

# Le Raisonnement Logique dans l'Assistant de Preuve Coq (draft in progress) L3, UE Logique

Reynald Affeldt

National Institute of Advanced Industrial Science and Technology

19 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 une 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

Logique Propositionnelle en Coq

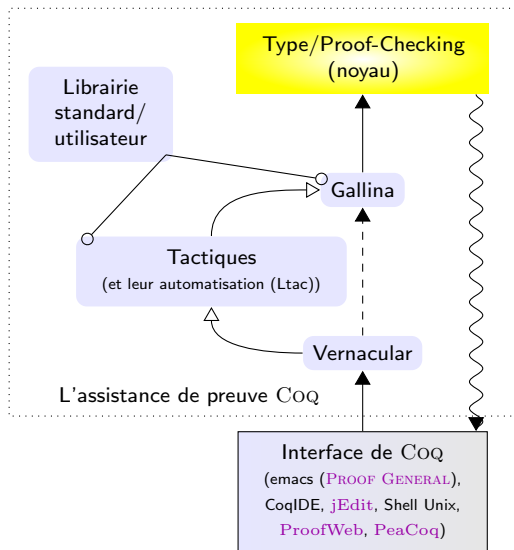
Logique du Premier Ordre en Coq

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
  - ▶ une majorité de publications dans la conférence de référence mais aussi dans les conférences sur les fondements de la programmation
  - ▶ Prix : ACM SIGPLAN Programming Languages Software 2013 award, ACM Software System 2013 award
- ▶ 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]
- ▶ 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

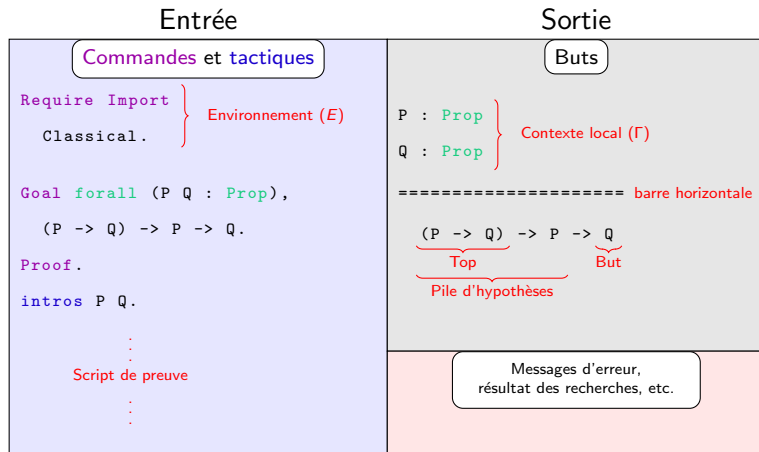
--► : Écriture directe de termes de preuve avec le langage typé Gallina

→► : 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



# Plan

Un Aperçu de COQ

Logique Propositionnelle en COQ

Logique du Premier Ordre en COQ

Index des tactiques vues dans ce document

# Axiome et affaiblissement

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

A : Prop

a : A

=====

A

→**exact** a.→ No more subgoals.

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

A, B : Prop

a : A

b : B

=====

A

→**clear** b.→

A, B : Prop

a : A

=====

A



# Implication

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

A, B : Prop  
=====

A -> B

→intros a. 1→  
←revert a. 2←

A, B : Prop  
a : A  
=====

B

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

A, B : Prop  
ab : A -> B  
a : A  
=====

B

→apply ab. 3→

A : Prop  
ab : A -> B  
a : A  
=====

A

1. Variante limitée à un seul argument : **intro**
2. **generalize** fait une copie
3. **cut** A. correspond davantage à  $\rightarrow_e$  mais est moins pratique

# Un Exemple de Preuve

## L'axiome S de Hilbert

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

```
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”

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

# 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 :

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

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

# Conjonction

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

A, B : Prop

a : A

b : B

=====

A /\ B

→split→

A, B : Prop

a : A

b : B

=====

A

A, B : Prop

a : A

b : B

=====

B

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d$$

A, B : Prop

ab : A /\ B

=====

A

→exact (proj1 ab)→ No more subgoals.

# Comment est définie la conjonction ?

$\wedge$ , proj1, proj2 sont définis dans Init/Logic.v.

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

$\text{split} = \text{constructor } 1 \approx \text{apply conj}$

**Theorem** proj1 :  $\text{A} \wedge \text{B} \rightarrow \text{A}.$

**Theorem** proj2 :  $\text{A} \wedge \text{B} \rightarrow \text{B}.$

Plutôt que d'utiliser proj1 at proj2, on préfère souvent :

$\text{A, B} : \text{Prop}$		$\text{A, B} : \text{Prop}$
$\text{ab} : \text{A} \wedge \text{B}$		$\text{a} : \text{A}$
=====	$\rightarrow \text{destruct ab as [a b]} \rightarrow$	$\text{b} : \text{B}$
$\text{A}$		=====
		$\text{A}$

# 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.
```

# Disjonction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_i^g \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_i^d$$

A, B : Prop

a : A

=====

A \ / B

→left<sup>5</sup>→

A, B : Prop

a : A

=====

A

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

A, B, C : Prop

ab : A \ / B

=====

C

→destruct ab as [a|b]→

A, B : Prop

a : A

=====

C

A, B : Prop

a : A

=====

C

## Comme est définie la disjonction ?

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

```
left = constructor 1 ≈ apply or_introl  
right = constructor 2 ≈ apply 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.

# Négation ( $\neg$ )

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

```
A : Prop
na : A -> False
=====
~ A
```

$\rightarrow$ exact na. $\rightarrow$  No more subgoals.

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

```
A : Prop
na : ~ A
a : A
=====
False
```

$\rightarrow$ exact (na a).<sup>6</sup> $\rightarrow$  No more subgoals.

# Comment est définie la négation ?

Définition de faux :

`Inductive`  $\overbrace{\text{False}}^{\text{Type}} : \overbrace{\text{Prop}}^{\text{Sort}} := .$

- ▶ Définition de la négation :
  - ▶  $\neg A$  ( $\sim A$ ) est défini comme  $A \rightarrow \perp$  ( $A \rightarrow \text{False}$ )
- ▶ L'élimination de la négation devient une instance de  $\rightarrow_e$
- ▶ On parle plutôt de l'élimination du faux (*ex falso quodlibet*) :

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

- ▶ L'absurdité classique  $\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A} \perp_c$  requiert un axiome

# Plan

Un Aperçu de COQ

Logique Propositionnelle en COQ

Logique du Premier Ordre en 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.7←
```

```
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.8→
```

No more  
subgoals.

7. **generalize** x. crée une nouvelle variable x0
8. 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
```

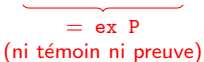
→destruct At as [t At]→

```
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>9</sup>→

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

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

Logique Propositionnelle en COQ

Logique du Premier Ordre en COQ

Index des tactiques vues dans ce document

# Tactiques utilisées dans ce document

`apply`, 9, 13, 16, 18, 21, 23

`clear`, 8

`constructor`, 13, 16, 23

`cut`, 9

`destruct`, 13, 15, 19, 22

`exact`, 8, 12, 18, 21

`exists`, 22, 23

`generalize`, 9, 21

`intros`, 9, 21

`left`, 15, 16

`reflexivity`, 24

`revert`, 9, 21

`rewrite`, 24

`right`, 15, 16

`split`, 12, 13

# 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.



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