Théorie des langages rationnels
TP 4 - Conversion d'expressions régulières

Adrien Pommellet

18 janvier 2023

L'objectif de ce TP est d'implémenter les algorithmes de Thompson et d'élimination arrière des ε-transitions que vous avez vu en cours.

Téléchargez au préalable les fichiers `thlr_automata.py` et `thlr_regex.py` sur Moodle. Ce TP nécessite une installation préalable de la bibliothèque `graphviz` ; si elle ne s'avère pas disponible sur votre poste, utilisez la suite d'instructions suivante :

```
python -m venv ~/envthlr
. ~/envthlr/bin/activate
pip install graphviz
```

Les questions marquées d'une étoile `(*)` seront **notées** et font l'objet d'un rendu dont **la rédaction doit obligatoirement respecter les règles suivantes** :

— Les réponses au TP doivent être regroupées dans un unique fichier nommé `thlr_4_code.py` (tout autre fichier sera ignoré). Inutile d'uploader les bibliothèques externes que l'on vous fournit telle que `thlr_automata.py`.

— Il est impératif que le fichier soit interprétable, sans la moindre erreur de syntaxe. Si ce n'est pas le cas, le rendu entier sera ignoré. Testez votre code sous Linux directement depuis la console avec l'instruction `python3 thlr_4_code.py` depuis un répertoire contenant ses éventuelles dépendances, sans passer par un IDE.

— Les noms de fonctions donnés dans l'énoncé doivent être respectés.

— La réponse à la question numéro `i` et les éventuelles sous-fonctions utilisées doivent être placées entre des balises de la forme `#[Qi]` et `#[/Qi]` (sans espace après `#`, avec un `/` et non un \). Ainsi, si l'on vous demande d'écrire une fonction `f`, votre réponse doit être de la forme :

```
#[Qi]
def f():
  # Insert your code here
#[/Qi]
```

— Testez vos fonctions sur des exemples pour vous assurer de leur correction ; n'incluez toutefois pas ces tests et ces exemples entre les tags `#[Qi]` et `#[/Qi]`.

— Certaines fonctions sont interdépendantes : une erreur sur la première fonction peut compromettre les suivantes, soyez donc vigilants.

Le non-respect de ces critères peut mettre en défaut le système de notation automatique et conduire à une note de 0 pour le rendu dans son intégralité, sans contestation possible.

1 Algorithme de Thompson

Partant d'une expression régulière e, on souhaite construire un automate fini non déterministe avec ε-transitions (ε-NFA) $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = e$.

1.1 Retour sur les expressions régulières

Nous allons réutiliser la bibliothèque d'expressions régulières introduite lors du TP 2, ainsi que celle d'automates présentée lors du TP 3. N'oubliez donc pas le préambule :

```
from thlr_automata import *
from thlr_regex import *
```

Rappelons que la classe `RegEx` implémente une structure d'arbre. Relisez le TP 2 pour vous remettre en tête son utilisation.

Question 1. Définissez les arbres `r1` et `r2` associés aux expressions régulières $r_1 = a^* \cdot b + c$ et $r_2 = (a + \varepsilon)^* \cdot c$, puis affichez-les. Copiez-collez le symbole ε depuis cet énoncé si besoin est.

1.2 Implémentation de l'algorithme

Rappelons que cet algorithme associe récursivement à des expressions régulières les motifs d'automates décrits dans la figure 1. Nous allons donc en faire une implémentation récursive :

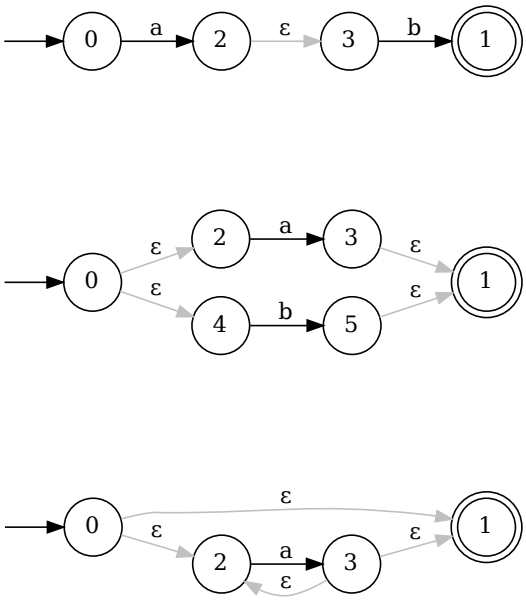

FIGURE 1 – Les motifs associés aux expressions régulières $a \cdot b$, $a + b$, et a^*.

partant d'une expression régulière `r`, en fonction de l'opérateur `r.root`, il faut créer de nouveaux

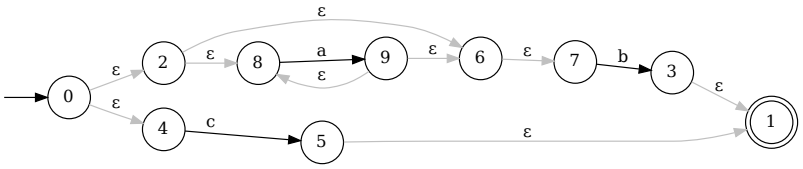

FIGURE 2 – L'automate `A1` correspondant à l'expression $r_1 = a^* \cdot b + c$.

