S03 : Haskell (Listes et analyse lexicale)

Enseignant : Stéphane LE PEUTREC
Assistant : Jonathan LAUPER

Instructions

- Deadline : jeudi suivant à 11:00

Exercice 1

Développez les fonctions suivantes (avec leur type).

- **flatten list** : cette fonction prend en paramètre une liste de liste et retourne en résultat la liste aplatie
  Exemple : flatten [[2,3], [4,2,7], [6,9]] retourne la liste [2,3,4,2,7,6,9]

- **partitions list** : cette fonction prend en paramètre une liste et retourne en résultat la liste des partitions de cette liste
  Exemple : partitions [1,2,3] retourne la liste [[],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]
  Utilisez une liste en compréhension. N'utilisez pas de fonctions d'ordre supérieur.

- **permutations list** : cette fonction prend en paramètre une liste et retourne en résultat la liste des permutations des éléments de cette liste
  Exemple : permutations [1,2,3] retourne la liste [[1,2,3], [1,3,2], [2,1,3], [2,3,1],[3,1,2],[3,2,1]]
  Utilisez une liste en compréhension et la fonction delete' de la série précédente.
  N'utilisez pas de fonctions d'ordre supérieur.

Exercice 2

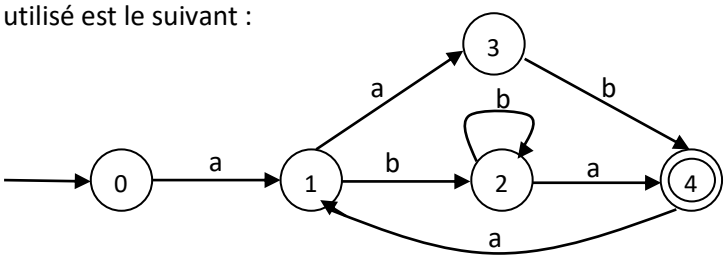
Développez à nouveau les fonctions de l'exercice 3 de la série 2 (avec leur type) mais cette fois en utilisant les liste en compréhension.

- length' l : retourne la longueur de la liste l (son nombre d'élément).
  exemple : length' [1,5,7] retourne 3
- deleteAll' e l : retourne la liste l sans toutes ses occurrences de e
  exemple : deleteAll' 5 [1,5,7,5,8] retourne [1,7,8]
- toUpperString ch : retourne la chaîne de caractères ch en majuscule
  exemple : toUpperString "hello" retourne "HELLO"

Exercice 3 : analyse lexicale - première étape

On propose d'implémenter de deux manières différentes le parcours d'un automate à états finis.

L'automate utilisé est le suivant :



a) Première méthode : Représentation de l'automate à l'aide de fonctions

On choisit de représenter l'automate à l'aide des fonctions suivantes :

- **isFinalState :: Int -> Bool** cette fonction retourne vrai si le Int passé en paramètre est un état de sortie et faux sinon

- **firstState :: Int** cette fonction sans paramètre retourne en résultat l'état initial

- **transition :: Int -> Char -> Int** cette fonction prend en paramètre un état e et un caractère c et retourne en résultat l'état d'arrivée en suivant la transition qui part de l'état e en portant le caractère c. Elle retourne -1 si il n'y a pas de transition portant le caractère c partant de l'état e.
  Exemples :
  l'appel **transition 1 'b'** retourne 2
  l'appel **transition 4 'b'** retourne -1

Développez les fonctions qui suivent :

- **isToken ch** cette fonction prend en paramètre une chaîne de caractère et retourne vrai si cette chaîne est reconnue par l'automate, et faux sinon
  Exemples :
  l'appel **isToken ''aababba''** retourne True
  l'appel **isToken ''aaa''** retourne False

- **reconizedFromState state ch** cette fonction prend en paramètre un état state et une chaîne de caractères. Elle retourne vrai si la chaîne ch est reconnue par l'automate en partant de l'état state.
  Exemples :
  l'appel **reconizedFromState 2 ''bba''** retourne **True** car il existe un chemin de trois transitions partant de l'état 2 et arrivant à un état final
  l'appel **reconizedFromState 1 ''aba''** retourne False car quand on suit le chemin partant de l'état 1 portant les transitions a, b alors on arrive sur l'état 1 qui n'est pas final.

b) Deuxième méthode : Représentation de l'automate par un triplet

On choisit de représenter un automate par un triplet de la forme (state, [state],[transition]) où le premier élément est l'état initial, le second élément est la liste des états finaux et le troisième élément est la liste des transitions.

Une transition est elle-même un triplet de la forme (state, char, state) où le premier élément est l'état de départ de la transition, le second élément est le caractère porté et le troisième élément est l'état d'arrivée.

Pour simplifier la lecture du code, on définit les types suivants :

```
type State = Int
type Transition = (State,Char,State)
type Automate = (State, [State], [Transition])
```

On crée l'automate de l'exercice comme suit :

monAutomate = (0,[4],[(0,'a',1), (1,'b',2),(1,'a',3), (2,'a',4),(2,'b',2), (3,'b',4), (4,'a',1)])

Développez les fonctions qui suivent :

- **isToken ch stateMachine** cette fonction prend en paramètre une chaîne de caractères et un automate et retourne vrai si cette chaîne est reconnue par l'automate, et faux sinon
  Exemples :
  l'appel **isToken ''aababba'' monAutomate** retourne True
  l'apple **isToken ''aaa'' monAutomate** retourne False

- **reconizedFromState state ch stateMachine** cette fonction prend en paramètre un état state, une chaîne de caractères et un automate. Elle retourne vrai si la chaîne ch est reconnue par l'automate en partant de l'état state.
  Exemples :
  l'appel **reconizedFromState 2 ''bba'' monAutomate** retourne **True** car il existe un chemin de trois transitions partant de l'état 2 et arrivant à un état final
  l'appel **reconizedFromState 1 ''aba'' monAutomate** retourne False car quand on suit le chemin partant de l'état 1 portant les transitions a, b alors on arrive sur l'état 1 qui n'est pas final.

- **isFinalState state stateMachine** cette fonction prend en paramètre un état et un automate. Elle retourne vrai si l'état est final dans cet automate
  Exemples :
  l'appel **isFinalState 4 monAutomate** retourne True
  l'apple **isFinalState 1 monAutomate** retourne False

- **nextState state char stateMachine** cette fonction prend en paramètre un état, un caractère et un automate. Elle retourne l'état d'arrivée de la transition partant de l'état et portant le caractère. Elle retourne -1 si la transition n'existe pas.
  Exemples :
  l'appel **nextState 3 'b' monAutomate** retourne 4
  l'apple **nextState 3 'a' monAutomate** retourne -1

c) Selon vous quels sont les avantages et les inconvénients respectifs de ces deux méthodes ? Quelle méthode est selon vous la meilleure et pourquoi ?