

# Le Raisonnement Logique dans l'Assistant de Preuve Coq (draft)

Licence Info, Automates et Logiques

Reynald Affeldt

National Institute of Advanced Industrial Science and Technology

24 février 2016

# Objectif

- ▶ Expliquer les principales tactiques de CoQ [CDT16] à la lumière des règles de déduction naturelle (telles que présentées dans [DNR03])
- ▶ CoQ est une implémentation de la théorie des types, c'est un formalisme dans lequel la logique est facilement encodable
- ▶ Dans CoQ, les formules de logique correspondent aux types d'un langage de programmation ; une formule est vraie quand on peut trouver un terme de preuve qui a le bon type
- ▶ C'est un exemple d'application de l'isomorphisme de Curry-Howard
- ▶ Bien qu'il y ait beaucoup de tactiques dans CoQ (voir [CDT16, Chapitre 8]), peu sont essentielles

# Plan

## Un Aperçu de Coq

### La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

### La logique du premier ordre dans Coq

### Index des tactiques vues dans ce document

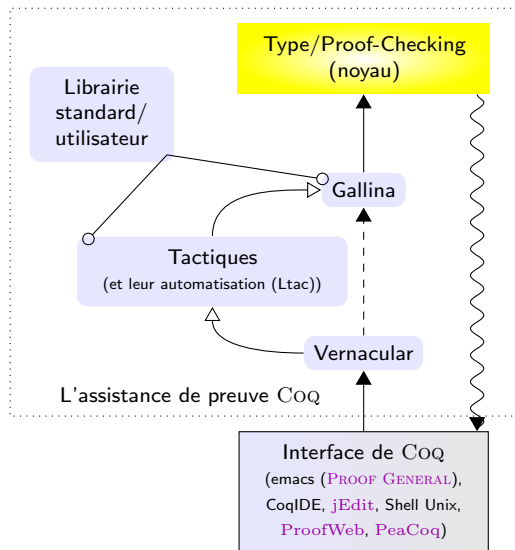
# L'Assistant de Preuve Coq

- ▶ Développé à l'INRIA depuis 1984 à l'initiative de T. Coquand et G. Huet
- ▶ Très utilisé
  - ▶ C'est essentiellement un langage de programmation fonctionnel typé
    - ▶ le Calcul des Constructions Inductives [CP90, PM92]
    - ▶ une extension du Calcul des Constructions [CH84, CH85, CH86, CH88]
    - ▶ TODO : aperç de la syntaxe et des règles de typage ?

Un formalisme qui joue le même rôle que la théorie des ensembles

- ▶ une majorité de publications dans la conférence de référence mais aussi beaucoup de publications dans les conférences sur les fondements de la programmation
- ▶ Prix : ACM SIGPLAN Programming Languages Software 2013 award, ACM Software System 2013 award
- ▶ Applications : le théorème des quatre couleurs [Gon05, Gon08], un compilateur C [Ler09, BL09], le théorème de l'ordre impair [GAA<sup>+</sup>13], etc.

# Organisation du Système



→ : Ajout de structures de données et de lemmes avec le langage de commandes Vernacular

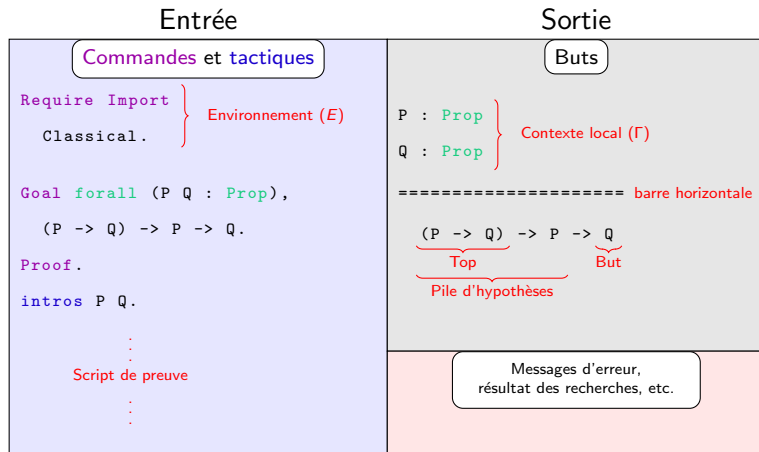
--► : Écriture directe de termes de preuve avec le langage typé Gallina

