

CUPRINS

~ Seminar 1 ~	3
➤ Operații cu limbaje	3
➤ Definiție DFA și mod de funcționare	4
➤ Definiție NFA și mod de funcționare	5
~ Seminar 2 ~	8
➤ Algoritm: Transformarea AFN \rightarrow AFD	8
➤ Verificare acceptare cuvânt de către AFD, AFN	9
➤ Automat Finit Nedeterminist cu λ-tranziții	9
➤ λ -închiderea unei stări	10
➤ Verificare acceptare cuvânt de către automat AFN- λ	10
~ Seminar 3 ~	13
➤ Algoritm: Transformarea AFN- $\lambda \rightarrow$ AFN (metoda 1) [vezi curs 3, pag 28 – 30]	13
➤ Algoritm: Transformarea AFN- $\lambda \rightarrow$ AFD (metoda 2) [asemănător AFN \rightarrow AFD, seminar 2]	16
➤ Lema de pompare pentru limbaje regulate (REG) [vezi curs 5, pag 19 – 21]	17
~ Seminar 4 ~	20
➤ Lema de pompare pentru limbaje regulate (REG) [continuare]	20
➤ Expresii regulate (RegEx)	21
➤ Algoritm: Transformarea RegEx \rightarrow AFN- λ	22
➤ Algoritm: Transformarea AFN- $\lambda \rightarrow$ RegEx	23
• Câteva formule utile	25
~ Seminar 5 ~	29
➤ Algoritm: Minimizare AFD (metoda 1 : cu partiționarea mulțimii Q a stărilor)	29
➤ Algoritm: Minimizare AFD (metoda 2 : cu teorema Myhill-Nerode)	31

~ Seminar 6 ~	36
➤ Gramatici / Grammars	36
❖ <i>Gramatici regulate / Regular Grammars</i>	36
• Echivalența dintre Gramaticile regulate și Automatele finite	36
• Verificare generare cuvânt de către gramatică:	37
❖ <i>Gramatici independente de context (GIC) / Context-free Grammars (CFG)</i>	37
➤ Forma Normală Chomsky	39
• <i>Algoritm:</i> Transformarea GIC → gramatică scrisă în F.N. Chomsky	39
~ Seminar 7 ~	45
➤ Definiție APD și mod de funcționare	45
• Modalități de acceptare a unui cuvânt de către un APD	45
• Verificare acceptare cuvânt de către un APD	46
➤ <i>Lema de pompare pentru limbaje independente de context</i>	48

~ Seminar 1 ~

alfabet = mulțime finită și nevidă de simboluri

cuvânt peste un alfabet Σ = secvență finită de simboluri din Σ

limbaj formal peste un alfabet Σ = 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:

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

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

$$\rightarrow L = \{aa, abd, acd, aba, abbd, abcd, abca, abcbd, abccd\}$$

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

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

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

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 λ 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. $L^+ = L^* \setminus \{\lambda\}$

Obs: Atenție la ridicarea la putere a cuvintelor! (Să nu distribuiți puterea ca la înmulțire, ci concatenați cuvântul cu el însuși de câte ori spune puterea.)

$(abc)^2 = abcabc$; $(abc)^2 \neq aabbcc = a^2b^2c^2$

Obs: Cuvântul vid este element neutru la concatenare: $w \cdot \lambda = \lambda \cdot w = w$

➤ Definiție DFA și mod de funcționare

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

DFA = (Q, Σ , q_0 , F, δ)

Q mulțimea de stări (mulțime finită și nevidă)

Σ 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.
- Cât timp este posibil, la fiecare pas încercăm să procesăm câte un simbol din cuvântul de intrare (de la stânga la dreapta) astfel:
→ conform funcției de tranziție (dacă $\delta(q_i, x) = q_j$), din starea curentă (q_i) citind simbolul curent (x) din cuvântul de intrare ajungem într-o anumită stare (q_j) (care va deveni noua stare curentă din care vom citi următorul simbol din cuvânt).
- Algoritmul se termină în 3 cazuri:
 - a) dacă simbolul curent din cuvânt nu poate fi procesat (din cauză că nu există tranziție din starea curentă cu acel simbol), atunci cuvântul este **respins**.
 - b) dacă după procesarea întregului cuvânt s-a ajuns într-o stare nefinală (din mulțimea $Q - F$), atunci cuvântul este **respins**.
 - c) dacă după procesarea întregului cuvânt s-a ajuns într-o stare finală (din mulțimea F), atunci cuvântul este **acceptat**.

- 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:

$L(M) = \{w \mid (q_0, w) \vdash^* (p, \lambda), p \in F\}$ sau

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

➤ Definiție NFA și mod de funcționare

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

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

Q mulțimea de stări (mulțime finită și nevidă)

Σ 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 $L(M) = \{w \mid \delta(q_0, w) \cap F \neq \emptyset\}$

• Algoritm care verifică dacă un cuvânt este sau nu acceptat de un automat NFA:

- Pornim din (mulțimea formată din) starea inițială și avem cuvântul de intrare.
- *Cât timp este posibil*, aplicăm aceeași idee ca la DFA, doar că în loc să avem o singură stare curentă, la fiecare pas avem o mulțime de stări curente. Din fiecare stare din această mulțime încercăm să citim simbolul curent din cuvânt și ajungem într-o nouă mulțime de stări (din care vom încerca să citim simbolul următor din cuvânt). Formal, pentru mulțimea de stări curente R și simbolul curent x avem:

$$\delta(R, x) = \bigcup_{q \in R} \delta(q, x)$$

- Algoritmul se termină în 3 cazuri:

- a) dacă simbolul curent din cuvânt *nu poate fi procesat* (din cauză că nu există tranziție din niciuna dintre stările curente cu acel simbol), atunci cuvântul este **respins**.
- b) dacă după procesarea întregului cuvânt s-a ajuns într-o mulțime de stări R care nu conține nicio stare finală ($R \cap F = \emptyset$), atunci cuvântul este **respins**.
- c) dacă după procesarea întregului cuvânt s-a ajuns într-o mulțime de stări R care conține cel puțin o stare finală ($R \cap F \neq \emptyset$), atunci cuvântul este **acceptat**.

Obs: La DFA și NFA, $\lambda \in L \Leftrightarrow q_0 \in F$ (cuvântul vid este acceptat de automat dacă și numai dacă starea inițială este și stare finală).

➤ Exerciții: Desenați DFA / NFA pentru limbajele date

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

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



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

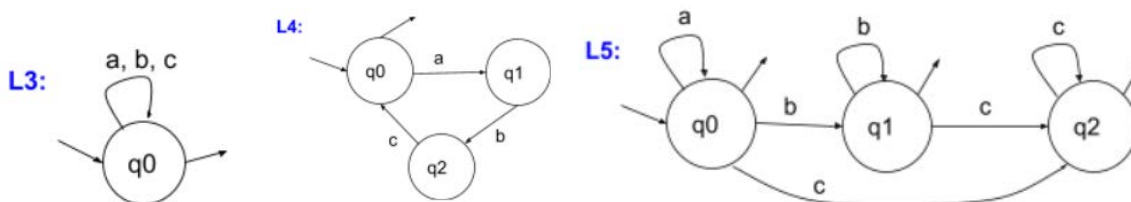
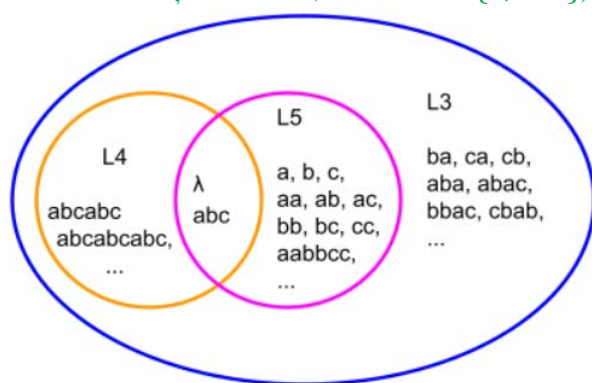
$L3 = \{a, b, c\}^* = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, \dots, caa, cab, \dots\}$

$L4 = (abc)^* = \{\lambda, abc, abcabc, abcabcabc, \dots\}$

$L5 = \{a^n b^k c^p \mid n \geq 0, k \geq 0, p \geq 0\} = \{\lambda, a, a^2, a^3, \dots, b, b^2, b^3, \dots, c, c^2, c^3, \dots, ab, ab^2, a^2b, ab^3, a^3b, a^2b^2, \dots, ac, ac^2, a^2c, ac^3, a^3c, a^2c^2, \dots, bc, bc^2, b^2c, bc^3, b^3c, b^2c^2, \dots, abc, a^2bc, ab^2c, abc^2, \dots, a^2b^2c^2, \dots\}$

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

(Da: $L4 \subset L3$ și $L5 \subset L3$; $L4 \cap L5 = \{\lambda, abc\}$)



Observații:

→ L3 conține toate cuvintele posibile peste alfabetul $\{a, b, c\}$, cuvinte de lungime ≥ 0 , în care literele pot fi în orice ordine. De aceea, punem toate simbolurile pe **bucla aceleiași stări** pentru a permite amestecarea lor.

→ L4 conține cuvinte obținute prin concatenarea cuvântului “abc” cu el însuși de $n \geq 0$ ori. De aceea, folosim un **ciclu** în care avem câte o muchie pentru fiecare literă din cuvântul “abc”, exact în această ordine. Setăm stare finală doar pe cea în care ajungem după ce citim c, ultima literă din cuvânt.

→ L5 conține cuvinte de forma $a^n b^k c^p$ pe care le obținem dând orice valori numere naturale pentru n, k și p. Literele trebuie să fie *neamestecate* și *neapărat în această ordine*: întâi a-uri, apoi b-uri, apoi c-uri. Pentru fiecare literă (a, b, c) trebuie să folosim **câte o buclă pe o stare diferită** pentru a nu amesteca literele; când începem să citim o literă nouă (primul b sau c) *trebuie să mergem într-o stare nouă* și nu avem voie să avem muchii de întoarcere spre stări plecând din care citeam fostele litere, pentru a asigura ordinea corectă (niciun a după b și niciun a sau b după c).

Pentru $k=0$ (lipsă b-uri) adăugăm muchia cu c care sare din q0 direct în q2. Setăm **stările finale** q0 (pt a accepta cuvântul vid / cuvinte doar cu a), q1 (pt cuvinte doar cu b / cu a și b) și q2 (pt cuvinte doar cu c / cu a și c / cu b și c / cu a, b și c).

- **Temă de gândire:**

$L6 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ (același număr de a-uri și b-uri, amestecate oricum)

$L7 = \{a^n b^n \mid n \geq 0\} \subset L6$ (același număr de a-uri și b-uri, obligatoriu întâi a-uri, apoi b-uri)

versus

$L8 = \{w \in \{a, b\}^* \mid |w|_a \bmod 2 = |w|_b \bmod 2\}$ (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?

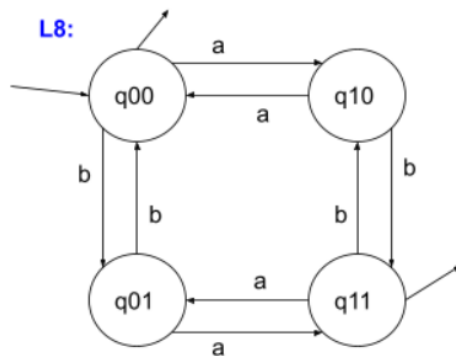
Observații:

→ $L6$ și $L7$ nu sunt limbaje regulate (nu există niciun automat finit care să le accepte).

Argument intuitiv: nu putem să verificăm egalitatea dintre **numărul total** de a-uri și cel de b-uri pentru o **infinitate de valori posibile** ale lui n (orice număr natural) deoarece DFA / NFA au la dispoziție doar **un număr finit de stări** în care putem “memora” o cantitate finită de informații.

→ În schimb $L8$ este limbaj regulat pentru că există **un număr finit de cazuri** pentru **paritatea numărului** de a-uri și cel de b-uri.

