TD 3 : Récursivité

Algorithmes et programmation.

2019

Exercice 1 — *Parcours de liste*

Une liste est une suite ordonnée d'entiers. Une liste est caractérisée par son premier élément et le reste de ses éléments (nommés généralement `head` et `tail`.) Ainsi la liste $[4, 2, 1, 7, 2, 3]$ a pour tête 4 et pour queue la liste $[2, 1, 7, 2, 3]$. On suppose que la liste vide s'écrit $\emptyset$. On dispose de 5 fonctions,

- `empty(L)` renvoie vrai si la liste L est vide
- `head(L)` renvoie la tête de L
- `tail(L)` renvoie la queue de L
- `print(i)` affiche l'entier i (suivi d'un saut de ligne)
- `cons(i, L)` construit et renvoie la liste L' dont la tête est i et dont la queue est L.

1. Écrire une fonction récursive qui affiche le contenu de la liste. Montrer ensuite que cette fonction se termine et qu'elle affiche le bon résultat.

▶ **Correction**

Une fonction pourrait être la suivante :

```
1: function PRINT_LIST(L)
ENTRÉES: Une liste L
SORTIES: Affiche le contenu de la liste, dans l'ordre.
2:     Si Non empty(L) Alors
3:         print(head(L))
4:         PRINT_LIST(tail(L))
```

La fonction se termine, on peut montrer plus précisément qu'elle se termine avec au plus $6(|L| + 1)$ instructions où $|L|$ est la taille de la liste.

On appelle instruction :

- un appel et l'exécution de l'une des 5 fonctions décrites au début du sujet (on supposera pour simplifier que l'appel et l'exécution se font en une seule instruction)
- on peut aussi considérer que l'appel à la fonction `print_list` se fait avec une instruction. (Par contre son exécution se fait en plus qu'une instruction.)
- le test `Si Non ... Alors`

Prouvons le par récurrence sur $|L|$:

Si $|L| = 0$ alors L est vide, donc la fonction effectue 2 instructions, le test et l'appel à `empty` à la ligne 2. On a bien $2 < 6(|L| + 1) = 6$.

Sinon, en supposant que la propriété est prouvée pour toute liste de taille n, prouvons la si $|L| = n + 1$. Dans ce cas la liste est non vide. La fonction effectue deux instructions à la ligne 2 ; deux de plus à la ligne 3 (la fonction `print` et la fonction `head`). A la ligne 4, on effectue un appel à `tail`, un appel à `print_list` et toutes les instructions nécessaires pour l'exécution de cette dernière fonction.

Puisque $|L| = n + 1$ alors $|\texttt{tail}(L)| = n$. Par hypothèse de récurrence, cette exécution se termine en au plus $6(n+1)$ instruction. Donc `print_list`(L) nécessite au plus $6+6(n+1) = 6(n + 2)$ instructions.

Donc la propriété est bien prouvée pour $|L|$. Par récurrence, la fonction se termine en au plus $6(|L|+1)$ instructions.

Montrons maintenant que la fonction affiche le bon résultat. Pour cela il faut définir ce qu'est ce bon résultat.

On peut tout simplement dire qu'il s'agit d'appeler la fonction `print` sur les éléments de L dans le bon ordre ou ne rien faire si la liste est vide. *Dans le bon ordre* signifie que si l'élément est en i^{e} position dans la liste, alors il est le i^{e} élément affiché.

Prouvons le par récurrence sur $|L|$. Si $|L| = 0$ alors L est vide, donc la fonction s'arrête sans exécuter la ligne 3 ou la ligne 4 et n'affiche rien. Si $|L| = n+1$ et si la fonction affiche le bon résultat pour toute liste de taille n, alors

- la liste est non vide donc `print_list` affiche la tête de L à la ligne 3 puis tous les entiers de $\texttt{tail}(L)$ à la ligne 4.
- La tête de L est le premier entier affiché ; il s'agit aussi du premier entier de L, donc cet entier est correctement affiché.
- $\texttt{tail}(L)$ est de taille n donc d'après l'hypothèse de récurrence, `print_list` affiche les entiers de $\texttt{tail}(L)$ dans le bon ordre. Le i^{e} entier de $\texttt{tail}(L)$ est donc le i^{e} entier affiché par cet appel.

  Donc, puisque tous ces entiers sont affichés après $\texttt{head}(L)$, il est le $i+1^{\text{e}}$ entier affiché par $\texttt{print\_list}(L)$.

