Paradigmes et Langages de Programmation

Haute École d'Ingénierie et de Gestion du Canton de Vaud

6. Analyse lexicale

2022

Exercice 1

Soit l'extrait de code Haskell suivant :

```
-- Perform the bubble sort
bsort s =
  case bsort' s of
    t | t == s     -> t
      | otherwise -> bsort t
  where
    bsort' (x:x2:xs)
      | x > x2     = x2:(bsort' (x:xs))
      | otherwise = x:(bsort' (x2:xs))
    bsort' s = s
```

Identifiez les lexèmes du code proposé et regroupez-les selon les unités lexicales suivantes :

- Identifiants
- Mots-clés
- Ponctuation
- Opérateurs
- Littéraux
- Commentaires

Exercice 2

Donnez les expressions régulières des langages suivants ayant pour alphabet $\Sigma = \{0, 1\}$:

a. { w | w commence avec un 1 et finit avec un 0 }
b. { w | w contient au moins trois 1 }
c. { w | w contient la sous-chaîne 0101, i.e. x0101y pour x et y }
d. { w | w a une longueur d'au moins trois et son troisième symbole est 0 }
e. { w | w commence avec 0 et a une longueur impaire, ou l'inverse }
f. { w | w ne contient pas la sous-chaîne 110 }
g. { w | la longueur de w est au plus cinq }
h. { w | w est n'importe quelle chaîne excepté 11 et 111 }
i. { w | chaque position impaire de w est un 1 }
j. { w | w contient au moins deux 0 et au plus un 1 }
k. { ϵ, 0 }
l. ø
m. Toutes les chaînes à l'exception de la chaîne vide

Exercice 3

Construisez un automate fini déterministe pour chacun des langages de l'exercice précédent.

Exercice 4

Écrivez une fonction d'appartenance *f : : State -> String -> State* décrivant l'automate *a.* de l'exercice précédent. Cette fonction devra consommer une chaîne de caractères à partir d'un état de départ et retourner l'état courant une fois la chaîne entièrement consommée.

Il vous faudra définir un type *State* énumérant les états possibles de votre automate.

Exercice 5

Construisez un automate fini non-déterministe pour chacun des langages suivants avec pour alphabet $\Sigma = \{0, 1\}$:

a. Le langage { w | w se termine avec 00 } en 3 états
b. Le langage { w | w contient la sous-chaîne 0101, i.e. x0101y pour certains x et y } en 5 états
c. Le langage { w | w contient un nombre nombre pair de 0 ou exactement deux 1 } en 6 états
d. Le langage { 0 } avec 2 états
e. Le langage 0*1*0+ avec 3 états
f. Le langage 1*(001+)* avec 3 états
g. Le langage $\{\epsilon\}$ avec 1 état
h. Le langage 0* avec 1 état

Exercice 6

Définissez un type *Token* et implémentez une fonction d'analyse lexicale qui permet de découper un flux de caractères. Vous veillerez à générer des *tokens* spécifiques pour les types de lexèmes suivants :

- les caractères spéciaux : espaces, parenthèses, ...
- les nombres (uniquement des entiers)
- les symboles qui désigneront des noms de variable

Optionnel : Ajoutez à votre analyse lexicale la notion de position dans le flux d'entrée afin que votre analyseur puisse indiquer la position d'une erreur éventuelle.

Exercice 7

Créez un analyseur lexical avec *Alex* pour un langage d'expressions supportant les constructions suivantes :

- variables
- nombres
- caractères
- opérations binaires
- déclarations de variable (*let-in*)
- conditions (*if-then-else*)

Bon travail !