Idee: Memorăm în indicii stărilor care sunt resturile împărțirii la 2 pentru numărul de a-uri și cel de b-uri citite (începând din starea inițială) până în acel moment; primul indice este pentru a-uri și al doilea pentru b-uri.



~ **Temă** ~

- Desenați câte un DFA sau NFA pentru fiecare limbaj dat.

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

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

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

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

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

$L18(X) = \{w \in \{0, 1, \dots, 9\}^* \mid w \text{ reprezintă un număr în baza 10 divizibil cu } X\}$,
unde $X \in \{3, 25, 4\}$ (câte un automat pt fiecare valoare a lui X).

~ Seminar 2 ~

➤ Algoritm: Transformarea AFN → AFD

Se dă un automat AFN. Se cere să se construiască un automat AFD echivalent (care să accepte același limbaj).

Idee: Atunci când în AFN dintr-o stare cu un anumit simbol avem ramificare către mai multe stări-destinație, în AFD vom grupa toate aceste stări-destinație într-o singură stare pentru a elimina ramificarea.

Algoritm: Pentru AFN-ul $(Q, \Sigma, q_0, F, \delta)$ construim AFD-ul $(Q', \Sigma, q_0, F', \delta')$ astfel:

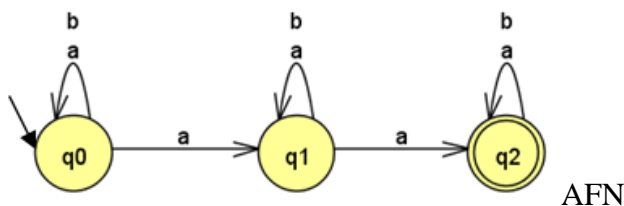
- Stările AFD-ului vor reprezenta *submulțimi* ale mulțimii stărilor AFN-ului ($Q' \subseteq \mathcal{P}(Q)$).
- Cele două automate vor avea același alfabet Σ și **aceeași stare inițială** q_0 .
- Funcția de tranziție a AFD-ului este calculată astfel:

$$\delta'(R, x) = \bigcup_{q \in R} \delta(q, x), \quad \forall R \in \mathcal{P}(Q), \forall x \in \Sigma$$

- Stările finale ale AFD-ului sunt mulțimile care **conțin cel puțin o stare finală** a AFN-ului $F' = \{R \mid R \in Q', R \cap F \neq \emptyset\}$.

Obs: La seminar vom folosi *metoda iterativă* de calcul pentru Q' (pornim din starea inițială și adăugăm stările AFD-ului pe măsură ce ajungem la ele calculând funcția de tranziție δ' pentru stările găsite anterior).

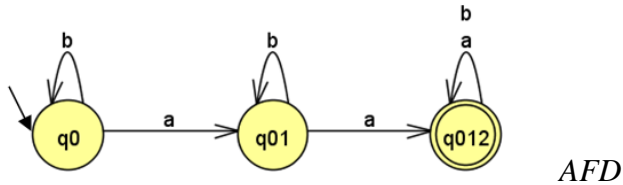
- **Exemplu:** Pentru AFN-ul următor construiți un AFD echivalent.



- 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.

δ_{AFN}	a	b
$q_0 \text{ init}$	$\{q_0, q_1\} = q_{01}$	$\{q_0\}$
q_1	$\{q_1, q_2\} = q_{12}$	$\{q_1\}$
$q_2 \in F$	$\{q_2\}$	$\{q_2\}$

δ_{AFD}	a	b
q_0 init	q_{01}	q_0
q_{01}	q_{012}	q_{01}
$q_{012} \in F$	q_{012}	q_{012}



➤ Verificare acceptare cuvânt de către 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:

$(q_0, bba) \vdash^b (q_0, ba) \vdash^b (q_0, a) \vdash^a (q_{01}, \lambda), q_{01} \notin F \Rightarrow$ bba respins

→ AFN, cuv bba:

$(q_0, bba) \vdash^b (q_0, ba) \vdash^b (q_0, a) \vdash^a \{(q_0, \lambda), (q_1, \lambda)\}, \{q_0, q_1\} \cap F = \emptyset \Rightarrow$ bba respins

→ AFD, cuv babbaba:

$(q_0, babbaba) \vdash^b (q_0, abbaba) \vdash^a (q_{01}, bbaba) \vdash^b (q_{01}, baba) \vdash^b (q_{01}, aba) \vdash^a (q_{012}, ba) \vdash^b (q_{012}, a) \vdash^a (q_{012}, \lambda), q_{012} \in F \Rightarrow$ babbaba acceptat

→ AFN, cuv babbaba:

$(q_0, babbaba) \vdash^b (q_0, abbaba) \vdash^a \{(q_0, bbaba), (q_1, bbaba)\} \vdash^b \{(q_0, baba), (q_0, baba)\} \vdash^b \{(q_0, aba), (q_1, aba)\} \vdash^a \{(q_0, ba), (q_1, ba), (q_2, ba)\} \vdash^b \{(q_0, a), (q_1, a), (q_2, a)\} \vdash^a \{(q_0, \lambda), (q_1, \lambda), (q_2, \lambda)\}, \{q_0, q_1, q_2\} \cap F = \{q_2\} \neq \emptyset \Rightarrow$ babbaba acceptat

➤ Automat Finit Nedeterminist cu λ -tranziții

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

Q mulțimea de stări

Σ 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 Σ , fie cuvântul vid λ . Deci se poate întâmpla să ajungem într-o stare nouă fără să fi citit nicio literă nouă din cuvântul de intrare, ci doar aplicând λ -tranziții.

➤ λ -închiderea unei stări

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

Obs: Orice stare face parte din propria λ -închidere (pentru că $\delta(q, \lambda^0) = q$; practic putem presupune că orice stare are o λ -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 λ :

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

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 :

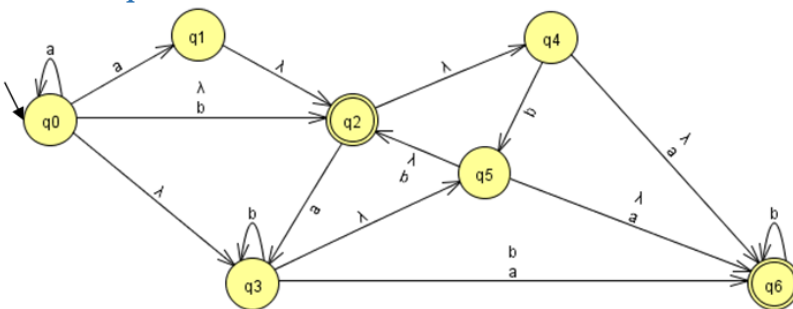
Se procedează analog ca în cazul AFN-ului, doar că înainte de a căuta toate stările posibile de continuare cu tranziții cu simbolul curent, trebuie să facem λ -închiderea mulțimii curente de stări. Iar după ce a fost citit tot cuvântul de intrare, trebuie să facem o ultimă λ -închidere a stărilor curente,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

Obs: λ -închiderea unei mulțimi de stări este egală cu reuniunea λ -închiderilor acelor stări.

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

Obs: La AFN- λ , $\lambda \in L \Leftrightarrow \langle q_0 \rangle \cap F \neq \emptyset$ (cuvântul vid este acceptat de automat dacă și numai dacă λ -închiderea stării inițiale conține cel puțin o stare finală).

• **Exemplu:** Se dă următorul AFN- λ .



a) Calculăm λ -închiderile tuturor stărilor.

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

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

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

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

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

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

$$\langle q_6 \rangle = \{q_6\}$$

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

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

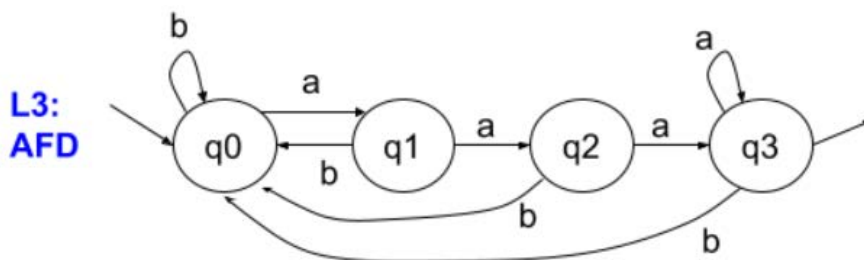
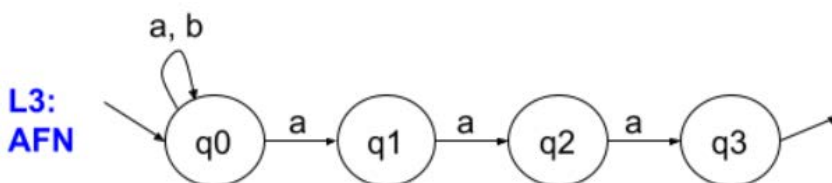
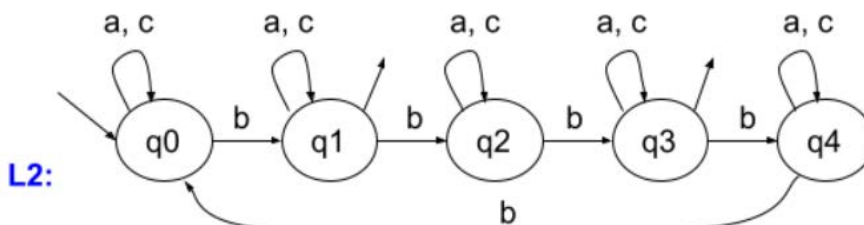
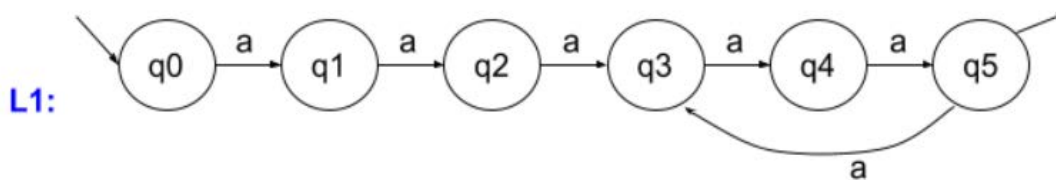
➤ Desenați câte un AFN / AFD pentru limbajele următoare.

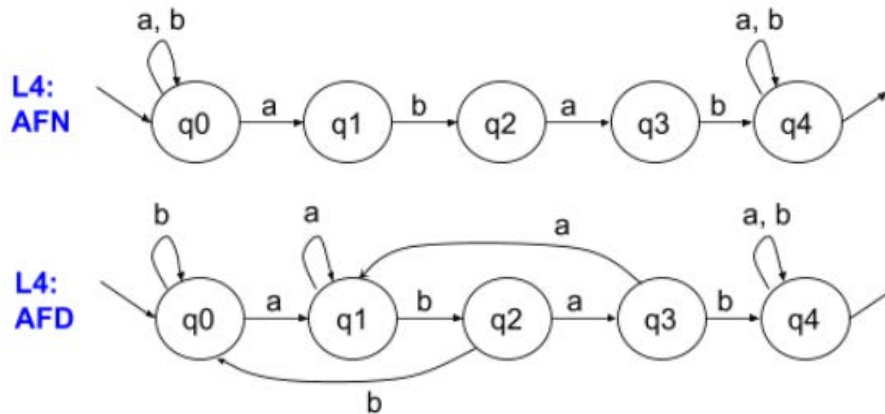
$L1 = \{a^{3k+2} \mid k \geq 1\}$ (AFD)

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

$L3 = \{waaa \mid w \in \{a, b\}^*\}$ (AFN și AFD)

$L4 = \{xababy \mid x, y \in \{a, b\}^*\}$ (AFN și AFD)





~ Temă ~

EX_1: Desenați câte un AFD pentru fiecare limbaj dat.

$L5 = \{a^{2n}b^{3k} \mid n \geq 0, k \geq 0\}$ Ce se schimbă dacă avem $k \geq 1$?

$L6 = \{a^{2n}b^{3k} \mid n \geq 1, k \geq 1\}$ Ce se schimbă dacă avem $k \geq 0$?