états, de nouvelles transitions, puis appliquer récursivement la procédure aux enfants dans la liste `r.children`. Les feuilles de l'arbre (dont la liste `r.children` est vide, et dont le symbole `r.root` ne correspond à aucun opérateur) correspondent aux lettres de l'alphabet.

Par exemple, pour insérer entre les états `0` et `1` l'automate d'une expression régulière de la forme `e = new_regex("e1.e2")`, il faut, après avoir identifié la racine de `e` comme étant égale à `"."`, créer deux nouveaux états `2` et `3`, une arête étiquetée par ε entre `2` et `3`, puis appeler récursivement l'algorithme pour `e1` entre `0` et `2` ainsi que pour `e2` entre `3` et `1`.

Revoyez le TP 3 et les opérations sur les automates de la classe `ENFA` si vous avez des doutes.

Question 2 ($\star$). Programmez une fonction `convert_regex(enfa, origin, destination, regex)` qui insère entre les états `origin` et `destination` de l'automate `enfa` une représentation par automate de l'expression régulière `regex`. Faites attention au cas ε : dans l'automate, une transition ne doit pas être étiquetée par ε mais par la chaîne vide `""` ; l'arête correspondante apparaît alors comme grisée dans la représentation graphique.

Question 3 ($\star$). Écrivez une fonction `to_enfa(regex)` qui renvoie un `ENFA` représentant l'expression régulière `regex`. Pour ce faire, appliquez la précédente fonction `convert_regex` à un automate à deux états très simple. L'application de cette fonction à l'expression régulière r_1 doit produire un automate semblable (à la numérotation des états près) à l'automate `A1` de la figure 2. De même, appliquez cette fonction à r_2 pour obtenir un automate `A2`.

2 Algorithme d'élimination des ε-transitions

Cette partie est consacrée à l'implémentation de l'algorithme canonique de conversion d'un ε-NFA $\mathcal{A}$ en un automate fini non-déterministe (NFA) $\mathcal{A}'$ équivalent.

2.1 Calcul de l'ε-fermeture

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. Par exemple, l'ε-fermeture avant de l'état `0` dans l'automate `A1` de la figure 2 correspond à l'ensemble `{0, 2, 4, 6, 7, 8}`.

La manière la plus simple est d'effectuer une recherche d'accessibilité qui ne prend en compte que les ε-transitions.

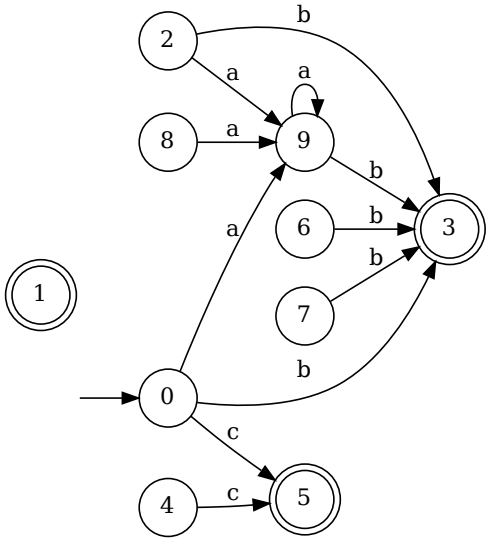

FIGURE 3 – L'automate `B1` obtenu par élimination arrière des ε-transitions de `A1`.

Question 4 ($\star$). Écrivez une fonction `get_epsilon_closure(enfa, origin)` qui renvoie sous forme d'ensemble l'ε-fermeture avant de l'état `origin` dans l'automate `enfa`. Le résultat attendu est très proche dans son implémentation de la fonction `get_accessible_states` du TP 3.

2.2 Passage à un automate sans ε-transitions

Le NFA $\mathcal{A}'$ équivalent au ε-NFA $\mathcal{A}$ originel 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 $\mathcal{A}'$.

Les NFA sont modélisés par la classe `NFA`, similaire dans sa manipulation et son initialisation à la classe `ENFA` si ce n'est que l'on interdit l'emploi de la lettre `""` représentant ε.

Question 5 ($\star$). Écrivez une fonction `to_nfa(enfa)` qui renvoie une instance de la classe `NFA` équivalente à l'automate `enfa`. L'application de cette fonction à l'automate `A1` de classe `ENFA` de la figure 2 doit fournir un automate de classe `NFA` semblable à `B1` de la figure 3. De même, appliquez cette fonction à `A2` pour obtenir un automate `B2`.

Question 6. Appliquez la fonction `prune` du TP 3 (en l'important avec la commande `import`) aux automates `B1` et `B2` pour obtenir les automates émondés `C1` (voir la figure 4) et `C2`. Prenez soin de ne **pas** inclure cette question ainsi que l'importation du TP 3 dans votre rendu, dans la mesure où le TP 3 ne sera pas disponible lors des tests.

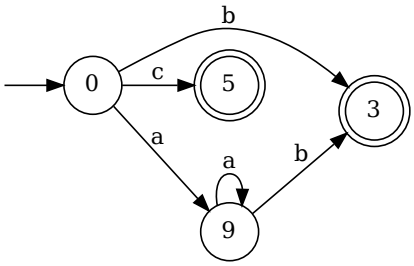

FIGURE 4 – L'automate `C1` obtenu par émondage de `B1`.