

## ~ Seminar 1 ~

**alfabet** = mulțime finită și nevidă de simboluri

**cuvânt** peste un alfabet  $\Sigma$  = secvență finită de simboluri din  $\Sigma$

**limbaj formal** peste un alfabet  $\Sigma$  = mulțime (finită sau infinită) de **cuvinte** (deci *lungimea* cuvintelor este finită, dar numărul de cuvinte din limbaj poate fi infinit)

$|w|$  = lungimea cuvântului  $w$  (numărul de simboluri) (**ex:**  $|\lambda| = 0$ ,  $|abac| = 4$ )

$|w|_a$  = numărul de apariții ale simbolului  $a$  în cuvântul  $w$  (**ex:**  $|01101|_0 = 2$ ,  $|cbcb|_a = 0$ )

### ➤ Operații cu limbaje

**Reuniune**  $L = L_1 \cup L_2$

$$L = \{w \mid w \in L_1 \text{ sau } w \in L_2\}$$

*Exemplu:*

$$L_1 = \{a, ab, abc\}$$

$$L_2 = \{a, bd, cd\}$$

$$\rightarrow L = \{a, ab, abc, bd, cd\}$$

**Concatenare**  $L = L_1 \cdot L_2$

$$L = \{w_i w_j \mid w_i \in L_1 \text{ și } w_j \in L_2\} \text{ (concatenarea NU este comutativă !!)}$$

*Exemplu:*

$$L_1 = \{a, ab, abc\}$$

$$L_2 = \{a, bd, cd\}$$

$$\rightarrow L = \{aa, abd, acd, aba, abbd, abcd, abca, abcbcd, abcccd\}$$

▲ **Stelare**  $L = L_1^* = \bigcup_{n \geq 0} (L_1^n) = \{\lambda\} \cup L_1 \cup (L_1)^2 \cup (L_1)^3 \cup \dots$

$L_1^0 = \{\lambda\}$  (cuvântul vid, de lungime 0)

$L_1^n = \{w_1 w_2 \dots w_n \mid w_i \in L_1 \text{ pentru orice } 1 \leq i \leq n\}$

Exemplu:

$L_1 = \{a, ab, abc\}$

$\rightarrow (L_1)^* = \{\lambda; a, ab, abc; aa, aab, aabc, aba, abab, ababc, abca, abcab, abcabc; \dots\}$

**Obs:** Cuvântul vid  $\lambda$  va aparține limbajului stelat, indiferent dacă înainte aparținea sau nu limbajului inițial.

**Ridicare la putere nenulă**  $L = L_1^+ = \bigcup_{n \geq 1} (L_1^n)$

La fel ca la stelare, doar că nu mai există cuvântul vid.  $L^+ = L^* \setminus \{\lambda\}$

## ➤ Definiție DFA și mod de funcționare

**Automat Finit Determinist (AFD) / Deterministic Finite Automaton (DFA)**

DFA = (Q,  $\Sigma$ ,  $q_0$ , F,  $\delta$ )

Q mulțimea de stări (mulțime finită și nevidă)

$\Sigma$  alfabetul de intrare (mulțime finită de simboluri)

$q_0 \in Q$  starea inițială

F  $\subseteq$  Q mulțimea de stări finale

$\delta : Q \times \Sigma \rightarrow Q$  funcția de tranziție („delta”)

- Algoritm care verifică dacă un cuvânt este sau nu acceptat de un automat DFA:
  - Pornim din starea inițială și avem cuvântul de intrare.
  - La fiecare pas, ...
  - Algoritmul se termină în 3 cazuri: ...
- Pentru a scrie pas cu pas verificarea acceptării/respingerii unui cuvânt de un DFA folosim “**configurații**” (sau “descrieri instantanee”) ale automatului, adică perechi formate din starea curentă și ce a mai rămas de citit din cuvântul de intrare.
 
$$(r_0, a_1 a_2 a_3 \dots a_n) \vdash (r_1, a_2 a_3 \dots a_n) \vdash (r_2, a_3 \dots a_n) \vdash \dots \vdash (r_n, \lambda), r_0 = q_0, r_n \in F$$
- Limbajul acceptat de un DFA numit M:
 
$$L(M) = \{w \mid (q_0, w) \vdash^* (p, \lambda), p \in F\} \text{ sau}$$

$$L(M) = \{w \mid \hat{\delta}(q_0, w) \in F\} \text{ (unde } \hat{\delta} : Q \times \Sigma^* \rightarrow Q \text{ este funcția de tranziție extinsă la cuvinte)}$$

## ➤ Definiție NFA și mod de funcționare

**Automat Finit Nedeterminist (AFN) / Nondeterministic Finite Automaton (NFA)**

$NFA = (Q, \Sigma, q_0, \delta, F)$

$Q$  mulțimea de stări (mulțime finită și nevidă)

$\Sigma$  alfabetul de intrare (mulțime finită de simboluri)

$q_0 \in Q$  starea inițială

$F \subseteq Q$  mulțimea de stări finale

$\delta: Q \times \Sigma \rightarrow 2^Q$  funcția de tranziție („delta”)

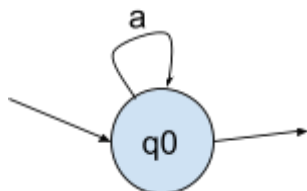
- Limbajul acceptat de un NFA numit  $M$ :  
 $L(M) = \{w \mid (q_0, w) \vdash^* \{(p_1, \lambda), \dots, (p_k, \lambda)\}, \{p_1, \dots, p_k\} \cap F \neq \emptyset\}$   
sau  $L(M) = \{w \mid \delta(q_0, w) \cap F \neq \emptyset\}$

## ➤ Desenați DFA / NFA pentru limbajele date

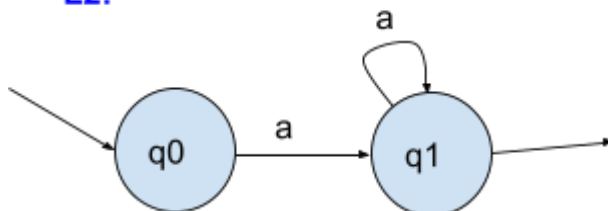
$L1 = a^* = \{a^n, n \geq 0\} = \{\lambda = a^0, a = a^1, aa = a^2, aaa = a^3, aaaa = a^4, \dots\}$

$L2 = a^+ = \{a^n, n \geq 1\} = \{a = a^1, aa = a^2, aaa = a^3, aaaa = a^4, \dots\}$

**L1:**



**L2:**



Când folosim în automat: buclă / ciclu / stări noi ?

$$L3 = \{a, b, c\}^* = \{\dots ??\}$$

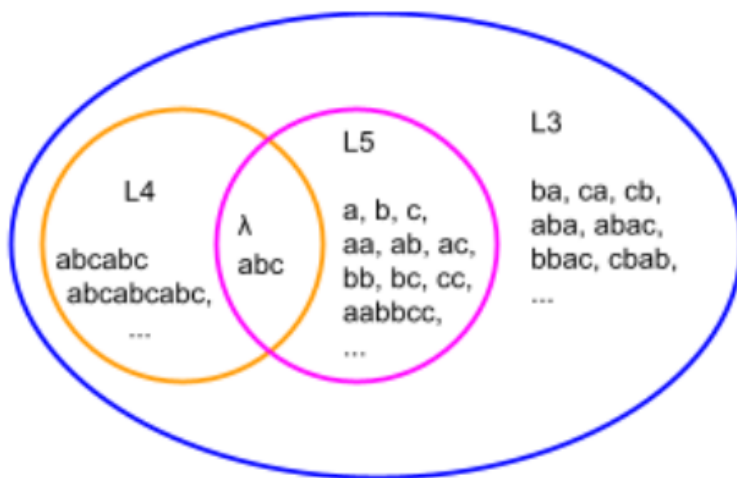
versus

$$L4 = (abc)^* = \{\dots ??\}$$

versus

$$L5 = \{a^n b^k c^p \mid n \geq 0, k \geq 0, p \geq 0\} = \{\dots ??\}$$

Există vreo relație de incluziune / intersecție între unele dintre limbajele L3, L4, L5 ?



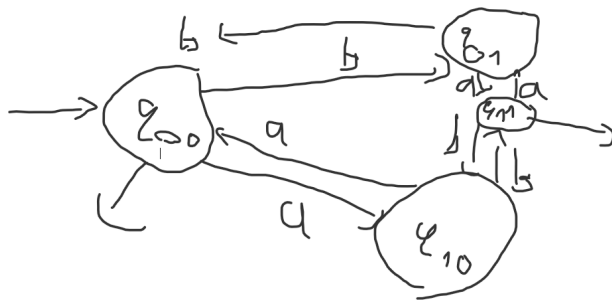
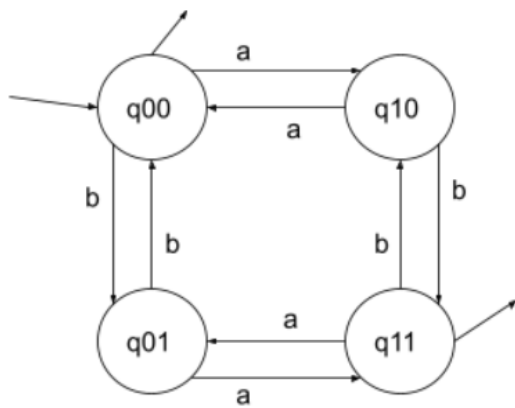
**Temă de gândire:**

$L = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$  (același număr de a-uri și b-uri)

versus

$L = \{w \in \{a, b\}^* \mid |w|_a \bmod 2 = |w|_b \bmod 2\}$  (aceeași paritate pt numărul de a-uri și b-uri)

- Putem să desenăm un DFA? Dar un NFA?
- Puteți să dați un exemplu de limbaj pentru care putem desena un NFA, dar nu un DFA?



### Temă ??

$$L14 = \{a^{2n}b^{2k+1} \mid n \geq 0, k \geq 0\} = \{...??\}$$

versus

$$L15 = \{w \in \{a, b\}^* \mid |w|_a = 2n, |w|_b = 2k + 1, n \geq 0, k \geq 0\} = \{...??\}$$

$$L16 = \{w \in \{a, b\}^* \mid |w|_a \bmod 2 \neq 0, |w|_b \bmod 3 \neq 0\}$$

Câte stări va avea automatul? Care vor fi starea inițială și cele finale?

$$L17 = \{w \in \{a, b\}^* \mid aaa \text{ și } bbb \text{ NU apar ca subșiruri în } w\}$$

(Adică literele a și b sunt amestecate în cuvânt astfel încât nicio literă nu apare de mai mult de 2 ori consecutiv în cuvânt.)

$$L18(X) = \{w \in \{0, 1, \dots, 9\}^* \mid w \text{ reprezintă un număr în baza 10 divizibil cu } X\},$$

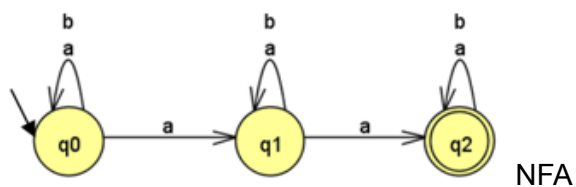
unde  $X \in \{3, 25, 4\}$  (câte un automat pt fiecare valoare a lui X).