  Or le i^{e} entier de $\texttt{tail}(L)$ est le $i+1^{\text{e}}$ entier de L, donc le $i+1^{\text{e}}$ entier de L est le $i+1^{\text{e}}$ entier affiché.

Par récurrence, la fonction est donc correcte.

Cette preuve a l'air de dire des trivialités, mais il est important de bien le prouver. Il ne suffit pas de dire par exemple qu'on affiche la tête et la queue de L pour prouver que la fonction est correcte. La preuve est que si on inverse la ligne 3 et la ligne 4 alors le résultat n'est plus du tout le même ; et la preuve ci-dessus ne fonctionnerait plus.

2. Écrire une fonction récursive qui concatène 2 listes.

▶ Correction

1: **function** CONCAT($L1, L2$)
Entrées: Deux listes $L1$ et $L2$
Sorties: Une liste contenant tous les éléments de $L1$ dans l'ordre puis tous les éléments de $L2$ dans l'ordre.
2: **Si** $\texttt{empty}(L1)$ **Alors**
3: **Renvoyer** $L2$
4: **Sinon**
5: $T \leftarrow \texttt{cons}(\texttt{head}(L_1), \text{CONCAT}(\texttt{tail}(L_1), L_2)$

3. En déduire une fonction récursive qui prend en entrée une liste et inverse le contenu de cette liste.

▶ Correction

1: **function** REVERSE(L)
Entrées: Une liste L
Sorties: Une liste contenant tous les éléments de L dans l'ordre inverse.
2: **Si** $\texttt{empty}(L)$ **Alors**
3: **Renvoyer** L
4: **Sinon**
5: $T \leftarrow \text{REVERSE}(\texttt{tail}(L))$
6: **Renvoyer** $\texttt{concat}(T, \texttt{cons}(\texttt{head}(L), \emptyset))$

Ce n'est pas demandé, mais on peut, comme à la question 1 prouver que cet algorithme est correct et se termine ; en supposant que la réponse à la question 2 soit correcte et se termine également.

Montrons par récurrence sur $|L|$ que `reverse`(L) se termine et renvoie le bon résultat ; c'est à dire une liste où le i^e élément de L est placé en position $|L| - i + 1$ de la liste résultante (on suppose que la tête est le premier élément, donc l'élément 1 et que le dernier élément est l'élément $|L|$).

Si $|L| = 0$ alors L est vide et donc l'algorithme n'effectue que la ligne 2 et s'arrête. On renvoie alors L qui est égal à L inversé parce que L est vide.

Sinon, en supposant que la propriété soit vérifiée pour toute liste de taille n et en supposant que $|L| = n + 1$, l'algorithme effectue les lignes 4 et 5.

Par hypothèse de récurrence, la ligne 4 se termine et renvoie la liste `tail`(L) correctement inversée. La ligne 5 se termine (puisqu'on suppose que `concat` a été correctement codée) et renvoie la concaténation C de la liste `tail`(L) inversée et de `head`(L).

Dans la liste C, `head`(L) est en dernière position, il est donc correctement placé. Montrons que le $(i+1)^e$ élément de L est en position $|L| - (i+1) + 1$ dans C. Cet élément est en fait le i^e élément de `tail`(L). Par hypothèse de récurrence, T est la liste `tail`(L) inversée, donc l'entier est en $|T| - i + 1^e$ position de la liste T.

Puisque C contient d'abord les éléments de T puis la tête de L, cet entier est en $|T| - i + 1^e$ position dans la liste C. Puisque $|T| = |\texttt{tail}(L)| = n$, alors cet élément est en position $n - i + 1$ dans la liste C. Et puisque $n - i + 1 = (n+1) - (i+1) + 1 = |L| - (i+1) + 1$ alors l'élément est en position $|L| - (i+1) + 1$ dans la liste C.

Par récurrence, l'algorithme se termine et répond correctement.

4. Réécrire cette dernière fonction sous forme récursive terminale, la nouvelle fonction prendra en entrée 2 listes.

▶ **Correction**

1: **function** REVERSETERM(L, R)
ENTRÉES: Une liste L et une liste R
SORTIES: La concaténation de la liste L inversée et de R.
2: **Si** `empty`(L) **Alors**
3: **Renvoyer** R
4: **Sinon**
5: **Renvoyer** REVERSETERM(`tail`(L), `cons`(`head`(L), R))

Exercice 2 — *Récursivité terminale*

Réécrire les fonctions suivantes sous forme récursive terminale.

