Logique

Coq

Thomas Pietrzak
Licence Informatique

Université de Lille
1 SCIENCES ET TECHNOLOGIES

Coq

http://coq.inria.fr/

Conçu en France par Thierry Coquand et Gérard Huet

λ-calcul typé : Gallina

Isomorphisme de Curry-Howard

Preuves avec coq

Entrée :

Commandes et tactiques

Buts

Messages
Sortie

Buts

Prolog	Séquent	Coq

`r :- p, q`	$P, Q \vdash R$	`p : P`
		`q : Q`
		`_____`
		`R`

Buts

p et q : preuves de P et Q

On peut voir la preuve de R avec : `Show Proof`

Coq

```
p : P
q : Q
______
R
```

Tactiques

exact

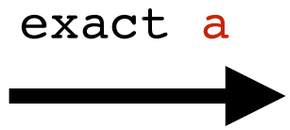
Axiome

$$\frac{}{\Gamma, A \vdash A}\ \mathrm{ax}$$

Utilise une preuve des hypothèses pour montrer un but

exact

```
A : Prop
a : A
_______(1/1)
A
```



```
exact a
No more subgoals.
```

intro / intros

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow_i$$

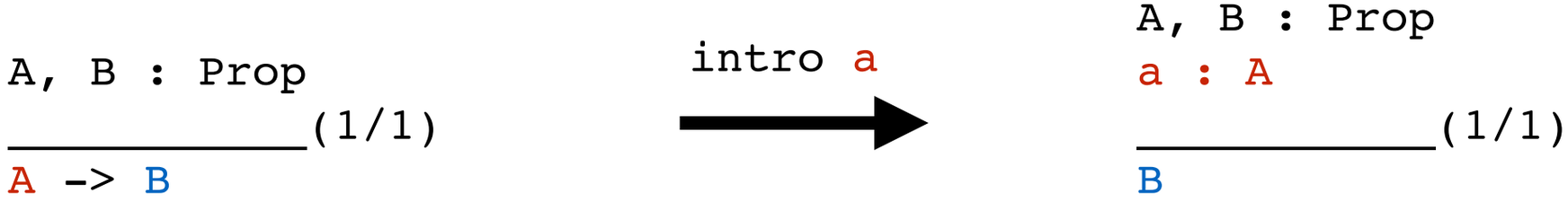
« Remonte » une hypothèse

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg_i$$

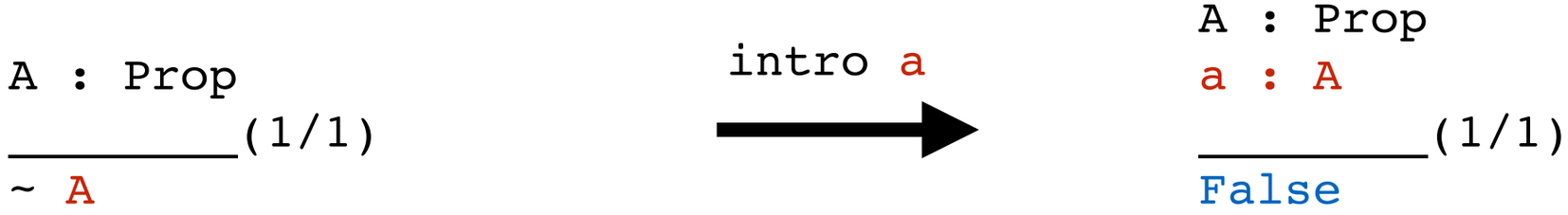
On obtient un objet représentant la preuve

$$\frac{\Gamma \vdash \varphi \qquad x \text{ n'est pas libre dans } \Gamma}{\Gamma \vdash \forall x.\varphi} \forall_i$$

intro / intros



intro / intros



intro / intros

```
X : Type
A : X -> Prop
_____________________(1/1)
forall x : X, A x
```

intro t

```
X : Type
A : X -> Prop
t : X
_____________________(1/1)
A t
```

```
Lemma ident (A : Prop) : A -> A.
Proof.
  intro a.
  assumption.
Show Proof.
Qed.
```

```
Lemma ident (A : Prop) : A -> A.
Proof.
  intro a.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A : Prop
_____________________________________(1/1)
A -> A
```

```
Lemma ident (A : Prop) : A -> A.
Proof.
  intro a.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A : Prop
a : A
_____________________________________(1/1)
A
```

```
Lemma ident (A : Prop) : A -> A.
Proof.
  intro a.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma ident (A : Prop) : A -> A.
Proof.
  intro a.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (A : Prop) (a : A) => a)
```

