Recherche de preuves en AF_2 par codage dans une logique d'ordre supérieur

Manuel Munier

Responsable : Didier Galmiche
Equipe Prograis

Introduction

- **Domaine d'étude :** la programmation par preuves dans une logique intuitionniste constructive.

- Construction de programmes en trois étapes :

  1. Donner une spécification formelle A du problème.
  2. Construire une preuve t de cette spécification.
  3. Extraire un algorithme de cette preuve (*formulae-as-types*).

$\Rightarrow$ construire un algorithme $\equiv$ rechercher une preuve en théorie des types

- Isomorphisme *formulae-as-types* de Curry-Howard :

t	:	A
terme		type
preuve		formule
programme		spécification

- Plusieurs théories ont été développées, dont :
  - la Théorie des Types de Martin-Löf ($MLTT$)
  - le Calcul des Constructions (CC)
  - l'Arithmétique Fonctionnelle du second ordre (AF_2) (Parigot, Krivine 88)

- Deux domaines d'étude interdépendents pour la recherche de preuves :
  - étude des liens entre preuves et programmes (efficacité des programmes obtenus)

    $\Rightarrow$ choix d'induction sur les types de données
  - automatisation :
    - tactiques et plans de preuves
    - preuves canoniques (ex: $\mathcal{R}$-preuves)
    - codage dans une autre logique

Une théorie des types du second ordre : AF_2

Termes

- Si t et u sont des termes alors $(t\ u)$ est un terme.
- Si x est une variable et t un terme alors $\lambda x.t$ est un terme.

Types

- Si f est une fonction d'individu d'arité n $(n \geq 0)$ et si $i_1, \cdots, i_n$ sont des individus, alors $f i_1 \cdots i_n$ est un individu.
- Si X est un prédicat d'arité n $(n \geq 0)$ et si $i_1, \cdots, i_n$ sont des individus, alors $X i_1 \cdots i_n$ est un type.
- Si A et B sont des types, alors $A \supset B$ est un type.
- Si A est un type et si x est une variable d'individu, alors $\forall x.A$ est un type.
- Si A est un type et si X est une variable de prédicat, alors $\forall X.A$ est un type.

Jugements

$$\begin{array}{lcl} t : A & \equiv & t \textit{ est un élément du type } A \\ & \equiv & t \textit{ est une preuve de la formule } A \end{array}$$

Règle de l'axiome ($axiom$)

$$\Gamma, t : A \vdash t : A$$

Règle d'abstraction (abs_i)

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}$$

Règle d'application (app_i)

$$\frac{\Gamma \vdash t : A \supset B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}$$

Règle de généralisation (gen_1)
avec x non libre dans Γ

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x.A}$$

Règle de spécialisation (spe_1)
où b est un individu

$$\frac{\Gamma \vdash t : \forall x.A}{\Gamma \vdash t : [b/x]A}$$

Règle de généralisation (gen_2)
avec X non libre dans Γ

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X.A}$$

Règle de spécialisation (spe_2)
où B est une formule

$$\frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t : [B/X]A}$$

Règle structurelle
où Δ est obtenu à partir de Γ par permutation, par affaiblissement ou par contraction

$$\frac{\Gamma \vdash t : A}{\Delta \vdash t : A}$$

Règles d'inférence en déduction naturelle pour AF_2

Programmer avec des preuves

1. Nous représentons les types de données par des formules.

   Ex.: $Nx \equiv \forall X.((\forall y.(Xy \supset X(\underline{s}\ y))) \supset (X\underline{0} \supset Xx))$

   Remarque : Le choix de cette représentation est essentiel et a une influence sur l'algorithme obtenu.

2. Nous exprimons les spécifications du programme par un ensemble d'équations définissant sémantiquement la fonction que nous voulons calculer.

   Ex.: $$\begin{aligned} plus(\underline{0}, x) &= x \\ plus(x, \underline{s}(y)) &= \underline{s}(plus(x, y)) \end{aligned}$$

3. Le programme est obtenu en dérivant la proposition indiquant que la fonction a le type désiré.

   Ex.: $\forall x.(Nx \supset \forall y.(Ny \supset Nplus(x, y)))$

Recherche de preuves

- *Tactiques et plans de preuves*

  1. représentation des règles d'inférence du système par des *tactiques*

  2. définition d'une *méta-tactique* à l'aide d'heuristiques

  $\Rightarrow$ classes de preuves considérées ?

