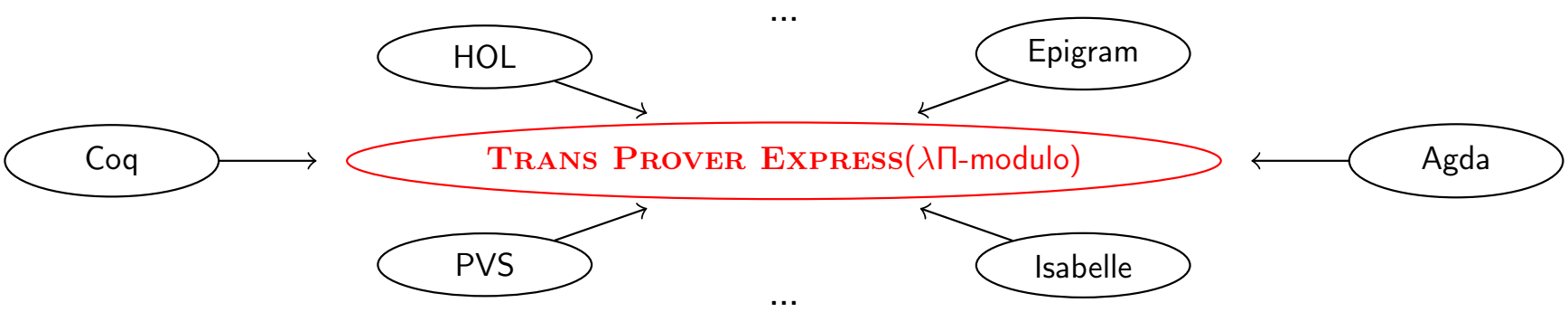
Trans Prover Express: un noyau pour $\lambda\Pi$-modulo

Mathieu Boespflug

École Polytechnique

14 septembre 2009


...
HOL
Epigram
Coq
Trans Prover Express(λΠ-modulo)
Agda
PVS
Isabelle
...

Grammaire

$$
\begin{array}{lcll}
\mathsf{Var} & \ni & x, y, z & \\
\mathsf{Term} & \ni & t, A, B & ::= x \mid \lambda x : A.\ t \mid \Pi x : A\ B \mid t\ t
\end{array}
$$

Règles d'inférence

$$s \in \{\mathit{Type}, \mathit{Kind}\}$$

$$\frac{}{[] \text{ well-formed}} \qquad \frac{\Gamma \vdash A : s}{\Gamma[x : A] \text{ well-formed}}$$

$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash \mathit{Type} : \mathit{Kind}} \qquad \frac{\Gamma \text{ well-formed} \qquad x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \qquad \Gamma[x : A] \vdash B : s}{\Gamma \vdash \Pi x : A\ B : s}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \qquad \Gamma[x : A] \vdash B : s \qquad \Gamma[x : A] \vdash t : B}{\Gamma \vdash \lambda x : A.\ t : \Pi x : A\ B}$$

$$\frac{\Gamma \vdash t : \Pi x : A\ B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : (u/x)B}$$

