Yalep: An environment for learning proof in high-school

Frédéric Tran-Minh & Laure Gonnord & Julien Narboux

Grenoble INP - UGA & IRIF, Université Paris Cité

Europroofnet-MCLP, September 2025

Funded by the European Union

- Learning proof is hard !
- Why not teaching with proof assistants ?
  - Hands-on approach
  - Students receive instant and frequent feedback
  - Helps them realize that a proof could be mechanically verified.
- PhD ! (w/ Julien Narboux, Laure Gonnord)
  $\longrightarrow$ explore some questions raised by this educational setting

Teaching with Proof Assistants

- Learning proof is hard !
- Why not teaching with proof assistants ?
  - Hands-on approach
  - Students receive instant and frequent feedback
  - Helps them realize that a proof could be mechanically verified.
- PhD ! (w/ Julien Narboux, Laure Gonnord)

  $\longrightarrow$ explore some questions raised by this educational setting

Teaching with Proof Assistants

- Learning proof is hard !
- Why not teaching with proof assistants ?
  - Hands-on approach
  - Students receive instant and frequent feedback
  - Helps them realize that a proof could be mechanically verified.
- PhD ! (w/ Julien Narboux, Laure Gonnord)
  $\longrightarrow$ explore some questions raised by this educational setting

- Many experiments of teaching w/ many different PAs[2]

  using Rocq (Kerjean, Mayero, Rousselin), Deaduction (Leroux), LeanVerbose (Massot), Edukera & Lean proof term (Tran Minh), Deaduction & LeanVerbose (Bartzia, Boutry, Narboux), Edukera (Modeste), Coq Waterproof (Wemmenhove), Proof Buddy (Karsten), . . .
- APPAM: didacticians, mathematicians and computer scientists
- An a priori analysis[3]

[1] Tran Minh, Gonnord, and Narboux 2025, [2] Kerjean et al. 2022, [3] Bartzia, Meyer, and Narboux 2022

The idea is as old as PA themselves[1]

- Many experiments of teaching w/ many different PAs[2]

  using Rocq (Kerjean, Mayero, Rousselin), Deaduction (Leroux), LeanVerbose (Massot), Edukera & Lean proof term (Tran Minh), Deaduction & LeanVerbose (Bartzia, Boutry, Narboux), Edukera (Modeste), Coq Waterproof (Wemmenhove), Proof Buddy (Karsten), . . .

- APPAM: didacticians, mathematicians and computer scientists
- An a priori analysis[3]

[1] Tran Minh, Gonnord, and Narboux 2025, [2] Kerjean et al. 2022, [3] Bartzia, Meyer, and Narboux 2022

The idea is as old as PA themselves[1]

- Many experiments of teaching w/ many different PAs[2]

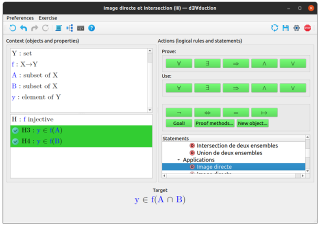
  using Rocq (Kerjean, Mayero, Rousselin), Deaduction (Leroux), LeanVerbose (Massot), Edukera & Lean proof term (Tran Minh), Deaduction & LeanVerbose (Bartzia, Boutry, Narboux), Edukera (Modeste), Coq Waterproof (Wemmenhove), Proof Buddy (Karsten), ...
- APPAM: didacticians, mathematicians and computer scientists
- An a priori analysis[3]

[1] Tran Minh, Gonnord, and Narboux 2025, [2] Kerjean et al. 2022, [3] Bartzia, Meyer, and Narboux 2022

The idea is as old as PA themselves[1]

- Many experiments of teaching w/ many different PAs[2]

  using Rocq (Kerjean, Mayero, Rousselin), Deaduction (Leroux), LeanVerbose (Massot), Edukera & Lean proof term (Tran Minh), Deaduction & LeanVerbose (Bartzia, Boutry, Narboux), Edukera (Modeste), Coq Waterproof (Wemmenhove), Proof Buddy (Karsten), . . .
- APPAM: didacticians, mathematicians and computer scientists
- An a priori analysis[3]



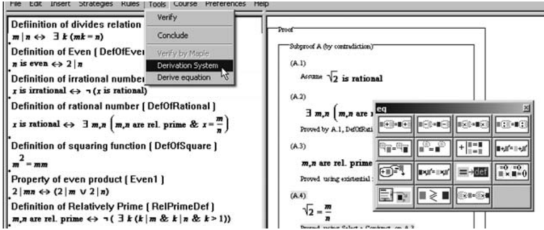
[1] Tran Minh, Gonnord, and Narboux 2025, [2] Kerjean et al. 2022, [3] Bartzia, Meyer, and Narboux 2022

The idea is as old as PA themselves[1]

- Many experiments of teaching w/ many different PAs[2]

  using Rocq (Kerjean, Mayero, Rousselin), Deaduction (Leroux), LeanVerbose (Massot), Edukera & Lean proof term (Tran Minh), Deaduction & LeanVerbose (Bartzia, Boutry, Narboux), Edukera (Modeste), Coq Waterproof (Wemmenhove), Proof Buddy (Karsten), . . .
- APPAM: didacticians, mathematicians and computer scientists
- An a priori analysis[3]

- GUI proofs may not be so instructive[3].
- Our focus : language
- Rare tools designed for High school (except for Geometry)



[1] Tran Minh, Gonnord, and Narboux 2025, [2] Kerjean et al. 2022, [3] Bartzia, Meyer, and Narboux 2022

