

Coq et les procédures de décision

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

Journée ADT Coq / 29 octobre 2008

- 1 Quoi
- 2 Pourquoi
- 3 Comment

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)

objectif : interfacier 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)

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

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

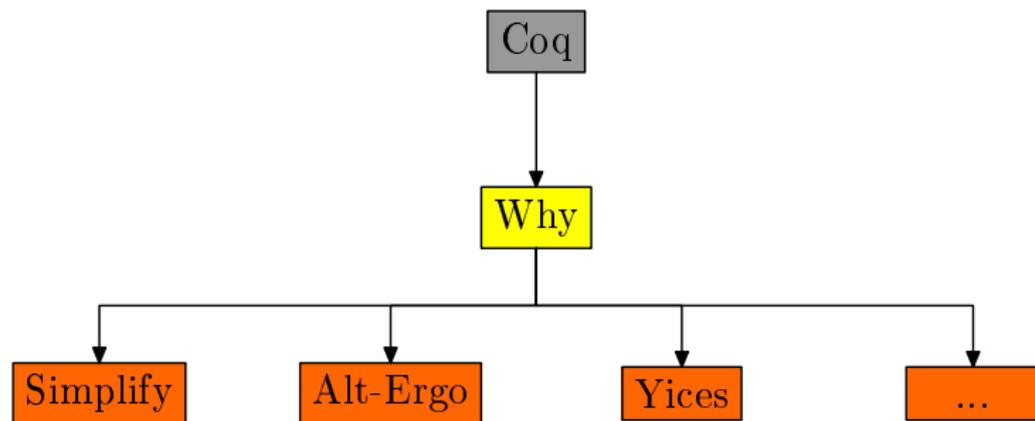
l'égalité et l'arithmétique (linéaire) sont les seules théories vraiment nécessaires ; le reste est axiomatisé

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

⇒ 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

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



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)

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)

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 en partant du but

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

une définition de la forme

$$f\ x_1 \dots x_n = t$$

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

une définition de la forme

$$f\ x_1 \dots x_n = \begin{array}{l} \text{match } x_i \text{ with} \\ | C_1\ \vec{y}_1 \Rightarrow t_1 \\ \vdots \\ | C_m\ \vec{y}_m \Rightarrow t_m \end{array}$$

est traduite en plusieurs axiomes

une définition de la forme

$$f\ x_1 \dots x_n = t$$

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

une définition de la forme

$$f\ x_1 \dots x_n = \begin{array}{l} \text{match } x_i \text{ with} \\ | C_1\ \vec{y}_1 \Rightarrow t_1 \\ \vdots \\ | C_m\ \vec{y}_m \Rightarrow t_m \end{array}$$

est traduite en plusieurs axiomes

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

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

la traduction des définitions Coq

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

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

Étape 2 : appel du prouveur

une fois le fichier `/tmp/f.why` produit, on appelle 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

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)

⇒ une **traduction** est nécessaire

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

Encodage des types

le codage usuel des types par prédicats ne fonctionne pas
⇒ 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) → CADE 2007

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

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

devient

```
c_sort(array(int), upd( c_sort(array(int), t),  
                        c_sort(int, i),  
                        c_sort(int, acc( c_sort(array(int), t),  
                                         c_sort(int, j))))))
```

Encodage des types

le codage usuel des types par prédicats ne fonctionne pas
⇒ 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) → CADE 2007

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

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

devient

$$\begin{aligned} \text{c_sort}(\text{array}(\text{int}), \text{upd}(\text{c_sort}(\text{array}(\text{int}), t), \\ \text{c_sort}(\text{int}, i), \\ \text{c_sort}(\text{int}, \text{acc}(\text{c_sort}(\text{array}(\text{int}), t), \\ \text{c_sort}(\text{int}, j)))))) \end{aligned}$$

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

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

- documentation
- robustesse
- Zenon + encodage des types

démo