→► : En pratique, on écrit les termes de preuve indirectement avec les tactiques

—○ : La librairie standard de Coq propose des structures de données, des tactiques et des lemmes déjà prouvés

↪ : Messages d'informations ou d'erreurs quand le type/proof-checking échoue

# L'interface de Coq



# Lecture de la sortie de Coq comme un séquent

$p : P$   
▶  $q : Q$   
===== se lit comme  $P, Q \vdash R$   
 $R$

- ▶  $P, Q, R$  sont des propositions
- ▶ Les preuves  $(p, q)$  des hypothèses  $(P, Q)$  sont explicites
- ▶ La preuve (en cours de construction) de la conclusion  $R$  peut être visualisée avec la commande **Show Proof**

# Plan

Un Aperçu de Coq

La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

La logique du premier ordre dans Coq

Index des tactiques vues dans ce document



# Axiome

- ▶ « On peut prouver les propositions dont on dispose d'une preuve »
- ▶ Ce raisonnement est implémenté par la tactique **exact**
  - ▶ La tactique la plus basique puisqu'elle ne sert qu'à appeler le type-checker

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

A : Prop

a : A

=====

→**exact** a.→ No more subgoals.

A

# Affaiblissement

- ▶ Certaines hypothèses peuvent être inutiles
- ▶ Ce raisonnement est implémenté par la tactique `clear`

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A} \text{ aff}$$

A, B : Prop

a : A

b : B

=====

A

→ `clear` b. →

A, B : Prop

a : A

=====

A

# Introduction de l'implication ( $\rightarrow$ )

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_i$$

A, B : Prop

=====

A  $\rightarrow$  B

$\rightarrow$  **intros** a. <sup>1</sup> $\rightarrow$

$\leftarrow$  **revert** a. <sup>2</sup> $\leftarrow$

A, B : Prop

a : A

=====

B

- 
1. Variante limitée à un seul argument : **intro**
  2. **generalize** fait une copie

# Élimination de l'implication ( $\rightarrow$ )

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow_e$$

A, B : Prop

ab : A  $\rightarrow$  B

a : A

=====

B

$\rightarrow$ apply ab.<sup>3</sup> $\rightarrow$

A : Prop

ab : A  $\rightarrow$  B

a : A

=====

A

3. **cut** A. correspond davantage à  $\rightarrow_e$  mais est moins pratique

# Un exemple de preuve (1/7)

Écriture du but

$$\frac{}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C}$$

```
Lemma hilbertS (A B C : Prop) :  
  (A -> B -> C) -> (A -> B) -> A -> C.  
Proof.
```

# Un exemple de preuve (2/7)

## Introduction de la première implication

$$\frac{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i$$

```
Lemma hilbertS (A B C : Prop) :  
  (A -> B -> C) -> (A -> B) -> A -> C.  
Proof.  
intros abc.
```

# Un exemple de preuve (3/7)

## Introduction des trois implications

$$\frac{\frac{\frac{\Gamma^4 \vdash C}{A \rightarrow B \rightarrow C, A \rightarrow B \vdash A \rightarrow C} \rightarrow_i}{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i$$

**Lemma** hilbertS (A B C : Prop) :  
 (A -> B -> C) -> (A -> B) -> A -> C.

**Proof.**

**intros** abc ab a.

---

4.  $\Gamma = A \rightarrow B \rightarrow C, A \rightarrow B, A$

# Un exemple de preuve (4/7)

## Première élimination d'une implication

$$\frac{\frac{\frac{\frac{\frac{\Gamma \vdash A}{\Gamma^5 \vdash C} \rightarrow_e}{A \rightarrow B \rightarrow C, A \rightarrow B \vdash A \rightarrow C} \rightarrow_i}{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i$$

**Lemma** hilbertS (A B C : Prop) :  
 (A -> B -> C) -> (A -> B) -> A -> C.

**Proof.**

**intros** abc ab a.

**apply** abc.

---

5.  $\Gamma = A \rightarrow B \rightarrow C, A \rightarrow B, A$



# Un exemple de preuve (5/7)

Axiome

$$\frac{\frac{\frac{\frac{\overline{\Gamma \vdash A} \text{ ax}}{\Gamma \vdash C} \rightarrow_e}{\Gamma \vdash C} \rightarrow_i}{A \rightarrow B \rightarrow C, A \rightarrow B \vdash A \rightarrow C} \rightarrow_i}{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i$$

```
Lemma hilbertS (A B C : Prop) :  
  (A -> B -> C) -> (A -> B) -> A -> C.
```

```
Proof.
```

```
intros abc ab a.
```

```
apply abc.
```

```
- exact a.
```

