

Examen du 12 juin 2007

Les notes de cours et de TD manuscrites ainsi que les supports de cours distribués cette année sont les seuls documents autorisés.

Veillez lire attentivement les questions. Veillez rédiger proprement, clairement et de manière concise et rigoureuse. Le barème est indicatif.

Important. Les types et l'utilisation des fonctions de la bibliothèque standard d'Ocaml mentionnées (ou pouvant être utiles) dans les questions suivantes sont donnés à la fin de ce document.

1 Typage (7 points)

Question 1.1 Les fonctions f suivantes sont-elles bien typées ? Dans l'affirmative, donner leur type, sinon préciser pourquoi.

```
let f g x y = (g x , g y)

let rec f x y = f y ([]::x)

let rec f x y z = f (y,y) z x

let f = List.fold_left (fun acc x->x/acc)

let rec f x y = match (x,y) with
  ([] , v) -> v
  | (_::l , _) -> f l true
```

Question 1.2 Donner le type de r après chaque expression.

```
let r = ref []

let _ = List.map (fun (f,x)->(f x)+1) !r

let _ = List.iter (fun (_,(x,y)) -> print_string x;print_int y) !r
```

2 Évaluation (4 points)

Question 2.1 Étant donnée la fonction `mystere` suivante :

```
let f l = List.map (fun x -> incr x; if !x>2 then (ref 10) else x) l
```

Donner le résultat de l'évaluation suivante :

```
f (let r = ref 1 in [r;r;r;r])
```

Question 2.2 On donne la fonction suivante :

```
let rec f b =  
  if !b <= 0 then raise Exit;  
  begin  
    b := !b - 2;  
    f b;  
    b := !b + 4  
  end
```

Donner, en le justifiant, le résultat de l'évaluation de l'expression suivante :

```
let a = ref 9 in try f a; !a+20 with Exit -> !a
```

3 Le problème des n reines (9 points)

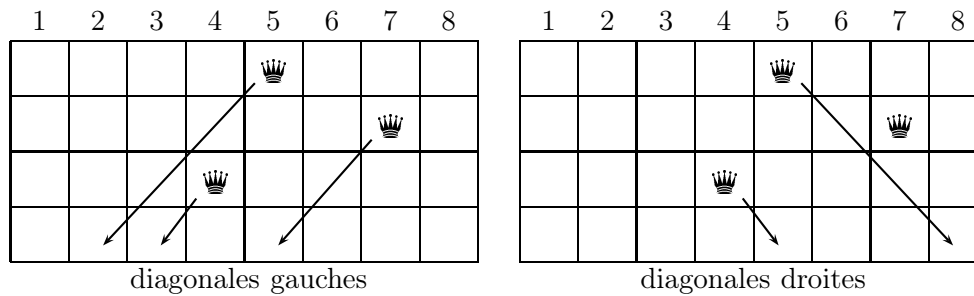
Le problème des n reines consiste à placer n reines sur un échiquier de taille $n \times n$ de telle sorte qu'aucune ne soit en prise : il ne faut donc pas plus d'une reine par ligne, par colonne et par diagonale. Voici un exemple pour $n = 8$.

	♔						
				♔			
						♔	
			♔				
♔							
							♔
					♔		
		♔					

Le but de cet exercice est d'écrire un programme en Ocaml qui permette de calculer le nombre de manières différentes de résoudre ce problème pour un nombre n quelconque de reines. Pour cela, nous allons utiliser un algorithme de recherche avec retour arrière (ou *backtracking* en anglais) qui va remplir les lignes de l'échiquier une à une. Afin de remplir une ligne, l'algorithme maintient 3 ensembles :

- a : contient les numéros des colonnes où il n'y a encore aucune reine de placée ;
- b : contient les numéros des colonnes sur lesquelles il n'est pas possible de placer une reine car ces positions se trouvent sur la diagonale *gauche* d'une reine déjà placée ;
- c : joue le même rôle que b mais pour les diagonales *droites*.