- *Preuves canoniques*

  1. définition d'une classe de preuves

     Ex.: les preuves uniformes, les $\mathcal{R}$-preuves

  2. développement d'une procédure de recherche adaptée

  $\Rightarrow$ approche automatique

- *Une troisième approche*

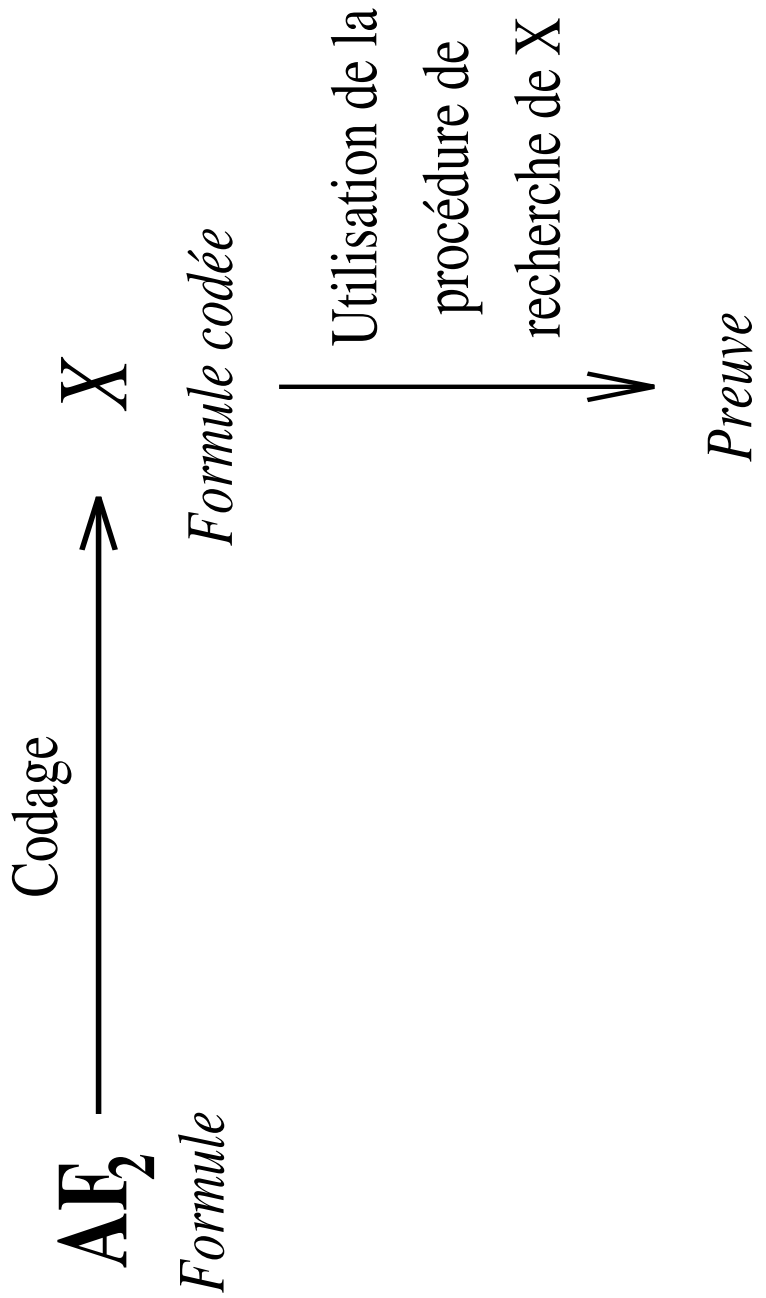

FIG. 1 -

La logique intuitionniste d'ordre supérieur $\mathcal{I}$

Termes = λ-termes simplement typés

Types

- Si $t_1, \cdots, t_n$ sont des termes et si p est un prédicat d'arité n ($n \geq 0$), alors $pt_1, \cdots, t_n$ est un type.

