TME2 : Automates à pile

(définitions, exemples, programmation en Python)

– version 2.0 (Janvier 2022) –

Mathieu.Jaume@lip6.fr

1 Automates à pile

Le code présenté dans cette section se trouve dans le fichier `automates_a_pile.py`.

1.1 Définition et représentation

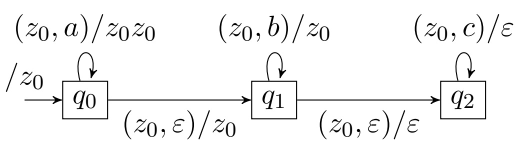
Un *automate à pile* $\mathcal{A}$ est un 6-uplet $\mathcal{A} = (Q, \Sigma, Z, T, (q_0, z_0), K)$ où Q est un ensemble fini d'*états*, Σ est l'*alphabet d'entrée*, Z est l'*alphabet de la pile*, $(q_0, z_0) \in Q \times Z$ est la *configuration initiale*, $T \subseteq ((Q \times Z \times (\Sigma \cup \{\varepsilon\})) \times (Q \times Z^\star))$ est la *relation de transition*, et $K \subseteq Q \times Z^\star$ est une *condition d'acceptation* telle que soit $K = F \times Z^\star$ pour un ensemble $F \subseteq Q$ d'états contenant les états acceptants, soit $K = Q \times \{\varepsilon\}$ (acceptation par pile vide). Dans la suite, on notera $(q, z) \xrightarrow{x} (q', w)$ la transition $((q, z, x), (q', w)) \in T$. Une pile sera représentée par une liste dont le premier élément (i.e. l'élément le plus à gauche) correspond au sommet de pile.

Implantation On représente un automate $\mathcal{A} = (Q, \Sigma, Z, T, (q_0, z_0), K)$ par un tuple :

```
a = (st,alph,stack_alph,t_rel,init_st,init_stack,accept_mode,final_st,eq_st)
```

où `st`, `alph`, `stack_alph` et `t_rel` sont les listes représentant respectivement les ensembles Q, Σ, Z et T, `init_st` est l'état q_0, `init_stack` est le symbole z_0, `accept_mode` est l'entier 0 si $K = Q \times \{\varepsilon\}$ (acceptation par pile vide) ou l'entier 1 si $K = F \times Z^\star$ (acceptation par état final), `final_st` est la liste représentant l'ensemble F des états acceptants si `accept_mode` vaut 1 et est la liste vide sinon, et `eq_st` est la fonction d'égalité sur les états.

Exemple 1.1 On considère l'automate à pile $\mathcal{A}_1$ (acceptant par pile vide) de la figure ??.



$Q_1 = \{q_0, q_1, q_2\}$
$\Sigma_1 = \{a, b, c\}$
$Z_1 = \{z_0\}$
$K_1 = Q_1 \times \{\varepsilon\}$

FIGURE 1 – $\mathcal{A}_1 = (Q_1, \Sigma_1, Z_1, T_1, (q_0, z_0), K_1)$

Cet automate peut être implanté comme suit :

exemple : automate à pile $\mathcal{A}_1$ (figure ??)

```
a1_st = ["q0","q1","q2"]
a1_alph = ["a","b","c"]
a1_stack_alph = ["z0"]
a1_t_rel = [(("q0","z0","a"),("q0",["z0","z0"])),\
            (("q0","z0",None),("q1",["z0"])),\
            (("q1","z0","b"),("q1",["z0"])),\
            (("q1","z0",None),("q2",[])),\
            (("q2","z0","c"),("q2",[]))]
a1_init_st = "q0"
a1_init_stack = "z0"
a1_accept_mode = 0
a1_final_st = []
a1_eq_st = eq_atom
a1_a = (a1_st,a1_alph,a1_stack_alph,a1_t_rel,a1_init_st,\
        a1_init_stack,a1_accept_mode,a1_final_st,a1_eq_st)
```

1.2 Exécution, langage reconnu par un automate à pile

Etant donné un automate à pile $\mathcal{A} = (Q, \Sigma, Z, T, (q_0, z_0), K)$, une *configuration* est un couple (q, p) où $q \in Q$ et $p \in Z^\star$ est une liste représentant la pile. C'est une *configuration acceptante* si $K = F \times Z^\star$ et $q \in F$, ou bien si $K = Q \times \{\varepsilon\}$ et p est vide. Une *exécution* de $\mathcal{A}$ à partir d'un mot $w \in \Sigma^\star$ est une séquence de transitions :

$$(q_0, z_0) \overset{x_1}{\rightarrowtail} (q_1, p_1) \overset{x_2}{\rightarrowtail} (q_2, p_2) \overset{x_3}{\rightarrowtail} \cdots \overset{x_n}{\rightarrowtail} (q_n, p_n)$$

notée $(q_0, z_0) \overset{w}{\rightarrowtail}{}^* (q_n, p_n)$, telle que :

