SAE 2-02 : Exploration algorithmique d'un problème

Objectif : L'objectif est d'implémenter les différents algorithmes vus en cours de graphes. Le langage de programmation utilisé est python3.
Le projet est divisé en différentes parties de difficultés croissantes. Vous devrez les aborder dans l'ordre indiqué. Dans les premières parties, le découpage du code en fonctions est détaillé précisément. Dans la suite et notamment dans la dernière partie vous devrez vous-même définir vos fonctions intermédiaires.
Pensez à commenter votre code et à le tester de la façon la plus régulière et complète possible.

Dans tout le projet, un mot sera codé par une chaîne de caractères, un langage par une liste de mots et un automate sera représenté sous forme de dictionnaire. L'alphabet, l'ensemble des états, l'ensemble des états initiaux et l'ensemble des états terminaux seront codés par des listes et l'ensemble des transitions par une liste de listes.
Par exemple l'automate suivant :

```
auto ={"alphabet":['a','b'],"etats": [1,2,3,4],
  "transitions":[[1,'a',2],[2,'a',2],[2,'b',3],[3,'a',4]],
  "I":[1],"F":[4]}
```

accepte le langage a^+ba.
Merci de respecter ces notations dans la définition des automates.

1 Mots, langages et automates...

Dans cette première partie, vous allez écrire quelques fonctions de base qui vous permettront de vous familiariser avec les structures de données associées aux mots, langages et automates.

1.1 Mots

- 1.1.1. Définir une fonction `pref` qui étant donné un mot u passé en paramètre renvoie la liste des préfixes de u.

```
print(pref(''coucou''))
['', 'c', 'co', 'cou', 'couc', 'couco', 'coucou']
```

- 1.1.2. Définir une fonction `suf` qui étant donné un mot u passé en paramètre renvoie la liste des suffixes de u.

```
print(suf(''coucou''))
['coucou', 'oucou', 'ucou', 'cou', 'ou', 'u', '']
```

- 1.1.3. Définir une fonction `fact` qui étant donné un mot u passé en paramètre renvoie la liste sans doublons des facteurs de u.

```
print(fact(''coucou''))
['', 'c', 'co', 'cou', 'couc', 'couco', 'coucou', 'o', 'ou',
  'ouc', 'ouco', 'oucou', 'u', 'uc', 'uco', 'ucou']
```

– 1.1.4. Définir une fonction `miroir` qui étant donné un mot u passé en paramètre renvoie le mot miroir de u.

```
print(miroir(''coucou''))
uocuoc
```

1.2 Langages

– 1.2.1. Définir une fonction `concatene` qui étant donnés deux langages L1 et L2 renvoie le produit de concaténation (sans doublons) de L1 et L2.

```
L1=['aa','ab','ba','bb']
L2=['a', 'b', '']
print( concatene(L1,L2))
['aaa', 'aab', 'aa', 'aba', 'abb', 'ab', 'baa', 'bab', 'ba', 'bba', 'bbb', 'bb']
```

– 1.2.2. Définir une fonction `puis` qui étant donnés un langage L et un entier n renvoie le langage L^n (sans doublons).

```
L1=['aa','ab','ba','bb']
print(puis(L1,2))
['aaaa', 'aaab', 'aaba', 'aabb', 'abaa', 'abab', 'abba', 'abbb', 'baaa', 'baab',
  'baba', 'babb', 'bbaa', 'bbab', 'bbba', 'bbbb']
```

– 1.2.3. Pourquoi ne peut-on pas faire de fonction calculant l'étoile d'un langage ?

– 1.2.4. Définir une fonction `tousmots` qui étant donné un alphabet A passé en paramètre renvoie la liste de tous les mots de A^* de longueur inférieure à n.

```
print(tousmots(['a','b'],3))
['a', 'b', 'aa', 'ab', 'ba', 'bb', 'aaa', 'aab',
  'aba', 'abb', 'baa', 'bab', 'bba', 'bbb', '']
```

1.3 Automates

– 1.3.1. Définir une fonction `defauto` qui permet de faire la saisie d'un automate (sans doublon).