- Si A et B sont des types, alors $A \wedge B$ et $A \supset B$ sont des types.

- Si A est un type et si t est une variable de terme, alors $\forall t.A$ est un type.

Les formules de Harrop héréditaires d'ordre supérieur : hh

- Dans hh, la tête d'une formule atomique (ie p dans $pt_1, \cdots, t_n$) doit être une **constante** alors que dans $\mathcal{I}$ on peut avoir une **variable**.

- Syntaxe logique de hh :

$$\mathcal{D} ::= \mathcal{A} \mid \mathcal{A} \subset \mathcal{G} \mid \forall x.\mathcal{D} \mid \mathcal{D} \wedge \mathcal{D}$$

$$\mathcal{G} ::= \mathcal{A} \mid \mathcal{G} \wedge \mathcal{G} \mid \mathcal{G} \vee \mathcal{G} \mid \mathcal{D} \supset \mathcal{G} \mid \forall x.\mathcal{G} \mid \exists x.\mathcal{G}$$

$$\mathcal{A} ::= formules\ atomiques$$

$$\frac{B, C, \mathcal{P} \to A}{B \wedge C, \mathcal{P} \to A}\ (\wedge L) \qquad \frac{\mathcal{P} \to B \quad \mathcal{P} \to C}{\mathcal{P} \to B \wedge C}\ (\wedge R)$$

$$\frac{\mathcal{P} \to B \quad C, \mathcal{P} \to A}{B \supset C, \mathcal{P} \to A}\ (\supset L) \qquad \frac{B, \mathcal{P} \to C}{\mathcal{P} \to B \supset C}\ (\supset R)$$

$$\frac{[t/x]B, \mathcal{P} \to A}{\forall_\tau x.B, \mathcal{P} \to A}\ (\forall L)\star \qquad \frac{\mathcal{P} \to [c/x]B}{\mathcal{P} \to \forall_\tau x.B}\ (\forall R)\star\star$$

$\star$ t est un terme de type τ

$\star\star$ c est une constante de type τ qui n'est pas libre dans le séquent du bas

Règles d'inférence pour $\mathcal{I}$

Preuve uniforme

Soit $\mathcal{P}$ un ensemble fini de formules et soit B une formule. Le séquent $\mathcal{P} \to B$ a une preuve ssi il a une preuve dans laquelle chaque séquent contenant une formule non-atomique comme succédent est la conclusion d'une règle d'introduction à droite.

Une procédure de recherche

- **AND :** Si G est $G_1 \wedge G_2$ alors essayer de démontrer que G_1 et G_2 se déduisent tous deux de $\mathcal{P}$.

- **GENERIC :** Si G est $\forall_\tau x.G'$ alors choisir une nouvelle constante c de type τ et essayer de démontrer que $[c/x]G'$ se déduit de $\mathcal{P}$.

- **AUGMENT :** Si G est $D \supset G'$ alors essayer de démontrer que G' se déduit de $\mathcal{P} \cup \{D\}$.

- **BACKCHAIN :** Si G est atomique et s'il existe une paire $\langle \mathcal{F}, G \rangle \in |\mathcal{P}|$ alors essayer de démontrer que chacune des formules de $\mathcal{F}$ se déduit de $\mathcal{P}$. La preuve se termine si $\mathcal{F}$ est vide.

Remarques :

- AUGMENT ajoute une clause au programme
- GENERIC ajoute un symbole à la signature

Codage d'AF_2 dans $\mathcal{I}$

Pour les objets

$$\begin{array}{rl} \langle\!\langle t \rangle\!\rangle & = \rho(t) = t \\ \langle\!\langle tu \rangle\!\rangle & = (app_o\ \langle\!\langle t \rangle\!\rangle\ \langle\!\langle u \rangle\!\rangle) \\ \langle\!\langle \lambda x.t \rangle\!\rangle & = (abs_o\ \lambda x.\langle\!\langle t \rangle\!\rangle) \end{array}$$

