De nouveaux outils pour Calculer avec les inductifs en Coq

Soutenance de thèse

Pierre Boutillier

sous la direction de

Hugo Herbelin

Univ Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126 CNRS, πr^2, INRIA Paris-Rocquencourt, F-75205 Paris, France

Mardi 18 Février 2014

Coq

- Un assistant de preuve
- à la fois système logique et langage de programmation
- qui s'inspire des découvertes des dernières décennies
- et est *the language of the year* selon l'ACM SIGPLAN

En ce qui concerne mon travail

- Un langage de programmation fonctionnel
- permettant de définir ses propres structures de données
- apportant des garanties fortes sur les programmes
- au moyen d'un noyau isolé.

Ma contribution

Dans l'environnement de programmation fourni à l'utilisateur

j'ai contribué à :

1. la machine de réduction des programmes
   (afin qu'elle donne des réponses syntaxiquement plus courtes)
2. le mécanisme d'analyse par cas
   (afin qu'il dépende moins d'un axiome supplémentaire)
3. le vérificateur de terminaison
   (dans l'espoir d'en faire une description plus formelle)

Comment définir un langage de programmation

Exemple : Les opérations arithmétiques

Syntaxe

$$m, n, p ::= \mathsf{O} \mid \mathsf{S}\ m \mid (m + n)$$

Sémantique

à petits pas $(\mathsf{S}\ m + n) \to \mathsf{S}\ (m + n)$ $\qquad (\mathsf{O} + m) \to m$

avec passage au contexte

$$\frac{m \to p}{\mathsf{S}\ m \to \mathsf{S}\ p} \qquad \frac{m \to p}{(m + n) \to (p + n)}$$

Comment définir un langage de programmation

Exemple : Les opérations arithmétiques

Syntaxe

$$m, n, p ::= \mathsf{O} \mid \mathsf{S}\ m \mid (m + n)$$

Sémantique

à petits pas $(\mathsf{S}\ m + n) \to \mathsf{S}\ (m + n)$ $\qquad$ $(\mathsf{O} + m) \to m$

avec passage au contexte

$$\frac{m \to p}{\mathsf{S}\ m \to \mathsf{S}\ p} \qquad \frac{m \to p}{(m + n) \to (p + n)}$$

Comment définir un langage de programmation

Exemple : Les opérations arithmétiques

Syntaxe

$$m, n, p ::= \mathsf{O} \mid \mathsf{S}\ m \mid (m + n)$$

Sémantique

à petits pas $(\mathsf{S}\ m + n) \to \mathsf{S}\ (m + n)$ $\qquad$ $(\mathsf{O} + m) \to m$

avec passage au contexte

$$\frac{m \to p}{\mathsf{S}\ m \to \mathsf{S}\ p} \qquad \frac{m \to p}{(m + n) \to (p + n)}$$

Des structures de données algébriques

par cas

```
Inductive binaire: Set :=
|Oui: binaire
|Non: binaire.
```

avec arguments

```
Inductive et (A B: Set): Set :=
|Paire: A → B → et A B.
```

```
Paire Oui Non
Paire Oui (Paire Oui Non)
```

récursifs

```
Inductive nat: Set :=
|O: nat
|S: nat → nat.
```

```
S (S (S O))
```

Des structures de données algébriques

par cas

```
Inductive binaire: Set :=
|Oui: binaire
|Non: binaire.
```

avec arguments

```
Inductive et (A B: Set): Set :=
|Paire: A → B → et A B.
```

```
Paire Oui Non
Paire Oui (Paire Oui Non)
```

récursifs

```
Inductive nat: Set :=
|O: nat
|S: nat → nat.
```

```
S (S (S O))
```

Des structures de données algébriques

par cas

```
Inductive binaire: Set :=
|Oui: binaire
|Non: binaire.
```

avec arguments

```
Inductive et (A B: Set): Set :=
|Paire: A → B → et A B.
```

Paire Oui Non
Paire Oui (Paire Oui Non)

récursifs

```
Inductive nat: Set :=
|O: nat
|S: nat → nat.
```

S (S (S O))

Des structures de données algébriques

par cas

```
Inductive binaire: Set :=
|Oui: binaire
|Non: binaire.
```

avec arguments

```
Inductive et (A B: Set): Set :=
|Paire: A → B → et A B.
```

Paire Oui Non
Paire Oui (Paire Oui Non)

récursifs

```
Inductive nat: Set :=
|O: nat
|S: nat → nat.
```

S (S (S O))

Des structures de données algébriques

par cas

```
Inductive binaire: Set :=
|Oui: binaire
|Non: binaire.
```

avec arguments

```
Inductive et (A B: Set): Set :=
|Paire: A → B → et A B.
```

Paire Oui Non
Paire Oui (Paire Oui Non)

récursifs

```
Inductive nat: Set :=
|O: nat
|S: nat → nat.
```

S (S (S O))

Fragment calculatoire du langage de Coq

Des termes

$$
\begin{array}{rcl}
t & ::= & x \mid \lambda x.t \mid t\ u \mid c \\
& \mid & \mathbf{C} \mid \texttt{fix}\ (f := t) \\
& \mid & \texttt{case}\ t\ \texttt{of} \\
& & \mid \mathbf{C} \Rightarrow u_1 \ldots \mid \mathbf{C} \Rightarrow u_s \\
& & \texttt{end}
\end{array}
$$

Discriminer selon le constructeur

$$
\begin{array}{l}
\texttt{case}\ x\ \texttt{of} \\
\mid \mathbf{Oui} \Rightarrow \mathbf{Non} \\
\mid \mathbf{Non} \Rightarrow \mathbf{Oui} \\
\texttt{end}
\end{array}
$$

$\lambda x.x$ $\qquad$ $\lambda x.\lambda y.x$ $\qquad$ $\lambda f.\lambda g.\lambda x.f\ x\ (g\ x)$

$$
\begin{array}{l}
\texttt{plus:=fix}\ (\texttt{pl} := \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} \\
\qquad \mid \mathbf{O} \Rightarrow n \\
\qquad \mid \mathbf{S} \Rightarrow \lambda m'.\mathbf{S}\ (pl\ m'\ n) \\
\qquad \texttt{end}
\end{array}
$$

Fragment calculatoire du langage de Coq

Des termes

$$
\begin{array}{rcl}
\mathtt{t} & ::= & \mathtt{x} \mid \lambda \mathtt{x}.\mathtt{t} \mid \mathtt{t}\ \mathtt{u} \mid \mathtt{c} \\
& \mid & \mathtt{C} \mid \mathtt{fix}\ (\mathtt{f} := \mathtt{t}) \\
& \mid & \mathtt{case}\ t\ \mathtt{of} \\
& & \mid \mathtt{C} \Rightarrow u_1 \ldots \mid \mathtt{C} \Rightarrow u_s \\
& & \mathtt{end}
\end{array}
$$

