Grammaire & Langage - Projet

TAYLOR MATT / SOCHAJ YOANN

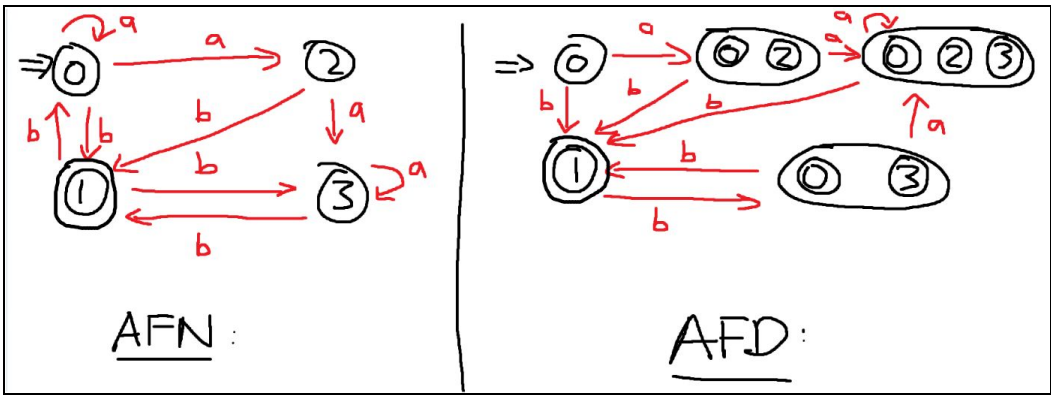
16.01.2021

Sommaire:

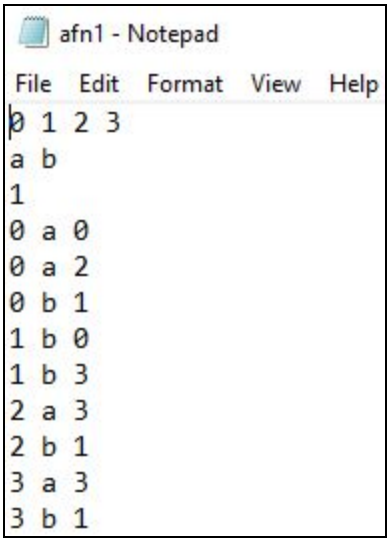
I. Algorithme de transformation AFN -> AFD

Pour ces deux exercices nous avons choisi d'utiliser **Python** comme langage de programmation. Nos structures de données sont simplement des **listes**.

Avant de commencer l'**implémentation** nous avons refait quelques exemples **à la main** pour remettre les idées au clair de comment nous allions procéder. Voici un des ces exemples:



Concernant notre **fichier texte** contenant l'AFN nous avons choisi la syntaxe suivante: (*afn1.txt* correspond a l'exemple au-dessus).



La **première** ligne contient **tous les états**, la **deuxième** l'**alphabet** et la **troisième** les **états terminaux,** le **reste** indique la **disposition** de des **transitions** dans l'**AFN**: état de départ, transition, état d'arrivée. *ex: 0 a 0*

Voici une **exécution** de notre programme (avec l'AFN au dessus (*afn1.txt*)):

```
E:\VMshared\UNIVERSITY\L3\gram & lang\projet\v1\afn2afd\v1>ex1.py
Enter file name: (without .txt)
afn1
Etats: ['0', '1', '2', '3']
Alphabet: ['a', 'b']
Etat(s) Terminaux: ['1']
AFN: [[0, 'a', 0], [0, 'a', 2], [0, 'b', 1], [1, 'b', 0], [1, 'b', 3], [2, 'a', 3], [2, 'b', 1], [3, 'a', 3],
 [3, 'b', 1]]

******* RESULTAT *******

Etats de l'AFD: [[0], [0, 2], [1], [0, 2, 3], [0, 3]]
Etat(s) terminaux de l'AFD [[1]]
From [0] with a -> [0, 2]
From [0, 2] with a -> [0, 2, 3]
From [0, 2, 3] with a -> [0, 2, 3]
From [0, 3] with a -> [0, 2, 3]
From [0] with b -> [1]
From [0, 2] with b -> [1]
From [1] with b -> [0, 3]
From [0, 2, 3] with b -> [1]
From [0, 3] with b -> [1]
```

On affiche ce qu'on a lu du fichier texte (Etats, Alphabet, État(s) Terminaux et l'AFN en forme de liste).

Ensuite on affiche la liste des **états** de l'**AFD** en précisant lesquels sont terminaux suivi de toutes ses **transitions** pour **chaque caractère** de l'alphabet.

Le principe de l'algorithme est simple, une des fonctions **principales** s'appelle ***findPath()***:

```python
#fonction pour trouver la destination d'une transition
#en specifiant le depart et le charactere
def findPath(afn, list, char, printV):
    res  = []
    for i in range(len(afn)):
        for state in list:
            if(afn[i][0] == state and afn[i][1] == char):
                if(afn[i][2] not in res):
                    res.append(afn[i][2])
    if(res != [] and printV):
        print("From", list, "with", char, "->", res)
    return res
```

Nous utilisons cette fonction au début sur l'**état 0** avec "**a**" puis "**b**", les retours de cette fonction sont stockés respectivement dans **listA** et **listB** ensuite nous rappelons cette fonction sur ces listes jusqu'à obtenir toutes les transitions.

