université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

Introduction à la programmation fonctionnelle
L2 Informatique

semaine du 2 octobre 2023

TP n° 4

Consignes les exercices ou questions marqués d'un ⋆ devront être rédigés sur papier (afin de se préparer aux épreuves écrites de l'examen). En particulier, il est recommandé d'être dans les même conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

1 Re-tours de Hanoï

Nous allons revisiter l'exercice sur les tours de Hanoï, dans le but d'afficher en temps réel (à chaque mouvement) l'état de chaque piquet.

On rappelle la solution récursive du problème :

```
1  (* On rappelle la fonction de hanoi du TD 2 *)
2  let hanoi n =
3    let rec hanoi_aux dep mil arr n =
4      if n > 0 then begin
5        hanoi_aux dep arr mil (n - 1);
6        Printf.printf "%s -> %s\n" dep arr;
7        hanoi_aux mil dep arr (n - 1)
8      end
9    in
10   hanoi_aux "dep" "mil" "arr" n
```

Le principe récursif est le suivant :

— cas de base $n = 0$: on n'a rien à faire (pas de **else**)

— cas récursif : on utilise récursivement la fonction `hanoi_aux` pour déplacer $n-1$ disques du piquet de départ vers le piquet du milieu en utilisant le piquet d'arrivée comme zone temporaire. Une fois que c'est fait on est donc dans une configuration où :

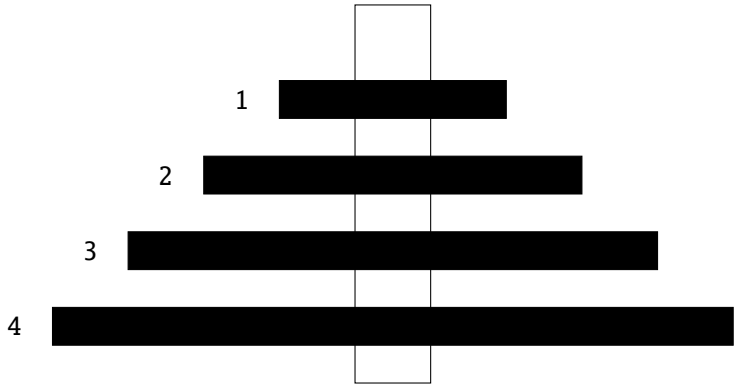
  — le plus grand disque est sur le piquet de départ (car on a bougé les $n-1$ au dessus de lui ;

  — tous les autres disques sont empilés dans le bon ordre sur le piquet du milieu ;

  — le piquet d'arrivée est vide.

  On peut donc déplacer librement le plus grand disque du piquet de départ au piquet d'arrivée (ligne 6). Ensuite, on peut de façon symétrique déplacer les $n-1$ disques du piquet du milieu vers le piquet d'arrivée en utilisant le piquet de départ (maintenant vide) comme zone temporaire.

1. Reporter la fonction `hanoi` ci-dessus au début d'un fichier `hanoi_list.ml` et la tester sur une tour à 4 éléments.
2. On modélise les piquets par une paire de type `string * int list` : le nom du piquet (qui sera `"dep"`, `"mid"` ou `"arr"`) et la liste des disques sur ce piquets. Les disques sont représentés par des entiers proportionnels à leur diamètre. Ainsi, la liste `[1;2;3;4]` représente le piquet :



Définir deux types :

```
type piquet = string * int list
type jeu = piquet list
```

3. Écrire une fonction `affiche_piquet : piquet -> unit` qui affiche un piquet dans la console. Par exemple, le piquet `("mid", [1;2;5])` sera affiché comme ceci :

   ```
   mid|5-2-1-
   ```

   C'est à dire le nom du piquet suivi d'une barre verticale, suivi de chaque nombre suivi d'un tiret. Attention, l'affichage doit commencer par la fin de la liste, il est donc judicieux de la renverser avec `List.rev`.

4. Écrire une fonction `choix_piquet : jeu -> string -> piquet`. Étant donné une liste de piquets et un nom, la fonction renvoie le piquet ayant ce nom. Par exemple :

```
let piquets = [ ("dep", [2;3;4]);
                ("mid", []);
                ("arr", [1])]

let m = choix_piquet piquets "mid"

(* m vaut ("mid", []) *)
```

   Attention, on ne connait pas l'ordre des piquets dans la liste (ce n'est pas forcément `"dep"` en premier. **Indication** : une fonction de la bibliothèque standard sur les listes d'OCaml effectue exactement ce que l'on souhaite.

5. Écrire une fonction `affiche_jeu : jeu -> unit`. Cette dernière doit utiliser `Printf.printf` pour :
   — Afficher la chaîne de caractères `"\x1b[2J\x1b[H"` [1].
   — Afficher d'abord le piquet `"dep"`, puis le piquet `"mid"` puis le piquet `"arr"`. Évidemment, il convient de réutiliser `affiche_piquet` et `choix_piquet`.
   — Afficher la séquence spéciale `"%!"`. Cette dernière, indique de forcer l'affichage dans le terminal.

