TP de λ-calcul

Pierre Courtieu

Résumé

Ce projet a pour but de vous faire manipuler les λ-termes et implanter la β-réduction et les stratégies de réduction. L'implantation utilise la représentation des termes avec des *indices de De Bruijn*.

1 λ-calcul, Rappels

Ci-dessous des rappels de définition concernant le $\lambda-$*calcul avec noms de variables explicites*. Vous constaterez que la présence de noms de variables rend complexes les définitions, notamment celle de la *substitutions*.

Le but de ce TP est d'implanter un $\lambda-$*calcul* sans nom de variable, afin d'éviter les difficultés liées aux noms de variables.

1.1 Grammaire

$$t ::= x \mid \lambda x.t \mid t\ t$$

Où x désigne un nom de variable quelconque. Un abus de notation permet d'écrire $t\ u\ v$ à la place de $(t\ u)\ v$ mais nous essaierons de ne pas l'utiliser.

1.2 α-équivalence

Le nom de variables n'importe pas (sauf en cas de capture), il existe donc une relation d'équivalence entre termes : l'α-équivalence :

$$\lambda x.t \equiv_\alpha \lambda y.(t[x \leftarrow y])$$

à condition que y ne soit pas déjà libre dans t et que la substitution ne provoque pas de capture. Cette condition supplémentaire rend difficile une implantation correcte. Ci-dessous deux exemples illustrant le problème.

Exemple 1 : $\lambda x.(x\ z) \equiv_\alpha \lambda y.(y\ z)$ mais $\lambda x.(x\ z) \not\equiv_\alpha \lambda z.(z\ z)$ car z est déjà libre dans $(x\ z)$.

Exemple 2 : $\lambda y.(\lambda x.(x\ y)) \not\equiv_\alpha \lambda x.(\lambda x.(x\ x))$ car x est capturé par le λ.

1.3 Substitution

Plus précisément, en présence de noms de variables, on peut définir la substitution de manière récursive comme suit :

$$
\begin{array}{rll}
x[x \leftarrow u] & \equiv u & \\
x[y \leftarrow u] & \equiv x & \text{si } x \neq y \\
(t_1\ t_2)[x \leftarrow u] & \equiv (t_1[x \leftarrow u])\ (t_2[x \leftarrow u]) & \\
(\lambda x.t)[x \leftarrow u] & \equiv \lambda x.t & \\
(\lambda x.t)[y \leftarrow u] & \equiv \lambda x.(t[y \leftarrow u]) & \text{si } x \neq y \text{ et si x n'est pas libre dans } u.
\end{array}
$$

Lorsqu'aucune règle ne s'applique la substitution n'est pas possible. Une solution consiste alors à renommer les variables (liées et/ou libres) qui posent problème avec des noms « frais » avant de faire la substitution.

1.4 Réduction

$$(\lambda x.t)\ u \rightarrow_\beta t[x \leftarrow u]$$

2 Les indices de De Bruijn

La gestion des noms de variables libre/non libres dans un terme rend pénibles les définitions et les implantations. Les indices de De Bruijn permettent de représenter les termes du λ-calcul sans nommer les variables, ce qui évite tous ces problèmes. De plus ils permettent de n'avoir qu'une seule façon de représenter un terme donné, ce qui évite de considérer l'α-équivalence.

Le principe est simple : une variable liée est représentée par une référence vers le λ qui la lie. Cette référence sera un entier désignant le nombre de λ qu'il faut « traverser » pour atteindre le λ voulu. Une variable est libre si son indice est plus grand que le nombre de λ qui la sépare de la racine du term. Le terme $\lambda x.\lambda y.\lambda z.(y\ (\lambda v.y\ v)\ u)$ est représenté par le terme suivant :

$$
\begin{array}{rl}
\text{Avec noms de variables :} & \lambda x.\lambda y.\lambda z.(y\ (\lambda v.y\ v)\ u) \\
\text{Avec indices de De Bruijn :} & \lambda.\ \lambda.\ \lambda.\ (1\ (\lambda.\ 2\ 0)\ \underset{\text{libre}}{3}) \\
\\
\text{Re-numérotation de la variable libre :} & \lambda.\ \lambda.\ \lambda.\ (1\ (\lambda.\ 2\ 0)\ 7)
\end{array}
$$

Notez en particulier que la variable y est désignée par 1 dans sa première occurrence et par 2 dans sa deuxième occurrence, car dans cette dernière il y a un λ de plus à « remonter ». La principale difficulté de ce TP est là : **LA MÊME VARIABLE EST DÉSIGNÉE PAR DES INDICES DIFFÉRENTS** dans le même terme. Cela complique l'implantation de la substitution et la β-réduction.

2.1 L'opération difficile : la substitution

En présence d'indices de De Bruijn la substitution est difficile à écrire car la même variable n'a pas le même indice en fonction de sa place dans le terme.

