Gestion d'automates finis en Python 3.x

Rappel : le travail effectué est individuel. En accord avec le guide de l'étudiant, toute utilisation de code externe (autre étudiant, internet, ...) sera sanctionné d'un zéro pour le module concerné ainsi que d'un conseil de discipline.

Le but de ce projet est l'implémentation des automates finis et des algorithmes relatifs à ceux-ci vus en cours. Pour des raisons déjà expliquées, il sera obligatoire d'utiliser le langage Python 3.x.

1 Modalités de rendu

Le projet sera à rendre le samedi 3 avril 2021 à 23h55, sur le dépôt Moodle créé pour l'occasion (**attention :** merci de procéder au rendu sur la page du module relative à votre promotion, 3A-CFA ou 3A initiaux). L'archive de votre projet devra être sous la forme VOTRENOM.zip.

2 Lecture d'un automate depuis un fichier

Les automates seront pour la plupart des tests stockés dans des fichiers. Un fichier contiendra donc l'alphabet, le nombre d'états avec leurs caractérisations (initial, acceptant) ainsi que les transitions. Nous travaillerons donc avec le format de fichier suivant :

- la première ligne contient l'alphabet (par exemple : abcd),
- la seconde ligne contient un entier N représentant le nombre d'états de l'automate,
- la troisième ligne contient la liste des états initiaux (séparés par des espaces),
- la quatrième ligne contient la liste des états acceptants (séparés par des espaces),

Ensuite, chaque ligne décrit une transition sous la forme d'un triplet : état source, état destination et étiquette. Prenons comme exemple l'automate suivant :

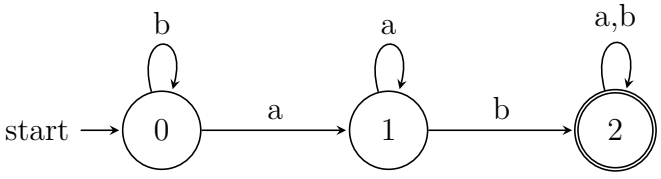

FIGURE 1 – Automate A_2

On remarquera que les états sont numérotés par des identifiants, ceux-ci commençant à 0. Si on suit notre format de fichier, le fichier caractérisant cet automate sera le suivant :

```
1  ab
2  3
3  0
4  2
5  0 0 b
6  0 1 a
7  1 1 a
8  1 2 b
9  2 2 a
10 2 2 b
```

Le fichier se lit de la manière suivante. La ligne 1 contient l'alphabet $\{a, b\}$, et l'automate est composé de 3 états (ligne 2). L'état 0 est initial (ligne 3) et l'état 2 est acceptant (ligne 4). Ensuite, on énumère les transitions :

- de 0 vers 0 avec l'étiquette 'b',
- de 0 vers 1 avec l'étiquette 'a',
- de 1 vers 1 avec l'étiquette 'a',
- ...

3 Travail à réaliser

Pour ce projet, vous devrez réaliser un exécutable dont la fonction différera selon les paramètres donnés en ligne de commande. Le but est d'implémenter un maximum des algorithmes vus en cours, et chacune des fonctionnalités décrites dans le sujet est classée selon sa difficulté d'implémentation. Il est expressément demandé d'utiliser une implémentation de graphe avec une classe représentant un état, une classe représentant une transition (pointeur vers état et étiquette), et une classe représentant un automate (liste d'états et alphabet).

3.1 Prise en charge des AFD

La première fonctionnalité à mettre en place est la simulation de fonctionnement d'un AFD. Une fois l'automate chargé depuis un fichier donné en ligne de commande, le programme sera chargé de tester une liste de mots, également présente dans un fichier, et, pour chaque mot, donner le résultat de l'exécution de l'automate avec 0 lorsque le mot n'a pas été accepté, et 1 dans le cas inverse. La syntaxe de la ligne de commande devra obligatoirement être la suivante :

```
1  python3 main.py 0 afd1.txt mots1.txt sortie1.txt
```

où main.py est le nom de votre fichier principal, 0 le mode (lecture d'une liste de mots par un automate), afd1.txt le fichier décrivant l'automate, mots1.txt le fichier contenant la liste des mots à tester et enfin sortie1.txt un fichier qui sera créer par le programme et contiendra le résultat de l'exécution. Reprenons l'automate donné en exemple ci-dessus. Si on lui passe le fichier de mots suivant :

```
1  baa
2  aababa
3  aaa
4  bab
```

On remarque que les mots 1 et 3 ne sont pas acceptés par l'automate, alors que les mots 2 et 4 le sont. On doit donc obtenir le fichier de sortie :

```
0
1
0
1
```

Vous pourrez tester votre programme avec les fichiers d'exemple disponibles sur Moodle.