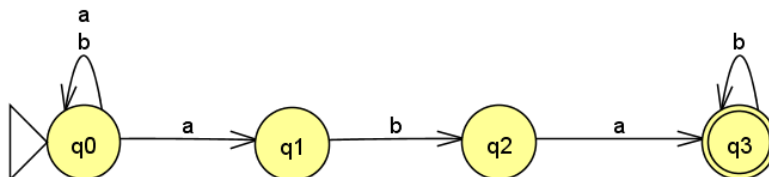
$L7 = \{w \in \{0,1\}^* \mid w \text{ începe cu } 1 \text{ și se termină cu } 0\}$

$L8 = \{w \in \{0,1\}^* \mid w \text{ conține cel puțin trei de } 1\}$

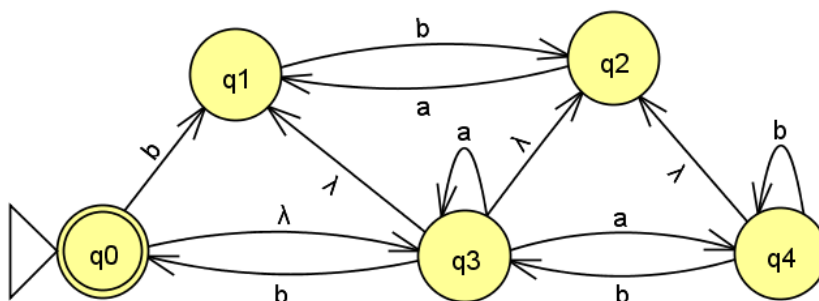
$L9 = \{w \in \{0,1\}^* \mid |w| \geq 3 \text{ și al treilea simbol din } w \text{ este un } 0\}$

EX_2: a) Pentru următorul AFN construiți un AFD echivalent.

b) Verificați pe AFD și AFN dacă cuvântul **bababb** este acceptat sau nu.



EX_3: Pentru următorul AFN- λ : **(a)** calculați λ -închiderile tuturor stărilor și **(b)** verificați dacă cuvântul **aababb** este acceptat sau nu.



~ Seminar 3 ~

➤ **Algoritm:** Transformarea AFN- $\lambda \rightarrow$ AFN (**metoda 1**) [vezi curs 3, pag 28 – 30]

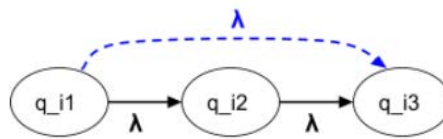
Se dă un automat AFN- λ . Se cere să se construiască un automat AFN echivalent.

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

➔ **Pas 1 („ λ -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 λ -tranziție către fiecare stare r (cu $r \neq q$) din mulțimea $\langle q \rangle$ (λ -închiderea stării q).

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

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

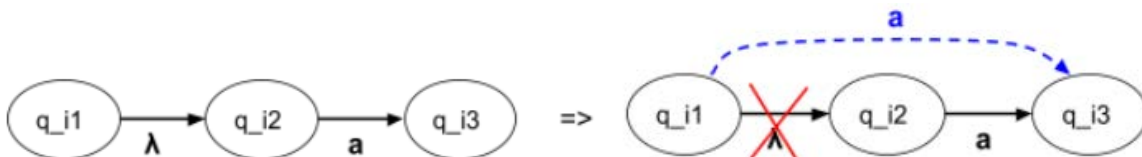


(Adică stările care conțineau în λ -închiderea lor cel puțin o stare finală vor deveni și ele stări finale.)

➔ **Pas 2 („ λ -transition removal”):**

- Adăugăm tranziții cu litere din alfabet care să înlocuiască efectul λ -tranzițiilor, astfel:

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

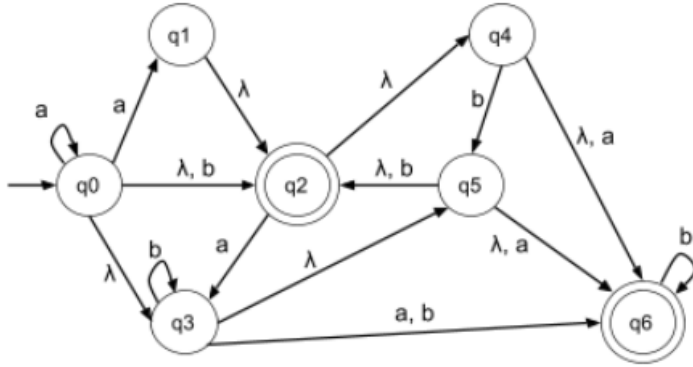


Adică dacă din starea q puteam ajunge în starea r citind λa ($\forall a \in \Sigma$), atunci **adăugăm o tranziție cu litera a** direct de la q la r .

- Apoi **eliminăm toate λ -tranzițiile** din automat.

- **Exemplu:** Se dă următorul AFN- λ .

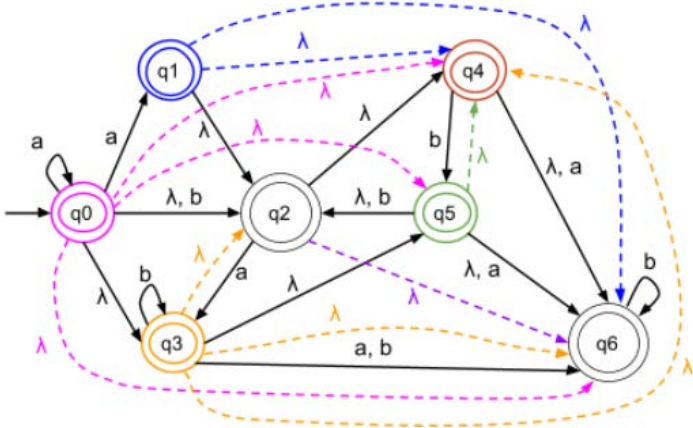
(Același din seminarul 2, pentru care calculasem deja λ -închiderile tuturor stărilor.)



→ Pas 1 („ λ -completion”):

- $\langle q_0 \rangle = \{q_0, q_2, q_3, q_4, q_5, q_6\} \Rightarrow$ adaug λ 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 λ de la q_1 spre q_4, q_6
- $\langle q_2 \rangle = \{q_2, q_4, q_6\} \Rightarrow$ adaug λ de la q_2 spre q_6
- $\langle q_3 \rangle = \{q_3, q_5, q_2, q_6, q_4\} \Rightarrow$ adaug λ 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 λ de la q_5 spre q_4
- $\langle q_6 \rangle = \{q_6\} \Rightarrow$ nu adaug nimic

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



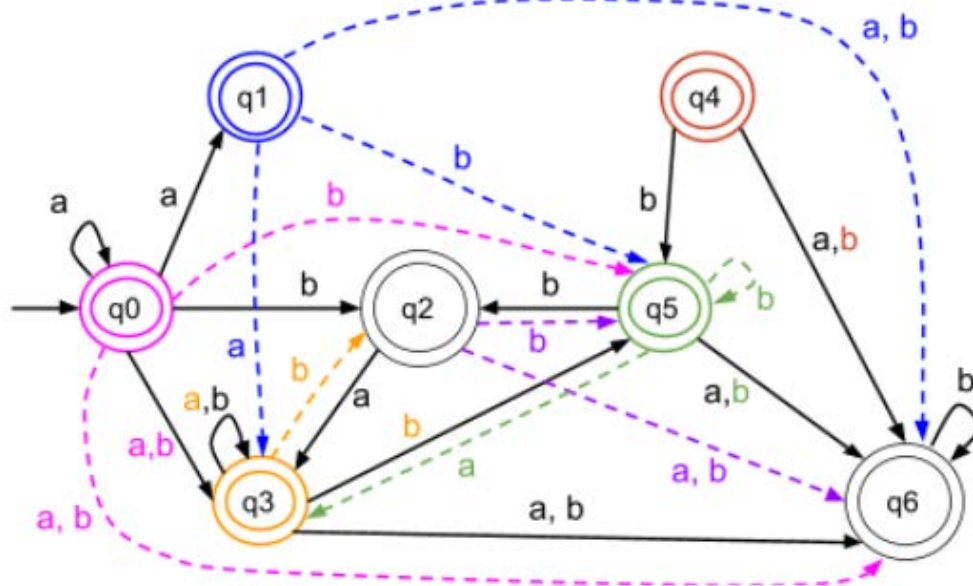
→ Pas 2 („ λ -transition removal”):

Vom elimina: $\delta(q_{i1}, \lambda) \ni q_{i2}$	Avem: $\delta(q_{i2}, x) \ni q_{i3}, x \in \Sigma$	Adăugăm: $\delta(q_{i1}, x) \ni q_{i3}$
$\delta(q_0, \lambda) \ni q_2$	$\delta(q_2, a) \ni q_3$	$\delta(q_0, a) \ni q_3$
$\delta(q_0, \lambda) \ni q_3$	$\delta(q_3, a) \ni q_6$ $\delta(q_3, b) \ni \{q_3, q_6\}$	$\delta(q_0, a) \ni q_6$ $\delta(q_0, b) \ni \{q_3, q_6\}$
$\delta(q_0, \lambda) \ni q_4$	$\delta(q_4, a) \ni q_6$ $\delta(q_4, b) \ni q_5$	$\delta(q_0, a) \ni q_6$ $\delta(q_0, b) \ni q_5$
$\delta(q_0, \lambda) \ni q_5$	$\delta(q_5, a) \ni q_6$ $\delta(q_5, b) \ni q_2$	$\delta(q_0, a) \ni q_6$ $\delta(q_0, b) \ni q_2$
$\delta(q_0, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_0, b) \ni q_6$

Vom elimina: $\delta(q_{i1}, \lambda) \ni q_{i2}$	Avem: $\delta(q_{i2}, x) \ni q_{i3}, x \in \Sigma$	Adăugăm: $\delta(q_{i1}, x) \ni q_{i3}$
$\delta(q_1, \lambda) \ni q_2$	$\delta(q_2, a) \ni q_3$	$\delta(q_1, a) \ni q_3$
$\delta(q_1, \lambda) \ni q_4$	$\delta(q_4, a) \ni q_6$ $\delta(q_4, b) \ni q_5$	$\delta(q_1, a) \ni q_6$ $\delta(q_1, b) \ni q_5$
$\delta(q_1, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_1, b) \ni q_6$
$\delta(q_2, \lambda) \ni q_4$	$\delta(q_4, a) \ni q_6$ $\delta(q_4, b) \ni q_5$	$\delta(q_2, a) \ni q_6$ $\delta(q_2, b) \ni q_5$
$\delta(q_2, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_2, b) \ni q_6$
$\delta(q_3, \lambda) \ni q_2$	$\delta(q_2, a) \ni q_3$	$\delta(q_3, a) \ni q_3$
$\delta(q_3, \lambda) \ni q_4$	$\delta(q_4, a) \ni q_6$ $\delta(q_4, b) \ni q_5$	$\delta(q_3, a) \ni q_6$ $\delta(q_3, b) \ni q_5$
$\delta(q_3, \lambda) \ni q_5$	$\delta(q_5, a) \ni q_6$ $\delta(q_5, b) \ni q_2$	$\delta(q_3, a) \ni q_6$ $\delta(q_3, b) \ni q_2$
$\delta(q_3, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_3, b) \ni q_6$
$\delta(q_4, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_4, b) \ni q_6$
$\delta(q_5, \lambda) \ni q_2$	$\delta(q_2, a) \ni q_3$	$\delta(q_5, a) \ni q_3$
$\delta(q_5, \lambda) \ni q_4$	$\delta(q_4, a) \ni q_6$ $\delta(q_4, b) \ni q_5$	$\delta(q_5, a) \ni q_6$ $\delta(q_5, b) \ni q_5$
$\delta(q_5, \lambda) \ni q_6$	$\delta(q_6, b) \ni q_6$	$\delta(q_5, b) \ni q_6$
$\delta(q_6, \lambda) = \emptyset \Rightarrow$		\Rightarrow Nu avem ce adăuga din q_6 .

