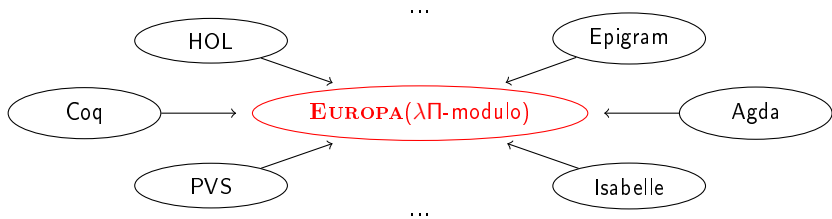


EUROPA: un noyau pour $\lambda\Pi$ -modulo

Mathieu Boespflug

6 juillet 2009



Présentation de $\lambda\Pi$ -modulo

Implantation

- Architecture

- La relation de typage

- Le test de conversion

Conclusion

Grammaire

Var $\ni x, y, z$

Term $\ni t, A, B ::= x \mid \lambda x : A. t \mid \Pi x : A B \mid t t$

Term \supset Term_{NF} $\ni t_n ::= t_a \mid \lambda x. t_n$

Term \supset Term_A $\ni t_a ::= x \mid t_a t_n$

Règles d'inférence

$$s \in \{Type, Kind\}$$
$$\frac{}{[] \text{ well-formed}}$$
$$\frac{\Gamma \vdash A : s}{\Gamma[x : A] \text{ well-formed}}$$
$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash Type : Kind}$$
$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$
$$\frac{\Gamma \vdash A : Type \quad \Gamma[x : A] \vdash B : s}{\Gamma \vdash \Pi x : A B : s}$$
$$\frac{\Gamma \vdash A : Type \quad \Gamma[x : A] \vdash B : s \quad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A. t : \Pi x : A B}$$
$$\frac{\Gamma \vdash t : \Pi x : A B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : (u/x)B}$$
$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s \quad \Gamma \vdash t : A}{\Gamma \vdash t : B} \quad A \equiv_{\beta\mathcal{R}} B$$

Exemple d'un script de preuve

```
nat : Type.
```

```
0 : nat.
```

```
S : nat -> nat.
```

```
1 : nat.
```

```
[] 1 --> (S 0).
```

```
plus : nat -> nat -> nat.
```

```
[x : nat] plus x 0 --> x.
```

```
[x : nat] plus 0 x --> x.
```

```
[x : nat, y : nat] plus x (S y) --> S (plus x y).
```

```
[x : nat, y : nat] plus (S x) y --> S (plus x y).
```

Remarques

Enjeu: créer un outil pour vérifier le typage des termes dans un script de preuves exprimées $\lambda\Pi$ -modulo.

- ▶ Conversion nécessaire même pour vérifier des termes simples dans des théories simples.
- ▶ Règle de conversion paramétrée par la théorie considérée.

Remarques

Enjeu: créer un outil pour vérifier le typage des termes dans un script de preuves exprimées $\lambda\Pi$ -modulo.

- ▶ Conversion nécessaire même pour vérifier des termes simples dans des théories simples.
- ▶ Règle de conversion paramétrée par la théorie considérée.

Idée 1: un générateur de vérificateurs de types spécialisés à la théorie considérée.

Remarques

Enjeu: créer un outil pour vérifier le typage des termes dans un script de preuves exprimées $\lambda\Pi$ -modulo.

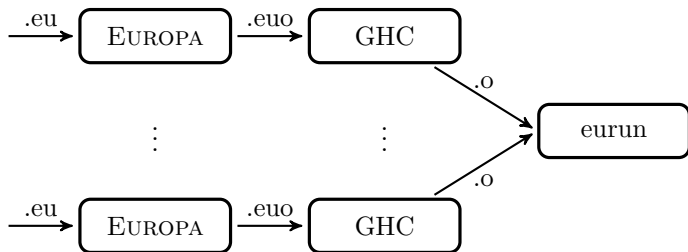
- ▶ Conversion nécessaire même pour vérifier des termes simples dans des théories simples.
- ▶ Règle de conversion paramétrée par la théorie considérée.