## ~ Seminar 2 ~

### ➤ Algoritm: Transformarea AFN → AFD

#### EXEMPLU:

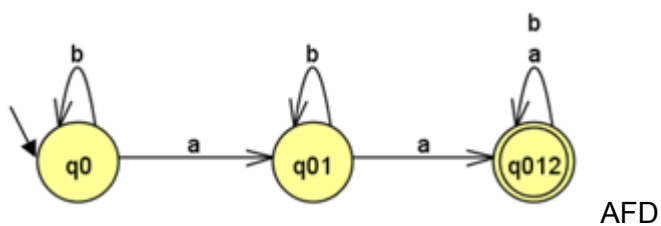
Se dă AFN-ul următor:



- Completăm tabel\_1 cu funcția de tranziție pentru AFN.
- Completăm tabel\_2 cu funcția de tranziție pentru AFD (pornim din starea inițială a AFN-ului și adăugăm pe rând stările obținute în interiorul tabel\_2).
- Desenăm graful pentru AFD conform tabel\_2.

$\delta_{\text{AFN}}$	a	b
q0 init	{q0, q1} = q01	{q0}
q1	{q1, q2} = q12	{q1}
q2 in F	{q2}	{q2}

$\delta_{\text{AFD}}$	a	b
q0 init	q01	q0
q01	q012	q01
q012 in F	q012	q012



### ➤ Verificare acceptare cuvânt pentru AFD, AFN

Care este limbajul recunoscut de cele două automate din exemplul de mai sus?

Verificați (pentru AFD, apoi pentru AFN) dacă cuvintele **bba** și **babbaba** sunt acceptate sau respinse, folosind configurații.

--- AFD, cuv bba

$(q_0, bba) \vdash^b (q_0, ba) \vdash^b (q_0, a) \vdash^a (q_0, \lambda), q_0 \notin F \Rightarrow bba \text{ respins}$

--- AFN, cuv bba

$(q_0, bba) \vdash^b (q_0, ba) \vdash^b (q_0, a) \vdash^a \{(q_0, \lambda), (q_1, \lambda)\}, \{q_0, q_1\} \text{ intersectat } F = \text{mult vida} \Rightarrow bba \text{ respins}$

--- AFD, cuv babbaba

$(q_0, babbaba) \vdash^b (q_0, abbaba) \vdash^a (q_0, bbaba) \vdash^b (q_0, baba) \vdash^b (q_0, aba) \vdash^a (q_0, \lambda), q_0 \in F \Rightarrow babbaba \text{ acceptat}$

--- AFN, cuv babbaba

$(q_0, babbaba) \vdash^b (q_0, abbaba) \vdash^a \{(q_0, bbaba), (q_1, bbaba)\}$   
 $\vdash^b \{(q_0, baba), (q_1, baba)\} \vdash^b \{(q_0, aba), (q_1, aba)\} \vdash^a \{(q_0, ba), (q_1, ba), (q_2, ba)\}$   
 $\vdash^b \{(q_0, a), (q_1, a), (q_2, a)\} \vdash^a \{(q_0, \lambda), (q_1, \lambda), (q_2, \lambda)\},$   
 $\{q_0, q_1, q_2\} \text{ intersectat } F = \{q_2\} \text{ (diferit de mult vida)} \Rightarrow babbaba \text{ acceptat}$

### ➤ Automat Finit Nedeterminist cu $\lambda$ -tranziii

$AFN-\lambda = (Q, \Sigma, q_0, \delta, F)$

$Q$  mulțimea de stări

$\Sigma$  alfabetul de intrare

$q_0 \in Q$  starea inițială

$F \subseteq Q$  mulțimea de stări finale

$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$  funcția de tranziție („delta”)

Diferența față de AFN-ul simplu este că suntem într-o stare și putem citi fie un caracter din alfabetul  $\Sigma$ , fie cuvântul vid  $\lambda$ . Deci se poate întâmpla să ajungem într-o stare nouă fără să fi citit nicio literă nouă din cuvântul de intrare, ci doar aplicând  $\lambda$ -tranziii.

### ➤ λ-închiderea unei stări

Mulțimea de stări în care se poate ajunge plecând din starea  $q$  și aplicând zero sau mai multe  $\lambda$ -tranziții se numește „ $\lambda$ -închiderea stării  $q$ ” și se notează cu  $\langle q \rangle$ .

**Obs:** Orice stare face parte din propria  $\lambda$ -închidere (pentru că  $\delta(q, \lambda^0) = q$ ; practic putem presupune că orice stare are o  $\lambda$ -tranziție *implicită* către ea însăși).

$$\langle q \rangle = \bigcup_{k \geq 0} \{r \mid r \in \hat{\delta}(q, \lambda^k)\}$$

$$\langle q \rangle = \{q\} \cup \{q_i \mid q_i \in \hat{\delta}(q, \lambda^1)\} \cup \{q_{ij} \mid q_{ij} \in \hat{\delta}(q, \lambda^2)\} \cup \dots$$

Observăm că mulțimile se pot calcula inductiv după puterea lui  $\lambda$ :

$$\{q_{ij} \mid q_{ij} \in \hat{\delta}(q, \lambda^2)\} = \{q_{ij} \mid q_i \in \hat{\delta}(q, \lambda^1), q_{ij} \in \delta(q_i, \lambda^1)\}.$$

Sau în general  $\{r \mid r \in \delta(q, \lambda^k)\} = \{r \mid s \in \hat{\delta}(q, \lambda^{k-1}), r \in \delta(s, \lambda^1)\}.$

### ➤ Verificare acceptare cuvânt de către automat AFN-λ

Pentru a verifica dacă un cuvânt este sau nu acceptat de un automat AFN-λ:

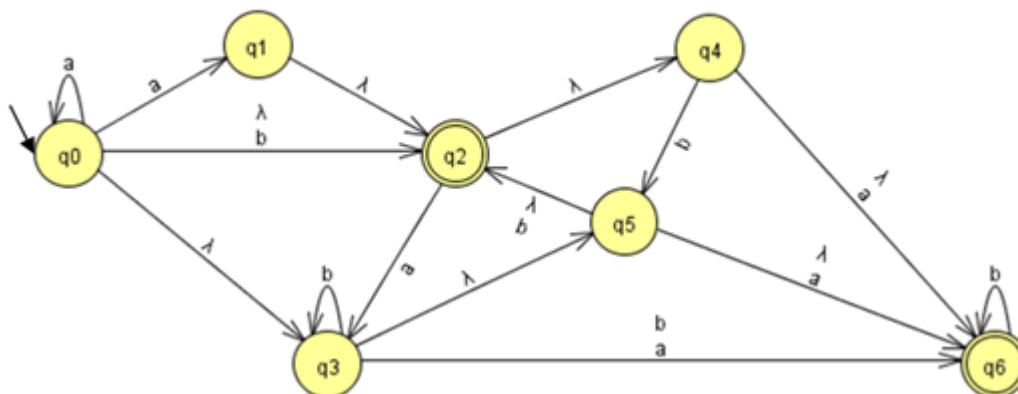
Se procedează analog ca în cazul AFN-ului, doar că înainte de a căuta toate stările posibile de continuare cu tranziții cu caracterul curent, trebuie să facem  $\lambda$ -închiderea mulțimii curente de stări. Iar după ce a fost citit tot cuvântul de intrare, trebuie să facem o ultimă  $\lambda$ -închidere a stărilor curente, pentru a obține mulțimea finală de stări în care poate ajunge automatul pentru cuvântul dat.

**Obs:**  $\lambda$ -închiderea unei mulțimi de stări este egală cu reuniunea  $\lambda$ -închiderilor acelor stări.

$$\langle \{q_{i1}, q_{i2}, \dots, q_{in}\} \rangle = \langle q_{i1} \rangle \cup \langle q_{i2} \rangle \cup \dots \cup \langle q_{in} \rangle$$

### EXEMPLU:

Se dă următorul AFN-λ:



a) Calculăm  $\lambda$ -închiderile tuturor stărilor.

$$\langle q_0 \rangle = \{q_0, q_2, q_3, q_4, q_5, q_6\}$$

$$\langle q_1 \rangle = \{q_1, q_2, q_4, q_6\}$$



$\langle q_2 \rangle = \{q_2, q_4, q_6\}$   
 $\langle q_3 \rangle = \{q_3, q_5, q_2, q_6, q_4\}$   
 $\langle q_4 \rangle = \{q_4, q_6\}$   
 $\langle q_5 \rangle = \{q_5, q_2, q_4, q_6\}$   
 $\langle q_6 \rangle = \{q_6\}$

b) Verificăm dacă cuvântul **abbaa** este acceptat sau respins de acest AFN- $\lambda$ , folosind configurații.

$(q_0, abbaa) \vdash^{\lambda^*} (q_{023456}, abbaa) \vdash^a (q_{0136}, bbaa) \vdash^{\lambda^*} (q_{0123456}, bbaa) \vdash^b$   
 $(q_{2365}, baa) \vdash^{\lambda^*} (q_{23456}, baa) \vdash^b (q_{3652}, aa) \vdash^{\lambda^*} (q_{23456}, aa) \vdash^a$   
 $(q_{36}, a) \vdash^{\lambda^*} (q_{23456}, a) \vdash^a (q_{36}, \lambda) \vdash^{\lambda^*} (q_{23456}, \lambda)$   
 $\{q_2, q_3, q_4, q_5, q_6\} \cap F = \{q_2, q_6\}$  (diferit de multimea vidă)  $\Rightarrow$  abbaa acceptat

**EX:** Desenați un AFN / AFD pentru limbajele următoare.

$$L1 = \{a^{3k+2} \mid k \geq 1\}$$

$$= \{a^5, a^8, a^{11}, a^{14}, \dots\}$$

$$L2 = \{w \in \{a, b, c\}^* \mid (|w|_b \bmod 5) \text{ este impar}\}$$

$$L3 = \{waaa \mid w \in \{a, b\}^*\}$$

$$L4 = \{xababy \mid x, y \in \{a, b\}^*\}$$

## ~ Seminar 3 ~

➤ *Algoritm:* Transformarea AFN- $\lambda \rightarrow$  AFN [vezi curs 3, pag 16 – 19]

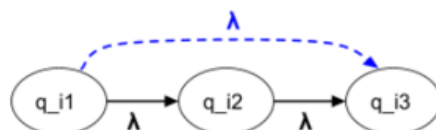
Se dă un automat AFN- $\lambda$ . Se cere să se construiască un automat AFN echivalent.