Am obținut un AFN echivalent cu AFN- λ 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 (metoda 2) [asemănător AFN \rightarrow AFD, seminar 2]*

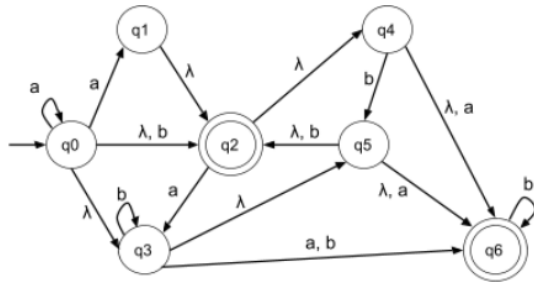
Idee: Dacă în AFN- λ din starea q citind $\lambda^* x \lambda^*$ ($\forall x \in \Sigma$) se ajunge în mulțimea de stări R , atunci în AFN din starea q citind litera x se va ajunge în mulțimea de stări R .

Starea inițială pentru AFN este aceeași ca la AFN- λ . Stările finale ale AFN-ului sunt cele ale căror λ -închideri în AFN- λ conțin cel puțin o stare finală.

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

• **Exemplu:** Se dă următorul AFN- λ .

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



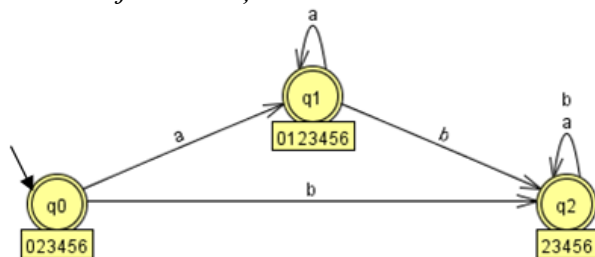
$\delta_{\text{AFN-}\lambda}$	a	b	λ	λ^* (λ -închiderea)
q0 init	{q0, q1}	{q2}	{q2, q3}	$\langle q0 \rangle = \{q0, q2, q3, q4, q5, q6\}$
q1	\emptyset	\emptyset	{q2}	$\langle q1 \rangle = \{q1, q2, q4, q6\}$
q2 in F	{q3}	\emptyset	{q4}	$\langle q2 \rangle = \{q2, q4, q6\}$
q3	{q6}	{q3, q6}	{q5}	$\langle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

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

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

δ_{AFD}	a	b
<q0> = q023456 init, in F	q0123456	q23456
q0123456 in F	q0123456	q23456
q23456 in F	q23456	q23456

Am obținut un AFD echivalent cu automatele AFN și AFN-λ 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

- Vrem să demonstrăm că L **nu este** limbaj regulat (adică nu se poate construi niciun automat finit care să-l recunoască pe L), folosind lema de pompare negată.

- Presupunem prin reducere la absurd că L **este** limbaj regulat. Atunci $\exists p \in \mathbb{N}$ (unde $p = |Q|$ numărul de stări ale unui automat finit care-l recunoaște pe L) și putem aplica lema de pompare. (*În continuare negăm afirmația lemei.*)

- **Alegem** (adică \exists) un cuvânt α din limbajul L , care să respecte ipoteza lemei de pompare, adică să aibă lungimea cel puțin p , deci $|\alpha| \geq p, \forall p \in \mathbb{N}$.

- **Pentru fiecare** (adică \forall) posibilă descompunere a cuvântului $\alpha = u \cdot v \cdot w$ care respectă condițiile (1) și (2) din lema ($|u \cdot v| \leq p$ și $|v| \geq 1$):

- **alegem** (adică \exists) convenabil câte un număr natural $i \geq 0$ 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.

Observație: Demonstrația este completă și corectă doar dacă se obține contradicție ($\beta \notin L$) pentru toate descompunerile posibile ale cuvântului α .

- **Exemplu:** $L_0 = \{a^n b^n \mid n \geq 0\} \notin REG$

Idee: Vrem să demonstrăm că L_0 **nu este limbaj regulat**. Observăm că L_0 conține cuvinte formate din n litere de "a" urmate tot de n litere de "b". În demonstrație trebuie să obținem un cuvânt β care să **nu** respecte această proprietate (adică să fie de forma $a^* b^*$, dar să aibă număr *diferit* de a-uri și b-uri).

Obs: Dacă aveam condiția $n \geq x$ (cu $x \in \mathbb{N}$ o constantă), atunci ar fi trebuit să alegem cuvântul $\alpha = a^{p+x} b^{p+x} \in L_0$ (pentru a fi sigur un cuvânt din limbaj $\forall p \in \mathbb{N}$).

Demonstrație: Presupunem prin reducere la absurd că L_0 este limbaj regulat. Atunci $\exists p \in \mathbb{N}$ și putem aplica lema de pompare. (*În continuare negăm afirmația lemei.*)

Alegem cuvântul $\alpha = a^p b^p \in L_0$, cu $|\alpha| = 2p \geq p, \forall p \in \mathbb{N}$ (*deci lungimea cuvântului respectă ipoteza lemei*). Conform lemei, cuvântul poate fi scris sub forma $\alpha = u \cdot v \cdot w$.

Din condiția (1) avem $|u \cdot v| \leq p$. Rezultă că cuvântul $u \cdot v$ conține doar litere de "a" (pentru că $u \cdot v$ este un prefix al primelor p caractere din α).

Atunci notăm $v = a^k$. Din condițiile (1) și (2) avem $1 \leq |v| \leq p$ (*pentru că se poate ca $u = \lambda$, adică $|u| = 0$*). Deci $1 \leq |a^k| \leq p$, adică $1 \leq k \leq p$ (*).

De asemenea, condiția (3) spune că $u \cdot v^i \cdot w \in L_0, \forall i \geq 0$. Alegem $i = 2$ și avem cuvântul $\beta = u \cdot v^2 \cdot w = a^{p+k} b^p \notin L_0$ (pentru că în relația (*) avem $k \geq 1$, deci numărul de "a"-uri este strict mai mare decât numărul de "b"-uri din cuvântul β). Am obținut o contradicție pentru (3), deci presupunerea făcută este falsă și L_0 nu este limbaj regulat. ■

• **Exemplu:** $L_1 = \{a^m b^n \mid m > n \geq 0\} \notin REG$

Idee: Vrem să demonstrăm că L_1 **nu este limbaj regulat**. Observăm că L_1 conține cuvinte formate din litere de "a" urmate de strict mai puține litere de "b". În demonstrație trebuie să obținem un cuvânt β care să **nu** respecte această proprietate (adică să fie de forma $a^* b^*$, dar să aibă $|\beta|_a \leq |\beta|_b$).

Obs: Ca să putem ajunge la contradicție, trebuie să alegem cuvântul α care respectă la limită inegalitatea dintre a-uri și b-uri (adică m exact cu o unitate mai mare decât n). De asemenea, b-urile trebuie să existe (deci alegem $n \neq 0$).

Demonstrație: Presupunem prin reducere la absurd că L_1 este limbaj regulat. Atunci $\exists p \in \mathbb{N}$ și putem aplica lema de pompă. (*În continuare negăm afirmația lemei.*)

Alegem cuvântul $\alpha = a^{p+1} b^p \in L_1$, cu $|\alpha| = 2p + 1 \geq p, \forall p \in \mathbb{N}$ (deci lungimea cuvântului respectă ipoteza lemei). Conform lemei, cuvântul poate fi scris sub forma $\alpha = u \cdot v \cdot w$.

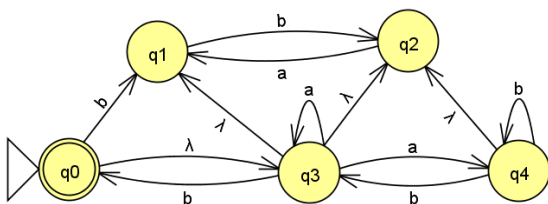
Din condiția (1) avem $|u \cdot v| \leq p$. Rezultă că cuvântul $u \cdot v$ conține doar litere de "a" (pentru că $u \cdot v$ este un prefix al primelor p caractere din α).

Atunci notăm $v = a^k$. Din condițiile (1) și (2) avem $1 \leq |v| \leq p$ (pentru că se poate ca $u = \lambda$, adică $|u| = 0$). Deci $1 \leq |a^k| \leq p$, adică $1 \leq k \leq p$ (*).

De asemenea, condiția (3) spune că $u \cdot v^i \cdot w \in L_1, \forall i \geq 0$. Alegem $i = 0$ și avem cuvântul $\beta = u \cdot v^0 \cdot w = u \cdot w = a^{p+1-k} b^p$. Conform (*) avem $1 \leq k \leq p$ (înmulțim cu -1) $\Leftrightarrow -p \leq -k \leq -1$ (adunăm $p+1$) $\Leftrightarrow 1 \leq p+1-k \leq p$. Avem $|\beta|_a \leq |\beta|_b \Rightarrow \beta \notin L_1$. Am obținut o contradicție pentru (3), deci presupunerea făcută este falsă și L_1 nu este limbaj regulat. ■

~ Temă ~

EX_1: Se dă următorul AFN- λ (același din tema seminar 2). Să se construiască un AFN și un AFD echivalente cu el.



(a) Transformați AFN- λ în AFN (folosind algoritmul de la curs: pas1 "λ-completion", apoi pas2 "λ-transition removal"). Desenați AFN-ul.

(b) Calculați λ-închiderile tuturor stărilor. Transformați AFN- λ în AFD (calculând tabelul cu drumurile de forma $\lambda^* x \lambda^*, \forall x \in \Sigma$, apoi tabelul pentru AFD). Desenați AFD-ul.

(c) Explicați care este limbajul recunoscut de AFD-ul obținut.

EX_2: Demonstrați că următoarele limbaje nu sunt regulate, folosind lema de pompă.

$L2 = \{a^{2n} b^{n+3} c^n \mid n \geq 1\} \notin REG$

$L3 = \{a^m b^{3n} c^{n+3} \mid m \geq 5, n \geq 1\} \notin REG$

~ Seminar 4 ~

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

• Exemple:

$L_4 = \{x \cdot x^R \mid x \in \{a, b\}^*\} \notin REG$ (discutat alegerea cuvântului α)

$L_5 = \{x \cdot x \mid x \in \{a, b\}^*\} \notin REG$ (discutat alegerea cuvântului α)

Obs: (**R = reversed**) Cuvântul w^R este oglindirea cuvântului w , de exemplu $(abc)^R = cba$.

➔ Pentru L_4 :

-- Cuvântul $\alpha = a^p a^p = a^{2p} = (aa)^p$ NU este o alegere bună (pentru acest α putem desena un AFD având un circuit de 2 de a).

- Dacă $v = a^{2k} \Rightarrow |\beta| = |u \cdot v^i \cdot w| = |u \cdot v \cdot w| + |v|^{(i-1)} = |a^{2p+(2k)*(i-1)}| = 2 * (p + k * (i - 1))$ este **par** (adică $\beta \in L_4$), $\forall i \geq 0$ (NU avem contradicție cu condiția 3 din lema).
- Dacă $v = a^{2k+1} \Rightarrow |\beta| = |a^{2p+(2k+1)*(i-1)}| = 2 * (p + k * (i - 1)) + (i - 1)$ este **impar**, adică $\beta \notin L_4$ (contradicție cu condiția 3 din lema) $\Leftrightarrow i$ este par (de exemplu alegem $i = 0$).

Concluzie: Dacă există descompuneri ale lui α (forme ale lui v) pentru care NU putem obține contradicție, înseamnă că demonstrația nu este corectă și trebuie ales un alt α .

-- Cuvântul $\alpha = a^p b b a^p$ este o alegere bună (pentru acest α nu putem desena un AFD).

- Dacă $v = a^k$ alegem $i = 2 \Rightarrow \beta = a^{p+k} b b a^p \notin L_4$ pentru că $1 \leq k \leq p$ (din primele două condiții din lema).

Concluzie: Avem un singur caz de descompunere a lui α (o singură formă a lui v), am obținut contradicție, deci demonstrația este corectă.

➔ Pentru L_5 :

-- Cuvântul $\alpha = a^p b a^p b$ este o alegere bună (pentru acest α nu putem desena un AFD).

Avem un singur caz de descompunere a lui α .