---

6.  $\Gamma = A \rightarrow B \rightarrow C, A \rightarrow B, A$

# Un exemple de preuve (6/7)

## Deuxième élimination d'une implication

$$\frac{\frac{\frac{}{\Gamma \vdash A} \text{ax}}{\Gamma \vdash A} \quad \frac{\frac{\Gamma \vdash A}{\Gamma \vdash B} \rightarrow_e}{\Gamma \vdash C} \rightarrow_e}{\frac{A \rightarrow B \rightarrow C, A \rightarrow B \vdash A \rightarrow C}{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i} \rightarrow_i$$

```
Lemma hilbertS (A B C : Prop) :  
  (A -> B -> C) -> (A -> B) -> A -> C.
```

```
Proof.
```

```
intros abc ab a.
```

```
apply abc.
```

```
- exact a.
```

```
- apply ab.
```

---

7.  $\Gamma = A \rightarrow B \rightarrow C, A \rightarrow B, A$

# Un exemple de preuve (7/7)

Axiome et Qed

$$\frac{\frac{\frac{\Gamma \vdash A}{\Gamma \vdash A} \text{ ax} \quad \frac{\frac{\Gamma \vdash A}{\Gamma \vdash A} \text{ ax} \quad \frac{\Gamma \vdash B}{\Gamma \vdash B} \rightarrow_e}{\Gamma \vdash B} \rightarrow_e}{\Gamma \vdash C} \rightarrow_i \quad \frac{A \rightarrow B \rightarrow C, A \rightarrow B \vdash A \rightarrow C}{A \rightarrow B \rightarrow C \vdash (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i}{\vdash (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \rightarrow_i$$

```
Lemma hilbertS (A B C : Prop) :  
  (A -> B -> C) -> (A -> B) -> A -> C.
```

Proof.

```
intros abc ab a.
```

```
apply abc.
```

```
- exact a.
```

```
- apply ab.
```

```
  exact a.
```

Qed.

Qed. : “Quod Erat Demonstrandum”

8.  $\Gamma = A \rightarrow B \rightarrow C, A \rightarrow B, A$

# Un Exemple de Terme de Preuve

## L'axiome S de Hilbert

- Les tactiques ne sont qu'un moyen détourné pour écrire des termes du  $\lambda$ -calcul :

$$\begin{array}{c}
 \frac{}{\Gamma \vdash \text{abc} : A \rightarrow B \rightarrow C} \text{Var} \quad \frac{}{\Gamma \vdash \text{a} : A} \text{Var} \quad \frac{}{\Gamma \vdash \text{ab} : A \rightarrow B} \text{Var} \quad \frac{}{\Gamma \vdash \text{a} : A} \text{Var} \\
 \frac{}{\Gamma \vdash \text{abc a} : B \rightarrow C} \text{App} \quad \frac{}{\Gamma \vdash \text{ab a} : B} \text{App} \\
 \frac{}{\Gamma \vdash (\text{abc a}) (\text{ab a}) : C} \text{App} \\
 \frac{}{\text{abc} : A \rightarrow B \rightarrow C, \text{ab} : A \rightarrow B \vdash \lambda a : A. (\text{abc a}) (\text{ab a}) : A \rightarrow C} \text{Lam} \\
 \frac{}{\text{abc} : A \rightarrow B \rightarrow C \vdash \lambda \text{ab} : A \rightarrow B. \lambda a : A. (\text{abc a}) (\text{ab a}) : (A \rightarrow B) \rightarrow A \rightarrow C} \text{Lam} \\
 \frac{}{\vdash \lambda \text{abc} : A \rightarrow B \rightarrow C. \lambda \text{ab} : A \rightarrow B. \lambda a : A. (\text{abc a}) (\text{ab a}) : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C} \text{Lam}
 \end{array}$$

- Rappel : **Show Proof** . permet d'observer les termes de preuve.

