TD3

mercredi 21 septembre 2022 09:48

7 Ordre supérieur

On appelle *fonction d'ordre supérieur*, ou *fonctionnelle* une fonction qui prend des fonctions en argument ou qui rend une fonction.
Par exemple, la fonction qui prend en argument deux fonctions et qui retourne la somme de celles-ci s'écrit de la façon suivante en OCaml :

let *somme_fonctions f g* = fun $x \rightarrow (f\ x) + (g\ x)$ ('a -> int) -> ('a -> int) -> 'a -> int

Exercice 89 : somme de fonctions

Indiquer deux autres écritures de la fonction *somme_fonctions*.

let somme_fonctions f = fun g -> fun x -> (f x) + (g x)

let somme_fonctions f g x = (f x)+(g x)

let somme_fonctions = fun f -> fun g -> fun x -> (f x) + (g x)

let f = fun
let g = fu
somme_fon

Exercice 90 : composition de fonctions

— Donner la définition d'une fonction qui rend la composée de deux fonctions de type `'a -> 'b` et `'b -> 'c`.
— Donner la définition d'une fonction qui rend la composée de deux fonctions de types `'a -> 'b * 'c` et `'b -> 'c -> 'd`.

let func f g x = f (g x)

let func2 = fun f g x -> let (y1, y2) = f x in g y1 y2

let (composi° : (
= fun f g x -

Note: let fst (a,
let snd (_,

Exercice 91 : Préparation

— Écrire une fonction OCaml qui calcule pour toute fonction f, avec $n \in \mathbb{N}$: $\prod_{i=1}^{n} f(i)$.
— Utiliser la fonction précédente pour définir la fonction factorielle.

let rec produit_fonction_un_to_n = fun f n -> if n = 1 then f 1 else

let f = fun x -> x
let fact = fun n -> produit_fonction_un_to_n f n

Exercice 68 : Preuve sur les arbres (Examen 2010 15 points)

Nous considérons les arbres binaires d'entiers de type :

```
type abr = F | N of abr * int * abr
```

1. (2 points) Écrire une fonction `taille` qui prend un arbre binaire et rend le nombre de nœuds de cet arbre, sachant qu'une feuille a une taille nulle.
2. (3 points) Écrire une fonction `double` qui prend un arbre binaire a et rend un arbre de même structure que a dont la valeur de chaque nœud est le double de la valeur du nœud correspondant de a.
3. (10 points) Prouver que pour tout arbre binaire a : `taille(double a) = taille a`

1) let rec nbn = fun abr -> match abr with
| F -> 0
| N(a1, x, a2) -> nbn(a1) + nbn(a2) + 1

2) let rec double = fun abr -> match abr with
| F -> F
| N(a1, x, a2) -> N (double a1, 2x, double

3)

F -> t = 0 $t_0 = 0$

N(a1, x, a2) -> t = 1 + ma1 + ma2 = 1 2 3 ; t_0 = 1 + ma1 + ma2

Par rec structurelle ✓

Exercice 92 : curryfication

Une fonction à plusieurs arguments $a_1 \dots a_n$ peut se ramener à une fonction à un seul argument de deux manières :

1. une fonction prenant en argument un n-uplet $(a_1 \dots a_n)$;
2. une fonction prenant en argument a_1 et rendant une fonction, prenant elle-même en argument a_2 et rendant une fonction … prenant elle-même en argument a_n et rendant un résultat dépendant de $a_1 \dots a_n$.

Dans cet exercice on prend $n = 2$.

— Écrire la fonctionnelle *curry* prenant en argument une fonction dans le premier style et rendant sa représentation dans le second.

— Écrire la fonctionnelle *uncurry* effectuant la transformation inverse.

— Vérifier que pour tout f, *curry* (*uncurry* f) et f sont extensionnellement égales :

$$\forall xy, curry\ (uncurry\ f)\ x\ y = f\ x\ y.$$

— Formaliser et démontrer que pour tout f, *uncurry*(*curry* f) et f sont extensionnellement égales.

let curry : ('a * 'b -> 'c) ->
(fun g -> F(x,y))

let uncurry : ('a -> 'b -> '
-> F x y

curry (uncurry F) x y =
= F x y

uncurry(curry F) = F
<=> uncurry ((fun g -> F
<=> F = F

correc°:

let curry : ('a * 'b -> 'c) -> ('a -> 'b -> 'c) = fun $\underset{'a*'b->'c}{F}$
-> fun $\underset{'a}{x}$ $\underset{'b}{y}$ -> $\underbrace{F(x,y)}_{'c}$

let uncurry : ('a -> 'b -> 'c) -> ('a * 'b -> 'c) = fun $\underset{'a->'b->'c}{F}$
-> fun $\underset{'a*'b}{c}$
-> let $(\underset{'a}{x}, \underset{'b}{y})$ = c in $\underbrace{F\ x\ y}_{'c}$

Démontrer :

2) uncurry(curry F) = F'

1) curry(uncurry F) = F'

curry (uncurry F) = $[\lambda F.\ \lambda xy.\ F(x,y)]$ (uncurry F)

$= \lambda xy.(\text{uncurry } F')(x,y)$

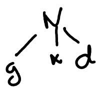
$= \lambda xy.\ F'\ x\ y$

$\forall u, v,\ \text{curry}(\text{uncurry } F')\ u\ v = F'\ u\ v$

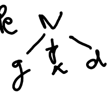
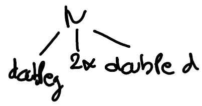
Soient u et v ; $(\lambda xy.F'\ x\ y)\ u\ v \overset{?}{=} F'\ u\ v$

---

taille : F = 0

taille :  = taille g + 1 + taille d

double F = F

double  = 

$\forall a$, taille (double a) = taille a

P F : taille (double F) $\overset{?}{=}$ taille F

= taille (F)

P() → taille (g) + 1 + taille (d) = taille (d (g)) + 1 + taille (d (d))

= P_g

= P

par HR :

```
let rec insert = fun l e -> match l with
                        | [] -> e::[]
                        | x::l1 -> x::(insert (l1 e))

let rec concatenate = fun l1 l2 -> match l2 with
                        | [] -> l1
                        | x::l3 -> concatenate (insert (l1 x)) l2

let rec concatenate_v2 = fun l1 l2 -> match l1 with
                        | [] -> l2
                        | x::l3 -> x :: concatenate_v2 l3 l2

let rec app = fun l1 l2 => match l1 with
                        | [] -> l2
                        | x :: l3 -> x :: app l3 l2
```