1. **ENTRÉES:** Deux entiers n et k
SORTIES: $\binom{n}{k}$
Si $k = n$ ou $k = 0$ **Alors**
Renvoyer 1.
Renvoyer $\binom{n-1}{k-1} * \frac{n}{k}$

▶ **Correction**

function BINOMTERM(n, k, d)
ENTRÉES: Deux entiers n, k et un rationnel d
SORTIES: $\binom{n}{k} * d$
Si $k = n$ ou $k = 0$ **Alors**
Renvoyer d
Renvoyer BINOMTERM($n-1, k-1, \frac{nd}{k}$)

On l'utilise avec BINOMTERM$(n, k, 1)$

2. Dans la fonction suivante, I est la matrice identité.
ENTRÉES: Une liste $L = [A_1, A_2, \ldots, A_n]$ de matrices,
SORTIES: $A_1 \times A_2 \times \cdots \times A_n$
function MULT(L)

Si $L = \emptyset$ **Alors**
Renvoyer I.
Renvoyer $A_1 \times \text{MULT}([A_2, \ldots, A_n])$

▶ **Correction**

function MULTTERM(n, k, d)
ENTRÉES: Une liste $L = [A_1, A_2, \ldots, A_n]$ de matrices, et une matrice M
SORTIES: $M \times A_1 \times A_2 \times \cdots \times A_n$
Si $L = \emptyset$ **Alors**
Renvoyer M.
Renvoyer MULTTERM($[A_2, \ldots, A_n], M \times A_1$)

On l'utilise avec MULTTERM(L, I)

3. Pour la suivante, vous aurez besoin d'exécuter 2 fonctions récursives terminales pour répondre.
ENTRÉES: Une liste $L = [b_1, b_2, \ldots, b_n]$ d'entiers, une fonction $f : \mathbb{N}^2 \to \mathbb{N}$, un entier a
SORTIES: $f(b_1, f(b_2, \ldots f(b_n, a) \ldots))$
function FOLD(L, f, a)
Si $L = \emptyset$ **Alors**
Renvoyer a.
Renvoyer $f(b_1, \text{FOLD}([b_2, \ldots, b_n], f, a))$

