Analyse de fonction en Rocq : une interface entre papier-crayon et preuves Rocq

Introduction

L'assistant de preuve Rocq

Coq Waterproof / Verbose Lean

```
Montrons que ∀ ε > 0, ∃ N, ∀ n ⩾ N, |f (u n) - f x₀| ⩽ ε
Soit ε > 0
Comme f est continue en x₀ et ε > 0 on obtient δ tel que
  (δ_pos : δ > 0) et (Hf : ∀ x, |x - x₀| ⩽ δ ⇒ |f x - f x₀| ⩽ ε)
...
```

Extrait d'une preuve en Verbose Lean ressemblant à une copie d'élève.

Objectifs

$f(x) = 5 \times x^2 - 1$

$f'(x) = 10 \times x$

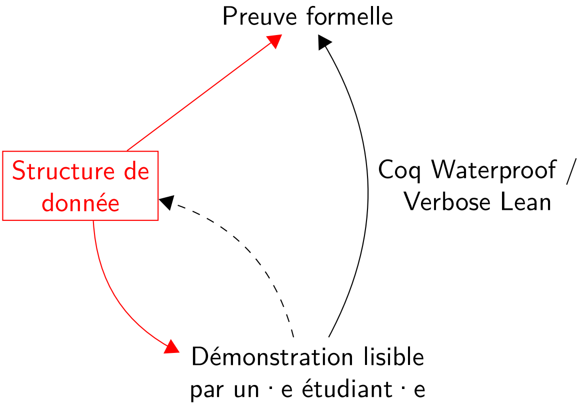
x	$-\infty$		0		$+\infty$
f'(x)	...?	-	0	+	...?
f(x)	?	↘	−1	↗	?

```
Require Import Tabvar.

Definition f (x : real) :=
  5x^2-1.
Definition f' (x : real) :=
  10x.
Assistant de preuve textuel
```

Prototype d'interface graphique entre tableau de variation et preuve Rocq

Contributions



Plan

Une première difficulté

```
Goal 2 + 2 = 4.
  compute.

  (* axiomes *)
  rewrite Rmult_plus_distr_l.
  rewrite Rmult_comm.
  rewrite Rmult_1_l.

  reflexivity.
Qed.
```

Une preuve utilisant les axiomes

Plus de difficultés

Énoncé

Déterminer la limite de la fonction $f\colon x \mapsto \dfrac{2 + 2\ln x}{x}$ en 0.

On sait que $\lim\limits_{x\to 0^+} \ln x = -\infty$ donc par limite d'un produit et d'une somme, $\lim\limits_{x\to 0^+} 2 + 2\ln x = -\infty$.
Comme par ailleurs $\lim\limits_{x\to 0^+} x = 0^+$, alors par limite d'un quotient on a $\lim\limits_{x\to 0^+} f(x) = -\infty$.

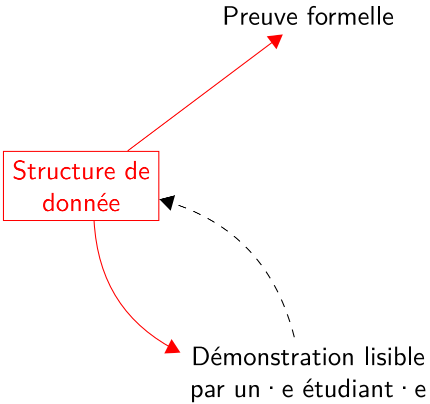
Une démonstration lisible

```
Definition f (x : R) : R := (2 + 2 * ln x) / x.

Lemma filterlim_f_0 : filterlim f (at_right 0)
(Rbar_locally m_infty).
Proof.
  unfold f.
  eapply (filterlim_comp_2 _ _ Rmult).
  eapply filterlim_comp_2.
  apply filterlim_const.
  eapply filterlim_comp_2.
  apply filterlim_const.
  by apply is_lim_ln_0.
  apply (filterlim_Rbar_mult 2 m_infty m_infty).
  unfold is_Rbar_mult, Rbar_mult'.
  case: Rle_dec (Rlt_le _ _ Rlt_0_2) => // H _ ;
  case: Rle_lt_or_eq_dec (Rlt_not_eq _ _ Rlt_0_2) => //.
  apply (filterlim_Rbar_plus 2 _ m_infty).
  by [].
  by apply filterlim_Rinv_0_right.
  by apply (filterlim_Rbar_mult m_infty p_infty).
Qed.
```

Une preuve illisible

Rappel de l'objectif



Une première approche

$$
\begin{aligned}
t &::= x \mid t_1 \;\; t_2 \mid \lambda(x : t_1).t_2 \mid \cdots \mid \texttt{thm} \\
\texttt{thm} &::= \forall P, Q, P \wedge Q \Rightarrow P \mid \cdots
\end{aligned}
$$

Preuve "en avant" vs "en arrière"

Énoncé

Déterminer la limite de la fonction $f\colon x \mapsto \dfrac{2 + 2\ln x}{x}$ en 0.