Cependant contrairement à la version avec noms de variable *la substitution est toujours possible*. la substitution est une opération qui prend trois arguments : le terme dans lequel on substitue

(le substitué), le numéro de la variable (libre) à substituer et le terme pas lequel substituer (le substitut). Attention : les numéros de variables libres changent quand on traverse un λ, il faut donc adapter 1) le numéro de la variable recherchée, et 2) le substitut.

Exemples :

$$\begin{aligned}
(2\ 1)[1 \leftarrow 2] &= (2\ 2) \\
(2\ 1)[1 \leftarrow 4] &= (2\ 4) \\
(\lambda.(1\ 0))[0 \leftarrow 3] &= \lambda.(4\ 0)(\leftarrow \text{ il faut comprendre ça pour faire le TP, prenez 5 minutes ici}) \\
(0\ \lambda.(1\ 0))[0 \leftarrow 3] &= 3\ \lambda.(4\ 0)(\leftarrow \text{ idem}) \\
(\lambda.(1\ (\lambda.(2\ 0))))[0 \leftarrow 3] &= (\lambda.(4\ (\lambda.(5\ 0))))(\leftarrow \text{ idem})
\end{aligned}$$

La première question de ce TP (section suivante), concerne la définition (sur papier) de la substitution avec indice de De Bruijn.

2.2 β-réduction

Une fois la substitution implantée, la β-réduction est relativement simple à implanter. Attention toutefois que cette opération enlève un λ et qu'il faut donc mettre à jour les indice de variables libres.

2.3 TP

2.4 Question préliminaire

1. Écrivez la définition de la substitution avec indice de De Bruijn en vous inspirant de la définition de la section 1.3.

$$\begin{aligned}
i[i \leftarrow u] &\equiv u \\
j[i \leftarrow u] &\equiv j \qquad \text{si } i \neq j \\
(t_1\ t_2)[i \leftarrow u] &\equiv \dots \\
\dots &\equiv
\end{aligned}$$

   où i, j désignent des variables (des indices donc), et t, u, v désignent des termes.
2. Écrivez la définition de la β-réduction en présence d'indice de De bruijn :

$$(\lambda.t)u \rightarrow_\beta ?$$

2.5 Implantation

On propose le type caml suivant pour représenter les λ-termes :

```
type term =
    Var of int (** Indice de De Bruijn *)
  | Lam of term (** λ-abstraction, pas de nom de variable *)
  | App of term * term (** (f t1), composer les App pour
                           représenter (f t1 t2 ...) *)
```

Le terme $\lambda.\lambda.\lambda.((1\ (\lambda.2\ 0))\ 3)$ est défini par :

```
le terme_ci_dessus =
 Lam(Lam(Lam(App(
              App(Var 1,Lam(App(Var 2, Var 0))),
              Var 3))))
```

2.6 Questions

3. Programmer la fonction `lift_level: int -> int -> term -> term` telle que `lift_level lvl n t` retourne le terme `t` dans lequel les variables libres d'indices supérieurs à `lvl` ont été augmentés de `n`. Attention aux indice « glissants ».

   En déduire une fonction `lift:int -> term -> term` telle que `lift n t` retourne le terme `t` dans lequel les indices des variables libres ont été augmentés de `n`.

4. Programmer la fonction `subst:int -> term -> term -> term` telle que `subst lvl t u` retourne le terme `t` dans lequel les indices correspondant à `lvl` sont remplacés par `u`. Attention aux indices « glissants ».

   Exemple : Supposons le term suivant `t` = `Lam(App(Var 1,Var 8))`, alors `subst 0 t (Var 2)` retourne `Lam(App(Var 3,Var 8))`

5. Programmez la fonction `beta` qui effectue un pas de β-réduction dans un terme (recherche du redex + calcul du terme obtenu).

6. Programmez la fonction `normale` qui applique la stratégie de réduction normale (à gauche d'abord, au sommet d'abord) tant que cela est possible. Tester que cela boucle sur $(\lambda x.(x\ x))(\lambda x.(x\ x))$ mais pas sur factorielle 3 :

   Rappel :
   — `ISZERO := λn.n (λx.FALSE) TRUE`
   — `IFTHENELSE := λp.λa.λb.p a b`
   — `G := IFTHENELSE (ISZERO n 0)1 (n × F F (n-1))`
   — `FACT := (λF.λn.G) (λF.λn.G)`

7. Programmez la fonction `byvalue` qui applique la stratégie de réduction par valeur tant que cela est possible. Tester que cela boucle sur $(\lambda x.(x\ x))(\lambda x.(x\ x))$ **et** sur factorielle 3 :

   La stratégie d'appel par valeur n'applique la β-réduction que si l'argument est une variable ou un λ.

8. programmez l'affichage d'un λ-terme simulant des noms de variables. Par exemple :

```
print (Lam(Lam(Lam(App(Var 1, Lam(App(Var 2,Lam(Var 1)))))))
---> (\x4.(\x3.(\x2.(x3 (\x1.(x3 (\x0. x1)))))))
```

   On ne s'intéresse pas à minimiser le nombre de parenthèses, donc on en mettra partout. On pourra utiliser le module `mapName.ml` fourni.