Théorie des langages réguliers : TP 2

Adrien Pommellet

1er septembre 2021

Téléchargez au préalable le fichier `thlr_tp2.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 Expressions régulières et automates

Nous modéliserons les automates et les expressions régulières en Python par des *classes*.

Les expressions régulières sont représentées sous forme d'arbre par la classe `RegEx`. Ses attributs sont la chaîne `symbol` à la racine de l'arbre qui décrit un opérateur `+`, `*`, `.` ou une lettre de l'alphabet, et la liste `children` des `RegEx` enfants de la racine (selon l'arité de l'opérateur à la racine). Son constructeur `RegEx(symbol, children)` permet de définir les attributs éponymes.

La méthode `to_string(self)` renvoie la représentation de la `RegEx` courante sous forme de chaîne. La construction de l'expression régulière $e = a^* + bc$ emploie les instructions suivantes :

```
a = RegEx("a", [])
b = RegEx("b", [])
c = RegEx("c", [])
a_star = RegEx("*", [a])
bc = RegEx(".", [b, c])
e = RegEx("+", [a_star, bc])
print("Expects (a)*+bc:", e.to_string())
```

Les automates finis non-déterministes avec ε-transitions (ε-NFA) sont modélisés par la classe `ENFA`, dont les attributs `all_states`, `initial_states`, `final_states` sont des ensembles d'entiers, `alphabet` un ensemble de chaînes, et dont le dictionnaire `next_states` permet de déterminer les successeurs d'un état : `next_states(p, a)` renvoie l'ensemble des états `q` tels qu'il y ait une arête $(\texttt{p}, \texttt{a}, \texttt{q})$ dans l'automate. La chaîne vide `""` représente le mot vide ε.

L'utilisation d'ensembles en langage Python permet d'éviter de stocker des doublons d'un même élément. On initialise un ensemble avec les instructions `set()` ou `{}` ; on peut y ajouter (resp. retirer) un élément `x` avec la méthode `.add(x)` (resp. `.remove(x)`).

Le constructeur `ENFA(self, all_states, initial_states, final_states, alphabet, edges)` prend en argument trois listes d'entiers `all_states`, `initial_states`, et `final_states`, une liste de chaînes `alphabet`, et une liste de triplets `edges` représentant les arêtes : le triplet `(0, "a", 1)` représente l'arête $0 \xrightarrow{a} 1$. L'automate `A` de la Figure 1 est obtenu en exécutant les instructions suivantes :

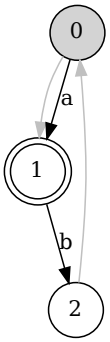

FIGURE 1 – Une représentation graphique de l'automate `A`.

```
A = ENFA([0, 1, 2], [0], [1], ["a", "b"], [(0, "a", 1), (0, "", 1),
(1, "b", 2), (2, "", 0)])
export_automaton(A, "A")
```

Les arêtes grisées représentent des ε-transitions.

Question 1

Ajoutez une méthode `new_state(self)` à la classe `ENFA` qui insère un nouvel état dans l'attribut `all_states` et renvoie le numéro de ce nouvel état.

Indication. Quel numéro donner à ce nouvel état ? N'oubliez pas de mettre à jour le dictionnaire `next_state`.

Question 2

Ajoutez une méthode `new_letter(self, letter)` à la classe `ENFA` qui insère la chaîne `letter` dans l'attribut `alphabet`.

Indication. Pensez à mettre le dictionnaire `next_state` à jour avec cette nouvelle lettre.

2 Algorithme de Thompson

Le premier objectif de ce TP est d'implémenter l'algorithme de Thompson que vous avez vu en cours : partant d'une expression régulière e, on souhaite construire un ε-NFA $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = e$. Rappelons que cet algorithme associe récursivement à des expressions régulières les motifs d'automates décrits dans la Figure 2.

Question 3

Ajoutez une méthode `convert_reg_ex(self, origin, destination, reg_ex)` à la classe `ENFA` qui insère entre les états `origin` et `destination` une représentation par automate de l'expression régulière `reg_ex`.