- What is specific to a high school proof ?
  - rather short, informal proofs
  - quantifiers, logical connectors in plain text
- What is specific to the high school context ?
  - students' supposed background: PA (none) ; CS (none) ; proof (tiny)
  - teachers' supposed background: PA (none) ; CS (none) ; logics (usually little)
  - no additional training
- what is specific to the French high school curriculum
  - recommends gradually guiding students toward the truth through proof.
  - some example proofs ; ex: $\sqrt{2} \notin \mathbb{Q}$
  - initiation to logical connectors and quantifiers, without the symbols

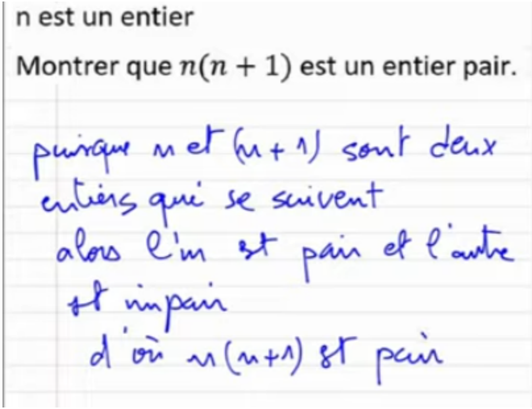
- What is specific to a high school proof ?
  - rather short, informal proofs
  - quantifiers, logical connectors in plain text
- What is specific to the high school context ?
  - students' supposed background: PA (none) ; CS (none) ; proof (tiny)
  - teachers' supposed background: PA (none) ; CS (none) ; logics (usually little)
  - no additional training
- what is specific to the French high school curriculum
  - recommends gradually guiding students toward the truth through proof.
  - some example proofs ; ex: $\sqrt{2} \notin \mathbb{Q}$
  - initiation to logical connectors and quantifiers, without the symbols

- What is specific to a high school proof ?
  - rather short, informal proofs
  - quantifiers, logical connectors in plain text
- What is specific to the high school context ?
  - students' supposed background: PA (none) ; CS (none) ; proof (tiny)
  - teachers' supposed background: PA (none) ; CS (none) ; logics (usually little)
  - no additional training
- what is specific to the French high school curriculum
  - recommends gradually guiding students toward the truth through proof.
  - some example proofs ; ex: $\sqrt{2} \notin \mathbb{Q}$
  - initiation to logical connectors and quantifiers, without the symbols

High school proofs?

Prove : $\forall n \in \mathbb{Z}, n(n+1)$ is even.

Prove : $\forall n \in \mathbb{Z}, n(n+1)$ is even.


n est un entier

Montrer que $n(n+1)$ est un entier pair.

puisque n et (n+1) sont deux
entiers qui se suivent
alors l'un est pair et l'autre
est impair
d'où n(n+1) est pair

Since n and $n+1$ are two consecutive integers, then one is even and the other is odd, so $n(n+1)$ is even.

High school proofs?

Prove : $\forall n \in \mathbb{Z}, n(n+1)$ is even.

Deux entiers consécutifs s'écrivent, par exemple, sous la forme n et $n+1$.

- Si n est pair, il existe alors un entier relatif k tel que $n = 2k$.
  Ainsi $n(n+1) = 2k(n+1)$ est pair.
- Si n est impair, il existe alors un entier relatif k tel que $n = 2k+1$.
  Par conséquent $n+1 = 2k+1+1 = 2k+2 = 2(k+1)$.
  Ainsi $n(n+1) = n \times 2(k+1)$ est pair.

Two consecutive integers are written, for instance, in the form n and $n+1$.

- If n is even, then there exists an integer k such that $n = 2k$. Therefore, $n(n+1) = 2k(n+1)$ is even.
- If n is odd, then there exists an integer k such that $n = 2k+1$. As a result, $n+1 = 2k+1+1 = 2k+2 = 2(k+1)$. Thus, $n(n+1) = n \times 2(k+1)$ is even.

Prove : $\forall n \in \mathbb{Z}, n(n+1)$ is even.

```
theorem product_even (n a:ℤ) :  Even n → Even (n*a)
  | ⟨k,hk⟩ => by use k*a ; rw[← right_distrib,hk]

theorem successor_odd (n  :ℤ) : Odd n → Even (n+1)
  | ⟨k,hk⟩ => by use k+1 ; rw[hk] ; ring_nf

example (h: ∀n:ℤ, Even n ∨ Odd n) : ∀ n:ℤ, Even (n*(n+1)) := by
  intro n
  cases (h n)
  apply product_even
  assumption
  rw [mul_comm]
  apply product_even
  apply successor_odd
  assumption
```

A pedagogical progression as a guideline

proof construct	*example of statement to prove*
new facts - goal	$\vdash 81 = 2 * 40 + 1$
Intro $\exists$	$\vdash$ 81 odd
Elim $\exists$	$n \in \mathbb{Z}, n$ odd $\vdash n + 1$ even $n \in \mathbb{Z}, n$ odd $\vdash n^2$ odd
Intro $\Longrightarrow$	n integer $\vdash n$ even $\Longrightarrow n^2$ even
Intro $\forall$	$\forall n \in \mathbb{Z}, \forall b \in \mathbb{Z}, n$ even $\Longrightarrow nb$ even
Intro $\vee$	$\vdash$ (0 even) $\vee$ (0 odd)
Elim $\vee$ (by cases)	$\forall n \in \mathbb{Z}$, (n even) $\vee$ (n odd) $\vdash \forall n \in \mathbb{Z}, n(n+1)$ even
Intro $\neg$	$\vdash \neg$(1 even) $\vdash \forall n \in \mathbb{Z}, \neg(n$ odd $\wedge$ n even)
induction	$\vdash \forall n \in \mathbb{N}, n \leqslant n^2$. $\vdash \forall n \in \mathbb{N}, n$ odd $\vee$ n even $\vdash \forall n \in \mathbb{Z}, n$ odd $\vee$ n even
Intro of $\Longleftrightarrow$	$\vdash \forall n \in \mathbb{Z}, n$ odd $\Longleftrightarrow \neg(n$ even)
contrapositive	$\vdash \forall n \in \mathbb{Z}, n^2$ even $\Longrightarrow n$ even
Synthesis	$\sqrt{2} \notin \mathbb{Q}$

