Séminaire Coq

Introduction et utilisation basique

Maxime Folschette

14 février 2013

Définitions

Définition inductive $\rightsquigarrow$ Nouveaux types

```
Inductive bool : Type :=
  | true : bool
  | false : bool.
```

Définition avec pattern matching $\rightsquigarrow$ Déconstruction

```
Definition negb (b:bool) : bool :=
  match b with
    | true => false
    | false => true
  end.
```

Définition d'un théorème et écriture de sa preuve

```
Theorem <nom> : <proposition>.
Proof.
  <tactiques et conclusions>
Qed.
```

Environnement de preuves

```
  Contexte (variables et hypotheses)
  ==================
    Objectif actif

Objectifs en attente
...
```

$\Rightarrow$ On cherche à prouver l'objectif actif avant de passer aux autres.

Proposition = Assertion, ≪ phrase ≫ non simplifiable.

Exemple

- $2 < 5$
- $\forall x \in \mathbb{R}, x^2 \geq 0$
- $\forall x \in \mathbb{R}, \exists y_1, y_2 \in \mathbb{R}, y_1 \neq y_2 \wedge y_1^2 = y_2^2 = x$
- $5 < 2$

Propositions et preuves

Proposition = Assertion, « phrase » non simplifiable.

Exemple

- $2 < 5$
- $\forall x \in \mathbb{R}, x^2 \geq 0$
- $\forall x \in \mathbb{R}, \exists y_1, y_2 \in \mathbb{R}, y_1 \neq y_2 \wedge y_1^2 = y_2^2 = x$
- $5 < 2$

Preuve = Succession de tactiques pour résoudre chaque objectif.

Exemple

- `simpl` → Simplifier l'objectif courant.
- `reflexivity` → Résoudre l'objectif courant s'il a la forme : $X = X$.
- `intros a` → On pose a (pour répondre à $\forall a, \ldots$).
- `intros H` → On suppose H (pour répondre à $H \Rightarrow \ldots$).
- `rewrite H` → Si $H : x = y$, remplacer tout x par y.
- `apply H` → Conclure en appliquant H.

Définitions récursives

Type paramétré = Type dépendant d'un autre type.

`option X` — Une option sur le type `X` est :

- soit un élément qui ne contient aucune valeur $\rightsquigarrow$ `None`
- soit un élément qui contient une valeur de type `X` $\rightsquigarrow$ `Some x`

Définitions récursives

Type paramétré = Type dépendant d'un autre type.

`option X` — Une option sur le type `X` est :

- soit un élément qui ne contient aucune valeur ⇝ `None`
- soit un élément qui contient une valeur de type `X` ⇝ `Some x`

Type récursif = Type qui fait référence à lui-même.

`nat` — Un entier naturel `nat` (dans l'arithmétique de Peano) est :

- soit l'élément nul ⇝ `O`
- soit le successeur d'un entier naturel ⇝ `S n`

Type paramétré = Type dépendant d'un autre type.

`option X` — Une option sur le type `X` est :

- soit un élément qui ne contient aucune valeur ⇝ `None`
- soit un élément qui contient une valeur de type `X` ⇝ `Some x`

Type récursif = Type qui fait référence à lui-même.

`nat` — Un entier naturel `nat` (dans l'arithmétique de Peano) est :

- soit l'élément nul ⇝ `O`
- soit le successeur d'un entier naturel ⇝ `S n`

Exemple de type récursif paramétré : les listes.

`list X` — Une liste d'éléments de type `X` est :

- soit la liste vide ⇝ `[]`
- soit un élément de `X` (tête) et une liste de `X` (queue) ⇝ `h :: t`

Fonction récursive = Fonction qui fait référence à elle-même.

$\rightarrow$ Problème : quid des fonctions récursives qui ne terminent pas ?
Quel résultat ? Quel type ?

$\Rightarrow$ Mathématiquement non défini

Fonction récursive = Fonction qui fait référence à elle-même.

$\rightarrow$ Problème : quid des fonctions récursives qui ne terminent pas ?
Quel résultat ? Quel type ?