**Idee:** Adăugăm tranziții cu litere din alfabet și stări finale astfel încât să simulăm comportamentul  $\lambda$ -tranzițiilor, pe care apoi le eliminăm.

➔ **Pas 1 („ $\lambda$ -completion”):**

- Cât timp e posibil:

$$\forall q_{i_1}, q_{i_2}, q_{i_3} \in Q, \text{daca } \delta(q_{i_1}, \lambda) \ni q_{i_2} \text{ si } (q_{i_2}, \lambda) \ni q_{i_3} \Rightarrow (q_{i_1}, \lambda) \ni q_{i_3}$$



Altfel spus, din fiecare stare  $q$  trebuie să **adăugăm** (dacă nu există deja) câte o  **$\lambda$ -tranziție** către fiecare stare  $r$  (cu  $r \neq q$ ) din mulțimea  $\langle q \rangle$  ( $\lambda$ -închiderea stării  $q$ ).

(Adică dacă aveam de la starea  $q$  la starea  $r$  un drum format din două sau mai multe  $\lambda$ -tranziții, atunci trebuie să adăugăm o  $\lambda$ -tranziție directă de la  $q$  la  $r$ .)

- Apoi trebuie să **adăugăm la mulțimea stărilor finale** acele stări din care cu  $\lambda$  ajungem într-o stare finală.



(Adică stările care conțineau în  $\lambda$ -închiderea lor cel puțin o stare finală vor deveni și ele stări finale.)

➔ **Pas 2 („ $\lambda$ -transition removal”):**

- Adăugăm tranziții cu litere din alfabet care să înlocuiască efectul  $\lambda$ -tranzițiilor, astfel:

$$\forall q_{i_1}, q_{i_2}, q_{i_3} \in Q, \forall a \in \Sigma, \text{daca } \delta(q_{i_1}, \lambda) \ni q_{i_2} \text{ si } (q_{i_2}, a) \ni q_{i_3} \Rightarrow (q_{i_1}, a) \ni q_{i_3}$$

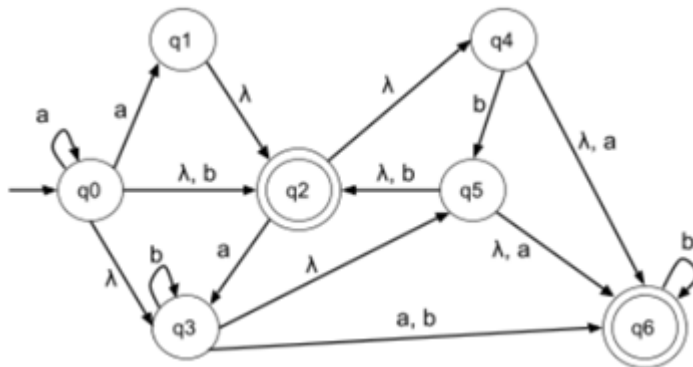


Adică dacă din starea  $q$  puteam ajunge în starea  $r$  citind  $\lambda a$  ( $\forall a \in \Sigma$ ), atunci **adăugăm o tranziție cu litera  $a$**  direct de la  $q$  la  $r$ .

- Apoi **eliminăm toate  $\lambda$ -tranzițiile** din automat.

- **Exemplu:** Se dă următorul AFN- $\lambda$ .

(Același din seminarul 2, pentru care calculasem deja  $\lambda$ -închiderile tuturor stărilor.)



-> Pas 1 („ $\lambda$ -completion”):

$\langle q_0 \rangle = \{q_0, q_2, q_3, q_4, q_5, q_6\} \Rightarrow$  adaug  $\lambda$  de la  $q_0$  spre  $q_4, q_5, q_6$

$\langle q_1 \rangle = \{q_1, q_2, q_4, q_6\} \Rightarrow$  adaug  $\lambda$  de la  $q_1$  spre  $q_4, q_6$

$\langle q_2 \rangle = \{q_2, q_4, q_6\} \Rightarrow$  adaug  $\lambda$  de la  $q_2$  spre  $q_6$

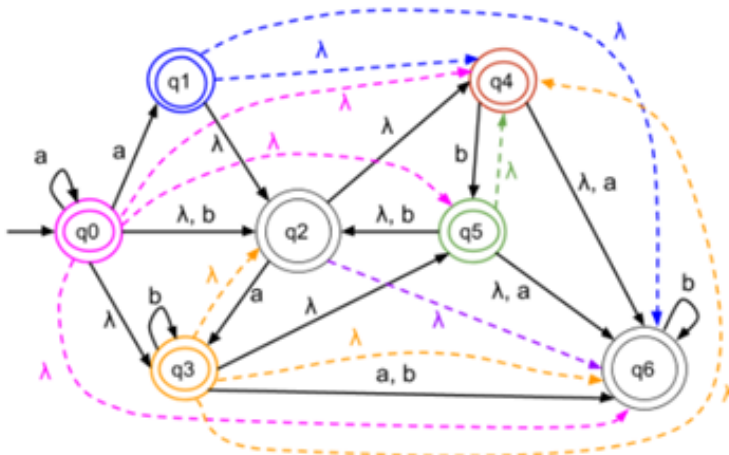
$\langle q_3 \rangle = \{q_3, q_5, q_2, q_6, q_4\} \Rightarrow$  adaug  $\lambda$  de la  $q_3$  spre  $q_2, q_4, q_6$

$\langle q_4 \rangle = \{q_4, q_6\} \Rightarrow$  nu adaug nimic

$\langle q_5 \rangle = \{q_5, q_2, q_4, q_6\} \Rightarrow$  adaug  $\lambda$  de la  $q_5$  spre  $q_4$

$\langle q_6 \rangle = \{q_6\} \Rightarrow$  nu adaug nimic

Toate stările au în  $\lambda$ -închiderile lor cel puțin o stare finală, deci toate stările vor deveni finale.

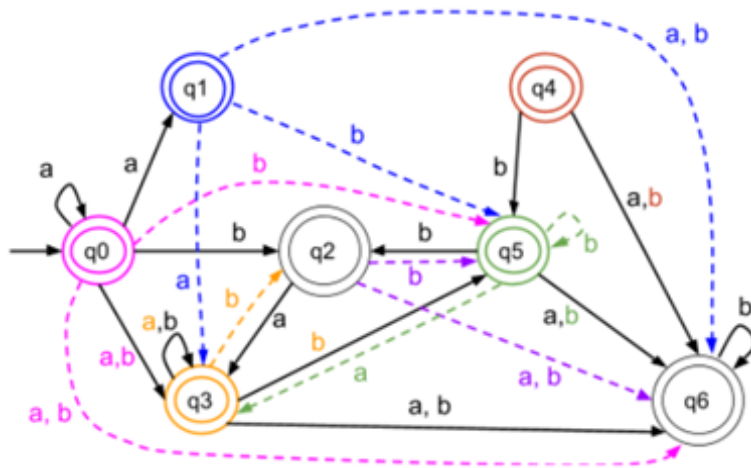


-> Pas 2 („ $\lambda$ -transition removal”):

- Eliminăm  $\delta(q_0, \lambda) \ni q_2$ , adăugăm  $\delta(q_0, a) \ni q_3$ .
- Eliminăm  $\delta(q_0, \lambda) \ni q_3$ , adăugăm  $\delta(q_0, a) \ni q_6$  și  $\delta(q_0, b) \ni \{q_3, q_6\}$ .
- Eliminăm  $\delta(q_0, \lambda) \ni q_4$ , adăugăm  $\delta(q_0, a) \ni q_6$  (era deja) și  $\delta(q_0, b) \ni q_5$ .
- Eliminăm  $\delta(q_0, \lambda) \ni q_5 \Rightarrow \delta(q_0, a) \ni q_6$  și  $\delta(q_0, b) \ni q_2$  (ambele existau deja).
- Eliminăm  $\delta(q_0, \lambda) \ni q_6$ , adăugăm  $\delta(q_0, b) \ni q_6$  (era deja).
  
- Eliminăm  $\delta(q_1, \lambda) \ni q_2$ , adăugăm  $\delta(q_1, a) \ni q_3$ .
- Eliminăm  $\delta(q_1, \lambda) \ni q_4$ , adăugăm  $\delta(q_1, a) \ni q_6$  și  $\delta(q_1, b) \ni q_5$ .
- Eliminăm  $\delta(q_1, \lambda) \ni q_6$ , adăugăm  $\delta(q_1, b) \ni q_6$ .
  
- Eliminăm  $\delta(q_2, \lambda) \ni q_4$ , adăugăm  $\delta(q_2, a) \ni q_6$  și  $\delta(q_2, b) \ni q_5$ .
- Eliminăm  $\delta(q_2, \lambda) \ni q_6$ , adăugăm  $\delta(q_2, b) \ni q_6$ .
  
- Eliminăm  $\delta(q_3, \lambda) \ni q_2$ , adăugăm  $\delta(q_3, a) \ni q_3$  și  $\delta(q_3, b) \ni q_5$ .
- Eliminăm  $\delta(q_3, \lambda) \ni q_4$ , adăugăm  $\delta(q_3, a) \ni q_6$  și  $\delta(q_3, b) \ni q_5$  (ambele erau deja).
- Eliminăm  $\delta(q_3, \lambda) \ni q_5 \Rightarrow \delta(q_3, a) \ni q_6$  (era deja) și  $\delta(q_3, b) \ni q_2$ .
- Eliminăm  $\delta(q_3, \lambda) \ni q_6$ , adăugăm  $\delta(q_3, b) \ni q_6$  (era deja).
  
- Eliminăm  $\delta(q_4, \lambda) \ni q_6$ , adăugăm  $\delta(q_4, b) \ni q_6$ .
  
- Eliminăm  $\delta(q_5, \lambda) \ni q_2$ , adăugăm  $\delta(q_5, a) \ni q_3$ .
- Eliminăm  $\delta(q_5, \lambda) \ni q_4$ , adăugăm  $\delta(q_5, a) \ni q_6$  și  $\delta(q_5, b) \ni \{q_5, q_6\}$  ( $q_6$  era deja).
- Eliminăm  $\delta(q_5, \lambda) \ni q_6$ , adăugăm  $\delta(q_5, b) \ni q_6$ .
  
- Din  $q_6$  nu pleacă nicio  $\lambda$ -tranziție, deci nu avem ce elimina / adăuga.

Am obținut un AFN echivalent cu AFN- $\lambda$  dat.

(Observăm că starea  $q_4$  nu este accesibilă din starea inițială, deci ar putea fi eliminată împreună cu trazițiile ei fără a afecta limbajul recunoscut de automat.)