$\lambda \mathtt{x}.\mathtt{x}$	$\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}$	$\lambda \mathtt{f}.\lambda \mathtt{g}.\lambda \mathtt{x}.\mathtt{f}\ \mathtt{x}\ (\mathtt{g}\ \mathtt{x})$
$x \mapsto x$	$(x, y) \mapsto x$	$(f, g, x) \mapsto f(x, g(x))$

$$
\begin{array}{l}
\mathtt{plus} := \mathtt{fix}\ (\mathtt{pl} := \lambda \mathtt{m}.\lambda \mathtt{n}.\ \mathtt{case}\ m\ \mathtt{of} \\
\qquad \mid \mathtt{O} \Rightarrow n \\
\qquad \mid \mathtt{S} \Rightarrow \lambda m'.\mathtt{S}\ (pl\ m'\ n) \\
\qquad \mathtt{end}
\end{array}
$$

Fragment calculatoire du langage de Coq

Des termes

$$
\begin{array}{rcl}
t & ::= & x \mid \lambda x.t \mid t\ u \mid c \\
& \mid & \mathbf{C} \mid \texttt{fix}\ (f := t) \\
& \mid & \texttt{case}\ t\ \texttt{of} \\
& & \mid \mathbf{C} \Rightarrow u_1 \ldots \mid \mathbf{C} \Rightarrow u_s \\
& & \texttt{end}
\end{array}
$$

Un catalogue de définitions

$\Delta ::= \emptyset \mid \Delta, c := t$

$\mathsf{id} := \lambda x.x$ $\quad$ $\mathsf{proj} := \lambda x.\lambda y.x$ $\quad$ $\mathsf{opS} := \lambda f.\lambda g.\lambda x.f\ x\ (g\ x)$

$$
\begin{array}{l}
\mathsf{plus} := \texttt{fix}\ (pl := \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} \\
\qquad\qquad\qquad\qquad \mid \mathbf{O} \Rightarrow n \\
\qquad\qquad\qquad\qquad \mid \mathbf{S} \Rightarrow \lambda m'.\mathbf{S}\ (pl\ m'\ n) \\
\qquad\qquad\qquad\qquad \texttt{end}
\end{array}
$$

Une sémantique

β-réduction

$\Delta \vdash (\lambda x.t)\ u \mapsto_\beta t[u/x]$

δ-réduction

$\Delta, c{:=}t, \Delta' \vdash c \mapsto_\delta t$

ι-réduction

$\Delta \vdash \texttt{case}\ C_i\ u_1 \cdots u_n\ \texttt{of}\ |t_1 \cdots |t_m\ \texttt{end} \quad \mapsto_\iota \quad t_i\ u_1 \cdots u_n$

ϕ-réduction

$\Delta \vdash (\texttt{fix}\ (f := t))\ (C\ v_1 \cdots v_n) \quad \mapsto_\phi \quad t[\texttt{fix}\ (f := t)/f]\ (C\ v_1 \cdots v_n)$

Comment implanter le calcul dans Coq ?

A l'aide de machines abstraites

- Pas de règle de passage au contexte : le contexte est réifié par une *pile*
- Un état de la machine $\langle t \mid \pi \rangle$ est un terme « de tête » t en face d'une pile π
- La pile représente la continuation du calcul
- La réduction est la combinaison de l'*évaluation* vers un état de machine puis la *reconstruction* d'une valeur à partir de cet état

Coq contient plusieurs machines de réduction. Au dessus de l'une d'elle, Christine Paulin a construit un mécanisme (`simpl`) pour montrer à l'utilisateur des termes réduits plus courts.

Partie 1 : `fix` et `simpl` (grâce à une machine abstraite)

La machine abstraite

Evaluation en tête

$$\begin{array}{lcl} \langle x \mid \pi \rangle & \not\mapsto & \\ \langle t\ u \mid \pi \rangle & \mapsto & \langle t \mid APP(u), \pi \rangle \\ \langle \lambda x.t \mid APP(u), \pi \rangle & \mapsto & \langle t[u/x] \mid \pi \rangle \end{array}$$

$$\langle \lambda x.t \mid \pi \rangle \mapsto \langle t \mid VAR(x), \pi \rangle \text{ sinon}$$

Reconstruction d'un terme

$$\begin{array}{lcl} \langle t \mid APP(u), \pi \rangle & \Uparrow & \langle t\ u \mid \pi \rangle \\ \langle t \mid VAR(x), \pi \rangle & \Uparrow & \langle \lambda x.t \mid \pi \rangle \\ \langle t \mid \emptyset \rangle & \Uparrow & t \end{array}$$

Normalisation

$$\frac{\langle t \mid \emptyset \rangle \mapsto^* \langle u \mid \pi \rangle \not\mapsto \qquad \pi \Rightarrow \pi' \qquad \langle u \mid \pi' \rangle \Uparrow v}{t \Downarrow v}$$

$$\frac{t \Downarrow v \qquad \pi \Rightarrow \pi'}{APP(t), \pi \Rightarrow APP(v), \pi'} \qquad \frac{\pi \Rightarrow \pi'}{VAR(x), \pi \Rightarrow VAR(x), \pi'}$$

La machine abstraite

Evaluation en tête

$$\begin{array}{lcl} \langle x \mid \pi \rangle & \not\mapsto & \\ \langle t\ u \mid \pi \rangle & \mapsto & \langle t \mid APP(u), \pi \rangle \\ \langle \lambda x.t \mid APP(u), \pi \rangle & \mapsto & \langle t[u/x] \mid \pi \rangle \end{array}$$

$$\langle \lambda x.t \mid \pi \rangle \mapsto \langle t \mid VAR(x), \pi \rangle \text{ sinon}$$

Reconstruction d'un terme

$$\begin{array}{lcl} \langle t \mid APP(u), \pi \rangle & \Uparrow & \langle t\ u \mid \pi \rangle \\ \langle t \mid VAR(x), \pi \rangle & \Uparrow & \langle \lambda x.t \mid \pi \rangle \\ \langle t \mid \emptyset \rangle & \Uparrow & t \end{array}$$

Normalisation

$$\frac{\langle t \mid \emptyset \rangle \mapsto^* \langle u \mid \pi \rangle \not\mapsto \qquad \pi \Rightarrow \pi' \qquad \langle u \mid \pi' \rangle \Uparrow v}{t \Downarrow v}$$

$$\frac{t \Downarrow v \qquad \pi \Rightarrow \pi'}{APP(t), \pi \Rightarrow APP(v), \pi'} \qquad \frac{\pi \Rightarrow \pi'}{VAR(x), \pi \Rightarrow VAR(x), \pi'}$$

La machine abstraite

Evaluation en tête

$$\begin{array}{lll} \langle x \mid \pi \rangle & \not\mapsto & \\ \langle t\ u \mid \pi \rangle & \mapsto & \langle t \mid APP(u), \pi \rangle \\ \langle \lambda x.t \mid APP(u), \pi \rangle & \mapsto & \langle t[u/x] \mid \pi \rangle \end{array}$$

$$\langle \lambda x.t \mid \pi \rangle \mapsto \langle t \mid VAR(x), \pi \rangle \text{ sinon}$$