apply

$$\frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow_e$$

Utilise une hypothèse

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot} \neg_e$$

$$\frac{\Gamma \vdash \forall x.\varphi}{\Gamma \vdash \varphi[t/x]} \forall_e$$

$$\frac{\Gamma \vdash \psi \qquad \Gamma \vdash \varphi \Leftrightarrow \psi}{\Gamma \vdash \varphi} \Leftrightarrow_{eg} \quad \frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \varphi \Leftrightarrow \psi}{\Gamma \vdash \psi} \Leftrightarrow_{ed}$$

apply

```
A, B : Prop
ab : A -> B
_____________(1/1)
B
```

```
apply ab
```

```
A, B : Prop
ab : A -> B
_____________(1/1)
A
```

apply

```
A : Prop
a : ~ A
_________(1/1)
False
```

apply a

```
A : Prop
a : ~ A
_________(1/1)
A
```

apply

```
X : Type
A : X -> Prop
t : X
f : forall x : X, A x
_____________________(1/1)
A t
```

```
apply f
```

```
No more subgoals.
```

apply

```
A, B : Prop
ab : A <-> B
_____________(1/1)
B
```

apply ab →

```
A, B : Prop
ab : A <-> B
_____________(1/1)
A
```

```
A, B : Prop
ab : A <-> B
_____________(1/1)
A
```

apply <- ab →

```
A, B : Prop
ab : A <-> B
_____________(1/1)
B
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
_____________________________________(1/1)
(A -> B) -> A -> B
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
ab : A -> B
a : A
_____________________________________(1/1)
B
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
ab : A -> B
a : A
_____________________________________(1/1)
A
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma app (A B : Prop) :
  (A -> B) -> A -> B.
Proof.
  intros ab a.
  apply ab.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (A B : Prop) (ab : A -> B)
(a : A) => ab a)
```

split

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i$$

$$\frac{\Gamma, \psi \vdash \varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \Leftrightarrow \psi} \Leftrightarrow_i$$

Preuves par cas

Équivalent à `apply conj`

```
Inductive and (A B : Prop) : Prop :=
  conj : A -> B -> A /\ B

Definition iff (A B : Prop) :=
  (A -> B) /\ (B -> A).
```

split

```
A, B : Prop
_____________(1/1)
A /\ B
```

split

```
A, B : Prop
_____________(1/2)
A
_____________(2/2)
B
```

```
Inductive and (A B : Prop) : Prop :=
  conj : A -> B -> A /\ B
```

split

```
A, B : Prop
_____________(1/1)
A <-> B
```

split →

```
A, B : Prop
_____________(1/2)
A -> B
_____________(2/2)
B -> A
```

```
Definition iff (A B : Prop) :=
  (A -> B) /\ (B -> A).
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
forall A B:Prop, A -> B -> A /\ B
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
a : A
b : B
_____________________________________(1/1)
A /\ B
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
2 subgoals
A, B : Prop
a : A
b : B
_____________________________________(1/2)
A
_____________________________________(2/2)
B
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
a : A
b : B
_____________________________________(1/1)
A
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
This subproof is complete, but
there are some unfocused goals:

_____________________________________(1/1)
B
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
a : A
b : B
_____________________________________(1/1)
B
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Theorem split: forall A B : Prop,
  A -> B -> A /\ B.
Proof.
  intros A B a b.
  split.
  + exact a.
  + exact b.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (A B : Prop) (a : A) (b : B)
=> conj a b)
```

destruct

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_{eg} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_{ed}$$

Décompose une preuve à partir d'un constructeur

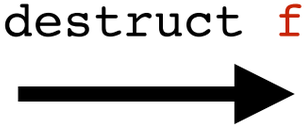
$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_e$$

$$\frac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \theta \qquad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}\ \vee_e$$

$$\frac{\Gamma \vdash \exists x.\varphi \qquad \Gamma, \varphi \vdash \psi \qquad x \text{ n'est libre ni dans } \Gamma \text{ ni dans } \psi}{\Gamma \vdash \psi}\ \exists_e$$

destruct

```
A : Prop
f : False
_____________(1/1)
A
```



```
destruct f
```

```
No more subgoals.
```

```
Inductive False : Prop :=
```

destruct

```
A, B, C : Prop
ab : A /\ B
______________(1/1)
C
```

destruct ab as [a b]

```
A, B, C : Prop
a : A
b : B
______________(1/1)
C
```

```
Inductive and (A B : Prop) : Prop :=
  conj : A -> B -> A /\ B
```

destruct

```
A, B, C : Prop
ab : A \/ B
_____________________(1/1)
C
```

destruct ab as [a|b]

```
A, B, C : Prop
a : A
_____________________(1/2)
C
A, B, C : Prop
b : B
_____________________(2/2)
C
```

b dans la 2e preuve

```
Inductive or (A B : Prop) : Prop :=
    or_introl : A -> A \/ B
  | or_intror : B -> A \/ B
```

destruct

```
X : Type
A : X -> Prop
B : Prop
e : exists x : X, A x
_____________________________(1/1)
B
```

destruct e

```
X : Type
A : X -> Prop
B : Prop
x : X
H : A x
_____________________(1/1)
B
```

