

TD de *Logique et Circuits* n° 1
(Correction)

Introduction à Caml

Exercice 1 Quel est le résultat (type et valeur) des commandes suivantes ? (Certaines peuvent comporter des erreurs.)

```
# 3 + 7;;
# 3.5 + 4.3;;
# 4 / 3;;
# 3 ** 2;;
# 5 = 6;;
# ceil((if 3 < 4 then 3.4 else 2.3) -. 4.5);;
```

Correction :

```
- : int = 10
This expression has type float but is here used with type int
- : int = 1
This expression has type int but is here used with type float
- : bool = false
- : float = -1
```

Exercice 2 *i)* On effectue les deux commandes suivantes :

```
let a = let a = 3 and b = 2 in let a = a + b and b = a - b in a * a - b * b;;
let b = 2 in b + a * b;;
```

Quel est le résultat de chaque commande ?

ii) *Albert a 15 ans, Béatrice en a 19. L'âge de Charles est la somme de ceux d'Albert et Béatrice, mais l'âge de la maison qu'ils habitent est 3 fois la somme des âges de Béatrice et de Charles. Écrire une expression Caml qui traduise cet énoncé et dont la valeur est l'âge de la maison.*

iii) On tape successivement les commandes suivantes en Caml. Quelles sont les valeurs successives des variables ? (Certaines lignes peuvent provoquer des erreurs.)

```
let y = let x = 4 in x + 1 and z = 25;;
let x = let t = y + z in t / y;;
let y = let z = t - y in x * z;;
let t = let x = 3 in z * x;;
let t = t / x;;
```

Correction : *i)* La première commande a pour résultat `val a : int = 24`. Elle est équivalente à

```
let a = let x = 3 and y = 2 in let z = x + y and t = x - y in z * z - t * t;;
```

Les valeurs des variables locales sont $x = 3$, $y = 2$, $z = 5$ et $t = 1$. Le résultat de la seconde est `- : int = 50` (b est une variable locale).

ii) `let a = 15 and b = 19 in let c = a + b in (b + c) * 3;;`

Finalemment, $c = 34$ et le résultat de l'expression est 159.

iii) `let y = let x = 4 in x + 1 and z = 25;;`

Le second `let` est clos par le `in`. Le `and` ne peut donc se rapporter qu'au premier `let`. Cette expression est donc la déclaration des variables y et z . Par contre, x est une variable locale. À la suite de cette ligne, on a $y = 5$ et $z = 25$.

`let x = let t = y + z in t / y;;`

Cette expression est la déclaration de la variables x . t est un variable locale de valeur 35, donc, après cette ligne, on a $x = 6$, $y = 5$ et $z = 25$.

`let y = let z = t - y in x * z;;`

La variable t utilisée dans cette expression n'est pas déclarée. On obtient une erreur.

`let t = let x = 3 in z * x;;`

On déclare ici la variable t . Une variable locale x est déclarée. Elle masque la variable globale. Après cette ligne, les valeurs des variables sont $x = 6$, $y = 10$, $z = 25$ et $t = 75$.

`\tt let t = t / x;;`

On obtient $x = 6$, $y = 10$, $z = 25$ et $t = 12$.

Exercice 3 Soit le polynôme $P(x) = x^2 + 5x + 4$. Écrire une fonction qui calcule la valeur de ce polynôme pour tout flottant x .

Correction :

`let evalP x = x*.x+.5*.x+.4.;;`

ou

`let evalP x = x**2.+5*.x+.4.;;`

ou encore (suivant la méthode de Horner) :

`let evalP x = x *. (x +. 5.) +. 4.;;`

Exercice 4 Écrire une fonction qui prend deux arguments de même type et retourne un couple contenant le plus petit comme première composante et le plus grand comme seconde.

Correction : `let sort2 x y = if x < y then (x,y) else (y,x);;`

Faire remarquer aux étudiants qu'avec la syntaxe

`let sort2 (x,y) = if x<y then (x,y) else (y,x);;`

la fonction ne prend qu'un argument (qui est un couple).

