Devoir d'Informatique n°5

1er février 2023

*
* *

Durée de l'épreuve : 3 heures

L'usage de tout dispositif électronique est interdit.

Consignes

Pour répondre à une question, il vous est permis de réutiliser le résultat d'une question antérieure même si vous n'avez pas réussi à établir ce résultat.

Quand l'énoncé demande de coder une fonction, sauf demande explicite, il n'est pas nécessaire de justifier la correction ou la terminaison de cette fonction, ou de la commenter.

Vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction.

Si vous repérez ce qu'il vous semble être une erreur d'énoncé, signalez-le sur votre copie et poursuivez la composition en expliquant les éventuelles initiatives que vous aurez pris.

Vous devrez traiter les questions de programmation dans le langage OCaml.

Les trois parties du sujet sont indépendantes

OCaml

Question 1. Trouver des expressions OCaml ayant exactement les types suivants :

1. `(int list * string) list`
2. `'a -> 'a`
3. `int -> int`
4. `'a -> int`
5. `('a * 'b) -> 'a`
6. `('a -> 'b -> 'a * 'b`
7. `('a -> 'b)-> 'a -> 'b`
8. `'a -> 'b -> 'a`
9. `('a -> 'b -> 'c)-> ('a * 'b -> 'c)`
10. `('a * 'b -> 'c) -> ('a -> 'b -> 'c)`

Question 2. Écrire une fonction `zip: ('a list * 'b list ) -> ('a * 'b) list` qui prend en entrée deux listes L1 et L2 et renvoie une liste constituée de tuples dont la première composante est un élément de L1, et la deuxième composante un élément de L2, en prenant les éléments des deux listes dans le même ordre. Par exemple : `zip ['a'; 'b'; 'c'] [1; 2; 3]` renvoie `[('a', 1); ('b', 2); ('c', 3)]`. Les deux listes doivent être de même taille.

Question 3. Écrire une fonction récursive terminale permettant de calculer le maximum d'une liste. Préciser son type.

Question 4. Écrire une fonction `empaqueter: float list -> float list list` qui prend en argument une liste L de valeurs de l'intervalle $[0, 1]$ et la subdivise en sous-listes d'éléments de L consécutifs de somme inférieure où égale à 1. La liste de listes renvoyées devra être telle que la concaténation des sous-listes donne L. Par exemple, pour $L = [0.1, 0.8, 0.3, 0.7, 0.1, 0.1, 0.5]$, la fonction renverra $[[0.1; 0.8]; [0.3; 0.7]; [0.1; 0.1; 0.5]]$.

En OCaml, les listes doivent être homogènes : tous les éléments d'une liste doivent être de même type. On se propose d'implémenter les ***listes imbriquées***, qui permettent de représenter des objets comme $[2; [3; 4]; [[5]; [6]; 7]]$ (qui n'existent donc pas en OCaml) où cohabitent des entiers, des listes d'entiers, des listes de listes d'entiers, etc... On utilise le type suivant :

```
1 type 'a nested_list =
2     | One of 'a (* élément simple *)
3     | Many of 'a nested_list list ;; (* liste *)
```

Par exemple, la liste imbriquée $[1; [2]; 3]$ sera représentée par la valeur `Many [One 1; Many [One 2]; One 3 ]`, et la liste imbriquée $[[]; 3; [[2]]$ par l'expression `Many [ Many []; One 3; Many [Many [One 2]] ]`

Question 5. Écrire une fonction `nested_map: ('a -> 'b)-> 'a nested_list -> 'b nested_list` appliquant une fonction à tous les éléments d'une liste imbriquée. (*Il pourra être utile de recoder la fonction map standard des listes pour vous aider*)

Question 6. Écrire une fonction `flatten: 'a nested_list -> 'a list` qui aplatit une liste imbriquée en une liste simple. Par exemple, pour $[2; [3; 4]; [[5]; [6]; 7]]$, la liste aplatie correspondante est $[2; 3; 4; 5; 6; 7]$.