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
  ex_intro : forall x : A, P x -> exists x, P x
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
_____________________________________(1/1)
(A -> C) -> (B -> C) ->
A \/ B -> C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
ac : A -> C
bc : B -> C
ab : A \/ B
_____________________________________(1/1)
C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
2 subgoals
A, B, C : Prop
ac : A -> C
bc : B -> C
a : A
_____________________________________(1/2)
C
_____________________________________(2/2)
C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
ac : A -> C
bc : B -> C
a : A
_____________________________________(1/1)
C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
ac : A -> C
bc : B -> C
a : A
_____________________________________(1/1)
A
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
This subproof is complete, but
there are some unfocused goals:

_____________________________________(1/1)
C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
ac : A -> C
bc : B -> C
b : B
_____________________________________(1/1)
C
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
1 subgoal
A, B, C : Prop
ac : A -> C
bc : B -> C
b : B
_____________________________________(1/1)
B
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma dest (A B C : Prop):
(A -> C) -> (B -> C) ->
(A \/ B) -> C.
Proof.
  intros ac bc ab.
  destruct ab as [a|b].
  + apply ac.
    exact a.
  + apply bc.
    exact b.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (A B C : Prop) (ac : A -> C)
   (bc : B -> C) (ab : A \/ B) =>
 match ab with
 | or_introl a => ac a
 | or_intror b => bc b
 end)
```

@

```
Inductive or (A B : Prop) : Prop :=
  or_introl : A -> A \/ B
| or_intror : B -> A \/ B.
```

Projections d'une disjonction

équivalent à `apply or_introl` ou `apply or_intror`

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \vee_{ig} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \vee_{id}$$

left / right

```
1 subgoal
A, B : Prop
_____________(1/1)
A \/ B
```

left →

```
1 subgoal
A, B : Prop
_____________(1/1)
A
```

```
1 subgoal
A, B : Prop
_____________(1/1)
A \/ B
```

right →

```
1 subgoal
A, B : Prop
_____________(1/1)
B
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
_____________________________________(1/1)
A -> A \/ B
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
a : A
_____________________________________(1/1)
A \/ B
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
1 subgoal
A, B : Prop
a : A
_____________________________________(1/1)
A
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma left (A B : Prop):
A -> (A \/ B).
Proof.
  intro a.
  left.
  exact a.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (A B : Prop) (a : A) =>
or_introl a)
```

exists

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
    ex_intro : forall x : A, P x -> exists x, P x
```

On prouve en explicitant la substitution

$$\frac{\Gamma \vdash \varphi[t/x]}{\Gamma \vdash \exists x.\varphi}\ \exists_i$$

exists

```
X : Type
t : X
P : X -> Prop
______________(1/1)
exists x : X, P x
```

exists t

```
X : Type
t : X
P : X -> Prop
______________(1/1)
P t
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
1 subgoal
X : Type
t : X
P : X -> Prop
_____________________________________(1/1)
P t -> exists x : X, P x
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
1 subgoal
X : Type
t : X
P : X -> Prop
pt : P t
_____________________________________(1/1)
exists x : X, P x
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
1 subgoal
X : Type
t : X
P : X -> Prop
pt : P t
_____________________________________(1/1)
P t
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma exst (X : Prop) (t : X)
(P : X -> Prop) :
P t -> exists x, P x.
Proof.
  intro pt.
  exists t.
  exact pt.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (X : Type) (t : X) (P : X ->
Prop) (pt : P t) =>
ex_intro (fun x : X => P x) t pt)
```

reflexivity

Axiome

$$\frac{}{\Gamma \vdash t = t}\ =_i$$

Un élément est toujours égal à lui-même

```
Inductive eq (A : Type) (x : A) : A -> Prop :=
    eq_refl : x = x
```

reflexivity

```
X : Type
x, y : X
_________(1/1)
x = x
```

reflexivity →

```
No more subgoals.
```

rewrite

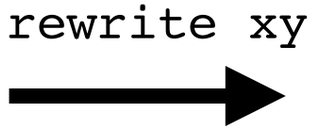
Utilise une égalité pour réécrire une partie du but

$$\frac{\Gamma \vdash \varphi[t/x] \qquad \Gamma \vdash t = u}{\Gamma \vdash \varphi[u/x]} =_e$$

