Yalep: from high-school to competitive exams

Frédéric Tran-Minh & Sébastien Lahaye

Grenoble INP - UGA & IRIF, CNRS, PiCube, Malinca, Université Paris Cité

LiberAbaci Meeting, IRIF, Paris, April 2026

Yalep: Yet Another Learning Environment for Proof[1] Frédéric Tran Minh under the supervision of Laure Gonnord and Julien Narboux

A proof assistant in high-school experiment lead by Zoé Mesnil, Frédéric Tran Minh, Iro Bartzia and Julien Narboux

A propositional logic course that is both readable and mechanically verified Sébastien Lahaye under the supervision of Pierre Letouzey and Julien Narboux

[1] Frédéric Tran Minh, Laure Gonnord, and Julien Narboux (2026). "A Lean-based Language for Teaching Proof in High School". In: *Intelligent Computer Mathematics*. Ed. by Valeria de Paiva and Peter Koepke. Cham: Springer Nature Switzerland, pp. 447–467

Goals

- teach the concept of proof to high-school students and undergraduates
- ... making them write their own proofs in a proof assistant
- ... thus provide a language readable and writable by this audience

Yalep : Yet Another Learning Environment for Proof

Yalep main features:

- Syntactic layer above Lean
- Intentionally restricted vocabulary with a ”natural” flavor[2]
- Automation enabling acceptable proof granularity
- Custom implementation of ”unique number type”[3]
- Formalization of ’type-partial’ functions

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

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

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

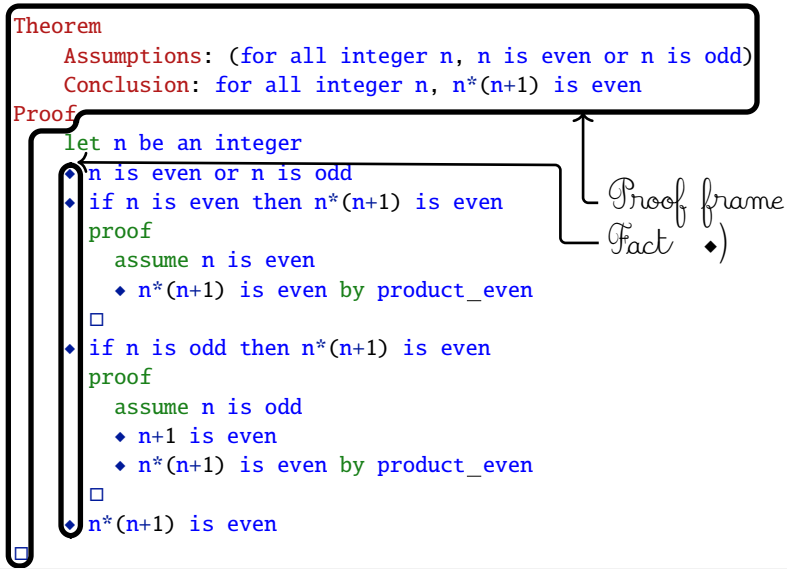
```
Theorem
    Assumptions: (for all integer n, n is even or n is odd)
    Conclusion: for all integer n, n*(n+1) is even
Proof
    let n be an integer
    ◆ n is even or n is odd
    ◆ if n is even then n*(n+1) is even
      proof
        assume n is even
        ◆ n*(n+1) is even by product_even
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



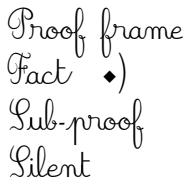
”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem
    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

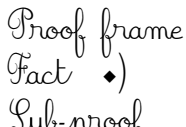
”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem
    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
□
```



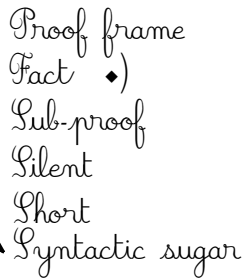
”Yet Another Learning Environment for Proof” (YALEP) - A first taste

```
Theorem
    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
    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
    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

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 make their goals explicit
- Relaxes constraints 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

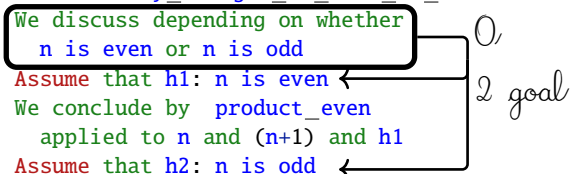
Forward chaining

```
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 make their goals explicit
- Relaxes constraints 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

```
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 make their goals explicit
- Relaxes constraints 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

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
```

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

```
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

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
```

