

# Normalisation par Evaluation en Coq

François Garillot

sous la direction de Benjamin Werner  
LIX, Ecole Polytechnique  
Mars-Août 2007

# Normalisation par evaluation : objectifs

---

- ◆ méthode de  $\beta\eta$ -normalisation
- ◆ usuellement abordée dans un contexte de réduction (normalisation sans réduction, évaluation partielle ...)
- ◆ peut-elle être vue comme **inverse de la fonctionnelle d'évaluation** ?
- ◆ ... formalisée en Coq ?
- ◆ ... servir à manipuler des termes d'ordre supérieur, donner une représentation de la **syntaxe abstraite d'ordre supérieur** ?

# Exemple

---

Deux types de représentations des termes du  $\lambda$ -calcul simplement typé:

◆ **représentation compilée**

```
fun x  $\Rightarrow$  x : T  $\rightarrow$  T
```

◆ **ou représentation explicite.**

```
Inductive term : Type :=
```

```
  Var : id  $\rightarrow$  term
```

```
| Lam : id  $\rightarrow$  term  $\rightarrow$  term
```

```
| App : term  $\rightarrow$  term  $\rightarrow$  term.
```

On veut décompiler  $f : (\Lambda \rightarrow \Lambda) \rightarrow (\Lambda \rightarrow \Lambda)$

f

# Exemple

Deux types de représentations des termes du  $\lambda$ -calcul simplement typé:

◆ **représentation compilée**

`fun x  $\Rightarrow$  x : T  $\rightarrow$  T`

◆ **ou représentation explicite.**

`Inductive term : Type :=`

`Var : id  $\rightarrow$  term`

`| Lam : id  $\rightarrow$  term  $\rightarrow$  term`

`| App : term  $\rightarrow$  term  $\rightarrow$  term.`

On veut décompiler  $f : (\Lambda \rightarrow \Lambda) \rightarrow (\Lambda \rightarrow \Lambda)$

`LAM(x $\Lambda \rightarrow \Lambda$ ,                    f                    (VAR x) $\Lambda \rightarrow \Lambda$                     )`

# Exemple

Deux types de représentations des termes du  $\lambda$ -calcul simplement typé:

◆ **représentation compilée**

`fun x  $\Rightarrow$  x : T  $\rightarrow$  T`

◆ **ou représentation explicite.**

`Inductive term : Type :=`

`Var : id  $\rightarrow$  term`

`| Lam : id  $\rightarrow$  term  $\rightarrow$  term`

`| App : term  $\rightarrow$  term  $\rightarrow$  term.`

On veut décompiler  $f : (\Lambda \rightarrow \Lambda) \rightarrow (\Lambda \rightarrow \Lambda)$

`LAM(x $\Lambda \rightarrow \Lambda$ ,            f ( $\lambda a^{\Lambda}$ .APP((VAR x) $\Lambda \rightarrow \Lambda$ , a $\Lambda$ )))`

# Exemple

Deux types de représentations des termes du  $\lambda$ -calcul simplement typé:

◆ **représentation compilée**

```
fun x => x : T -> T
```

◆ ou **représentation explicite.**

```
Inductive term : Type :=
```

```
  Var : id -> term
```

```
| Lam : id -> term -> term
```

```
| App : term -> term -> term.
```

On veut décompiler  $f : (\Lambda \rightarrow \Lambda) \rightarrow (\Lambda \rightarrow \Lambda)$

$$\text{LAM}(x^{\Lambda \rightarrow \Lambda}, \text{LAM}(y^{\Lambda}, (f (\lambda a^{\Lambda}. \text{APP}((\text{VAR } x)^{\Lambda \rightarrow \Lambda}, a^{\Lambda}))))(\text{VAR } y)^{\Lambda}))$$

# Normalisation par Evaluation

Utiliser le mécanisme d'évaluation d'un méta-langage pour *normaliser* et *représenter* les termes du lambda-calcul.