– 1.3.2. Définir une fonction `lirelettre` qui étant donnés en paramètres une liste de transitions T, une liste d'états E et une lettre a, renvoie la liste des états dans lesquels on peut arriver en partant d'un état de E et en lisant la lettre a.

```
print(lirelettre(auto["transitions"],auto["etats"],'a'))
[2, 4]
```

– 1.3.3. Définir une fonction `liremot` qui étant donnés en paramètres une liste de transitions T, une liste d'états E et un mot m, renvoie la liste des états dans lesquels on peut arriver en partant d'un état de E et en lisant le mot m.

```
print(liremot(auto["transitions"],auto["etats"],'aba'))
[4]
```

- 1.3.4. Définir une fonction `accepte` qui prend en paramètres un automate et un mot m et renvoie True si le mot m est accepté par l'automate.
- 1.3.5. Définir une fonction `langage_accept` qui prend en paramètres un automate et un entier n et renvoie la liste des mots de longueur inférieure à n acceptés par l'automate.
- 1.3.6. Pourquoi ne peut-on pas faire une fonction qui renvoie le langage accepté par un automate ?

2 Déterminisation.

Dans cette partie vous allez implémenter l'algorithme de déterminisation.
Voici quelques exemples simples d'automates qui permettront d'illustrer les énoncés des parties 2 et 3 mais vous devez tester vos fonctions avec vos propres exemples d'automates.

```
auto0 ={"alphabet":['a','b'],"etats": [0,1,2,3],
  "transitions":[[0,'a',1],[1,'a',1],[1,'b',2],[2,'a',3]], "I":[0],"F":[3]}


auto1 ={"alphabet":['a','b'],"etats": [0,1],
  "transitions":[[0,'a',0],[0,'b',1],[1,'b',1],[1,'a',1]], "I":[0],"F":[1]}


auto2={"alphabet":['a','b'],"etats": [0,1],
  "transitions":[[0,'a',0],[0,'a',1],[1,'b',1],[1,'a',1]], "I":[0],"F":[1]}
```

- 2.1. Définir une fonction `deterministe` qui étant donné un automate passé en paramètre renvoie True s'il est déterministe et False sinon.

```
print(deterministe(auto0))
True
print(deterministe(auto2))
False
```

- 2.2. Définir une fonction `determinise` qui déterminise l'automate passé en paramètre.

```
print(determinise(auto2))
'alphabet': ['a', 'b'], 'I': [[0]],
'transitions': [[[0], 'a', [0, 1]], [[0, 1], 'a', [0, 1]], [[0, 1], 'b', [1]],
  [[1], 'a', [1]], [[1], 'b', [1]]],
'etats': [[0], [0, 1], [1]], 'F': [[0, 1], [1]]}
```

- 2.3. Définir une fonction `renommage` qui étant donné un automate passé en paramètre renomme ses états avec les premiers entiers.

```
print(renommage(determinise(auto2)))
{'alphabet': ['a', 'b'], 'etats': [0, 1, 2],'transitions': [[0, 'a', 1], [1, 'a', 1],
    [1, 'b', 2], [2, 'a', 2], [2, 'b', 2]],'I': [0], 'F': [1, 2]}
```

Vous pourrez pour cela définir des fonctions intermédiaires qui remplacent un état par un autre dans un ensemble d'états et dans un ensemble de transitions.