Progression

- theme: parity
- leads to $\sqrt{2} \notin \mathbb{Q}$
- explores all logical connectors

Milesones

- Dec 2024 w/ 9th-graders
- June 2025 w/ 10th-grade

A pedagogical progression as a guideline

proof construct	*example of statement to prove*
new facts - goal	$\vdash 81 = 2 * 40 + 1$
Intro $\exists$	$\vdash 81$ odd
Elim $\exists$	$n \in \mathbb{Z}, n$ odd $\vdash n + 1$ even $n \in \mathbb{Z}, n$ odd $\vdash n^2$ odd
Intro $\Longrightarrow$	n integer $\vdash n$ even $\Longrightarrow n^2$ even
Intro $\forall$	$\forall n \in \mathbb{Z}, \forall b \in \mathbb{Z}, n$ even $\Longrightarrow nb$ even
Intro $\vee$	$\vdash$ (0 even) $\vee$ (0 odd)
Elim $\vee$ (by cases)	$\forall n \in \mathbb{Z}, (n$ even$) \vee (n$ odd$) \vdash \forall n \in \mathbb{Z}, n(n+1)$ even
Intro $\neg$	$\vdash \neg$(1 even) $\vdash \forall n \in \mathbb{Z}, \neg(n$ odd $\wedge$ n even$)$
induction	$\vdash \forall n \in \mathbb{N}, n \leqslant n^2$. $\vdash \forall n \in \mathbb{N}, n$ odd $\vee$ n even $\vdash \forall n \in \mathbb{Z}, n$ odd $\vee$ n even
Intro of $\Longleftrightarrow$	$\vdash \forall n \in \mathbb{Z}, n$ odd $\Longleftrightarrow \neg(n$ even$)$
contrapositive	$\vdash \forall n \in \mathbb{Z}, n^2$ even $\Longrightarrow n$ even
Synthesis	$\sqrt{2} \notin \mathbb{Q}$

Progression

- theme: parity
- leads to $\sqrt{2} \notin \mathbb{Q}$
- explores all logical connectors

Milesones

- Dec 2024 w/ 9th-graders
- June 2025 w/ 10th-grade

A pedagogical progression as a guideline

proof construct	*example of statement to prove*
new facts - goal	$\vdash 81 = 2 * 40 + 1$
Intro $\exists$	$\vdash 81$ odd
Elim $\exists$	$n \in \mathbb{Z}, n$ odd $\vdash n + 1$ even $n \in \mathbb{Z}, n$ odd $\vdash n^2$ odd
Intro $\Longrightarrow$	n integer $\vdash n$ even $\Longrightarrow n^2$ even
Intro $\forall$	$\forall n \in \mathbb{Z}, \forall b \in \mathbb{Z}, n$ even $\Longrightarrow nb$ even
Intro $\vee$	$\vdash$ (0 even) $\vee$ (0 odd)
Elim $\vee$ (by cases)	$\forall n \in \mathbb{Z}, (n$ even$) \vee (n$ odd$) \vdash \forall n \in \mathbb{Z}, n(n+1)$ even
Intro $\neg$	$\vdash \neg$(1 even) $\vdash \forall n \in \mathbb{Z}, \neg(n$ odd $\wedge$ n even$)$
induction	$\vdash \forall n \in \mathbb{N}, n \leqslant n^2$. $\vdash \forall n \in \mathbb{N}, n$ odd $\vee$ n even $\vdash \forall n \in \mathbb{Z}, n$ odd $\vee$ n even
Intro of $\Longleftrightarrow$	$\vdash \forall n \in \mathbb{Z}, n$ odd $\Longleftrightarrow \neg(n$ even$)$
contrapositive	$\vdash \forall n \in \mathbb{Z}, n^2$ even $\Longrightarrow n$ even
Synthesis	$\sqrt{2} \notin \mathbb{Q}$

Progression

- theme: parity
- leads to $\sqrt{2} \notin \mathbb{Q}$
- explores all logical connectors

Milesones

- Dec 2024 w/ 9th-graders
- June 2025 w/ 10th-grade

"Yet Another Learning Environment for Proof" (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

"Yet Another Learning Environment for Proof" (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame

"Yet Another Learning Environment for Proof" (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame

Fact (◆)

"Yet Another Learning Environment for Proof" (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame
Fact ◆)
Sub-proof

”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame
Fact ◆)
Sub-proof
Silent

”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame
Fact ◆)
Sub-proof
Silent
Short

”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame
Fact ◆)
Sub-proof
Silent
Short
Syntactic sugar

”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Proof frame
Fact ◆)
Sub-proof
Silent
Short
Syntactic sugar
Reduced vocabulary

What language is appropriate to teach proof to high-school students?

Under the constraint : *without additional training*, the language should be:

- Readable by students (without proof state display)
  - $\longrightarrow$ imitate pen and paper writing
  - $\longrightarrow$ declarative
- Writable by students
  - $\longrightarrow$ reduced vocabulary and set of proof constructs

Hope : practicing Yalep will help to transfer *proving skills* to pen and paper proof activity (and not copying proof scripts!).

Why does our need diverge from NL parsing?

(Diproche Carl, Lorenzen, and Schmitz 2022)

```
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.
```

NL parsing [Naproche Koepke 2019]:

- has different objectives (parsing textbooks)
- to accept any NL phrase, has to cope with implicit and ambiguities

Why does our need diverge from NL parsing?