- Dacă $v = a^k$ alegem $i = 2 \Rightarrow \beta = a^{p+k} b a^p b \notin L_5$ pentru că $1 \leq k \leq p$ (din primele două condiții din lema). Deci avem contradicție și demonstrația este corectă.

-- Cuvântul $\alpha = b a^p b a^p$ este tot o alegere bună (pentru acest α nu putem desena un AFD), dar avem mai multe cazuri de descompunere a lui α .

- Dacă $v = b a^k$ (cu $0 \leq k \leq p - 1$) alegem $i = 0 \Rightarrow \beta = a^{p-k} b a^p \notin L_5$.
- Dacă $v = a^k$ (cu $1 \leq k \leq p$) alegem $i = 2 \Rightarrow \beta = b a^{p+k} b a^p \notin L_5$.

Concluzie: Avem contradicție pe fiecare caz posibil de descompunere, deci demonstrația este corectă.

• **Exemplu:**

$$L_6 = \{a^{n^2} | n \geq 1\} = \{a^1, a^4, a^9, a^{16}, a^{25}, a^{36}, \dots\} \notin REG$$

Obs: Proprietatea cuvintelor din limbajul L_6 este aceea că sunt formate doar din litere de "a" și lungimea lor este un număr pătrat perfect. Deci pentru a obține contradicția ($\beta \notin L_6$) trebuie să arătăm că lungimea lui β **nu** poate fi un pătrat perfect.

Demonstrație: Presupunem prin reducere la absurd că L_6 este limbaj regulat. Atunci $\exists p \in \mathbb{N}$ și putem aplica lema de pompă. (*În continuare negăm afirmația lemei.*)

Alegem cuvântul $\alpha = a^{p^2} \in L_6$, cu $|\alpha| = p^2 \geq p, \forall p \in \mathbb{N}$ (deci lungimea cuvântului respectă ipoteza lemei). Conform lemei, cuvântul poate fi scris sub forma $\alpha = u \cdot v \cdot w$.

Observăm că toate cuvintele din L_6 sunt formate doar din litere de "a". Atunci notăm $v = a^k$. Din condițiile (1) și (2) ale lemei avem $1 \leq |v| \leq p$. Deci $1 \leq |a^k| \leq p$, adică $1 \leq k \leq p$ (*).

De asemenea, condiția (3) din lema spune că $u \cdot v^i \cdot w \in L_6, \forall i \geq 0$.

Alegem $i = 2$ și avem cuvântul $\beta = u \cdot v^2 \cdot w$ de lungime

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

Conform (*), avem $1 \leq k \leq p$ (adunăm peste tot p^2) $\Leftrightarrow p^2 + 1 \leq p^2 + k \leq p^2 + p$.

Dar $p^2 < p^2 + 1$ și $p^2 + p < (p+1)^2$. Rezultă că $p^2 < p^2 + k < (p+1)^2$, adică $p^2 < |\beta| < (p+1)^2$. Deci $|\beta|$ nu poate fi pătrat perfect (pentru că este inclus strict între două pătrate perfecte consecutive) $\Rightarrow \beta \notin L_6$, contradicție cu condiția (3) din lema, deci presupunerea făcută este falsă și L_6 nu este limbaj regulat. ■

➤ **Expresii regulate (Regex)**

Definiție: Se numește familia expresiilor regulate peste Σ 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:

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



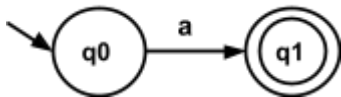
$$L = L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ și } w_2 \in L_2\}$$

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

➤ *Algoritm: Transformarea RegEx → AFN-λ*

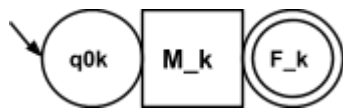
Pentru fiecare caz din definiția RegEx vom construi câte un automat finit echivalent.

Caz i)

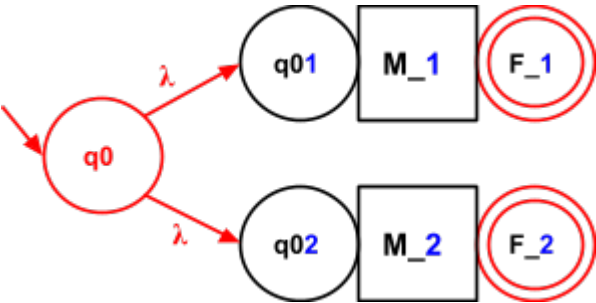
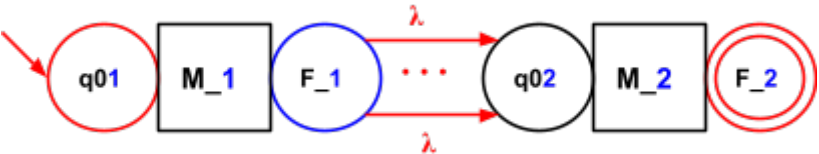
RegEx	$e = \emptyset$	$e = \lambda$	$e = a$, unde $a \in \Sigma$
Limбай	$L = \emptyset$	$L = \{\lambda\}$	$L = \{a\}$
Automat Finit			

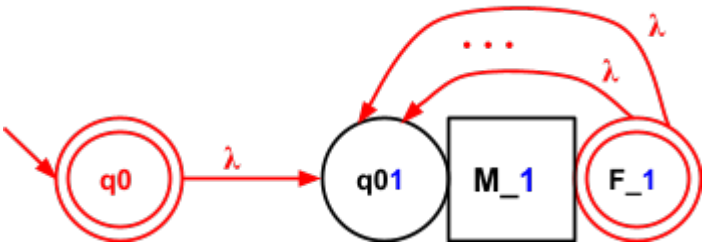
În **cazurile ii), iii) și iv)** presupunem că pentru expresia regulată e_k și limbajul $L(e_k)$, $k \in \{1, 2\}$ avem deja automate finite $AF(L(e_k)) = (Q_k, \Sigma_k, q_{0k}, F_k, \delta_k)$, cu $Q_1 \cap Q_2 = \emptyset$ (stări disjuncte).

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.

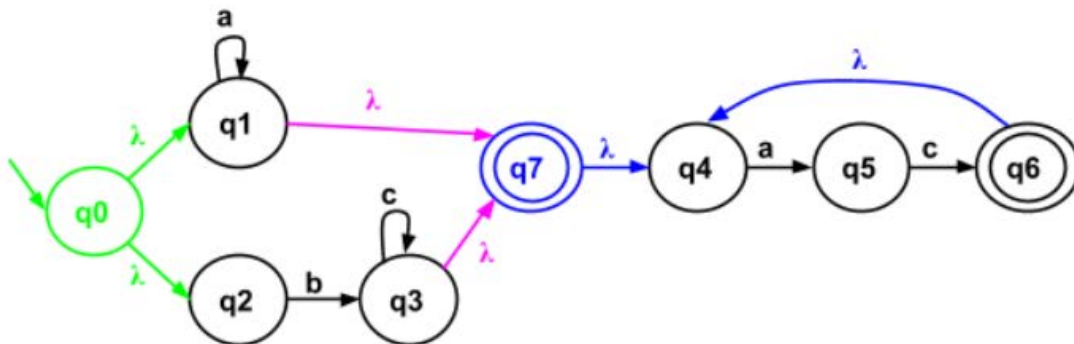


Vom construi automatele pentru operațiile de reuniune, concatenare și stelare.

<p>Caz ii) <i>(reuniune)</i></p> <p>RegEx $e = e_1 + e_2$</p> <p>Limбай $L(e) = L(e_1) \cup L(e_2)$</p>	<p>Automat Finit $AF(L(e_1 + e_2)) = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, q_0, F_1 \cup F_2, \delta_1 \cup \delta_2 \cup \{\delta(q_0, \lambda) = \{q_{01}, q_{02}\}\})$</p> 
<p>Caz iii) <i>(concatenare)</i></p> <p>RegEx $e = e_1 \cdot e_2$</p> <p>Limбай $L(e) = L(e_1) \cdot L(e_2)$</p>	<p>Automat Finit $AF(L(e_1 \cdot e_2)) = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, q_{01}, F_2, \delta_1 \cup \delta_2 \cup \{\delta(q_{f1}, \lambda) \ni q_{02} \mid \forall f_1 \in F_1\})$</p> 

<p>Caz iv) (stelare)</p> <p>RegEx $e = (e_1)^*$</p> <p>Limбай $L(e) = (L(e_1))^*$</p>	<p>Automat Finit $AF(L((e_1)^*)) = (Q_1 \cup \{q_0\}, \Sigma_1, q_0, F_1 \cup \{q_0\},$ $\delta_1 \cup \{\delta(q_0, \lambda) = q_{01}\} \cup \{\delta(q_{f_1}, \lambda) \ni q_{01} \mid \forall f_1 \in F_1\})$</p> 
--	--

- **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 $L_4 = (a^* + bc^*) \cdot (ac)^* = (L_1 + L_2) \cdot (L_3)^*$.



➤ **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, Σ este alfabetul, q_0 este starea inițială, F este mulțimea stărilor finale. Aici (în locul funcției de tranziție) avem **funcția de etichetare** $et: Q \times Q \rightarrow \mathbf{RegEx}(\Sigma)$.

Notăm $et(p, q)$ prin e_{pq} (expresia regulată asociată săgeții de la starea p la starea q)

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 astfel: 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.

$$et(q_x, q_y) = \{w \in \mathbf{RegEx}(\Sigma) \mid w = a_1 + a_2 + \dots + a_n ;$$

$$q_y \in \delta(q_x, a_i), a_i \in (\Sigma \cup \{\lambda\}), \forall i \in \{1, \dots, n\}\}$$

Pas 2: Dacă starea inițială este și finală sau dacă există săgeți care vin către starea inițială, atunci se adaugă la automat o nouă stare care va fi inițială și va avea o săgeată cu expresia λ 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 unica finală și va avea săgeți cu expresia λ 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, astfel:

→ Presupunem că vrem să eliminăm starea q și că există săgeți cu etichetele (expresiile regulate) $et(p, q)$, $et(q, s)$ și eventual bucla cu $et(q, q)$.

→ Atunci obținem noua etichetă (expresie regulată) de pe săgeata de la starea p la starea s :

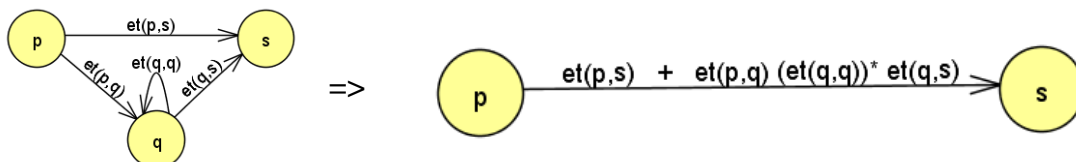
- [(fosta etichetă directă de la p la s) sau (\emptyset dacă nu există săgeată directă)]
reunită cu
- [(eticheta de la p la q) concatenată cu
(stelarea etichetei buclei de la q la q , sau λ dacă bucla nu există) concatenată cu
(eticheta de la q la s)]. (Vezi desenul de mai jos.)

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).

• **Observații:**

(1) La pas 4, pentru starea q pe care dorim să o eliminăm (împreună cu toate săgețile lipite de ea), trebuie să găsim orice “predecesor” $p \neq q$ (adică există o săgeată de la p la q) și orice “succesor” $s \neq q$ (adică există săgeată de la q la s). Deci făcând abstracție de eventuala buclă a lui q , căutăm și *grupăm orice săgeată care intră spre q cu orice săgeată care iese din q și astfel obținem expresia regulată de pe săgeata de la p la s cu formula explicată mai sus. Atenție, dacă $p = s$, înseamnă că vom obține o buclă.*

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



(2) Dacă una din expresii conține reuniune (“+”), atunci o *incluăm între paranteze*, pentru a se executa întâi acea reuniune și abia apoi concatenarea cu expresiile de pe alte săgeți. Fiecare expresie obținută între p și s încercăm să o *simplificăm* cât mai mult folosind formulele de mai jos.

(3) În funcție de *ordinea* în care alegem să eliminăm stările la pasul 4, vom obține o anumită expresie, dar toate sunt echivalente între ele. **Sfat:** În general, eliminăm starea care are momentan cele mai puține săgeți pentru a calcula cât mai puține drumuri.