➤ **Algoritm:** Transformarea  $AFN-\lambda \rightarrow AFD$  [asemănător  $AFN \rightarrow AFD$ , seminar 2]

**Idee:** Dacă în  $AFN-\lambda$  din starea  $q$  citind  $\lambda^* a \lambda^*$  ( $\forall a \in \Sigma$ ) se ajunge în mulțimea de stări  $R$ , atunci în  $AFN$  din starea  $q$  citind litera  $a$  se va ajunge în mulțimea de stări  $R$ .

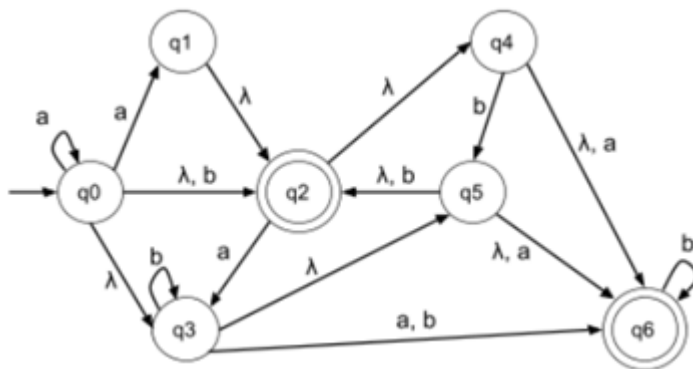
Starea inițială pentru  $AFN$  este aceeași ca la  $AFN-\lambda$ .

Stările finale ale  $AFN$ -ului sunt cele din care se putea ajunge cu  $\lambda$ -drumuri într-o stare finală.

**Obs:** Dacă dorim să obținem  $AFD$ , atunci **starea inițială din  $AFD$  este  $\lambda$ -închiderea stării inițiale din  $AFN-\lambda$** . Stările finale ale  $AFD$ -ului sunt cele care aveau în  $\lambda$ -închidere cel puțin o stare finală.

• **Exemplu:** Se dă următorul  $AFN-\lambda$ .

(Același din seminarul 2, pentru care calculasem deja  $\lambda$ -închiderile tuturor stărilor.)



$\delta_{AFN-\lambda}$	<b>a</b>	<b>b</b>	$\lambda$	$\lambda^*$ ( $\lambda$ -închiderea)
<b>q0 init</b>	{q0, q1}	{q2}	{q2, q3}	<q0> = {q0, q2, q3, q4, q5, q6}
<b>q1</b>	$\emptyset$	$\emptyset$	{q2}	<q1> = {q1, q2, q4, q6}
<b>q2 in F</b>	{q3}	$\emptyset$	{q4}	<q2> = {q2, q4, q6}
<b>q3</b>	{q6}	{q3, q6}	{q5}	<q3> = {q3, q5, q2, q6, q4}
<b>q4</b>	{q6}	{q5}	{q6}	<q4> = {q4, q6}
<b>q5</b>	{q6}	{q2}	{q2, q6}	<q5> = {q5, q2, q4, q6}
<b>q6 in F</b>	$\emptyset$	{q6}	$\emptyset$	<q6> = {q6}

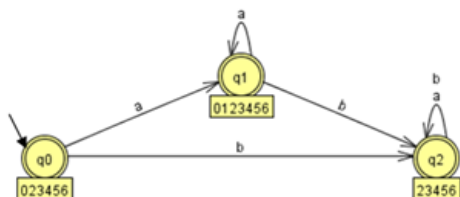
în AFN- $\lambda$ :	$\lambda^* a \lambda^*$	$\lambda^* b \lambda^*$
<b>q0 init</b>	$q_0 \xrightarrow{\lambda^*} q_{023456} \xrightarrow{a} q_{0136} \xrightarrow{\lambda^*} q_{0123456}$	$q_0 \xrightarrow{\lambda^*} q_{023456} \xrightarrow{b} q_{0356} \xrightarrow{\lambda^*} q_{23456}$
<b>q1</b>	$q_1 \xrightarrow{\lambda^*} q_{1246} \xrightarrow{a} q_{36} \xrightarrow{\lambda^*} q_{23456}$	$q_1 \xrightarrow{\lambda^*} q_{1246} \xrightarrow{b} q_{56} \xrightarrow{\lambda^*} q_{2456}$
<b>q2 in F</b>	$q_2 \xrightarrow{\lambda^*} q_{246} \xrightarrow{a} q_{36} \xrightarrow{\lambda^*} q_{23456}$	$q_2 \xrightarrow{\lambda^*} q_{246} \xrightarrow{b} q_{56} \xrightarrow{\lambda^*} q_{2456}$
<b>q3</b>	$q_3 \xrightarrow{\lambda^*} q_{23456} \xrightarrow{a} q_{36} \xrightarrow{\lambda^*} q_{23456}$	$q_3 \xrightarrow{\lambda^*} q_{23456} \xrightarrow{b} q_{2356} \xrightarrow{\lambda^*} q_{23456}$
<b>q4</b>	$q_4 \xrightarrow{\lambda^*} q_{46} \xrightarrow{a} q_6 \xrightarrow{\lambda^*} q_6$	$q_4 \xrightarrow{\lambda^*} q_{46} \xrightarrow{b} q_{56} \xrightarrow{\lambda^*} q_{2456}$
<b>q5</b>	$q_5 \xrightarrow{\lambda^*} q_{2456} \xrightarrow{a} q_{36} \xrightarrow{\lambda^*} q_{23456}$	$q_5 \xrightarrow{\lambda^*} q_{2456} \xrightarrow{b} q_{256} \xrightarrow{\lambda^*} q_{2456}$
<b>q6 in F</b>	$q_6 \xrightarrow{\lambda^*} q_6 \xrightarrow{a} \emptyset \xrightarrow{\lambda^*} \emptyset$	$q_6 \xrightarrow{\lambda^*} q_6 \xrightarrow{b} q_6 \xrightarrow{\lambda^*} q_6$

$\delta_{AFN}$	<b>a</b>	<b>b</b>
<b>q0 init, in F</b>	$q_{0123456} = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$
<b>q1 in F</b>	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$	$q_{2456} = \{q_2, q_4, q_5, q_6\}$
<b>q2 in F</b>	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$	$q_{2456} = \{q_2, q_4, q_5, q_6\}$
<b>q3 in F</b>	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$
<b>q4 in F</b>	$\{q_6\}$	$q_{2456} = \{q_2, q_4, q_5, q_6\}$
<b>q5 in F</b>	$q_{23456} = \{q_2, q_3, q_4, q_5, q_6\}$	$q_{2456} = \{q_2, q_4, q_5, q_6\}$
<b>q6 in F</b>	$\emptyset$	$\{q_6\}$

$\delta_{AFD}$	<b>a</b>	<b>b</b>
<b>&lt;q0&gt; = q023456 init, in F</b>	q0123456	q23456
<b>q0123456 in F</b>	q0123456	q23456
<b>q23456 in F</b>	q23456	q23456

Am obținut un AFD echivalent cu automatele AFN și AFN- $\lambda$  de mai sus.

Ce limbaj recunoaște acest AFD?



➤ **Lema de pompare pentru limbaje regulate (REG)** [vezi curs 5, pag 19 – 21]

Fie  $L$  un limbaj regulat. Atunci  $\exists p \in \mathbb{N}$  (număr natural) astfel încât pentru  $\forall \alpha \in L$  cuvânt, cu  $|\alpha| \geq p$  și există o descompunere  $\alpha = u \cdot v \cdot w$  cu proprietățile:

(1)  $|u \cdot v| \leq p$

(2)  $|v| \geq 1$

(3)  $u \cdot v^i \cdot w \in L, \forall i \geq 0$ .

- Vrem să demonstrăm că un limbaj NU este regulat.

Presupunem prin reducere la absurd că “limbajul este regulat” (predicatul  $P$ ) și atunci rezultă că “afirmația din lema este adevărată” (predicatul  $Q$ ).

**Obs:** Știm de la logică faptul că  $(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$ . Așa că vom nega afirmația lemei  $(\neg Q)$  și va rezulta că limbajul nu este regulat  $(\neg P)$ . Practic negarea constă în interschimbarea cuantificatorilor logici ( $\exists$  și  $\forall$ ) între ei, iar la condiția (3)  $\in$  devine  $\notin$ .

→ **Schema demonstrației**

Alegem (adică  $\exists$ ) un cuvânt  $\alpha$  din limbajul  $L$  dat, care să respecte ipoteza lemei anterioare, adică să aibă lungimea cel puțin  $p$ ,  $\forall p \in \mathbb{N}$ , iar descompunerea sa  $\alpha = u \cdot v \cdot w$  să respecte condițiile (1) și (2). Apoi alegem convenabil un număr natural  $i$  pentru care să obținem o contradicție a condiției (3), adică să rezulte că cuvântul  $\beta = u \cdot v^i \cdot w \notin L$  și deci presupunerea făcută este falsă.

$L = \{a^m b^n \mid m > n \geq 0\}$  not in REG

**Demonstratie:**

Presupunem prin reducere la absurd ca  $L$  este in REG. Rezulta p nr natural din lema.  
(Incepem sa negam lema)

Alegem  $\alpha = a^{p+1} b^p$  in  $L \Rightarrow |\alpha| = 2p+1 \geq p$ , pt orice p nr nat.

Cuv  $\alpha$  se decompune in  $\alpha = uvw$  a.i.  $|uv| \leq p$  si  $|v| \geq 1$  (deci  $1 \leq |v| \leq p$ )

$\Rightarrow uv$  contin doar a-uri, deci notam  $v = a^k \Rightarrow |v|=k$ .

Din cond (1) si (2)  $\Rightarrow 1 \leq k \leq p$  (rel A).

Pt cuv pompat alegem  $i = 0 \Rightarrow \beta = u v^0 w = a^{p+1-k} b^p$  in  $L \Leftrightarrow$

$p+1-k > p \Leftrightarrow 1 > k$  contradictie cu (rel A).

Deci  $L$  nu e REG.

## ~ Seminar 4 ~

### ➤ Lema de pompare pentru limbaje regulate (REG) [continuare]

#### • Exemple:

$L4 = \{w \cdot w^R \mid w \in \{a, b\}^*\} \notin REG$  (discutat alegerea lui alpha)

$L5 = \{w \cdot w \mid w \in \{a, b\}^*\} \notin REG$  (discutat alegerea lui alpha)

Pt L4,  $\alpha = a^p b b a^p$  DA

$\alpha = a^p a^p = a^{(2p)} = (aa)^p$  NU e o alegere buna !

Pt L5,  $\alpha = a^p b a^p b$

$L6 = \{a^{n^2} \mid n \geq 1\} = \{a^1, a^4, a^9, a^{16}, a^{25}, a^{36}, \dots\} \notin REG$

Presupunem prin reducere la absurd ca L6 este REG. Atunci exista p nr natural din lema.

