

LIFPF – Programmation fonctionnelle

TD2 – λ -calcul et OCaml

Licence informatique UCBL – Printemps 2025

Exercice 1 : λ -calcul et OCaml

En OCaml, on peut utiliser la notation `fun x -> ...` pour décrire une fonction anonyme qui prend en argument `x` et renvoie ce qui est calculé par l'expression se trouvant après la flèche. Cette notation est donc un bon moyen de traduire la notation $\lambda x. \dots$. Par exemple, $\lambda x. x$ se traduit par `fun x -> x`, $\lambda x. \lambda y. (x + y)$ se traduit par `fun x -> fun y -> x + y` et $\lambda x. \lambda y. (x y)$ se traduit par `fun x -> fun y -> x y`.

1. Typé, puis traduire les termes suivants du λ -calcul vers OCaml.
 - a. $\lambda x. (x + 2)$
 - b. $\lambda x. \lambda y. ((2 * x) - y)$
 - c. $\lambda x. ((x 2) + (x 3))$
 - d. $\lambda x. (x \lambda y. (y + 3))$

2. Soit le code OCaml suivant :

```
(fun u -> fun y -> u (fun v -> v)) (fun x -> x y) 3
```

Ce code produit l'erreur suivante : `Error: Unbound value y.`

- a. Traduire ce code en λ -calcul, puis donner les variables libres du terme obtenu et faire le lien avec l'erreur ci-dessus.
- b. β -réduire le terme en prêtant une attention particulière aux renommages nécessaires. Que peut-on en déduire vis-à-vis des différentes occurrences de `y` dans le code OCaml de départ ?

Exercice 2 : Codage d'arguments multiples

On ajoute au λ -calcul le constructeur de type \times . On considère de plus les fonction prédéfinies suivantes, avec leur type et les règles de réduction dédiées :

- $\text{paire} : \tau \rightarrow \tau' \rightarrow (\tau \times \tau')$
- $\text{fst} : (\tau \times \tau') \rightarrow \tau$
- $\text{snd} : (\tau \times \tau') \rightarrow \tau'$
- $(\text{fst} (\text{paire } a \ b)) \rightsquigarrow a$
- $(\text{snd} (\text{paire } a \ b)) \rightsquigarrow b$

1. Montrer que $(\lambda p. ((\text{fst } p) + (\text{snd } p)) ((\text{paire } x) \ y))$ se réduit sur le même terme que $((\lambda p_1. \lambda p_2. (p_1 + p_2) \ x) \ y)$. Discuter sur les similitudes et les différences entre ces deux stratégies de passage d'arguments multiples dans une même fonction.
2. Coder `paire`, `fst` et `snd` en OCaml en utilisant un `match` pour les deux derniers.
3. (Pour aller plus loin) On se donne le codage suivant en λ -calcul :

— $\text{paire} = \lambda x. \lambda y. \lambda s. ((s \ x) \ y)$

— $fst = \lambda p.(p \ \lambda u.\lambda w.u)$
— $snd = \lambda p.(p \ \lambda u.\lambda w.w)$

Montrer que ce codage correspond aux règles de réductions données au début de l'exercice.

Exercice 3 : Filtrage de motif OCaml

On considère les types OCaml définis par le code suivant :

```
type couleur = Pique | Coeur | Carreau | Trefle;;  
type valeur = As | Roi | Dame | Valet | Nombre of int;;  
type carte = Normale of couleur * valeur | Jocker;;
```

1. Donner un exemple de carte qui n'existe pas mais qui est représentable avec ces types.
2. Écrire une fonction `valeur_blackjack` de type `carte -> int` qui donne la valeur d'une carte au Blackjack, à savoir : 11 pour les as, 10 pour chaque figure, autant que leur nombre pour les cartes numérotées et enfin 0 pour les jockers ou les cartes qui n'existent pas. On fera deux versions de cette fonction : une avec des `match` imbriqués pour gérer les as, les cartes inexistantes et les figures et une qui utilise le filtrage de motif en profondeur, c'est-à-dire qui filtre sur des constructeurs imbriqués. Dans les deux cas, on pourra utiliser un `if` pour écarter les cartes inexistantes. Pour la deuxième version, proposer une valeur correspondant à chaque cas du `match`.
3. Écrire une fonction `valeur_belote` de type `carte -> couleur -> int` qui donne la valeur d'une carte à la belote en fonction de la couleur d'atout. Par défaut l'as vaut 11, le 10 vaut 10, le roi 4, la dame 3, le valet 2 et les autres cartes 0. Cependant si la couleur de la carte est la couleur d'atout, le valet vaut 20 et le 10 vaut 14 (les autres cartes gardent la même valeur). On codera la fonction avec un seul `match`, mais avec des `if` quand cela est nécessaire. Expliquer pourquoi on ne peut pas utiliser le paramètre de la couleur d'atout directement dans les cas du `match`.