(Diproche Carl, Lorenzen, and Schmitz 2022)

```
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.
```

NL parsing [Naproche Koepke 2019]:

- has different objectives (parsing textbooks)
- to accept any NL phrase, has to cope with implicit and ambiguities

- Proof λ-term : internal representation of proof in a proof assistant based on typed λ-calculus.

```
example (x:integer) : x is even ⇒ 2-3*x is even :=
 λ assumption1 ↦
  Exists.elim assumption1 λ k fact1 ↦
   Exists.intro (1-3*k) <|
    Eq.trans (congrArg (2-3*·) fact1)                    <|
    Eq.trans (congrArg (2-·)   $ (mul_assoc 3 2 k).symm)  <|
    Eq.trans (congrArg (2-·*k) $ (mul_comm 3 2))          <|
    Eq.trans (congrArg (2-·)   $ (mul_assoc 2 3 k))       <|
    Eq.trans (congrArg (· - 2*(3*k)) (rfl: (2:Int)=2*1))
      (mul_sub (2:Int) 1 (3*k)).symm
```

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
  obtain ⟨k,fact1⟩ := assumption1
  use (1-3*k)
  rw[fact1]
  ring_nf
```

Proof state

```
1 goal
x : integer
⊢ x is even ⇒ 2 - 3 * x is even
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : procedural proof script (Lean)

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
→ obtain ⟨k,fact1⟩ := assumption1
  use (1-3*k)
  rw[fact1]
  ring_nf
```

Proof state

```
1 goal
x : integer
assumption1 : x is even
⊢ 2 - 3 * x is even
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : procedural proof script (Lean)

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
  obtain ⟨k,fact1⟩ := assumption1
→ use (1-3*k)
  rw[fact1]
  ring_nf
```

Proof state

```
1 goal
case intro
x k : integer
fact1 : x = 2 * k
⊢ 2 - 3 * x is even
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : procedural proof script (Lean)

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
  obtain ⟨k,fact1⟩ := assumption1
  use (1-3*k)
→ rw[fact1]
  ring_nf
```

Proof state

```
1 goal
case h
x k : integer
fact1 : x = 2 * k
⊢ 2 - 3 * x = 2 * (1 - 3 * k)
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : procedural proof script (Lean)

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
  obtain ⟨k,fact1⟩ := assumption1
  use (1-3*k)
  rw[fact1]
→ ring_nf
```

Proof state

```
1 goal
case h
x k : integer
fact1 : x = 2 * k
⊢ 2 - 3 * (2 * k) = 2 * (1 - 3 * k)
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : procedural proof script (Lean)

```
example (x:integer) :
  x is even ⇒ 2-3*x is even :=
by
  intro assumption1
  obtain ⟨k,fact1⟩ := assumption1
  use (1-3*k)
  rw[fact1]
  ring_nf
```

Proof state

```
No goals
```

- Procedural proof script : sequence of ”tactics” building a proof term
- Focus on proof actions
- Not meant to be human-readable: needs proof state display to replay the proof

Different proof styles : declarative proof script (Lean)

```
example (x:integer) : x is even ⇒ 2-3*x is even :=
by
  intro (_ : x is even)
  obtain ⟨k,fact1 : x = 2*k⟩ := by assumption

  have : 2-3*x = 2*(1-3*k) := calc
                              2-3*x = 2-3*(2*k)   := by rw[fact1]
                              _      = 2*(1-3*k ) := by ring_nf

  have : ∃ u:ℤ, 2-3*x = 2*u := by use (1-3*k)
  have : 2-3*x is even       := by assumption

  assumption
```

- Declarative proof script : list of claims with corresponding subproof
- Focus on statement : closer to pen and paper practice

Different proof styles : Controlled Natural Language for education

```
Exercise "if x is even then 2-3*x is even"
  Given: (x:integer)
  Conclusion: x is even ⇒ 2-3*x is even
Proof:
  Assume that assumption1: x is even
  Since x is even we get k such that fact1: x = 2*k
  Let's prove that (1-3*k) works
  Calc
    2-3*x = 2-3*(2*k)  by We rewrite using fact1
    _     = 2*(1-3*k ) by computation
QED
```

Lean Verbose (Massot 2024)

```
Goal 2 is the infimum of [2, 5).
Proof.
We need to show that (2 is a _lower bound_ for [2, 5)
  ∧ (∀ l ∈ ℝ, l is a _lower bound_ for [2, 5) ⇒ l ⩽ 2)).
We show both statements.
  - We need to show that (2 is a lower bound for [2, 5)).
    We need to show that (∀ c ∈ [2, 5), 2 ⩽ c).
    Take c ∈ [2, 5).
    We conclude that (2 ⩽ c).
  - We need to show that
      (∀ l ∈ ℝ, l is a lower bound for [2, 5) ⇒ l ⩽ 2).
    Take l ∈ ℝ. Assume that (l is a lower bound for [2, 5)).
    We conclude that (l ⩽ 2).
Qed.
```

Coq Waterproof (Wemmenhove et al. 2024)

- Original tactics are renamed or redefined to better fit vernacular language
- Can mix declarative and procedural
- Goal : improve transfer to pen and paper proof
- Possible bias : it is strict programming, not natural language
- Remedy : extend vocabulary (policy?)

```
Theorem "if x is even then 2-3*x is even"
  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
□
```

dialect	*keep*	*abandon*
λ-term	structure, proof object	requires to learn functional programming
procedural	proof state	needs replaying with proof state to be understood
declarative	statements	proofs (at least if too detailed)
CNL	syntactic sugar	vocabulary extensibly large to imitate NL parsing