3.2 Prise en charge des AFN

Vous devez désormais modifier votre programme afin qu'il prenne en charge les automates finis non déterministes. En fonction de l'implémentation que vous aurez choisie, il vous sera nécessaire de modifier (1) la structure ou la classe représentant un automate et (2) la fonction testant l'acceptation d'un mot par un automate. Si le premier point est contextuel à votre implémentation (non pertinent dans le cas d'une implémentation avec des nœuds contenant des listes de transitions), le deuxième point vous concerne très probablement. L'exécution du programme doit se faire de la même manière que dans la section précédent :

```
python3 main.py afn1.txt mots1.txt sortie1.txt
```

et doit fonctionner indifféremment du déterminisme ou non de l'automate passé en paramètre.

3.3 Minimisation d'un AFN

Il vous est ici demandé de procéder à l'implémentation de l'algorithme de Moore afin de procéder à la minimisation d'un automate passé en paramètre. La ligne de commande :

```
python3 main.py 1 afd2.txt afd3.txt
```

procédera donc à :

- la lecture de l'automate fini déterministe décrit dans le fichier afd2.txt,
- l'exécution de l'algorithme de Moore sur cet automate, générant un nouvel automate fini déterministe équivalent mais minimal,
- l'écriture de l'automate fini déterministe dans le fichier af3.txt.

Lors de l'évaluation, l'automate fini déterministe généré par votre programme sera testé par un programme extérieur qui vérifiera qu'il s'agit d'un AFD minimal, et qui l'exécutera sur un grand nombre de mots, afin de valider son équivalence à l'automate fini déterministe source.

3.4 Déterminisation d'un AFN

Maintenant que votre programme prend en charge les automates finis non déterministes, il vous est demandé de procéder à l'implémentation de la déterminisation d'un automate passé en paramètre. La ligne de commande :

```
python3 main.py 2 afn2.txt afd2.txt
```

procédera donc à :

- la lecture de l'automate fini décrit dans le fichier afn2.txt,
- l'exécution de l'algorithme de déterminisation sur cet automate, générant un nouvel automate fini déterministe équivalent,
- l'écriture de l'automate fini déterministe dans le fichier afd2.txt.

Lors de l'évaluation, l'automate fini déterministe généré par votre programme sera testé par un programme extérieur qui vérifiera qu'il s'agit d'un AFD, et qui l'exécutera sur un grand nombre de mots, afin de valider son équivalence à l'automate fini non-déterministe source.

3.5 Prise en charge des ϵ-transitions

Nous abordons ici la partie la plus complexe du projet : la prise en compte des ϵ-transitions dans un automate. Attention : il n'est pas demandé d'assurer la lecture d'un mot par un ϵ-automate, mais de procéder à la transformation de celui-ci en un automate fini sans ϵ-transitions.

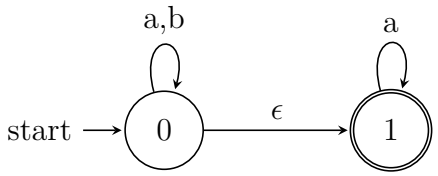

FIGURE 2 – Automate A_3

Les ϵ-transitions seront notées avec l'étiquette '#' dans les fichiers caractérisant les automates. Dans notre exemple, on obtient le fichier suivant :

```
1  ab
2  2
3  0
4  1
5  0 0 a
6  0 0 b
7  0 1 #
8  1 1 a
```

Point important pour la suite : pour éviter le bouclage de votre algorithme, vous devrez interdire la création d'une ϵ-transition d'un état sur lui-même : lors de l'ajout d'une nouvelle transition, si l'état source est le même que l'état destination, et que l'étiquette est '#', alors ignorez la création de la transition. On travaillera sur un algorithme à deux phases : on procède à la fusion des états équivalents, permettant la suppression des ϵ-cycles, puis à la suppression des ϵ-transitions telle qu'étudiée en cours.

3.5.a Fusion d'états équivalents

Deux états i et j sont dits équivalents si l'on peut passer de i à j via une chaîne d'ϵ-transitions et inversement.

On commence par procéder à la redirection des transitions ciblant i pour qu'elles ciblent j. Si i est initial (resp. acceptant), alors j devient initial (resp. acceptant). Ensuite,

on ajoute toutes les transitions partant de i à l'état j. Ainsi, pour une transition (i, k, s), avec k un état et s une étiquette, on créé la transition (j, k, s), à la condition que celle-ci n'existe pas déjà, et qu'il ne s'agisse pas d'une ϵ-transition de j vers j. On copie également toute transition du type (i, i, s) en (j, j, s).