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

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

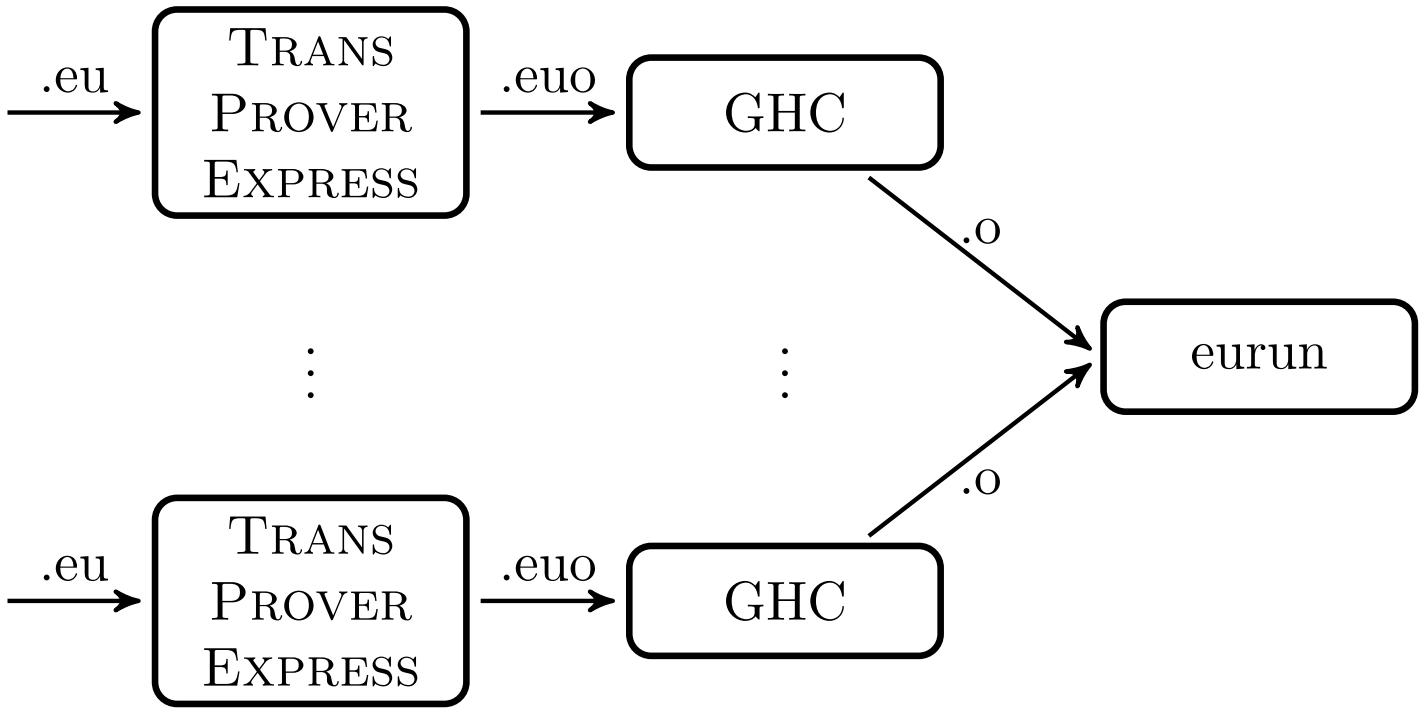
Enjeu: créer un outil pour vérifier le typage des termes dans un script de preuves exprimées $\lambda\Pi$-modulo.

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

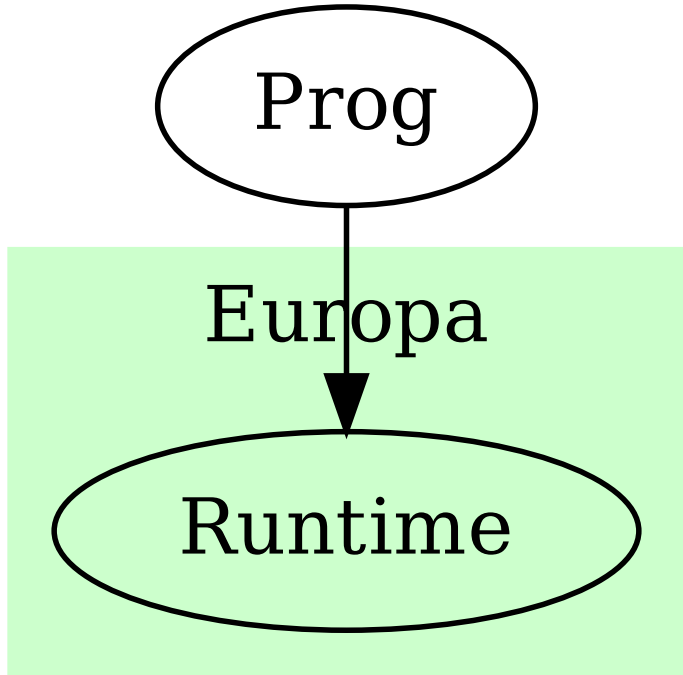
Idée 1: un générateur de vérificateurs de types spécialisés à la théorie considérée.
Idée 2: un générateur de vérificateurs de types spécialisés aux termes de preuves donnés.