Pour les individus

$$\langle\!\langle i \rangle\!\rangle = \rho(i) = i$$

Pour les types

$$\begin{array}{rl} \langle\!\langle A \rangle\!\rangle & = \rho(A) = A \\ \langle\!\langle P(i_1, \cdots, i_n) \rangle\!\rangle & = (app_t\ \langle\!\langle P(i_1, \cdots, i_{n-1}) \rangle\!\rangle\ \langle\!\langle i_n \rangle\!\rangle) \\ \langle\!\langle A \to B \rangle\!\rangle & = (\lambda f.(\forall x.([\![x : A]\!] \supset [\![fx : B]\!]))) \\ \langle\!\langle \forall x.A \rangle\!\rangle & = (\lambda f.(\forall x.[\![f : A]\!])) \\ \langle\!\langle \forall X.A \rangle\!\rangle & = (\lambda f.(\forall X.[\![f : A]\!])) \end{array}$$

Pour les jugements

$$[\![t : A]\!] = (\langle\!\langle A \rangle\!\rangle\ \langle\!\langle t \rangle\!\rangle)$$

Codage des termes et des jugements d'AF_2

Remarque : On retrouve la sémantique de Heyting.

Exemple de codage

La formule en AF_2 suivante :

$$\forall X.((\forall y.(Xy \supset X(\underline{s}\ y))) \supset (X\underline{0} \supset Xx))$$

sera codée par la formule de $\mathcal{I}$ suivante :

$$\lambda f(\forall X.(\lambda g(\forall u.($$

$$(\lambda h(\forall y.(\lambda i(\forall v.((app_t\ X\ y)\ v$$
$$\supset (app_t\ X\ (app_i\ succ\ y))\ (app_o\ i\ v)))))\ h)))\ u)$$

$$\supset$$

$$(\lambda j(\forall w.((app_t\ X\ zero)\ w \supset (app_t\ X\ x)\ (app_o\ j\ w))))$$

$$(app_o\ g\ u)))))\ f))))$$

$conv_o\ (app_o\ (abs_o\ B)\ A)\ (B\ A)$
$conv_o\ A\ C \wedge conv_o\ B\ D \supset conv_o\ (app_o\ A\ B)\ (app_o\ C\ D)$
$conv_o\ A\ A$
$conv_o\ B\ A \supset conv_o\ A\ B$
$conv_o\ A\ C \wedge conv_o\ C\ B \supset conv_o\ A\ B$

Relations de convertibilité pour les objets d'AF_2

– Pour prendre en compte la convertibilité dans les buts et dans les hypothèses nous introduisons les deux formules suivantes :

$conv_t\ A\ B \wedge conv_o\ M\ N \wedge B\ N \supset A\ M$
$conv_t\ A\ B \wedge conv_o\ M\ N \wedge B\ N \wedge (A\ M \supset G) \supset G$

⇒ **Correction**

Pour toute dérivation d'AF_2 il existe une preuve dans $\mathcal{I}$.

La correction de ce codage a été démontrée.

⇒ **Complétude**

Toute preuve de $\mathcal{I}$ correspond à une dérivation d'AF_2.

Pour obtenir la complétude, il est nécessaire de modifier le codage.

La procédure de recherche pour $\mathcal{I}$

L'opération BACKCHAIN peut être décrite comme suit :

1. Choix d'une paire $\langle \mathcal{F}, A \rangle$ telle que A soit atomique.
2. Remplacer les variables libres de A par des variables logiques.
3. Unifier A avec le but courant G.
4. Si cela réussit, appliquer la substitution résultante aux formules de $\mathcal{F}$ et essayer de prouver chacune d'entre elles.

Exemple :

Nous avons le programme logique suivant :

$$\begin{aligned} &et\ U\ V\ :-\ U\ ,\ V\ . \\ &ou\ U\ V\ :-\ U\ ;\ V\ . \end{aligned}$$

Le but courant est : $et\ vrai\ Z$

1. On choisit la paire $\langle \{U, V\}, et\ U\ V \rangle$
2. On remplace les variables libres par des variables logiques $\Rightarrow et\ X\ Y$
3. On unifie avec le but courant $\Rightarrow \begin{cases} X \rightarrow vrai \\ Y \rightarrow Z \end{cases}$
4. On continue avec les deux sous-buts $vrai$ et Z.

