

INE22 : Preuves de Programmes

Introduction à la correspondance de Curry-Howard

Assia Mahboubi

INRIA - TypiCal, LIX École Polytechnique

29/04/2009

Un peu de logique : déduction naturelle

La grande idée de Gentzen :

une **preuve**, c'est un **arbre**

- ▶ Abandon du mode axiomatique
- ▶ Seulement des règles de déduction
- ▶ Pour chaque connecteur, une règle d'introduction et une d'élimination

Un peu de logique : déduction naturelle

- ▶ Chaque preuve démontre un *but* **sous** des *hypothèses* :

$$\frac{h_1 \dots h_n}{g}$$

- ▶ Les hypothèses doivent elles-mêmes être justifiées (être sous une barre).
- ▶ Les règles de construction s'appellent des **règles d'inférence**.

Un peu de logique : déduction naturelle

Règles d'introduction : comment prouver un théorème qui commence par ...

- ▶ Une disjonction :

$$\frac{A}{A \vee B} \quad \frac{B}{A \vee B}$$

- ▶ Une conjonction :

$$\frac{A \quad B}{A \wedge B}$$

- ▶ Une implication :

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \Rightarrow B}$$

Un peu de logique : déduction naturelle

Règles d'introduction : comment prouver un théorème qui commence par ...

- ▶ Quantificateur \exists :

$$\frac{P[x := t]}{\exists x, P}$$

- ▶ Quantificateur \forall :

$$\frac{P}{\forall x, P}$$

si x n'apparaît dans aucune hypothèse déchargée

- ▶ Axiome : \overline{P}

Un peu de logique : déduction naturelle

Règles d'élimination : comment utiliser une hypothèse de la forme

...

- ▶ Une disjonction :

$$\frac{\begin{array}{cc} [A] & [B] \\ \vdots & \vdots \\ \dot{P} & \dot{P} \end{array} \quad A \vee B}{P}$$

- ▶ Une conjonction :

$$\frac{A \wedge B}{A} \quad \frac{A \wedge B}{B}$$

- ▶ Une implication :

$$\frac{A \Rightarrow B \quad A}{B}$$

Un peu de logique : déduction naturelle

Règles d'élimination : comment **utiliser** une hypothèse de la forme

...

- ▶ Quantificateur \exists :

$$\frac{\begin{array}{c} [A] \\ \vdots \\ \exists x A \quad P \end{array}}{P}$$

- ▶ Quantificateur \forall :

$$\frac{\forall x A}{A[x := t]}$$

- ▶ Faux :

$$\frac{\perp}{P}$$

- ▶ Égalité :

$$\frac{}{x = x}$$

Rmq : la négation $\neg P$ est une notation pour $P \Rightarrow \perp$

Exercices

Faire une preuve en déduction naturelle de :

▶ $A \wedge B \Rightarrow B \wedge A$

▶ $(\forall x, Px) \Rightarrow (\exists x, Px)$

Une règle un peu à part : le raisonnement par l'absurde

Ce n'est pas une règle d'introduction ni d'élimination :

$$\frac{[\neg P] \quad \vdots \quad \perp}{P}$$

On peut l'admettre ou pas : si oui, on est dans un cadre **classique**.

Langages et théories

Formaliser c'est :

- ▶ Définir une classe d'objets
- ▶ Étudier la prouvabilité de théorèmes sur ces objets

On se donne donc :

- ▶ un langage : variables, constantes, fonctions
ex : les entiers de Péano
- ▶ une théorie : relations, prédicats, quantifications, formules, axiomes
ex : théorie du 1er ordre des nombres réels

Parfois, on sait même écrire des algorithmes qui **décident** des théories.

Mais parfois ce n'est pas possible (et on le prouve) : ce sont des théories **indécidables**.

