TD 1 : Typage de programmes fonctionnels

1 Parenthésage

Exercice 1 (Parenthésage d'expressions)

Rappel :

— L'application associe à gauche : $f\ a_1 \ldots a_{n-1}\ a_n = (((f\ a_1) \ldots a_{n-1}) a_n)$

— L'abstraction associe à droite :
`fun` x_1 `->` `fun` x_2 `->` `...fun` x_n `->` `e` = `(fun` x_1 `->` `(fun` x_2 `->` `...(fun` x_n `->` `e)))`

— L'application est prioritaire sur `,` :
$f\ a,\ b = ((f\ a), b)$

— ...et `,` est prioritaire sur l'abstraction : `fun x -> a, b` = `fun x -> (a, b)`

1. Parenthésez les expressions suivantes :
   (a) `f 2 3`
   (b) `fun x -> fun y -> f x y`
   (c) `(fun x -> x, 3)`
2. Enlevez un maximum de parenthèses des expressions suivantes :
   (a) `(f (g x))`
   (b) `((f g) x)`
   (c) `(fun x -> (fun y -> (f y)) x)`
   (d) `(fun x -> (fun y -> ((f y) x)))`
   (e) `((fun x -> x), 3)`

Exercice 2 (Parenthésage de types) *Rappel :*

— `->` associe à droite

— `*` est prioritaire sur `->`

1. Parenthésez les types suivants :
   (a) `int * int -> bool -> int`
   (b) `(int -> int * int) -> bool * int`
2. Lesquels des types suivants sont équivalents ?
   (a) `int -> bool -> int * bool`
   (b) `(int -> bool) -> int * bool`
   (c) `int -> (bool -> (int * bool))`
   (d) `(int -> bool -> int) * bool`
   (e) `(int -> bool) -> (int * bool)`

2 Vérification de typage de programmes fonctionnels

Faites la vérification du typage pour les expressions suivantes sous l'environnement `Env = [(b, bool); (x, int); (f, int -> int -> int); (p, int -> bool)]` en supposant le typage traditionnel des constantes et des opérateurs arithmétiques et booléens :

Vous pouvez vous assurer d'avoir trouvé le bon type en faisant la vérification des expressions sous Caml, comme vu en cours.

1. `(f x 3)`
2. `(f x)`
3. `fun (y : int) -> (f x y)`
4. `fun (y : int) -> (f x)`
5. `fun (y : int) -> (p y)`
6. `fun (y : int) -> (p b)`
7. `(p 3) = b`
8. `fun (x : int) -> p x, 3`
9. `(fun (x : int) -> p x), 3`
10. `fun (g : (int -> int) -> bool) -> g (f 4)`
11. `(fun (b: bool) -> b) (p 5)`

Quelle est la réaction de l'interpréteur Caml si vous essayez de comparer deux fonctions, par exemple : `(fun x -> x) = (fun y -> y);;`