Pour cela, construire une fonction de normalisation ayant deux propriétés

(1) Correction

$$r \triangleright_{\beta\eta} s \Rightarrow [[r]]_{\sigma} = [[s]]_{\sigma}$$

(2) Reproduction

$$r \text{ en forme normale} \Rightarrow \downarrow [[r]]_{\uparrow} = r$$

$\uparrow$  une valuation.

On recherchera aussi le résultat de complétude:

$$t \text{ bien typé} \Rightarrow t \triangleright_{\beta\eta} \downarrow [[t]]_{\uparrow}$$

# Normalisation par Evaluation

Construisons une interprétation sur ces bases

$$\begin{array}{l} [[\tau_0]] = B \\ \hline [[\tau_1 \rightarrow \tau_2]] = [[\tau_1]] \rightarrow [[\tau_2]] \end{array}$$

$$\begin{array}{l} [[\text{VAR}(x)]]_\rho = \rho_B(x) \\ [[\text{LAM}(x^\tau, m_0)]]_\rho = \lambda a^{[[\rho]]}. [[m_0]]_{\rho[x \mapsto a]} \\ \text{abstraction de Coq !} \\ [[\text{APP}(m_1, m_2)]]_\rho = [[m_1]]_\rho ([[m_2]]_\rho) \end{array}$$

Variable alpha : Type.

```
Fixpoint tr (T:ST){struct T}: Type:=  
  match T with  
  | Iota => alpha  
  | Arr A B => (tr alpha A) -> (tr alpha B)  
end.
```

**Problème:**  
**choisir** alpha

Si  $r \triangleright_{\beta\eta} s$ , alors  $[[r]] = [[s]]$  (**correction**).

**Problème:** la sémantique n'est définie que pour les termes bien typés

# Construire une fonction de normalisation

$\text{nf}_\tau \in [[\tau]] \rightarrow \Lambda$

$E$  en forme  $\beta$ -normale  $\eta$ -longue  $\Rightarrow \text{nf}_\tau([[E]]) = E$ . (reproduction)

Si  $\tau = \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow b$ , où  $\forall i \tau_i = \tau_{i1} \rightarrow \dots \rightarrow \tau_{im_i} \rightarrow b_i$ ,

$E : \tau \equiv \lambda x_1 \dots \lambda x_n. [[E]] (\dots (x_i E_1 \dots E_{m_i}) \dots)$   
normaux  $\uparrow$

$\text{nf}_\tau : f \rightarrow \text{LAM}(v_1, \dots, \text{LAM}(v_n,$   
 $f(\lambda a_1 \dots \lambda a_{m_1} \text{APP}(\dots \text{APP}(\text{VAR}v_1, \text{nf}_{\tau_{11}}(a_1)) \dots, \text{nf}_{\tau_{1m_1}}(a_{m_1})))$   
 $\vdots$   
 $(\lambda a_1 \dots \lambda a_{m_n} \text{APP}(\dots \text{APP}(\text{VAR}v_n, \text{nf}_{\tau_{n1}}(a_1)) \dots, \text{nf}_{\tau_{nm_n}}(a_{m_n}))))$ )

# Construire une fonction de normalisation

◆ *réifie*,

$$\downarrow^\tau: [[\tau]] \rightarrow \Lambda$$

◆ *réfléchit*,

$$\uparrow^\tau: \Lambda \rightarrow [[\tau]]$$

$$\downarrow^b \mathfrak{l} = \mathfrak{l}$$

$$\downarrow^{\tau_1 \rightarrow \tau_2} f = \text{LAM}(x^{\tau_1}, \downarrow^{\tau_2} (f(\uparrow^{\tau_1} \text{VAR}(x)))) \text{ pour } x \text{ variable fraîche}$$

$$\uparrow^b \mathfrak{l} = \mathfrak{l}$$

$$\uparrow^{\tau_1 \rightarrow \tau_2} \mathfrak{l} = \lambda a^{[[\tau_1]]}. \uparrow^{\tau_2} (\text{APP}(\mathfrak{l}, \downarrow^{\tau_1} a))$$

De nombreuses interprétations associées à cet algorithme, et à ses extensions: ensembliste (Berger et al.), domaines (Berger et al.), catégorielle (Altenkirch et al.), modèles de Kripke (Coquand) ...

