# Cloisonnement des contenus calculatoire et logique du filtrage dépendant en théorie des types

Pierre Boutillier - équipe $\Pi r^2$
pierre.boutillier_AT_inria.fr

7 septembre 2010

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche PARIS - ROCQUENCOURT

# Le filtrage : du raisonnement par cas I

## ML

- Les types inductifs n'ont que des paramètres constants
- Le type attendu ne dépend pas de la branche

```
Inductive list (A : Type)
  : Type :=
 |Nil : list A
 |Cons : A → list A → list A
```

```
fix append A (x y : list A) : list A =>
match x with
 |Nil => y
 |Cons h t => Cons h (append A t y)
end.
```

# Avec des dépendances maintenant

```
Inductive vector (A : Type) : nat -> Type :=
 |Vnil : Vector A 0
 |Vcons : forall (a : A) (n : nat), vector A n → vector A (S n)
```

```
fix append A m n (x : vector A m) (y : vector A n)
  : vector A (m + n) :=
match x with
 |Vnil => y      (* : vector A n *)
 |Vcons h m0 t => Vcons h (m0 + n) (append A m0 n t y)
                  (* : vector A (S m0 + n) *)
end.
```

## Notation formelle

match t : $I\ \overrightarrow{u}$ with
$|C_1\ \overrightarrow{x}_1 : I\ \overrightarrow{v}_1 \Rightarrow s_1 : Q_1$
$\vdots$
$|C_n\ \overrightarrow{x}_n : I\ \overrightarrow{v}_n \Rightarrow s_n : Q_n$
end : R

## Types dépendants

On veut toujours R dans chaque cas mais on sait que $\left( \begin{array}{c} t = C_i\ \overrightarrow{x} \\ \overrightarrow{u = v_i} \end{array} [eqs] \right)$. Il y a cette fois une possible influence sur $Q_i$.
Utiliser les égalités est primordial pour le typage alors que c'est inutile pour le calcul.

# Comment alors typer sans faire de bruit pour le calcul ? I

## Coq

- L'utilisateur donne « une clause de retour » P qui doit être une abstraction de R selon $\overrightarrow{u}$ et t
- Le système s'en assure et vérifie que $Q_i = P\,\overrightarrow{v_i}\,(C_i\,\overrightarrow{x})$

\+ on a quasiment explicitement les preuves d'équivalences entre R et Q qui peuvent être quelconques

\- ces preuves d'égalités bloquent le calcul

## Outil Program (Matthieu Sozeau)

match t : $I\ \overrightarrow{u}$ as z in $I\ \overrightarrow{y}$ return $\overrightarrow{y = u}$ -> z = t -> R with
$|C\ \overrightarrow{x} : I\ \overrightarrow{v}$ => fun eqs => term_program eqs (s : Q)
end : R

# filtrage et $\beta$-expansion comme coercion de type

## Faire Agda dans Coq

# L'idée

$$\text{fix f } \overrightarrow{x} \text{ => t}$$

## La base

- terminaison assurée par la décroissance structurelle du $i$ème argument de f
- f est remplacé par t dans f $\overrightarrow{u}$ ssi $u_i = C\ \overrightarrow{v}$

## Jugement naïf

spécification $\mathcal{N}$on sous-terme, Sous-$\mathcal{T}$erme, $\mathcal{S}$ous-terme strict

environnement $E$ qui donne la spécification de chacune des variables libres (à l'origine $x_{j\neq i} = \mathcal{N}$, $x_i = \mathcal{T}$)

$$E \vdash t\ WF \qquad \frac{\overrightarrow{E; y := \mathcal{S} \vdash a\ WF} \qquad E(z) = \mathcal{T}}{E \vdash \texttt{match}\ z\ \texttt{with} \overrightarrow{|\, C\ \overrightarrow{y} \Rightarrow a}\ WF} \qquad \frac{E(z_i) = \mathcal{S}}{E \vdash f\, z_1 \ldots z_i\ WF}$$

# La pratique

1. fix plus m n => match m with
   |0 => n
   |S m' => (fun x => plus x n) m'
   end

$E \vdash t\ WF$ ssi $t \mapsto_\beta t'$ et $E \vdash t'\ WF$

## Sauf que

Coq < Fixpoint f (x : nat) : nat := (fun y => 3) (f x).
f is recursively defined (decreasing on 1st argument)
Coq < Eval compute in (f 2).
Stack overflow.

# Ma proposition

## Idée

- Avoir des jugements qui réalisent les substitutions explicitement

$E \vdash t \sim$

## Outils

Pile dans laquelle se trouve les arguments non appliqués

pour vérifier les arguments

- pour spécifier les variables nécessairement libres

## Clés

Beta

$$\frac{E; p \vdash u[t/x]\ WF \qquad \ldots}{E; t :: p \vdash \lambda x : T.u\ WF}$$

App

$$\frac{E; u :: p \vdash t\ WF \qquad E; \emptyset \vdash u \sim}{E; p \vdash t\ u\ WF}$$

Match

$$\frac{\ldots \qquad \overrightarrow{E \ldots; p \vdash t\ WF}}{E; p \vdash \texttt{match}\ u\ \texttt{with} \overrightarrow{| C\ \overrightarrow{x} \Rightarrow t}\ WF}$$

# Bilan

- nouvelle règle pour assurer la terminaison des points fixes plus rigoureuse et prenant en compte les généralisations
- bibliothèque `fin` et `vector` utilisant des clauses de retours plus compexe pour isoler la partie logique de la partie calculatoire
- Un algoritme exploitant mieux les types pour les tactiques de raisonnement par cas.

Avez vous des questions ?