En avant :

Comme $\lim\limits_{x\to 0^+} \ln x = -\infty$ alors $\lim\limits_{x\to 0^+} 2 + 2\ln x = -\infty$ $\cdots$

En arrière :

Puisque f est un quotient, alors $\cdots$

Structure avec annotations

$$t ::= x \mid t_1 \updownarrow t_2 \mid \lambda(x : t_1).t_2 \mid \cdots \mid \texttt{thm}$$
$$\updownarrow ::= \uparrow \mid \downarrow$$

Structure avec annotations

$$t ::= x \mid t_1 \updownarrow t_2 \mid \lambda(x : t_1).t_2 \mid \cdots \mid \texttt{thm}$$

$$\updownarrow ::= \uparrow \mid \downarrow$$

Structure avec annotations

$$t ::= x \mid t_1 \updownarrow t_2 \mid \lambda(x : t_1).t_2 \mid \cdots \mid \mathtt{thm}$$
$$\updownarrow ::= \uparrow \mid \downarrow$$

"Soit ..." vs "Supposons que ..."

Type inductif

```
Inductive Construction : Type :=
| ...
```

Conclusion :

- par calcul
- par hypothèse
- par abandon

Avancer dans le but :

- introduction de variable
- introduction d'hypothèse
- application d'un théorème
- affirmation d'un fait

```
Inductive Arbre : Type :=
| racine : forall {but : Type}, but -> Construction -> Arbre.
```

Construction `introduction_variable`

```
| introduction_variable : Construction -> Construction
```

Tactique en Rocq :

`intro`

Traduction lisible :

Soit ...

Construction application

```
| application_nomme : string -> list Thm_arg -> Construction
```

```
Inductive Thm_arg : Type :=
  | forward : forall {enonce : Type}, enonce -> Thm_arg
  | backward : forall {enonce : Type}, enonce -> Construction -> Thm_arg
```

Tactique en Rocq :

`apply`

Traduction lisible :

Par <nom du théorème> en utilisant que <arguments "en avant"> il suffit de montrer que <arguments "en arrière">.

Exemple concret

Une preuve de la proposition `forall P Q : Prop, P /\ Q -> Q /\ P`

Soient `P`, `Q` des propositions.
Supposons `(P /\ Q)`.
On affirme que `Q`. Montrons le.
Comme `(forall P Q : Prop, P /\ Q -> Q)` et `(P /\ Q)`, on conclut que `Q`.
Par conjonction en utilisant que `Q`, il suffit de montrer que `P`.
Montrons que `P`.
Comme `(forall P Q : Prop, P /\ Q -> P)` et `(P /\ Q)`, on conclut que `P`.

Démonstration générée automatiquement

Deux tactiques automatisées

`solve_equ`

buts calculatoires

`solve_trivially`

théorèmes anonymes

Deux tactiques automatisées

`solve_equ`

buts calculatoires

`solve_trivially`

théorèmes anonymes

Soient `P`, `Q` des propositions.
Supposons `(P /\ Q)`.
On affirme que `Q`. Montrons le.
Comme `(forall P Q : Prop, P /\ Q -> Q)` et `(P /\ Q)`, on conclut que `Q`.
Par conjonction en utilisant que `Q`, il suffit de montrer que `P`.
Montrons que `P`.
Comme `(forall P Q : Prop, P /\ Q -> P)` et `(P /\ Q)`, on conclut que `P`.

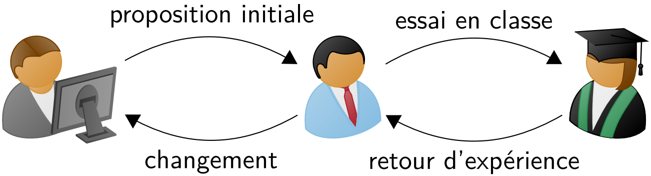
Utilisation de théorèmes anonymes

Liste des théorèmes disponibles

Soient `P`, `Q` des propositions.
Supposons `(P /\ Q)`.
On affirme que `Q`. Montrons le.
Comme `(forall P Q : Prop, P /\ Q -> Q)` et `(P /\ Q)`, on conclut que `Q`.
Par conjonction en utilisant que `Q`, il suffit de montrer que `P`.
Montrons que `P`.
Comme `(forall P Q : Prop, P /\ Q -> P)` et `(P /\ Q)`, on conclut que `P`.

