POLYTECHNIQUE DE MONTRÉAL

LOG2810
STRUCTURE DISCRÈTES

TP 2 : Automates

Mejdi Ghannem (1679027)
Gabriel-Andrew Pollo-Guilbert (1837776)

Remis à
Juliette TIBAYRENC

18 avril 2018

Table des matières

1 Introduction

Dans ce laboratoire, l'étudiant qui a réalisé la série de braquages lors du précédant TP a récidivé. En effet, cette fois-ci il s'attaque a la sécurité des coffres forts des banques. Il a réussi a obtenir une liste de mots de passe pour différentes banques mais sachant qu'ils ont pu être corrompus, il nous demande de l'aider en créant l'algorithme lui permettant de trouver les bons mots de passe en utilisant les différents règles de productions qu'il a trouvé.

Le laboratoire a donc pour objectifs de nous familiariser avec les notions théoriques sur les langages et automates vus dans le cadre du cours LOG2810.

2 Solution

La solution utilisée pour résoudre le problème est relativement simple. Deux classes ont été conçues à cet effet. La figure 1 représente le schéma UML de ces deux classes.

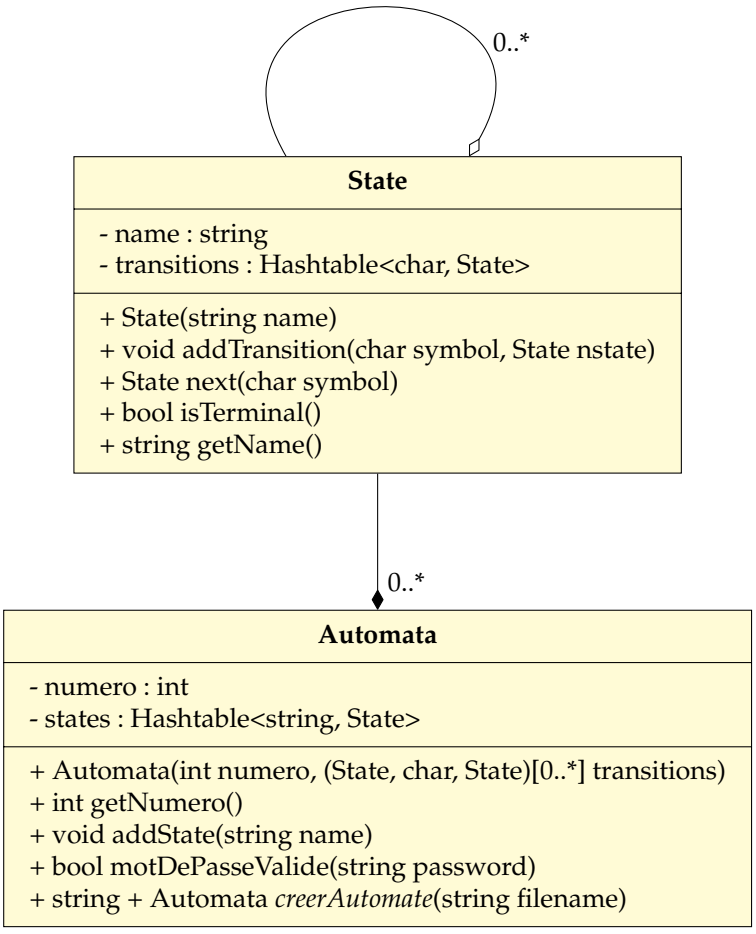

FIGURE 1 – Schéma UML des classes

2.1 Classe State

Cette classe `State` représente un état dans l'automate. Un état a un nom et une liste de transitions possibles représentées par un dictionnaire. Puisque seulement une automate utilise cette classe d'état dans notre projet, il n'est pas nécessaire d'ajouter des sorties aux transitions.

Il y a 3 méthodes importantes à cette classe. La première est `addTransition`. Celle-ci ajoute une transition dans le dictionnaire s'il n'existe pas déjà une entrée avec le symbole. Il y a ensuite la méthode `next` qui permet d'effectuer une transitions selon un symbole entrée. S'il n'existe aucune transition selon l'entrée spécifiée, alors une valeur nulle est retournée.

Finalement, il y a la méthode `isTerminal` qui retourne une valeur booléenne si l'état un état terminal ou non. Un état est terminal s'il n'y a aucune transition à partir de celui-ci.

2.2 Classe Automate

La deuxième classe utilisée dans le programme est la classe `Automata`. Celle-ci est composée d'une liste d'états et d'un numéro d'automate. Elle a deux méthodes d'intêrets. La première est `addState` qui ajoute un état vide avec un nom spécifié si et seulement si un état avec le même nom n'existe pas. Il y a ensuite la méthode `motDePasseValide` qui vérifie si un mot de passe reconnu par l'automate.

Soit un mot de passe $\mathcal{P}$ une chaîne $(\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n)$ de longueur n où $\mathcal{S}_i \in \mathcal{V}$ sont des symboles du vocabulaire $\mathcal{V}$. L'algorithme commence à l'état initiale $\mathcal{I}$ étant la lettre `S` dans notre cas. Par la suite, il avance dans $\mathcal{P}$ tout en tentant de changer l'état actuel $\mathcal{A}$. Dans le cas où l'état actuel est null, alors aucune transition n'existait et le mot n'est pas reconnu par l'automate. Dans le cas où tout les symbols de $\mathcal{P}$ sont traversés, il faut regarder si l'état final est un état terminal. L'algorithme 1 montre l'algorithme énoncé ci-dessus.

Algorithme 1 Algorithme reconnaissant un mot de passe

```
1:  function MotDePasseValide(P)
2:      A ← I
3:
4:      for all S ∈ P do
5:          A ← Next(A, S)
6:
7:          if A = ∅ then return False
8:      end for
9:
10:     return IsTerminal(A)
11: end function
```

3 Difficultés rencontrées

Compte tenu de la simplicité des productions offertes dans les tests, peu, voir aucune, difficulté a été rencontré.

4 Conclusion

Pour conclure, ce laboratoire nous a permis de mettre en pratique la théorie vue en classe. Nous avons pu implémenter l'automate et trouver les bons mots de passe des coffres forts des banques. Le temps passé dans ce laboratoire a été adéquat.