Alegem cuvântul  $\alpha = a^{(p^2)}$  in L6  $\Rightarrow |\alpha| = p^2 \geq p$ , pt orice p nr natural.

Avem  $\alpha = uvw$  a.i.  $|uv| \leq p$  si  $|v| \geq 1$  (deci  $1 \leq |v| \leq p$ ).

Notam  $v = a^k \Rightarrow |v| = k \Rightarrow 1 \leq k \leq p$  (\*).

Alegem  $i = 2 \Rightarrow \beta = u v^2 w$  de lungime

$|\beta| = |u v^2 w| = |u v w| + |v| = |\alpha| + |v| = p^2 + k$ .

Din (\*) stim ca  $1 \leq k \leq p$  (adunam  $p^2$ )  $\Leftrightarrow p^2 + 1 \leq p^2 + k \leq p^2 + p$

Dar  $p^2 < p^2 + 1$  si  $p^2 + p < (p+1)^2$

$\Rightarrow p^2 < p^2 + k < (p+1)^2 \Leftrightarrow p^2 < |\beta| < (p+1)^2 \Rightarrow$  deci  $|\beta|$  nu poate fi patrat perfect

$\Rightarrow \beta$  nu apartine L6, contradictie cu conditia (3) din lema  $\Rightarrow L6$  nu e in REG.

### ➤ Expresii regulate (RegEx)

**Definiție:** Se numește familia expresiilor regulate peste  $\Sigma$  și se notează  $RegEx(\Sigma)$  mulțimea de cuvinte peste alfabetul  $\Sigma \cup \{ (, ), +, \cdot, *, \emptyset, \lambda \}$  definită recursiv astfel:

- i)  $\emptyset, \lambda \in RegEx$  și  $a \in RegEx, \forall a \in \Sigma$ .
- ii) Dacă  $e_1, e_2 \in RegEx$ , atunci  $(e_1 + e_2) \in RegEx$ . (reuniune)
- iii) Dacă  $e_1, e_2 \in RegEx$ , atunci  $(e_1 \cdot e_2) \in RegEx$ . (concatenare)
- iv) Dacă  $e \in RegEx$ , atunci  $(e^*) \in RegEx$ . (stelare)

- **Precedența operațiilor:**  $( ) > * > \cdot > +$  (paranteze > stelare > concatenare > reuniune)

**Obs:** În evaluarea unei expresii regulate se ține cont în primul rând de paranteze, iar apoi ordinea în care se evaluează operațiile este: stelare, apoi concatenare, apoi reuniune.

(Dacă vrei să fii siguri că nu le încurcați, puteți să faceți o analogie cu operațiile aritmetice, unde se evaluează întâi ridicarea la putere, apoi înmulțirea și apoi adunarea.)



- Reamintim din seminarul 1:

$$L = L_1 \cup L_2 = \{w \mid w \in L_1 \text{ sau } w \in L_2\}$$



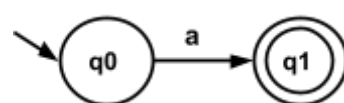
$$L = L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ și } w_2 \in L_2\}$$

$$L = (L_1)^* = \bigcup_{n \geq 0} \{w_1 w_2 \dots w_n \mid w_i \in L_1, \forall 1 \leq i \leq n\}$$

### ➤ Algoritm: Transformarea RegEx → AFN-λ

Pentru fiecare caz din definiția RegEx vom construi câte un automat finit echivalent.

#### Caz i)

RegEx	$e = \emptyset$	$e = \lambda$	$e = a$ , unde $a \in \Sigma$
Limbaj	$L = \emptyset$	$L = \{\lambda\}$	$L = \{a\}$
Automat Finit			

În **cazurile ii), iii) și iv)** presupunem că

pentru expresia regulată  $e_k$  și limbajul  $L(e_k)$ ,  $k \in \{1, 2\}$  avem deja

automate finite  $AF(L(e_k)) = (Q_k, \Sigma_k, q_{0k}, F_k, \delta_k)$ , cu  $Q_1 \cap Q_2 = \emptyset$  (stări disjuncte).

Desenăm schema unui automat punând în evidență starea inițială  $q_{0k}$  și mulțimea stărilor finale  $F_k$ . Dreptunghiul  $M_k$  include toate celelalte stări și tranzițiile automatului.



Vom construi automatele pentru operațiile de reuniune, concatenare și stelare.

<b>Caz ii)</b> <i>(reuniune)</i>  <b>RegEx</b> $e = e_1 + e_2$  <b>Limbaj</b> $L(e) = L(e_1) \cup L(e_2)$	<b>Automat Finit</b> $AF(L(e_1 + e_2)) = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, q_0, F_1 \cup F_2, \delta_1 \cup \delta_2 \cup \{\delta(q_0, \lambda) = \{q_{01}, q_{02}\}\})$
--	--

<b>Caz iii)</b> <i>(concatenare)</i>  <b>RegEx</b> $e = e_1 \cdot e_2$  <b>Limbaj</b> $L(e) = L(e_1) \cdot L(e_2)$	<b>Automat Finit</b> $AF(L(e_1 \cdot e_2)) = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, q_{01}, F_2, \delta_1 \cup \delta_2 \cup \{\delta(q_{f_1}, \lambda) \ni q_{02} \mid \forall f_1 \in F_1\})$ 
<b>Caz iv)</b> <i>(stelare)</i>  <b>RegEx</b> $e = (e_1)^*$  <b>Limbaj</b> $L(e) = (L(e_1))^*$	<b>Automat Finit</b> $AF(L((e_1)^*)) = (Q_1 \cup \{q_0\}, \Sigma_1, q_0, F_1 \cup \{q_0\}, \delta_1 \cup \{\delta(q_0, \lambda) = q_{01}\} \cup \{\delta(q_{f_1}, \lambda) \ni q_{01} \mid \forall f_1 \in F_1\})$ 

- Exemplu:** Desenați 3 automate finite pentru  $L_1 = a^*$ ,  $L_2 = bc^*$ ,  $L_3 = ac$ , apoi folosind algoritmi pentru reuniune, concatenare, stelare și ținând cont de paranteze și de ordinea operațiilor, desenați automatul pentru  $L_4 = (a^* + bc^*) \cdot (ac)^* = (L_1 + L_2) \cdot (L_3)^*$

(desen pe whiteboard)

➤ *Algoritm:* Transformarea AFN- $\lambda \rightarrow$  RegEx

**Definiție:** Se numește *AFE (automat finit extins)*,  $M = (Q, \Sigma, et, q_0, F)$ , unde, la fel ca la celelalte automate finite,  $Q$  este mulțimea stărilor,  $\Sigma$  este alfabetul,  $q_0$  este starea inițială,  $F$  este mulțimea stărilor finale. Aici avem *funcția de etichetare*  $et : Q \times Q \rightarrow \text{RegEx}(\Sigma)$ .

Notăm  $et(p, q)$  prin  $e_{pq}$ .

**Ideea algoritmului** este de a transforma automatul finit într-un automat finit extins, și apoi a elimina una câte una stările până ajungem la o expresie regulată echivalentă cu automatul inițial.

• **Algoritm:**

**Pas 1:** Transformăm automatul finit dat într-un AFE: dacă de la starea  $q_x$  către starea  $q_y$  există mai multe tranziții, atunci le înlocuim cu *expresia regulată* obținută prin reunirea (operatorul “+”) simbolurilor de pe acele tranziții.

$$et(q_x, q_y) = \{w \in \text{RegEx}(\Sigma) \mid w = a_1 + a_2 + \dots + a_n ; \\ q_y \in \delta(q_x, a_i), a_i \in (\Sigma \cup \{\lambda\}), \forall i \in \{1, \dots, n\}\}$$

**Pas 2:** Dacă există săgeți care ajung către starea inițială, atunci se adaugă la automat o nouă stare care va fi inițială și va avea o săgeată cu expresia  $\lambda$  către fosta stare inițială.

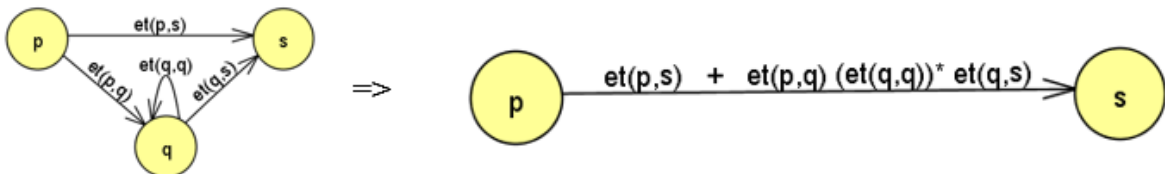
**Pas 3:** Dacă există mai multe stări finale sau dacă există săgeți care pleacă din vreo stare finală, atunci se adaugă la automat o nouă stare care va fi singura finală și va avea săgeți cu expresia  $\lambda$  din toate fostele stări finale către ea.

**Pas 4:** În orice ordine, se elimină pe rând, una câte una, toate stările în afară de cea inițială și cea finală. Presupunem că vrem să eliminăm starea  $q_k$  și că există etichetele

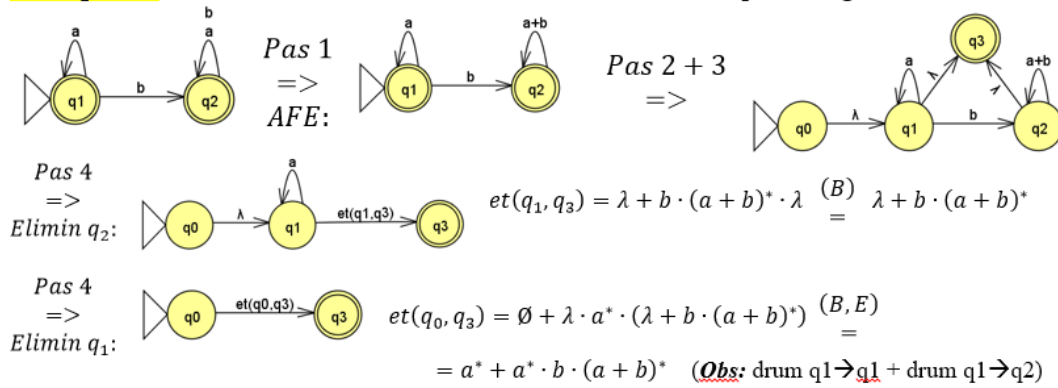
$et(q_i, q_k)$ ,  $et(q_k, q_j)$  și eventual bucla  $et(q_k, q_k)$ . Atunci obținem noua etichetă între stările  $q_i$  și  $q_j$  reunind [(fosta etichetă directă de la  $q_i$  la  $q_j$ ), sau nimic ( $\emptyset$ ) dacă nu există drum direct] cu [(eticheta de la  $q_i$  la  $q_k$ ) concatenată cu (stela etichetei buclei de la  $q_k$  la  $q_k$ , sau  $\lambda$  dacă bucla nu există) concatenată cu (eticheta de la  $q_k$  la  $q_j$ )]. (Vezi desenul de mai jos.)