Exercice 5 On définit la fonction `xor` (« ou » exclusif) dont l'argument est un couple de deux booléens et qui retourne la valeur *vrai* si et seulement si un seul des deux booléens est vrai. Écrire la table de la fonction `xor`. Comment définir la fonction `xor` à l'aide des opérateurs `or`, `&` et `not`? Écrire la fonction Caml correspondante.

Écrire une fonction équivalente qui s'applique sur un couple de deux booléens. Quel opérateur agit en Caml sur les booléens comme la fonction `xor`?

Correction :

| | | | |
|------------------|--|------|------|
| <code>xor</code> | | vrai | faux |
| vrai | | faux | vrai |
| faux | | vrai | faux |

`let xor (x,y) = (x & (not y)) or (y & (not x));;`

Version avec deux arguments :

`let xor x y = (x & (not y)) or (y & (not x));;`

`let xor (x,y) = x <> y;;` (i.e. `xor` est `<>`)

Exercice 6 Que fait la fonction suivante? Quelle est son type?

```
let f(x) = char_of_int ((int_of_char x) + 1);;
```

Correction : Cette fonction retourne le caractère qui a le code suivant celui du paramètre. (S'il s'agit d'une lettre, on obtient la lettre suivante dans l'ordre alphabétique.) Elle est de type `char -> char`. Dans la mesure où les codes de caractères sont compris entre 0 et 255, l'appel de la fonction `f` sur le caractère `'\255'` déclenche une erreur.

Exercice 7 Écrire une fonction qui prend un argument n de type `int` et renvoie 0 si n vaut 0 ou 1 et $n - 2$ sinon. On écrira d'abord cette fonction avec `if...then...else` puis avec `match`.

Correction :

```
let moins2 n = if n=0 or n=1 then 0 else n-2;;
```

et

```
let moins2 n =  
  match n with  
  | 0 -> 0  
  | 1 -> 0  
  | _ -> n-2;;
```

Exercice 8 Considérons la fonction suivante :

```
let f x y =  
  if y = 3 then 5 else if x then y + 1 else 0;;
```

Quel est le type des paramètres? Traduire cette fonction en utilisant `match`.

Traduire la fonction suivante en utilisant `if...then...else` :

```
let g(x, y) =  
  match (x - y, y) with  
  | (0, _) -> x  
  | (_, 0) -> 0  
  | _      -> x + y;;
```

Correction : Le premier paramètre est un booléen (utilisé dans `if x then...`), le second est un entier.

```
let f x y =  
  match (x, y) with  
  | (_, 3) -> 5  
  | (true, _) -> y + 1  
  | _ -> 0;;
```

Tester si $x - y$ vaut zéro revient à tester l'égalité de x et de y . La définition de g est donc équivalente à :

```
let g(x,y) = if x = y then x else if y = 0 then 0 else x + y;;
```

Exercice 9 On considère l'expression suivante, extraite d'un programme qui peut comporter des définitions antérieures que l'on ne connaît pas ici.

```

let x = 2.5 and y = true in
if f x (g(y, "true")) > 3
then let z = g(y, t y) in (z/.3.14)*.i(x)
else let w = j("true", 'a') in k w;;

```

Énumérer les différents identifiants qui apparaissent dans cette expression. Déterminer leur type ainsi que celui de l'expression.

Correction : Les différents identifiants sont : `x y f g z t i w j k`.
Supposons que les fonctions ne sont pas polymorphes.

Les types de ces symboles sont :

1. `x` est un `float`.
2. `y` est un `bool`.
3. `f` est une fonction qui s'applique à une suite de deux paramètres : le premier est un `float` (`x`), le second est le type de retour de `g`, pour l'instant inconnu, appelons-le `T`. Le type de retour de `f` est `int` (comparaison avec 3).
4. `g` est une fonction qui s'applique à un couple de deux paramètres, le premier est un `bool`, le second un `string`. On ne peut pas encore déterminer `T`.
5. `z` est un `float` (`z/.3.14`), et est du type de retour de `g`. Donc `T=float`.
6. `t` est une fonction dont l'unique paramètre est de type `bool` (`y`). Son type de retour est celui du second paramètre de `g`, soit `string`.
7. `i` est une fonction dont l'unique paramètre est de type `float` (`x`). Son type de retour est `float`.
8. `w` est de type inconnu, disons `U`.
9. `j` est une fonction qui s'applique à un couple de deux paramètres, le premier de type `string`, le second de type `char`. Son type de retour est `U`.
10. `k` est une fonction dont le paramètre est de type `U` et le type de retour est `float` (puisque le type qui suit le `else` doit être identique à celui qui suit le `then`).