Revenons à Coq

On peut *inspecter* les termes :

- ▶ Check 3. `3 : nat`
- ▶ Check plus. `plus : nat → nat → nat`
- ▶ Check plus 1 2. `3 : nat`

Et aussi des termes que l'on construit soi-même :

- ▶ Check `(fun x : nat => x + 3).` : `nat → nat`
- ▶ Avec moins d'information de typage (inférence) :
Check `(fun x => x + 3).` : `nat → nat`

Merci à Y. Bertot

Revenons à Coq

On peut aussi **calculer** les termes :

- ▶ Check $((\text{fun } x \Rightarrow x + 3) 4)$. : **nat**
- ▶ Eval compute in $((\text{fun } x \Rightarrow x + 3) 4)$. **7** : **nat**

L'égalité tient compte de cette possibilité de calcul :

Eval compute in $(\text{fun } x \Rightarrow x + 3) 4 = 7$. **7 = 7**

Des preuves aux programmes

- ▶ On a vu la règle du modus ponens (élim. du \Rightarrow): $\frac{A \Rightarrow B \quad A}{B}$
- ▶ Elle cache une fonction qui transforme (le type d') une preuve de A en (le type d') une preuve de B
- ▶ On a vu la règle d'élimination du \forall : $\frac{\forall x : A, P}{P[x := t]}$
- ▶ Elle cache une fonction qui transforme tout élément t de A en une preuve de $P[x := t]$

Des programmes aux preuves ?

En fait, le contraire marche aussi, avec des fonctions totales : une fonction $A \rightarrow B$ sert à montrer le théorème $A \Rightarrow B$.

Et pour les autres règles ?

- ▶ Tant qu'on a pas de quantificateurs on parle de types simples (OCaml, Haskell,...).
- ▶ Mais pour les quantificateurs, il faut des types plus sophistiqués.

Correspondance de Curry-Howard, en Coq

- ▶ `Prop` est un type, qui contient des types.
- ▶ Ces types, habitants de `Prop` sont eux-même des types de preuves.
- ▶ Soit `A : Prop` et `B : Prop` des types de preuves.
- ▶ `A → B` est aussi un type de preuves.
- ▶ Un “type de preuves” est en fait une “proposition”.
- ▶ Une proposition est valide si elle est habitée par une preuve.

Correspondance de Curry-Howard, en Coq

On recommence :

- ▶ Si A , B et C sont des propositions :
- ▶ Une fonction de type $A \rightarrow B$ transforme une preuve de A en une preuve de B .
- ▶ Cette fonction représente donc une preuve de $A \rightarrow B$.
- ▶ On peut même parfois le faire directement :
 - ▶ $\text{fun } x : A \Rightarrow x : A \rightarrow A$.
 - ▶ $\text{fun } (f : A \rightarrow B \rightarrow C)(x : B)(y : A) \Rightarrow f y x$
: $(A \rightarrow B \rightarrow C) \rightarrow B \rightarrow A \rightarrow C$.

Correspondance de Curry-Howard

- ▶ On aurait un moyen de montrer toutes les tautologies ? Non.
- ▶ Contrexemple : la loi de Pierce $((A \rightarrow B) \rightarrow A) \rightarrow A$
- ▶ On travaille en fait avec la logique **intuitionniste**.
- ▶ Pour avoir la logique classique, il faut aussi un axiome : le tiers-exclus : $\forall P, P \vee \neg P$

Correspondance de Curry-Howard : le \forall

Pour étendre la correspondance preuves-fonctions on introduit:

- ▶ Un type des types : Type
- ▶ les familles de types : $(B_i)_{i \in A}$
ie une fonction : $B : A \rightarrow \text{Type}$
- ▶ les fonctions qui à un x dans A associe une valeur de type $B(x)$

Le système de types de Coq donne le type $\forall x, B x$ à une telle fonction.

Correspondance de Curry-Howard : le \forall

- ▶ On peut maintenant écrire le type de théorèmes :
`forall x y, even x → divides x y → even y`
- ▶ Mais aussi le type de fonctions :
`fun x : nat => (fun h : even x => even x)`
`: forall x, even x => even x`
- ▶ Et on peut les appliquer :
`(fun x : nat => (fun h : even x => even x)) 3`
`: even 3 => even 3`

Correspondance de Curry-Howard

- ▶ Elle s'étend à toute la logique intuitionniste.
- ▶ Elle donne un **contenu calculatoire** aux preuves.
- ▶ Le contenu calculatoire des preuves classiques est un sujet de recherche actif.

Un peu de modélisation en Coq

En TP.