$\Rightarrow$ Mathématiquement non défini

$\rightarrow$ Solution : forcer la terminaison
Décroissance structurelle = Une fonction ne peut s'appeler elle-même qu'avec au moins un argument strictement inférieur structurellement
X **est structurellement inférieur à** Y = On peut construire Y à partir de X

$\Rightarrow$ On finit toujours dans un cas dégénéré

Fonction récursive = Fonction qui fait référence à elle-même.

$\rightarrow$ Problème : quid des fonctions récursives qui ne terminent pas ?
Quel résultat ? Quel type ?

$\Rightarrow$ Mathématiquement non défini

$\rightarrow$ Solution : forcer la terminaison
Décroissance structurelle = Une fonction ne peut s'appeler elle-même qu'avec au moins un argument strictement inférieur structurellement
X **est structurellement inférieur à** Y = On peut construire Y à partir de X

$\Rightarrow$ On finit toujours dans un cas dégénéré

Exemple

La fonction `length` s'appelle elle-même sur la queue de la liste, qui est structurellement plus petite. (On peut construire la liste de départ à partir de sa tête et de sa queue.)

Preuves par récurrence

Récurrence sur $\mathbb{N}$ (scolaire)

Si on parvient à montrer :

- P_0,
- $\forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}$,

Alors on a :

- $\forall n \in \mathbb{N}, P_n$.

Autrement dit : $(P_0 \wedge \forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}) \Rightarrow \forall n \in \mathbb{N}, P_n$.

Récurrence sur $\mathbb{N}$ (scolaire)

Si on parvient à montrer :

- P_0,
- $\forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}$,

Alors on a :

- $\forall n \in \mathbb{N}, P_n$.

Autrement dit : $(P_0 \wedge \forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}) \Rightarrow \forall n \in \mathbb{N}, P_n$.

Récurrence sur "nat" (structurelle)

$P(\mathtt{0}) \Rightarrow (\forall \mathtt{n} \in \mathtt{nat}, P(\mathtt{n}) \Rightarrow P(\mathtt{S\ n})) \Rightarrow \forall \mathtt{n} \in \mathtt{nat}, P(\mathtt{n})$

avec associativité à droite de "$\Rightarrow$" :

$$A \Rightarrow B \Rightarrow C \quad \equiv \quad A \Rightarrow (B \Rightarrow C) \quad \equiv \quad (A \wedge B) \Rightarrow C$$

Récurrence sur $\mathbb{N}$ (scolaire)

Si on parvient à montrer :

- P_0,
- $\forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}$,

Alors on a :

- $\forall n \in \mathbb{N}, P_n$.

Autrement dit : $(P_0 \land \forall n \in \mathbb{N}, P_n \Rightarrow P_{n+1}) \Rightarrow \forall n \in \mathbb{N}, P_n$.

Récurrence sur "nat" (structurelle)

$$P(\mathtt{0}) \Rightarrow (\forall \mathtt{n} \in \mathtt{nat}, P(\mathtt{n}) \Rightarrow P(\mathtt{S\ n})) \Rightarrow \forall \mathtt{n} \in \mathtt{nat}, P(\mathtt{n})$$

Récurrence sur "list X" (structurelle)

$$P(\mathtt{[]}) \Rightarrow (\forall \mathtt{x} \in \mathtt{X}, \forall \mathtt{l} \in \mathtt{list\ X}, P(\mathtt{l}) \Rightarrow P(\mathtt{x\ ::\ l})) \Rightarrow \forall \mathtt{l} \in \mathtt{list\ X}, P(\mathtt{l})$$

avec associativité à droite de "$\Rightarrow$" :

$$A \Rightarrow B \Rightarrow C \quad \equiv \quad A \Rightarrow (B \Rightarrow C) \quad \equiv \quad (A \land B) \Rightarrow C$$

```
H : P -> Q
```

```
H : P -> Q
f : P -> Q
```

Séminaire Coq

Introduction et utilisation basique

Maxime Folschette

https://github.com/nantes-fp/seance-coq

14 février 2013

http://coq.inria.fr/
http://www.cis.upenn.edu/~bcpierce/sf/
http://www.coursera.org/course/progfun
http://www.labri.fr/perso/casteran/CoqArt/index.html