Atenție să nu confundați semnul “+” dintre expresii (folosit pentru *reuniunea* lor) cu semnul “+” pus la putere (folosit pentru *concatenare repetată*, cel puțin puterea 1).

Obs: Algoritmul de mai sus descoperă și *reunește expresiile regulate corespunzătoare tuturor drumurilor de la starea inițială la o stare finală*. Puteți verifica asta pe exemplele următoare, comparând automatul finit dat cu expresia regulată obținută la finalul algoritmului.

• **Câteva formule utile**

- (A) $e \cdot \emptyset = \emptyset$ și $\emptyset \cdot e = \emptyset$ (\emptyset este pentru concatenare cum este 0 pentru înmulțire)
 (B) $e \cdot \lambda = e$ și $\lambda \cdot e = e$ (λ este pentru concatenare cum este 1 pentru înmulțire)
 (C) $e^* \cdot e = e^+$ și $e \cdot e^* = e^+$ (Dar e^+ nu va fi folosită în RegEx pt că nu respectă definiția lor.)
 (D) $\{e_1, e_2\}^* = (e_1 + e_2)^* = (e_1^* \cdot e_2^*)^*$ (Formulă valabilă pentru oricâte expresii, nu doar 2.)
 (E) $e_1 \cdot (e_2 + e_3) = (e_1 \cdot e_2) + (e_1 \cdot e_3)$ și $(e_1 + e_2) \cdot e_3 = (e_1 \cdot e_3) + (e_2 \cdot e_3)$
 (F) $e + \emptyset = \emptyset + e = e$ (\emptyset este pentru reuniune cum este 0 pentru adunare)
 (G) $\emptyset^* = \{\lambda\}$ (conform definiției stelării) și $\lambda^* = \lambda$ (conform formulei B de mai sus)
 (H) Dacă $e_1 \supseteq e_2$, atunci $e_1 + e_2 = e_2 + e_1 = e_1$. (De exemplu: $a + ab^* = ab^*$)
 (I) În loc de $\lambda + (e)^+ = \lambda + e \cdot e^*$ scriem e^* .

• **Exemplu rezolvat:**

Să se transforme următorul automat finit într-o expresie regulată echivalentă.

<p>Pas 1 (AF \rightarrow AFE): \Rightarrow Reunim tranzițiile aflate pe aceeași săgeată.</p>	
<p>Pas 2 (Verificăm <i>starea inițială</i>: - să nu fie stare finală și - să nu vină săgeți către ea) \Rightarrow adăugăm o nouă stare inițială cu λ către fosta stare inițială.</p>	
<p>Pas 3 (Verificăm <i>starea finală</i>: - să fie unica finală și - să nu plece săgeți din ea) \Rightarrow adăugăm o nouă unică stare finală spre care vin λ din fostele stări finale.</p>	
<p>Pas 4 (eliminăm q2): $\Rightarrow et(q_1, q_3) = et(q_1, q_3) + et(q_1, q_2) \cdot (et(q_2, q_2))^* \cdot et(q_2, q_3)$</p>	
<p>Pas 4 (eliminăm q1): $\Rightarrow et(q_0, q_3) = et(q_0, q_3) + et(q_0, q_1) \cdot (et(q_1, q_1))^* \cdot et(q_1, q_3)$</p>	

• **Exemplu:** [discutat la seminar]

Să se transforme următorul automat finit într-o expresie regulată echivalentă.

<p>Pas 1 (AF → AFE):</p> <p>=> Reunim tranzițiile aflate pe aceeași săgeată.</p>	
<p>Pas 2 (Verificăm starea inițială: - să nu fie stare finală și - să nu vină săgeți către ea)</p> <p>=> adăugăm o nouă stare inițială cu λ către fosta stare inițială.</p>	
<p>Pas 3 (Verificăm starea finală: - să fie unica finală și - să nu plece săgeți din ea)</p> <p>=> adăugăm o nouă unică stare finală spre care vin λ din fostele stări finale.</p>	
<p>Pas 4 (eliminăm q3):</p> <p>=> $et(q_2, q_4) = et(q_2, q_4) + et(q_2, q_3) \cdot (et(q_3, q_3))^* \cdot et(q_3, q_4)$</p> <p>=> $et(q_1, q_4) = et(q_1, q_4) + et(q_1, q_3) \cdot (et(q_3, q_3))^* \cdot et(q_3, q_4)$</p>	
<p>Pas 4 (eliminăm q2):</p> <p>=> $et(q_1, q_4) = et(q_1, q_4) + et(q_1, q_2) \cdot (et(q_2, q_2))^* \cdot et(q_2, q_4)$</p>	
<p>Pas 4 (eliminăm q1):</p> <p>=> $et(q_0, q_4) = et(q_0, q_4) + et(q_0, q_1) \cdot (et(q_1, q_1))^* \cdot et(q_1, q_4)$</p>	

~ Temă ~

EX_1: Demonstrați că următoarele limbaje nu sunt regulate, folosind lema de pompare.

$$L7 = \{a^{2^n} \mid n \geq 0\} = \{a^1, a^2, a^4, a^8, a^{16}, a^{32}, a^{64}, \dots\} \notin REG$$

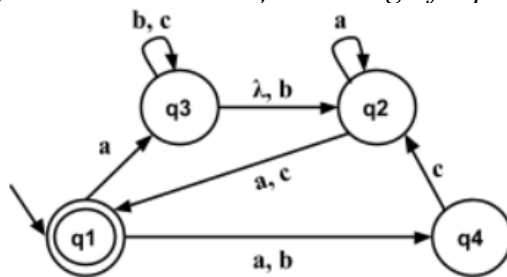
$$L8 = \{a^n \mid n \text{ număr prim}\} = \{a^2, a^3, a^5, a^7, a^{11}, a^{13}, a^{17}, a^{19}, a^{23}, \dots\} \notin REG$$

(Încercați să redactați demonstrațiile fără să aveți sub ochi alte exemple, să verificați dacă ați înțeles și reținut structura acestor demonstrații cu lema de pompare.)

EX_2: Desenați 5 automate finite (cu stări disjuncte) pentru $L_1 = b^* \cdot a \cdot (bc)^*$, $L_2 = \lambda + c$, $L_3 = (a + c)^*$, $L_4 = b^* \cdot c \cdot (a + b)$, $L_5 = (c + a) \cdot (c + b) \cdot a^*$, apoi folosind algoritmii pentru reuniune, concatenare, stelare și ținând cont de paranteze și de ordinea operațiilor, desenați automatul pentru $L_6 = ((L_1 + L_2)^* + (L_3 \cdot L_4)^*) \cdot L_5$.

EX_3: Să se transforme următorul automat finit într-o expresie regulată echivalentă.

(Pentru pașii 1+2+3 puteți să desenați un singur graf, apoi la pasul 4 pentru eliminarea fiecărei stări desenați câte un graf separat.)



• (Exerciții recapitulative)

→ **Închiderea limbajelor regulate la operații** (complement; intersecție, diferență, reuniune)

Obs: Dacă limbajul regulat L este acceptat de un AFD complet definit (fără tranziții lipsă) $AFD(L) = (Q, \Sigma, \delta, q_0, F)$, atunci putem construi un AFD care să accepte **complementul lui L** ($\Sigma^* \setminus L$) prin **interschimbarea stărilor finale cu cele nefinale** $AFD(\bar{L}) = (Q, \Sigma, \delta, q_0, Q \setminus F)$.

Obs: Dacă limbajele regulate L_1 și L_2 sunt acceptate de 2 automate AFD complet definite $AFD(L_1) = (Q_1 = \{q_0, q_1, \dots\}, \Sigma, \delta_1, q_0, F_1)$ și $AFD(L_2) = (Q_2 = \{r_0, r_1, \dots\}, \Sigma, \delta_2, r_0, F_2)$, atunci putem construi un AFD cu stări obținute prin **produs cartezian între mulțimile de stări** ale celor 2 automate: $AFD(L) = (Q, \Sigma, \delta, (q_0, r_0), F)$ având

- stările $Q = Q_1 \times Q_2 = \{(q_i, r_j) \mid q_i \in Q_1 \text{ și } r_j \in Q_2\}$,
- tranzițiile $\delta((q_i, r_j), x) = (\delta_1(q_i, x), \delta_2(r_j, x)), \forall (q_i, r_j) \in Q, \forall x \in \Sigma$,
- starea inițială (q_0, r_0) (perechea formată din cele două stări inițiale),
- stările finale F depind dacă automatul acceptă limbajul:

- ✓ (intersecție) $L = L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ și } w \in L_2\} \Rightarrow F = F_1 \times F_2$
- ✓ (diferență) $L = L_1 \setminus L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ și } w \notin L_2\} \Rightarrow F = F_1 \times (Q_2 \setminus F_2)$
sau $L = L_2 \setminus L_1 = \{w \in \Sigma^* \mid w \notin L_1 \text{ și } w \in L_2\} \Rightarrow F = (Q_1 \setminus F_1) \times F_2$
- ✓ (reuniune) $L = L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ sau } w \in L_2\} \Rightarrow F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Obs: Toate limbajele de la EX_4 și EX_5 sunt definite peste alfabetul $\Sigma = \{a, b\}$.

EX_4: Pentru fiecare limbaj L dat, desenați un AFD complet definit care să accepte L (scrieți alături care ar fi stările finale F), dar pe graf setați stările finale F' astfel încât să accepte \bar{L} (complementul limbajului L dat).

(a) $L = \{w \mid w \in a^*b^*\}$

(b) $L = \{w \mid w \in (ab^+)^*\}$

(c) $L = \{w \mid w \in a^* \cup b^*\}$

EX_5: Pentru $L1$ și $L2$ limbaje regulate date, desenați două AFD complet definite (peste alfabetul $\Sigma = \{a, b\}$) având stări disjuncte. Desenați AFD-ul cu stări obținute prin produs cartezian între stările automatelor pentru $L1$ și $L2$ (în această ordine), apoi scrieți alături care ar fi stările finale pentru a accepta limbajele:

$$L3 = L1 \cap L2, \quad L4 = L1 \setminus L2, \quad L5 = L2 \setminus L1, \quad L6 = L1 \cup L2.$$

(a) $L1 = \{w \mid |w|_a \text{ este par}\}$ și $L2 = \{w \mid w \text{ conține 1 sau 2 de } b\}$

(b) $L1 = \{w \mid w \text{ începe cu un } a\}$ și $L2 = \{w \mid w \text{ conține cel mult un } b\}$

(c) $L1 = \{w \mid |w|_a \text{ este impar}\}$ și $L2 = \{w \mid w \text{ se termină cu un } b\}$

~ Seminar 5 ~

➤ Algoritm: Minimizare AFD

Se dă un AFD complet definit (adică din fiecare stare din mulțimea Q pleacă câte o tranziție cu fiecare simbol din alfabetul Σ). 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: Minimizare AFD (metoda 1: cu partiționarea mulțimii Q a stărilor)

Ideea algoritmului este de a împărți mulțimea Q în partiții din ce în ce mai mici (pe măsură ce descoperim că două stări din *aceeași* partiție sunt separabile, le vom pune în partiții *diferite*), astfel încât la final orice partiție să conțină doar stări echivalente între ele.

Pas 0: Împărțim mulțimea Q în două partiții, una care conține stările nefinale ($A_0 = Q \setminus F$) și una care conține stările finale ($B_0 = F$). Orice stare din A_0 este separabilă de orice stare din B_0 prin cuvântul λ de lungime 0.

Pas $k \in \{1, 2, \dots\}$:

a) În cadrul fiecărei partiții X_{k-1} (cu $|X_{k-1}| \geq 2$, $X \in \{A, B, C, \dots\}$), verificăm pentru orice pereche de două stări ($q_i \in X_{k-1}$ și $q_j \in X_{k-1}$) dacă sunt separabile, adică dacă există vreo literă α din alfabetul Σ pentru care $\delta(q_i, \alpha) \in Y_{k-1}$ și $\delta(q_j, \alpha) \in Z_{k-1}$, cu $Y_{k-1} \neq Z_{k-1}$ (adică tranzițiile de la cele două stări q_i și q_j , cu o aceeași literă α , duc spre stări aflate deja (la pasul $k-1$) în partiții diferite).