II. Algorithme de minimisation

Pour résoudre cet exercice nous avons réutilisé la plupart de nos fonctions créées pour l'exercice 1 en apportant **quelques modifications**.

findPath() est un de ces exemples, nous avons du rajouter quelques vérifications:

```
#fonction pour trouver la destination d'une transition en specifiant le depart et le character
def findPath(afd, list, char, printV, newListEtatFinal):
    res  = []
    for state in list:
        for i in range(len(afd)):
            if(afd[i][0] == state and afd[i][1] == char):
                if(afd[i][2] not in res and len(getSublist(newListEtatFinal, afd[i][2])) < 2):
                    res.append(afd[i][2])
                else:
                    ans = getSublist(newListEtatFinal, afd[i][2]) #on utilise getSublist() pou
                    if(ans not in res and ans[0] not in res):
                        res.append(ans)
    if(res != [] and printV):
        print("From", list, "with", char, "->", res) #on affiche les informations pour etre lu
    return res
```

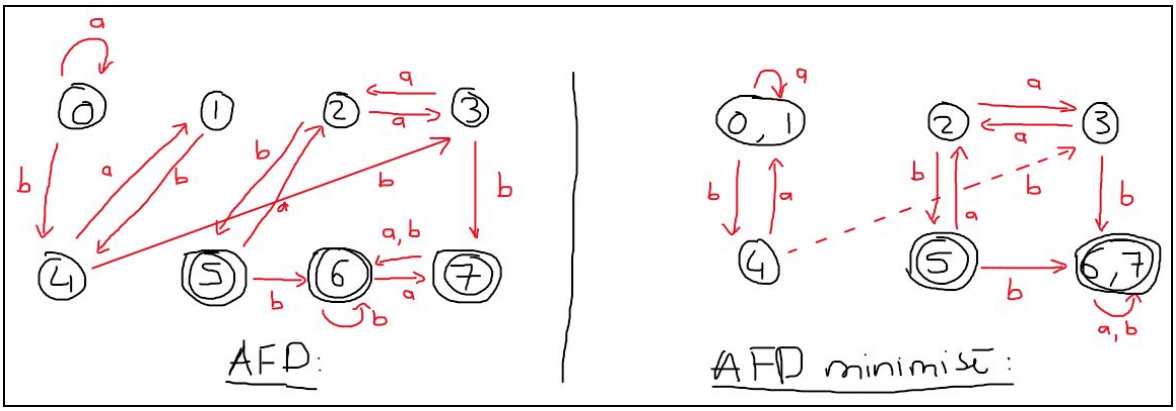
Une nouvelle fonction que nous avons créé est: ***destination()*** elle prend en paramètre un état et retourne le(s) état(s) vers lesquels il "va" avec "a" et "b" (tout l'alphabet).

```
#fonction qui prend en parametre un etat et retourne le(s
def destination(state):
    a, b, c, d = readFile(fileName)
    splitL = customArr(fileName)
    newC = iter(c)
    afd = [list(islice(newC, size)) for size in splitL ]

    listDestination = [] #declare an empty list which wil
    for sublist in afd:
        if sublist[0] == state:
            listDestination.append(sublist[2]) #on rajout
    return listDestination #return our result
```

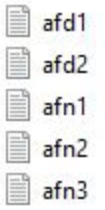
Voici une **exécution** de ce programme:

```
E:\VMshared\UNIVERSITY\L3\gram & lang\projet\v1\afn2afd\v1>ex2.py
Enter file name: (without .txt)
afd1
Etats: ['0', '1', '2', '3', '4', '5', '6', '7']
Transitions: ['a', 'b']
Etat(s) terminaux: ['5', '6', '7']
Etat(s) non terminaux: ['0', '1', '2', '3', '4']
AFD: [[0, 'a', 0], [0, 'b', 4], [1, 'a', 1], [1, 'b', 4], [2, 'a', 3], [2, 'b', 5], [3, 'a', 2], [3, 'b', 7],
 [4, 'a', 1], [4, 'b', 3], [5, 'a', 2], [5, 'b', 6], [6, 'a', 7], [6, 'b', 6], [7, 'a', 6], [7, 'b', 6]]

******* RESULTAT *******

Etats de l'AFD minimise: [[5], [2], [3], [4], [6, 7], [0, 1]]
Etats terminaux de l'AFD minimise [[5], [6, 7]]
From [5] with a -> [2]
From [5] with b -> [[6, 7]]
From [2] with a -> [3]
From [2] with b -> [5]
From [3] with a -> [2]
From [3] with b -> [[6, 7]]
From [4] with a -> [[0, 1]]
From [4] with b -> [3]
From [6, 7] with a -> [[6, 7]]
From [6, 7] with b -> [[6, 7]]
From [0, 1] with a -> [[0, 1]]
From [0, 1] with b -> [4]
```

basé sur cet **AFD**: (*afd1.txt*)



On récapitule tout d'abord les informations relatives au fichier lu comme pour l'exercice 1, ensuite on affiche les **états** de **l'AFD minimisé** suivi des **états** qui sont **terminaux** et enfin on affiche les différentes **transitions** qui existent pour **faciliter** la **visualisation** de cet **AFD minimise** par **l'utilisateur**.

Notre archive contient plusieurs fichiers textes 

ou **vous pourrez tester** nos différents programmes (il est évident qu'il faut utiliser les fichiers **afn** pour l'**exercice 1** et **afd** pour l'**exercice 2**).

Tous les résultats et les schémas de ces AFN / AFD sont dans le fichier ***results***.