

Partiel
Automates
2020

Exercice 1

1) ϵ n'est pas déterministe car

- * il y a des ϵ -transitions
- * $|\delta_{\epsilon}(A, a)| > 1$
- * il y a deux états initiaux.

déterminisation: On pose $Q_0 = \{A, D\} \cup \{F(A)\} \cup \{F(D)\}$

$$\underline{D_0 = \{A, D, B\}}$$

$$\begin{aligned}\delta_{\epsilon}(Q_0, a) &= \{C, B\} \cup \{F(C)\} \cup \{F(B)\} \\ &= \{C, B, D\} = Q_1.\end{aligned}$$

$$\delta_{\epsilon}(Q_0, b) = \{B\} \cup \{F(B)\} = \{B, D\} = Q_2$$

$$\begin{aligned}\delta_{\epsilon}(Q_1, a) &= \{D, C\} \cup \{F(D)\} \cup \{F(C)\} \\ &= \{C, D, B\} = Q_1.\end{aligned}$$

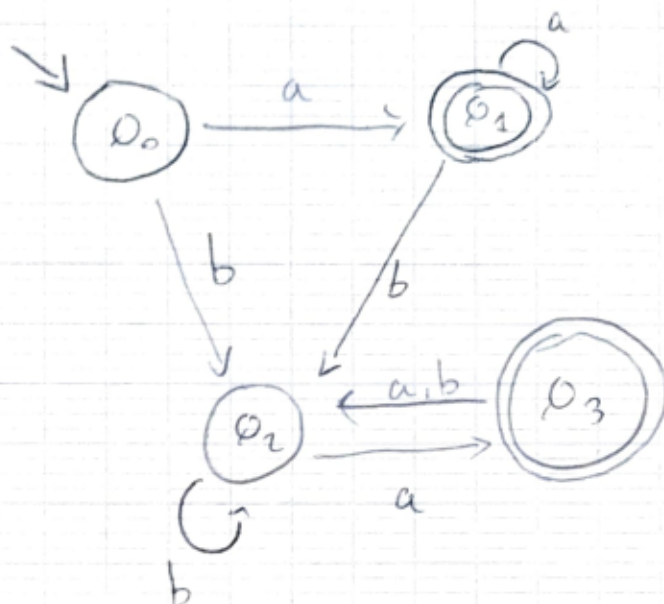
$$\delta_{\epsilon}(Q_1, b) = \{B\} \cup \{F(B)\} = \{B, D\} = Q_2$$

$$\delta_{\epsilon}(Q_2, a) = \{C\} \cup \{F(C)\} = \{C\} = Q_3$$

$$\delta_f(q_2, b) = \{B\} \cup \{F(B)\} = \{B, D\} = q_2$$

$$\delta_f(q_3, a) = \{D\} \cup \{F(D)\} = \{B, D\} = q_2$$

$$\delta_f(q_3, b) = \{B\} \cup \{F(B)\} = \{B, D\} = q_2$$



2) table de transition :

	état	a	b
→	q ₀	q ₁	q ₂
*	q ₁	q ₁	q ₂
	q ₂	q ₃	q ₂
*	q ₃	q ₂	q ₂

→ initial

* final

Exercice 2 :

$$1) \begin{cases} L_E = bL_E + aL_G & (1) \\ L_G = bL_F + \Lambda & (2) \\ L_F = bL_F + aL_E & (3) \end{cases}$$

$$2) (3) \rightarrow L_F = bL_F + aL_E = b^* a L_E \text{ (Lemme d'Arden)}$$

$$(2) \rightarrow L_G = bL_F + \Lambda \stackrel{(3)}{=} b b^* a L_E + \Lambda$$

$$(1) \rightarrow L_E = bL_E + aL_G = \cancel{b^* a L_G} \text{ (Lemme d'Arden)}$$

$$= bL_E + a b b^* a L_E + a$$

$$\begin{cases} \underline{L_E = (b + a b b^* a)^* a} \\ \underline{L_F = b^* a (b + a b b^* a)^* a} \\ \underline{L_G = b b^* a (b + a b b^* a)^* a + \Lambda} \end{cases}$$