# Reproduction

---

$$\text{nf}_\tau([[E]]) = \downarrow^\tau ([[E]]_\uparrow) = E$$

pour E en forme  $\beta$ -normale  $\eta$ -longue, par induction sur E:

# Reproduction

---

$$\text{nf}_\tau([[E]]) = \downarrow^\tau ([[E]]_\uparrow) = E$$

pour E en forme  $\beta$ -normale  $\eta$ -longue, par induction sur E:

◆ **Cas:**  $\text{APP}(\text{VAR}(x)^{\rho \rightarrow \tau}, N^\rho)$

$$\downarrow^\tau ([[xN]]_\uparrow) = \uparrow^{\rho \rightarrow \tau} (x)([[N]]_\uparrow) = \uparrow^\tau (\text{APP}(\text{VAR}(x), \downarrow^\rho ([[N]]_\uparrow))) = \text{APP}(\text{VAR}(x), N)$$

# Reproduction

$$\text{nf}_\tau([[E]]) = \downarrow^\tau ([[E]]_\uparrow) = E$$

pour E en forme  $\beta$ -normale  $\eta$ -longue, par induction sur E:

◆ **Cas: APP(VAR(x) <sup>$\rho \rightarrow \tau$</sup> , N <sup>$\rho$</sup> )**

$$\downarrow^\tau ([[xN]]_\uparrow) = \uparrow^{\rho \rightarrow \tau} (x)([[N]]_\uparrow) = \uparrow^\tau (\text{APP}(\text{VAR}(x), \downarrow^\rho ([[N]]_\uparrow))) = \text{APP}(\text{VAR}(x), N)$$

◆ **Cas: LAM(y, N)**

$$\begin{aligned} \downarrow^{\rho \rightarrow \sigma} ([[LAM(y, N)]]_\uparrow) &= \text{LAM}(x, \downarrow^\sigma ([[LAM(y, N)]]_\uparrow (\uparrow^\rho \text{VAR}(x)))) \text{ pour } x \text{ fraiche} \\ &= \text{LAM}(x, \downarrow^\sigma ([[N]]_\uparrow [y \mapsto x])) \\ &= \text{LAM}(x, N_{[\text{VAR}(y) \mapsto \text{VAR}(x)]}) \\ &=_{\alpha} \text{LAM}(y, N) \end{aligned}$$



# Choix de formalisation

---

Les problèmes rencontrés jusqu'ici:

- ◆ Que veut dire "x variable fraîche" dans la définition de  $\downarrow^\tau$  ? Représentation des lieux

Les implémentations montrent deux écoles :

- ◆ variables nommées, conditions de fraîcheur garanties par un générateur à effets de bords (`gensym`). Générateur rarement justifié (excepté [Filinski 2001] par monades). [Danvy 1996] et al.
  - ◆ familles de termes, tour similaire à ci-dessus [Berger, Eberl, Schwichtenberg 1998]
  - ◆ (logique nominale)
- ◆ **Choix du type de base de notre interprétation**  
Pour la formalisation nommée:  $\Lambda$ , ensemble des lambda-termes  
Pour la formalisation par familles de termes:  $\mathbb{N} \rightarrow \Lambda$

# Implémentation de l'algorithme : définitions

---

```
Inductive ST : Set :=
```

```
  Iota : ST
```

```
| Arr : ST → ST → ST.
```

```
Record id : Type :=
```

```
  mkid idx : nat_eqType ; idT : ST.
```

```
Inductive term : Type :=
```

```
  Var : id → term
```

```
| Lam : id → term → term
```

```
| App : term → term → term.
```

```
Lam x (Var x) : term
```