3 Complémentation.

Dans cette partie, vous allez implémenter la complémentation. Prennez le temps de tester vos fonctions avec vos propres exemples d'automates.

```
    auto3 ={"alphabet":['a','b'],"etats": [0,1,2,],
 "transitions":[[0,'a',1],[0,'a',0],[1,'b',2],[1,'b',1]], "I":[0],"F":[2]}
```

- 3.1. Définir une fonction `complet` qui étant donné un automate passé en paramètre renvoie True s'il est complet et False sinon.

```
print(complet(auto0))
False
print(complet(auto1))
True
```

- 3.2. Définir une fonction `complete` qui complète l'automate passé en paramètre en ajoutant un état puits.

```
print(complete(auto0))
{'alphabet': ['a', 'b'], 'etats': [0, 1, 2, 3, 4],
  'transitions': [[0, 'a', 1], [1, 'a', 1], [1, 'b', 2], [2, 'a', 3],
    [0, 'b', 4], [2, 'b', 4], [3, 'a', 4], [3, 'b', 4], [4, 'a', 4], [4, 'b', 4]],
  'I': [0], 'F': [3]}
```

- 3.3. Définir une fonction `complement` qui étant donné un automate passé en paramètre acceptant un langage L renvoie un automate acceptant le complement $\bar{L}$. N'hésitez pas à utiliser les fonctions précédemment définies.

```
print(complement(auto3))
{'alphabet': ['a', 'b'], 'etats': [0, 1, 2, 3],
  'transitions': [[0, 'a', 1], [1, 'a', 1], [1, 'b', 2], [2, 'b', 2],
    [0, 'b', 3], [2, 'a', 3], [3, 'a', 3], [3, 'b', 3]], 'I': [0], 'F': [0, 1, 3]}
```

4 Automate produit.

Dans cette partie, vous allez implémenter l'intersection et la différence via l'automate produit.

```
auto4 ={"alphabet":['a','b'],"etats": [0,1,2,],
    "transitions":[[0,'a',1],[1,'b',2],[2,'b',2],[2,'a',2]], "I":[0],"F":[2]}

auto5 ={"alphabet":['a','b'],"etats": [0,1,2],
  "transitions":[[0,'a',0],[0,'b',1],[1,'a',1],[1,'b',2],[2,'a',2],[2,'b',0]],
  "I":[0],"F":[0,1]}
```

- 4.1. Définir une fonction `inter` qui étant donnés deux automates **déterministes** passés en paramètres acceptant respectivement les langages L_1 et L_2, renvoie l'automate produit acceptant l'intersection $L_1 \cap L_2$.

```
print(inter(auto4,auto5))
'alphabet': ['a', 'b'], 'I': [(0, 0)], 'transitions': [[(0, 0), 'a', (1, 0)],
  [(1, 0), 'b', (2, 1)], [(2, 1), 'a', (2, 1)], [(2, 1), 'b', (2, 2)],
```

```
  [(2, 2), 'a', (2, 2)], [(2, 2), 'b', (2, 0)], [(2, 0), 'a', (2, 0)],
  [(2, 0), 'b', (2, 1)]],
'etats': [(0, 0), (1, 0), (2, 1), (2, 2), (2, 0)], 'F': [(2, 0), (2, 1)]}

 print(renommage(inter(auto4,auto5)))
 {'alphabet': ['a', 'b'], 'etats': [0, 1, 2, 3, 4], 'transitions': [[0, 'a', 1],
     [1, 'b', 2], [2, 'a', 2], [2, 'b', 3], [3, 'a', 3], [3, 'b', 4], [4, 'a', 4],
     [4, 'b', 2]], 'I': [0], 'F': [4, 2]}
```