Towards a minimal language for high-school proofs

Key ingredients for minimality

1. No redundancy : `let` ~~fix~~ ~~take~~ / `assume` ~~suppose~~ ~~let~~ / `obtain` ~~fix~~ ~~let~~ ...
2. Avoid commands : instead of unfold, rewrite, compute : state a new fact
3. Implicit proof actions eliminate the need for specific tactics

   $\longrightarrow$ Approach inspired from coherent logic (Stojanovic et al. 2014)

Connector	Introduction	Elimination
`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)

4. Favor forward chaining (see next slide...)

Towards a minimal language for high-school proofs

Key ingredients for minimality

1. No redundancy : `let` ~~fix~~ ~~take~~ / `assume` ~~suppose~~ ~~let~~ / `obtain` ~~fix~~ ~~let~~ ...
2. Avoid commands : instead of unfold, rewrite, compute : state a new fact
3. Implicit proof actions eliminate the need for specific tactics
   ⟶ Approach inspired from coherent logic (Stojanovic et al. 2014)

Connector	Introduction	Elimination
`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)

4. Favor forward chaining (see next slide...)

Key ingredients for minimality

1. No redundancy : `let` ~~fix~~ ~~take~~ / `assume` ~~suppose~~ ~~let~~ / `obtain` ~~fix~~ ~~let~~ ...
2. Avoid commands : instead of unfold, rewrite, compute : state a new fact
3. Implicit proof actions eliminate the need for specific tactics
   $\longrightarrow$ Approach inspired from coherent logic (Stojanovic et al. 2014)

Connector	Introduction	Elimination
`P and Q`	(silent)	(silent)
`P or Q`	(silent)	(silent)
$P \implies Q$	`assume P`	(silent)
$\forall x \in E, P(x)$	`let x` $\in$ `E`	(silent)
$\exists x \in E, P(x)$	(silent)	`obtain such x`
$P \iff Q$	(silent)	(silent)

4. Favor forward chaining (see next slide...)

Towards a minimal language for high-school proofs

Key ingredients for minimality

1. No redundancy : `let` ~~fix~~ ~~take~~ / `assume` ~~suppose~~ ~~let~~ / `obtain` ~~fix~~ ~~let~~ ...
2. Avoid commands : instead of unfold, rewrite, compute : state a new fact
3. Implicit proof actions eliminate the need for specific tactics
   $\longrightarrow$ Approach inspired from coherent logic (Stojanovic et al. 2014)

Connector	Introduction	Elimination
`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)

4. Favor forward chaining (see next slide...)

Favoring forward chaining

Forward chaining	Backward chaining

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
	Fact f1: n is even ∨ n is odd
	  from every_integer_is_even_or_odd
	We discuss depending on whether
	  n is even or n is odd
	Assume that h1: n is even
	We conclude by product_even
	  applied to n and (n+1) and h1
	Assume that h2: n is odd
	We rewrite using mul_comm
	We apply product_even
	We conclude by successor_odd
	  applied to n and h2

• Forces students to explicit their goals • Relaxes constraint on order • Requires less vocabulary	• Automatic opening of several goals • Order of sub-goals imposed • Appropriate vocabulary needed

Favoring forward chaining

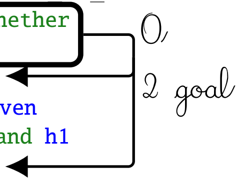
Forward chaining

```
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
```

- Forces students to explicit their goals
- Relaxes constraint on order
- Requires less vocabulary

Backward chaining

```
Fact f1: n is even ∨ n is odd
  from every_integer_is_even_or_odd
We discuss depending on whether
  n is even or n is odd
Assume that h1: n is even
We conclude by  product_even
  applied to n and (n+1) and h1
Assume that h2: n is odd
We rewrite using mul_comm
We apply product_even
We conclude by successor_odd
  applied to n and h2
```



- Automatic opening of several goals
- Order of sub-goals imposed
- Appropriate vocabulary needed

Favoring forward chaining

Forward chaining	Backward chaining

Forward chaining

```
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
```

fact

Backward chaining

```
Fact f1: n is even ∨ n is odd
  from every_integer_is_even_or_odd
We discuss depending on whether
  n is even or n is odd
Assume that h1: n is even
We conclude by  product_even
  applied to n and (n+1) and h1
Assume that h2: n is odd
We rewrite using mul_comm
We apply product_even
We conclude by successor_odd
  applied to n and h2
```

2 goal

- Forces students to explicit their goals
- Relaxes constraint on order
- Requires less vocabulary

- Automatic opening of several goals
- Order of sub-goals imposed
- Appropriate vocabulary needed

- Proof assistants already provide real time display of proof state (context, goal) and correctness: 1st level of feedback
- Lean provides
  - Flexible parser and pretty printer (Massot 2024)
  - Comprehensive, unified and community maintained Math Library (The Mathlib Community 2020)
  - Powerful tactic automation
  - Lean module allowing to interact with React/JavaScript widgets (Nawrocki, Ayers, and Ebner 2023)
  - LeanWeb interface enables to run Lean in a web browser[2]