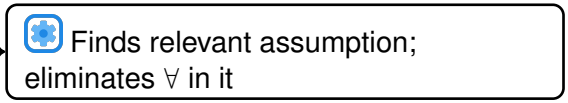
Hiding types : a proof of $\sqrt{2} \notin \mathbb{Q}$ in Yalep

```
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

 ◎ pˆ2 = (p/q)ˆ2 * qˆ2
   _   = (√2)ˆ2  * qˆ2
   _   =  2      * qˆ2
```

```
 ◆ pˆ2 is even
 ◆ p is even by n2_even_implies_n_even
 obtain an integer k such that p = 2*k
 ◎ qˆ2 = (2*qˆ2)/2
   _   = pˆ2 / 2
   _   = (2*k)ˆ2 /2
   _   = 2 * kˆ2
 ◆ qˆ2 is even
 ◆ q is even by n2_even_implies_n_even
 ◆ Absurd
□
```

```
Theorem
    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
□
```

```
Theorem
    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
□
```



```
Theorem
    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; eliminates ∀ in it

Applies specified lemma to relevant assumptions in context

```
Theorem
    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; eliminates $\forall$ in it

Applies specified lemma to relevant assumptions in context

The lemma gives $(n+1)n$ even. Unifies with $n(n+1)$ by commutativity.

```
Theorem
    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; eliminates $\forall$ in it

Applies specified lemma to relevant assumptions in context

The lemma gives $(n+1)n$ even. Unifies with $n(n+1)$ by commutativity.

Eliminates 'or' in relevant assm

4 experiments with Yalep

- 9th-graders (3ième) 2 hours session during internship at IRIF
- 10th-graders (Seconde), 6 hours during internship at IRIF
- First year (ESISAR, engineering school in Valence)
- **10th-graders (Seconde) in a high school in Paris**

A ”real-life” experiment with 10th-grade students

Pedagogical context:

- 10th-graders : 2 groups of 35 pupils; 1 group ”with Yalep” + 1 group ”without”
- Classes directed by their teacher (Zoé Mesnil)
- 4 x 55 minutes **including lectures about proof**, spread over 1 week

IT context:

- unmodifiable machines

Outputs:

- pre-test (student conception of proof)
- post-test (marked evaluation)

Sessions 1 and 2

- context, goal
- introduction of ”for all” and ”if then”
- introduction and elimination of ”there exists”

- for all integers n, if n is even then $n - 3$ is odd
- for all integers n, if n is odd then $n + 1$ is even

Sessions 3 and 4 : negation, contrapositive

- for all integers n, if n^2 is even then n is even
- for all integers n, if n is odd then n is not multiple of 4

Were planned but not worked out : case disjunction, irrationality of $\sqrt{2}$

Remarks about sessions

Pro:

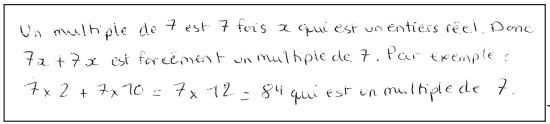
- interest of students in practical work
- most students try something, contrary to on paper
- gamification : some pupils wanted to complete the exercise
- possible because Yalep was available as a Web service

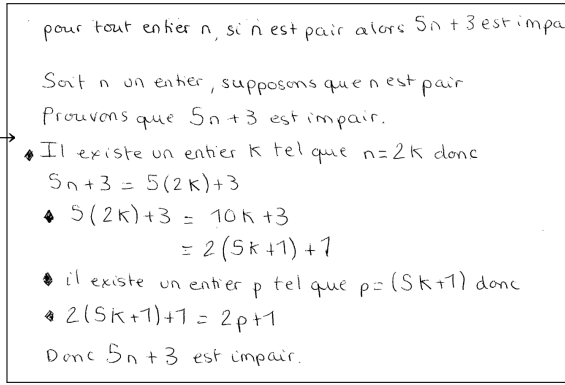
Cons:

- some technical difficulties with time and material constraints
- too short due to time constraints
- issues without textual input (eg: typing \ or ^ ... ; case sensitivity)
- heavy resources needed (Yalep/Lean runs on server)

Evolution:

- Proofs influenced by lexical tokens / vocabulary / syntax of Yalep
- Proofs are more formal ; introduction of forall and implies (let/assume)





Obstacles:

- existential quantifier:
  explicit elimination and introduction
- meaning of variables
- literal calculations

Yalep coverage

- Yalep designed for high-school pupils and undergraduates
- Tested in real conditions with parity of numbers
- Also covers : induction, functions, sequences, limits (as demo)
- Never designed to teach logic ...

- Starting a PhD about the use of proof assistants for teaching maths and computer science in *CPGE*.
- Starting with an easy chapter: propositional logic.
- Using Yalep, in order to have a low entry barrier.
- Pre-requisite: definition of data structures (algebraic datatypes), definition of functions using pattern matching and recursion

MPI/MP program

- Syntax of propositional logic
- Proofs by structural induction
- Semantics of propositional logic
- Meta-theorem of deduction
- Negative normal form, CNF and DNF
- Substitution
- Partial evaluation and indistinguishability lemma
- Natural deduction
- Quine's algorithm

Exercises and (part of) two competitive exams:

- ENS 2024
  Informatique C; MPI
- CCINP 2021
  Informatique; MP

Formalization choices

- The language of propositional logic is: V^0, F^0, Non^1, Et^2, Ou^2 (implication and equivalence are defined as derived connectives).
- Variables are represented as natural numbers (not as strings).
- NNF, CNF and DNF are defined as functions to an ad-hoc type (not a subset of propositional formulas).
- NNF transformation uses the dual transformation to ensure termination.
- Quine's algorithm is defined on propositional formulas (not on clauses).
- For natural deduction, context is a list of propositions not a set.

Experiment with competitive exams

CCINP 2021

Content

- Satisfiability of Horn clauses

Challenges

- Some implicit statements

ENS 2024

Content

- Satisfiability of a class of formulas
- Complexity

Challenges

- Graph theory
- Iterative algorithms

Example of student errors

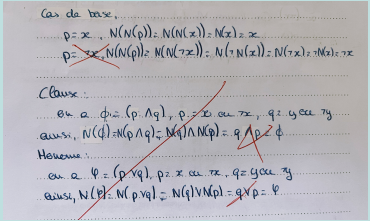
Exercise:

We consider the function N defined on propositional formulas by:

- $N(x) = x$
- $N(\neg f) = \neg N(f)$
- $N(f \wedge g) = N(g) \wedge N(f)$
- $N(f \vee g) = N(g) \vee N(f)$

Prove by structural induction that for all formula f, $N(N(f)) = f$.

Student answer:



Common mistakes:

- Confusion between variable denoting variable/formula.
- Confusion between structural induction on NNF/arbitrary formula.

Focus on induction in Yalep

- Formulas are defined as an inductive type, therefore many proofs are done by structural induction on formulas.
- The `induction` tactic helps to complete an induction by automatically generating the induction hypothesis and the case analysis.
- Yalep forces us to write the induction hypothesis explicitly, and to write each case of the proof separately.

Lean : efficient formalization but less explicit

