Université Paris Diderot
L2 Informatique et Math-Info

Analyse de Données Structurées
Année 2014-2015

TP n°2

Les automates en Java

En fin de ce document, vous trouverez un appendice contenant quelques rappels sur les classes `HashSet` *et* `HashMap`*, mettant en ouvre, respectivement, les interfaces* `Set` *et* `Map`*.* [1]

1 Implémentation des automates

Fichiers à télécharger : `Automate.java, Etat.java`.

Le but de ce cette section est de simuler le fonctionnement d'un automate fini déterministe. On aura deux types d'objets correspondant aux deux fichiers à télécharger. Un état *est identifié par un nombre entier. On précisera également si l'état est terminal. Enfin, l'ensemble des transitions qui partent de cet état, sera représenté par une fonction (classe* `HashMap` *en Java) qui à une lettre associera un état (on est dans le cas déterministe).*

```
public class Etat {
    int id;
    boolean estInit;
    boolean estTerm;
    HashMap<Character,Etat> transitions
...}
```

Un automate *sera représenté par un ensemble d'états. C'est pourquoi on le définit comme une classe dérivée de* `HashSet<Etat>`*.*

```
public class Automate extends HashSet<Etat> {
    Etat init;

    public Automate(Etat e) {
      super();
      this.init = e;
      this.ajouteEtatRecursif(e);
}

...}
```

On lui rajoute un champ `init` *représentant l'état initial. De plus, quand on crée un objet de type Automate on doit fournir un état initial. À partir de cet état initial on ajoute à l'automate tous les états atteignables en utilisant la méthode* `ajouteEtatRecursif`*.*

Exercice 1 *Ajouter à la classe Etat la méthode* `Etat succ(char c)`*, qui renvoie le successeur (selon* `transitions`*) de l'état courant pour une lettre donnée.*

Exercice 2 *Ecrire dans la classe Etat une méthode* `void ajouteTransition(char c, Etat e)` *permettant d'ajouter une transition vers l'état* `e`*. Qu'est-ce qu'il se passe si* `c` *est déjà une clé de la fonction* `transitions` *?*

1. Voir `http://docs.oracle.com/javase/tutorial/collections/interfaces/set.html` et `http://docs.oracle.com/javase/tutorial/collections/interfaces/map.html` pour plus d'informations.

Exercice 3 *Ajouter la méthode* `boolean ajouteEtatSeul(Etat e)` *à la classe Automate. Elle doit renvoyer* `false` *si l'état est dèjà présent et* `true` *sinon.*

Exercice 4 *Ajouter la méthode* `boolean ajouteEtatRecursif(Etat e)` *à la classe Automate. Cette méthode tente d'ajouter l'état* `e` *en utilisant la méthode de la fonction précédente, puis si l'état n'est pas déjà présent, ajoute récursivement à l'automate les successeurs de l'état* `e`*. On utilisera la méthode* `alphabet` *de la classe* `Etat` *pour récupérer toutes les lettres pour lesquelles il y a une transition à partir d'un état.*

Exercice 5 *Écrire une méthode* `main` *(dans la classe* `Automate` *ou dans un fichier dédié) pour tester vos méthodes avec les automates suivants. Il faut d'abord créer le nombre d'états qu'il faut, ajouter les transitions, créer un automate et l'imprimer. Notez que la méthode* `toString` *des deux classes* `Automate` *et* `Etat` *a été redéfinie pour avoir une bonne représentation en chaine de caractères de leur objets.*

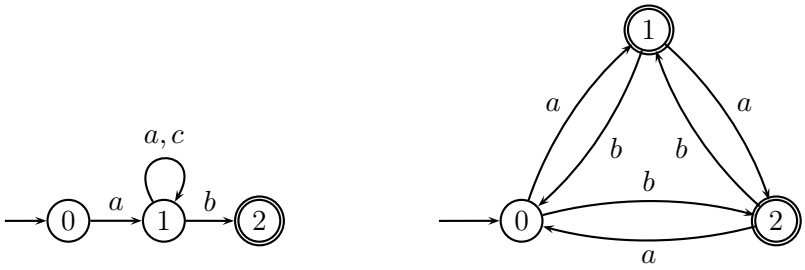

FIGURE 1 –

2 Reconnaissance

On va écrire la reconnaissance d'un mot par un automate de façon récursive d'abord dans la classe `Etat`*, puis dans la classe Automate.*