Par exemple, après avoir placé les 3 premières reines sur les colonnes 5, 7 et 4 (dans cet ordre) d'un échiquier 8×8 , on a la situation suivante :



qui est représentée par les trois ensembles $a = \{1, 2, 3, 6, 8\}$, $b = \{2, 3, 5\}$ et $c = \{5, 8\}$. L'ensemble des colonnes où il est encore possible de placer une reine pour la 4^e ligne est donc tout simplement $(a \setminus b) \setminus c$, soit ici $\{1, 6\}$. L'algorithme consiste alors à essayer *une à une* ces positions : récursivement, on cherche les solutions pour lesquelles la 4^e reine est placée sur la colonne 1 puis, en revenant en arrière, récursivement celles pour lesquelles la reine est placée sur la colonne 6. À chaque appel récursif, les ensembles a , b et c sont modifiés en fonction de la colonne i choisie : i est supprimée de a et ajoutée aux ensembles b et c ; ces ensembles b et c sont alors mis à jour en décrémentant (resp. incrémentant) les éléments qu'ils contiennent afin de calculer les nouvelles diagonales des reines placées sur l'échiquier.

Dans la suite, nous allons simplement représenter les ensembles a , b et c par des listes d'entiers (`int list`).

Question 3.1 Écrire une fonction `list_of_int : int -> int list` telle que `list_of_int n` retourne la liste `[1 ; 2 ; ... ; n]` si n est positif et la liste vide sinon.

Question 3.2 Écrire les fonctions `succ_list : int list -> int list` et `pred_list : int list -> int list` telles que `succ_list [a1;a2;...;an]` retourne la liste `[a1+1;a2+1;...;an+1]` et `pred_list [a1;a2;...;an]` retourne la liste `[a1-1;a2-1;...;an-1]`.

Question 3.3 En utilisant un itérateur sur les listes, écrire la fonction `diff : 'a list -> 'a list -> 'a list` telle que `diff l1 l2` retourne une liste contenant les éléments de $l1$ qui ne sont pas éléments de $l2$.

Question 3.4 Écrire une fonction `remove : 'a list -> 'a -> 'a list` telle que `remove l x` retourne une liste contenant les éléments de l sauf x .

La question suivante est l'algorithme principal du problème des n reines. Il s'agit d'écrire une fonction récursive prenant en arguments les 3 ensembles a , b et c décrits précédemment. Cette fonction retourne le nombre de solutions qui prolongent la solution partielle décrite par a , b et c . En particulier, elle retourne l'entier 1 quand l'ensemble a est vide (puisque'il n'y a plus de reines à placer c'est qu'une solution a été trouvée).

Question 3.5 À l'aide des fonctions précédentes, et en utilisant un itérateur sur les listes, écrire une fonction récursive `nb_solutions : int list -> int list -> int list -> int` telle que `nb_solutions a b c` retourne le nombre de solutions correspondant aux ensembles a , b et c donnés en paramètres.

Enfin, la question suivante consiste à écrire la fonction principale du programme qui se contente d'appeler la fonction `nb_solutions` avec comme paramètres les ensembles $a = \{1, 2, \dots, n\}$ et $b = c = \emptyset$.

Question 3.6 Écrire une fonction `reines : int -> int` telle que `reines n` retourne le nombre de solutions au problème des `n` reines.

Rappels.

`List.iter : ('a -> unit) -> 'a list -> unit`

`List.iter f [a1; ...; an]` est `f a1; ...; f an`

`List.map : ('a -> 'b) -> 'a list -> 'b list`

`List.map f [a1; ...; an]` est `[f a1; ...; f an]`

`List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

`List.fold_left f a [b1; ...; bn]` est `f (... (f (f a b1) b2) ...)` `bn`

`List.filter : ('a -> bool) -> 'a list -> 'a list`

`List.filter p l` retourne tous les éléments de `l` qui satisfont le prédicat `p` .

`List.mem : 'a -> 'a list -> bool`

`List.mem x l` retourne `true` si `x` est dans la liste `l` et `false` sinon .