

# INE22 : Preuves de Programmes

## Types Inductifs en Coq

(d'après C. Paulin, Types SummerSchool 2009)

Assia Mahboubi

INRIA - TypiCal, LIX École Polytechnique

07/05/2009

# Rappel des épisodes précédents

- ▶ Preuves formelles : une définition (méta-)mathématique des preuves
- ▶ Gentzen : déduction naturelle et arbres de preuves
- ▶ Plusieurs formalismes pour ces fondations : théorie des ensembles, logique d'ordre supérieur,...
- ▶ Théories des types : la famille de Coq
- ▶ Correspondance de Curry-Howard :
  - ▶ un énoncé est un type
  - ▶ une preuve est un habitant (un programme) de ce type

# Encodage des entiers

Deux choix initiaux de fondations différents :

- ▶ En théorie des ensembles tout est ensembliste :  $x \in A$
- ▶ En théorie des types, tout est fonctionnel :  $x : A$

Deux types d'observations sur les objets :

- ▶ En théorie des ensembles, on compare les éléments d'un ensemble.
- ▶ En théorie des types, on compare le corps des fonctions.

Comment représenter des données comme les entiers ?

Comment prouver que  $0 \neq 1$ ?

# Définitions inductives

Une définition inductive *introduit* une nouvelle collection  $I$  d'objets donnés par :

- ▶ un ensemble de **règles de construction** pour les éléments de  $I$
- ▶ un critère d'**initialité** : si  $T$  admet ces mêmes règles de constructions, alors  $I \subseteq T$  :
  - ▶  $I$  est “la plus petite” collection close par ces règles
  - ▶ deux règles de construction distinctes donnent des objets distincts

# Définitions inductives en Coq

Une notion primitive de **types inductifs** :

- ▶ Très puissante
- ▶ Fondamentale pour le système : égalité, points fixes, filtrage,...
- ▶ Nécessitant des choix techniques subtils pour préserver la consistance et la puissance

# Définitions inductives en Coq : Exemples

- ▶ Les booléens : `Inductive bool := true | false.`
- ▶ Le faux : `Inductive empty := .`
- ▶ Le vrai : `Inductive unit : tt.`
- ▶ La disjonction constructive :  
`Inductive sum A B :=`  
`inl : A → sum A B`  
`| inr : B → sum A B.`

## Définitions inductives en Coq : Exemples (suite)

- ▶ La conjonction :  
`Inductive` `prod A B := pair : A → B → prod A B.`
- ▶ Le produit dépendant (existantiel) :  
`Inductive` `sig (P : A → Prop) :=`  
`sigi : ∀ x : A, P x → sig P.`

Le  $\forall$  du constructeur est un existentiel constructif ...

# Définitions inductives en Coq : Forme générale

**Inductive**  $I$  *params* :=

...

|  $c_i : \forall x_1 : A_1 \dots x_n : A_n, I$  *params* ...

- ▶ Cas non récursif :  $I$  n'apparaît pas dans les  $A_i$ .
- ▶ Introduction : tout  $x : I$  a la forme  $c_i a_1 \dots a_n$ .
- ▶ Élimination : Analyse par cas exhaustive.
- ▶ Des conditions de bonne formation (positivité).

**Definition**  $f$  ( $x : \text{bool}$ ) :=

match  $x$  with

| true  $\Rightarrow$  ...

| false  $\Rightarrow$  ...

end.

# Définitions récursives

Génération d'un principe de récursion dite structurelle.

Exemples :

- ▶ Entiers
- ▶ Listes (polymorphes)
- ▶ Arbres binaires (polymorphes)
- ▶ Arbres à branchement dénombrable

Pour chacun donnez une définition inductive et le principe de récurrence généré.

La tactique (`induction x`) applique le principe de récurrence associé à `x` pour prouver un but.

# Définitions récursives

On peut aussi définir des prédicats inductifs :

- ▶ cf move et moves du tp2
- ▶ Relation d'ordre sur les entiers (exercice)
- ▶ Relation d'égalité :

**Inductive**  $eq := eqrefl : forall\ x, eq\ x\ x.$

Principe généré (Leibniz) pour le confort :

$\forall P\ x\ y, P\ x \Rightarrow eq\ x\ y \Rightarrow Q\ y.$

C'est la base de la réécriture (`rewrite`).

# Fonctions récursives

- ▶ Pour garder la consistance, on doit bannir les fonctions non terminantes.
- ▶ On ne peut pas demander une preuve de terminaison à chaque définition.
- ▶ On accepte directement les définitions à décroissance structurelle.

## Fonctions récursives : exemple

```
Fixpoint exemple (x : nat) :=  
  match x with  
  | 0 => 0  
  | 1 => 0  
  | S (S n) => exemple n  
end.
```

On traverse explicitement un nombre strictement positif de constructeurs avant de faire l'appel récursif.  
Sinon il faut donner explicitement l'argument de terminaison.

# Programmer et calculer en Coq

- ▶ On peut écrire des programmes en Coq (presque...) comme en OCaml.
- ▶ On peut prouver des théorèmes qui parlent de ces programmes, par exemple des procédures de décision.
- ▶ On peut les exécuter (`compute`), et même utiliser une compilation byte-code (`vm_compute`).
- ▶ On peut aussi les extraire (traduire) vers un langage comme OCaml ou Haskell.

## Et pour du code déjà écrit dans un autre langage?

- ▶ On peut annoter les programmes avec des formules adaptées (logique de Hoare) qui sont les spécifications.
- ▶ Ensuite des outils automatiques génèrent les obligations de preuves exprimant la correction du programme vis-à-vis de cette spécification.
- ▶ Puis on envoie ces théorèmes à prouver vers des outils automatiques et/ou interactifs.

# Et pour du code déjà écrit dans un autre langage?

- ▶ Une analyse statique : information inférée sans exécuter le code source
- ▶ Langages annotés : Java, C
- ▶ Techniques : logiques spécifiques, slicing,...
- ▶ Exemples :
  - ▶ Why : <http://why.lri.fr/index.fr.html>
  - ▶ Frama-C : <http://frama-c.cea.fr/>
  - ▶ Boogie (Microsoft Research)
  - ▶ ...