

Modèles de Calcul [Lambda-Calcul]

Initiation, syntaxe et portées



Martin Bodin, Pascal Fradet, Jean-François Monin*

Rappels et notations

Soit \mathcal{V} un ensemble infini dénombrable. Ses éléments sont appelés des variables. On définit inductivement le langage Λ des λ -termes par

$$\Lambda ::= x \mid \Lambda\Lambda \mid \lambda x.\Lambda \quad \text{avec } x \in \mathcal{V}.$$

Attention, l'application d'une fonction f à son argument x ne se note *pas* $f(x)$ mais $f x$, ou $(f x)$ pour regrouper en cas d'ambiguïté. Les parenthèses sont utilisées uniquement pour regrouper.

On représente une fonction à deux arguments x et y et de corps T par $\lambda x.(\lambda y.T)$. Justification : en appliquant cette expression à un premier argument E_1 , on obtient $(\lambda x.(\lambda y.T))E_1$; puis en l'appliquant à un second argument E_2 , on obtient $((\lambda x.(\lambda y.T))E_1)E_2$ qui se réduit comme désiré, c.f. exercice suivant.

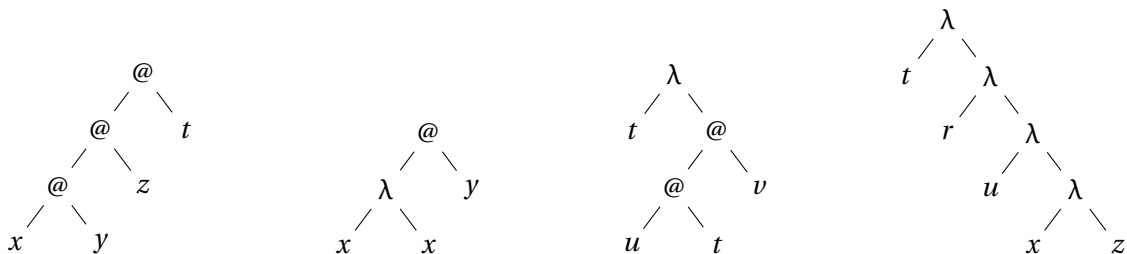
Exercice préliminaire. Dessiner l'arbre syntaxique correspondant, en prenant par exemple pour T le terme $x(xy)$, voire $x + x + y$ dans un λ -calcul avec arithmétique ; on peut sans inconvénient laisser E_1 et E_2 tels quels. Effectuer alors deux β -réductions successives sur cette expression, ce qui donne $(\lambda y.T[x := E_1])E_2$, puis $T[x := E_1][y := E_2]$ et observer que comme attendu, c'est le corps T de la fonction dans lequel x et y ont été respectivement substitués par E_1 et E_2 .

On adopte alors les abréviations et conventions suivantes :

- $\lambda x.\lambda y.T$ et $\lambda xy.T$ sont des notations abrégées pour $\lambda x.(\lambda y.T)$
- l'application associe à gauche : UVW est la notation abrégée de $(UV)W$; si l'on veut indiquer l'application de U au résultat de l'application de V à W , il faut écrire : $U(VW)$
- l'application a priorité sur l'abstraction : $\lambda x.UV$ désigne $\lambda x.(UV)$ et non pas $(\lambda x.U)V$.

1 Arbres syntaxiques

1. Donner le λ -terme correspondant à chacun de ces arbres syntaxiques :



2. Pour chacun des termes suivants, donner l'arbre qui lui correspond :

1/ $(\lambda x.(\lambda v.x))$

3/ $(x (y t))$

5/ $((x y) (((z t) t) v))$

2/ $(\lambda x.(x y))$

4/ $(\lambda v.(\lambda t.(\lambda r.((u x) z))))$

6/ $((((\lambda v.((u t) v) r)) (y z)) (\lambda x.x))$

3. Simplifier au maximum le parenthésage des termes de l'exercice 2.

2 Abréviations

Donner la notation complètement parenthésée et non abrégée des λ -termes suivants :

*Enseignants ayant collaboré les années précédentes : Clément Charpentier, Catherine Parent-Vigouroux

1/ $(x y z)$

2/ $\lambda v.u x z$

3/ $\lambda v x y.u$

4/ $\lambda v t r.(u x z)$

5/ $\lambda x.(x (\lambda y.y x))$

6/ $(u x (y z)(\lambda v.v y))$

7/ $(\lambda x y z.(x z (y z)) u v w)$

8/ $(w (\lambda x y z.(x z (y z)) u v))$

3 Définitions de fonctions

Rendre à chaque terme sa définition en français.