→ Dacă duc către aceeași partiție ($Y_{k-1} = Z_{k-1}$), atunci stările q_i și q_j vor rămâne la pasul k în aceeași partiție T_k , pentru că *până acum* (pasul k) sunt echivalente (pentru orice cuvânt $\forall w \in \Sigma^*$, cu $0 \leq |w| \leq k$).

→ Iar dacă duc spre partiții diferite ($Y_{k-1} \neq Z_{k-1}$) atunci vom separa stările q_i și q_j în partiții diferite S_k și T_k , cu $S_k \neq T_k$ (există un cuvânt de lungime k , având ultima literă α , pentru care stările q_i și q_j sunt separabile).

b) Dacă la **pasul k** , **a)** s-a modificat vreo partiție, continuăm cu **pasul $k+1$** , **a)**. Altfel, algoritmul se oprește și în cadrul fiecărei partiții X_k avem doar stări echivalente între ele.

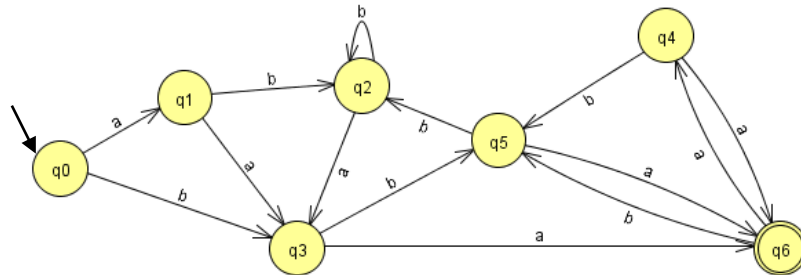
Atenție! La pasul k a) nu verificăm niciodată două stări aflate deja în partiții diferite (știm deja că stările sunt separabile), ci doar pe cele aflate în *aceeași* partiție.

Întrebări:

- (1) Dacă AFD-ul dat era deja minimal, în ce situație vom ajunge la finalul algoritmului?
- (2) Care este valoarea maximă la care poate ajunge k? (Adică maxim câți pași putem avea?)

- **Exemplu:** Să se minimizeze următorul AFD.

Fie automatul AFD complet definit din desen. Se cere automatul minimal echivalent cu el.



Pas 0:

Orice stare nefinală este separabilă prin λ de orice stare finală.

Deci împărțim stările din mulțimea Q în cele două partiții inițiale, A_0 și B_0 .

Apoi în interiorul tabelului cu tranziții, pentru fiecare stare destinație scriem din ce partiție de la pasul curent face parte.

Partițiile	δ	a	b
$A_0 = Q \setminus F$	q_0	$q_1 \in A_0$	$q_3 \in A_0$
	q_1	$q_3 \in A_0$	$q_2 \in A_0$
	q_2	$q_3 \in A_0$	$q_2 \in A_0$
	q_3	$q_6 \in B_0$	$q_5 \in A_0$
	q_4	$q_6 \in B_0$	$q_5 \in A_0$
	q_5	$q_6 \in B_0$	$q_2 \in A_0$
$B_0 = F$	q_6	$q_4 \in A_0$	$q_5 \in A_0$

Obs: La fiecare pas k o să redenumim partițiile A_k , B_k , C_k , etc.

Pas k = 1:

→ Verificăm toate perechile de stări aflate în A_0 și observăm că stările din $\{q_0, q_1, q_2\}$ sunt separabile pe coloana literei „a” de cele din $\{q_3, q_4, q_5\}$, pentru că tranzițiile primelor ajung cu „a” în A_0 , iar pentru celelalte tot cu „a” ajung în B_0 .

→ Partiția B_0 conține o singură stare, deci aici nu avem ce compara.

Partițiile	δ	a	b
A_1	q_0	$q_1 \in A_1$	$q_3 \in B_1$
	q_1	$q_3 \in B_1$	$q_2 \in A_1$
	q_2	$q_3 \in B_1$	$q_2 \in A_1$
B_1	q_3	$q_6 \in C_1$	$q_5 \in B_1$
	q_4	$q_6 \in C_1$	$q_5 \in B_1$
	q_5	$q_6 \in C_1$	$q_2 \in A_1$
C_1	q_6	$q_4 \in B_1$	$q_5 \in B_1$

Pas k = 2:

→ Verificăm toate perechile de stări aflate în A_1 și observăm că starea q_0 este separabilă de stările din $\{q_1, q_2\}$ (atât pe coloana „a”, cât și pe coloana „b”).

→ Verificăm toate perechile de stări aflate în B_1 și observăm că starea q_5 este separabilă de stările din $\{q_3, q_4\}$ (pe coloana „b”).

→ Partiția C_1 conține o singură stare, deci aici nu avem ce compara.

Partițiile	δ	a	b
A_2	q_0	$q_1 \in B_2$	$q_3 \in C_2$
B_2	q_1	$q_3 \in C_2$	$q_2 \in B_2$
	q_2	$q_3 \in C_2$	$q_2 \in B_2$
C_2	q_3	$q_6 \in E_2$	$q_5 \in D_2$
	q_4	$q_6 \in E_2$	$q_5 \in D_2$
D_2	q_5	$q_6 \in E_2$	$q_2 \in B_2$
E_2	q_6	$q_4 \in C_2$	$q_5 \in D_2$

Pas k = 3:

→ Verificăm perechea de stări din partiția B_2 și observăm că q_1 și q_2 rămân echivalente (pentru fiecare coloană în parte, cele două stări din pereche duc către aceeași partiție).

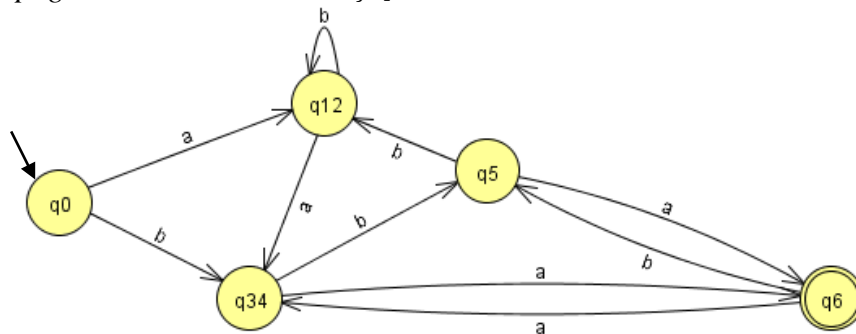
→ Verificăm perechea de stări din partiția C_2 și observăm că q_3 și q_4 rămân echivalente (pentru fiecare coloană în parte, cele două stări din pereche duc către aceeași partiție).

→ Partițiile A_2 , D_2 și E_2 conțin fiecare câte o singură stare, deci aici nu avem ce compara.

Nicio partiție nu s-a mai modificat, deci algoritmul se termină cu concluzia stărilor echivalente: $q_1 \equiv q_2$ și $q_3 \equiv q_4$.

Automatul AFD minimal obținut va avea $Q = \{q_0, q_{12}, q_{34}, q_5, q_6\}$ și $F = \{q_6\}$.

Tranzițiile le desenăm conform automatului inițial, dar ținând cont de această grupare a stărilor. [vezi pag 30, înainte de observații]



➤ **Algoritm: Minimizare AFD (metoda 2: cu teorema Myhill-Nerode)**

- Vom construi un tabel, pe linii și pe coloane având stările automatului AFD complet definit. Vom completa tabelul (triunghiul de sub diagonală 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 λ (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 Σ :

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.

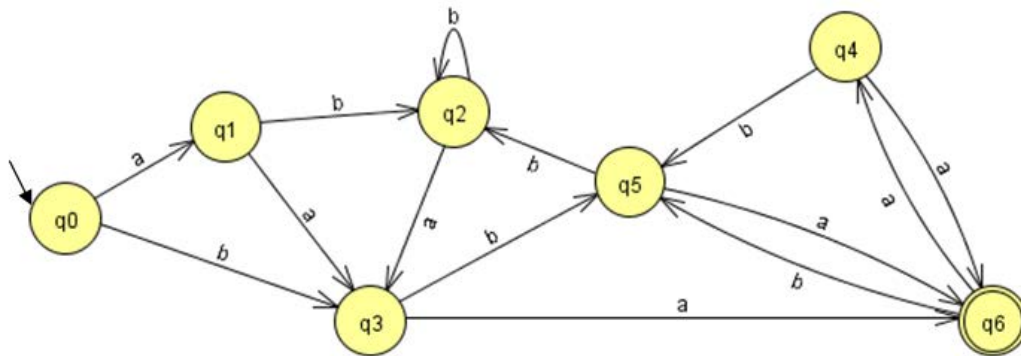
- Apoi desenăm AFD-ul minimal astfel:
 - Desenăm întâi toate stările, grupându-le pe cele echivalente între ele într-o singură stare a AFD-ului minimal.
 - **Starea inițială** este grupul format din stările echivalente cu fosta stare inițială q_0 .
 - **Stările finale** sunt grupurile formate din stări 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: [valabile pentru metoda 1 și metoda 2 ale algoritmului de minimizare AFD]

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 construieți un AFD minimal echivalent.



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

Pas 0: Căutăm perechile de stări separabile prin cuvântul λ de lungime zero. Avem o singură stare finală în acest exemplu, deci ea va fi separabilă prin λ de toate celelalte stări, care sunt nefinale. Completăm în tabel, pentru toate perechile (q_6, q_i) , $0 \leq i \leq 5$.

	q0	q1	q2	q3	q4	q5	q6
q0							
q1							
q2							
q3							
q4							
q5							
q6	λ	λ	λ	λ	λ	λ	

Pas 1: Căutăm cuvinte de lungime 1.

Avem $\delta(q_0, a) = q_1 \notin F$; $\delta(q_1, a) = q_3 \notin F$; $\delta(q_2, a) = q_3 \notin F$
și $\delta(q_3, a) = q_6 \in F$; $\delta(q_4, a) = q_6 \in F$; $\delta(q_5, a) = q_6 \in F$.

Rezultă că orice stare din mulțimea $\{q_0, q_1, q_2\}$ va fi separabilă de orice stare din mulțimea $\{q_3, q_4, q_5\}$ prin cuvântul "a".

(Observăm că toate tranzițiile cu "b" merg către stări nefinale, deci nu va exista nicio pereche de stări care să fie separabile prin cuvântul "b".)

Obs: E posibil să fie separabile și prin altceva, dar este suficient să găsim un singur cuvânt.

	q0	q1	q2	q3	q4	q5	q6
q0							
q1							
q2							
q3	a	a	a				
q4	a	a	a				
q5	a	a	a				
q6	λ	λ	λ	λ	λ	λ	

Pas 2: Căutăm cuvinte de lungime 2.

