Sujet 1 (09/10/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Question de cours

Rappeler la définition d'un ensemble inductif et le principe d'induction.

Exercice 1 : Types en OCaml

Donner le type (s'il existe) de `f`, `g`, `h` (pour `h`, détailler les types des fonctions auxiliaires `hp` et `hpp`) :

```
let f x y l = (x+.y)::l

let g x = x x

let h x y =
    let hp x = (x+.y)>6. in
    let hpp y = x::y in
    hpp [hp 5.]
```

Exercice 2 : Arbre Binaire de Recherche

On définit un arbre binaire de recherche comme un arbre binaire tel que pour chaque nœud $N(x, \text{sous arbre gauche}, \text{sous arbre droit})$:

- x est plus petit que tous les éléments du sous arbre gauche.
- x est plus grand que tous les éléments du sous arbre droit.

1. Donner le type d'un arbre binaire `'a arbre_binaire`.

2. Écrire la fonction `est_abr : 'a arbre_binaire -> bool` qui renvoie vrai si l'arbre passé en argument est un arbre binaire e recherche.

3. Écrire la fonction `est_dans_arbre : 'a -> 'a arbre_binaire -> bool` qui renvoie vrai si un élément est présent dans un arbre binaire.

4. Écrire la fonction `est_dans_abr : 'a -> 'a arbre_binaire -> bool` qui renvoie vrai si un élément est présent dans un arbre binaire de recherche.

5. Est-ce qu'il est plus efficace de chercher dans un arbre binaire ou dans un arbre binaire de recherche ? Donner un exemple.

6. Donner une borne du nombre d'appels récursif de `est_dans_abr`.

Sujet 2 (09/10/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Question de cours

Rappeler la définition des arbres binaires. Définir les types OCaml pour les arbres généraux et les arbres binaires.

Exercice 1 : Comptage dans une liste

1. Écrire une fonction `assoc : 'a -> ('a * 'b) list -> 'b option` qui renvoie l'élément associé à l'élément de type `'a` de la liste de couples s'il existe.

2. Écrire une fonction `set : 'a -> 'b -> ('a * 'b) list -> ('a * 'b) list` qui modifie l'élément `'b` associé à l'élément `'a` de la liste de couples. Cette fonction doit insérer le couple s'il n'existe pas.

3. Écrire une fonction de type `int list -> (int * int) list` qui pour une liste de nombres renvoie une liste de couples (entier, nombre d'occurrences).

Exercice 2 : Arbre d'intervalles

On note un intervalle (a, b) comme un couple d'entiers (`type intervalle = int * int`) avec $a \leq b$. On souhaite réaliser un système de réservations de salle. Pour cela, on utilise un arbre d'intervalles:

- Une feuille représente une réservation sur un intervalle.
- Un noeud $N((a, b), g, d)$ représente un ensemble de réservation sur un l'intervalle (a, b), et tel que si le sous arbre gauche représente les réservations sur l'intervalle (u, v) et le sous arbre droit sur l'intervalle (x, y) alors on a $a \leq u \leq v < x \leq y \leq b$.

On a donc un type:

```
type tree =
| F of intervalle
| N of intervalle * tree * tree
```

On ne considérera que des arbres ayant au moins une réservation.

1. Écrire une fonction `disponible : intervalle -> tree -> bool` qui vérifie un intervalle est disponible dans l'arbre de réservation.

2. Écrire une fonction `fusionne : tree option -> tree option -> tree option` qui fusionne deux arbres d'intervalles dont les intervalles sont disjoints

3. Écrire une fonction `scinde : intervalle -> tree -> tree option * tree option * tree option` qui à partir d'un intervalle (a, b) et d'un arbre d'intervalle, scinde l'arbre en trois arbres x, y, z tel que

- x ne contient que des intervalles inférieurs à a
- y ne contient que des intervalles entre a et b
- z ne contient que des intervalles supérieurs à b

4. Écrire une fonction `inserer : intervalle -> tree -> tree` qui insère un intervalle disponible dans un arbre d'intervalle.