Reconstruction d'un terme

$$\begin{array}{lll} \langle t \mid APP(u), \pi \rangle & \Uparrow & \langle t\ u \mid \pi \rangle \\ \langle t \mid VAR(x), \pi \rangle & \Uparrow & \langle \lambda x.t \mid \pi \rangle \\ \langle t \mid \emptyset \rangle & \Uparrow & t \end{array}$$

Normalisation

$$\frac{\langle t \mid \emptyset \rangle \mapsto^* \langle u \mid \pi \rangle \not\mapsto \qquad \pi \Rightarrow \pi' \qquad \langle u \mid \pi' \rangle \Uparrow v}{t \Downarrow v}$$

$$\frac{t \Downarrow v \qquad \pi \Rightarrow \pi'}{APP(t), \pi \Rightarrow APP(v), \pi'} \qquad \frac{\pi \Rightarrow \pi'}{VAR(x), \pi \Rightarrow VAR(x), \pi'}$$

C'est parti pour un exemple

$$<((\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x)\ \lambda x0.\lambda y.x0)\ \lambda x1.\lambda y0.x1 \quad | \quad \emptyset >$$

C'est parti pour un exemple

$$<((\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x)\ \lambda x0.\lambda y.x0)\ \lambda x1.\lambda y0.x1 \quad | \quad \emptyset >$$

$$<(\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x)\ \lambda x0.\lambda y.x0 \quad | \quad \overset{\emptyset}{APP}(\lambda x.\lambda y.x) >$$

C'est parti pour un exemple

$$<(\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x)\ \lambda x0.\lambda y.x0 \quad | \quad \begin{matrix}\emptyset \\ APP(\lambda x.\lambda y.x)\end{matrix} >$$

$$<\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x \quad | \quad \begin{matrix}\emptyset \\ APP(\lambda x.\lambda y.x) \\ APP(\lambda x.\lambda y.x)\end{matrix} >$$

C'est parti pour un exemple

$$<\lambda f.\lambda g.\lambda x.((f)\ x)\ (g)\ x \quad | \quad \emptyset \quad APP(\lambda x.\lambda y.x) \quad APP(\lambda x.\lambda y.x) >$$

$$<\lambda g.\lambda x.((\lambda x0.\lambda y.x0)\ x)\ (g)\ x \quad | \quad \emptyset \quad APP(\lambda x.\lambda y.x) >$$

C'est parti pour un exemple

$<\lambda g.\lambda x.((\lambda x0.\lambda y.x0)\ x)\ (g)\ x \quad | \quad \emptyset \quad APP(\lambda x.\lambda y.x) >$

$<\lambda x.((\lambda x0.\lambda y.x0)\ x)\ (\lambda x1.\lambda y0.x1)\ x \quad | \quad \emptyset >$

C'est parti pour un exemple

$<\lambda x.((\lambda x0.\lambda y.x0)\ x)\ (\lambda x1.\lambda y0.x1)\ x \quad | \quad \emptyset >$

$$<((\lambda x.\lambda y.x)\ x)\ (\lambda x0.\lambda y0.x0)\ x \quad | \quad \begin{matrix}\emptyset \\ VAR(x)\end{matrix} >$$

C'est parti pour un exemple

$$<((\lambda x.\lambda y.x)\ x)\ (\lambda x0.\lambda y0.x0)\ x \quad | \quad \begin{matrix}\emptyset \\ VAR(x)\end{matrix} >$$

$$<(\lambda x.\lambda y.x)\ x \quad | \quad \begin{matrix}\emptyset \\ VAR(x) \\ APP((\lambda x.\lambda y.x)\ x)\end{matrix} >$$

C'est parti pour un exemple

$<(\lambda x.\lambda y.x)\ x \mid \emptyset \mid APP((\lambda x.\lambda y.x)\ x) :: VAR(x) :: \emptyset >$

$<\lambda x.\lambda y.x \mid APP(x) :: APP((\lambda x.\lambda y.x)\ x) :: VAR(x) :: \emptyset >$

C'est parti pour un exemple

$<\lambda x.\lambda y.x \mid \emptyset \; VAR(x) \; APP((\lambda x.\lambda y.x)\ x) \; APP(x) >$

$<\lambda y.x \mid \emptyset \; VAR(x) \; APP((\lambda x.\lambda y.x)\ x) >$

C'est parti pour un exemple

$$<\lambda y.x \quad | \quad \begin{matrix} \emptyset \\ VAR(x) \\ APP((\lambda x.\lambda y.x)\ x)\ x \end{matrix} >$$

$$<x \quad | \quad \begin{matrix} \emptyset \\ VAR(x) \end{matrix} >$$

C'est parti pour un exemple

$$<x \quad | \quad VAR(x) >^{\emptyset}$$

Après reconstruction

$$\lambda x.x$$

δ-expansion

$\Delta,\ \mathtt{c}:=\mathtt{t},\ \Delta' \vdash \langle c \mid \pi \rangle \mapsto \langle t \mid \pi \rangle$

Machinerie pour ι

$$\Delta \vdash \left\langle \begin{array}{l} \mathtt{case}\ u\ \mathtt{of} \\ |\mathbf{C} \Rightarrow t_1 \ldots |\mathbf{C} \Rightarrow t_s \\ \mathtt{end} \end{array} \mid \pi \right\rangle \mapsto \langle u \mid CASE(\ t_1 \ldots t_s\ ), \pi \rangle$$

$$\Delta \vdash \langle \mathbf{C}_i \mid APP(u)_1 \ldots APP(u)_n, CASE(\ t_1 \ldots t_s\ ), \pi \rangle \mapsto \langle t_i \mid APP(u)_1 \ldots APP(u)_n, \pi \rangle$$

Machinerie pour ϕ

$$\Delta \vdash \langle \mathtt{fix}\ (f := t) \mid APP(u), \pi \rangle \mapsto \langle u \mid FIX(f, t), \pi \rangle$$

$$\Delta \vdash \langle \mathbf{C} \mid APP(u)_1 \ldots APP(u)_n, FIX(f, t), \pi \rangle \mapsto \langle t[\mathtt{fix}\ (f := t)/f] \mid APP(\mathbf{C}\ u_1 \ldots u_n), \pi \rangle$$

Un nouvel exemple

$<\lambda a.\lambda b.((\text{plus})\ (\mathbf{S})\ a)\ b \quad | \quad \emptyset >$

Un nouvel exemple

$$<\lambda b.((\text{plus})\ (\mathbf{S})\ a)\ b \quad | \quad \overset{\emptyset}{VAR(a)} >$$

Un nouvel exemple

$$<((\text{plus})\ (\mathbf{S})\ a)\ b \quad | \quad \begin{matrix} \emptyset \\ VAR(a) \\ VAR(b) \end{matrix} >$$

Un nouvel exemple

$$<(\text{plus})\ (\mathbf{S})\ a \quad | \quad \begin{matrix} \emptyset \\ VAR(a) \\ VAR(b) \\ APP(b) \end{matrix} >$$

Un nouvel exemple

$$\begin{array}{lcr} & & \emptyset \\ & & VAR(a) \\ & & VAR(b) \\ & & APP(b) \\ <\text{plus} & | & APP((\mathbf{S})\ a)\ > \end{array}$$

