# Coqshop

Lorenzo Pace, Alessio Duè

May 5, 2024

# Le coq

# What is coq?

Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs.

- Le coq website: `https://coq.inria.fr/`

# Some quotes about coq

- *"It takes years to master the coq."*
- *"Let's get our hands dirty with the... theorem prover."*
- *"And now I am become coq, the excluder of middles."*

## Friendly reminder

We are a bunch of civilized fellows, and *le coq* is a most serious tool used by respectable computer scientists all over France. There is nothing to laugh about.

# Some theory

Before starting with the coqshop, we introduce two concepts.

- Curry-Howard Correspondence.
- Calculus of Constructions.

# Curry-Howard Correspondence

$$\text{Computation} \longleftrightarrow \text{Logic}$$

## Constructive logic

Logic **without** axioms and rules that allow to perform proof by contradiction, e.g.:

$$\overset{\textit{Premises}}{\emptyset} \vdash \overset{\textit{Conclusions}}{P \vee \neg P} \qquad \text{(Excluded Middle)}$$

$$\neg\neg P \vdash P \qquad \text{(Double neg. elim.)}$$

# Curry-Howard Correspondence

## Computation

The simply-typed $\lambda$ calculus is a model of computation:

$$
\begin{aligned}
T, U &::= b \mid T \to U \mid T \times U && \text{(Types)} \\
t, u &::= x \mid tu \mid \lambda x.t \mid \langle t, u \rangle \mid \pi_1 u \mid \pi_2 u && \text{(Terms)} \\
\Gamma &::= \emptyset \mid x : T, \Gamma \qquad ((x : \_) \notin \Gamma) && \text{(Typing contexts)}
\end{aligned}
$$

# Curry-Howard Correspondence

Natural Deduction (Logic):

| Identity | Conjunction | Implication |
|---|---|---|
| $\dfrac{}{\Gamma, A \vdash A}$ Id | $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$ $\wedge$intro | $\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}$ $\supset$intro |
| | $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}$ $\wedge$elim$_1$ | $\dfrac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}$ $\supset$elim |
| | $\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$ $\wedge$elim$_2$ | |

Simply typed $\lambda$-calculus (Computation):

| Variable | Product | Function |
|---|---|---|
| $\dfrac{}{\Gamma, x : T \vdash x : T}$ | $\dfrac{\Gamma \vdash t : T \qquad \Gamma \vdash u : U}{\Gamma \vdash \langle t, u \rangle : T \times U}$ | $\dfrac{\Gamma, x : U \vdash t : T}{\Gamma \vdash \lambda x.t : U \to T}$ |
| | $\dfrac{\Gamma \vdash v : T \times U}{\Gamma \vdash \pi_1 v : T}$ | $\dfrac{\Gamma \vdash t : U \to T \qquad \Gamma \vdash u : U}{\Gamma \vdash t\, u : T}$ |
| | $\dfrac{\Gamma \vdash v : T \times U}{\Gamma \vdash \pi_2 v : U}$ | |

# Curry-Howard Correspondence

$$
\begin{aligned}
\text{Natural Deduction System} &\longleftrightarrow \lambda^{\rightarrow\times}\text{-calculus} \\
\text{Formulas} &\longleftrightarrow \text{Types} \\
\text{Proofs} &\longleftrightarrow \text{Terms} \\
\text{Proof transformations} &\longleftrightarrow \text{Term Reductions}
\end{aligned}
$$

Writing a program is the same as writing a *constructive* proof.

# Calculus of Constructions

It extends the CH Correspondence.

## Syntax of Terms

$$e ::= \texttt{Type} \mid \texttt{Prop} \mid x \mid ee \mid \lambda x : e.e \mid \forall x : e.e$$

- **Proofs**: terms whose type is a *proposition*, e.g.

$$\texttt{sum\_commut} : \forall(x : \mathbb{N}).\forall(y : \mathbb{N})\ .\ x + y = y + x$$

- **Propositions**: logical statements such as the above. They have type `Prop`.
- **Predicates**: functions that returns propositions.
- **Large types**: Types of predicates, e.g. `Prop`
- `Type`: Type of large types.

# Calculus of Constructions: Defining logical operators

$$
\begin{array}{rcll}
A \Rightarrow B & \equiv & \forall x : A.\, B & (x \notin B) \\
A \wedge B & \equiv & \forall C : \mathbf{P}.\, (A \Rightarrow B \Rightarrow C) \Rightarrow C & \\
A \vee B & \equiv & \forall C : \mathbf{P}.\, (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C & \\
\neg A & \equiv & \forall C : \mathbf{P}.\, (A \Rightarrow C) & \\
\exists x : A.\, B & \equiv & \forall C : \mathbf{P}.\, (\forall x : A.\, (B \Rightarrow C)) \Rightarrow C &
\end{array}
$$

# Coq and Gallina

- Coq is an implementation of CoC.
- Base language: **Gallina** - basically a clone of OCaml.

Inductive type definitions:

```
Inductive day : Type :=
  | monday
  | tuesday
  | wednesday
  | thursday
  | friday
  | saturday
  | sunday.
```

# Coq and Gallina

Function definitions & pattern matching:

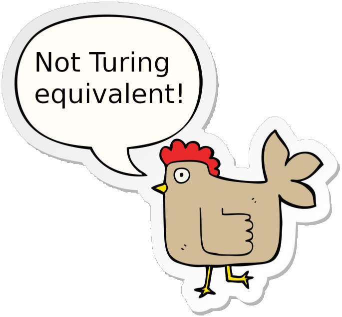
```
Definition next_weekday (d:day) : day :=
  match d with
  | monday ⇒ tuesday
  | tuesday ⇒ wednesday
  | wednesday ⇒ thursday
  | thursday ⇒ friday
  | friday ⇒ monday
  | saturday ⇒ monday
  | sunday ⇒ monday
  end.
```

# Coq and Gallina

Recursive function definition:

```
Fixpoint plus (n : nat) (m : nat) : nat :=
  match n with
  | O ⇒ m
  | S n’ ⇒ S (plus n’ m)
  end.
```

The argument of recursion must be decreasing, Coq does not support infinite recursion!



# How to write proofs in Coq

Say we want to provide a proof of the following:

$$\forall(x : \mathbb{N}).\forall(y : \mathbb{N}) \, . \, x + y = y + x$$

Two ways:

- Construct a term by hand:

```
Definition proof : (forall n m :
  nat, n + m = m + n) := [...]
```

  And provide it as proof of the above theorem.
- Use coq's **tactic system**.

# Tactics

When proving a statement in a proof environment, the theorem becomes a **goal**. You can use *tactics* to **introduce hypotheses**, split the goal into **subgoals** and apply **transformations** to your goal and hypotheses.

```
Goal 1
n : my_nat
IHn : forall m : my_nat, S (n .+ m) = n .+ S m
m : my_nat

(1 / 1) ______________________________________________
S (S n .+ m) = S n .+ S m
```

# Tactics


Induction
Creates a goal for each base case and inductive case, adding the appropriate induction hypotheses to the inductive cases.


Unfold
Applies δ-reduction, then reduces selected goals and hypotheses to βιζ-normal form.

# Conversion Rules

- $\beta$-reduction: the standard reduction for $\lambda$-terms
- $\delta$-reduction: replaces variables with their definition.
- $\iota$-reduction: reduction of pattern-matched or fixpoint definitions.
- $\zeta$-reduction: reduction of let-in definitions.

(No need to really remember these)

# End of presentation - open coq

**Install coq here**