# Plan

Un Aperçu de Coq

La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

La logique du premier ordre dans Coq

Index des tactiques vues dans ce document

# Introduction de la conjonction ( $\wedge$ )

- ▶ « Pour prouver  $A \wedge B$ , il suffit de prouver  $A$  et  $B$  »<sup>9</sup>
  - ▶ Création d'une branche dans un arbre de preuve
- ▶ Ce raisonnement est implémenté par la tactique `split`

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

$A, B : \text{Prop}$

$a : A$

$b : B$

=====

$A \wedge B$

$\rightarrow \text{split} \rightarrow$

$A, B : \text{Prop}$

$a : A$

$b : B$

=====

$A$

$A, B : \text{Prop}$

$a : A$

$b : B$

=====

$B$

---

9. oui, ça résonne comme une lapalissade

# Élimination de la conjonction ( $\wedge$ )

- ▶ « Pour prouver  $A$  (resp.  $B$ ), il suffit de prouver  $A \wedge B$  » (sic)
  - ▶ Idée : une conjonction peut être spécialisée
- ▶ Combinaison de la tactique **exact** et des lemmes `proj1/proj2`

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g \quad \left( \text{resp. } \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d \right)$$

```
A, B : Prop
ab : A /\ B
=====
A      →exact (proj1 ab) 10→ No more subgoals.
```

# Comment est définie la conjonction ( $\wedge$ ) ?

La conjonction est définie comme le type des paires de preuves :

$\text{Inductive } \overbrace{\text{and}}^{\text{type}} (\overbrace{A \ B}^{\text{paramètres}} : \text{Prop}) : \overbrace{\text{Prop}}^{\text{sorte}} :=$   
 $\text{constructeur } \rightarrow \text{conj} : A \rightarrow B \rightarrow \underbrace{A \ /\ B}_{\text{type du constructeur}}.$

- Soient A et B deux propositions, alors  $A \ /\ B$  est une proposition
- $A \ /\ B$  est une notation pour  $\text{and } A \ B$
- Soient a une preuve de A et b une preuve de B, alors  $\text{conj } a \ b$  est une preuve de  $A \ /\ B$



# Aspects techniques de l'utilisation de la conjonction ( $\wedge$ )

- La tactique `split` correspond à l'application du constructeur `conj` vu comme une fonction :

`split`  $\approx$  `apply conj`

- `proj1`/`proj2` sont deux lemmes standards :

**Theorem** `proj1` :  $A \wedge B \rightarrow A$ .

**Theorem** `proj2` :  $A \wedge B \rightarrow B$ .

- En pratique, plutôt que `proj1` et `proj2`, on préfère souvent utiliser la tactique `destruct` :

`A, B : Prop`

`ab : A  $\wedge$  B`

=====

`A`

$\rightarrow$  `destruct ab as [a b]`  $\rightarrow$

`A, B : Prop`

`a : A`

`b : B`

=====

`A`

# Exemples de termes de preuve avec la conjonction

L'introduction correspond à une application de fonction :

```
Lemma mysplit (A B : Prop) : A -> B -> A /\ B.  
Proof.  
exact (fun (a : A) (b : B) => conj a b).  
Qed.
```

**destruct** fait du pattern-matching :

```
Lemma myproj1 (A B : Prop) : A /\ B -> A.  
Proof.  
exact (fun ab : A /\ B =>  
  match ab with conj a b => a end).  
Qed.
```

# Introduction de la disjonction ( $\vee$ )

- ▶ « Pour prouver  $A \vee B$ , il suffit de prouver  $A$  ou  $B$  »
  - ▶ Idée : on utilise cette règle pour spécialiser un but
  - ▶ Comparez avec  $\wedge_e^g / \wedge_e^d$
- ▶ Ce raisonnement est implémenté par les tactiques `left` et `right`

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_i^g \quad \left( \text{resp. } \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_i^d \right)$$

$A, B : \text{Prop}$

$a : A$

=====

$A \ \backslash / \ B$

$\rightarrow \text{left}^{11} \rightarrow$

$A, B : \text{Prop}$

$a : A$

=====

$A$

# Élimination de la disjonction ( $\vee$ )

- ▶ Une disjonction permet de dichotomiser un raisonnement
  - ▶ Création d'une branche dans un arbre de preuve
  - ▶ Comparez avec  $\wedge_i$
- ▶ On utilise la tactique `destruct`