expression régulière : $L_E = \boxed{(b + a b b^* a)^* a}$

Exercise 3: On pos $q_0 = (a \mid \perp) \mid ((ba) * \mid ba)$

$$D_a(q_0) = (D_a(a \mid \perp) \mid ((ba) * \mid ba)) \mid D_a((ba) * \mid ba)$$

$$= ((ba) * \mid ba) \mid \underbrace{D_a((ba) *)}_{\emptyset} \mid \underbrace{D_a(ba)}_{=q}$$

$$= (ba) * \mid ba = q_1.$$

$$D_b(q_0) = (D_b(a \mid \perp) \mid ((ba) * \mid ba)) \mid D_b((ba) * \mid ba)$$

$$\mid D_b(ba)$$

$$= \emptyset \mid D_b(ba) (ba) * \mid a$$

$$= a (ba) * \mid a = q_2$$

$$D_a(q_1) = \emptyset$$

$$D_b(q_1) = q_2$$

$$D_a(q_2) = (ba) * \mid \perp = q_3.$$

$$D_b(q_2) = \emptyset$$

$$D_a(q_3) = \emptyset$$

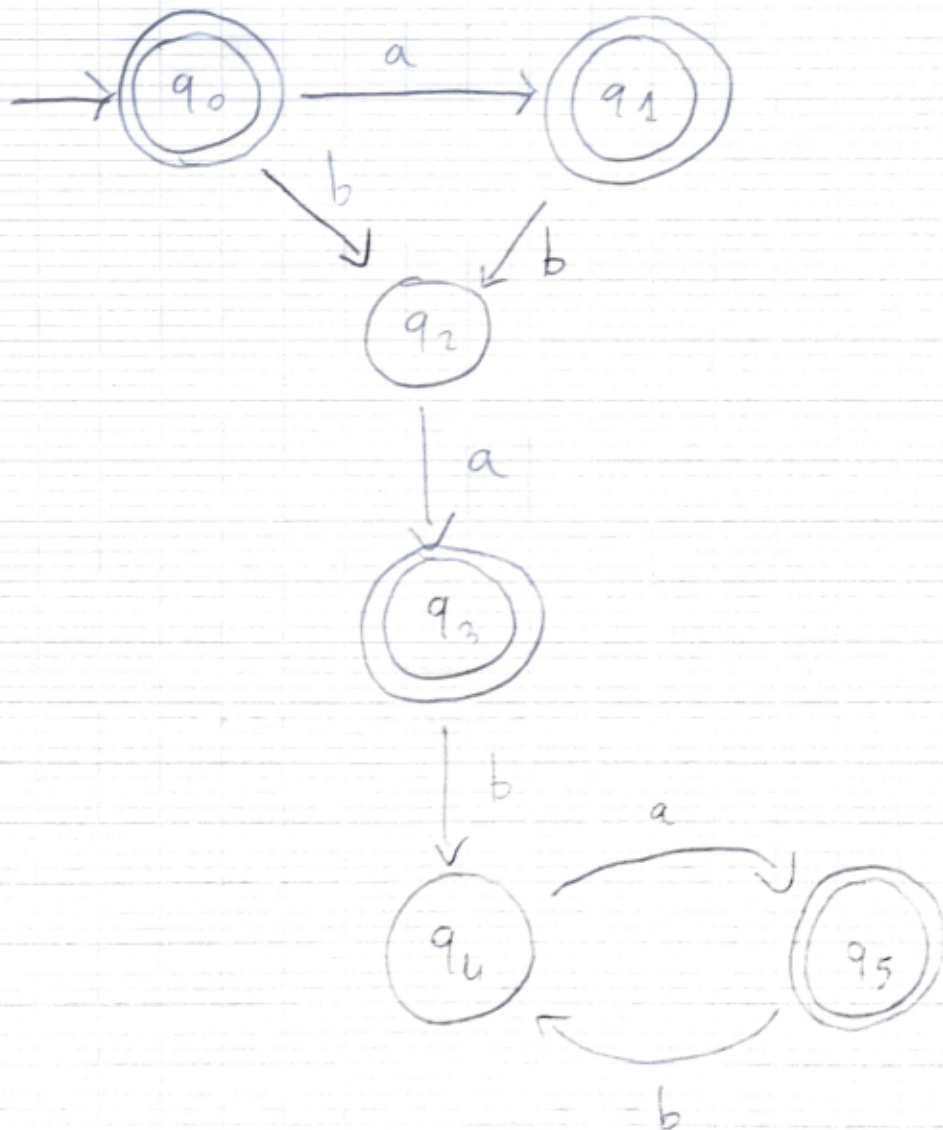
$$D_b(q_3) = a (ba) * = q_4$$

$$D_a(q_4) = (ba)^+ = q_5$$

$$D_b(q_4) = \emptyset$$

$$D_a(q_5) = \emptyset$$

$$D_b(q_5) = a(ba)^+ = q_4$$



Exercice 4:

1) Nouvelle grammaire sans récursivité à gauche:

$$\begin{cases} N \rightarrow R p I \mid R \mid I \\ I \rightarrow R i \\ R \rightarrow c R' \\ R' \rightarrow c R' \mid \Delta \end{cases}$$

transformation en grammaire régulière:

$$N \rightarrow c R' p I \mid c R' \mid c R' i$$

$$\rightarrow c (R' p I \mid R' \mid R' i)$$

$$\rightarrow c R' (\underbrace{p I \mid i \mid \Delta}_x)$$

$$\underbrace{\hspace{10em}}_y$$

$$\rightarrow c Y$$

$$Y \rightarrow R' X \rightarrow c R' X \mid X \rightarrow c Y \mid p I \mid i \mid \Delta$$

$$\text{On pose } z \rightarrow \Delta$$

$$\text{d'où } Y \rightarrow c Y \mid p I \mid i \mid z \mid \Delta$$

Pour l'instant, nous avons

$$N \rightarrow cY$$

$$Y \rightarrow cY \mid pI \mid iZ \mid \Lambda$$

$$Z \rightarrow \Lambda$$

$$I \rightarrow Ri \rightarrow c \underbrace{R'iz}_A \rightarrow cA$$

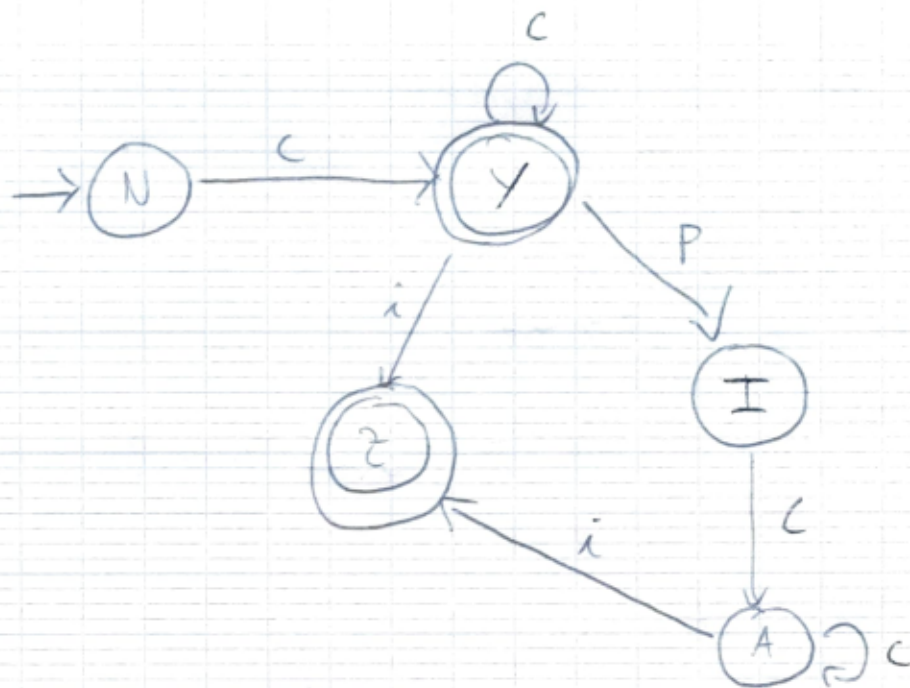
$$A \rightarrow R'iz \rightarrow c \underbrace{R'iz}_A \mid iz$$