```
def N f :=
    match f with
  | Var x => Var x
  | Non g => Non (N g)
  | Ou g d => Ou (N d) (N g)
  | Et g d => Et (N d) (N g)
  | _ => f
```

Implementation of N in Lean

```
theorem N_involutif' :
  N (N f) = f := by
  induction f; repeat simp [N]; repeat tauto
```

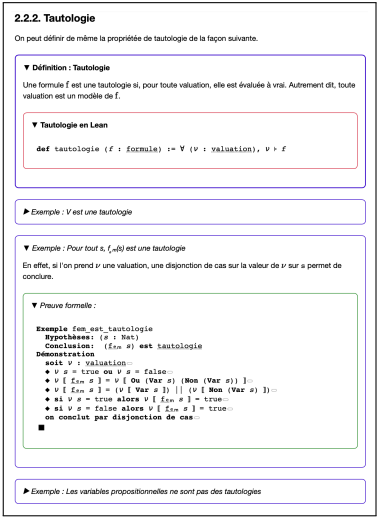
- Short proof using the `induction` tactic.
- Proof assistant silently generates the induction hypothesis.
- Solves many cases at once.

Yalep: closer to student reasoning

```
Exercice N_involutif
  Hypothèses: (f : formule)
  Conclusion: (N (N f) = f)
Démonstration
  posons P := fun f => N (N f) = f
  ◆ ∀ s, P (Var s)
  ◆ ∀ g, P g → P (Non g)
    preuve
      soit g : formule
      supposons P g
      ⊙ N (N (Non g)) = N (Non (N g))
        _             = Non (N (N g))
        _             = Non g
      ■
  ◆ ∀ g d, P g → P d → P (Ou g d)
    preuve
      soit g : formule
      soit d : formule
      supposons P g
      supposons P d
      ⊙ N (N (Ou g d)) = N (Ou (N d) (N g))
        _              = Ou (N (N g)) (N (N d))
        _              = Ou g d
      ■
  ◆ ∀ g d, P g → P d → P (Et g d)
    preuve
      soit g : formule
      soit d : formule
      supposons P g
      supposons P d
      ⊙ N (N (Et g d)) = N (Et (N d) (N g))
        _              = Et (N (N g)) (N (N d))
        _              = Et g d
      ■
  ◆ ∀ f, P f par induction structurelle
■
```

- Long proof, where the word induction appears late.
- Have to write the induction hypothesis ourselves.
- We see how each case is solved.

An online course with Verso and Lean4Web



2.2.2. Tautologie

On peut définir de même la propriétée de tautologie de la façon suivante.

▼ **Définition : Tautologie**

Une formule f est une tautologie si, pour toute valuation, elle est évaluée à vrai. Autrement dit, toute valuation est un modèle de f.

▼ **Tautologie en Lean**