1/ $\lambda x.x$

2/ $\lambda x.y$

3/ $\lambda x.\lambda v.x$

4/ $\lambda x.\lambda y.x y$

5/ $\lambda x.x x$

1/ Fonction à deux arguments, et qui rend le premier d'entre eux.

2/ Fonction identité.

3/ Fonction qui applique son argument à lui-même.

4/ Fonction à deux arguments, qui rend l'application du premier au deuxième.

5/ Fonction à un argument, et qui rend une valeur extérieure ne dépendant pas de cet argument.

4 Variables libres / liées

Pour chaque λ -terme, donner les occurrences libres et liées, puis marquer la portée de chaque variable x .

1/ $\lambda x.x y$

2/ $\lambda v x y.u v y$

3/ $\lambda x.x (\lambda y.y x)$

4/ $(\lambda x y z.x z (y z)) u x w$

5/ $u x (y z) (\lambda x.x y)$

5 Redexes et Réduction

Souligner les redexes des λ -termes suivants puis les réduire jusqu'à leur forme normale.

1/ $(\lambda x.x x) (\lambda y.x) (\lambda y.x)$

2/ $(\lambda y.x) ((\lambda y.x) (\lambda y.x))$

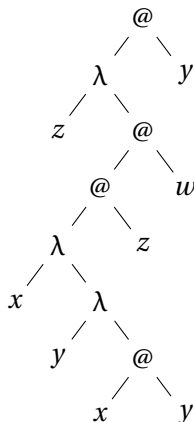
3/ $\lambda y.x (\lambda y.x) (\lambda y.x)$

4/ $\lambda y.(\lambda y.x) (\lambda y.x) x$

6 Capture de variable libre

1/ Le λ -terme $(\lambda x y.x y) y$ comporte un redex. Le réduire, avec ou sans capture de variable libre, et donner la signification intuitive des fonctions obtenues. Laquelle faut-il retenir?

2/ Repérer les redexes du λ -terme $(\lambda z.(\lambda x y.x y) z w) y$ soit :



Le réduire en utilisant différentes stratégies de réduction et observer ce qui se passe si on oublie de se prémunir contre la capture des variables libres.

7 Évaluations en Coq

Coq est un assistant à la preuve qui embarque un λ -calcul typé avec constantes que nous utiliserons plus tard. Nous avons défini en Coq le λ -calcul pur et non typé comme des expressions de type `lexp`. Les λ -abstractions $\lambda x.E$ se notent avec un λ puis un point haut ou avec un `\` puis un point haut un tilde, ce qui donne comme possibilités $\lambda x \cdot E$, $\backslash x \cdot E$ et $\backslash x \tilde{\cdot} E$. Les variables autorisées sont limitées aux minuscules de l'alphabet (a, b, c, \dots, z). Le point haut se saisit sur clavier AZERTY par la combinaison : `AltGR +` : Pour le λ , vous pouvez procéder par copié-collé.

Par exemple, le λ -terme $\lambda v.u x z$ s'écrit comme l'expression `lexp` :

`\v · u x z` ou `\v · u x z` ou `\v ~ u x z`

L'expression abrégée $\lambda v x y.u$ peut s'écrire comme l'expression `lexp` abrégée :

`\v x y · u` ou `\v x y · u` ou `\v x y ~ u`

Nous allons utiliser le système de preuves Coq pour vérifier la bonne construction des λ -termes et les évaluer. Récupérer préalablement le fichier `untypedLC.vo` sous votre répertoire de travail. Pour lancer Coq, sous Linux, taper

`coqide`

Ouvrez un nouveau fichier et faites le commencer par la requête

`Require Import untypedLC.`

qui importe la définition des expressions `lexp`.