On ne peut pas déterminer le type `U`. Pour résumer, on a :

Variables locales :

Variables globales :

```

x : float          f : float -> float -> int
y : bool          g : bool * string -> float
z : float         t : bool -> string
w : U             i : float -> float
                  j : string * char -> U
                  k : U -> float

```

Exercice 10 Écrire une fonction `car` qui, à tout couple (x, y) d'éléments de type quelconque formé d'un élément de type quelconque `T` et d'un entier, associe une fonction qui prend un élément de type `T` et retourne `y` si l'argument est égal à `x` et 0 sinon.
Quel est le type de la fonction `car` ?

Correction :

```

let car (x,y) z = if x=z then y else 0;;

```

On peut aussi écrire :

```

let car (x,y) = function z -> if x=z then y else 0;;

```

La fonction est de type $(\text{'a} * \text{int}) \rightarrow \text{'a} \rightarrow \text{int}$. On peut évidemment voir `car` comme une fonction qui prend une suite de deux paramètres (le premier étant un couple) et qui teste si le second est égal à la première composante du couple en renvoyant, si c'est le cas, la seconde composante (et 0 sinon, ce qui force la seconde composante à être un entier).

Exercice 11 Préciser les types des fonctions `f`, `g` et `h` définies ci-dessous :

```
let f x (a, b) = (x a, x b);;
let g x y = (f x (2, 1), f y (true, false));;
let h x y (a, b) = f x (f y (a, b));;
```

Correction : `f` consiste à appliquer `x` à chacune des composantes du couple. `x` est donc une fonction polymorphe de type $\text{'a} \rightarrow \text{'b}$, ce qui implique que `a` et `b` sont de type 'a . Finalement `f` est de type $(\text{'a} \rightarrow \text{'b}) \rightarrow \text{'a} * \text{'a} \rightarrow \text{'b} * \text{'b}$.

Dans la définition de `g`, `x` sera de type $\text{int} \rightarrow \text{'a}$ et `y` de type $\text{bool} \rightarrow \text{'b}$; donc `g` est de type $(\text{int} \rightarrow \text{'a}) \rightarrow (\text{bool} \rightarrow \text{'b}) \rightarrow (\text{'a} * \text{'a}) * (\text{'b} * \text{'b})$

Dans la définition de `h`, `x` est de type $\text{'a} \rightarrow \text{'b}$. Il doit être appliqué au résultat de `f y (a,b)`, donc `y` est de type $\text{'c} \rightarrow \text{'a}$ et `a` et `b` sont de type 'c .

Donc `h` est de type $(\text{'a} \rightarrow \text{'b}) \rightarrow (\text{'c} \rightarrow \text{'a}) \rightarrow \text{'c} * \text{'c} \rightarrow \text{'b} * \text{'b}$

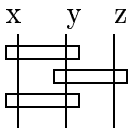
Exercice 12 Que fait la fonction `f2` définie ci-dessous ?

```
let f1 a b =
  if a < b then (a, b) else (b, a);;

let f2 x y =
  function z ->
    let (x, y) = (f1 x y) in
    let (y, z) = (f1 y z) in
    let (x, y) = (f1 x y) in (x, y, z);;
```

Quel est le type de `f2` ?

Correction : `f1` trie les éléments `a` et `b` qui doivent être de même type. `f2` trie les éléments `x`, `y` et `z`. Il s'agit en fait d'un tri "à bulles".



Il est facile de voir que les deux premiers appels de `f1` donnent à `z` la valeur maximale. De même, les deux derniers appels de `f1` donnent à `x` la valeur minimale.