TD5 – Arbres binaires et chemins

Exercice 5.1 (Extrait Examen Juin 2023 - suite exercice 2.3)**.**

1. Définir une fonction de signature `pref_complete (l : ('a list) list) : bool` qui détermine si tous les préfixes de chaque élément de `l` sont aussi des éléments de `l`.

```
# (pref_complete [[];[0];[0;1];[1];[1;0];[1;0;0];[1;1];[1;1;0];[1;1;1]]);;
- : bool = true
# (pref_complete [[];[0];[0;1];[1];[1;0;0];[1;1];[1;1;0];[1;1;1]]);;
- : bool = false
```

2. Définir une fonction de signature :

   `is_max_pref2 (l1 : 'b list) (l2 : ('a * ('b list)) list) : bool`

   qui détermine si tous les éléments `(x,c)` de la liste `l2` tels que `l1` est un préfixe de `c` vérifient `c=l1`. En d'autres termes, il s'agit de vérifier que si `(x,c)` est un élément de `l2` et si `l1` est un préfixe de `c` alors `c=l1`. Dans ce cas on dit que `l1` est un préfixe maximal pour `l2`.

   Exemples.

```
# (is_max_pref2 [0;1]
                [('a',[]);('b',[0]);('c',[0;1]);('d',[1]);('e',[1; 0]);
                 ('f',[1;0;0]);('g',[1;1]);('h',[1;1;0]);('i',[1;1;1])]);;
    - : bool = true
```

   Le seul couple `(x,c)` tel que `[0;1]` est préfixe de `c` est le couple `('c',[0;1])` et `[0;1]=[0;1]`.

```
# (is_max_pref2 [1;0]
                [('a',[]);('b',[0]);('c',[0;1]);('d',[1]);('e',[1; 0]);
                 ('f',[1;0;0]);('g',[1;1]);('h',[1;1;0]);('i',[1;1;1])]);;
    - : bool = false
```

   La liste de couples contient `('f',[1;0;0])` et `[1;0]` est un préfixe de `[1;0;0]` mais `[1;0;0]`$\neq$`[1;0]`.

3. Définir une fonction de signature :

   `max_pref_list2 (l : ('a * ('b list)) list) : ('a * ('b list)) list`

   qui permet d'obtenir la liste des éléments `(x,c)` de `l` tel que la liste `l` ne contient aucun couple `(x',c')` tel que `c` est un préfixe de `c'` différent de `c'`. En d'autres termes, il s'agit de ne conserver dans la liste `l` que les éléments `(x,c)` qui sont un préfixe maximal pour `l`.

   Exemple.

```
# (max_pref_list2
    [('a',[]);('b',[0]);('c',[0;1]);('d',[1]);('e',[1; 0]);
     ('f',[1;0;0]);('g',[1;1]);('h',[1;1;0]);('i',[1;1;1])]);;
    - : (char * int list) list =
[('c', [0; 1]); ('f', [1; 0; 0]); ('h', [1; 1; 0]); ('i', [1; 1; 1])]
```

   Le couple `('c',[0;1])` est dans la liste construite car le seul couple `(x,c)` de la liste `l` tel que `[0;1]` est préfixe de `c` est le couple `('c',[0;1])` et `[0;1]=[0;1]`. Le couple `('f',[1;0;0])` n'est pas dans la liste construite car le couple `('e',[1; 0])` est dans la liste `l`, la liste `[1;0]` est un préfixe de `[1;0;0]` mais `[1;0;0]`$\neq$`[1;0]`.

Exercice 5.2 (Extrait Examen Juin 2023 - Arbres et chemins)**.**
Dans cet exercice, on pourra utiliser les fonctions définies dans l'exercice précédent. On considère le type des arbres binaires définis par :

```
type 'a btree = Empty | Node of 'a * ('a btree) * ('a btree)
```

Un chemin dans un arbre binaire d'un nœud N_1 à un nœud N_2 est une liste c contenant uniquement des 0 et des 1 telle que le nœud N_2 est accessible à partir du nœud N_1 en se déplaçant à chaque étape du chemin dans le sous-arbre gauche lorsque l'entier est 0 et dans le sous-arbre droit lorsque l'entier est 1. Les exemples sont donnés pour l'arbre `tree1` ci-dessous.