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

Hiding types : a proof of $\sqrt{2} \notin \mathbb{Q}$ in bare Lean

```
def Rationals : Set ℝ := {x:ℝ | ∃ p:ℤ, ∃ q:ℕ, q ≠ 0 ∧ x=(p:ℝ)/(q:ℝ) ∧ ¬(p is even ∧ q is even)}
notation (priority := high) "ℚ" => Rationals
example : √2 ∉ ℚ := by
    intro (assumption1: √2 ∈ ℚ)
    let ⟨p,q,(f03: q ≠ 0),(f04: √2 = p/q),(f05 : ¬ (p is even ∧ q is even))⟩ := assumption1

    -- ensure that all computations are done with (p:ℝ) and (q:ℝ)
    have f06 : (p:ℝ)^2 = (2:ℝ)*(q:ℝ)^2  :=
      calc
        (p:ℝ)^2 = ((p:ℝ)^2/(q:ℝ)^2)*q^2      := by field_simp
        _       = (((p:ℝ)/(q:ℝ))^2) * q^2    := by ring_nf
        _       = ((√2)^2) * (q:ℝ)^2          := by rw [<√2 = (p:ℝ)/(q:ℝ)>]
        _       = (2:ℝ)*(q:ℝ)^2               := by norm_num

    -- back to (p:ℤ) and (q:ℤ) (using injectivity of coercion morphism)
    have f07 : p^2 = 2*q^2      := by rify ; rw[f06]
    have f08 : (p^2) is even    := by use q^2
    have f09 : p is even        := by apply n2_even_implies_n_even ; assumption

    let ⟨(k:ℤ), (f10: p = 2*k)⟩ := f09

    have f11 :  (2*k)^2    = 2*q^2      := by rw [← <p = 2*k> , <p^2 = 2*q^2>]
    have f13 :  2*k^2      = q^2        := by linarith
    have f14 :  (q^2) is even           := by use k^2 ; apply Eq.symm ; assumption
    have f15 : q is even                := by apply n2_even_implies_n_even ; assumption
    have f16 : p is even                := by apply n2_even_implies_n_even ; assumption
    have f17 : (p is even ∧ q is even) := by constructor <;> assumption
    contradiction
```

```
-- ...obtain  (p:ℤ) and (q:ℕ) such that f04: √2 = p/q
-- ... and p and q not both even

-- ensure that all computations are done with (p:ℝ) and (q:ℝ)
have f06 : (p:ℝ)^2 = (2:ℝ)*(q:ℝ)^2  :=
  calc
    (p:ℝ)^2 = ((p:ℝ)^2/(q:ℝ)^2)*q^2     := by field_simp
    _       = (((p:ℝ)/(q:ℝ))^2) * q^2   := by ring_nf
    _       = ((√2)^2) * (q:ℝ)^2         := by rw [f04]
    _       = (2:ℝ)*(q:ℝ)^2              := by norm_num

-- back to (p:ℤ) and (q:ℤ) (using injectivity of coercion morphism)
have f07 : p^2 = 2*q^2      := by rify ; rw[f06]
-- ...
```

- Bertot and Portet 2025 advocate for a unique number type.
  $\longrightarrow$ They formalize $\mathbb{N}$ as an inductive predicate over the `Real` type.
- Yalep integrates the same idea with a simpler formalization :

Given	a reference type a type to be represented an injective coercion morphism	T : Type E : Type $c : E \to T$
we expose	the image of E under c : $c\langle E\rangle := \{x : T \mid \exists y : E, x = c(y)\}$	
we hide	the underlying types : E , T	

- When $T =$ `Real`, this applies similarly for $E \in \{$`Nat`, `Int`, `Rat`$\}$.
- Automation should prove stability statements like
  $\forall x, \forall y, x \in c\langle E\rangle \wedge y \in c\langle E\rangle \Rightarrow x + y \in c\langle E\rangle$.

Hiding types : a proof of $\sqrt{2} \notin \mathbb{Q}$ in Yalep

```
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 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
 obtain an integer k such that p = 2*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
□
```

```
let's define the function f
 from [2;+∞[ to [3;+∞[ that maps x to x^2
Exercise "f is increasing on [4;+∞["
 Conclusion: for all x ∈ [ 4 ; +∞ [,
               for all y ⩾ x,
                x⩾2 and y⩾2 and  f(y) ⩾ f(x)
Proof
 let x ∈  [ 4 ; +∞ [
 let y ⩾ x
 ◆ x ⩾ 2
 ◆ y ⩾ 2
 ◆ y^2 - x^2 = (y-x)*(y+x)
 ◆ y + x ⩾ 0
 ◆ y - x ⩾ 0
 ◆ (y + x)*(y - x) ⩾ 0
 ◆ y^2 - x^2       ⩾ 0
 ◆ x^2 ⩽ y^2
 ◆ f(x) ⩽ f(y)
□
```

Problematics:

- Proof assistants based on type theory manipulate type-total functions

$$f : \alpha \to \beta$$

- How to encode type-partial functions ?

We expect :

(i) f contains its domain ($[2; +\infty[$) and codomain ($[3; +\infty[$)

(ii) writing $f(x)$ is allowed because $x \in$ domain is provable

A first representation[3] guarantees (i) and (ii):

$$f : \uparrow D \to \uparrow F$$

where $\uparrow D := \sum_{x:\alpha} (x \in D)$ denotes the type of dependent pairs $\langle$`x:α,hx: x ∈ D`$\rangle$.

Syntactic sugar[4] :

- `f(3)` denotes `(f ⟨3,(by norm_num:(2:Real)⩽3)⟩).val`
- `let's define the function f from [2;+∞[ to [3;+∞[ that maps x to x^2` denotes:

```
def f:↑[2;+∞[→↑[3;+∞[:=fun ⟨x,(hx:x⩾2)⟩=>⟨x^2,((by nlinarith):(x^2 ⩾ 3))⟩
```

---

[3] described in C. Paulin's lecture `https://www.lri.fr/~paulin/LASER/coq-slides4.pdf`

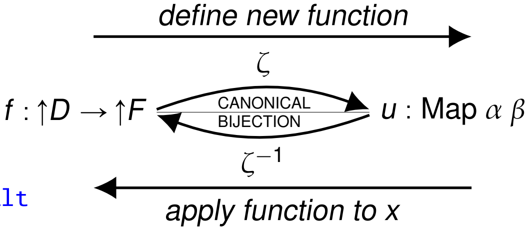
[4] Without sugar, discarded by Coen and Zoli 2007 in educational context