rewrite

```
x, y : nat
xy : x = y
_____________(1/1)
S x = S y
```

```
rewrite xy
```



```
x, y : nat
xy : x = y
_____________(1/1)
S y = S y
```

```
x, y : nat
xy : x = y
_____________(1/1)
S x = S y
```

```
rewrite <- xy
```

```
x, y : nat
xy : x = y
_____________(1/1)
S x = S x
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
1 subgoal
x, y : nat
_____________________________________(1/1)
x = y -> S x = S y
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
1 subgoal
x, y : nat
xy : x = y
_____________________________________(1/1)
S x = S y
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
1 subgoal
x, y : nat
xy : x = y
_____________________________________(1/1)
S x = S y
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
Lemma rew (x y : nat) :
x = y -> S x = S y.
Proof.
  intro xy.
  rewrite <- xy.
  reflexivity.
Show Proof.
Qed.
```

```
No more subgoals.
```

```
(fun (x y : nat) (xy : x = y) =>
eq_ind x (fun y0 : nat => S x = S
y0) eq_refl y xy)
```

Preuves et programmes

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
1 subgoal
A : Type
_____________________________________(1/1)
A -> A
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
1 subgoal
A : Type
a : A
_____________________________________(1/1)
A
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
No more subgoals.
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
id = fun (A : Type) (a : A) => a
     : forall A : Type, A -> A
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
1 subgoal
A : Type
_____________________________________(1/1)
A -> A
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
Lemma id (A : Type) : A -> A.
Proof.
  intro a.
  exact a.
Qed.
Print id.

Lemma id2 (A : Type) : A -> A.
Proof fun a : A => a.
Print id2.
```

```
id2 = fun (A : Type) (a : A) => a
     : forall A : Type, A -> A
```

Définitions inductives

Type inductif

Types définis par cas

Chaque cas est une fonction : un constructeur

Constructeur : paramètres -> type

Constructeur sans paramètre : cas de base

```
Inductive nom : Type :=
  | … : …
  | … : …
  ….
```

Programmation

Un cas par constructeur

```
match … with
  | … => …
  | … => …
end.
```

Fonction récursive

« Point fixe »

Fixpoint permet les récurrences.

```
Fixpoint nom (…) … : … := ….
```

L'étape de récurrence doit décroitre.

Contre exemple

```
Fixpoint f (x : nat) : nat := f x
```

Contre exemple

```
Fixpoint f (x : nat) : nat := f x
```

```
Error:
Recursive definition of test is
ill-formed.
…
Recursive call to test has
principal argument equal to "x"
instead of
a subterm of « x ».
…
```

f(x) = f(x)

Si on autorise les récursions sur le même objet

```
Fixpoint f (x : nat) : False := f x
```

Type : `nat -> False`

Donc f 0 : `False`

On peut construire `False` !

Incohérence !

Entiers naturels

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.
```

```
Fixpoint add (n m : nat) : nat :=
  match n with
     | O => m
     | S n' => S (add n' m)
  end.
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
     | O => m
     | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  unfold add.
  reflexivity.
Qed.
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
forall n : nat, add O n = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________________________________(1/1)
add O n = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________________________________(1/1)
add O n = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________________________________(1/1)
add O n = n
```

Nouvelle tactique

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________________________________(1/1)
n = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0 : forall n : nat,
  add O n = n.
Proof.
  intro n.
  simpl.
  reflexivity.
Qed.
```

```
No more subgoals.
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
     | O => m
     | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
     | O => m
     | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
forall n : nat, add n O = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
_____________________________________(1/1)
add n O = n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
2 subgoals
_____________________________________(1/2)
add O O = O
_____________________________________(2/2)
add (S n) O = S n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
add O O = O
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
_____________________________________(1/1)
O = O
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
This subproof is complete, but
there are some unfocused goals:

_____________________________________(1/1)
add (S n) O = S n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
IHn : add n O = n
_____________________________________(1/1)
add (S n) O = S n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
IHn : add n O = n
_____________________________________(1/1)
S (add n O) = S n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
1 subgoal
n : nat
IHn : add n O = n
_____________________________________(1/1)
S n = S n
```

```
Inductive nat : Set :=
  O : nat
| S : nat -> nat.

Fixpoint add (n m : nat) : nat :=
  match n with
    | O => m
    | S n' => S (add n' m)
  end.

Theorem add0' : forall n : nat,
  add n O = n.
Proof.
intro n.
induction n.
- unfold add.
  reflexivity.
- simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

```
No more subgoals.
```

Listes

```
Inductive list : (A : Type) : Type :=
  nil : list A
| cons : A  -> list A -> list A.
```

```
Definition longueur (T : Type)(l : liste T) : nat :=
  match l with
    | nil _ => 0
    | cons _ _ l' => 1 + long T l'
  end.
```