FISE ICy 3A - Automates et Langages.
Ubassy Bastien et Zimmermann Maxime
2022-2023

Sommaire

Partie 1

Squelette

> Nous avons divisé le problème en deux classes principales : `Automate` et `Transition` et une classe de test : `Main`.

Classe `Automate`

- Un objet automate est constitué de :
  - Un **alphabet** `ArrayList<Character>`
  - Un **état initial** `String`
  - Une **liste d'états finaux** `ArrayList<String>`
  - Une **liste de transitions** `ArrayList<Transition>`
- Cette classe contient les getters, les setters et les checkers mais aussi la méthode `afficher()` qui permet d'afficher les informations de l'automate et la

méthode `getFinStates()` qui permet de récupérer l'état final d'une transition à partir de l'état initial et du symbole si tant est que cette transitions est recensé dans la liste de transitions.

Classe `Transition`

- Un objet transition est constitué de :
  - Un **état initial** `String`
  - Un **état final** `String`
  - Un **symbole** `char`
- Cette classe contient les getters, les setters mais aussi une réécriture de la méthode `toString()` qui permet d'afficher les informations de la transition.

Classe `Main`

- La classe `Main` contient la méthode `main` qui permet de tester les méthodes de la classe `Automate`.

Constructeur à partir d'un fichier `Automate(String nomDeFichier)`

- Cette méthode lit un fichier grâce à la classe `FileReader`. Ce fichier est copié dans un **tableau de caractères** puis dans une **chaîne de caractères**. La chaîne est divisée en lignes dans un **tableau de chaînes de caractères**. C'est ce tableau que la méthode parcourt pour récupérer les informations de l'automate :

```
File doc = new File(nomDeFichier);
doc.createNewFile();
FileReader freader = new FileReader(doc);
char [] fichierTab = new char[(int) doc.length()];
freader.read(fichierTab);
freader.close();
String fichierString = new String(fichierTab);
String[] lines = fichierString.split("\n");
```

- Pour récupérer les bonnes informations au bon endroit, la méthode a un compteur qui s'incrémente à chaque fois que le caractère `#` est rencontré. Un `switch case` s'occupe de définir :

L'**alphabet** :

```
case 1:
    alphabet.add(line.charAt(0));
    break;
```

L'**état initial** :

```
case 3:
    initialState = line.substring(0, line.length() - 1);
    break;
```

> `line.substring(0, line.length() - 1);` enlève le dernier caractère de la ligne, en l'occurence : `\n`

Les **états finaux** :

```
case 4:
    finalStates.add(line.substring(0, line.length() - 1));
    break;
```

Les **transitions**.

Zoom sur les transitions

- Pour les transitions, la ligne est parcourue caractère par caractère et il y a un `switch case` supplémentaire afin de définir :

L'**état initial de la transition** :

```
case 1:
    etatInit += caract;
    break;
```

Le **symbole de la transition** :

```
case 2:
    symbol = caract;
    break;
```

L'**état final de la transition** :

```
case 3:
    etatFin += caract;
    break;
```

- A la fin de ce `switch case`, un objet `Transition` est créé :

```
transitions.add(new Transitions(etatInit, etatFin, symbol));
```

Méthode `boolean appartient(String mot)`

Pour effectuer la méthode `appartient`, qui permet de vérifier si un mot donné appartient au langage, nous commençons par définir des fonctions annexes nécessaires à la vérification :

- `boolean isFinalState(String state)` : vérifie si l'état donné en paramètre est un état final.
- `boolean isInTransitions(String s, char c)` : vérifie si la transition partant de l'état `s` et ayant pour symbole `c` existe.
- `String getFinState(String s, char c)` : renvoie l'état final de la transition partant de l'état `s` et ayant pour symbole `c`.
- `boolean isInAlphabet(char c)` : vérifie si le caractère `c` est dans l'alphabet de l'automate.

