obtain such q

    ◆ √2 ⩾ 0

    define m := |p|
    define n := |q|

    ◆ m ∈ ℕ
    ◆ n ∈ ℕ
    ◆ n ≠ 0

    ◎ √2 = |√2|
      _  = |p/q|
      _  = |p|/|q|
      _  = m/n

    ◎ m^2 = (m/n)^2 * n^2
      _   = √2 ^2 * n^2
      _   = 2*n^2

    ◆ m=0 and n=0          by lemma1
    ◆ Absurd
□
```

Fig. 14: Proof script in Yalep of Barendregt's proof of the irrationality of $\sqrt{2}$