Drawbacks of first representation :

- A type appears to the user ($\uparrow D \to \uparrow F$)
- $f : \uparrow D \to \uparrow F$ and its derivative $f' : \uparrow D'_f \to \uparrow F'$ have distinct types, thus $\left\{ f \mid D'_f = D \text{ and } f = f' \right\}$ does not typecheck

Second representation:

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



```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Automation : example 1/3

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Finds relevant assumption; eliminate ∀ in it

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Finds relevant assumption;
eliminate ∀ in it

As
assumptions in context

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Finds relevant assumption; eliminate ∀ in it

Ar

assumptions in context

The lemma give $(n+1)n$ even. Unifie $n(n+1)$ by commutativity.

```
Theorem nn1even "7. Using (... or ...) : proof by cases"
    Assumptions: (for all 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
□
```

Finds relevant assumption; eliminate ∀ in it

As

assumptions in context

The lemma give $(n+1)n$ even. Unifie $n(n+1)$ by commutativity.

Eliminate

Automation : example 2/3

```
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 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
 obtain an integer k such that p = 2*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
□
```

```
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 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
```

$q \neq 0$ *and* $\sqrt{2} = p/q$ *and* ...
broken into 3 fact

```
◆ p^2 is even
◆ p is even by n2_even_implies_n_even
obtain an integer k such that p = 2*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
□
```

```
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 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
```

q ≠ 0 and √2 = p/q and . . .
broken into 3 fact

rewrite
compute

```
◆ p^2 is even
◆ p is even by n2_even_implies_n_even
obtain an integer k such that p = 2*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
□
```

```
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 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
```

q ≠ 0 and √2 = p/q and...
broken into 3 fact

rewrite
compute

```
◆ p^2 is even
◆ p is even by n2_even_implies_n_even
obtain an integer k such that p = 2*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
□
```

Introduce ∃

```
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 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
```

$q \neq 0$ and $\sqrt{2} = p/q$ and ...
broken into 3 fact

rewrite
compute

```
◆ p^2 is even
◆ p is even by n2_even_implies_n_even
obtain an integer k such that p = 2*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
□
```

Introduce ∃

Auto infers
$q \in \mathbb{Z}$

```
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 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
```

q ≠ 0 and √2 = p/q and ...
broken into 3 fact

rewrite
compute

```
◆ p^2 is even
◆ p is even by n2_even_implies_n_even
obtain an integer k such that p = 2*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
□
```

Introduce ∃

Auto infers
$q \in \mathbb{Z}$

Introduce
Eliminate ⇒' in
relevant assumption

```
let's define the function f
 from [2;+∞[ to [3;+∞[ that maps x to x^2
Exercise "f is increasing on [4;+∞["
 Conclusion: for all x ∈ [ 4 ; +∞ [,
               for all y ⩾ x,
                x⩾2 and y⩾2 and  f(y) ⩾ f(x)
Proof
 let x ∈  [ 4 ; +∞ [
 let y ⩾ x
 ◆ x ⩾ 2
 ◆ y ⩾ 2
 ◆ y^2 - x^2 = (y-x)*(y+x)
 ◆ y + x ⩾ 0
 ◆ y - x ⩾ 0
 ◆ (y + x)*(y - x) ⩾ 0
 ◆ y^2 - x^2       ⩾ 0
 ◆ x^2 ⩽ y^2
 ◆ f(x) ⩽ f(y)
□
```

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

Prove $\forall x \in [2;+\infty[, x^2 \in [3;+\infty[$

```
Exercise "f is increasing on [4;+∞["
 Conclusion: for all x ∈ [ 4 ; +∞ [,
             for all y ⩾ x,
              x⩾2 and y⩾2 and  f(y) ⩾ f(x)
Proof
 let x ∈  [ 4 ; +∞ [
 let y ⩾ x
 ◆ x ⩾ 2
 ◆ y ⩾ 2
 ◆ y^2 - x^2 = (y-x)*(y+x)
 ◆ y + x ⩾ 0
 ◆ y - x ⩾ 0
 ◆ (y + x)*(y - x) ⩾ 0
 ◆ y^2 - x^2       ⩾ 0
 ◆ x^2 ⩽ y^2
 ◆ f(x) ⩽ f(y)
□
```

```
let's define the function f
 from [2;+∞[ to [3;+∞[ that maps x to x^2
Exercise "f is increasing on [4;+∞["
 Conclusion: for all x ∈ [ 4 ; +∞ [,
              for all y ⩾ x,
               x⩾2 and y⩾2 and f(y) ⩾ f(x)
Proof
 let x ∈  [ 4 ; +∞ [
 let y ⩾ x
 ◆ x ⩾ 2
 ◆ y ⩾ 2
 ◆ y^2 - x^2 = (y-x)*(y+x)
 ◆ y + x ⩾ 0
 ◆ y - x ⩾ 0
 ◆ (y + x)*(y - x) ⩾ 0
 ◆ y^2 - x^2        ⩾ 0
 ◆ x^2 ⩽ y^2
 ◆ f(x) ⩽ f(y)
□
```

Prove $\forall x \in [2;+\infty[, x^2 \in [3;+\infty[$

Prove $x \in [2;+\infty[$

Prove $y \in [2;+\infty[$

Automation : example 3/3

