Arbres et récursions terminales

Pour calculer la taille d'un arbre, on peut agir classiquement :

```
type 'a arbre = Nil | Node of ('a arbre * 'a * 'a arbre);;

let rec taille a =
  match a with
  | Nil ->0
  | Node(g,_,d)->1+taille g + taille d;;
```

En exercice, nous avons présenté faussement la version suivante comme une récursion terminale :

```
let taille2 a =
  let rec size acc a =
    match a with
    | Nil -> acc
    | Node(g,_,d)->
       size (size (acc+1) g) d
  in aux 0 a;;
```

Malheureusement, ce n'est pas vrai car la pile d'appel n'est pas vidée : on doit garder en mémoire la valeur de d[1] dans la stack frame courante. Les stack frames qui s'empilent du fait de l'appel **aux (acc+1) g** ne peuvent effacer cette valeur car elle est nécessaire au calcul final. En clair, la pile d'appel n'est pas vidée et les stack frames s'accumulent. Oublions donc cette version malheureuse qui ne remplit pas son objectif.

La solution pour obtenir une véritable récursion terminale consiste à utiliser une méthode de programmation appelée *Continuation-passing style* (CPS). Le *flot de contrôle*[2] est passé explicitement en paramètre d'une fonction auxiliaire sous la forme d'une *fonction de continuation*. Par convention, cette fonction de continuation est souvent représentée par un paramètre noté **k**.

Plutôt que d'appeler **taille g** et d'attendre la fin du calcul pour ensuite calculer des valeurs basées sur son résultat, on appelle **aux g (fun resg -> ...)**, où le second argument est une fonction qui indique quoi faire une fois que le résultat (**resg**) est disponible.

```
(*Récursion terminale*)
let taille a =
  let rec aux a k = match a with
    | Nil -> k 0
    | Node(g,_,d) ->
      aux g (fun resg ->
          aux d (fun resd ->
                    k (1 + resg + resd)))
  in aux a (fun x -> x);;
```

1. Ce qu'on met dans la pile d'appel, c'est en fait un pointeur sur le père de d : c'est à dire le premier paramètre de **aux**.

2. C'est à dire l'ordre dans lequel les déclarations, les instructions ou les appels de fonction sont exécutés ou évalués

Ici, la fonction de continuation **k** est simplement l'identité. On constate que la fonction de continuation **fun resg ->...** passée en argument dans le second filtrage reconstruit en quelque sorte l'arbre étudié : sa structure est calquée sur celle de **a**. On peut voir cela comme l'idée selon laquelle on « emmène l'arbre avec soi » durant le parcours. L'accumulateur est alors la fonction construite durant le parcours et, une fois qu'il est complètement déterminé, on l'applique à son unique argument pour obtenir le résultat escompté. Il s'agit bien d'une récursion terminale.

Exercice 1. Écrire la fonction **hauteur (a : 'a tree) : int** qui calcule la hauteur d'un arbre grâce à une fonction de continuation.