(Librairies & tactiques de réflexion booléenne de Georges  
Gonthier)

```
Fixpoint FV (t:term): seq idd :=
```

```
  match t with
```

```
  | Var n ⇒ Seq n
```

```
  | App t u ⇒ cat (FV t)(FV u)
```

```
  | Lam n t ⇒ filter (setC1 n) (FV t)
```

```
end.
```

# Implémentation de l'algorithme : La fonction de normalisation

---

```
Fixpoint decomp (T:ST)struct T : (tr term T)→term :=
  match T as U return (tr term U)→term with
| Iota ⇒ fun (t:term) ⇒t
| Arr A B ⇒ fun t ⇒
    let x:= gensym (FV (decomp B (t (long A (Var dummy)))))) A in
      Lam x
        (decomp B (t (long A (Var x))))
  end
with
long (T:ST) struct T: term → (tr term T) := fun t ⇒
  match T as U return (tr term U) with
| Iota ⇒ t
| Arr A B ⇒ fun x:(tr term A) ⇒
    (long B (App t (decomp A x)))
end.
```

# variables nommées à la Stoughton

$$(\text{App } t \ u)[\sigma] \equiv \text{App } t[\sigma] \ u[\sigma]$$

$$(\text{Var } x)[\sigma] \equiv \text{Var } x \quad | (x, u) \notin \sigma$$

$$(\text{Var } x)[\sigma] \equiv u \quad | (x, u) \text{ première occurrence}$$

$$(\text{Lam } x \ t)[\sigma] \equiv \text{Lam } x' \ t[(x, \text{Var } x') :: \sigma]$$

$$\text{où } x' = \text{fresh} \left( \bigcup_{z \in FV(t) \setminus \{x\}} FV(\text{Var } z[\sigma]) \right)$$

- ◆ Remplacement systématique de la variable liée lors de la substitution
- ◆ Variable liée choisie canoniquement par induction sur les termes

**Lemma:**  $\bigcup_{z \in FV(t) \setminus \{x\}} FV(\text{Var } z[\sigma]) = FV((\text{Lam } x \ t)[\sigma])$

**Theorem:**  $M =_{\alpha} N \Rightarrow M\sigma = N\sigma$

Substitution *normalisante* pour l' $\alpha$ -conversion

# Implémentation : complexité

---

... let x:= gensym (FV (decomp B (t (long A (Var dummy)))))) A in ...

- ◆ Nous permet de réutiliser sans problème la formalisation nommée de Stoughton
- ◆ Croissance exponentielle du nombre d'appels
- ◆ Réemploie et améliore une astuce de [Berger, Schwichtenberg 1991]
- ◆ Possible de faire mieux ?

# Implémentation de l'algorithme : par familles de termes

```
Fixpoint decomp (T:ST) struct T : (tr (nat → term) T) → (nat → term) :=
  match T as U return (tr (nat → term) U) → (nat → term) with
| Iota ⇒ fun (t:nat → term) ⇒ t
| Arr A B ⇒ fun t ⇒
      fun n ⇒ Lam A (decomp B (t (long A (fun k ⇒ (Bvar (n)))))) (n+1))
  end
with
long (T:ST) struct T: (nat → term) → (tr (nat → term) T) := fun t ⇒
  match T as U return (tr (nat → term) U) with
| Iota ⇒ t
| Arr A B ⇒ fun x:(tr (nat → term) A) ⇒
      (long B (fun n ⇒ App (t n) (decomp A x n)))
end.
```

(Bvar sous-entend une convention de Barendregt, incrément à la méthode de Berger)

**Problème : traitement des termes à variables libres ?**

# Formalisation : point technique

$$t \text{ bien typé} \Rightarrow t \triangleright_{\beta\eta} \downarrow [[t]]_{\uparrow}$$

Or, si on rappelle notre manipulation à l'aide de dummy:

```
let x := gensym (FV (decomp B (t (long A (Var dummy))))) A in
```

```
Definition dummy := (mkid 0 Iota).
```

Et notre définition de l' $\eta$ -expansion:

```
| ceta : forall x t, x notin (FV t) → conv t (Lam x (App t (Var x)))
```

**Lemma:**

$$\forall x t u, (\text{App } t \text{ (Var } x) \rightsquigarrow u \wedge y \notin \text{FV}(u)) \implies \exists v, (t \rightsquigarrow v \wedge y \notin \text{FV}(v))$$

Ce lemme implique de passer par une  $\beta\eta$ -conversion *non typée*, sur laquelle on doit prouver le théorème de Church-Rosser, qui est admis ([Nipkow 96]).

