TD20 : Ordres et induction

Exercice 1 Soit trois symboles a, f et g. On considère un ensemble E d'expressions (formelles) défini inductivement par :
assertions $\{a\}$
règles d'inférence $\{(e_1, e_2) \mapsto f(e_1, e_2), (e_1, e_2, e_3) \mapsto g(e_1, e_2, e_3)\}$
Par exemple E contient l'expression $f(a, g(a, a, f(a, a)))$.
On note $|e|_a$, $|e|_f$ et $|e|_g$ le nombre respectif de symboles a, f et g dans l'expression e.

Question 1 : Montrer par induction structurelle que :

$$\forall e \in E, |e|_a = 2|e|_g + |e|_f + 1\,.$$

Exercice 2 On définit inductivement l'ensemble E par :
assertions $\{(0,0)\}$
règles d'inférence $\{(m, n) \mapsto (m, n+1), (m, n) \mapsto (m+1, n+1), (m, n) \mapsto (m+2, n+1)\}$.

Question 2 : Montrer que pour tout $(m, n) \in E$, on a : $m \leq 2n$.

Exercice 3 On définit inductivement $\mathbb{N}^2$ par :
assertions $\{(0,0)\}$
règles d'inférence $\{(m, n) \mapsto (m+1, n), (m, n) \mapsto (m, n+1)\}$
Soit l'opération de soustraction suivante :

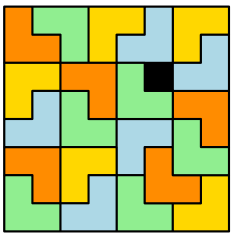
$$\begin{array}{rcl} \mathrm{sub}(m, 0) & = & m \\ \mathrm{sub}(0, n) & = & 0 \\ \mathrm{sub}(m, n) & = & \mathrm{sub}(m-1, n-1) \end{array}$$

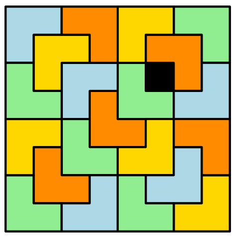
Question 3 : Montrer que cette opération est bien définie pour tout élément de $\mathbb{N}^2$.
Question 4 : Montrer que pour tout $(m, n) \in \mathbb{N}^2$, on a : $\mathrm{sub}(m, n) \leq m$.

Exercice 4 On considère un robot qui se promène sur une grille bi-dimensionnelle infinie à coordonnées entières. Son point de départ est le point de coordonnées $(0,0)$ et il ne peut faire à chaque étape qu'un pas en diagonale. Donc après une étape, il sera sur un des 4 points suivants : $(1,1)$, $(1,-1)$, $(-1,1)$, $(-1,-1)$.

Question 5 : Le robot peut-il atteindre la case $(0,1)$? (à prouver avec une induction structurelle)

Exercice 5
On considère une grille carrée de côté n dans laquelle une case (i, j) est interdite ($1 \leq i, j \leq n$). On suppose que n est une puissance de 2 : $n = 2^k$.
On veut paver la grille (excepté la case interdite) avec des tuiles de trois cases en forme de L, orientées dans n'importe quel sens, comme sur les exemples ci-contre (pour lesquels $n = 8$).





Question 6 : Donner une définition inductive de l'ensemble des carrés de côté 2^n pour $n \in \mathbb{N}$.

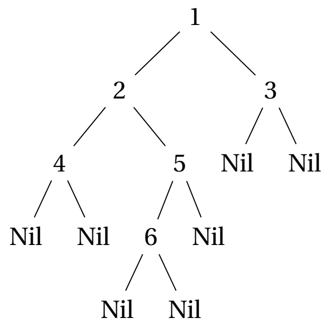
Question 7 : On considère un carré de côté 2^n. Montrer par induction structurelle qu'on peut toujours le paver avec des tuiles de trois cases en forme de L en évitant la case interdite.

Exercice 6 On considère le code suivant :

```
type bitree = Nil | Node of bitree * bitree

let rec z a b = match a, b with
  | Nil , t -> t
  | Node (t1 , t2), t3 -> z t1 (Node (t3 , t2))
```

Question 8 : Quelle est la valeur de `(z a Nil)` pour `a` l'arbre suivant (les numéros sont là uniquement pour identifier les nœuds, ils ne font pas partie du `bitree`) :



Question 9 : Que vaut `(z (z a Nil)Nil)` où `a` est toujours l'arbre précédent ?

Question 10 : De façon générale, montrer l'équivalence suivante si `b` et `c` sont des `bitree` quelconques :

`z (z b c)Nil` $=$ `z c b`

On pourra raisonner par induction sur `b1` pour des bitree `b` de la forme `Node (b1, b2)`.

Question 11 : En déduire que l'expression suivante est équivalente à `b` :

`z (z b Nil)Nil`

Exercice 7

On donne le code suivant :

```
type integer = Z | S of integer

let rec plus a b =
  match a with
  | Z -> b
  | S k -> plus k (S b)

let rec fois a b =
  match a with
  | Z -> Z
  | S k -> plus b (fois k b)
```

Question 12 : Montrer par induction structurelle que `fois n Z = Z`.

Question 13 : Montrer que pour tout `n` de type `integer` et tout entier k, on a `plus n (`$\texttt{S}^k$` Z) = `$\texttt{S}^k$` (plus n Z)`, où on a noté $\texttt{S}^k$ pour `S` répété k fois, par induction structurelle sur le type `integer`.

Question 14 : En déduire que `plus n Z = n`.

Exercice 8

On considère le code suivant :

```
let rec reverse = function
  | [] -> []
  | x :: xs -> reverse xs @ [x]
```

On veut montrer que : `reverse (reverse lst)= lst`. On décompose cette preuve en plusieurs étapes.
Récupérons le code la fonction `rev` du module `List` :

```
let rec rev_append l1 l2 =
  match l1 with
    [] -> l2
  | a :: l -> rev_append l (a :: l2)

let rev l = rev_append l []
```

Question 15 : Montrer que `rev_append lst1 lst2 = (rev lst1)@ lst2` par induction structurelle sur `lst1`.

Question 16 : Montrer que `reverse lst = rev lst` par induction structurelle.

Question 17 : Montrer que `rev ( lst1 @ lst2 )= (rev lst2)@ (rev lst1)` par induction structurelle sur `lst1`.

Question 18 : Montrer par induction structurelle que : `rev (rev lst)= lst`.