Théorie des langages réguliers : TP 3

Adrien Pommellet

1er septembre 2021

Téléchargez au préalable le fichier `thlr_tp3.py` associé à ce TP.

Le fichier associé `display_automaton.py` contient des routines d'affichage utiles mais nécessite une installation préalable de la bibliothèque `graphviz`.

1 Retour sur le TP 2

Si vous n'avez pas eu le temps de terminer le précédent TP, vous pouvez essayer de le finir avant de passer à la suite.

2 Acceptation de mots

Les automates finis non-déterministes (NFA) sont modélisés par la classe `NFA` introduite précédemment au TP 2. L'automate `A` de la Figure 1 est obtenu en exécutant les instructions suivantes :

```
A = NFA([0, 1, 2, 3, 4], [0], [2], ["a", "b"], [(0, "a", 1), (1, "a", 2), \
(1, "b", 1), (0, "b", 3), (3, "a", 3), (4, "a", 2)])
export_automaton(A, "A")
```

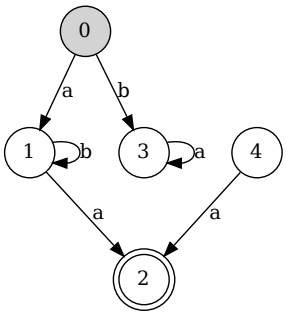

Figure 1 – Une représentation graphique de l'automate `A`.

Un mot est *accepté* par un NFA si et seulement si il existe un chemin étiqueté par ce mot allant d'un état initial à un état final. Le premier objectif de ce TP est d'implémenter cette notion d'acceptation.

Question 1

Ajoutez une méthode `reachable_states(self, origin, word)` à la classe `NFA` qui renvoie l'ensemble des états accessibles depuis l'état `origin` après lecture de la chaîne `word`.

Indication. Une définition récursive est de rigueur : si `word` est vide (vérifiable par `not word`), on renvoie `origin` ; sinon, on détermine les successeurs de `origin` selon la premier lettre de `word`, et on applique la fonction récursivement.

Question 2

Ajoutez une méthode `accept(self, word)` à la classe `NFA` qui renvoie `True` si le NFA accepte la chaîne `word` et `False` sinon.

3 Automates émondés

L'*émondage* d'un NFA consiste à éliminer ses états inutiles. Un état q est dit *accessible* s'il existe un chemin depuis un état initial vers q ; il est dit *co-accessible* s'il existe un chemin depuis q vers un état final ; il est *utile* s'il est à la fois accessible et co-accessible. Le second objectif de ce TP est d'implémenter un algorithme d'émondage.

Retirer les états inutiles d'un automate ne change pas le langage accepté. Ainsi, l'automate `B` de la Figure 2 obtenu en retirant les états inutiles `3` et `4` accepte un langage équivalent à `A`.

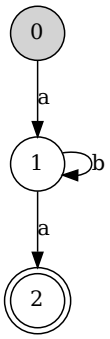

FIGURE 2 – L'automate `B` obtenu en émondant l'automate `A`.

Question 3

Ajoutez une méthode `accept(self, origin, target)` à la classe `NFA` qui renvoie `True` si l'état `target` est accessible depuis l'état `origin` et `False` sinon.

Indication. Un simple parcours de graphe devrait convenir.

Question 4

Ajoutez une méthode `is_useful(self, state)` à la classe `NFA` qui renvoie `True` si l'état `state` est utile et `False` sinon.

Question 5

Ajoutez une méthode `remove_state(self, state)` à la classe `NFA` qui retire l'état `state` du NFA courant.

Indication. Pensez à éliminer `state` de tous les ensembles d'états en attribut avec la méthode `remove`, et à retirer du dictionnaire des transitions les entrées associées à `state` avec l'instruction `del`.

Question 6

Ajoutez une méthode `prune(self)` à la classe `NFA` qui retire tous les états inutiles du NFA courant.