# Syntaxe abstraite d'ordre supérieur

---

HOAS

```
Inductive term : Type :=  
| Lam : (term → term) → term  
| App : term → term → term.
```

```
Lam (f) : term
```

```
Inductive rred : term → term → Type :=  
| red_beta : forall (F: term → term) (t:term),  
                rred (app (lam F),t) (F t)
```

...

Ocurrence négative de `term` dans `Lam`

# Syntaxe abstraite d'ordre supérieur

## HOAS

```
Inductive term : Type :=  
| Lam : (term → term) → term  
| App : term → term → term.
```

```
Lam (f) : term
```

```
Inductive rred : term → term → Type :=  
|red_beta : forall (F: term → term) (t:term),  
                rred (app (lam F),t) (F t)
```

...

## Ocurrence négative de term dans Lam

## INDUCTIVE

```
Inductive term : Type :=  
  Var : id → term  
| Lam : id → term → term  
| App : term → term → term.
```

```
Lam x (Var x) : term
```

```
Inductive conv : term → term → Prop :=  
| crefl : forall t, conv t t  
| csym : forall t u, conv t u → conv u t  
| ctrans : forall t u v, conv t u → conv u v → conv t v  
| cbeta : forall t u x,  
            conv (App (Lam x t) u) (subst t ((x,u) :: nil))  
| calpha : forall x y T t1 t2,  
            (y^^T notin (FV (Lam (x^^T) t1))) →  
            conv (t1 [((x^^T), Var (y^^T))::nil]) t2 →  
                conv (Lam (x^^T) t1)(Lam (y^^T) t2)  
| ceta : forall x t, x notin (FV t) →  
            conv t (Lam x (App t (Var x)))  
| capp1 : forall u1 u2 v, conv u1 u2 → conv (App u1 v)(App u2 v)  
| capp2 : forall u v1 v2, conv v1 v2 → conv (App u v1)(App u v2)  
| clam : forall x t1 t2, conv t1 t2 → conv (Lam x t1)(Lam x t2).
```

# Syntaxe abstraite d'ordre supérieur

On peut définir des *termes pathologiques*

$$tt = (\text{Lam } x \rightarrow (\text{Lam } y \rightarrow x))$$
$$ff = (\text{Lam } x \rightarrow (\text{Lam } y \rightarrow y))$$
$$\begin{array}{lcl} \text{is}_{\text{app}} & : & \text{term} \quad \rightarrow \quad \text{term} \\ & & \text{App}(t, t') \quad \rightarrow \quad tt \\ & & \text{Lam}(t) \quad \rightarrow \quad ff \end{array}$$

Les différentes implémentations de HOAS:

- ◆ ELF : théorie des types faible, restreinte à  $\Pi_2$  ( $\forall\exists$ )
- ◆ weak HOAS :  $(\text{nat} \rightarrow \text{term})$ , inadapté [Despeyroux et al 1995], [Honsell et al 2007]
- ◆ two-level HOAS : metaPRL et suivants [Chen et Xi 2005]

Difficultés théoriques :

- ◆ Despeyroux, Pfenning 1999, lambda-calcul modal
- ◆ Martin Hoffman 1999, interprétation catégorielle : catégories de foncteurs

# Conclusion

---

- ◆ Normalisation par évaluation possible et formalisable rigoureusement avec une interprétation dans  $\Lambda$ .
- ◆ Formalisation des termes possédant des variables libres dans un certain contexte
- ◆ Implémentation plus efficace pressentie possible
- ◆ Application à HOAS vue comme un sujet intéressant, mais demande encore du travail