Exercice 6 *Ecrire dans la classe* `Etat` *une fonction* `boolean accepte(String s)` *qui renvoie* `true` *si à partir de l'état courant, et en lisant le mot* `s`*, on arrive dans un état terminal. En déduire la méthode* `boolean accepte(String s)` *de la classe Automate.*

3 Chercher et remplacer un motif dans un mot

Exercice 7 *Ecrire dans la classe* `Automate` *une fonction* `boolean facteur (String s)` *qui renvoie* `true` *si* `s` *contient un facteur accepté par l'automate. Ajouter d'abord une méthode adéquate dans* `Etat`*.*

Exercice 8 *Ecrire dans la classe* `Automate` *une fonction* `int[] motif (String s)` *qui renvoie les indexes du facteur reconnu par l'automate qui commence le plus tôt possible dans* `s` *et qui est le plus long possible.* `motif` *devrait renvoyer* `null` *s'il n'y a pas de facteur.*

Par example, si `a1` *est l'automate à gauche de la Figure 1, alors* `a1.motif(cacbaacb)` *renvoie le tableau* `{1,4}`*. En effet,* `"cacbaacb".substring(1,4)` *vaut* `acb`*.*

(Suggestion : utiliser la méthode `accepte` *de l'Exercice 6)*

Exercice 9 *En utilisant les méthodes précédentes, écrire dans la classe* `Automate` *une méthode* `String replace(String a_remplacer, String remplacant)` *qui remplace dans la chaine de caractères* `a_remplacer` *le facteur reconnu par la méthode* `motif` *avec la chaine de caractères* `remplancant`*. En effet, cette derniere chaine peut contenir des esperluettes* `&`*, lequelles seront remplacées par le facteur, sauf si elles sont precedées par le symbole d'échappement* `\`*.*

Par example, si `a1` *est l'automate à gauche de la Figure 1, alors l'expression :*

```
a1.replace("cacbaacb", "f&\&&f")
```

doit renvoyer `cfacb&acbfaacb`

4 Automates non-détérministes

Vous pouvez refaire tout en utilisant les automates non-détérministes. Pour cela il est utile de définir une classe `EnsEtat` *qui est une classe dérivée de* `HashSet<Etat>`*.*

5 Appendice : rappels

5.1 Set,Hashset

`Set<E>` *est une classe abstraite représentant un ensemble d'objet de type* `E` *(*`E` *pourra être n'importe quelle classe). Une implémentation possible est* `HashSet<E>`*. On pourra donc écrire* `Set<Integer> s = new HashSet<Integer>();`

Les méthodes les plus courantes pour cette classe sont :

- `boolean contains(E e)`
- `boolean isEmpty()`
- `int size()`
- `boolean add(E e)` *qui tente d'ajouter l'élément à l'ensemble, renvoie* `false` *s'il est déjà présent, et* `true` *sinon.*
- `boolean remove(Object o)`
- `boolean addAll(Set<E> s)` *qui rajoute a l'ensemble courant tous les éléments d'un ensemble* `s`*.*

A noter, il est possible de parcourir les éléments d'un `Set<E> s` *à l'aide d'une boucle comme celle ci*

```
for(E e: s){...}
```

qui peut être vue comme un équivalent pour un tableau `E[] t` *de*

```
for(int i =0; i<t.length ; i++){E e = t[i]; ....}
```

5.2 Map et HashMap

La Classe `Map<E,F>` *permet de représenter des fonctions sur un ensemble* `E` *dans un ensemble* `F`*. Comme pour* `Set`*, c'est une classe abstraite, une implémentation pratique est* `HashMap(<E,F>)`*. L'ensemble de départ* `E` *est appelé ensemble de cles et l'ensemble* `F` *ensemble de valeurs. Les associations clés-valeurs definissant la fonction sont stockées sous formes de paires. On récupère la valeur associée à une clé par la méthode* `F get(E e)` *(qui renvoie* `null` *si aucune valeur n'est associée à cette clé) et on rajoute un nouvelle association clé-valeur à l'aide de la méthode* `m.put(E e, F f)`*. Il existe plein d'autres méthodes pratiques, voir la doc.*