Utilisation d'un théorème nommé

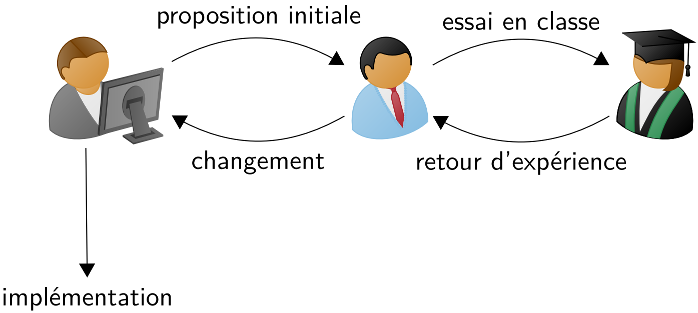
Une méthode d'évaluation



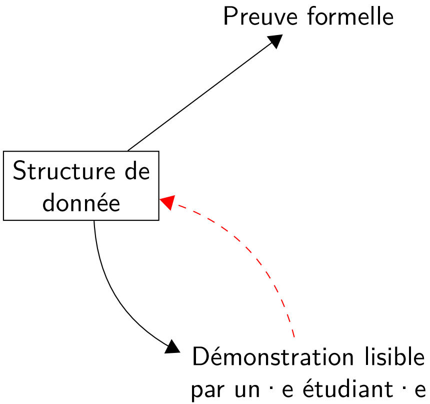
Constructions manquantes

- disjonction de cas
- raisonnement par l'absurde
- raisonnement par récurrence/induction
- autres types de raisonnement

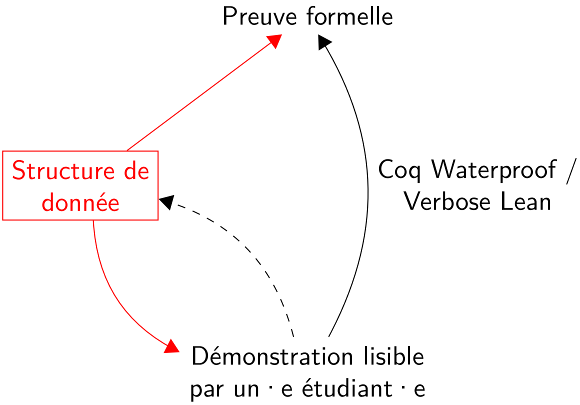
Évaluation des besoins et implémentation



Tactiques de construction



Conclusion



Preuve formelle
Structure de donnée
Coq Waterproof / Verbose Lean
Démonstration lisible par un · e étudiant · e

Section 5

Annexes

Bibliographie I

Coq Waterproof / Verbose Lean

```
Exercice "La continuité implique la
continuité séquentielle."
  Données : (f : ℝ → ℝ) (u : ℕ → ℝ)
(x₀ : ℝ)
  Hypothèses : (hu : u tend vers x₀) (hf
: f est continue en x₀)
  Conclusion : f ∘ u tend vers f x₀
Démonstration :
Montrons que ∀ ε > 0, ∃ N, ∀ n ⩾ N, |f
(u n) - f x₀| ⩽ ε
Soit ε > 0
Comme f est continue en x₀ et ε > 0 on
obtient δ tel que
  (δ_pos : δ > 0) et (Hf : ∀ x, |x - x₀|
⩽ δ ⇒ |f x - f x₀| ⩽ ε)
Comme u tend vers x₀ et δ > 0 on obtient
N tel que Hu : ∀ n ⩾ N, |u n - x₀| ⩽ δ
...
QED
```

Verbose Lean

```
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

Formalisation de ℝ en Rocq

Théorème

Tout corps ordonné, archimédien et complet est isomorphe à $\mathbb{R}$.

```
Axiom Rplus_comm : forall r1 r2:R, r1 + r2 = r2 + r1.
Axiom Rplus_assoc : forall r1 r2 r3:R, r1 + r2 + r3 = r1 + (r2 + r3).
Axiom Rplus_opp_r : forall r:R, r + - r = 0.
Axiom Rplus_0_l : forall r:R, 0 + r = r.