Nota Bene : *Toute phrase Coq se termine par un point.*

Pour donner un nom (ici `app`) au λ -terme $\lambda x.\lambda y.x y$, on utilise la commande

`Definition app := \x · \y · x y.`

Pour voir ce terme et son type on utilise la commande

`Print app.`

Pour définir le λ -terme $(\lambda x.\lambda y.x y) a b$ on peut :

- soit procéder directement : `Definition t1 := (\x · \y · x y) a b.`
- soit sous forme abrégée : `Definition t2 := (\x y · x y) a b.`
- soit utiliser la définition précédente : `Definition t3 := app a b.`

Pour voir la forme dépliée de `t3` (par exemple), utiliser la commande

`Compute t3.`

On peut également obtenir (toujours par la commande `Compute`, voir-ci-dessous) des informations sur un terme ou effectuer des réductions au moyen des fonctions suivantes.

- `vlibres` : rend l'ensemble des variables libres d'une `lexp`;
- `vliees` : rend l'ensemble des variables liées d'une `lexp`;
- `pos_redexes` : rend l'ensemble des positions de redexes d'une `lexp`;
- `see_redex n` : rend le redex numéro n d'une `lexp`;
- `red_beta n` : rend une `lexp` après β -réduction de son redex numéro n ;
- `red1_cbn` : réduit le redex le plus externe et le plus à gauche (appel par nom);
- `red1_cbv` : réduit le redex le plus interne et le plus à droite (appel par valeur);
- `red_cbn` : réduit en appel par nom l'expression jusqu'à sa forme normale;
- `show_cbv` : réduit en appel par valeur l'expression jusqu'à sa forme normale;
- `show_cbn` : montre les étapes de réduction en appel par nom de l'expression jusqu'à la forme normale;
- `show_cbv_r` : montre les étapes de réduction en appel par valeur partant de la droite de l'expression jusqu'à la forme normale;
- `show_cbv_l` : montre les étapes de réduction en appel par valeur partant de la gauche de l'expression jusqu'à la forme normale;
- `show_wcbn`, `show_wcbv`, etc. : variantes des réductions précédentes sans réduction sous les λ ;
- `fnorm` : rend `true` si la `lexp` est sous forme normale;
- `eq_lexp` : rend `true` si deux `lexp` sont égales syntaxiquement (modulo renommage);
- `equiv_lexp` : rend `true` si deux `lexp` sont équivalentes (i.e., ont la même forme normale).

Pour utiliser ces fonctions, on écrira `Compute (fct exp)`.

ou plus simplement `Compute fct exp`.

Par exemple `Compute show_cbn t3`.

On peut donner un nom à un terme obtenu par β -réduction comme illustré sur l'exemple suivant.

```
Definition t3_suiv := red_beta 1 t3.
```

```
Compute t3_suiv.
```

Exercices

- 1/ Vérifier pour deux expressions de la section 4 vos résultats pour les variables libres et liées.
- 2/ Vérifier pour deux expressions de la section 5 vos résultats pour l'ensemble des redexes et la réduction.
- 3/ Observer la réduction (par `pos_redexes`, `see_redex`, `red_beta`, puis par `show_cbn`, `show_cbv`) des deux expressions de la section 6 et l'absence de capture.
- 4/ Traduire les λ -termes suivants en expressions `lexp`

— $(\lambda x y. x) (\lambda a. a) ((\lambda b. b)(\lambda c. c))$

Definition `exp1` := ...

— $(\lambda x. xx) ((\lambda a. a) (\lambda b. b))$

Definition `exp2` := ...

Réduire `exp1` et `exp2` en appel par nom (`show_cbn`) et en appel par valeur (`show_cbv`). Noter les points communs et les différences. Quelle est selon vous la meilleure stratégie (`cbv` ou `cbn`)? Essayer également `show_wcbn` et `show_wcbv`.

Exercices optionnels

- Quelle est la forme normale des λ -termes suivants?
 - $(\lambda x. xx)(\lambda x. xx)$
 - $(\lambda x. y)((\lambda x. xx)(\lambda x. xx))$
- les deux λ -termes suivants sont-ils équivalents?
 - $(\lambda t o. t o t o)$ et $(\lambda p a. p a p a)$
 - $(\lambda t. t o t o)$ et $(\lambda m. m o m o)$