Sujet 3 (09/10/2024)

contact : jean-baptiste.doderlein@ens-rennes.fr

Exercice de cours

1. Rappeler la définition d'un arbre binaire strict.

2. Montrer par récurrence que le nombre de feuilles d'un arbre binaire strict est égal à son nombre de nœuds interne + 1.

Exercice 1 : Stream

On introduit un nouveau type, les stream. On les définit d'une manière similaire aux listes chainées d'OCaml avec le type :

```
type 'a stream =
| Nil
| Cons of 'a * (unit -> 'a stream)
```

1. Rappeler le type `list` de OCaml. Quelles différences avec les stream ?

2. Écrire la fonction `singleton : 'a -> 'a stream` qui renvoie un stream qui contient un unique élément.

3. Écrire la fonction `cons : 'a -> 'a stream -> 'a stream` qui ajoute un élément en tête du stream.

4. Écrire la fonction `uncons : 'a stream -> ('a * 'a stream) option` qui renvoie l'élément en tête et le reste de la liste.

5. Écrire la fonction `stream_to_list : 'a stream -> 'a list` qui convertit un stream en liste OCaml.

6. Écrire la fonction `list_to_stream : 'a list -> 'a stream` qui convertit une liste OCaml en un stream.

7. Soit la fonction `entiers` définie ainsi :

```
let rec entiers n =
    let suite () = entiers (n+1) in
    Cons(n, suite)
```

Que fait cette fonction ? Quel serait le résultat de `uncons (entiers 10)` ? et `stream_to_list (entiers 1)` ?

Exercice 2 : Arbre presque complet

1. Définir un type OCaml `tree` qui représente les arbres binaires avec des valeurs entières sur les nœuds.

2. Écrire une fonction `entiers : int -> tree` qui à partir d'un entier $n \geq 1$ renvoie un arbre binaire contenant n nœuds et qui contient les entiers de 1 à n.

3. Quelle est la hauteur des arbres générés par votre fonction `tree` ? Si ce n'est pas déjà le cas, proposer une fonction `entiers_pc : int -> tree` tel que l'arbre généré soit presque complet.

Exercices supplémentaires

Exercice 1 : Bijection entre arbres

1. Proposer un type OCaml `'a arbre_generaux` et un type `'a arbre_binaire` pour représenter des arbres généraux et généraux où les valeurs sont stockées sur les nœuds (i.e. pas de valeur aux feuilles).

2. Proposer une transformation entre arbres généraux et arbres binaires et inversement. Implémenter:

- `general_to_binaire : 'a arbre_generaux -> 'a arbre_binaire`
- `binaire_to_general : 'a arbre_binaire -> 'a arbre_binaire`

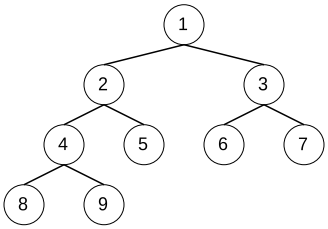
Si `arbre` est un arbre général, on doit avoir `arbre = binaire_to_general (general_to_binaire arbre`.

Exercice 2 : Sérialisation d'arbre strictement binaire

On cherche à retrouver la forme d'un arbre strictement binaire à partir de ses parcours en profondeur et en largeur. On considère dans cet exercice le type

```
type tree =
  | N of tree * int * tree
  | L of int
```

1. Donner les parcours en profondeur et en largeur de



2. Proposer une fonction `index : int list -> int -> int` qui retourne l'index d'un entier dans une liste. On suppose que la liste ne contient pas de doublon.

3. Écrire une fonction `split : int -> int list -> int * (int list) * (int list)` qui à partir d'un index et d'une liste

3. Proposer une fonction `tree_of_list : int list -> int list -> tree` qui reconstruit un arbre à partir de ses deux parcours