Axiom Rmult_comm : forall r1 r2:R, r1 * r2 = r2 * r1.
Axiom Rmult_assoc : forall r1 r2 r3:R, r1 * r2 * r3 = r1 * (r2 * r3).
Axiom Rinv_l : forall r:R, r <> 0 -> / r * r = 1.
Axiom Rmult_1_l : forall r:R, 1 * r = r.
Axiom R1_neq_R0 : 1 <> 0.

Axiom Rmult_plus_distr_l : forall r1 r2 r3:R, r1 * (r2 + r3) = r1 * r2 + r1 * r3.

Axiom total_order_T : forall r1 r2:R, {r1 < r2} + {r1 = r2} + {r1 > r2}.
Axiom Rlt_asym : forall r1 r2:R, r1 < r2 -> ~ r2 < r1.
Axiom Rlt_trans : forall r1 r2 r3:R, r1 < r2 -> r2 < r3 -> r1 < r3.
Axiom Rplus_lt_compat_l : forall r r1 r2:R, r1 < r2 -> r + r1 < r + r2.
Axiom Rmult_lt_compat_l : forall r r1 r2:R, 0 < r -> r1 < r2 -> r * r1 < r * r2.

Axiom archimed : forall r:R, IZR (up r) > r /\ IZR (up r) - r <= 1.

Axiom completeness : forall E:R -> Prop, bound E ->
  (exists x : R, E x) -> { m:R | is_lub E m }.
```

Inductif complet

```
Inductive Thm : Type :=
  | nomme : string -> Thm
  | anonyme : forall {enonce : Type}, enonce -> Thm
  | hyp : forall {enonce : Type}, enonce -> Thm.

Inductive Construction : Type :=
  | trou : Construction
  | hypothese : Construction
  | calcul : Construction
  | introduction_variable : Construction -> Construction
  | introduction_hypothese : Construction -> Construction
  | application : Thm -> list Thm_arg -> Construction
  | assertion : forall {assertion : Type}, assertion -> Construction -> Construction ->
 Construction

with Thm_arg : Type :=
  | forward : forall {enonce : Type}, enonce -> Thm_arg
  | backward : forall {enonce : Type}, enonce -> Construction -> Thm_arg.

Inductive Arbre : Type :=
  | racine : forall {but : Type}, but -> Construction -> Arbre.
```

Arbre de l'exemple

Une preuve de la proposition `forall P Q : Prop, P /\ Q -> Q /\ P`

```
racine (forall P Q : Prop, P /\ Q -> Q /\ P) (
  introduction_variable (
    introduction_variable (
      introduction_hypothese (
        assertion
          (forall P Q : Prop, P /\ Q -> goal Q)
          (application
            (anonyme (forall P Q : Prop, P /\ Q -> Q))
            [ forward (forall P Q : Prop, P /\ Q -> goal (P /\ Q)) ])
          (application
            (nomme "conjonction")
            [ forward (forall P Q : Prop, P /\ Q -> goal Q);
              backward (forall P Q : Prop, P /\ Q -> goal P)
                (application
                  (anonyme (forall P Q : Prop, P /\ Q -> P))
                  [ forward (forall P Q : Prop, P /\ Q -> goal (P /\ Q)) ])])
      )
    )
  )
)
```

Preuve de l'exemple

Une preuve de la proposition `forall P Q : Prop, P /\ Q -> Q /\ P`

```
(fun (P Q : Prop) (H : P /\ Q) =>
  let H0 : forall P0 Q0 : Prop, P0 /\ Q0 -> goal Q0 := fun (P0 Q0 : Prop) (H0 :
  P0 /\ Q0) => proj2 H0  : goal Q0 in
  let H1 : forall Q0 : Prop, P /\ Q0 -> goal Q0 := H0 P in
  let H2 : P /\ Q -> goal Q := H1 Q in
  let H3 : goal Q := H2 H in
  let H4 : forall P0 Q0 : Prop, P0 /\ Q0 -> goal P0 := fun (P0 Q0 : Prop) (H4 :
  P0 /\ Q0) => proj1 H4  : goal P0 in
  let H5 : forall Q0 : Prop, P /\ Q0 -> goal P := H4 P in
  let H6 : P /\ Q -> goal P := H5 Q in
  let H7 : goal P := H6 H in
  conj H3 H7)
```

Preuve générée automatiquement