$$A \rightarrow cA \mid iz$$

Enfinement:

$$\left\{ \begin{array}{l} N \rightarrow cY \\ Y \rightarrow cY \\ Y \rightarrow pI \\ Y \rightarrow iZ \\ Y \rightarrow \Lambda \\ I \rightarrow cA \\ A \rightarrow cA \mid iz \\ Z \rightarrow \Lambda \end{array} \right.$$

2)



Exercice 5

1)

(a) G_0 est récursive à gauche à cause de la règle (3)

$$\boxed{G_1} \begin{cases} O \rightarrow \{ \} \\ O \rightarrow \{ L \} \\ L \rightarrow F L' \\ L' \rightarrow , F L' \\ F \rightarrow id : num \\ L' \rightarrow \wedge \end{cases}$$

par la suite nous noterons ' $\{$ ' et ' $\}$ ' pour les symboles terminaux ' $\{$ ' et ' $\}$ ' pour éviter toute confusion dans les ensembles...

(b)

$$SD(G \rightarrow \{ \}) = \{ ' \}$$

$$SD(G \rightarrow \{ L \}) = \{ ' \}$$

$$SD(L' \rightarrow FL') = First(F) = id \quad (\text{car } \perp \notin First(F))$$

$$SD(L' \rightarrow , FL') = \{ , \}$$

$$SD(L' \rightarrow \perp) = Follow(L') = Follow(L) = \{ ' \}$$

$$SD(F \rightarrow id : num) = \{ id \}$$

(c) G_1 n'est pas LL(1) car $SD(G \rightarrow \{ \}) \cap SD(G \rightarrow \{ L \}) = \{ ' \} \neq \emptyset$

G_2

$$\begin{cases} G \rightarrow \{ X \\ X \rightarrow \{ \\ X \rightarrow L \} \\ L \rightarrow FL' \\ L' \rightarrow , FL' \\ L' \rightarrow \perp \\ F \rightarrow id : num \end{cases}$$

de X nous ne recalculerons que les symboles directs de G et (les autres état inchangés):

$$SD(G \rightarrow \{ X \}) = \{ ' \}$$

$$SD(X \rightarrow \{ \}) = \{ ' \}$$

$$SD(X \rightarrow L \}) = First(L) = First(F) = \{ id \}$$

Cette fois G_2 est bien $LL(1)$ puisque l'intersection des symboles directs des symboles non terminaux de G_2 est vide.

2)

(a) La construction modulaire dans la programmation des analyseurs descendants récursifs permet d'éviter les lourdeurs que l'on aurait sans (pattern matching imbriqués, gestion des cas d'erreur, etc.)

(b) * L'analyseur lexical permet de découper une chaîne de caractères en petits bouts appelés tokens. Cela (stream)

va nous permettre de reconnaître des patterns dans l'analyseur syntaxique. Une erreur est levée si un "mot" n'est pas conforme à au moins l'une des règles du lexique :

syntaxique

* L'analyseur nous permet de vérifier que les tokens reconnus par l'analyseur lexical respectent bien certaines règles de production. C'est à dire que les tokens se trouvent dans le bon ordre vis-à-vis des règles imposées. Si le stream ne suit pas les règles de production, une erreur est levée.

(c) Dans les analyseurs syntaxiques, les actions sont appelées à : vérifier qu'un token est reconnu par la règle en cours, (s'il non, lever une erreur), le passer à une autre règle si le premier symbole est non-terminal ou consommer le symbole s'il match avec le premier symbole terminal.