Avec ces méthodes on aura ainsi une fonction plus lisible mais aussi une différenciation des raisons pour lesquelles un mot n'appartient pas au langage plus simple.

La méthode `appartient` a alors un fonctionnement assez simple. On va partir de l'état initial du mot et parcourir le mot donné jusqu'à une erreur ou la fin du mot.

Dans les cas où l'on arrive à la fin du mot, on vérifie si l'état final est un état final. Si c'est le cas, le mot appartient au langage, sinon il ne lui appartient pas :

```
if (i == mot.length()) {
  fin = true;
  if(isInFinalStates(etat)) appartient = true;
  else System.err.println("fin du mot avant d'atteindre un état final");
}
```

Dans le cas où l'on ne serait pas à la fin du mot, on procède alors à différentes vérifications sur le caractère courant `val = mot.charAt(i)` :

- l'appartenance du caractère à l'alphabet : `if(isInAlphabet(val)){...}`

- l'existence de la transition : `if(isInTransitions(etat, val)){...}`

Dans les cas où les fonctions booléennes sur lesquelles on a mis des conditions renvoient `false`, on affiche alors un message d'erreur correspondant et on sort de la boucle `while`.

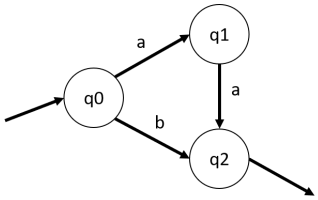
Partie 2

Représentation graphique de l'automate

Résultats des tests

> Nous avons tout d'abord effectué des tests sur les deux méthodes demandées avec des automates simples. Nous avons choisi 3 automates étudiés en TD pour pouvoir vérifier les sorties. Ensuite nous avons testés plusieurs mots pour le distributeur de café.

Automate 1



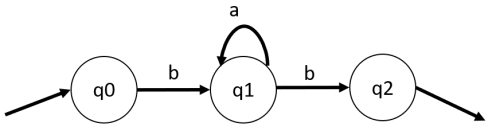
Entrées : `aa` et `ab`.

Sortie :

```
Automate 1 :
  - mot : aa
true

  - mot : ab
absence de transition a partir de l'etat courant avec le symbole lu (b)
false
```

Automate 2



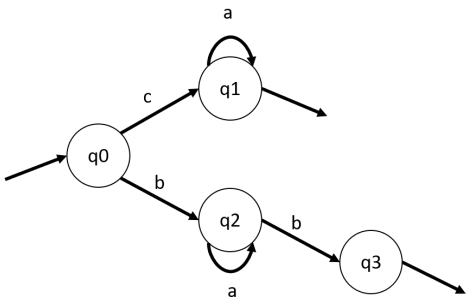
Entrées : `baaaaaaab` et `ba`.

Sortie :

```
Automate 2 :
  - mot : baaaaaaab
true

  - mot : ba
fin du mot avant d'atteindre un etat final.
Etat a la fin du mot : q1
false
```

Automate 3



Entrées : `caaa`, `baaaaaaab` et `cdaaaab`

Sorties :

```
Automate 3 :
  - mot : caaa
true

  - mot : baaaaaaab

true

  - mot : cdaaaab
Le symbole 'd' n'appartient pas a l'alphabet de l'automate.
false
```

> Ces trois séries de tests sont concluantes.

Distributeur de café

Exemples de séquences acceptées : `21Ss`, `5Svs` et `L22L1Ll`

Sortie :

```
  - mot '21Ss' : true
  - mot '5Svs' : true
```

```
  - mot 'L22L1Ll' : true
```

Exemples de séquences refusées : `2Ss`, `5S5s` et `d51l`

Sortie :

```
  -mot '2Ss' :
absence de transition a partir de l'etat courant avec le symbole lu (s)
false
  - mot '5S5s' :
absence de transition a partir de l'etat courant avec le symbole lu (5)
false
  - mot 'd51l' :
absence de transition a partir de l'etat courant avec le symbole lu (d)
false
```

Conclusion