Idée 1: un générateur de vérificateurs de types spécialisés à la théorie considérée.

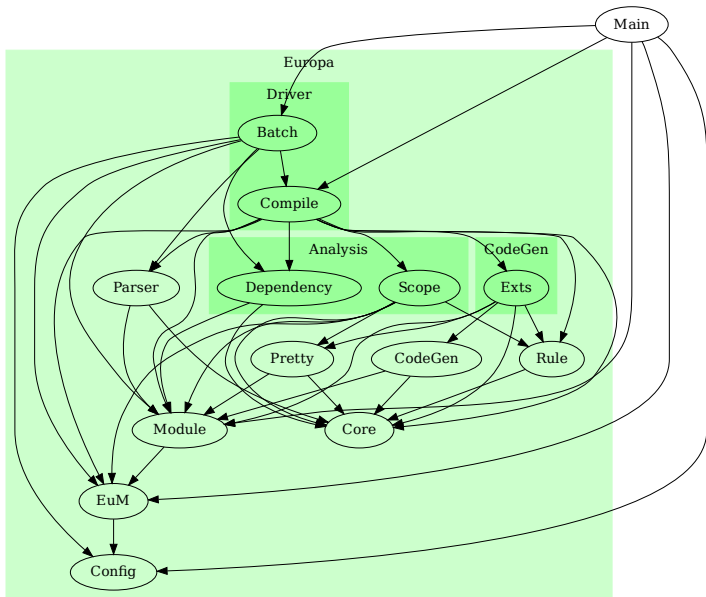
Idée 2: un générateur de vérificateurs de types spécialisés aux termes de preuves donnés.

EUROPA: un méta-vérificateur

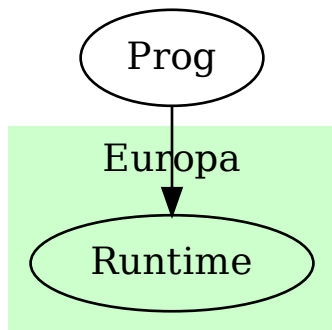
Vue macroscopique



... et au microscope



Runtime



Interface de *Europa.Runtime*:

- ▶ Types de données,
- ▶ La fonction *typeOf*.

Représentations de termes

- ▶ La relation de typage analyse la forme du terme donné en argument.
→ *Besoin d'une représentation **statique**.*
- ▶ Le prédicat de conversion compare les *valeurs* de termes.
→ *Besoin d'une représentation **dynamique**.*

Représentation statique

```
data Term = TLam Term (Term → Term)
  | TPi Term (Term → Term)
  | TApp Term Term
  | TType
  | Box Code Code
  | UBox Term Code
```

- ▶ Représentation en syntaxe abstraite d'ordre supérieure (HOAS).
- ▶ Nul besoin d'environnement de typage — remplacé par des boîtes.

Représentation statique

```
data Term = TLam Term (Term → Term)  
  | TPi Term (Term → Term)  
  | TApp Term Term  
  | TType  
  | Box Code Code  
  | UBox Term Code
```

- ▶ Représentation en syntaxe abstraite d'ordre supérieure (HOAS).
- ▶ Nul besoin d'environnement de typage — remplacé par des boîtes.

Boîte: paire d'un terme avec son type.

La relation de typage

$$\frac{}{t \vdash_n \boxed{t : A} : A}$$

$$\frac{}{t \vdash_n \text{Type} : \text{Kind}}$$

$$\frac{f \boxed{x : A} \vdash_n _ : B}{t \vdash_n \lambda \boxed{A : \text{Type}} f : \Pi A (\lambda x. B)}$$

$$\frac{f \boxed{n : A} \vdash_{n+1} _ : s}{t \vdash_n \Pi A f : s}$$

$$\frac{t_1 \vdash_n _ : \Pi A' f}{t \vdash_n t_1 \boxed{t_2 : A} : f t_2} A \equiv A'$$