Un nouvel exemple

Attention

$$
\begin{array}{lll}
\texttt{<fix } (\mathsf{pl} := \lambda \mathsf{m}.\lambda \mathsf{n}.\ \texttt{case } m \texttt{ of} &) & \emptyset \\
\quad |O \Rightarrow n & & VAR(a) \\
\quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n & & VAR(b) \\
\quad \texttt{end} & & APP(b) \\
 & | & APP((\mathbf{S})\ a) >
\end{array}
$$

Un nouvel exemple

Attention

$$
\begin{array}{lll}
< (\mathbf{S})\ a & | & FIX(pl, \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} \\
 & & \quad |O \Rightarrow n \\
 & & \quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n \\
 & & \texttt{end}
\end{array}
\qquad
\begin{array}{r}
\emptyset \\
VAR(a) \\
VAR(b) \\
APP(b) \\
) >
\end{array}
$$

Un nouvel exemple

Attention

$$\emptyset$$
$$VAR(a)$$
$$VAR(b)$$
$$APP(b)$$

$$FIX(pl, \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of}\ |O \Rightarrow n\ |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n\ \texttt{end})$$

$$<\mathbf{S} \quad | \quad APP(a) >$$

Un nouvel exemple

Aïe

$$
\begin{array}{l}
<\lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} \\
\qquad |O \Rightarrow n \\
\qquad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \lambda m0.\lambda n0.\ \texttt{case}\ m0\ \texttt{of} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad |O \Rightarrow n0 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad |\mathbf{S} \Rightarrow \lambda m'0.(\mathbf{S})\ ((pl)\ m'0)\ n \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{end} \\
\qquad \texttt{end} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \emptyset \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad VAR(a) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad VAR(b) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad APP(b) \\
| \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad APP((\mathbf{S})\ a) >
\end{array}
$$

Un nouvel exemple

Aïe

$$
\begin{array}{l}
<\lambda \mathrm{m}.\lambda \mathrm{n}.\ \texttt{case}\ m\ \texttt{of} \\
\qquad |O \Rightarrow n \\
\qquad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \ldots))\ m')\ n \\
\qquad \texttt{end}
\end{array}
\quad \Big| \quad
\begin{array}{r}
\emptyset \\
VAR(a) \\
VAR(b) \\
APP(b) \\
APP((\mathbf{S})\ a)\ >
\end{array}
$$

Un nouvel exemple

Aïe

$$
\begin{array}{l}
<\lambda \mathrm{n}.\ \texttt{case}\ (\mathbf{S})\ a\ \texttt{of} \\
\quad |O \Rightarrow n \\
\quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \lambda m.\lambda n0.\ldots))\ m')\ n \\
\quad \texttt{end}
\end{array}
\quad \Big| \quad
\begin{array}{r}
\emptyset \\
VAR(a) \\
VAR(b) \\
APP(b) >
\end{array}
$$

Un nouvel exemple

Aïe

$$
\begin{array}{lll}
< & \texttt{case } (\mathbf{S})\ a\ \texttt{of} & \mid \quad \emptyset \\
 & \mid O \Rightarrow b & \quad\ \mathrm{VAR}(a) \\
 & \mid \mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix } (pl := \lambda m.\lambda n. \ldots))\ m')\ b & \quad\ \mathrm{VAR}(b) > \\
 & \texttt{end} &
\end{array}
$$

Un nouvel exemple

Aïe

$$
\begin{array}{lll}
 & & \emptyset \\
 & & VAR(a) \\
 & & VAR(b) \\
<(\mathbf{S})\ a \quad | & \quad CASE(\ |O \Rightarrow b &) > \\
 & \quad\ |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \lambda m.\lambda n. \ldots))\ m')\ b &
\end{array}
$$

Un nouvel exemple

Aïe

$$
\begin{array}{rl}
 & \emptyset \\
 & VAR(a) \\
 & VAR(b) \\
CASE(\ |O \Rightarrow b &) \\
\quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \lambda m.\lambda n. \ldots))\ m')\ b & \\
<\mathbf{S} \quad | & APP(a) >
\end{array}
$$

Un nouvel exemple

Aïe

$$<\lambda m'.(\mathbf{S})\ ((\texttt{fix}\ \ (pl := \lambda m.\lambda n. \ldots))\ m')\ b \quad | \quad \begin{matrix} \emptyset \\ VAR(a) \\ VAR(b) \\ APP(a) \end{matrix} >$$

Un nouvel exemple

Aïe

$$<(\mathbf{S})\ ((\texttt{fix}\ \ (\mathsf{pl} := \lambda \mathsf{m}.\lambda \mathsf{n}. \ldots))\ \mathsf{a})\ \mathsf{b} \quad | \quad \begin{matrix} \emptyset \\ VAR(a) \\ VAR(b) \end{matrix} >$$

Un nouvel exemple

$$
\begin{array}{lll}
 & & \emptyset \\
 & & VAR(a) \\
 & & VAR(b) \\
\mathbf{S} \quad | \quad APP(((\texttt{fix}\ (pl := \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} & &))\ a)\ b) \\
 & \quad | O \Rightarrow n & \\
 & \quad | \mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n & \\
 & \texttt{end} &
\end{array}
$$

Un nouvel exemple

$$
\begin{array}{lll}
 & & \emptyset \\
 & & VAR(a) \\
 & & VAR(b) \\
\mathbf{S} \quad | \quad APP(((\texttt{fix}\ (pl := \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} & &))\ a)\ b) \\
 & \quad |O \Rightarrow n & \\
 & \quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n & \\
 & \texttt{end} &
\end{array}
$$

Après reconstruction

$$
\begin{array}{l}
\lambda a.\lambda b.\mathbf{S}\ (\texttt{fix}\ (pl := \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of}\ )\ a\ b) \\
\quad |\mathbf{O} \Rightarrow n \\
\quad |\mathbf{S} \Rightarrow \lambda m'.\mathbf{S}\ (pl\ m'\ n) \\
\texttt{end}
\end{array}
$$

Catastrophe

Ce qu'on aurait aimé obtenir

$$\texttt{<S} \quad | \quad \emptyset \; VAR(a) \; VAR(b) \; APP((plus\ a)\ b)\ \texttt{>}$$

Une constante globale plutôt que sa définition

- plus petite donc lisible directement (pour l'utilisateur)
- plus compacte (pour le système ensuite)

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$\lambda a.\lambda b.((\text{plus})\ (\mathbf{S})\ a)\ b$ | | $\emptyset$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

λb.((plus) (S) a) b | | $\emptyset$ | | $VAR(a)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

((plus) (**S**) a) b | | $\emptyset$
$VAR(a)$
$VAR(b)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