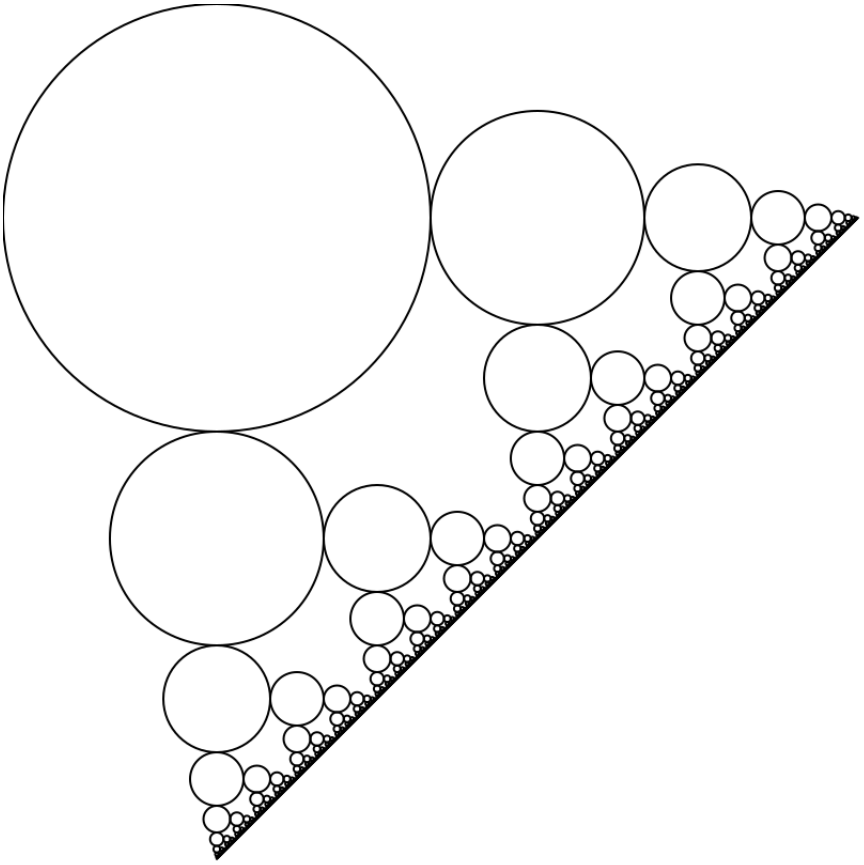
▶ **Correction**

On va créer une première fonction qui fait presque ce qui est demandé, à ceci près que la liste est inversée.
ENTRÉES: Une liste $L = [b_n, b_{n-1}, \ldots, b_1]$ d'entiers, une fonction $f : \mathbb{N}^2 \to \mathbb{N}$, un entier a
SORTIES: $f(b_1, f(b_2, \ldots f(b_n, a) \ldots))$
function FOLDLEFT(L, f, a)
Si $L = \emptyset$ **Alors**
Renvoyer a.
Renvoyer FOLDLEFT($[b_{n-1}, b_{n-2}, \ldots, b_1], f, f(b_n, a)$)

On l'applique avec l'algorithme suivant :
ENTRÉES: Une liste $L = [b_1, b_2, \ldots, b_n]$ d'entiers, une fonction $f : \mathbb{N}^2 \to \mathbb{N}$, un entier a
SORTIES: $f(b_1, f(b_2, \ldots f(b_n, a) \ldots))$
function FOLD(L, f, a)
Renvoyer FOLDLEFT(REVERSE(L), f, a)

Exercice 3 — *Cercles*

1. Ecrire une fonction récursive qui, connaissant une fonction `CERCLE`(p, r) affiche un cercle de centre p et de rayon r, la fractale suivante :



Pour chaque cercle, deux copies de ce cercle ayant subi une réduction d'un facteur 2 sont placés tangents extérieurement au cercle initial et tels que les lignes des centres sont parallèles aux axes du repère.

► **Correction**

Il existe plusieurs manière d'écrire la fonction. Il faut noter que la fonction suivante est fausse :

Entrées: Un point p, un rayon r
Sorties: la fractale, démarrant au point p et au rayon r

1: **function** Frac(p, r)
2: `Cercle`(p, r)
3: Frac$((p.x + 3r/2, p.y), r/2)$
4: Frac$((p.x, p.y + 3r/2), r/2)$

En effet, la fonction ne se termine pas (ce qui est normal, puisque la fractale est infinie). Donc la ligne 3 ne se termine pas, donc on effectue jamais la ligne 4. Si on exécutait cet algorithme, on verrait uniquement les cercles de la première ligne.

Pour corriger le tir, il existe plusieurs méthodes :

La première consiste à n'afficher que les cercles de profondeur n :

Entrées: Un point p, un rayon r, un entier n
Sorties: les cercles de profondeur n de la fractale, démarrant au point p et au rayon r

1: **function** Frac1(p, r)
2: **Si** $n = 0$ **Alors**

```
3:        Cercle(p, r)
4:    Sinon
5:        Frac1((p.x + 3r/2, p.y), r/2, n − 1)
6:        Frac1((p.x, p.y + 3r/2), r/2, n − 1)
```

Puis à exécuter l'algorithme suivant :

Entrées: Un point p, un rayon r
Sorties: la fractale, démarrant au point p et au rayon r $n = 0$

```
1: Tant que n > −1 Faire
2:     Frac1(p, r, n)
3:     n ← n + 1
```

Une version purement récursive consisterait à appliquer la fonction suivante avec $L = (p, r)$.

Entrées: une liste L de couples point, rayon
Sorties: un ensemble de fractales, démarrant en chacun des points décrits par la liste L

```
1: function Frac2(L)
2:     L' ← ∅
3:     Pour (p, r) ∈ L Faire
4:         Cercle(p, r)
5:         Ajouter (p.x + 3r/2, p.y), r/2) à L'
6:         Ajouter (p.x, p.y + 3r/2), r/2) à L'
7:     Frac2(L')
```

La première version a le défaut, à chaque itération, de repasser par tous les cercles déjà dessinés pour dessiner les nouveaux cercles.

La seconde version a le défaut de maintenir la liste L à chaque appel récursif et cette liste double de taille à chaque itération.

2. Montrer qu'aucun cercle n'intersecte un autre.

▶ **Correction**

Soit un cercle c de centre $p = (x, y)$ et de rayon r ; soit c_1 et c_2 les deux cercles en dessous et à droite de c. On va montrer que tous les cercles issus de c_1 vers sa droite ne rencontrent pas les cercles issus de c_2 vers le bas.

