Coq et les procédures de décision

Nicolas Ayache (CEA) et Jean-Christophe Filliâtre (CNRS)

ARC Quotient / 3 avril 2008

Plan

1. Quoi
2. Pourquoi
3. Comment

Quoi

objectif : interfacer Coq avec des **procédures de décision**

stage de M2 de Nicolas Ayache (2005), poursuivi depuis

ici procédure de décision = prouveur automatique du premier ordre avec quantificateurs ; exemples :

- **Simplify** (G. Nelson, projet ESC/Modula puis ESC/Java)
- **Yices** (L. de Moura & B. Dutertre, SRI ; successeur de ICS)
  → **Z3** (L. de Moura, Microsoft)
- **Alt-Ergo** (S. Conchon & E ; Contejean, ProVal)
- **CVC Lite** (C. Barrett & S. Berezin, Stanford)
  → **CVC3** (C. Barrett & C. Tinelli, NYU & U Iowa)
- **haRVey** (S. Ranise, D. Deharbe, P. Fontaine, LORIA)
- **Zenon** (D. Doligez, Focal)

Quoi

objectif : interfacer Coq avec des **procédures de décision**

stage de M2 de Nicolas Ayache (2005), poursuivi depuis

ici procédure de décision = prouveur automatique du premier ordre avec quantificateurs ; exemples :

- **Simplify** (G. Nelson, projet ESC/Modula puis ESC/Java)
- **Yices** (L. de Moura & B. Dutertre, SRI ; successeur de ICS)
  $\longrightarrow$ **Z3** (L. de Moura, Microsoft)
- **Alt-Ergo** (S. Conchon & E ; Contejean, ProVal)
- **CVC Lite** (C. Barrett & S. Berezin, Stanford)
  $\longrightarrow$ **CVC3** (C. Barrett & C. Tinelli, NYU & U Iowa)
- **haRVey** (S. Ranise, D. Deharbe, P. Fontaine, LORIA)
- **Zenon** (D. Doligez, Focal)

Pourquoi

le contexte est celui de la **preuve de programmes** (ProVal)

$\Rightarrow$ les buts sont **nombreux**, **énormes**, souvent **fastidieux** à prouver sans être de grande complexité logique

l'égalité et l'arithmétique (linéaire) sont les seules théories vraiment nécessaires ; le reste est axiomatisé

Pourquoi

le contexte est celui de la **preuve de programmes** (ProVal)

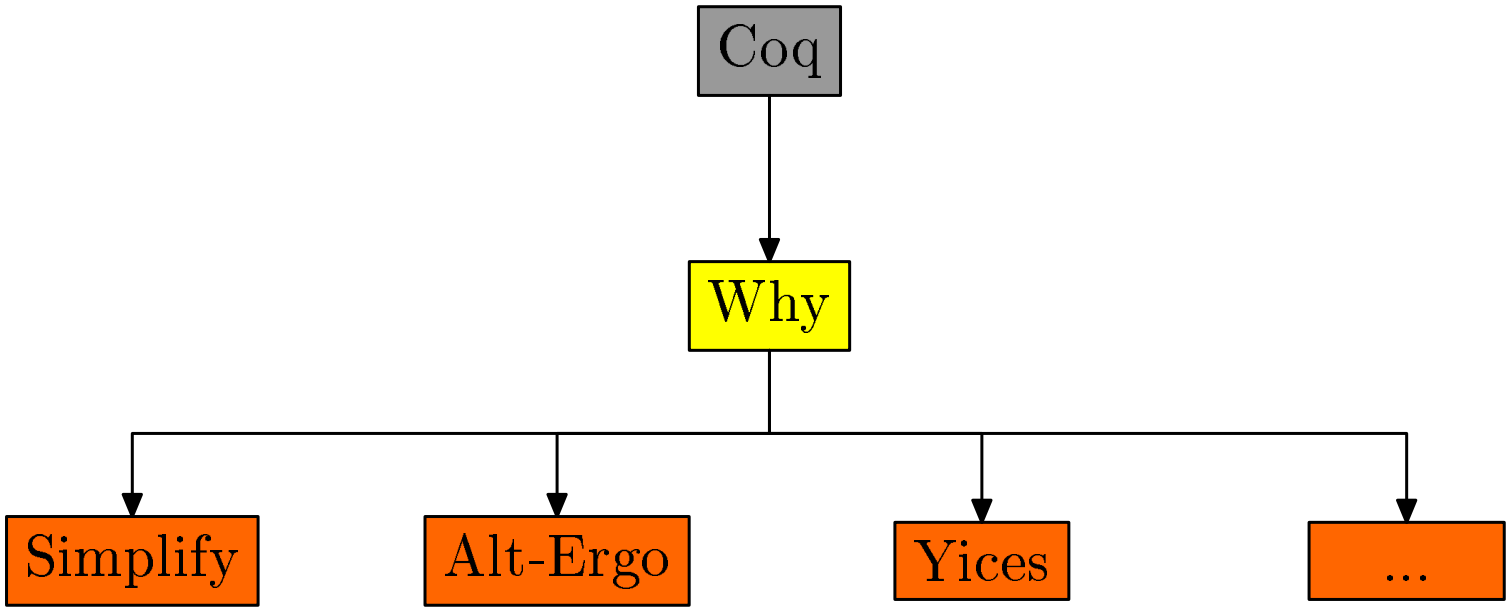
$\Rightarrow$ les buts sont **nombreux**, **énormes**, souvent **fastidieux** à prouver sans être de grande complexité logique