**Pas 5:** Atunci când rămân doar două stări, expresia obținută între starea inițială și cea finală este răspunsul final (o expresie regulată echivalentă cu automatul finit dat).

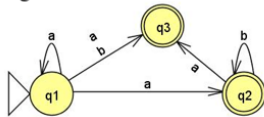
→ Vrem să eliminăm  $q$  și avem un  $p$  (“predecesor”) și un  $s$  („succesor”).



**Exemplul 2:** Să se transforme următorul automat finit într-o expresie regulată echivalentă.



**EX 5:** Transformați următorul automat finit într-o expresie regulată echivalentă, folosind algoritmul de mai sus.



## ~ Seminar 5 ~

### ➤ *Algoritm: Minimizare AFD (metoda cu teorema Myhill-Nerode)*

Se dă un AFD complet definit (adică din fiecare stare din mulțimea  $Q$  pleacă câte o tranziție cu fiecare simbol din alfabetul  $\Sigma$ ). Se cere să se construiască un AFD echivalent (care să accepte același limbaj) care să aibă un număr minim de stări.

**Obs:** Dacă AFD-ul dat **nu** este complet definit, atunci pentru completarea lui se adaugă o *nouă stare nefinală*  $q_{aux}$ . Toate tranzițiile lipsă din celelalte stări se adaugă spre această nouă stare, apoi pentru starea  $q_{aux}$  se adaugă o buclă cu toate simbolurile din alfabet.

**Ideea** algoritmului este de a găsi acele stări care au comportament echivalent, pentru a le grupa și a obține o unică stare nouă în locul acestora.

- Două stări sunt „**echivalente**” dacă pentru orice cuvânt am alege, plecând din cele două stări, fie ajungem în două stări finale, fie ajungem în două stări nefinale.  
$$\forall p, q \in Q, p \equiv q \Leftrightarrow [\forall w \in \Sigma^*, \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F]$$
- Două stări sunt „**separabile**” dacă există un cuvânt pentru care plecând din cele două stări ajungem într-o stare finală și într-una nefinală.  
$$\forall p, q \in Q, p \not\equiv_w q \Leftrightarrow [\exists w \in \Sigma^*, \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \notin F]$$

### Algoritm:

- Vom construi un tabel, pe linii și pe coloane având stările automatului AFD complet definit. Vom completa tabelul (triunghiul de sub diagonala principală) pentru fiecare pereche de stări  $(q_i, q_j)$ , având  $i > j$ , cu un cuvânt prin care cele două stări sunt *separabile*. Dacă nu vom găsi un astfel de cuvânt atunci acele stări sunt echivalente.
- Vom completa tabelul căutând cuvintele recursiv, în ordinea crescătoare a lungimii lor. Cuvintele de lungime  $k$  le vom obține cu ajutorul celor de lungime  $k-1$  calculate anterior.

→ **Pas 0:** Două stări sunt *separabile* prin cuvântul *vid*  $\lambda$  (de lungime zero), dacă una din ele este finală și cealaltă este nefinală în automatul dat.

→ Dacă la pasul **k-1** s-a marcat în tabel cel puțin o pereche de stări separabile, atunci **Repetăm Pas k:** *(Pentru k luând valori de la 1 la maxim cât ?)*

a) Pentru **fiecare** pereche de stări  $(q_i, q_j)$ , cu  $i > j$ , care **nu** a fost încă marcată ca fiind separabilă prin niciun cuvânt, verificăm:

b) pentru **fiecare** simbol  $x$  din alfabetul  $\Sigma$ :

c) **dacă** plecând din perechea de stări  $(q_i, q_j)$ , cu  $i > j$ , și aplicând tranzițiile cu simbolul  $x$  din alfabet ajungem în perechea de stări  $(q_s, q_t)$ , cu  $s > t$ , iar stările  $q_s$  și  $q_t$  erau marcate în tabel ca fiind separabile prin cuvântul  $w$ ,

**atunci** rezultă că stările  $q_i$  și  $q_j$  sunt separabile prin cuvântul  $xw$  și le marcăm în tabel cu acest cuvânt. **Stop pas b) și continuăm pas a).**

**b') Dacă** nu s-a găsit niciun simbol  $x$  care să ne ducă într-o pereche de stări separabile (adică *toate* simbolurile din alfabet ne-au dus fie în perechi de stări nemarcate încă în tabel, fie în perechi de stări identice ( $q_s = q_t$ )),

**atunci** perechea  $(q_i, q_j)$  rămâne *momentan* nemarcată în tabel și **continuăm pas a).**

**a')** Dacă la aplicarea pasului curent  $k$  am marcat cel puțin o pereche de stări separabile în tabel, **atunci** incrementăm valoarea lui  $k$  și repetăm acest pas (adică vom căuta stări separabile prin cuvinte de lungime  $k+1$ ).

**Altfel** (dacă nu s-a modificat nimic în tabel la pasul  $k$ ), **algoritmul se termină** cu concluzia că stările rămase **nemarcate** în tabel sunt stări echivalente.

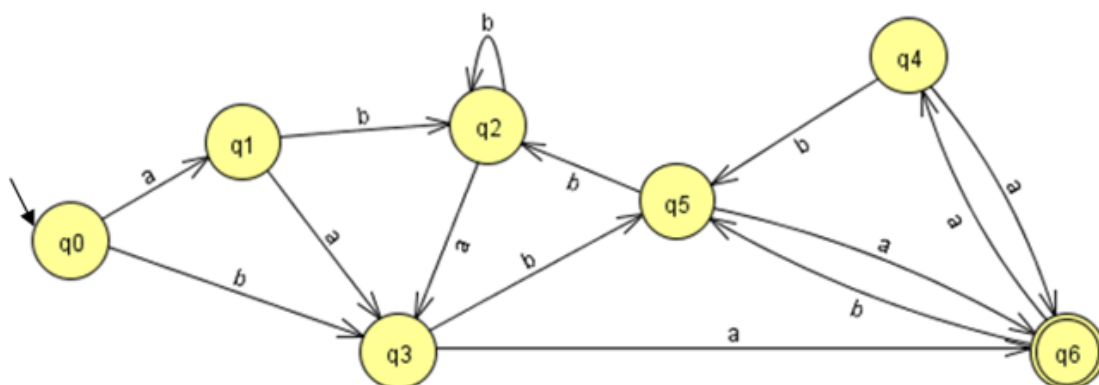
- Apoi desenăm AFD-ul minimal astfel:
  - Desenăm întâi toate stările, grupându-le pe cele echivalente între ele într-o singură stare a AFD-ului minimal.
  - **Starea inițială** este grupul format din stările echivalente cu fosta stare inițială  $q_0$ .
  - **Stările finale** sunt grupurile formate din stări care erau finale în automatul original.
  - Pentru a duce **tranzițiile**, dacă în automatul original aveam tranziție din starea  $q$  cu simbolul  $x$  către starea  $r$ , atunci în AFD-ul minimal, din grupul de stări echivalente cu starea  $q$  ducem tranziție cu simbolul  $x$  către grupul de stări echivalente cu starea  $r$ .

### Observații:

Înainte de aplicarea algoritmului descris mai sus, **eliminăm** din automat toate **stările inaccesibile** (cele până la care nu există niciun drum care pleacă din starea inițială) împreună cu toate tranzițiile care pleacă sau ajung în ele.

După aplicarea algoritmului descris mai sus, **eliminăm** toate stările plecând din care nu se poate ajunge în nicio stare finală, împreună cu tranzițiile care pleacă sau ajung în ele. *De exemplu* va fi eliminată acea stare nefinală  $q_{aux}$  adăugată pentru completarea AFD-ului.

- **Exemplu:** Pentru următorul AFD construiți un AFD minimal echivalent.



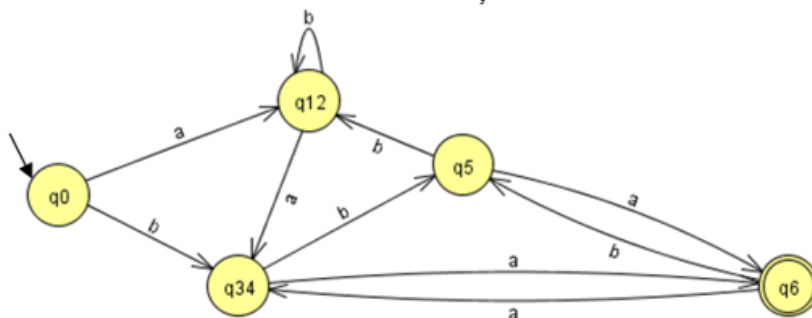
Observăm că AFD-ul dat este complet definit și nu există stări inaccesibile.

	q0	q1	q2	q3	q4	q5	q6
q0							
q1	aa						
q2	aa	mult vida					
q3	a	a	a				
q4	a	a	a	mult vida			
q5	a	a	a	ba	ba		
q6	$\lambda$	$\lambda$	$\lambda$	$\lambda$	$\lambda$	$\lambda$	

Am terminat de completat tabelul și am obținut stări echivalente:  $q_1 \equiv q_2$  și  $q_3 \equiv q_4$ .

Automatul AFD minimal obținut va avea stările  $Q = \{q_0, q_{12}, q_{34}, q_5, q_6\}$ , starea inițială  $q_0$  și stările finale  $F = \{q_6\}$ . Tranzițiile le desenăm conform automatului inițial, dar ținând cont de această grupare a stărilor.

Observăm că nu există nicio stare plecând din care să nu avem drum până într-o stare finală, deci nu avem de eliminat nicio stare și acesta este automatul minimal echivalent cu cel dat.



## ➤ Definiție APD și mod de funcționare

### **Automat Push-Down (APD) / Push-Down Automaton (PDA)**

$APD = (Q, \Sigma, q_0, F, \delta, \Gamma, Z_0)$

$Q$  mulțimea de stări (mulțime finită și nevidă)

$\Sigma$  alfabetul de intrare („sigma”) (mulțime finită)

$q_0 \in Q$  starea inițială

$F \subseteq Q$  mulțimea de stări finale

$\Gamma$  alfabetul stivei („gamma”) (mulțime finită și nevidă)

$Z_0 \in \Gamma$  simbolul inițial al stivei

$\delta: Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$  funcția de tranziție („delta”)



Modul în care funcționează tranzițiile pentru un APD

- La intrare avem nevoie de 3 parametri:
  - starea curentă (*element din mulțimea Q*)
  - caracterul curent din cuvântul de intrare (*element din mulțimea  $\Sigma$* ) sau cuvântul vid  $\lambda$
  - **simbolul** aflat în vârful stivei (*element din mulțimea  $\Gamma$* )
- La ieșire vom avea 2 parametri:
  - starea în care ajungem (*element din mulțimea Q*)
  - **cuvântul** cu care înlocuim **simbolul** din vârful stivei (*element din mulțimea  $\Gamma^*$* )

**Atenție!** Simbolul din vârful stivei este mereu *eliminat* din stivă, înainte de a se adăuga noul cuvânt în locul lui.