Tours de Hanoï

On s'intéresse au jeu des tours de Hanoï. Ce jeu est constitué de N disques de diamètres $D_0 < D_1 < \cdots < D_{N-1}$, percés en leur centre, et de trois piquets notés A, B, C. Initialement, les N disques sont enfilés sur le piquet A, dans l'ordre, avec le disque le plus large en bas. Le but du jeu est de déplacer les disques, de façon à ce qu'ils soient enfilés sur le piquet C, dans le même ordre qu'au départ. On ne peut déplacer qu'un disque à la fois, et on ne peut pas placer un disque sur un disque plus petit.

Dans cet exercice, on s'intéresse à la modélisation puis à la résolution de ce jeu.

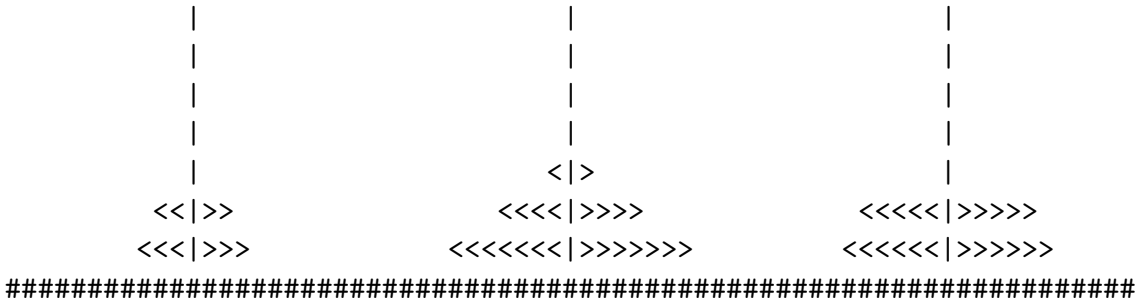


FIGURE 1 – Un jeu de tours de Hanoï à 7 disques imprimé dans le terminal. Dans cet état de jeu, les piquets A et C ont deux disques, et le piquet B en a trois.

On se fixe un entier N dans la suite, et l'on suppose que le contexte dans lequel on définit nos fonctions et expressions en OCaml contient N.

Disques

Pour simplifier l'affichage, on suppose que pour $k \in [\![0, N-1]\!]$, le disque d'indice k est de diamètre $2k+1$. On rappelle que l'opérateur `^` en OCaml permet de concaténer deux chaînes de caractères.

Question 1. Donnez une fonction `string_of_disc: int -> int -> string` qui étant donné deux entiers k et n calcule une chaîne de longueur totale $2n+1$ dont le centre est de la forme $<<<\ |\ >>>$ avec k paires de chevrons, et dont les $2n+1-2k$ autres caractères sont des espaces. Par exemple, `string_of_disc 2 2` renvoie `"<<|>>"` mais `string_of_disc 2 5` renvoie `" <<|>> "`.

Tours

On appelle ***tour*** un empilement de disques de diamètres croissants. Un tour est donc entièrement déterminée par l'ensemble des disques qu'elle contient, car l'ordre est alors fixé. On peut donc représenter une tour par un sous-ensemble de $[\![0, N-1]\!]$, que l'on encodera par un entier de N bits :

$$\begin{array}{rccc} & \mathcal{P}([\![0, N-1]\!]) & \longrightarrow & [\![0, 2^N - 1]\!] \\ \varphi : & S & \longmapsto & \displaystyle\sum_{i \in S} 2^i \end{array}$$

Dans le code, on utilisera le type `tour` pour signaler que les fonctions prennent en entrée un entier représentant une tour :

```
type tour = int ;;
```

Question 2. Dessiner la tour encodée par l'entier 12.

Question 3. Comment se traduit dans l'encodage des tours le fait de déplacer un disque d'une tour X à une tour Y ?

Question 4. Donner une fonction `val_disque_min: tour -> int` qui prend en entrée une tour X, et qui calcule 2^{k_0} où k_0 est l'indice du plus petit disque de X (i.e. celui au sommet).