(plus) (S) a

| |

$\emptyset$
$VAR(a)$
$VAR(b)$
$APP(b)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

plus | | $\emptyset$
$VAR(a)$
$VAR(b)$
$APP(b)$
$APP((\mathbf{S})\ a)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$\emptyset$$
$$VAR(a)$$
$$VAR(b)$$
$$APP(b)$$

$$\texttt{fix}\ (\mathsf{pl} := \lambda \mathsf{m}.\lambda \mathsf{n}.\ \texttt{case}\ m\ \texttt{of} \quad) \quad | \ \mathsf{plus} \ | \quad APP((\mathbf{S})\ a)$$
$$|O \Rightarrow n$$
$$|\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n$$
$$\texttt{end}$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$
\begin{array}{lll}
 & & \emptyset \\
 & & VAR(a) \\
 & & VAR(b) \\
 & & APP(b) \\
(\mathbf{S})\ a & \mid\mid & FIX_{plus}(pl, \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} \\
 & & \mid O \Rightarrow n \\
 & & \mid \mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n \\
 & & \texttt{end}
\end{array}
\quad)
$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$
\begin{array}{lll}
 & & \emptyset \\
 & & VAR(a) \\
 & & VAR(b) \\
 & & APP(b) \\
 & FIX_{plus}(pl, \lambda m.\lambda n.\ \texttt{case}\ m\ \texttt{of} &) \\
 & \quad |O \Rightarrow n & \\
 & \quad |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((pl)\ m')\ n & \\
 & \texttt{end} & \\
\mathbf{S} \quad || & & APP(a)
\end{array}
$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

λm.λn. case m of
$|O \Rightarrow n$
$|\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((plus)\ m')\ n$
end

	$\emptyset$
	$VAR(a)$
	$VAR(b)$
	$APP(b)$
	$APP((\mathbf{S})\ a)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$\lambda n.\ \texttt{case}\ (\mathbf{S})\ a\ \texttt{of}\ |\ O \Rightarrow n\ |\ \mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((plus)\ m')\ n\ \texttt{end} \quad | \ | \quad \begin{array}{r} \emptyset \\ VAR(a) \\ VAR(b) \\ APP(b) \end{array}$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

```
case (S) a of
|O⇒ b
|S⇒ λm'.(S) ((plus) m') b
end
```

$\emptyset$

$VAR(a)$

$VAR(b)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$(\mathbf{S})\ a \quad || \quad CASE(\ |O \Rightarrow b \quad)$$
$$|\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((plus)\ m')\ b$$

$$\emptyset$$
$$VAR(a)$$
$$VAR(b)$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$\begin{array}{rl} & \emptyset \\ & VAR(a) \\ & VAR(b) \\ CASE(\ & |O \Rightarrow b \qquad\qquad\qquad\qquad\qquad\qquad\ ) \\ & |\mathbf{S} \Rightarrow \lambda m'.(\mathbf{S})\ ((plus)\ m')\ b \\ \mathbf{S} \qquad\qquad\qquad\qquad\quad || & APP(a) \end{array}$$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

		$\emptyset$
		$VAR(a)$
		$VAR(b)$
λm'.(**S**) ((plus) m') b	| |	$APP(a)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

(**S**) ((plus) a) b | | $\emptyset$
$VAR(a)$
$VAR(b)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

S | | $\emptyset$

$VAR(a)$

$VAR(b)$

$APP(((plus)\ a)\ b)$

Une nouvelle machine

Conserver le nom des constantes dépliées pour les réutiliser

$$\begin{array}{rcr} & & \emptyset \\ & & VAR(a) \\ & & VAR(b) \\ \mathbf{S} & || & APP(((plus)\ a)\ b) \end{array}$$

Attention à choisir la constante à la reconstruction

$$\langle plus\ a\ b \mid\mid \emptyset \rangle \rightarrow^* \langle a \mid\mid FIX_{plus}(plus, \ldots), APP(b), \emptyset \rangle$$

Repliages

Le fonctionnement

Il faut penser à

- Des constantes avec paramètres map:=λf.`fix` (aux := ...)
- Des cascades de constantes sum:=fold plus

Pile de repliage

type cst_stk = (cst * term list * term list) list

- Le nom de la constante * La liste des paramètres * La liste des arguments
- Si $\langle t \mid \Psi \mid \pi \rangle$, pour tout (c,$p_1 \ldots p_n, u_1 \ldots u_s$)$\in \Psi$, c $p_1 \ldots p_n$ = t $u_1 \ldots u_s$.

Résumé des apports de la nouvelle approche

Pratiquement

Nous n'avons plus les bugs :

~~#1827: simpl diverges on deep expressions~~
~~#2312: [simpl] doesn't reduce [fix...with]~~
~~#2972: Opaque terms can be reduced by simpl~~

Formellement

On a le résultat que le terme obtenu se $\beta\delta$ réduit vers la forme $\beta\delta\iota\phi$ normale du terme d'origine.

Partie 2 : Compilation du filtrage dépendant

Partie 2 : Compilation du filtrage dépendant [1]

1. Attachez votre ceinture, ça secoue pour les non initiés sur la deuxième moitié

Où sont les garanties sur le calcul ?

Calculs infinis

$\emptyset \vdash (\lambda x.x\ x)\ (\lambda x.x\ x) \mapsto (\lambda x.x\ x)\ (\lambda x.x\ x)$

Blocage

```
case Oui of
|Paire⇒λx y.y
end
```

Peut-on s'assurer que ces programmes ne seront jamais exécutés ?

Les types

$$S, T ::= \textbf{binaire} \mid \textbf{nat} \mid \ldots \mid S \to T$$

Des règles de bonne formation d'un programme

$$\dfrac{\dfrac{\dfrac{}{y : \textbf{int}, x : \textbf{nat}, \emptyset \vdash x \in \textbf{nat}}}{x : \textbf{nat}, \emptyset \vdash \lambda y.x \in \textbf{int} \to \textbf{nat}}}{\emptyset \vdash \lambda x.\lambda y.x \in \textbf{nat} \to \textbf{int} \to \textbf{nat}}$$

$\Gamma \vdash \mathsf{opS} \in (A \to B \to C) \to (A \to B) \to A \to C$

Le produit dépendant

$$A \to B$$

$$\forall x : A, B(x)$$

$$\forall b : \textbf{binaire}, \quad \begin{array}{l} \texttt{case } b \texttt{ of} \\ \quad \mid \textbf{Oui} \to \textbf{nat} \\ \quad \mid \textbf{Non} \to \textbf{binaire} \\ \texttt{end} \end{array}$$

Le langage de Coq avec annotation de type

$$
\begin{array}{c}
u, t, S, T ::= x \mid c \mid \lambda(x : T) \Rightarrow t \mid t\ u \mid \mathbf{C} \\
\mid \texttt{case}\ t\ \texttt{predicate}\ P\ \texttt{of}\ t_1 \cdots t_s\ \texttt{end} \\
\mid \texttt{fix}_i\ (f : T := t) \\
\mid \forall(x : S), T \mid s \mid \mathbf{I}
\end{array}
$$

Figure : Grammaire des termes (mini-CCI)

Des familles inductives

Prédicat de parité sur les entiers

```
Inductive even : nat → Set :=
| even_O  : even O
| even_SS : ∀(n : nat) (_ : even n), even (S (S n)).
```