l'égalité et l'arithmétique (linéaire) sont les seules théories vraiment nécessaires ; le reste est axiomatisé

En pratique

- 6 tactiques :
  - `simplify`
  - `ergo`
  - `yices`
  - `cvcl`
  - `harvey`
  - `zenon`
- 3 commandes :
  - `Dp_hint` $x_1 \ldots x_n$
  - `Dp_timeout` n
  - `Dp_debug`
- un effet de bord :
  - la tactique `admit`

Comment



Comment

trois étapes

1. extraction du **fragment premier ordre** du but Coq,
   dans la logique de Why = logique du premier ordre polymorphe

2. production de la syntaxe d'entrée du prouveur,
   grâce à l'outil Why, et appel du prouveur

3. interprétation du résultat comme un axiome, sauf si un terme de preuve est produit (Zenon dès à présent, plus tard Alt-Ergo)

Comment

trois étapes

1. extraction du **fragment premier ordre** du but Coq,
   dans la logique de Why = logique du premier ordre polymorphe

2. production de la syntaxe d'entrée du prouveur,
   grâce à l'outil Why, et appel du prouveur

3. interprétation du résultat comme un axiome, sauf si un terme de preuve est produit (Zenon dès à présent, plus tard Alt-Ergo)

Comment

trois étapes

1. extraction du **fragment premier ordre** du but Coq,
   dans la logique de Why = logique du premier ordre polymorphe

2. production de la syntaxe d'entrée du prouveur,
   grâce à l'outil Why, et appel du prouveur

3. interprétation du résultat comme un axiome, sauf si un terme de preuve est produit (Zenon dès à présent, plus tard Alt-Ergo)

Extraction du fragment premier ordre du but Coq

extraction en partant du but

les arités, termes et formules du premier ordre sont traduites, le reste jeté

Définitions

une définition de la forme

$$f\ x_1 \ldots x_n = t$$

est traduite en un axiome si t = terme ou prédicat du premier ordre

une définition de la forme

$$\begin{aligned} f\ x_1 \ldots x_n = \ & \texttt{match}\ x_i\ \texttt{with} \\ & \mid C_1\ \vec{y_1} \Rightarrow t_1 \\ & \vdots \\ & \mid C_m\ \vec{y_m} \Rightarrow t_m \end{aligned}$$

est traduite en plusieurs axiomes

Définitions

une définition de la forme

$$f\ x_1 \ldots x_n = t$$

est traduite en un axiome si t = terme ou prédicat du premier ordre

une définition de la forme

$$\begin{aligned} f\ x_1 \ldots x_n = \ & \texttt{match}\ x_i\ \texttt{with} \\ & \mid C_1\ \vec{y_1} \Rightarrow t_1 \\ & \vdots \\ & \mid C_m\ \vec{y_m} \Rightarrow t_m \end{aligned}$$

est traduite en plusieurs axiomes

Définitions inductives

les constructeurs d'un prédicat inductif sont traduits en autant d'axiomes, lorsqu'ils correspondent à des formules du premier ordre

les constructeurs d'un type inductif dans `Set` deviennent des constantes / fonctions + axiomes exprimant le caractère d'algèbre libre du type inductif

