Automates - Implémentation

Florian Bourse

1 Ensembles

Pour implémenter des automates, nous avons besoin de représenter des ensembles d'états. Nous allons les représenter par des listes d'entiers **sans doublon** et **triées**.
Pour s'échauffer, nous allons implémenter plusieurs fonctions qui pourrons nous être utiles plus tard pour manipuler ces ensembles.

Question 1. Écrire une fonction `insere : 'a -> 'a list -> 'a list` qui ajoute un élément à un ensemble si il n'y appartient pas déjà.

Question 2. Écrire une fonction `appartient : 'a -> 'a list -> bool` qui teste l'appartenance d'un élément à un ensemble.

Question 3. Écrire une fonction `egaux : 'a list -> 'a list -> bool` qui teste l'égalité entre deux ensembles.

Question 4. Écrire une fonction `union : 'a list -> 'a list -> 'a list` qui renvoie l'ensemble des éléments appartenant à au moins un des deux ensembles.

Question 5. Écrire une fonction `intersection : 'a list -> 'a list -> 'a list` qui renvoie l'ensemble des éléments appartenant aux deux ensembles.

2 Automates déterministes

Pour commencer, on va s'intéresser aux automates déterministes. Ils sont définis par le type

```
type automate_det =
  {
    transitions : (char,int) Hashtbl.t array;
    init : int;
    final : int list;
  };;
```

où les états sont les nombres de 0 à $n-1$, et n est la taille du tableau `transitions`. `init` est l'état initial. `final` est l'ensemble des états finaux (liste triée sans doublons). Et pour `auto` un automate déterministe, q un état et c un caractère, `Hashtbl.find auto.transitions.(q) c` donne l'état q' tel que $q \xrightarrow{c} q'$.

Question 6. Compléter la fonction `evalue_det` qui prend en entrée un automate déterministe et une liste de caractères et qui renvoie `true` si l'automate accepte le mot et `false` sinon.

3 Automates non déterminisites

On généralise maintenant notre implémentation aux automates non déterministes, du type

```
type automate =
  {
    transitions : (char, int list) Hashtbl.t array;
    init : int list;
    final : int list;
  };;
```

`init` est maintenant un ensemble d'états, et `Hashtbl.find auto.transitions.(q) c` aussi.

Question 7. Compléter la fonction `evalue` qui prend en entrée un automate non déterministe et une liste de caractères et qui renvoie `true` si l'automate accepte le mot et `false` sinon. On pourra créer une fonction auxiliaire récursive `evalue_depuis` qui prend en argument une liste d'états et une liste de caractères à lire.

Question 8. Donner un automate non déterministe qui reconnaît les nombres dont le dernier chiffre n'apparaît qu'une seule fois, et l'implémenter.

Nous allons maintenant écrire une fonction pour émonder un automate non déterministe.

Question 9. Compléter les fonctions `accessibles_depuis`, `accessibles` et `coaccessibles`.

Question 10. Compléter la fonction `emonde` qui émonde un automate non déterministe. Pour renommer les états, on pourra créer un tableau de type `int option array` dont la i-ème case contient `Some j` si le i-ème état devient l'état j et `None` si il n'est pas accessible ou pas coaccessible.

4 Transitions spontanées

Pour ajouter les transitions spontanées, on change le type des automates pour étiqueter les transitions par un type `label` qui peut être soit `Epsilon` pour une transition spontanée ou `C c` pour une transition étiquettée par le caractère `c`.

```
type automate_eps =
  {
    transitions : (label, int list) Hashtbl.t array;
    mutable init : int list;
    mutable final : int list;
  }
and label = Epsilon | C of char;;
```

Pour simplifier le travail sur les automates avec transitions spontanées, on a choisi de rendre tous ses champs mutables pour pouvoir le modifier plutôt que de reconstruire un nouvel automate à chaque étape.

Question 11. Compléter la fonction `fermeture_arriere` qui modifie un automate avec transition spontanées pour ajouter toutes les transitions qui peuvent être faites en un caractère (qui peuvent être composées de plusieurs ε-transitions puis d'une transition avec un caractère), et qui met à jour les états finaux pour inclure les états qui peuvent atteindre un état final avec des ε-transitions uniquement.

Question 12. Compléter la fonction `remove_eps` qui renvoie un automate équivalent sans transition spontanées.