Les listes à *n* éléments

```
Inductive Vect A : nat → Set :=
| Vnil  : Vect A O
| Vcons : ∀(n : nat)(h : A), Vect A n → Vect A (S n).
```

ex_falso_quodlibet

```
Inductive False : Set :=.
```

Analyse de cas raffinée

L'égalité

```
Inductive eq (A : Type) (a : A) : A → Type :=
| eq_refl :eq A a a.
```

Brique élémentaire de la réécriture

```
Definition rew (A : Type) (P : A → Type) (a b : A)
(p : P a) (e : eq A a b) : P b :=
case e predicate λ(c : A) (e' : eq A a c) . P c of
|eq_refl⇒p
end.
```

Prédicat de retour

branche $(\lambda(c : A)\ (e' : \mathbf{eq}\ A\ a\ c) \Rightarrow P\ c)\ a\ (\mathbf{eq}_{\mathbf{refl}}\ A\ a)$

extérieur $(\lambda(c : A)\ (e' : \mathbf{eq}\ A\ a\ c) \Rightarrow P\ c)\ b\ e$

Analyse de cas raffinée

L'égalité

```
Inductive eq (A : Type) (a : A) : A → Type :=
| eq_refl :eq A a a.
```

Brique élémentaire de la réécriture

```
Definition rew (A : Type) (P : A → Type) (a b : A)
(p : P a) (e : eq A a b) : P b :=
case e[ : _ = b] predicate λ(c : A) (e' : eq A a c) . P c of
|eq_refl[ :_ = a]⇒p
end.
```

Prédicat de retour

branche $(\lambda(c : A)\ (e' : \mathbf{eq}\ A\ a\ c) \Rightarrow P\ c)\ a\ (\mathbf{eq_{refl}}\ A\ a)$

extérieur $(\lambda(c : A)\ (e' : \mathbf{eq}\ A\ a\ c) \Rightarrow P\ c)\ b\ e$

Du filtrage

En programmation fonctionnelle

Exemple d'inspection simultanée et en profondeur :
n est il la moitié de m ?

```
Fixpoint beq_half (m n : nat) : binaire :=
match m, n with
| O, O => Oui
| S (S i), S j => beq_half i j
| _ , _ => Non
end.
```

Propriétés clés des constructeurs

discrimination $\forall(n : \mathbf{nat}), S\ n = O \rightarrow \mathbf{False}$

inversibilité $\forall(m\ n : \mathbf{nat}), S\ m = S\ n \rightarrow m = n$

acyclicité $\forall(n : \mathbf{nat}), S\ n = n \rightarrow \mathbf{False}$

vrai pour tous les $S\ (S \ldots (S\ n)) = n$.

Il est possible de construire mécaniquement ces preuves. (McBride - McKinna - Goguen)

Du filtrage

avec types dépendants (Agda)

De l'analyse de couverture

```
Definition inv_even n (e : even (S (S n))) :
  even n  :=
match e with
| even_O ⇒ !
| even_SS m e' ⇒ e'
end.
```

par unification premier ordre

$O == S\ (S\ n) \mapsto \bot$

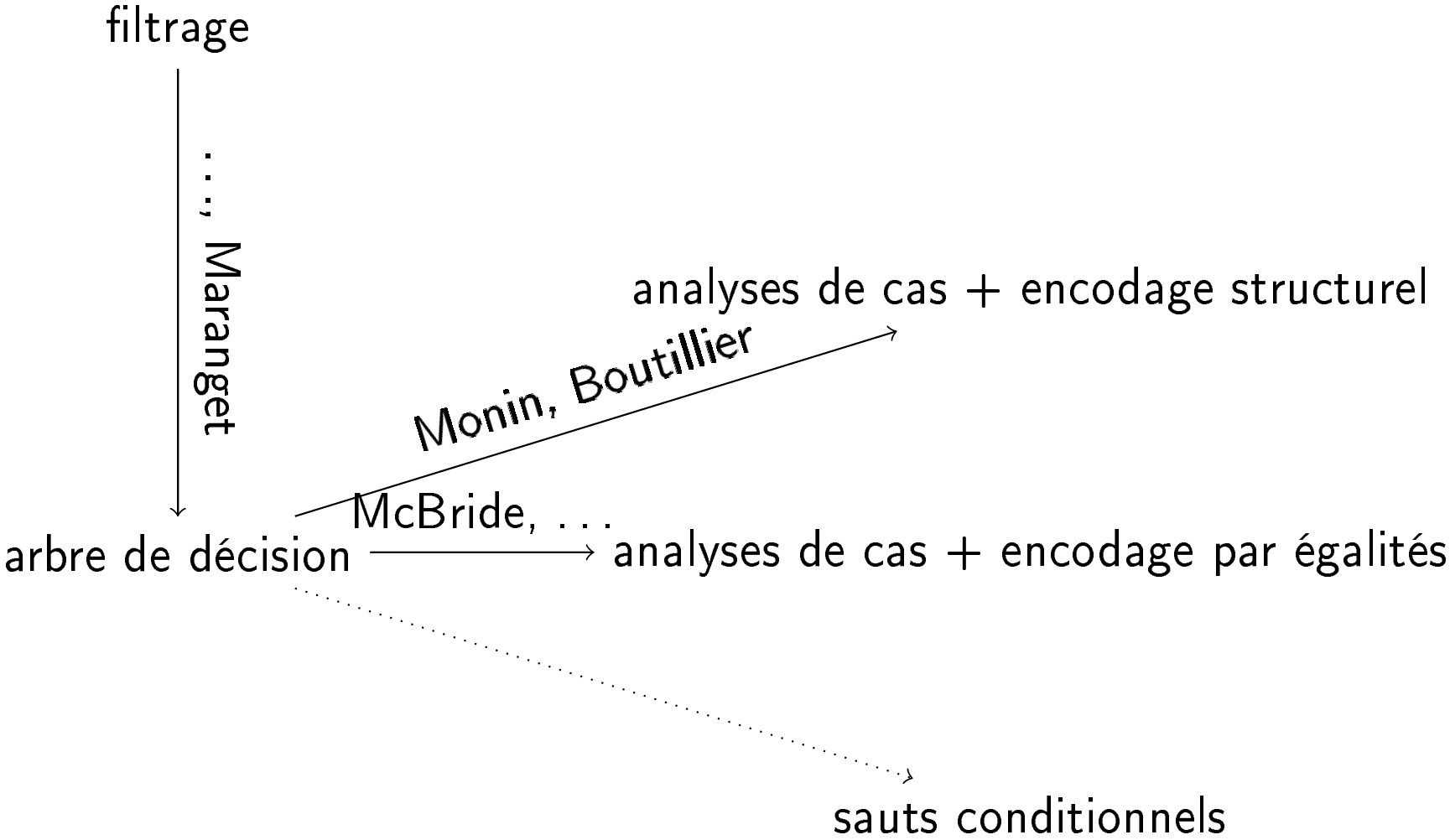
$S\ (S\ m) == S\ (S\ n) \mapsto$

$m == n \mapsto$

$m := n$

```
even_map2 : ∀ (n : nat) ↦ even n ↦ even n ↦ _
even_map2 .O even_O even_O = ?
even_map2 .(S (S m)) (even_SS .m e1) (even_SS m e2) = ?
```

