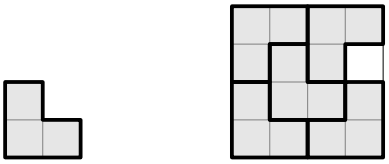
Outils logiques et algorithmiques – TD 7 – Listes et raisonnement par récurrence

Exercise 1 (Récurrence simple.) Rédigez une récurrence classique pour montrer la propriété suivante pour tout $n \in \mathbb{N}$:

$$1 + 4 + 9 + \ldots + n^2 = \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

□

Exercise 2 (Récurrence sur les entiers) Un triomino est une pièce formée de trois cases disposées en angle (figure (a) ci-dessous). On considère la propriété suivante : si on retire une case quelconque à un quadrillage de côté 2^n, (avec $n \geq 1$), il est toujours possible de paver le reste du quadrillage avec des triominos. La figure (b) donne un exemple de cette propriété pour un carré de côté $4 = 2^2$. Montrer par récurrence sur n que la propriété est vraie pour tout $n \geq 1$, et tout choix de case à retirer.



(a) (b)

□

Exercise 3 (Récurrence à récurer) Toto veut montrer par récurrence que dans toutes les boîtes de crayons, tous les crayons sont de la même couleur. Il fait la preuve suivante

- C'est vrai de manière évidente pour toutes les boîtes avec un seul crayon.
- On va montrer que si la propriété est vraie dans une boite qui contient n crayons alors elle est encore vraie dans une boîte qui en contient $n+1$. On prend donc une boîte qui contient $n+1$ crayons. On retire le premier crayon de la boîte, par hypothèse de récurrence, les crayons restants sont tous de la même couleur. On retire maintenant le dernier crayon de la boîte, par hypothèse de récurrence, les crayons restants sont tous de la même couleur (la couleur des crayons du milieu). Tous les crayons de la boîte ont donc la même couleur.
- Par récurrence on peut donc conclure que tous les crayons de la boîte ont la même couleur.

Toto s'est évidemment trompé quelque part dans son raisonnement car le résultat est faux, pouvez-vous dire à quel endroit ? □

Exercise 4 (Comptages) Voici deux définitions pour des fonctions caml comptant le nombre d'occurrences d'un élément `e` dans une liste `l`.

```
let rec count_1 e l = match l with
  | []      -> 0
  | x :: tl -> if x = e
               then 1 + count_1 e tl
               else     count_1 e tl
```

```
let rec count_2 e l n = match l with
  | []      -> n
  | x :: tl -> if x = e
               then count_2 e tl (n+1)
               else count_2 e tl  n
```

On peut les associer aux équations suivantes.

$$\begin{cases} \texttt{count}_1(e, []) &= 0 \\ \texttt{count}_1(e, e :: \ell) &= 1 + \texttt{count}_1(e, \ell) \\ \texttt{count}_1(e, e' :: \ell) &= \texttt{count}_1(e, \ell) \qquad e' \neq e \end{cases}$$

$$\begin{cases} \texttt{count}_2(e, [], n) &= n \\ \texttt{count}_2(e, e :: \ell, n) &= \texttt{count}_2(e, \ell, n+1) \\ \texttt{count}_2(e, e' :: \ell, n) &= \texttt{count}_2(e, \ell, n) \qquad e' \neq e \end{cases}$$

Montrer par récurrence sur ℓ que pour tous e et n, on a $\texttt{count}_2(e, \ell, n) = n + \texttt{count}_1(e, \ell)$. □

Exercise 5 (Renversement de liste) Voici une définition pour une fonction caml `rev: 'a list -> 'a list` de renversement de liste.

```
let rec rev l = match l with
  | []     -> []
  | x :: l -> concat (rev l) [x]
```

On peut l'associer aux équations suivantes.

$$\begin{cases} \mathsf{rev}([]) &= [] \\ \mathsf{rev}(x :: \ell) &= \mathsf{concat}(\mathsf{rev}(\ell),\ x :: []) \end{cases}$$

On réutilise les fonctions `concat` et `longueur` définies dans le cours, et vous pouvez réutiliser leurs propriétés.

1. Démontrer que `rev` préserve la longueur :

   $$\forall \ell, \quad \mathsf{longueur}(\mathsf{rev}(\ell)) = \mathsf{longueur}(\ell)$$

2. Démontrer que `rev` se distribue sur `concat` de la manière suivante :

   $$\forall \ell_1, \ell_2, \quad \mathsf{rev}(\mathsf{concat}(\ell_1, \ell_2)) = \mathsf{concat}(\mathsf{rev}(\ell_2),\ \mathsf{rev}(\ell_1))$$

3. Démontrer que `rev` est involutive, c'est-à-dire que

   $$\forall \ell, \quad \mathsf{rev}(\mathsf{rev}(\ell)) = \ell$$

□

Exercise 6 (Correction du renversement récursif terminal) La définition précédente de `rev` est adaptée pour spécifier le renversement d'une liste et démontrer ses propriétés, mais elle est terriblement inefficace. En voici une plus réaliste.

```
let rec rev_rt l = rev_append l []

and rev_append l1 l2 = match l1 with
  | []     -> l2
  | x :: l -> rev_append l (x :: l2)
```

1. Au fait, qu'est-ce qui était inefficace dans le code de l'exercice précédent ?
2. Donner des équations récursives spécifiant la fonction de retournement `rev_rt` et sa fonction auxiliaire `rev_append`.
3. Démontrer que `rev_rt` est correcte, c'est-à-dire que

   $$\forall \ell, \quad \mathsf{rev\_rt}(\ell) = \mathsf{rev}(\ell)$$

   Indication : il faudra démontrer quelque chose par récurrence, mais quoi exactement ?

□