```
def tautologie (f : formule) := ∀ (ν : valuation), ν ⊢ f
```

▶ *Exemple : V est une tautologie*

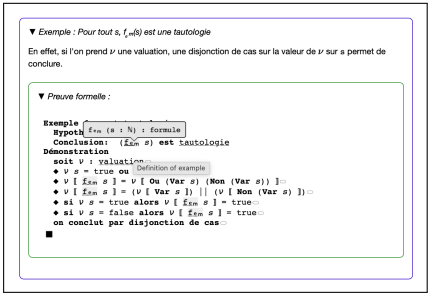
▼ *Exemple : Pour tout s, $f_{em}(s)$ est une tautologie*

En effet, si l'on prend ν une valuation, une disjonction de cas sur la valeur de ν sur s permet de conclure.

▼ *Preuve formelle :*

```
Exemple fem_est_tautologie
  Hypothèses: (s : Nat)
  Conclusion:  (f_em s) est tautologie
Démonstration
  soit ν : valuation
  ◆ ν s = true ou ν s = false
  ◆ ν [ f_em s ] = ν [ Ou (Var s) (Non (Var s)) ]
  ◆ ν [ f_em s ] = (ν [ Var s ]) || (ν [ Non (Var s) ])
  ◆ si ν s = true alors ν [ f_em s ] = true
  ◆ si ν s = false alors ν [ f_em s ] = true
  on conclut par disjonction de cas
  ■
```

▶ *Exemple : Les variables propositionnelles ne sont pas des tautologies*



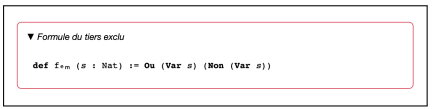
▼ *Exemple : Pour tout s, $f_{em}(s)$ est une tautologie*

En effet, si l'on prend ν une valuation, une disjonction de cas sur la valeur de ν sur s permet de conclure.

▼ *Preuve formelle :*

```
Exemple
  Hypoth  f_em (s : ℕ) : formule
  Conclusion:  (f_em s) est tautologie
Démonstration
  soit ν : valuation
  ◆ ν s = true ou
  ◆ ν [ f_em s ] = ν [ Ou (Var s) (Non (Var s)) ]
  ◆ ν [ f_em s ] = (ν [ Var s ]) || (ν [ Non (Var s) ])
  ◆ si ν s = true alors ν [ f_em s ] = true
  ◆ si ν s = false alors ν [ f_em s ] = true
  on conclut par disjonction de cas
  ■
```

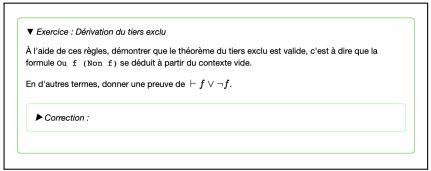
Definition of example

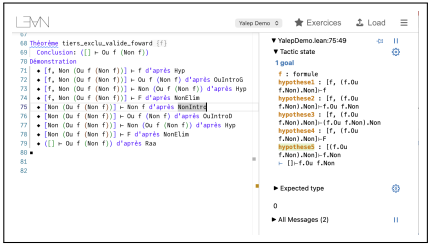


▼ *Formule du tiers exclu*

```
def f_em (s : Nat) := Ou (Var s) (Non (Var s))
```

An online course with Verso and Lean4Web





Experimental (Local)

Conclusion 1/2

Yalep:

- enabled high school pupils to practice proof in a highly constrained context
- could be made even more accessible

Future work:

- conduct a deeper analysis of experiment outputs
- improve, configure automation
- provide better feedback
- cover wider high school curriculum

Try it! `https://yalep.informatique.u-paris.fr/`

Git: `https://gricad-gitlab.univ-grenoble-alpes.fr/yalep/Yalep`

Conclusion 2/2

- Readable formalization of propositional logic (low entry barrier)
- Force students to make their goals explicit
- Interactive online material of the course

Future work:

- Introduce the tools to students
- Tackle more complex chapters of the mathematics curriculum

Online documentation :

`https://www.irif.fr/~lahaye/logique_propositionnelle/index.html`

References I

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.

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.

Tran Minh, Frédéric, Laure Gonnord, and Julien Narboux (2026). “A Lean-based Language for Teaching Proof in High School”. In: *Intelligent Computer Mathematics*. Ed. by Valeria de Paiva and Peter Koepke. Cham: Springer Nature Switzerland, pp. 447–467.

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.