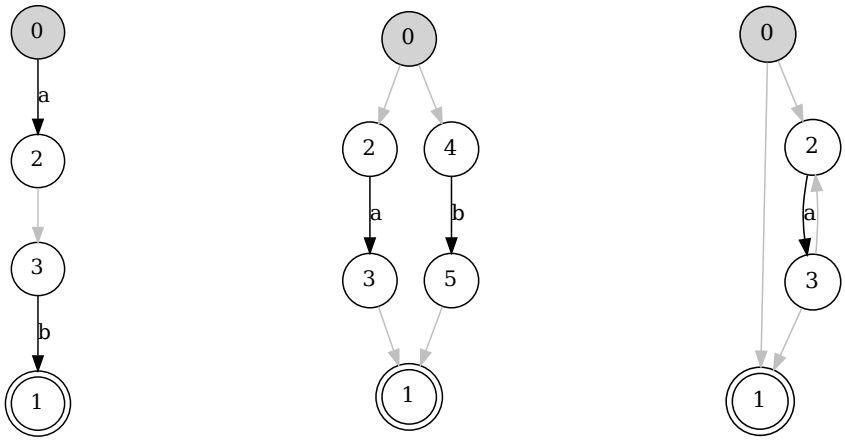

FIGURE 2 – Les motifs associés aux expressions régulières $a \cdot b$, $a + b$, et a^*.

Indication. Utilisez le motif associé à l'opérateur `symbol` situé à la racine de l'expression régulière `reg_ex` pour créer de nouveaux états et de nouvelles transitions puis appliquez récursivement la même fonction aux éléments de `children`.

Question 4

Ajoutez une méthode `to_enfa(self)` à la classe `RegEx` qui renvoie un `ENFA` représentant l'expression régulière courante.

3 Algorithme d'élimination des ε-transitions

Le second objectif de ce TP est l'implémentation de l'algorithme canonique de conversion d'un ε-NFA $\mathcal{A}$ en un automate fini non-déterministe (NFA) $\mathcal{A}'$ équivalent. Les NFA sont modélisés par la classe `NFA`, similaire à la classe `ENFA` si ce n'est que l'on interdit l'emploi de la lettre `""` représentant ε.

L'algorithme de conversion nécessite au préalable le calcul de l'ε-fermeture avant $\varepsilon^{\mathcal{A}}_{for}(q)$ de chaque état q de l'automate $\mathcal{A}$, à savoir l'ensemble des états accessibles depuis q en utilisant uniquement des ε-transitions. Ce calcul s'effectue par point fixe itératif : initialement, seul $q \in \varepsilon^{\mathcal{A}}_{for}(q)$. Puis, si $q_1 \in \varepsilon^{\mathcal{A}}_{for}(q)$ et $q_1 \xrightarrow{\varepsilon}_{\mathcal{A}} q_2$, on ajoute alors $q_2 \in \varepsilon^{\mathcal{A}}_{for}(q)$. Le calcul s'arrête lorsqu'il n'y a plus de nouvel état à ajouter : un point fixe est alors atteint.

Le NFA $\mathcal{A}'$ est alors défini de la manière suivante : ses états et états initiaux sont égaux à ceux de $\mathcal{A}$; pour tout état q, si dans l'automate originel $q_1 \in \varepsilon^{\mathcal{A}}_{for}(q)$ et $q_1 \xrightarrow{a}_{\mathcal{A}} q_2$, alors on ajoute une arête $q \xrightarrow{a}_{\mathcal{A}'} q_2$; enfin, si $q_1 \in \varepsilon^{\mathcal{A}}_{for}(q)$ et q_1 est un état final de $\mathcal{A}$, alors q doit devenir un état final de $\mathcal{A}'$. Partant de l'automate `A`, on souhaite donc obtenir l'automate `B` de la Figure 3.

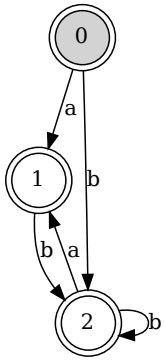

FIGURE 3 – Un NFA `B` équivalent à l'ε-NFA `A`.

Question 5

Ajoutez une méthode `epsilon_reachable(self, origin)` à la classe `ENFA` qui renvoie l'ε-fermeture avant de l'état `origin`.

Indication. Utilisez alors une variable `fixed_point_reached` initialisée à `true` pour mémoriser si la fermeture a été modifiée lors de l'itération courante, et terminez le calcul si ce n'est pas le cas.

Question 6

Ajoutez une méthode `to_nfa(self)` à la classe `ENFA` qui renvoie un `NFA` équivalent au `ENFA` courant.

Indication. N'oubliez pas de mettre à jour les états finaux !