Question 5. Donner une fonction `deplace: (tour*tour) -> (tour*tour)` prenant en entrée un couple (X, Y) de tours, et renvoyant les tours (X', Y') obtenues en déplaçant le plus petit disque de X vers Y.

Question 6. Définir une fonction `tour_pleine: int -> tour` qui prend en argument un entier n et renvoie l'encodage de la tour contenant tous les disques d'indice 0 à $n-1$. La fonction devra être de complexité $\mathcal{O}(\log n)$.

Etat de jeu

On donne deux opérateurs OCaml utiles : `lor`, `land`, les opérateurs bit-à-bit, qui appliquent un OU logique (resp. un ET logique) sur chacun des bits de leurs deux entrées. Ce sont des opérateurs ***infixes*** :

```
5 land 3 ;; (* 101 ET 011: vaudra 001, c'est à dire 1 *)
5 lor 3 ;; (* 101 OU 011: vaudra 111, c'est à dire 7 *)
```

On appelle ***état du jeu*** la donnée des trois tours.

Question 7. Définir le type `etat` à partir du type `tour`.

Question 8. Définir une fonction `etat_valide: etat -> bool` prenant en entrée un état de jeu, et déterminant si les deux conditions suivantes sont réalisées :

- Les tours font apparaître tous les disques $0 \dots n-1$
- Aucun disque n'apparaît deux fois

Question 9. Définir une fonction `etat_init` qui prend en argument un entier n et renvoie l'état initial d'une partie de jeu à n disques.

Piquets

On définit le type suivant :

```
type piquet = A | B | C ;;
```

Question 10. Définir une fonction `autre_piquet` prenant en entrée deux piquets distincts et renvoyant le troisième. Si les deux piquets en entrée sont égaux, la fonction devra déclencher une erreur.

Question 11. Définir une fonction `deplacer: etat -> piquet -> piquet` qui déplace le disque au sommet d'un piquet vers un autre piquet, dans un état de jeu donné. Votre fonction devra vérifier que les entrées sont cohérentes, par exemple que le piquet de départ n'est pas vide, et lever une exception avec message d'erreur explicatif lorsqu'il le faut. Listez explicitement les différentes erreurs que votre fonction traite.

Résolution

On suppose que l'on a une fonction `print_etat: etat -> unit` qui affiche l'état d'une partie.

Question 12. Donner une suite de coups permettant de résoudre le jeu des tours de Hanoï avec 3 disques. Pour avoir tous les points, vous devez y arriver en 7 coups.

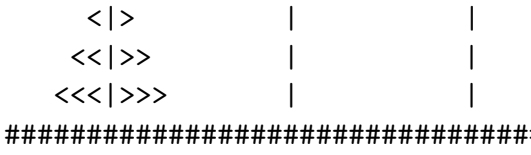


FIGURE 2 – État initial pour 3 disques

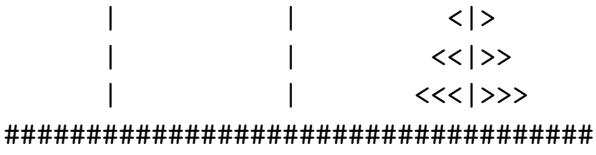


FIGURE 3 – État cible pour 3 disques

Question 13. Écrire une fonction récursive `deplacer_tour: etat -> piquet -> piquet -> int -> etat`, telle que `deplacer_tour e x y n` déplace les n disques au sommet de x vers y, depuis l'état e, en affichant tous les états successifs (état initial inclus, état final exclus). On supposera comme précondition que les n disques au sommet de x sont les n plus petits disques du jeu

Question 14. En déduire une fonction `resoudre_hanoi: unit -> unit` qui affiche les états successifs permettant de résoudre le jeu des tours de Hanoï à N disques. Combien de coups nécessite cette résolution ?