Du filtrage à l'analyse de cas



Réconcilier Coq et filtrage

Implanté dans Equations par Mathieu Sozeau

```
Definition even_inv (E : even (S (S O))) :
  eq (even (S (S O))) E (even_SS O even_O) :=
case E
predicate λ(m : nat) (H : even m) . (S (S O)) = m →
    eq (even (S (S O))) E (even_SS O even_O)
of
|even_O⇒λ(e1 : (S (S O)) = O) . ...
|even_SS⇒λ(n : nat) (E' : even n) . λ(e1 : (S (S O)) = (S (S n))) .
...
end (eq_refl (S (S O))).
```

Réconcilier Coq et filtrage

Implanté dans Equations par Mathieu Sozeau

```
Definition even_inv (E : even (S (S O))) :
  eq (even (S (S O))) E (even_SS O even_O) :=
case E
predicate λ(m : nat) (H : even m) . (S (S O)) = m →
    E even (S (S O)) ≡even m H → eq (even (S (S O))) E (even_SS O
even_O)
of
|even_O⇒λ(e1 : (S (S O)) = O)
(e2 : E even (s (s o)) ≡even O even_O) . ...
|even_SS⇒λ(n : nat) (E' : even n) . λ(e1 : (S (S O)) = (S (S n)))
    (e2 : E even (s (s o)) ≡even (S (S n)) (even_SS n E')) . ...
end (eq_refl (S (S O))) (JMeq_refl (even (S (S O))) E).
```

L'égalité hétérogène

Définition de l'égalité hétérogène

```
Inductive JMeq (A : Type) (a : A) : ∀(B : Type) (b : B),
Type :=
  | JMeq_refl :JMeq A a A a.
```

Egalité modulo réécriture de type

```
Axiom JMeq_eq :
∀(A:Set) (a b:A), JMeq A a A b → Eq A a b.
Axiom eq_rect_eq :
  ∀(U : Type) (p : U) (Q : U → Type) (x : Q p) (h : p = p),
    x = rew U Q p p x h.
```

Coupures commutatives

```
Fixpoint plus (m n : nat) : nat :=
case n
 predicate λ(_ : nat) . nat of
 |O⇒(λ(m' : nat) . m') m
 |S⇒λ(n' : nat) . (λ(m' : nat) . S (plus m' n')) m
end.
```

```
Fixpoint plus (m n : nat) : nat :=
case n
 predicate λ(_ : nat) . nat → nat of
 |O⇒λ(m' : nat) . m'
 |S⇒λ(n' m' : nat) . S (plus m' n')
end m.
```

Coupures entrelacées

```
Definition even_inv (n : nat) (e1 e2 : even n) : ... :=
case e2
 predicate λ(m : nat) (_ : even m) . ... of
 |even_0⇒...
 |even_SS⇒λ(m' : nat) (e : even (S (S m'))) . ...
end.
```

```
Definition even_inv (n : nat) (e1 e2 : [even n]) : ... :=
case e2
 predicate λ(m : nat) (_ : even m) . even m → ... of
 |even_0⇒λ(e1' : [even O]) . ...
 |even_SS⇒λ(m' : nat) (e e1' : [even (S (S m'))]) . ...
end e1.
```

Branches impossibles

```
Definition diag_even1 (n : nat) (e : even n) : Type :=
case n predicate λ(_ : nat) . Type of
|O⇒binaire
|S⇒λ(m : nat) . case m predicate λ(_ : nat) . Type of
  |O⇒False
  |S⇒λ(_ : nat) . binaire
end
end.
Definition even1 (H : even (S O)) : False :=
case H
predicate λ(n : nat) (e : even n) . diag_even1 n e of
 |even_O⇒Oui
 |even_SS⇒λ(m : nat) . Non
end.
```

Difficile interaction avec les vérificateurs de terminaison

- La relation sous-terme doit être conservée par coupures entrelacées
- La présence de branches impossibles ne doit pas polluer la relation sous-terme

Les règles actuelles pour autoriser cela induisent que :
Pour tout inductif I,

$$\forall(B : \mathsf{Type}), \mathsf{I} \to ((\mathbf{False} \to B) = \mathsf{I}) \to \mathbf{False}$$

Ma contribution

Dans l'environnement de programmation fourni à l'utilisateur

j'ai contribué à :

1. la machine de réduction des programmes
   (afin qu'elle donne des réponses syntaxiquement plus courtes)
2. le mécanisme d'analyse par cas
   (afin qu'il dépende moins d'un axiome supplémentaire)
3. le vérificateur de terminaison
   (dans l'espoir d'en faire une description plus formelle)

Difficile interaction avec les vérificateur de terminaison

Si v est un sous terme d'un argument récursif dans la définition du point fixe f, le terme f (Vhd v) est un appel récursif admissible pour la définition

```
Definition Vhd (A : Type) (n : nat) (v : vector A (S n)) : A :=
let diag := λ(n : nat) ⇒
  case n predicate λ(n' : nat) ⇒ vector A n' → Type of
  |O:λ(_ : vector O) ⇒ False → True
  |S:λ(m : nat) ⇒ λ(_ : vector (S m)) ⇒ A
  end in
case v predicate diag of
|Vnil:λ(x : False) ⇒ case x predicate λ(_ : False) ⇒ True of
end
|Vcons:λ(n : nat) (h : A) (t : vector A n) ⇒ h
end.
```

Quelle primitive pour représenter l'analyse de cas ?

Première parenthèse

Avec des abstractions

swap:=λx. case x of
$|\mathbf{G} : \lambda x.\mathbf{D}\ x$
$|\mathbf{D} : \lambda y.\mathbf{G}\ y$
end

Avec des projections

swap:=λx. case x of
$|\mathbf{G} : \mathbf{D}\ \diamond_1(x)$
$|\mathbf{D} : \mathbf{G}\ \diamond_1(y)$
end

C'est qui le plus fort ?

type I = A of I | B | C of nat match x with (| A y -> y | _ -> x).

abstraction

case x of
$|\mathbf{A} : \lambda y.y|\mathbf{B} : x|\mathbf{C} : \lambda y.x$
end

projection

case x of
$|\mathbf{A} : \diamond_1(y)|\mathbf{B} : x|\mathbf{C} : x$
end

Pour le typage des record aussi, c'est gagnant : $\diamond_2$(y) vs proj2 A (λx.B) y

Garde structurelle

Propriété assurant que toutes les fonctions reccursives ont un argument reccursif.

