# Théorie des langages réguliers : TP 4

Adrien Pommellet

1er septembre 2021

Téléchargez au préalable le fichier `thlr_tp4.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 Automates déterministes et complets

Les automates finis non-déterministes (NFA) sont modélisés par la classe `NFA` introduite précédemment au TP 2, en y incluant les méthodes ajoutées lors du TP 3. L'automate `A` de la Figure 1 est obtenu en exécutant les instructions suivantes :

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

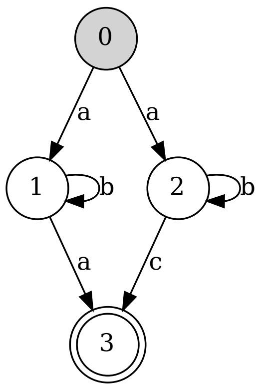


FIGURE 1 – Une représentation graphique de l'automate `A`.

Un automate est dit *complet* (resp. *déterministe*) si pour toute lettre $a$ et état $q$ il existe au moins (resp. au plus) une transition sortant de $q$ étiquettée par $a$.

### Question 1

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

### Question 2

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

## 2 Déterminisation d'automates

Le premier objectif de ce TP est d'implémenter l'algorithme canonique de *déterminisation* d'un NFA $\mathcal{A}$, c'est-à-dire de calculer un NFA déterministe complet $\mathcal{A}'$ de langage équivalent. Cette construction par sous-ensemble consiste à définir un ensemble $Q'$ d'états de $\mathcal{A}'$ isomorphes aux parties $2^Q$ de l'ensemble d'états $Q$ de $\mathcal{A}$. La notion de successeur est alors étendue aux ensembles d'états : le successeur d'un ensemble d'états $E \subseteq Q$ selon la lettre $a$ correspond à l'ensemble des états accessibles depuis un état de $E$ par la lettre $a$.

L'algorithme de déterminisation commence par explorer l'ensemble des états initiaux $I \subseteq Q$ de l'automate originel $\mathcal{A}$, c'est-à-dire détermine les successeurs de $I$ selon les différentes lettres de l'alphabet, puis explore ces successeurs à leur tour jusqu'à ne plus avoir de nouvel ensemble à explorer. Les états de l'automate déterminisé sont alors isomorphes à ces sous-ensembles explorés ; l'unique état initial de $\mathcal{A}'$ correspond à $I$ ; sont finaux les états de $Q'$ qui correspondent à un ensemble d'états dans $2^Q$ contenant au moins un état final de l'automate originel $\mathcal{A}$.

L'automate `B` de la Figure 2 est le déterminisé de `A` ; l'état `0` correspond à l'ensemble $\{0\}$, l'état `1` à $\emptyset$, l'état 2 à $\{1, 2\}$, et l'état `3` à $\{3\}$.

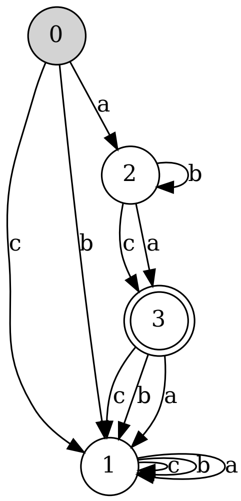


FIGURE 2 – L'automate `B` obtenu en déterminisant `A`.

### Question 3

Ajoutez une méthode `reachable_set(self, origins, letter)` à la classe `NFA` qui renvoie l'ensemble des états accessibles depuis l'ensemble `origins` par la lettre `letter`. Attention, il doit bien s'agir d'un ensemble et non d'une liste.

### Question 4

Ajoutez une méthode `determinize(self)` à la classe `NFA` qui renvoie un nouveau NFA qui est le déterminisé du NFA courant.

**Indication.** L'implémentation de cet algorithme reste délicate. Il est recommandé de maintenir deux listes d'ensembles : une liste `power_sets` représentant les ensembles d'états déjà explorés, et une autre `incoming_sets` les états à explorer. Quand on explore un état, il faut le retirer de `incoming_sets`, l'ajouter à `power_sets`, puis ajouter ses successeurs qui ne sont pas déjà dans `incoming_sets` ou `power_sets` à `incoming_sets`.

La difficulté vient de l'implémentation des états dans la classe `NFA` : ces derniers sont représentés par des entiers et non des ensembles d'entiers. Par conséquent, l'indice d'un ensemble d'états dans la liste `power_sets` sera le numéro attribué à l'état correspondant dans l'automate déterminisé. Pensez également à maintenir une liste d'arêtes `new_edges` du déterminisé exprimée en utilisant cette numérotation par indice.

Une fois l'exploration des ensembles d'états terminée (`incoming_sets` est vide), déterminez les états finaux du déterminisé. Puis utilisez le constructeur de la classe `NFA` pour renvoyer l'automate déterminisé.

## 3 Minimisation d'automates

Le dernier objectif de ce TP est d'implémenter l'algorithme de Brzozowski de *minimisation* d'un NFA $\mathcal{A}$ qui permet d'obtenir un NFA déterministe $\mathcal{A}'$ équivalent avec le plus petit nombre d'états possible. Cette méthode, qui diffère de l'algorithme canonique vu en cours, nécessite la *transposition* d'un automate : l'automate transposé, obtenu en inversant le sens des transitions ainsi que les états initiaux et finaux, accepte le langage miroir de celui de l'automate originel. L'automate `C` de la Figure 3 est ainsi le transposé de `A`.

Si l'on note *det* l'opération de déterminisation (que l'on peut faire suivre immédiatement d'un émondage) et *tr* celle de transposition, alors l'automate déterministe minimal équivalent à un automate $\mathcal{A}$ est $\mathcal{A}' = det(tr(det(tr(\mathcal{A}))))$. L'automate `D` de la Figure 4 est ainsi le minimisé de `A`.

### Question 5

Ajoutez une méthode `mirror(self)` à la classe `NFA` qui renvoie le NFA transposé du NFA courant.

### Question 6

Ajoutez une méthode `minimize(self)` à la classe `NFA` qui renvoie le DFA minimal associé au NFA courant.

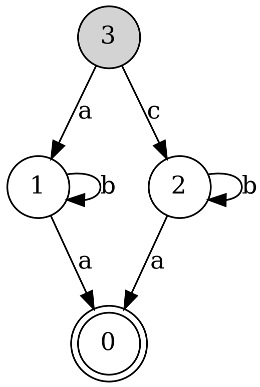


FIGURE 3 – L'automate `C`, transposé de l'automate `A`.

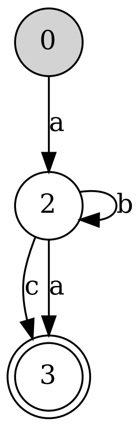


FIGURE 4 – L'automate `D` obtenu en minimisant l'automate `A`.