Pour tout autre cercle :

- On prouve par récurrence que tous les cercles issus de c_1 ne se touchent pas
- On prouve par récurrence que tous les cercles issus de c_2 ne se touchent pas
- Les cercles issus de c_1 et en dessous de c_1 ne risquent pas de rencontrer ceux issus de c_2 qui sont à droite de c_2.

c_1 est de centre $(x, y + 3r/2)$ et de rayon $r/2$. Le i^{e} cercle à droite de c_1 est donc de rayon $r/2^{i+1}$ en position

$$\begin{aligned}&(x + 3r/4 + 3r/8 + 3r/16 + \cdots + 3r/2^{i+1}, y + 3r/2)\\ &= (x + 3r/2 \cdot (1/2 + 1/4 + 1/8 + ... + 1/2^i), y + 3r/2)\\ &= (x + 3r/2 \cdot (1 - 1/2^i), y + 3r/2)\end{aligned}$$

c_2 est de centre $((x+3r/2, y))$. Le j^{e} cercle en dessous de c_2 est de rayon $r/2^{j+1}$ et en position

$$\begin{aligned}&(x + 3r/2, y + 3r/4 + 3r/8 + 3r/16 + \cdots + 3r/2^{j+1})\\ &= (x + 3r/2, y + 3r/2 \cdot (1/2 + 1/4 + 1/8 + ... + 1/2^j))\\ &= (x + 3r/2, y + 3r/2 \cdot (1 - 1/2^j))\end{aligned}$$

c_i et c_j s'intersecte si la distance entre leurs centres est plus grande que la somme de leurs rayons. La distance est :

(c'est un gros calcul parceque c'est un calcul très détaillé)

$$\begin{aligned}
&\sqrt{\left(x+\frac{3r}{2}\cdot(1-\left(\frac{1}{2}\right)^i)-x-\frac{3r}{2}\right)^2+\left(y+\frac{3r}{2}\cdot(1-\left(\frac{1}{2}\right)^j)-y-\frac{3r}{2}\right)^2}\\
&=\sqrt{\left(\frac{3r}{2}\cdot(1-\left(\frac{1}{2}\right)^i)-\frac{3r}{2}\right)^2+\left(\frac{3r}{2}\cdot(1-\left(\frac{1}{2}\right)^j)-\frac{3r}{2}\right)^2}\\
&=\sqrt{\left(\frac{3r}{2}-\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^i-\frac{3r}{2}\right)^2+\left(\frac{3r}{2}-\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^j-\frac{3r}{2}\right)^2}\\
&=\sqrt{\left(-\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^i\right)^2+\left(-\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^j\right)^2}\\
&=\sqrt{\left(\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^i\right)^2+\left(\frac{3r}{2}\cdot\left(\frac{1}{2}\right)^j\right)^2}\\
&=\sqrt{\left(\frac{9r^2}{4}\cdot\left(\frac{1}{4}\right)^i\right)+\left(\frac{9r^2}{4}\cdot\left(\frac{1}{4}\right)^j\right)}\\
&=\sqrt{\frac{9r^2}{4}\left(\left(\frac{1}{4}\right)^i+\left(\frac{1}{4}\right)^j\right)}\\
&=\frac{3r}{2}\sqrt{\left(\frac{1}{4}\right)^i+\left(\frac{1}{4}\right)^j}
\end{aligned}$$

On veut donc montrer

$$\frac{r}{2^{i+1}}+\frac{r}{2^{j+1}}<\frac{3r}{2}\sqrt{\frac{1}{4^i}+\frac{1}{4^j}}$$

Or, par inégalité de Cauchy Swartz (ouch)

$$\frac{1}{2^{i+1}}+\frac{1}{2^{j+1}}\leq 2\sqrt{\frac{1}{4^{i+1}}+\frac{1}{4^{j+1}}}$$

Donc

$$\begin{aligned}
&\frac{1}{2^{i+1}}+\frac{1}{2^{j+1}}\leq\sqrt{\frac{1}{4^i}+\frac{1}{4^j}}\\
&\frac{1}{2^{i+1}}+\frac{1}{2^{j+1}}<\frac{3}{2}\sqrt{\frac{1}{4^i}+\frac{1}{4^j}}
\end{aligned}$$

Donc les deux cercles ne se touchent pas.