Un corrigé du TD n° 2 : Récursivité

EXERCICE 1 Il suffit de remarquer sur $f\left(f^{n-1}(x)\right) = f^n(x) = f^{n-1}(f(x))$.

```
let rec itere n f x =
  match n with
   | 0 -> x
   | _ -> itere (n - 1) f (f x)
;;
```

EXERCICE 2 (*Concaténation récursive terminale*)

1.

```
let rec concat liste1 liste2 =
  match liste1 with
   | [] -> liste2
   | tete :: queue -> tete :: (concat queue liste2)
;;
```

Si n_1 et n_2 sont les tailles des deux listes, $C(0, n_2) = 0$ et si $n_1 > 0$,

$$C(n_1, n_2) = C(n_1 - 1, n_2) + \Theta(1).$$

Donc $C(n_1, n_2) = \Theta(n_1)$ (ne dépend pas de n_2).
La récursivité n'est pas terminale.

2.

```
let rec rev_append liste1 liste2 =
  match liste1 with
   | [] -> liste2
   | tete :: queue -> rev_append queue (tete :: liste2)
;;
```

Si n_1 et n_2 sont les tailles des deux listes, $C(0, n_2) = 0$ et si $n_1 > 0$,

$$C(n_1, n_2) = C(n_1 - 1, n_2 + 1) + \Theta(1).$$

Donc

$$\begin{aligned} C(n_1, n_2) &= C(n_1 - 1, n_2 + 1) + \Theta(1) \\ &= C(n_1 - 2, n_2 + 2) + \Theta(1) + \Theta(1) \\ &\vdots \\ &= C(0, n_1 + n_2) + \sum_{k=0}^{n_1} \Theta(1) \\ &= \Theta(n_1) \end{aligned}$$

(ne dépend pas de n_2).
La récursivité est terminale.

3.

```
let rev liste = rev_append liste [];;
```

4.

```
let append liste1 liste2 =
  rev_append (rev liste1) liste2;;
```

Complexité : $C(n_1, n_2) = C_{\texttt{rev\_append}}(n_1, n_2) + C_{\texttt{rev}}(n_1) = \Theta(n_1)$.
Cette fonction n'est pas récursive mais ne fait intervenir que des fonctions récursives terminales.

EXERCICE 3

```
let sigma n =
  (* Invariant de sigma_aux : somme(k, k = 0 .. n) + accu *)
  let rec sigma_aux n accu =
    match n with
     | 0 -> accu
     | _ -> sigma_aux (n - 1) (accu + n)
  in
  sigma_aux n 0
;;
```

EXERCICE 4

```
let length liste =
  (* Invariant de length_aux : longueur(liste) + compteur *)
  let rec length_aux liste compteur =
    match liste with
    | [] -> compteur
    | _ :: queue -> length_aux queue (succ compteur)
  in
  length_aux liste 0
;;
```

EXERCICE 5 (*Ordre de Sarkovski*)

1. On calcule
   - $1988 = 4 \cdot 497$
   - $1989 = 1989$
   - $1990 = 2 \cdot 995$

   donc $1988 \prec 1990 \prec 1989$.
2. Si $n \in \mathbb{N}$, $3 \cdot 2^{n+1} \prec 3 \cdot 2^n$.
   $A = \{3 \cdot 2^n \ ; \ n \in \mathbb{N}\}$ n'a pas d'élément minimal donc l'ordre n'est pas bien fondé. (On a exhibé une suite strictement décroissante.)

EXERCICE 6 (*Caractérisation équivalente d'un ensemble bien fondé*)

Par contraposée :

- Si l'ensemble ordonné n'est pas bien fondé, on a une partie A de E sans élément minimal. On définit alors par récurrence une suite $(a_n)_n$ d'éléments de A avec $a_0 \in A$ quelconque et si, pour un $n \in \mathbb{N}$, on $a_n \in A$, on peut trouver un élément $a_{n+1} \in A$ tel que $a_{n+1} \prec a_n$ car a_n n'est pas minimal, ce qui permet de construire une suite strictement décroissante d'éléments de A.
- Si on a une suite $(a_n)_n$ strictement décroissante, alors $A = \{a_n \ ; \ n \in \mathbb{N}\}$ n'a pas d'élément minimal (tout terme de la suite est strictement minoré par le terme suivant), donc l'ensemble ordonné n'est pas bien fondé.