Trans Prover Express: un méta-vérificateur

Vue macroscopique



Runtime



Interface de *TPE.Runtime*:

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

Représentations de termes

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

Représentation statique

$$
\begin{aligned}
&\textbf{data}\ \mathit{Term} = \mathit{TLam}\ \mathit{Term}\ (\mathit{Term} \to \mathit{Term}) \\
&\quad \mid \mathit{TPi}\ \mathit{Term}\ (\mathit{Term} \to \mathit{Term}) \\
&\quad \mid \mathit{TApp}\ \mathit{Term}\ \mathit{Term} \\
&\quad \mid \mathit{TType} \\
&\quad \mid \mathit{Box}\ \mathit{Code}\ \mathit{Code} \\
&\quad \mid \mathit{UBox}\ \mathit{Term}\ \mathit{Code}
\end{aligned}
$$

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

Représentation statique

$$
\begin{aligned}
\textbf{data}\ \mathit{Term} = {} & \mathit{TLam}\ \mathit{Term}\ (\mathit{Term} \to \mathit{Term}) \\
& \mid \mathit{TPi}\ \mathit{Term}\ (\mathit{Term} \to \mathit{Term}) \\
& \mid \mathit{TApp}\ \mathit{Term}\ \mathit{Term} \\
& \mid \mathit{TType} \\
& \mid \mathit{Box}\ \mathit{Code}\ \mathit{Code} \\
& \mid \mathit{UBox}\ \mathit{Term}\ \mathit{Code}
\end{aligned}
$$

- 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

$$\frac{}{t \vdash_n \boxed{t : A} : A} \qquad \frac{}{t \vdash_n \mathit{Type} : \mathit{Kind}}$$

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

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

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

Dérivation pour $nat : Type$:

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

Dérivation pour $O : nat$:

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

Dérivation pour $vec : \Pi x : nat\ Type$:

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

Dérivation pour $nil : vec\ O$:

$$\frac{\dfrac{}{vec_{ty} \vdash_n \boxed{\Pi nat\ f : vec} : \Pi nat\ f}}{nil_{ty} \vdash_n vec_{ty}\ \boxed{O : nat} : f\ O}\ nat \equiv nat$$

Le test de conversion

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

Représentation dynamique

$$
\begin{aligned}
&\textbf{data}\ \mathit{Code} = \mathit{Var}\ \mathit{Int} \\
&\quad \mid \mathit{Con}\ \mathit{String} \\
&\quad \mid \mathit{Lam}\ (\mathit{Code} \to \mathit{Code}) \\
&\quad \mid \mathit{Pi}\ \mathit{Code}\ (\mathit{Code} \to \mathit{Code}) \\
&\quad \mid \mathit{N}\ \mathit{Code}\ \mathit{Code} \\
&\quad \mid \mathit{Type} \\
&\quad \mid \mathit{Kind} \\
&\mathit{ap} :: \mathit{Code} \to \mathit{Code} \to \mathit{Code} \\
&\mathit{ap}\ (\mathit{Lam}\ f)\ t = f\ t \\
&\mathit{ap}\ t1\ t2 = \mathit{N}\ t1\ t2
\end{aligned}
$$

- Annotations de type sur les λ effacés.

Représentation dynamique: exemple

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

Représentation dynamique: exemple

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

$$ap\ (Abs\ (\lambda x \to Abs\ (\lambda y \to ap\ y\ x)))\ (Con\ \texttt{"z"})$$

