Travaux Pratiques ML No.1

Cours d'Informatique de Deuxième Année

—Deug SSM M4—

Premier TP en Camllight.

Le but de ce *TP* est de comprendre de façon progressive comment *Caml-Light* type les objets qu'on lui donne en argument. On étudie aussi la différence entre variables locales et globales. Certains exercices présentés ne sont pas évidents à priori [1] et il sera intéressant pour vous d'essayer, à titre d'entrainement, d'autres exercices tels que ceux qui sont proposés en *Cours* ou en *TD*.

Types → Fonctions.

▸ **Exercice 2**. Donner un exemple de fonction en *Caml-Light* qui produit l'affichage du type suivant:

```
int -> int -> int
```

. . . . . .

▸ **Exercice 3**. *Idem*

```
(int -> int) -> int
```

. . . . . .

▸ **Exercice 4**. *Idem*

```
int -> int -> int -> int
```

. . . . . .

▸ **Exercice 5**. *Idem*

```
int -> (int -> int) -> int
```

. . . . . .

▸ **Exercice 6**. *Idem*

```
(int -> int -> int) -> int
```

. . . . . .

▸ **Exercice 7**. *Idem* —S'aider de l'exercice 4.

```
(int -> int) -> int -> int
```

. . . . . .

[1] Les exercices commençant par (*) sont plus subtils que les autres qui ne sont souvent que des exercices applicatifs. Les exercices commençant par (**) demandent encore plus d'attention.

▸ **Exercice 8**. *Idem*!

```
('a -> 'b * 'c) -> ('a * 'b) -> 'c
```

. . . . . .

Fonctions → Types.

▸ **Exercice 9**. Quel est le type de la fonction suivante?

```
#let h (f,g) = function x ->  f(g x) ;;
```

. . . . . .

▸ **Exercice 10**. *Idem*

```
#function f -> fun g h -> function x -> h f ((g x+1),true) ;;
```

. . . . . .

Variables locales et globales

▸ **Exercice 11**. Complétez les lignes manquantes:

```
#let x = 1 and z = 2 ;;
```

. . . . . .

. . . . . .

```
#let f = function y -> let z = function x -> x+1 in
#        y+x+(z 2) ;;
```

. . . . . .

```
#f 3 ;;
```

. . . . . .

```
#let x = 2 ;;
x :  int = 2
#f 3 ;;
```

. . . . . .

Exercices supplémentaires†...

▸ **Exercice 12**. Dans cet exercice, on cherche à exprimer la suite de Fibonacci comme somme de deux suites, la première ayant toujours un argument pair et la seconde un argument impair [(2)]. Pour ce faire, on utilisera une définition locale de ces deux suites qui seront nécessairement mutuellement récursives.

▸ **Exercice 13**. On définit une fonction `loop` qui, appliquée à un prédicat P, une fonction F et une variable x, rend la valeur $F^n(x)$ où n est le plus petit entier qui vérifie $P(F^n(x)) = Vrai$:

```
#let rec loop P F x =
#        if P x then x
#        else loop P F (F x) ;;
```

Utiliser `loop` pour définir une fonction `find_power` qui, étant donnés un entier `x` et un entier `n`, retourne la première puissance de `x` supérieure ou égale à `n`.

---

† à faire aujourd'hui pour les plus rapides, à tête reposée pour les autres.

(2) En fait, $Fib(n) = Fib1(n-1) + Fib2(n-2)$ si n est impair, $Fib(n) = Fib2(n-1) + Fib1(n-2)$ sinon.