### • Modalități de acceptare a unui cuvânt de către un APD

#### (a) cu stări finale

$$L_F(APD) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \lambda, \alpha), p \in F, \alpha \in \Gamma^*\}$$

#### (b) cu vidarea stivei

$$L_\lambda(APD) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \lambda, \lambda), p \in Q\}$$

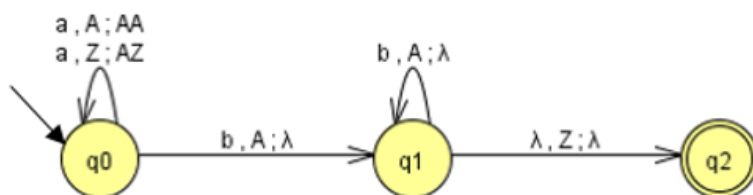
#### (c) cu stări finale și cu vidarea stivei

$$L_{F \& \lambda}(APD) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \lambda, \lambda), p \in F\}$$

#### Obs:

- 1) În general vom folosi *litere mari pentru alfabetul stivei*, pentru a fi mai clară diferența între caracterele citite din cuvântul de intrare și cele din stivă.
- 2) La seminar vom folosi a 3-a metodă de acceptare (*cu stări finale și vidarea stivei*).

- **Exemplu rezolvat:**  $L1 = \{a^n b^n \mid n \geq 1\}$  (discutat și la curs)





- **Verificare acceptare cuvânt de către un APD**

Folosim „configurații” (sau „descrieri instantanee”) având 3 componente: starea curentă, ce a mai rămas de citit din cuvântul de intrare și (întreg!) conținutul stivei (partea stângă fiind vârful stivei, unde se fac ștergerile și scrierile).

La fiecare pas, căutăm pe graf tranziția care

- pleacă din starea curentă,
- citește prima literă din cuvântul curent (sau eventual  $\lambda$ ) și
- citește din stivă primul caracter din cuvântul care reprezintă întreg conținutul stivei

=> în configurația următoare :

- punem starea spre care a ajuns acea tranziție,
- prima literă din cuvânt dispare dacă a fost citită (sau rămâne dacă s-a citit  $\lambda$ ),
- iar vârful stivei se înlocuiește cu cuvântul spus de tranziție (cel de după “/” sau “;”), iar restul stivei se concatenează după.

**Exemplu:** Fie  $n = 3$  deci avem cuvântul  $a^3b^3$ . (pt graful de mai sus)

$(q_0, a^3b^3Z_0) \vdash (q_0, a^2b^3, AZ_0) \vdash (q_0, ab^3, AAZ_0) \vdash (q_0, b^3, AAAZ_0)$

$\vdash (q_1, b^2, AAZ_0) \vdash (q_1, b, AZ_0) \vdash (q_1, \lambda, Z_0) \vdash (q_2, \lambda, \lambda), q_2 \in F \Rightarrow a^3b^3$  cuvânt acceptat.

## ~ Seminar 5' ~ (continuare)

### ➤ Gramatici / Grammars

$G = (N, T, S, P)$

$N = \{A, B, C, \dots\}$  mulțimea de simboluri **neterminale** (mulțime finită și nevidă)

$T = \{a, b, c, \dots\}$  mulțimea de simboluri **terminale** (mulțime finită)

$S \in N$  **simbolul de start**

$P$  mulțimea de **producții** (sau reguli de producție)

**Obs:**  $T = \Sigma$  (alfabetul peste care sunt definite cuvintele *generate* de gramatică)

### ❖ Gramatici regulate / Regular Grammars

Au producții de forma:  $A \rightarrow aB \mid a \mid \lambda$ , unde  $A, B \in N, a \in T$

- Echivalența dintre Gramaticile regulate și Automatele finite

Gramatici regulate	Automate finite
$N$ = mulțimea de simboluri <b>neterminale</b>	$Q$ = mulțimea de <b>stări</b>
$T$ = mulțimea de simboluri <b>terminale</b>	$\Sigma$ = <b>alfabetul</b> de intrare
$S \in N$ <b>simbolul de start</b>	$q_0 \in Q$ <b>starea inițială</b>
- producția $A \rightarrow aB$ , cu $A, B \in N, a \in T$	- tranziția $\delta(q_A, a) = q_B$ , cu $q_A, q_B \in Q, a \in \Sigma$
- producția $A \rightarrow a$ , cu $A \in N, a \in T$	- tranziția $\delta(q_A, a) = q_f$ , cu $q_A \in Q, a \in \Sigma, q_f \in F$
- producția $A \rightarrow \lambda$ , cu $A \in N$	- tranziția $\delta(q_A, \lambda) = q_f$ , cu $q_A \in Q, q_f \in F$ (sau direct <i>stare finală</i> $q_A \in F$ )

### ❖ Gramatici independente de context (GIC) / Context-free Grammars (CFG)

Au producții de forma:  $P \subseteq N \times (N \cup T)^*$

## ~ Seminar 6 ~

### ➤ Forma Normală Chomsky

O gramatică este (scrisă) în **forma normală Chomsky** dacă are doar producții care au în membrul stâng un neterminal, iar în membrul drept fie un terminal, fie două neterminale.

$$A \rightarrow a$$

$$A \rightarrow BC, \text{ unde } A, B, C \in N, a \in T$$

În plus, dacă cuvântul vid  $\lambda$  trebuie generat de gramatică, atunci avem voie să avem și producția  $S \rightarrow \lambda$ , dar atunci  $S$  nu are voie să apară în *membrul drept* al niciunei producții.

### • Algoritm: Transformarea GIC → gramatică scrisă în F.N. Chomsky

- **Pas 1:** Se aplică **algoritmul de reducere** (se elimină simbolurile și producțiile nefolositoare):
  - a) Se elimină simbolurile și producțiile “**neutilizabile**”.  
**Def:**  $X \in (N \cup T)$  este un simbol “neutilizabil” dacă nu există nicio derivare  $\alpha X \beta \Rightarrow^* \alpha w \beta$  cu  $\alpha, w, \beta \in T^*$ .  
(Adică plecând de la un cuvânt care conține simbolul  $X$  și aplicând oricât de multe producții, nu putem ajunge să generăm un cuvânt care să nu conțină neterminale.)
  - b) Se elimină simbolurile și producțiile “**inaccesibile**”.  
**Def:**  $X \in (N \cup T)$  este un simbol “inaccesibil” dacă nu există nicio derivare  $S \Rightarrow^* \alpha X \beta$ , cu  $\alpha, \beta \in (N \cup T)^*$ .  
(Adică plecând din  $S$  și aplicând oricât de multe producții, nu putem ajunge să generăm un cuvânt care să conțină simbolul  $X$ .)

#### Exemplu:

$$S \rightarrow aABa \mid CD \mid bbAC$$

$$A \rightarrow bc \mid d \mid \lambda$$

$$B \rightarrow \lambda \mid E$$

$$C \rightarrow A \mid dcabb \mid S$$

$$D \rightarrow BAd \mid B$$

$$E \rightarrow Ea \mid bbE$$

$$F \rightarrow abc$$

a) Pastrăm  $A, B, C, F; S, D$ . Eliminăm  $E$  și toate producțiile lui.

$$S \rightarrow aABa \mid CD \mid bbAC$$

$$A \rightarrow bc \mid d \mid \lambda$$

$$B \rightarrow \lambda$$

$$C \rightarrow A \mid dcabb \mid S$$

$$D \rightarrow BAd \mid B$$

$$F \rightarrow abc$$

b) Pastram S; A, B, C, D. Eliminam F si toate productiile lui.

$S \rightarrow aABa \mid CD \mid bbAC$

$A \rightarrow bc \mid d \mid \lambda$

$B \rightarrow \lambda$

$C \rightarrow A \mid dcabb \mid S$

$D \rightarrow BAd \mid B$

▪ **Pas 2:** Se elimină  $\lambda$ -producțiile.

**Obs:** Dacă  $\lambda \in L(G)$  (cuvântul vid trebuie să fie generat de gramatică), atunci avem voie să avem unica  $\lambda$ -producție a neterminalului de start, dar acesta nu are voie să mai apară în gramatică nicăieri (în membrul drept al producțiilor).

→ Se adaugă **un nou simbol de start**  $S'$  și producțiile  $S' \rightarrow S \mid \lambda$ .

Vrem să nu mai avem în gramatică producții care au ca membru drept cuvântul vid.

Elimin  $B \rightarrow \lambda$ .

$S \rightarrow aAa \mid CD \mid bbAC$

$A \rightarrow bc \mid d \mid \lambda$

$C \rightarrow A \mid dcabb \mid S$

$D \rightarrow Ad \mid \lambda$

Elimin  $D \rightarrow \lambda$ .

$S \rightarrow aAa \mid CD \mid C \mid bbAC$

$A \rightarrow bc \mid d \mid \lambda$

$C \rightarrow A \mid dcabb \mid S$

$D \rightarrow Ad$

Elimin  $A \rightarrow \lambda$ .

$S \rightarrow aAa \mid aa \mid CD \mid C \mid bbAC \mid bbC$

$A \rightarrow bc \mid d$

$C \rightarrow A \mid \lambda \mid dcabb \mid S$

$D \rightarrow Ad \mid d$

Elimin  $C \rightarrow \lambda$ .

$S \rightarrow aAa \mid aa \mid CD \mid D \mid C \mid \lambda \mid bbAC \mid bbA \mid bbC \mid bb$

$A \rightarrow bc \mid d$

$C \rightarrow A \mid dcabb \mid S$

$D \rightarrow Ad \mid d$

Elimin  $S \rightarrow \lambda$ . Adaug  $S'$  noul simbol de start.

$S' \rightarrow S \mid \lambda$

$S \rightarrow aAa \mid aa \mid CD \mid D \mid C \mid bbAC \mid bbA \mid bbC \mid bb$

$A \rightarrow bc \mid d$

$C \rightarrow A \mid dcabb \mid S$

$D \rightarrow Ad \mid d$

▪ **Pas 3:** *Se elimină redenumirile.*

Vrem să nu mai avem în gramatică producții care au ca membru drept un neterminal.

Înlocuim neterminalul din dreapta cu toate cuvintele care sunt membru drept în producțiile sale.

Verificăm dacă acest neterminal din dreapta mai apare în dreapta în cadrul altor producții

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid CD \mid D \mid C \mid bbAC \mid bbA \mid bbC \mid bb \\ A &\rightarrow bc \mid d \\ C &\rightarrow A \mid dcabb \mid S \\ D &\rightarrow Ad \mid d \end{aligned}$$

Eliminam  $C \rightarrow A$  (Dacă  $A \rightarrow$  cuvinte, adaug  $C \rightarrow$  cuvinte).