– 4.2. Définir une fonction `difference` qui étant donnés deux automates **déterministes** passés en paramètres acceptant respectivement les langages L_1 et L_2, renvoie l'automate produit acceptant la difference $L_1 \setminus L_2$. Ne pas oublier de compléter les automates avant d'en faire le produit.

```
print(difference(auto4,auto5))
{'alphabet': ['a', 'b'], 'I': [(0, 0)], 'transitions': [[(0, 0), 'a', (1, 0)],
    [(0, 0), 'b', (3, 1)], [(3, 1), 'a', (3, 1)], [(3, 1), 'b', (3, 2)],
    [(3, 2), 'a', (3, 2)], [(3, 2), 'b', (3, 0)], [(3, 0), 'a', (3, 0)],
    [(3, 0), 'b',(3, 1)], [(1, 0), 'a', (3, 0)], [(1, 0), 'b', (2, 1)],
    [(2, 1), 'a', (2, 1)], [(2, 1), 'b', (2, 2)], [(2, 2), 'a', (2, 2)],
    [(2, 2), 'b', (2, 0)], [(2, 0), 'a', (2, 0)], [(2, 0), 'b', (2, 1)]],
  'etats': [(0, 0), (3, 1), (3, 2), (3, 0), (1, 0), (2, 1), (2, 2), (2, 0)],
  'F': [(2, 2)]}

print(renommage(difference(auto4,auto5)))
{'alphabet': ['a', 'b'], 'etats': [0, 1, 2, 3, 4, 5, 6, 7],
  'transitions': [[0, 'a', 4], [0, 'b', 1], [1, 'a', 1], [1, 'b', 2],
    [2, 'a', 2], [2, 'b', 3], [3, 'a', 3], [3, 'b', 1], [4, 'a', 3],
    [4, 'b', 5], [5, 'a', 5], [5, 'b', 6], [6, 'a', 6], [6, 'b', 7],
    [7, 'a', 7], [7, 'b', 5]], 'I': [0], 'F': [6]}
```

5 Propriétés de fermeture.

Dans cet partie, vous pourrez utiliser les constructions de l'exercice 3 du TD 4.

– 5.1. Définir une fonction `prefixe` qui étant donné un automate **émondé** acceptant L renvoie un automate acceptant l'ensemble des préfixes des mots de L.

– 5.2. Définir une fonction `suffixe` qui étant donné un automate **émondé** acceptant L renvoie un automate acceptant l'ensemble des suffixes des mots de L.

– 5.3. Définir une fonction `facteur` qui étant donné un automate **émondé** acceptant L renvoie un automate acceptant l'ensemble des facteurs des mots de L.

– 5.4. Définir une fonction `miroir` qui étant donné un automate **émondé** acceptant L renvoie un automate acceptant l'ensemble des miroirs des mots de L.

6 Minimisation.

Dans cette partie, vous allez implémenter l'algorithme de Moore.
Définir une fonction `minimise` qui étant donné un automate complet et déterministe renvoie

l'automate minimisé.

```
auto6 ={"alphabet":['a','b'],"etats": [0,1,2,3,4,5],
  "transitions":[[0,'a',4],[0,'b',3],[1,'a',5],[1,'b',5],[2,'a',5],[2,'b',2],[3,'a',1],[3,'b',0],
    [4,'a',1],[4,'b',2],[5,'a',2],[5,'b',5]],
  "I":[0],"F":[0,1,2,5]}

print(minimise(auto6))
'alphabet': ['a', 'b'], 'etats': [[0], [1, 2, 5], [3], [4]], 'I': [[0]],
'transitions': [[[0], 'a', [4]], [[0], 'b', [3]], [[1, 2, 5], 'a', [1, 2, 5]],
  [[1, 2, 5], 'b', [1, 2, 5]], [[3], 'a', [1, 2, 5]], [[3], 'b', [0]],
  [[4], 'a', [1, 2, 5]], [[4], 'b', [1, 2, 5]]], 'F': [[0], [1, 2, 5]]}

print(renommage(minimise(auto6)))
{'alphabet': ['a', 'b'], 'etats': [0, 1, 2, 3], 'transitions': [[0, 'a', 3],
    [0, 'b', 2], [1, 'a', 1], [1, 'b', 1], [2, 'a', 1], [2, 'b', 0], [3, 'a', 1],
    [3, 'b', 1]], 'I': [0], 'F': [0, 1]}
```

Vous devrez découper votre code en plusieurs fonctions intermédiaires que vous commenterez soigneusement.