```
let's define the function f
 from [2;+∞[ to [3;+∞[ that maps x to x^2
Exercise "f is increasing on [4;+∞["
 Conclusion: for all x ∈ [ 4 ; +∞ [,
               for all y ⩾ x,
                x⩾2 and y⩾2 and f(y) ⩾ f(x)
Proof
 let x ∈  [ 4 ; +∞ [
 let y ⩾ x
 ◆ x ⩾ 2
 ◆ y ⩾ 2
 ◆ y^2 - x^2 = (y-x)*(y+x)
 ◆ y + x ⩾ 0
 ◆ y - x ⩾ 0
 ◆ (y + x)*(y - x) ⩾ 0
 ◆ y^2 - x^2         ⩾ 0
 ◆ x^2 ⩽ y^2
 ◆ f(x) ⩽ f(y)
□
```

Prove $\forall x \in [2; +\infty[, x^2 \in [3; +\infty[$

Prove $x \in [2; +\infty[$

Prove $y \in [2; +\infty[$

Prove $y \in [2; +\infty[$

Prove $x \in [2; +\infty[$

```
Theorem nn1even' "7. Using (... or ...) : proof by cases"
    Assumptions: (for all integer n, n is even or n is odd)
    Conclusion: for all integer n, n*(n+1) is even
Proof
  let n ∈ ℤ
  ◆ n is even or n is odd
  ?
■
```

1 goal

```
assumption1 : ∀ {n : Number}, if (n is integer
n : Number
assumption2 : n is integer
assumption3 : n is even or n is odd
⊢ n * (n + 1) is even
```

1

Use n is even or n is odd

▼ Messages (1)

▼ 01_HighSchoolProgression.lean:560:2

"Proof unfinished."

Widgets

```
Theorem nn1even' "7. Using (... or ...) : proof by cases"
    Assumptions: (for all integer n, n is even or n is odd)
    Conclusion: for all integer n, n*(n+1) is even
Proof
  let n ∈ ℤ
  ◆ n is even or n is odd
  ◆ if n is even then n * (n + 1) is even
    proof
      assume n is even
      ?
    ■
  ◆ if n is odd then n * (n + 1) is even
    proof
      assume n is odd
      ?
    ■
  ◆ n * (n + 1) is even by cases
■
```

```
1 goal
assumption1 : ∀ {n : Number}, if (n is integer
n : Number
assumption2 : n is integer
assumption3 : n is even or n is odd
⊢ n * (n + 1) is even
```

1

Use n is even or n is odd

▼ Messages (1)

▼ 01_HighSchoolProgression.lean:560:2

Conclusion

Yalep, geared towards high-school students, features:

- A minimal language declarative language based on forward chaining
- Enough automation to allow most basic proof steps left unjustified
- A mechanism to hide type foundations of the underlying P.A., namely:
  - Transparent handling of user number sets $\mathbb{N}$,$\mathbb{Z}$,$\mathbb{Q}$,$\mathbb{R}$.
  - Transparent manipulation of type-partial functions

Try it!
Web interface:

Git:

Future work

- Automation : rationalize, optimize, limit, configure
- Feedback (pretty printer, error messages)
- Cover high school curriculum (including Première, Terminale)

Demo

Questions?

Thank you !

Bartzia, Evmorfia, Antoine Meyer, and Julien Narboux (Oct. 2022). “Proof assistants for undergraduate mathematics and computer science education: elements of a priori analysis”. In: *INDRUM 2022: Fourth conference of the International Network for Didactic Research in University Mathematics*. Ed. by María Trigueros. Reinhard Hochmuth. Hanovre, Germany, pp. 253–262.

Bertot, Yves and Thomas Portet (Jan. 2025). “Chassez le naturel dans la formalisation des mathématiques”. In: *36es Journées Francophones des Langages Applicatifs (JFLA 2025)*. Roiffé, France.

Carl, Merlin, Hinrich Lorenzen, and Michael Schmitz (Feb. 2022). “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.

Coen, Claudio Sacerdoti and Enrico Zoli (2007). “A Note on Formalising Undefined Terms in Real Analysis”. In: *Proceedings of International Workshop on Proof Assistants and Types in Education (PATE07)*.

Kerjean, Marie et al. (Aug. 2022). “Utilisation Des Assistants de Preuves Pour l'enseignement En L1 - Retours d'expériences”. In: *Gazette Société Mathématique de France*.

Koepke, Peter (2019). “Textbook Mathematics in the Naproche-SAD System”. In: *Doctoral Program and Work in Progress at the Conference on Intelligent Computer Mathematics*.

Massot, Patrick (2024). “Teaching Mathematics Using Lean and Controlled Natural Language”. In: *15th International Conference on Interactive Theorem Proving (ITP 2024)*. Ed. by Yves Bertot, Temur Kutsia, and Michael Norrish. Vol. 309. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 27:1–27:19.

Nawrocki, Wojciech, Edward W. Ayers, and Gabriel Ebner (2023). “An Extensible User Interface for Lean 4”. In: *14th International Conference on Interactive Theorem Proving (ITP 2023)*. Ed. by Adam Naumowicz and René Thiemann. Vol. 268. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 24:1–24:20.

Stojanovic, Sana et al. (July 2014). “A Vernacular for Coherent Logic”. In: *Lecture Notes in Computer Science*. Vol. 8543. Lecture Notes in Computer Science. Coimbra, Portugal: Springer, p. 16.

The Mathlib Community (Jan. 2020). “The lean mathematical library”. en. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. New Orleans LA USA: ACM, pp. 367–381.

Tran Minh, Frédéric, Laure Gonnord, and Julien Narboux (2025). “Proof Assistants for Teaching: a Survey”. In: *Post Proceedings of ThEdu24*. Vol. 419. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, pp. 1–27.

Wemmenhove, Jelle et al. (Apr. 2024). “Waterproof: Educational Software for Learning How to Write Mathematical Proofs”. In: *Electronic Proceedings in Theoretical Computer Science* 400, 96–119.