EXERCICE 7 (*Fonction 91 de McCarthy*)

— $\boxed{\text{Si } n > 100,\ f_{91}(n) = n - 10.}$

— $\boxed{\text{Si } n = 100 - k \in [\![90, 100]\!]}$ avec $k \in [\![0, 10]\!]$, on montre par récurrence simple finie sur k que $\boxed{f_{91}(n) = 91.}$

- C'est vrai pour $k = 0$ car
$$f_{91}(100) = f_{91}(f_{91}(111)) = f_{91}(101) = 91,$$
- Si c'est vrai pour un $k \in [\![0, 9]\!]$, alors
$$f_{91}(100 - (k+1)) = f_{91}(99 - k) = f_{91}(f_{91}(110 - k))$$
avec $110 - k > 100$ donc
$$f_{91}(100 - (k+1)) = f_{91}(100 - k) = 91$$
par hypothèse de récurrence.

On peut aussi se passer de récurrence : si $n \in [\![91, 100]\!]$,
$$f_{91}(n) = f_{91}(f_{91}(n+11)) = f(n+1)$$
car $n + 11 > 100$. Ainsi, pour un tel n, $f_{91}(n) = f_{91}(101) = 91$.
(En fait la récurrence est implicite. Où est-elle ?)

— Puis, $\boxed{\text{si } n = 100 - k \leqslant 100}$ avec $k \in \mathbb{N}$, on montre par récurrence sur k que $\boxed{f_{91}(n) = 91.}$

- C'est vrai pour $k \in [\![0, 10]\!]$ d'après le point précédent et **toutes ces initialisations sont nécessaires**.
- Si pour un $k \geqslant 11$, c'est vrai pour $k - 11$, alors
$$f_{91}(100-k) = f_{91}(f_{91}(111-k)) = f_{91}(f_{91}(100-(k-11))) = f_{91}(91) = 91$$
par par hypothèse de récurrence puis par le cas $k = 9$.

Remarque : C'est de l'induction structurelle sur $\mathbb{N}$ *avec* $\varphi : k \mapsto k - 11$ *et* $A = [\![0, 10]\!]$.

EXERCICE 8 (*Fonction d'Ackermann*)

Pour la fonction d'Ackermann, qui n'est pas récursive terminale, on prouve la terminaison par induction structurelle sur $\mathbb{N}^2$ muni de l'ordre lexicographique (bien fondé).

— La fonction termine sur $A = \{0\} \times \mathbb{N}$ (qui contient le minimum $(0,0)$).

— Si $(n,p) \notin A$, et si la fonction termine pour tout couple $(n',p') < (n,p)$, alors la fonction appelle récursivement :

- soit, si $p = 0$, sur $(n-1,1) < (n,0)$ donc termine,
- soit, sinon, sur $(n,p-1) < (n,p)$ qui termine puis sur

$$(n-1, \texttt{Ackermann}(n,p-1)) < (n,p)$$

qui termine aussi.

Donc la fonction d'Ackermann termine par principe d'induction.

EXERCICE 9 (*Tour de Hanoï*)

Pour les tours de Hanoï avec n disques (voir ci-contre), si on note $D(n)$ le nombre de déplacements, $D(0) = 0$ et si $n \geqslant 1$, il y a deux appels récursifs au rang $n-1$ et 1 déplacement, donc

$$D(n) = 2D(n-1) + 1.$$

Cela se résout en remarquant que

$$D(n) + 1 = 2(D(n-1) + 1).$$

Donc $D(n) + 1 = 2^n$ (suite géométrique) puis $\boxed{D(n) = 2^n - 1.}$

```
let deplacement n origine arrivee =
  "Deplacer le disque " ^ string_of_int n
        ^ " de " ^ origine ^ " a " ^ arrivee ^ "."


let rec hanoi_liste n debut fin intermediaire liste_depl =
  if n = 0 then

    (* Plus de déplacement, on renvoie la liste *)
    liste_depl

  else

    (* Déplacement de n - 1 disque de debut vers interm. *)
    let liste1 = hanoi_liste (n - 1) debut intermediaire fin
                                          liste_depl in

    (* Déplacement du disque num. n de debut vers fin *)
    let liste2 = (deplacement n debut fin) :: liste1 in

    (* Déplacement de n - 1 disque de interm. vers fin *)
    hanoi_liste (n - 1) intermediaire fin debut liste2
;;


let hanoi n =
  rev (hanoi_liste n "gauche" "droite" "milieu" [])
;;
```