6. Écrire un fonction `deplace_sommet : piquet -> piquet -> piquet * piquet`. Cette fonction prend deux piquets en argument et déplace le disque au sommet du premier vers le sommet du second piquet. La fonction renvoie les nouveaux piquets. Dans les cas suivant, la fonction lève une exception au moyen de la fonction `failwith` :
   — Le premier piquet est vide.
   — L'entier au sommet du premier piquet est supérieur à celui au sommet du second piquet.

7. Écrire une fonction `joue : jeu -> string -> string -> string -> jeu` telle que `joue piquets src dst` déplace le sommet du piquet `src` au sommet du piquet `dst` et reconstitue le jeu avec :
   — le piquet `autre`
   — les deux piquets `src` et `dst` modifiés.

8. Écrire une fonction `gen_list : int -> int list` telle que `gen_list n` crée une liste contenant les entiers `[1; 2; ... ; n]`.

1. Pour les curieux ou les curieuses, vous pouvez vous renseigner sur les séquences d'échappement ANSI, des caractères spéciaux qui permettent de contrôler le curseur du terminal

9. Enfin, écrire une fonction `hanoi_list n`. Cette dernière résout le problème des tours de Hanoï pour n disques.

   La fonction utilise en interne une fonction auxiliaire

   `hanoi_aux : jeu -> string -> string -> string -> int -> jeu`

   Cette fonction a la même structure récursive que la version simple en début de fichier mais elle doit renvoyer le `jeu` (c'est à dire la liste des trois piquets modifiés).

   On appellera `hanoi_aux` sur la configuration initiale :

```
1  let hanoi_list n =
2   let rec hanoi_aux piquets dep mil arr n =
3     if n > 0 then begin
4       ...
5     end else ...
6   in
7   let final =
8     hanoi_aux [ ("dep", gen_list n);
9                 ("mil", []);
10                ("arr", []) ] "dep" "mil" "arr" n
11  in
12  affiche_jeu final
```

   L'affichage étant trop rapide pour être suivi, on pourra ralentir le programme en mettant, dans le corps de la fonction `hanoi_aux` l'instruction `Unix.sleepf 0.5;`, juste après l'affichage du jeu, ce qui permet au programme de se mettre en pause pendant 0.5s. Pour utiliser cette fonction, vous devez compiler votre programme avec la ligne de commande :

```
$  ocamlc -o hanoi_list unix.cma hanoi_list.ml
```

   ou

```
$  ocamlopt -o hanoi_list unix.cmxa hanoi_list.ml
```

2 Partie Bonus

On souhaite maintenant afficher les disques en couleur, en utilisant les capacités des terminaux ANSI (par exemple l'émulateur de terminal utilisé pour lancer le programme). On peut changer la couleur du texte d'un terminal de deux façons[2]. En affichant la chaîne de caractère `"\x1b[48;5;`n`m"`, la couleur de code n est utilisée comme couleur de fond. En affichant la chaîne de caractères `"\x1b[38;5;`n`m"`, la couleur de code n est utilisée pour le texte. Enfin, la chaîne fixe `"\x1b[0m"` réinitialise la couleur (de fond et du texte).

La plupart des terminaux modernes supportent un code de couleur entre 0 et 255.

Par exemple, le code OCaml suivant :

```
Printf.printf "\x1b[48;5;154m\x1b[38;5;12mHello World!\x1b[0m\n"
```

produit dans le terminal :

Hello World!

(la couleur 154 représente le vert et la couleur 12 le bleu turquoise).

1. Créer une fonction `affiche_disque_couleur : int -> unit` qui affiche le disque donné en couleur. On pourra par exemple utiliser comme couleur de texte le noir et comme couleur de fond $15 - n$ pour un disque de numéro n (on supposera qu'on ne va pas au delà de 15 disques ...). Tout autre choix de couleur est autorisé.
2. Créer une fonction `affiche_disque_mono : int -> unit` qui affiche simplement le disque
3. Créer une fonction `affiche_disque : int -> unit`, telle que cette fonction soit l'une des deux précédente en fonction de la présence de l'argument `-color` sur la ligne de commande
4. Faire en sorte que la fonction `affiche_piquet` appelle `affiche_disque` et tester votre programme.

2. Pour plus de détail, voir `https://en.wikipedia.org/wiki/ANSI_escape_code`

Remarque en cas de bug dans votre programme, votre terminal peut être laissé dans un état inutilisable (e.g. couleur de texte et de fond identique). Vous pouvez dans ce cas faire `CTRL-C` (pour pouvoir interrompre un éventuel programme), puis taper `reset` suivi de la touche entrée. Le terminal sera ré-initialisé, sans avoir besoin de le fermer et le rouvrir.