Définitions inductives

les constructeurs d'un prédicat inductif sont traduits en autant d'axiomes, lorsqu'ils correspondent à des formules du premier ordre

les constructeurs d'un type inductif dans `Set` deviennent des constantes / fonctions + axiomes exprimant le caractère d'algèbre libre du type inductif

Définitions

la traduction des définitions Coq

- automatique
- paresseuse
- un mécanisme de **cache** conserve les traductions déjà effectuées

Traduction des lemmes

manuellement, par une commande `Dp_hint` semblable à `Hint`

Étape 2 : appel du prouveur

une fois le fichier `/tmp/f.why` produit, on appel ainsi Simplify

```
why --no-prelude --simplify /tmp/f.why

timeout 10 Simplify /tmp/f_why.sx > out 2>&1

grep -q -w Valid out
```

de même pour les autres prouveurs

il faut donc installer : `why` + `timeout` + le prouveur

Typage

les systèmes de types des prouveurs diffèrent :

- Simplify, Zenon, haRVey : **non typés**
- Yices, CVC Lite : **simplement typés**
- Alt-Ergo : **polymorphe** (même système de types que Why)

$\Rightarrow$ une **traduction** est nécessaire

$(\forall\ x : \mathtt{unit},\ x = \mathtt{tt}) \Rightarrow 1 = 2$

Encodage des types

le codage usuel des types par prédicats ne fonctionne pas
$\Rightarrow$ codage des types dans les termes

stage de M2 de Stéphane Lescuyer (2006) pour le cas non typé
adapté au cas simplement typé par J.-F. Couchot (postdoc) $\longrightarrow$ CADE 2007

exemple : avec $t : \texttt{int array}$, $i, j : \texttt{int}$ le terme

$$\texttt{upd}(t, i, \texttt{acc}(t, j))$$

devient

$$\begin{array}{l}\texttt{c\_sort}(\texttt{array}(\texttt{int}), \texttt{upd}(\ \texttt{c\_sort}(\texttt{array}(\texttt{int}), t),\\ \qquad\qquad\qquad\qquad\ \texttt{c\_sort}(\texttt{int}, i),\\ \qquad\qquad\qquad\qquad\ \texttt{c\_sort}(\texttt{int}, \texttt{acc}(\ \texttt{c\_sort}(\texttt{array}(\texttt{int}), t),\\ \qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{c\_sort}(\texttt{int}, j)))))\end{array}$$

Encodage des types

le codage usuel des types par prédicats ne fonctionne pas
$\Rightarrow$ codage des types dans les termes

stage de M2 de Stéphane Lescuyer (2006) pour le cas non typé
adapté au cas simplement typé par J.-F. Couchot (postdoc) $\longrightarrow$ CADE 2007

exemple : avec t : `int array`, i, j : `int` le terme

$$\mathtt{upd}(t, i, \mathtt{acc}(t, j))$$

devient

$$\begin{array}{l}\mathtt{c\_sort}(\mathtt{array}(\mathtt{int}), \mathtt{upd}(\ \mathtt{c\_sort}(\mathtt{array}(\mathtt{int}), t),\\ \qquad\qquad\qquad\qquad\quad \mathtt{c\_sort}(\mathtt{int}, i),\\ \qquad\qquad\qquad\qquad\quad \mathtt{c\_sort}(\mathtt{int}, \mathtt{acc}(\ \mathtt{c\_sort}(\mathtt{array}(\mathtt{int}), t),\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathtt{c\_sort}(\mathtt{int}, j))))))\end{array}$$

Terme de preuve

dans le cas de Zenon, un terme de preuve est construit

il est parsé et passé à `exact`

note : il peut y avoir des lemmes $\Rightarrow$ utilisation de `assert`

Travail en cours

tactique gappa pour appeler l'outil **Gappa** de Guillaume Melquiond

- réels et flottants, dans des intervalles constants

$$\left(\begin{array}{l} c \in [-0.3, -0.1] \wedge \\ (2a \in [3,4] \Rightarrow b + c \in [1,2]) \wedge \\ a - c \in [1.9, 2.05] \end{array} \right) \Rightarrow b + 1 \in [2, 3.5]$$

- produit une preuve Coq

Reste à faire

- documentation
- robustesse
- Zenon + encodage des types

démo