Dérivation pour $nat : Type$:

$$\frac{}{nat \vdash_n \boxed{nat : Type} : Type}$$

Dérivation pour $O : nat$:

$$\frac{}{O \vdash_n \boxed{O : nat} : nat}$$

Dérivation pour $vec : \Pi x : nat \text{ Type}$:

$$\frac{f \boxed{n : nat} \vdash_{n+1} Type : Kind}{vec \vdash_n \Pi \boxed{nat : Type} f : Kind}$$

Dérivation pour $nil : vec O$:

$$\frac{vec \vdash_n \boxed{\Pi nat f : vec} : \Pi nat f}{nil \vdash_n vec \boxed{O : nat} : f O} \quad nat \equiv nat$$

Le test de conversion

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s \quad \Gamma \vdash t : A}{\Gamma \vdash t : B} A \equiv_{\beta\mathcal{R}} B$$

Représentation dynamique

```
data Code = Var Int
  | Con String
  | Lam (Code → Code)
  | Pi Code (Code → Code)
  | N Code Code
  | Type
  | Kind

ap :: Code → Code → Code
ap (Lam f) t = f t
ap t1 t2 = N t1 t2
```

- ▶ Annotations de type sur les λ effacés.

Représentation dynamique: exemple

$$(\lambda x. (\lambda y. y x)) z$$

Représentation dynamique: exemple

$$(\lambda x. (\lambda y. y x)) z$$

$$ap (Abs (\lambda x \rightarrow Abs (\lambda y \rightarrow ap y x))) (Con "z")$$

Règles de réécriture

- ▶ patterns: $p ::= _ \mid x \mid c(p_1, \dots, p_n)$
 - ▶ valeurs: $v ::= c(v_1, \dots, v_n)$
-

$$\begin{array}{l} \llbracket e \rrbracket = e \\ \llbracket _ \rrbracket = _ \end{array} \quad \begin{array}{l} \llbracket x \rrbracket = x \\ \llbracket c(p_1, \dots, p_n) \rrbracket = N \dots (N (Con \text{"c"}) \llbracket p_1 \rrbracket) \dots \llbracket p_n \rrbracket \end{array}$$

$$\left[\begin{array}{l} k \ p_{11} \cdots p_{1n} \longrightarrow e_1 \\ \vdots \\ k \ p_{m1} \cdots p_{mn} \longrightarrow e_m \end{array} \right] = \begin{array}{l} c \ x1 \ \dots \ xn = Abs_n \ f \ \mathbf{where} \\ f \ \llbracket p_{11} \rrbracket \ \dots \ \llbracket p_{1n} \rrbracket = \llbracket e_1 \rrbracket \\ \dots \\ f \ \llbracket p_{m1} \rrbracket \ \dots \ \llbracket p_{mn} \rrbracket = \llbracket e_m \rrbracket \\ _ = N \ \dots \ (N \ (Con \ \text{"c"}) \ x1) \ \dots \end{array}$$

Normalisation par évaluation

convertible :: Int → Code → Code → Bool

convertible n (Var x) (Var x') = x ≡ x'

convertible n (Con c) (Con c') = c ≡ c'

convertible n (Lam t) (Lam t') =
 convertible (n + 1) (t (Var n)) (t' (Var n))

convertible n (Pi ty1 ty2) (Pi ty3 ty4) =
 convertible n ty1 ty3 ∧
 convertible (n + 1) (ty2 (Var n)) (ty4 (Var n))

convertible n (N t1 t2) (N t3 t4) =
 convertible n t1 t3 ∧ *convertible* n t2 t4

convertible n Type Type = True

convertible n Kind Kind = True

convertible n _ _ = False

Conclusion

- ▶ Implantation d'un méta-vérificateur de types.
- ▶ Obtenu une famille de vérificateurs efficaces pour nombre de théories.
- ▶ Prochaines étapes: accélérer la compilation, lever certaines restrictions sur les règles de réécriture, ajouter associatif-commutatif.