Pieuvre: le mini assitant de preuve

Marwan et Augustin

1 Présentation

Nous avons programmé **Pieuvre**, un assistant de preuve pour la logique intuitionniste en **OCaml**. Il permet, à l'aide d'une boucle interactive, de prouver des prédicats simples.

Nous indiquons dans la partie 2 comment compiler, lancer et utiliser notre programme. Nous exposons ensuite à la partie 3 comment notre programme est structuré. Nous présentons enfin dans 4 notre implémentation du λ-calcul simplement typé puis 5 notre implémentation des preuves.

2 Compiler et exécuter

Pour compiler : `make pieuvre`

Pour exécuter **Pieuvre** et entrer dans la boucle interactive de l'assistant de preuve, il suffit d'entrer la commande `./pieuvre`. Sinon, on peut spécifier un des modes suivants :

- `-alpha`
  qui vérifie l'α-équivalence de deux λ-termes séparés par `&`.
- `-reduce`
  qui affiche les β-réduction successives du λ-terme.

3 Organisation du code

Le code est structuré de la manière suivante :

- `parser.mly` et `lexel.mll` qui définissent l'analyse syntaxique du λ-calcul simplément typé ainsi que des tactiques.
- `lam.ml` qui contient notre implémentation du λ-calcul simplement typé.
- `types.ml` qui contient la définition des `types` et `typing.ml` qui implémente l'algorithme de typage bidirectionnel défini dans la partie 4.
- `proof.ml` qui définit la structure d'une preuve et implémente les tactiques.
- `main.ml` le point d'entrée de notre programme qui fait l'interface avec l'utilisateur.

4 λ-calcul simplement typé

Avant de pouvoir réaliser des preuves, il faut pouvoir les encoder. On utilise le λ-calcul simplement typé. Dans notre programme, on utilise la syntaxe suivante pour les types et pour les termes :

$\langle A, B \rangle$ ::= `X`
| `A -> B`
| `A /\ B`
| `A \/ B`
| `false`

$\langle M, N \rangle$ ::= `x`
| `fun (x : A) => M`
| `M N`
| `exf (M : A)`

Les termes ainsi définis se réduisent à l'aide des règles de β-réduction rappelées dans les notes de Selinger [1]. Avant d'être β-réduit par `betastep`, les termes sont d'abord α-convertis avec `alpha_convert` afin d'éviter les problèmes de substitution.

On remarque que l'on impose d'annoter le type de l'argument d'une λ-abstraction. Tous les termes sont alors typables avec les règles bidirectionnelles suivantes :

règles de typage

$$\frac{\texttt{x : A} \in \Gamma}{\Gamma \vdash \texttt{x : A}}$$

$$\frac{\Gamma \vdash \texttt{M : B}}{\Gamma \vdash \texttt{fun (x : A)=> M : A -> B}}$$

$$\frac{\Gamma \vdash M : \texttt{A -> B} \quad \Gamma \vdash N : \texttt{A}}{\Gamma \vdash \texttt{M N} : \texttt{B}}$$

$$\frac{\Gamma \vdash \texttt{M : false}}{\Gamma \vdash \texttt{exf (M : A)}: \texttt{A}}$$

5 Preuves

Augustin

6 Vérification des preuves

parler de la vérification de preuve, quelles difficultés ? quelles certitudes ?

Références

[1] Peter Selinger. Lecture notes on the lambda calculus. *CoRR*, 2008.