TP n° 11 – Lambda-calcul

etienne.jl.simon@gmail.com thomas.h@ens.fr
thomash.fr/teaching/

1 Introduction

Le lambda-calcul est un langage de programmation réduit au minimum, il ne comporte que trois constructions syntaxiques. Un terme de lambda-calcul est soit une variable (notée simplement x), soit l'application d'un terme u à un terme v (notée uv), soit un lambda (c'est-à-dire une fonction anonyme notée $\lambda x.u$ et qui est l'équivalent de `fun x -> u` en caml). Ainsi le lambda-terme correspondant à la fonction identité est $\lambda x.x$ et $\lambda f.\lambda x.fx$ applique une fonction à un argument.

Question 1. Écrire les lambda-termes correspondants à la fonction qui compose deux fonctions et à la fonction qui prend deux arguments et renvoie le premier.

2 Évaluation

"L'évaluation" d'un terme est appelée béta-réduction, c'est l'opération $(\lambda x.u)v \mapsto u[v/x]$ où $u[v/x]$ est le terme u dans lequel les occurrences de x sont remplacées par v. Attention, il peut être nécessaire de renommer les variables lors de cette opération. Par exemple le terme $(\lambda x.\lambda y.xy)y$ se réduit en $\lambda z.yz$ et non en $\lambda y.yy$. La règle est que $(\lambda x.u)[v/y] = \lambda x.u[v/y]$ uniquement si x n'est pas une variable libre de v, sinon il faut renommer x ($(\lambda x.u)[v/y] = \lambda z.u[z/x][v/y]$ où z n'est libre ni dans u ni dans v).

On représente les lambda-termes par le type suivant.

```
type term =
| Var of string
| App of term * term
| Lambda of string * term
```

Question 2. Écrire la fonction `fv : term -> string list` qui calcule les variables libres d'un lambda-terme.

Question 3. Écrire la fonction `substitute : string -> term -> term -> term` qui substitue un terme à une variable.

Question 4. L'ordre dans lequel les redex (les termes de la forme $(\lambda x.u)v$) sont réduits est non spécifié, il existe donc plusieurs stratégies pour choisir l'ordre de réduction. Donner un exemple de terme pour lequel différentes stratégies de réduction donnent des résultats différents.

Question 5. Écrire la fonction `call_by_value : term -> term` qui réduit un terme autant que possible en évaluant toujours l'argument d'une fonction avant l'appel à la fonction.

Question 6. Écrire la fonction `call_by_name : term -> term` qui réduit un terme autant que possible en évaluant jamais l'argument d'une fonction avant l'appel à la fonction.

3 Oups

Notons $\omega = \lambda x.xx$ et $\Omega = \omega\omega$. Yorel Reivax veut réduire le terme Ω, va-t-il y arriver ?

4 Types

Afin d'interdire les termes mauvais comme Ω, nous allons introduire un système de types. On considère le système de type composé de variables de type (comme les `'a` en caml) et du type fonction. On représente les types par le type

```
type t =
| A of string
| Fun of t * t
```

et les termes typés par

```
type typed_term =
| TVar of string * t
| TApp of typed_term * typed_term * t
| TLambda of string * typed_term * t
```

Question 7. Écrire une fonction `check_type : typed_term -> bool` qui vérifie qu'un terme est correctement typé.

Question 8. * Écrire une fonction `infer_type : term -> typed_term` qui type un terme non typé (en levant une exception si le terme est incorrect).

5 Universalité du lambda-calcul

Le lambda-calcul peut sembler faible mais n'importe quel autre langage peut être encodé en lambda-calcul ce qui en fait un outil de choix pour les preuves théoriques.

Question 9. Proposer un encodage pour les paires (écrire des termes pour construire des paires et en extraire le premier ou le second élément).

Question 10. Proposer un encodage pour les booléens (écrire les termes "vrai", "faux", "et", "ou", "non" et "if then else").

Question 11. On peut représenter les entiers naturels par l'encodage de Church qui à l'entier n associe la fonction $f \mapsto f^n$. Écrire l'encodage des nombres 0, 1 et 2.

Question 12. Écrire un terme qui additionne deux entiers de Church.

Question 13. Écrire un terme qui multiplie deux entiers de Church.

Question 14. Écrire un terme qui calcule la fonction puissance pour les entiers de Church.