

Informatique

DS1 correction



1 Fonctions proches du TP

Q01

```
let rec fact n =
    if n = 0
    then 1
    else n * fact (n-1)
```

Q02

```
let rec pow2 n =
    if n = 0
    then 1
    else 2 * pow2 (n-1)
```

Écrire une fonction `pow2_rap : int -> int` telle que `pow2_rap n` renvoie 2^n où $n \geq 0$ en remarquant que :

$$2^{2p} = (2^p)^2 \qquad 2^{2p+1} = 2 \times 2^{2p}$$

Q03

```
let rec pow2_rap n =
    if n = 0
    then 1
    else if n mod 2 = 1
        then 2 * pow2_rap (n-1)
        else let a = pow2_rap (n/2) in a * a
```

Attention ici, si on écrit `pow2_rap (n/2) * pow2_rap (n/2)` cela va faire deux appels bêtement et la complexité sera exponentielle en n.

Q04

1. en utilisant pow2

```
let rec suplog2_aux n k =
    if pow2 k > n
    then k
    else suplog2_aux n (k+1)

let suplog2 n = suplog2_aux n 0
```

2. sans utiliser pow2

```
let rec suplog2 n =
    if n = 0
    then 0
    else 1 + suplog2 (n/2)
```

Q05

```
let rec applique l f =
    match l with
    | [] -> []
    | t::q -> f t :: applique q f
```

Q06

```
let rec nieme l n =
    match l, n with
    | [], _ -> failwith "Liste vide"
    | t::_, 0 -> t
```

```
    | _::q, _ -> nieme q (n-1)
```

2 Fonctions et typage

Q07
```
let evalue f x = f x
```

Q08
```
('a -> 'b) -> 'a -> 'b
```

Q09
```
let compose f g x = f (g x)
```

Q10
```
('a -> 'b) -> ('c -> 'b) -> 'c -> 'b
```

3 Animal de compagnie

```
type pet = Cat | Dog
```

Q11
```
let ennemy p = match p with
    | Dog -> Cat
    | Cat -> Dog
```

Q12
```
let dog_years n = n * 7
```

Q13
```
let cat_years n =
    match n with
    | 1 -> 12
    | _ -> 24 + 5 * (n-2)
```

Q14
```
let pet_years p n =
    match p with
    | Cat -> cat_years n
    | Dog -> dog_years n
```
Notons qu'ici on peut omettre n :
```
let pet_years p =
    match p with
    | Cat -> cat_years
    | Dog -> dog_years
```
Cela revient à imaginer qu'on calcule une fonction qui pourra être utilisée dans tous les cas avec un entier.

4 Couvertures

Q15

Dans le cas dégénéré $n = 0$ il n'y a qu'une couverture : c'est l'ensemble vide. Dans le csa $n = 1$ on ne peut placer qu'une tuile de taille 1.
Sinon on place une tuile de taille 1 et on couvre les $n - 1$ restants ou on place une tuile de taille 2 et on couvre les $n - 2$ restants.
```
let rec couvertures12 n =
    if n = 0 || n = 1
    then 1
```

```
    else couvertures12 (n-1) + couvertures12 (n-2)
```

Q16 Il s'agit de la suite de Fibonacci !

Q17 Pour couvrir un échiquier $2xn$ on a le choix sur la position du premier domino : vertical ou horizontal. Si on le place verticalement, il ne reste comme espace libre que de quoi placer un domino parallèle exactement à coté. Le choix que l'on fait est donc entre un domino horizontal ou deux dominos verticaux. C'est exactement le même choix qu'avec les tuiles précédentes. D'ailleurs on passe de l'un à l'autre en coupant l'échiquier en deux.

Q18 Si on colorie l'échiquier comme au échecs, chaque domino est à cheval sur une case blanche et une case noire. Il faut donc que le nombre de cases blanches de l'échiquier soit égal au nombre de cases noires.
Si $n = 2p+1$, les lignes sont tantôt à p cases noires et $p+1$ cases blanches, tantôt l'inverse. Pour que cela s'équilibre, il faut que le nombre de lignes soit pair.
Si n est pair, l'équilibre est automatique.
La condition est donc d'avoir n ou m pair. Dans ce cas on peut découper l'échiquier en bandes de tailles $2xm$ ou $nx2$ et on sait que l'échiquier restant est couvrable (par exemple uniquement avec des dominos parallèles).
On a donc $n \times m$ admet une couverture $\iff nm$ pair (car n ou m pair $\iff nm$ pair).

Q19
```
let rec couverture tl n =
    let rec aux l = match l with
        | [] -> 0
        | t::q -> (if n>=t then couverture tl (n-t) else 0) + aux q
    in if n = 0 then 1 else aux tl;;
```

Q20 Ici on a quatre tuiles identiques de longueur 1, donc on a quatre choix par cases. Ainsi, s'il y a n cases alors on a 4^n couvertures distinctes.