Règles de réécriture

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

$$\begin{array}{rcl} [\![e]\!] & = & \mathtt{e} \\ [\![\_]\!] & = & \mathtt{\_} \end{array} \qquad \begin{array}{rcl} [\![x]\!] & = & \mathtt{x} \\ [\![c(p_1, \ldots, p_n)]\!] & = & N \ldots (N\ (Con\ \text{"c"})\ [\![p_1]\!]) \ldots [\![p_n]\!] \end{array}$$

$$\left[\!\!\left[\begin{array}{ccc} k\ p_{11} \cdots p_{1n} & \longrightarrow & e_1 \\ \vdots & & \vdots \\ k\ p_{m1} \cdots p_{mn} & \longrightarrow & e_m \end{array} \right]\!\!\right] =$$

$$\begin{array}{l} c\ x1 \ldots xn = Abs_n\ f\ \textbf{where} \\ \quad f\ [\![p_{11}]\!] \ldots [\![p_{1n}]\!] = [\![e_1]\!] \\ \quad \ldots \\ \quad f\ [\![p_{m1}]\!] \ldots [\![p_{mn}]\!] = [\![e_m]\!] \\ \quad \_ = N \ldots (N\ (Con\ \text{"c"})\ x1) \ldots \end{array}$$

Normalisation par évaluation

$$
\begin{aligned}
&\mathit{convertible} :: \mathit{Int} \to \mathit{Code} \to \mathit{Code} \to \mathit{Bool} \\
&\mathit{convertible}\ n\ (\mathit{Var}\ x)\ (\mathit{Var}\ x') = x \equiv x' \\
&\mathit{convertible}\ n\ (\mathit{Con}\ c)\ (\mathit{Con}\ c') = c \equiv c' \\
&\mathit{convertible}\ n\ (\mathit{Lam}\ t)\ (\mathit{Lam}\ t') = \\
&\quad \mathit{convertible}\ (n+1)\ (t\ (\mathit{Var}\ n))\ (t'\ (\mathit{Var}\ n)) \\
&\mathit{convertible}\ n\ (\mathit{Pi}\ \mathit{ty1}\ \mathit{ty2})\ (\mathit{Pi}\ \mathit{ty3}\ \mathit{ty4}) = \\
&\quad \mathit{convertible}\ n\ \mathit{ty1}\ \mathit{ty3} \wedge \\
&\quad \mathit{convertible}\ (n+1)\ (\mathit{ty2}\ (\mathit{Var}\ n))\ (\mathit{ty4}\ (\mathit{Var}\ n)) \\
&\mathit{convertible}\ n\ (N\ \mathit{t1}\ \mathit{t2})\ (N\ \mathit{t3}\ \mathit{t4}) = \\
&\quad \mathit{convertible}\ n\ \mathit{t1}\ \mathit{t3} \wedge \mathit{convertible}\ n\ \mathit{t2}\ \mathit{t4} \\
&\mathit{convertible}\ n\ \mathit{Type}\ \mathit{Type} = \mathit{True} \\
&\mathit{convertible}\ n\ \mathit{Kind}\ \mathit{Kind} = \mathit{True} \\
&\mathit{convertible}\ n\ \_\ \_ = \mathit{False}
\end{aligned}
$$

Conclusion

- Implantation d'un méta-vérificateur de types.
- 1256 lignes de code (hors commentaires).
- 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.

Encodage du calcul des constructions

```
Utype : Type.
Ukind : Type.

etype : Utype -> Type.
ekind : Ukind -> Type.

dottype : Ukind.
```

Encodage du calcul des constructions (suite)

```
dotpi1 : x : Utype -> y : (etype x -> Utype) -> Utype.
dotpi2 : x : Utype -> y : (etype x -> Ukind) -> Ukind.
dotpi3 : x : Ukind -> y : (ekind x -> Utype) -> Utype.
dotpi4 : x : Ukind -> y : (ekind x -> Ukind) -> Ukind.

[x:Utype, y : etype x -> Utype]
    etype (dotpi1 x y) --> w : etype x -> etype (y w).
[x:Ukind, y : ekind x -> Utype]
    etype (dotpi3 x y) --> w : ekind x -> etype (y w).

[] ekind dottype --> Utype.
[x:Utype, y : etype x -> Ukind]
    ekind (dotpi2 x y) --> w : etype x -> ekind (y w).
[x:Ukind, y : ekind x -> Ukind]
    ekind (dotpi4 x y) --> w : ekind x -> ekind (y w).
```

```
a : x : Utype -> y : etype x -> etype x.
[] a --> x : Utype => y : etype x => y.
```