Prenons l'exemple suivant :

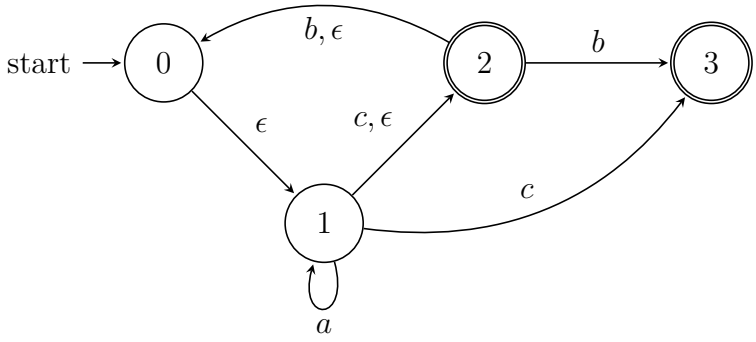

FIGURE 3 – Automate A_4

Il existe un ϵ-cycle comprenant les états 0, 1 et 2 : on peut passer de l'un de ces états à un autre via une chaîne d'ϵ-transitions. Comme il existe un chemin d'ϵ-transitions entre 0 et 1, et entre 1 et 0, alors on peut fusionner ces deux états :

- la transition b, ϵ partant de 2 vers 0 est redirigée vers 1,
- comme 0 est initial, 1 devient initial,
- la transition ϵ partant de 0 vers 1 part désormais de 1. Cependant, il s'agit désormais d'une ϵ-transition partant de 1 vers lui-même, on décide donc de la supprimer.

On obtient l'automate suivant :

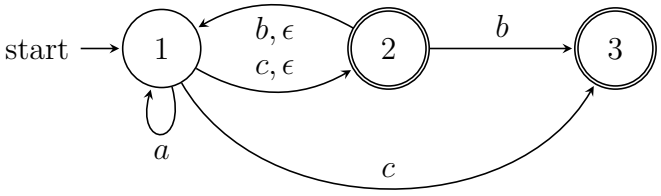

FIGURE 4 – Automate A_4

On procède de la même manière pour fusionner les états 1 et 2 :

- la transition b, ϵ partant de 2 vers 1 est redirigée vers 2, en supprimant l'ϵ-transition de 2 vers 2 générée par la manœuvre : on obtient une boucle sur 2 avec l'étiquette b,
- la transition a bouclant sur 1 est dupliquée sur 2,
- la transition c, ϵ partant de 1 vers 2 part désormais de 2, en supprimant l'ϵ-transition de 2 vers 2 générée par la manœuvre : on obtient une boucle sur 2 avec l'étiquette c,
- la transition c partant de 1 vers 3 part désormais de 2,
- comme 1 est initial, 2 devient initial.

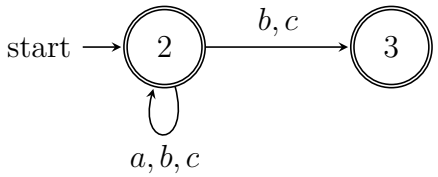

FIGURE 5 – Automate A_4

3.5.b Suppression des ϵ-transitions

Pour procéder à la suppression des ϵ-transitions, vous devrez implémenter le processus suivant. Tant qu'il reste des ϵ-transitions, on prend le premier état i émetteur d'une ϵ-transition vers j :

- si i est initial, alors j devient initial,
- si j est acceptant, alors i devient acceptant,
- pour toute transition (k, i) avec une étiquette s, on créé une transition (k, j) avec une étiquette s,
- enfin, on détruit l'ϵ-transition.

```
1 python3 main.py 3 eafn3.txt afn3.txt
```

Lors de l'évaluation, l'automate fini déterministe généré par votre programme sera testé par un programme extérieur qui vérifiera qu'il s'agit d'un AFN sans ϵ-transitions, et qui l'exécutera sur un grand nombre de mots, afin de valider son équivalence à l'ϵ-automate source.

3.6 Bonus

Pour les bonus, vous pourrez implémenter toutes fonctionnalités en rapport avec le cours et le projet. Le nombre de points rapporté par un bonus est relatif à (1) la difficulté d'implémentation et (2) l'intérêt vis à vis du projet. On pourrait citer :

- l'implémentation de Thomson : on passe en paramètre une expression régulière, et un automate fini est généré,
- l'implémentation de BMC : on procède au calcul de l'expression régulière d'un automate,
- la réalisation d'une interface graphique permettant la visualisation d'un automate et le déroulement de test d'acceptation d'un mot,
- prise en compte des grammaires,
- prise en compte des automates à pile.