Implémentation d'un Automate fini déterministe en Python

Dans ce TP nous allons définir une classe `Automate` qui permet de créer un automate dont on rappelle qu'il est déterminé par :

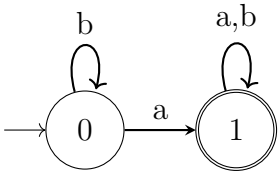
1. Un **alphabet** qui sera représenté par une chaîne de symboles;
2. Une **liste d'états**;
3. La donnée **d'un état initial**;
4. Une **liste d'états finaux**.
5. Les **transitions**, qui seront représentées par un dictionnaire dont les clefs sont les états de départ et les valeurs des couples (symbole,état cible) .

Le constructeur de la classe vous est donné dans le fichier `myautomate.py` disponible sur l'ENT.

Exercice 1.

1. Définir la méthode `ajoute_etat` qui rajoute un état après avoir vérifié qu'il n'existe pas déjà.
2. Définir la méthode `ajoute_transition` qui ajoute une transition après avoir vérifié que les états et le symbole sont valides et que la transition n'existe pas déjà.
   (On pourra introduire les fonctions `valide_symbole` *qui vérifie si le symbole est dans l'alphabet et* `destination_etat` *qui vérifie si la transition n'existe pas déjà).*

L'automate



peut donc être défini par la suite d'instructions :

```
import myautomate as au

a=au.Automate("ab")
a.ajoute_etat("0")
a.ajoute_etat("1", True)
a.initial = "0"
a.ajoute_transition("0", "a", "1")
a.ajoute_transition("0", "b", "0")
a.ajoute_transition("1", "a", "1")
a.ajoute_transition("1", "b", "1")
```

Exercice 2. 1. Vérifiez que l'instruction `print(a)` vous retourne bien :

```
Automate fini :
    - alphabet   : 'ab'
    - état initial       : 0
    - états finaux     : ['1']
    - nombre d'états : 2
    - transitions :
        - depuis (0):
            --(a)--> (1)
            --(b)--> (0)
        - depuis (1):
            --(a)--> (1)
            --(b)--> (1)
```

2. Définir ensuite les méthodes `supprime_etat`, `supprime_transition` et `supprime_symbole`.

Reconnaissance d'un langage

Il s'agit maintenant d'implémenter une méthode de reconnaissance d'un mot donné par un automate défini au préalable.
L'algorithme démarre à l'état initial, et pour chaque symbole du mot à reconnaître, on se rend à l'état correspondant à la transition pour ce symbole et l'état courant. Si on rencontre un symbole pour lequel il n'existe pas de transition, le mot est refusé.

Si à l'inverse on arrive à la fin du mot, il suffit de regarder si le dernier état atteint est accepteur. Si c'est le cas le mot est accepté, sinon il est refusé.

Exercice 3. 1. Coder une méthode `test_mot(self,mot)` qui teste si `mot` est accepté par l'automate `self`.

2. Modifier `test_mot` pour ajouter des messages d'erreurs et des informations sur l'état de l'exécution (symbole n'appartenant pas à l'alphabet, transition impossible, mot reconnu à un état non final, mot reconne à un état final).

Vous pourrez faire des tests sur les mots **aaba, abaa, bababa** avec l'automate suivant :