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid CD \mid D \mid C \mid bbAC \mid bbA \mid bbC \mid bb \\ A &\rightarrow bc \mid d \\ C &\rightarrow bc \mid d \mid dcabb \mid S \\ D &\rightarrow Ad \mid d \end{aligned}$$

Eliminam  $S \rightarrow D$  (Dacă  $D \rightarrow$  cuvinte, adaug  $S \rightarrow$  cuvinte).

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid CD \mid Ad \mid d \mid C \mid bbAC \mid bbA \mid bbC \mid bb \\ A &\rightarrow bc \mid d \\ C &\rightarrow bc \mid d \mid dcabb \mid S \\ D &\rightarrow Ad \mid d \end{aligned}$$

Eliminam  $S \rightarrow C$  (Dacă  $C \rightarrow$  cuvinte, adaug  $S \rightarrow$  cuvinte).

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid CD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid \textcolor{red}{S} \mid bbAC \mid bbA \mid bbC \mid bb \\ A &\rightarrow bc \mid d \\ C &\rightarrow bc \mid d \mid dcabb \mid S \\ D &\rightarrow Ad \mid d \end{aligned}$$

Eliminam  $C \rightarrow S$  (Dacă  $S \rightarrow$  cuvinte, adaug  $C \rightarrow$  cuvinte).

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid CD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid bbAC \mid bbA \mid bbC \mid bb \\ A &\rightarrow bc \mid d \\ C &\rightarrow aAa \mid aa \mid CD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid bbAC \mid bbA \mid bbC \mid bb \\ D &\rightarrow Ad \mid d \end{aligned}$$

Observăm că  $S$  și  $C$  au exact aceleași producții, deci eliminăm  $C$  și îl înlocuim peste tot cu  $S$ .

$$\begin{aligned} S' &\rightarrow S \mid \lambda \\ S &\rightarrow aAa \mid aa \mid SD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid bbAS \mid bbA \mid bbS \mid bb \\ A &\rightarrow bc \mid d \\ D &\rightarrow Ad \mid d \end{aligned}$$

Eliminăm  $S' \rightarrow S$  (Dacă  $S \rightarrow$  cuvinte, adaug  $S' \rightarrow$  cuvinte).  
 $S' \rightarrow aAa \mid aa \mid SD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid bbAS \mid bbA \mid bbS \mid bb \mid \lambda$   
 $S \rightarrow aAa \mid aa \mid SD \mid Ad \mid d \mid bc \mid d \mid dcabb \mid bbAS \mid bbA \mid bbS \mid bb$   
 $A \rightarrow bc \mid d$   
 $D \rightarrow Ad \mid d$

- **Pas 4:** Se aplică din nou *algoritmul de reducere* (vezi Pas 1).

- **Exemplu:**

Observăm că nu avem ce modifica, nu există neterminale neutilizabile sau inaccesibile.

- **Pas 5:** Se adaugă neterminale noi pentru terminalele din cuvinte de lungime  $>1$ .

Vrem ca terminalele să apară doar singure în membrul drept. De aceea, peste tot unde apar în componența unui cuvânt de lungime minim 2, le vom înlocui cu un neterminal nou și vom adăuga producția de la neterminalul nou la terminalul pe care l-a înlocuit.

$$S' \rightarrow X_1X_1 \mid X_1AX_1 \mid AX_4 \mid X_2X_2 \mid X_2X_2A \mid X_2X_2AS \mid X_2X_2S \mid X_2X_3 \mid d \mid X_4X_3X_1X_2X_2 \mid SD \mid \lambda$$

$$S \rightarrow X_1X_1 \mid X_1AX_1 \mid AX_4 \mid X_2X_2 \mid X_2X_2A \mid X_2X_2AS \mid X_2X_2S \mid X_2X_3 \mid d \mid X_4X_3X_1X_2X_2 \mid SD$$

$$A \rightarrow X_2X_3 \mid d$$

$$D \rightarrow AX_4 \mid d$$

$$X_1 \rightarrow a \ ; \ X_2 \rightarrow b \ ; \ X_3 \rightarrow c \ ; \ X_4 \rightarrow d$$

- **Pas 6:** Se adaugă neterminale noi pentru „spargerea” cuvintelor de lungime  $>2$ .

Vrem ca în dreapta să avem cuvinte formate din exact două neterminale. De aceea, unde avem cuvinte mai lungi, păstrăm doar primul neterminal din cuvânt și îi alipim un neterminal nou, iar noul neterminal va avea o producție cu membrul drept cuvântul pe care l-a înlocuit. Reluăm procedeul până când toate cuvintele ajung la lungimea 2.

**Obs:** Fiecare producție care avea un cuvânt de lungime  $k$  va fi înlocuită de  $k-1$  producții cu cuvinte de lungime 2.

$$S' \rightarrow X_1X_1 \mid X_1Y_1 \mid AX_4 \mid X_2X_2 \mid X_2Y_2 \mid X_2Y_3 \mid X_2Y_5 \mid X_2X_3 \mid d \mid X_4Y_6 \mid SD \mid \lambda$$

$$S \rightarrow X_1X_1 \mid X_1Y_1 \mid AX_4 \mid X_2X_2 \mid X_2Y_2 \mid X_2Y_3 \mid X_2Y_5 \mid X_2X_3 \mid d \mid X_4Y_6 \mid SD$$

$$A \rightarrow X_2X_3 \mid d$$

$$D \rightarrow AX_4 \mid d$$

$$X_1 \rightarrow a \ ; \ X_2 \rightarrow b \ ; \ X_3 \rightarrow c \ ; \ X_4 \rightarrow d$$

$$Y_1 \rightarrow AX_1 \ ; \ Y_2 \rightarrow X_2A \ ; \ Y_3 \rightarrow X_2Y_4 \ ; \ Y_4 \rightarrow AS \ ; \ Y_5 \rightarrow X_2S$$

$$Y_6 \rightarrow X_3Y_7 \ ; \ Y_7 \rightarrow X_1Y_8 \ ; \ Y_8 \rightarrow X_2X_2$$

### ➤ *Lema de pompare pentru limbajele independente de context*

Fie  $L$  un limbaj independent de context.

Atunci  $\exists p \in \mathbb{N}$  (număr natural) astfel încât pentru  $\forall \alpha \in L$  cuvânt, cu  $|\alpha| \geq p$ , există o descompunere  $\alpha = u \cdot v \cdot w \cdot x \cdot y$  cu proprietățile:

- (1)  $|v \cdot w \cdot x| \leq p$
- (2)  $|v \cdot x| \geq 1$
- (3)  $u \cdot v^i \cdot w \cdot x^i \cdot y \in L, \forall i \geq 0$ .

#### → *Schema demonstrației*

- Vrem să demonstrăm că  $L$  **nu este** limbaj independent de context, folosind lema de pompare **negată**.
- Presupunem prin reducere la absurd că  $L$  **este** limbaj independent de context.  
Atunci  $\exists p \in \mathbb{N}$  și putem aplica lema de pompare. (*În continuare negăm afirmația lemei.*)
- ( $\exists$ ) Alegem un cuvânt  $\alpha$  din limbajul  $L$  astfel încât  $|\alpha| \geq p$ ,  $\forall p \in \mathbb{N}$ . *Este important ca pentru acel  $\alpha$  să NU poată fi construit un automat push-down (sau o gramatică independentă de context), pentru că altfel NU vom putea obține contradicția dorită.*
- Știm că  $\alpha = u \cdot v \cdot w \cdot x \cdot y$ . Vom presupune proprietățile (1) și (2) îndeplinite și vom găsi o contradicție pentru (3).
- ( $\forall u, v, w, x, y$ ) Trebuie să analizăm pe rând **orice** împărțire posibilă a lui  $\alpha$  în cele 5 componente (altfel spus, trebuie să poziționăm  $vw$  în toate modurile posibile în  $\alpha$ ). Pentru **fiecare** caz, trebuie să alegem ( $\exists$ ) câte un număr natural  $i$  (nu neapărat același pentru toate cazurile) astfel încât cuvântul  $\beta = u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$ , rezultând o contradicție a lemei (a proprietății (3)), deci a presupunerii că limbajul  $L$  era independent de context.  
(*Atenție, demonstrația este completă doar dacă obținem contradicție pentru fiecare posibilitate de descompunere a lui  $\alpha$ , nu doar pe unele cazuri.*)

- **Exemple:** Demonstrați că următoarele limbaje NU sunt independente de context.

$$L1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L2 = \{a^n b^m c^n d^m \mid n \geq 0, m \geq 0\}$$

$$L3 = \{z \cdot z \mid z \in \{a, b\}^*\}$$

$$L4 = \{a^n b^m c^r \mid n > m > r \geq 0\}$$

#### **Demonstratie pt L1:**

Presupunem prin reducere la absurd ca  $L1$  este CFL. Atunci exista  $p$  nr natural din lema.  
(*Incepem negarea lemei.*)

Alegem  $\alpha = a^p b^p c^p$  din  $L1 \Rightarrow |\alpha| = 3p \geq p$ , pt orice  $p$  nr natural.

Avem  $\alpha = uvwxy$  a.i.  $|vwx| \leq p$  si  $|vx| \geq 1$ .

**Caz I:**  $vx = a^k \Rightarrow 1 \leq k \leq p$  (\*).

Alegem  $i=2 \Rightarrow \beta = u v^2 w x^2 y = a^{(p+k)} b^p c^p$ , din (\*)  $\Rightarrow |\beta|_a > |\beta|_b \Rightarrow \beta$  nu este din  $L1$ , contradicție cu condiția (3) din lema (**rel 1**).

**Caz II:**  $vx = b^k$  analog caz I ...  $\Rightarrow$  (**rel 2**)

**Caz III:**  $vx = c^k$  analog caz I ...  $\Rightarrow$  (rel 3)

**Caz IV:**  $vw x$  in  $a^k b^s \Rightarrow 1 \leq k+s \leq p$ .

Alegem  $i=0 \Rightarrow \beta = u v^0 w x^0 y = uwy = a^{(p-k)} b^{(p-s)} c^p$ , din (\*)  $\Rightarrow |\beta|_a$  sau  $|\beta|_b$  sunt stricte mai putine decat  $|\beta|_c \Rightarrow \beta$  nu este din  $L1$ , contradictie cu conditia (3) din lema (rel 4).

**Caz V:**  $vw x$  in  $b^k c^s \Rightarrow$  analog caz IV ...  $\Rightarrow$  (rel 5).

Din rel 1, rel 2, rel 3, rel 4, rel 5  $\Rightarrow$  presupunerea facuta este falsa  $\Rightarrow L1$  nu esre CFL.



## ~ Seminar 7 ~