Réseaux de tri

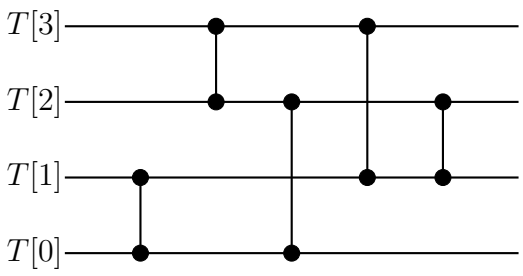
On se propose d'étudier les ***réseaux de tri***. Il s'agit d'une certaine catégorie d'algorithmes de tri, dans lesquels on fixe à l'avance une liste de comparaisons et d'échanges qui constituent ledit réseau.

Plus précisément, un réseau agit sur un tableau T de taille n, et est décrit par une liste de ***modules de comparaison***. Un module de comparaison est décrit par la donnée d'une paire (i, j) de deux indices tels que $0 \leq i < j < n$. Un tel module a pour effet de comparer les cases $T[i]$ et $T[j]$, et de les échanger si et seulement si $T[j] < T[i]$.

On représentera un réseau de tri de la manière suivante :

- Chaque case du tableau est représenté par une ligne horizontale
- Les modules de comparaison sont représentés de gauche à droite. Chacun d'entre eux est représenté par un segment vertical reliant les lignes horizontales correspondant aux deux cases concernées.

Par exemple, le réseau de tri $(0, 1), (2, 3), (0, 2), (1, 3), (1, 2)$ est représenté par le schéma suivant :



Attention, malgré son nom, un réseau de tri ne trie pas nécessairement son entrée. Par exemple, le réseau vide ne contenant aucun module est un réseau de tri !

La ***taille*** d'un réseau de tri est le nombre de modules de comparaison qu'il contient. La ***profondeur*** d'un réseau de tri est le nombre maximal de modules qu'une valeur doit traverser avant d'arriver à la fin du réseau.

Dans la suite, on considèrera des tableaux d'entiers uniquement, mais les résultats seraient applicables à n'importe quel ensemble totalement ordonné.

Question 1. Exécuter le réseau de tri donné en exemple sur le tableau $[2, 4, 5, 1]$.

Question 2. Donner la taille et la profondeur de ce réseau, en donnant un exemple de chemin réalisant cette profondeur.

Question 3. Ce réseau permet-il de trier n'importe quel tableau ? Donner un contre-exemple ou une preuve.

Soit T un tableau de taille n. Étant donné un réseau R, on note RT le tableau obtenu après exécution de R sur T. Étant donné une fonction $f : \mathbb{N} \to \mathbb{N}$, on note fT le tableau $[f(T[0]), f(T[1]), \ldots, f(T[n-1])]$.

Question 4. Donner un réseau de tri S_n de taille $(n-1)$ tel que pour T un tableau de taille n, $S_nT[n-1] = \max T$.

Question 5. Pour $n \in \mathbb{N}$, donner un réseau de tri qui trie tout tableau de taille n. On ne demande aucun critère d'optimalité. Quelle est sa taille ? Sa profondeur ?

Question 6. Montrer que si $f : \mathbb{N} \to \mathbb{N}$ est croissante, alors pour tout réseau de tri R, $f(RT) = R(fT)$, autrement dit, f commute avec R.

Question 7. Montrer qu'un réseau de tri trie tout tableau d'entiers de taille n si et seulement si il trie tous les tableaux booléens de taille n.

Question 8. On considère dans cette question que $n = 2^k$ avec $k > 1$. Donner un réseau de tri permettant de trier un tableau T de taille n si $T[0] \ldots T[\frac{n}{2} - 1]$ et $T[\frac{n}{2}] \ldots T[n-1]$ sont triés, utilisant $\mathcal{O}(n \log n)$ modules de comparaison. (*Indication : on pourra utiliser une approche récursive, en commençant par trier les positions paires et impaires indépendamment.*)

Question 9. En déduire un réseau de tri efficace permettant de trier tout tableau de taille n si n est une puissance de 2.

Question 10. Donner la taille et la profondeur du réseau donné à la question précédente.

Question 11. Adapter le réseau des questions précédentes pour n quelconque.

*
* *