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee_e$$

$A, B, C : \text{Prop}$

$ab : A \ \backslash / \ B$

=====

$C$

$\rightarrow \text{destruct } ab \text{ as } [a|b] \rightarrow$

$A, B : \text{Prop}$

$a : A$

=====

$C$

$A, B : \text{Prop}$

$a : A$

=====

$C$

( $[a|b]$  et non  $[a \ b]$  comme dans le cas de la conjonction—[slide 25](#))

# Comme est définie la disjonction ?

La disjonction est définie comme un type avec deux constructeurs :

```
Inductive or (A B : Prop) : Prop :=  
  constructeur —▶ or_introl : A -> A \/\ B  
  | or_intror : B -> A \/\ B  
                                     type du constructeur
```

- ▶ Soient A et B deux propositions, alors  $A \vee B$  est une proposition
- ▶  $A \vee B$  est une notation pour `or A B`
- ▶ Soit a une preuve de A, alors `or_introl B a` est une preuve de  $A \vee B$
- ▶ Soit b une preuve de B, alors `or_intror A b` est une preuve de  $A \vee B$

# Aspects techniques de l'utilisation de la disjonction ( $\vee$ )

La tactique `left` correspond à l'application du construction `or_introl` vu comme une fonction :

$$\text{left} \approx \text{apply } \text{or\_introl}$$

La tactique `right` correspond à l'application du construction `or_intror` vu comme une fonction :

$$\text{right} \approx \text{apply } \text{or\_intror}$$

# Un exemple de terme de preuve avec la disjonction

## La règle d'élimination de la disjonction

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

ou bien `apply or_ind...`

```
Lemma or_elim_tactique (A B C : Prop) :  
  (A -> C) -> (B -> C) -> A \/ B -> C.  
Proof.  
intros ac bc.  
destruct 1 as [a | b].  
- apply ac.  
  exact a.  
- apply bc.  
  exact b.  
Qed.
```

En réalité, tous ces scripts produisent exactement la même preuve.

# Plan

Un Aperçu de Coq

La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

La logique du premier ordre dans Coq

Index des tactiques vues dans ce document



# Le faux ( $\perp$ ) et la négation ( $\neg$ ) dans Coq

- ▶ le faux (notation papier :  $\perp$ ) s'écrit `False` dans Coq
  - ▶ `False` n'est pas un objet de base, il est défini ([slide 37](#))
- ▶ la négation de  $A$  (notation papier :  $\neg A$ ) s'écrit  $\sim A$  (lire « tilde  $A$  ») dans Coq
  - ▶  $\sim$  n'est pas une connective de base, elle est définie :
    - ▶  $\neg A$  est défini (sur papier) comme  $A \rightarrow \perp$
    - ▶ dans Coq,  $A \rightarrow \perp$  devient  $A \rightarrow \text{False}$
    - ▶ on note  $\sim A$  pour  $A \rightarrow \text{False}$  pour simplifier la présentation

# Introduction et élimination de la négation ( $\neg$ )

Rappel du cours [Pie16] :

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg_i \quad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg_e$$

On remplace  $\neg A$  par sa définition  $A \rightarrow \perp$  dans les règles ci-dessus :

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash A \rightarrow \perp} \neg_i \quad \frac{\Gamma \vdash A \rightarrow \perp \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg_e$$

On s'aperçoit que les règles pour la négation sont une instance des règles pour l'implication (remplacer  $B$  par  $\perp$  ci-dessous) :

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow_i \quad \frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \rightarrow_e$$

On n'a donc pas à ajouter de tactiques dans Coq pour gérer la négation

# Introduction de la négation ( $\neg$ ) dans Coq

« pour prouver  $\neg A$ , il suffit de prouver  $\perp$  dans un contexte où  $A$  est vrai »

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

A : Prop

=====

~ A

→ intros a. →

A : Prop

a : A

=====

False

# Élimination de la négation ( $\neg$ ) dans Coq

« pour prouver  $\perp$ , il suffit de trouver  $A$  tel que  $A$  et  $\neg A$  soient vraies »

$$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg_e$$

```
A : Prop
na : ~ A
a : A      → apply na. 12→ No more subgoals.
=====
False
```

# Comment est défini faux ( $\perp$ ) dans Coq?

- ▶  $\perp$  est une proposition dont on ne peut construire de preuve
- ▶ dans Coq,  $\perp$  devient donc un type dont on ne peut construire les objets
- ▶ concrètement, False est un type inductif sans constructeur défini par la commande suivante :

```
Inductive Type False : Sort Prop := .
```