— $w = x_1 x_2 \cdots x_n$ avec $x_1, x_2, \cdots, x_n \in \Sigma \cup \{\varepsilon\}$

— pour chaque $(q_i, p_i) \overset{x_{i+1}}{\rightarrowtail} (q_{i+1}, p_{i+1})$, il existe dans T une transition $(q_i, z) \xrightarrow{x} (q_{i+1}, u)$ telle que $p_i = zp'_i$ et $p_{i+1} = up'_i$

C'est une *exécution acceptante* si (q_n, p_n) est une configuration acceptante. La *langage reconnu* par $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$, est l'ensemble des mots de $\Sigma^\star$ à partir desquels il existe au moins une exécution acceptante :

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^\star \mid (q_0, z_0) \overset{w}{\rightarrowtail}{}^* (q_n, p_n) \text{ et } (q_n, p_n) \in K \right\}$$

Exemple Le langage reconnu par l'automate $\mathcal{A}_1$ de la figure **??** est $\mathcal{L}(\mathcal{A}_1) = \{a^n b^m c^n \mid n, m \geq 0\}$.

Appartenance d'un mot au langage reconnu par un automate à pile

On cherche ici à implanter une fonction permettant de tester l'appartenance d'un mot au langage reconnu par un automate à pile. Pour cela, on définit tout d'abord la fonction `find_trans` permet de construire la liste des parties droites des transitions de `t_rel` étiquetées par $\mathtt{x} \in \Sigma \cup \{\varepsilon\}$ à partir d'une configuration `(q,s)`.

$$\left\{ (\mathtt{q}', \mathtt{w}) \mid (\mathtt{q}, \mathtt{s}) \xrightarrow{\mathtt{x}} (\mathtt{q}', \mathtt{w}) \in \mathtt{t\_rel} \right\}$$

`find_trans`

```
def find_trans(q,eq_st,t_rel,s,x):
    # q : etat
```

```
    # eq_st : fonction d'egalite sur les etats
    # t_rel : relation de transition
    # s : symbole de pile
    # x : etiquette (None ou symbole de l'alphabet)
    res = []
    for ((qr,zr,xr),(nqr,wr)) in t_rel:
        if xr==x and s==zr and eq_st(q,qr):
            res = res + [(nqr,wr)]
    return res
```

Cette fonction permet de définir la fonction `next_configs` qui étant donnés une configuration `c` et un mot `w` (représenté par une liste) permet de construire la liste de toutes les paires `(c',w')` telles que :

$$\left(\mathtt{c} \overset{\varepsilon}{\rightarrowtail} \mathtt{c}' \text{ et } \mathtt{w} = \mathtt{w}'\right) \text{ ou } \left(\mathtt{c} \overset{\mathtt{x}}{\rightarrowtail} \mathtt{c}' \text{ et } \mathtt{w} = \mathtt{x}\mathtt{w}'\right)$$

à compléter : `next_configs`

```
def next_configs(a,c,w):
    # a : automate a pile
    # c : configuration (etat,pile)
    # w : mot represente par une liste
```

Nous pouvons à présent définir la fonction `is_in_LA` qui simule l'exécution d'un automate à pile `a` étant donné un mot `w` (représenté par une chaîne de caractères), et lorsque cette fonction termine, détermine si le mot `w` appartient au langage reconnu par l'automate `a`. Cette fonction procède de manière récursive sur les exécutions possibles de l'automate à partir de `w`.

à compléter : acceptation d'un mot par un automate à pile

```
def is_in_LA(a,w):
    # a : automate a pile
    # w : mot represente par une liste
```

exemples : ε, *ac*, *aabbbc*, *aaaccc*, *aabbbbbbbcc* et *bbaaaccc* $\in \mathcal{L}(\mathcal{A}_1)$?

```
>>> is_in_LA(a1_a,"")
True
>>> is_in_LA(a1_a,"ac")
True
>>> is_in_LA(a1_a,"aabbbbc")
False
>>> is_in_LA(a1_a,"aaaccc")
True
>>> is_in_LA(a1_a,"aabbbbbbbcc")
True
>>> is_in_LA(a1_a,"bbaaaccc")
False
```