Spécifications

$<$: Sous terme $\quad \leq$: Élément de décroissance $\quad \bot$: Quelquonque

Objectif

$$\frac{\ldots \qquad \overrightarrow{d \blacktriangleright \bot}, x \blacktriangleright \leq \vdash_f \ t \ \#}{\Delta \vdash \texttt{fix} \ (f := \overrightarrow{d}\, x) T\, t \in T}$$

Fix
$$\frac{\Gamma \vdash_f \ u \ \#_1 \ldots \Gamma \vdash_f \ u \ \#_s \qquad \Gamma \vdash_f \ t \ \# \qquad \Gamma \vdash_f \ t \blacktriangleright <}{\Gamma \vdash_f \ f \ \overrightarrow{u} \ t \ \#}$$

Sous Terme

$$\boxed{\Gamma \vdash_f \ t \blacktriangleright o}$$

Controle PointFixe

$$\boxed{\Gamma \vdash_f \ t \ \#}$$

Mécanique

MatchSpec

$$\frac{\Gamma, \overrightarrow{x \blacktriangleright \|a\|} \vdash_f u_i \blacktriangleright a_{i1} \ldots \Gamma, \overrightarrow{x \blacktriangleright \|a\|} \vdash_f u_i \blacktriangleright a_{is} \qquad \Gamma \vdash_f t \blacktriangleright a \qquad \bigwedge_i a_i = b}{\Gamma \vdash_f \texttt{case}\ t\ \texttt{of}\ |\, C_i\, \overrightarrow{x} \Rightarrow u_{i1} \ldots |\, C_i\, \overrightarrow{x} \Rightarrow u_{is}\ \texttt{end} \blacktriangleright b}$$

FixSpec

$$\frac{\Gamma \vdash_f v \blacktriangleright b \qquad \Gamma, g \blacktriangleright \bot, \overrightarrow{d \blacktriangleright \bot}, x \blacktriangleright b \vdash_f u \blacktriangleright a}{\Gamma \vdash_f \left(\texttt{fix}\ (g := \overrightarrow{d}\ x)\, T'\, u\right)\ \overrightarrow{t}\ v \blacktriangleright a}$$

Les coupures

Limite

```
let app_pred f x = case x of
|O⇒ O
|S n⇒ f n end
```

```
fix (div2 := x)case x of
|O⇒ O
|S n⇒ app_pred div2 n
 end
```

```
fix (div2 := x)case x of
|O⇒ O
|S n⇒ case n of |O⇒ O |S n'⇒ div2 n' end
 end
```

- Souvenez vous que vérifier la garde sur la forme normale de tête autorise bad.

Conséquence

$$\boxed{(\Gamma,\ \sigma) \vdash_f s \mid t \blacktriangleright o} \qquad \boxed{(\Gamma,\ \sigma) \vdash_f s \mid t\ \#}$$

App

$$\frac{(\Gamma,\ \sigma) \vdash_f (\Gamma, \sigma, v) :: s \mid u\ \#}{(\Gamma,\ \sigma) \vdash_f s \mid u\ v\ \#}\ \text{sanity}$$

Def

$$\frac{\sigma(y) = (\Gamma', \sigma', t) \qquad (\Gamma',\ \sigma') \vdash_f s \mid t\ \#}{(\Gamma,\ \sigma) \vdash_f s \mid y\ \#}$$

Pop

$$\frac{(\Gamma,\ \sigma) \vdash_f \emptyset \mid c \blacktriangleright a \qquad (\Gamma, x \blacktriangleright a,\ \sigma, x \mapsto c) \vdash_f s \mid u\ \#}{(\Gamma,\ \sigma) \vdash_f c :: s \mid \lambda x.u\ \#}$$

Sanity : Sûr dans un contexte vide

$\boxed{(\Gamma,\ \sigma) \vdash_f s \mid v \approx}$

nouveauté

$\diamond$: Terme neutre

$$\text{Lam}\ \frac{(\Gamma, x \blacktriangleright \diamond,\ \sigma, x) \vdash_f \emptyset \mid t \approx}{(\Gamma,\ \sigma) \vdash_f \emptyset \mid \lambda x : T.t \approx}$$

$$\text{StrongApp}\ \frac{(\Gamma,\ \sigma) \vdash_f \emptyset \mid v \#\qquad (\Gamma,\ \sigma) \vdash_f \sqcup :: s \mid u \#}{(\Gamma,\ \sigma) \vdash_f s \mid u\ v \#}$$

Attention aux constructeurs

Le pire

```
fix (loop := n)case n of
|O⇒ True
|S O⇒ False
|S (S m)⇒ loop (S m)
 end
```

Le meilleur

```
case S n of
|O⇒ evil
|S m⇒ good
 end
```

Coupures d'inductifs

Plus de spécification encore

$C\,\overrightarrow{a}$: Constructeur dont les arguments ont pour propriété $\overrightarrow{a}$

Iota

$$\frac{\begin{array}{c}\Gamma \vdash_f u \blacktriangleright C_i\ a_1 \ldots a_s \\ \Gamma, x \blacktriangleright a_1 \ldots x \blacktriangleright a_s \vdash_f t_{C_i} \# \\ \Gamma, x \blacktriangleright <_1 \ldots x \blacktriangleright <_s \vdash_f t_{C_j} \#_1 \ldots \Gamma, x \blacktriangleright <_1 \ldots x \blacktriangleright <_s \vdash_f t_{C_j} \#_s\end{array}}{\Gamma \vdash_f \texttt{case}\ u\ \texttt{of}\ |\, C\,\overrightarrow{x} \Rightarrow t_1 \ldots |\, C\,\overrightarrow{x} \Rightarrow t_s\ \texttt{end}\ \#}$$

Fix'

$$\frac{\begin{array}{c}\Gamma \vdash_f v \blacktriangleright C\ \overrightarrow{s} \\ \Gamma \vdash_f u \blacktriangleright b_1 \ldots \Gamma \vdash_f u \blacktriangleright b_s \qquad \Gamma, \overrightarrow{d \blacktriangleright b}, x \blacktriangleright C\ \overrightarrow{s} \vdash_f t \#\end{array}}{\Gamma \vdash_f f\ \overrightarrow{u}\ v \#}\ \text{for}\ \texttt{fix}\ (\text{f} := \overrightarrow{d}\ \text{x})\text{Tt}$$

Conclusion

$$\text{Match}\ \frac{(\Gamma, \overrightarrow{x \blacktriangleright \|a\|},\ \sigma, \overrightarrow{x}) \vdash_f \mathbf{s} \mid b_i\ \#_1 \ldots (\Gamma, \overrightarrow{x \blacktriangleright \|a\|},\ \sigma, \overrightarrow{x}) \vdash_f \mathbf{s} \mid b_i\ \#_s \quad (\Gamma,\ \sigma) \vdash_f \emptyset \mid t \blacktriangleright a \quad (\Gamma, \overrightarrow{y \in \mathbf{p} \blacktriangleright \bot}, z \blacktriangleright \bot,\ \sigma, \overrightarrow{y \in \mathbf{p}}, z) \vdash_f \emptyset \mid T\ \#}{(\Gamma,\ \sigma) \vdash_f s \mid \begin{array}{l}\texttt{case}\ t\ \texttt{predicate}\ \mid \overrightarrow{\mathbf{p}}\ z\ \rightarrow\ T\ \texttt{of} \\ \mid C_i\ \overrightarrow{x}\ \Rightarrow\ b_{i1} \ldots \mid C_i\ \overrightarrow{x}\ \Rightarrow\ b_{is}\ \texttt{end}\end{array}\ \#}$$

- Le comportement de Coq formellement décrit
- sans ses désagrements
- sans preuve non plus

Merci de votre attention