# La règle d'élimination du faux ( $\perp$ )

- ▶ en pratique, quand on a  $\perp$  dans le contexte, on « prouve » le but en l'éliminant
- ▶ *ex falso quodlibet* : à partir de faux, on prouve n'importe quoi

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp_e$$

```
A : Prop
abs : False
===== →destruct abs.→ No more subgoals.
A
```

# La logique classique dans Coq

- ▶ Coq est un formalisme *constructif* : on s'interdit par défaut les propriétés de la logique classique
  - ▶ exemple : le *tiers exclus* (« une proposition est nécessairement vraie ou bien fausse »)
- ▶ néanmoins, admettre la logique classique n'entraîne pas de contradiction
  - ▶ Coq fournit des caractérisations de la logique classique sous forme d'*axiomes* (des lemmes sans preuve)

Exemple de raisonnement classique « prouvable » :

« si à partir de  $\neg A$  on peut prouver  $\perp$ , alors  $A$  est vraie »

```
Require Import Classical.
```

```
Lemma bottom_c (A : Prop) : ((~A) -> False) -> A.
```

```
Proof. ... Qed.
```

# L'absurdité classique

On retrouve la règle de l'absurdité classique en utilisant le lemma `bottom_c` ([slide 39](#)) :

$$\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A} \perp_c$$

```
A : Prop
=====
A                                →  apply bottom_c.
                                   intros na.                                →
A : Prop
na : ~ A
=====
False
```



# Plan

Un Aperçu de Coq

La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

La logique du premier ordre dans Coq

Index des tactiques vues dans ce document

# Quantificateur Universel

$$\frac{\Gamma \vdash A \quad x \text{ n'est pas libre dans les formules de } \Gamma}{\Gamma \vdash \forall x. A} \forall_i$$

```
X : Type
A : X -> Prop
=====
forall x : X, A x
```

```
→intros x.→
←revert x. 13←
```

```
X : Type
A : X -> Prop
x : X
=====
A x
```

$$\frac{\Gamma \vdash \forall x. A}{\Gamma \vdash A[x := t]} \forall_e$$

```
X : Type
A : X -> Prop
Ax : forall x : X, A x
t : X
=====
A t
```

```
→apply Ax. 14→
```

No more  
subgoals.

13. `generalize` `x`. crée une nouvelle variable `x0`

14. Ou bien `exact` `(Ax t)`

# Quantificateur Existentiel

$$\frac{\Gamma \vdash A[x := t]}{\Gamma \vdash \exists x.A} \exists_i$$

```
X : Type
x : X
A : X -> Prop
t : X
At : A t
=====
exists x0 : X, A x0
```

→exists t→

```
X : Type
x : X
A : X -> Prop
t : X
At : A t
=====
A t
```

$\Gamma \vdash \exists x.A$

$\Gamma, A \vdash C$

$x$  n'est libre ni dans les formules de  $\Gamma$ , ni dans  $C$

$\Gamma \vdash C$

$\exists_e$


```
C : Prop
X : Type
x : X
A : X -> Prop
At : exists t : X,
      A t
=====
C
```

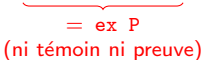
→destruct At as [t At]→

```
C : Prop
X : Type
x : X
A : X -> Prop
t : X
At : A t
=====
C
```

# Comme est défini le quantificateur existentiel ?

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

  
témoin      preuve

  
= ex P  
(ni témoin ni preuve)

```
exists t = constructor 1 with t = apply (ex_intro _ t)
```

# Égalité

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

```
X : Type
t : X
=====
t = t
```

→**reflexivity**→ No more subgoals.

$$\frac{\Gamma \vdash A[x := t] \quad \Gamma \vdash t = u}{\Gamma \vdash A[x := u]} =_e$$

```
X : Type
t, u : X
A : X -> Prop
At : A t
tu : t = u
=====
A u
```

→**rewrite** <-tu<sup>15</sup>→

```
X : Type
t, u : X
A : X -> Prop
At : A t
tu : t = u
=====
A t
```

15. Et dans l'autre sens : **rewrite** ->tu ou plus simplement **rewrite** tu

# Comment est définie l'égalité?

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

Diagram illustrating the structure of the `eq` inductive definition:

- `eq` is the inductive constructor.
- `A` is the family parameter (paramètre de famille).
- `x` is the index/predicate parameter (index/predicate parameter).
- `A` is the sort (Sorte).
- `eq` is the family (family).
- `x` is the predicate (predicate).
- `x` is the argument (argument).