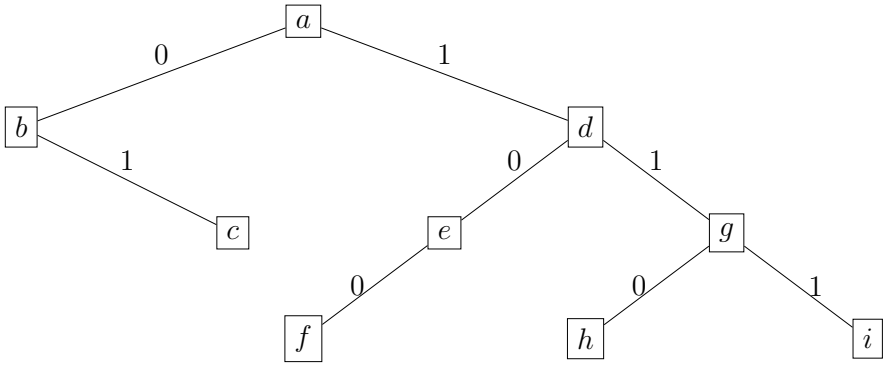

FIGURE 2 – Exemple : arbre binaire `tree1`

1. (Etiquette désignée par un chemin)
   Définir une fonction de signature `at_path (t:'a btree) (c:int list) : 'a` qui permet d'obtenir l'étiquette du nœud accessible à partir de la racine de l'arbre `t` en suivant le chemin `c`. La fonction `at_path` lèvera l'exception `Invalid_argument` lorsque le chemin `c` n'est pas un chemin de l'arbre `t` ou contient un entier différent de 0 ou 1.

```
# at_path tree1 [];;
- : char = 'a'
# at_path tree1 [1;0];;
- : char = 'e'
# at_path tree1 [0;0];;
Exception: Invalid_argument "at_path".
# at_path tree1 [2;0];;
Exception: Invalid_argument "at_path".
```

2. (Représentation d'un arbre par une liste des chemins)
   Définir une fonction de signature `paths_tree (t : 'a btree) : ('a * (int list)) list` qui construit la liste de tous les couples `(e,c)` tels que `c` est un chemin de l'arbre binaire `t` et `e` est l'étiquette désignée par le chemin `c`.

```
# paths_tree tree1;;
- : (char * int list) list =
[('a', []); ('b', [0]); ('c', [0; 1]); ('d', [1]); ('e', [1; 0]);
 ('f', [1; 0; 0]); ('g', [1; 1]); ('h', [1; 1; 0]); ('i', [1; 1; 1])]
```

3. Une liste `c` ne contenant que des 0 et des 1 est un chemin d'un arbre binaire `t` si chaque préfixe de `c` est aussi un chemin de `t`. Définir une fonction de signature `is_tree (l : (int list) list) : bool` qui détermine s'il existe un arbre binaire tel que `l` contient exactement tous ses chemins. On suppose que les éléments de `l` sont des listes ne contenant que des 0 et des 1.

```
# is_tree (List.map snd (paths_tree tree1));;
- : bool = true
```

```
# is_tree [[]; [1]; [1; 0]; [1; 0; 1; 1]];;
- : bool = false
```

4. (Feuilles d'un arbre binaire)
   Les chemins d'un arbre binaire `t` qui ne sont préfixes d'aucun autre chemin de `t` désignent des feuilles de `t`. Définir une fonction de signature `leaves_list (l : ('a * (int list)) list) : 'a list` qui construit la liste des étiquettes des feuilles d'un arbre binaire représenté par une liste de couples `(e,c)` pour chaque chemin `c` de `t` désignant l'étiquette `e`.

   ```
   # leaves_list (paths_tree tree1);;
   - : char list = ['c'; 'f'; 'h'; 'i']
   ```