Institut National
Universitaire
Champollion

TD n° 2 - Récurrence et induction structurelle

Types de données, preuves
L3 INFO - Semestre 6 - 2018

Exercice 1 - Récurrence sur les entiers naturels

1. Montrer que : $1+2+...+n = \frac{n(n+1)}{2}$
2. Considérons la suite $(u_n)_{n\geqslant 0}$ définie par $\begin{cases} u_0 = 0 \\ \forall n \in \mathbb{N}, u_{n+1} = \frac{1}{3}u_n + 2 \end{cases}$
   Montrer que cette suite est majorée par 3, c'est à dire $\forall n \in \mathbb{N}, u_n \leqslant 3$.

Exercice 2 - Arithmétique de Peano

Rappelons que Peano a redéfini les entiers naturels par le type inductif suivant :

type ent_nat =
Zero
| Succ of ent_nat ;;

Ainsi, 2 s'écrit *Succ(Succ(Zero))*.

Avec cette représentation, Peano a redéfini l'addition par les deux axiomes suivants (x et y sont du type *ent_nat*) :

(Axiome 1) x + *Zero* = *Zero* + x = x
(Axiome 2) x + *Succ(y)* = *Succ(x + y)*

(Il est fortement conseillé de faire des exemples pour bien comprendre !)

1. Montrer que l'addition de l'arithmétique de Peano est associative (c'est à dire $(x+y)+z = x+(y+z)$ - on pourra raisonner par récurrence sur z).
2. Montrer que l'addition est également commutative (c'est à dire $x+y = y+x$ - on pourra raisonner par récurrence sur y)

Exercice 3 - Induction sur les listes

1. Écrire les fonctions suivantes :
   - *sum* qui renvoie la somme de tous les éléments d'une liste d'entiers
   - *concat* qui prend une liste de listes et concatène tous ses éléments
     Exemple :
     concat [[1 ;2] ; [] ; [3 ;4 ;5]] ;;
     - : int list = [1 ; 2 ; 3 ; 4 ; 5]
2. Montrer par induction structurelle les assertions suivantes suivantes :
   (a) *long (l1 @ l2) = long l1 + long l2*
   (la fonction *long* est la fonction vue en cours qui calcule la longueur d'une liste)
   (b) *sum (l1 @ l2) = sum l1 + sum l2*

(c) *long (concat l1) = sum (listmap long l1)*

(d) *concat (l1 @ l2) = (concat l1) @ (concat l2)*

Activité - Induction sur les listes

1. Écrire une fonction *listmap* qui applique une fonction à tous les éléments de la liste.
   Exemple :

   listmap (fun x -> x + 2) [1 ; 2 ; 3] ;;
   - : int list = [3 ; 4 ; 5]

   listmap (fun x -> [x]) [1 ; 2 ; 3] ;;
   - : int list list = [[1];[2];[3]]

   Deux applications consécutives de *Listmap* peuvent être réalisées en une seule comme dans l'exemple suivant :
   Exemple :
   *listmap (fun x -> x + 4) (listmap (fun x -> 2 * x) [1 ; 2 ; 3]) ;;*
   - : int list = [6 ; 8 ; 10]
   *(listmap (fun x -> 2 * x + 4) [1 ; 2 ; 3]) ;;*
   - : int list = [6 ; 8 ; 10]
2. Écrire une fonction *compose* qui compose deux fonctions.
   Exemple :

   *comp (fun x -> x + 4) (fun x -> 2 * x) 3 ;;*
   - : int = 10

   *comp (fun x -> [x]) (fun x -> 2 * x) 3 ;;*
   - : int list = [6]
3. (a) Que donne *listmap (fun x -> x + 4) (listmap (fun x -> 2 * x) [1 ; 2 ; 3]) ;;* ?
   (b) Que donne *listmap (comp (fun x -> x + 4) (fun x -> 2 * x)) [1 ; 2 ; 3] ;;*
   (c) Que remarquez-vous ?
4. Démontrer par induction structurelle l'assertion suivante :

   pour toutes fonctions *f,g* et pour toute liste *list* ,
   listmap f (listmap g list) = listmap (comp f g) list
5. Comparer et commenter.

Exercice 4 - Induction sur les arbres

On considère toujours le type

```
type 'a arbre_bin =
Feuille of 'a
| Noeud of 'a * 'a arbre_bin * 'a arbre_bin ;;
```

1. Définir une fonction swap qui échange partout les sous-arbres gauches et droits.

```
swap (Node(1, Leaf 2, Node(3, Leaf 4, Leaf 5))) ;;
-  : Node (1, Node (3, Leaf 5, Leaf 4), Leaf 2)
```

2. Montrer par induction :

$$\forall t : \texttt{arbre\_bin} \ , \ \texttt{swap(swap(}t\texttt{))} = t$$