`reflexivity = apply eq_refl`

# Que fait `rewrite` ?

## Exemple de principe d'induction

Au moment de la définition d'un type inductive, un principe d'induction est généré :

```
eq_ind : forall (A : Type) (x : A) (P : A -> Prop),  
  P x -> forall y : A, x = y -> P y
```

C'est essentiellement une fonction qui transforme  $P\ x$  en  $P\ y$

En fait, dans l'exemple précédent, `rewrite`  $\leftarrow$  tu est équivalent à :

```
apply (eq_ind _ _ At _ tu).
```

Au final, les tactiques qui comptent vraiment ici sont :

```
intros/revert, apply, destruct ... as [...] (, clear)
```

# Plan

Un Aperçu de Coq

La logique propositionnelle dans Coq

Conjonction et disjonction

Le faux ( $\perp$ ) et la négation ( $\neg$ )

La logique du premier ordre dans Coq

Index des tactiques vues dans ce document



# Tactiques utilisées dans ce document

`apply`, 12, 25, 30, 36, 40, 42, 44  
`clear`, 10  
`constructor`, 44  
`cut`, 12  
`destruct`, 25, 28, 38, 43  
`exact`, 9, 23, 36, 42  
`intros`, 35  
`exists`, 43, 44  
`generalize`, 11, 42  
`intros`, 11, 40, 42  
`left`, 27, 30  
`reflexivity`, 45  
`revert`, 11, 42  
`rewrite`, 45  
`right`, 27, 30  
`split`, 22, 25

# Bibliographie I



Sandrine Blazy and Xavier Leroy, *Mechanized semantics for the Clight subset of the C language*, J. Autom. Reasoning **43** (2009), no. 3, 263–288.



The Coq Development Team, *The Coq proof assistant reference manual*, INRIA, 2016, Version 8.5.



Thierry Coquand and Gérard Huet, *A theory of constructions (preliminary version)*, International Symposium on Semantics of Data Types, Sophia-Antipolis, 1984, Jun. 1984.



———, *Constructions : A higher order proof system for mechanizing mathematics*, Proceedings of the European Conference on Computer Algebra, Linz, Austria EUROCAL 85, April 1–3, 1985, Linz, Austria, vol. 1 (Invited Lectures), Apr. 1985, pp. 151–184.



———, *Concepts mathématiques et informatiques formalisés dans le calcul des constructions*, Tech. Report 515, INRIA Rocquencourt, Apr. 1986.



———, *The calculus of constructions*, Information and Computation **76** (1988), 95–120.



Thierry Coquand and Christine Paulin, *Inductively defined types*, Proceedings of the International Conference on Computer Logic (COLOG-88), Tallinn, USSR, December 1988, Lecture Notes in Computer Science, vol. 417, Springer, 1990, pp. 50–66.



René David, Karim Nour, and Christophe Raffalli, *Introduction à la logique*, 2ème ed., Dunod, 2003.



Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry, *A machine-checked proof of the odd order theorem*, Proceedings of the 4th International Conference on Interactive Theorem Proving, ITP 2013, Rennes, France, July 22–26, 2013, Lecture Notes in Computer Science, vol. 7998, Springer, 2013, pp. 163–179.

# Bibliographie II



Georges Gonthier, *A computer-checked proof of the four colour theorem*, Tech. report, Microsoft Research, Cambridge, 2005, Available at : <http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf>. Last access : 2014/08/04.



———, *Formal proof—the four-color theorem*, Notices of the American Mathematical Society **55** (2008), no. 11, 1382–1393.



Xavier Leroy, *A formally verified compiler back-end*, J. Autom. Reasoning **43** (2009), no. 4, 363–446.



Thomas Pietrzak, *Logique—logique propositionnelle—*, Licence Informatique, Université de Lille 1 Sciences et Technologies, 2016, <http://www.thomaspietrzak.com/download.php?f=CoursLogique0.pdf>.



Christine Paulin-Mohring, *Inductive definitions in the sytem coq rules and properties*, Tech. Report 92–49, LIP, École Normales Supérieure de Lyon, Dec. 1992.