Problèmes

1. Puisque la formule A de la paire choisie est atomique, elle est de la forme $p\ t_1 \cdots t_n$.

   - Dans hh, p est une **constante** $\Rightarrow$ le résultat de la substitution est une formule atomique.
   - Dans $\mathcal{I}$, p peut être une **variable** $\Rightarrow$ si l'unification échoue, nous devons quand même tenir compte des substitutions qui transforment A en une formule non-atomique.

   Exemple :

   Considérons une substitution qui transforme A en $B \supset A'$ où A' est atomique. Si A' s'unifie avec G, alors il faut démontrer chacune des formules de $\mathcal{F} \cup \{B\}$, sinon il faut continuer le processus.

   $\Rightarrow$ Dans $\mathcal{I}$ nous avons un indéterminisme supplémentaire dû au fait que nous devons *deviner* la substitution pour A avant même d'utiliser l'unification.

2. Dans hh il y a une restriction sur les connecteurs $\supset$

   $\Rightarrow$ Le codage de Nx est correct pour $\mathcal{I}$ mais est **incorrect** pour hh.

Exemple de pb

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash Xx \supset X(\underline{s}\ x) & \Gamma \vdash Xx\end{array}}{Nx\ ,\ (\forall y.(Xy \supset X(\underline{s}\ y)))\ ,\ X\underline{0} \vdash X(\underline{s}\ x)}}{Nx\ ,\ (\forall y.(Xy \supset X(\underline{s}\ y))) \vdash X\underline{0} \supset X(\underline{s}\ x)}}{Nx \vdash (\forall y.(Xy \supset X(\underline{s}\ y))) \supset (X\underline{0} \supset X(\underline{s}\ x))}}{Nx \vdash \forall X.((\forall y.(Xy \supset X(\underline{s}\ y))) \supset (X\underline{0} \supset X(\underline{s}\ x)))}$$

Une solution

- Modifier l'interpréteur pour obtenir une procédure de recherche *transitivement complète*, i.e.,

  lorsque nous essayons de prouver $\Gamma \vdash P : Q$, il est possible de démontrer une série de *lemmes* conduisant éventuellement au résultat désiré

  Ex.: $\Gamma \vdash P_1 : Q_1$
  $\Gamma, P_1 : Q_1 \vdash P_2 : Q_2$
  $\vdots$
  $\Gamma, P_1 : Q_1, \cdots, P_n : Q_n \vdash P : Q$ où $n \geq 0$

- Application :

  $\Gamma \vdash u : A$
  $\Gamma, u : A \vdash t : A \supset B$
  $\Gamma, u : A, t : A \supset B \vdash (tu) : B$

  $\Rightarrow$ C'est exactement le cas de la règle (app_i)

- D'où la modification proposée :

  1. Ne pas restreindre l'application de l'opération BACKCHAIN aux formules atomiques

  2. Toujours tenter d'appliquer BACKCHAIN avant toute autre opération de recherche

  3. Toujours utiliser l'unification pout tenter d'unifier le but courant avec la sous-formule de tête d'une hypothèse

- Conclusion :

  Codage d'AF_2 dans $\mathcal{I}$

  $\Rightarrow$ Mise en œuvre en λProlog

  $\Rightarrow$ La procédure de recherche utilisée est celle de hh

- Solutions proposées :

  – λProlog basé sur $\mathcal{I}$ et non plus sur hh

  – preuves canoniques en AF_2 + procédure de recherche tenant compte des problèmes soulevés

Tactiques en λProlog pour AF_2

- *tactiques* = fonction réduisant un but en une liste de sous-buts

```
app_i_tac  (Gamma --> (app_i T U) :: B)
           (andgoal (Gamma --> U :: A)
                    (Gamma --> T :: (imp A B)))

           :- format "Terme sur lequel on
                 applique la regle : " [ ],

              input A, nl.
```

- *méta-tactique* = procédure déterminant l'ordre dans lequel seront appliquées les tactiques

  $\Rightarrow$ On peut définir une procédure de recherche en utilisant la notion de méta-tactique.

- Une procédure de recherche interactive a été implantée, i.e., elle demande à chaque étape quelle est la tactique à appliquer au but courant