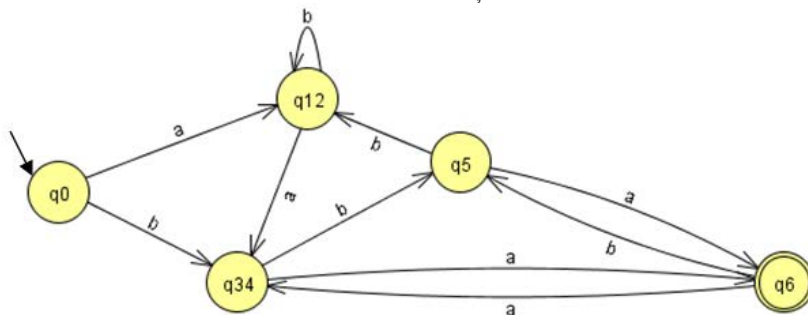
Dacă $(q_i, q_j) \xrightarrow{x \in \Sigma} (q_s, q_t)$	și $q_s \not\equiv_w q_t$	Rezultă: $q_i \not\equiv_{xw} q_j$
$(q_1, q_0) \xrightarrow{a} (q_3, q_1)$	$q_3 \not\equiv_a q_1$	$\Rightarrow q_1 \not\equiv_{aa} q_0$
$(q_2, q_0) \xrightarrow{a} (q_3, q_1)$	$q_3 \not\equiv_a q_1$	$\Rightarrow q_2 \not\equiv_{aa} q_0$
$(q_2, q_1) \xrightarrow{a} (q_3, q_3)$ $(q_2, q_1) \xrightarrow{b} (q_2, q_2)$	(q_3, q_3) și (q_2, q_2) sunt perechi de stări echivalente	$\Rightarrow q_2 \equiv q_1$
$(q_4, q_3) \xrightarrow{a} (q_6, q_6)$ $(q_4, q_3) \xrightarrow{b} (q_5, q_5)$	(q_6, q_6) și (q_5, q_5) sunt perechi de stări echivalente	$\Rightarrow q_4 \equiv q_3$
$(q_5, q_3) \xrightarrow{a} (q_6, q_6)$ $(q_5, q_3) \xrightarrow{b} (q_5, q_2)$	(q_6, q_6) stări echivalente, dar $q_5 \not\equiv_a q_2$	$\Rightarrow q_5 \not\equiv_{ba} q_3$
$(q_5, q_4) \xrightarrow{a} (q_6, q_6)$ $(q_5, q_4) \xrightarrow{b} (q_5, q_2)$	(q_6, q_6) stări echivalente, dar $q_5 \not\equiv_a q_2$	$\Rightarrow q_5 \not\equiv_{ba} q_4$

	q0	q1	q2	q3	q4	q5	q6
q0							
q1	aa						
q2	aa	\emptyset					
q3	a	a	a				
q4	a	a	a	\emptyset			
q5	a	a	a	ba	ba		
q6	λ	λ	λ	λ	λ	λ	

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.

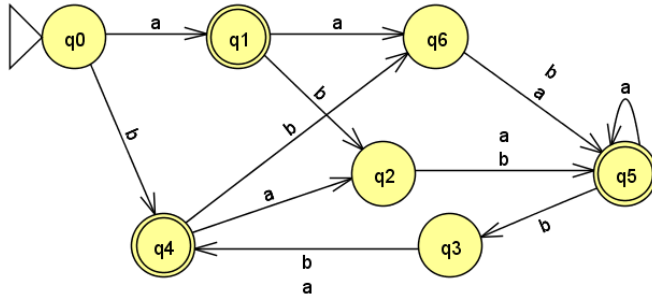
Observăm că nu există nicio stare plecând din care să nu avem drum până într-o stare finală, deci nu avem de eliminat nicio stare și acesta este automatul minimal echivalent cu cel dat.



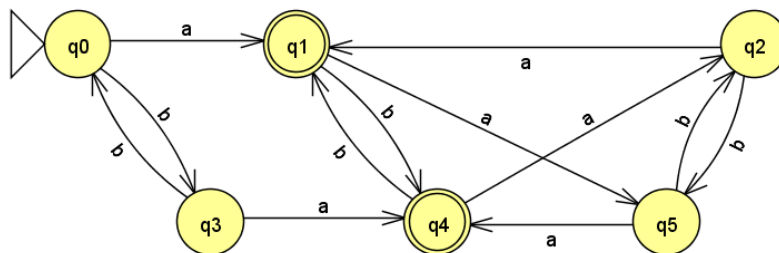
~ Temă ~

EX_1: Se dau următoarele AFD-uri. Construiți AFD minimale echivalente cu ele.

(a)



(b)



Care este limbajul recunoscut de AFD-ul minimal obținut la (b)?

~ Seminar 6 ~

➤ 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 neterminale	Q = mulțimea de stări
T = mulțimea de simboluri terminale	Σ = alfabetul de intrare
$S \in N$ simbolul de start	$q_0 \in Q$ starea inițială
- 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> $q_A \in F$)

- Mod funcționare gramatică:
 - Se începe de la cuvântul format doar din simbolul de start S .
 - Cât timp e posibil, se aplică un pas de derivare:
 - o se alege un neterminal din cuvântul curent și
 - o se alege o producție care are ca membru stâng acest neterminal,
 - o apoi se înlocuiește în cuvântul curent acel neterminal (o singură apariție a sa) cu membrul drept al acelei producții și se obține noul cuvânt curent.
 - Când cuvântul nu mai conține niciun neterminal înseamnă că s-a terminat generarea (și acel cuvânt obținut face parte din limbajul generat de gramatică).

Exemplu:

Pentru cuvântul $\alpha X \beta$, cu $X \in N$ și $\alpha, \beta \in (N \cup T)^*$

și producția $X \rightarrow w$, cu $X \in N$ și $w \in (N \cup T)^*$,
rezultă cuvântul $\alpha w \beta$.

- Exemple:**

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

$$S \rightarrow aS \mid \lambda$$

$$L2 = \{a^n \mid n \geq 1\}$$

$$S \rightarrow aS \mid a$$

$$L3 = \{a^{2n} \mid n \geq 0\}$$

$$S \rightarrow aA \mid \lambda$$

$$A \rightarrow aS$$

$$L4 = \{a^{2n} \mid n \geq 1\}$$

$$S \rightarrow aA$$

$$A \rightarrow aS \mid a$$

$$L5 = \{a^{2n+1} \mid n \geq 0\}$$

$$S \rightarrow aA \mid a$$

$$A \rightarrow aS$$

$$L6 = \{a^{2n+1} \mid n \geq 1\}$$

$$S \rightarrow aA$$

$$A \rightarrow aS \mid aB$$

$$B \rightarrow a$$

sau

$$S \rightarrow aA$$

$$A \rightarrow aB$$

$$B \rightarrow aA \mid a$$

$$L7 = \{a^{2n}b^{3k} \mid n \geq 1, k \geq 0\}$$

$$S \rightarrow aA \quad (1)$$

$$A \rightarrow aS \quad (2) \mid aB \quad (3)$$

$$B \rightarrow bC \quad (4) \mid \lambda \quad (5)$$

$$C \rightarrow bD \quad (6)$$

$$D \rightarrow bB \quad (7)$$

- Verificare generare cuvânt de către gramatică:**

Verificare generare cuvânt a^2b^3 :

$$S \quad (1) \Rightarrow aA \quad (3) \Rightarrow aaB \quad (4) \Rightarrow aabC \quad (6) \Rightarrow aabbD \quad (7) \Rightarrow aabbbB \quad (5) \Rightarrow aabbb$$

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

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

Adică membrul stâng al producției este un neterminal, iar membrul drept al producției poate fi cuvântul vid sau un cuvânt de orice lungime (finită) care conține oricâte neterminale și/sau terminale amestecate oricum.

- Exemple:**

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

$$S \rightarrow aSb \quad (1) \mid \lambda \quad (2)$$

Pentru a genera cuvântul a^3b^3 :

$$S \quad (1) \Rightarrow aSb \quad (1) \Rightarrow aaSbb \quad (1) \Rightarrow aaaSbbb \quad (2) \Rightarrow aaabbb$$

$$L2 = \{a^n b^n \mid n \geq 1\}$$

$$S \rightarrow aSb \mid ab$$

$$L3 = \{a^n b^{2n} \mid n \geq 1\}$$

$$S \rightarrow aSbb \mid abb$$

$$L4 = \{a^{2n} b^k c^{3n} \mid n \geq 0, k \geq 1\}$$

$$S \rightarrow aaSccc \mid bA$$

$$A \rightarrow bA \mid \lambda$$

sau

$$S \rightarrow aaSccc \mid A$$

$$A \rightarrow bA \mid b$$

$$L5 = \{a^{2n} b^k c^{3n} \mid n \geq 1, k \geq 1\}$$

$$S \rightarrow aaSccc \mid aaAccc$$

$$A \rightarrow bA \mid b$$

$$L6 = \{a^n b^k c^{2k} d^n \mid n \geq 1, k \geq 0\}$$

$$S \rightarrow aSd \mid aAd$$

$$A \rightarrow bAcc \mid \lambda$$

$$L7 = \{a^n b^k c^{2k} d^n \mid n \geq 0, k \geq 1\}$$

$$S \rightarrow aSd \mid A$$

$$A \rightarrow bAcc \mid bcc$$

sau

$$S \rightarrow aSd \mid bAcc$$

$$A \rightarrow bAcc \mid \lambda$$

$$L8 = \{a^n b^{2n} c^{2k} d^k \mid n \geq 0, k \geq 1\}$$

$$S \rightarrow AB$$

$$A \rightarrow aAbb \mid \lambda$$

$$B \rightarrow ccBd \mid ccd$$

➤ Forma Normală Chomsky

O gramatică este (scrisă) în **forma normală Chomsky** dacă are doar producții care au în membrul stâng un neterminat, 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 λ 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

$$\alpha X \beta \Rightarrow \alpha w \beta \text{ cu } \alpha, w, \beta \in T^*.$$

(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.)

Ideea intuitivă: Toate simbolurile din $T \cup \{\lambda\}$ sunt utilizabile. Găsim mulțimea simbolurilor din N care sunt **utilizabile**:

-- Plecăm de la cele care au cel puțin o producție cu membrul drept (de tip “caz de oprire”) în T^* .

-- Iar apoi tot căutăm alte neterminale care au cel puțin o producție cu membrul drept în $(N \cup T)^*$, dar în care toate neterminalele fac deja parte din mulțimea celor utilizabile.

- b) Se elimină simbolurile și producțiile “**inaccesibile**”.

Def: $X \in (N \cup T)$ este un simbol “inaccesibil” dacă nu există nicio derivare

$$S \Rightarrow^* \alpha X \beta, \text{ cu } \alpha, \beta \in (N \cup T)^*.$$

(Adică plecând din S și aplicând oricât de multe producții, nu putem ajunge să generăm un cuvânt care să conțină simbolul X.)

Ideea intuitivă: Găsim mulțimea simbolurilor din N care sunt **accesibile**:

-- Plecăm de la neterminalele care apar în membrul drept al producțiilor lui S.

-- Apoi tot căutăm alte neterminale care apar în membrul drept al producțiilor acelor neterminale care fac deja parte din mulțimea celor accesibile.

- **Exemplu:** Aplicați algoritmul de transformare în F.N.Chomsky pe gramatica:

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

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

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

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

$$D \rightarrow BAd \mid B$$

$$E \rightarrow Ea \mid bbE$$

$$F \rightarrow abc$$

Pas 1: Aplicăm *algoritmul de reducere*.

a) Calculăm N_1 mulțimea neterminalelor *utilizabile*.

$A \in N_1$ (pentru că A are cel puțin o producție cu membrul drept în T^* , de fapt are 3)

$B \in N_1$ (pentru că $B \rightarrow \lambda$ și $\lambda \in T^*$)

$C \in N_1$ (pentru că $C \rightarrow dcabb$ și $dcabb \in T^*$)

$F \in N_1$ (pentru că $F \rightarrow abc$ și $abc \in T^*$)

$S \in N_1$ (pentru că $S \rightarrow aABa$ și $aABa \in (T \cup N_1)^*$, la fel și pentru $S \rightarrow bbAC$)

$D \in N_1$ (pentru că $D \rightarrow B$ și $B \in T^*$, la fel și pentru $D \rightarrow BAd$)

$E \notin N_1$ (pentru că E nu are nicio producție cu membrul drept în $(T \cup N_1)^*$) \Rightarrow eliminăm din gramatică neterminalul E și toate producțiile în care apare acesta și obținem:

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

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

$B \rightarrow \lambda$

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

$D \rightarrow BAd \mid B$

$F \rightarrow abc$

b) Calculăm N_2 mulțimea neterminalelor *accesibile*.

$S \in N_2$ (pentru că S este simbolul de start al gramaticii)

$A \in N_2$ și $B \in N_2$ (pentru că $S \in N_2$ și $S \rightarrow aABa$)

$C \in N_2$ și $D \in N_2$ (pentru că $S \in N_2$ și $S \rightarrow CD$)

$F \notin N_2$ (pentru că nu există niciun neterminal în N_2 care să aibă o producție în care să apară F în membrul drept) \Rightarrow eliminăm din gramatică neterminalul F și toate producțiile în care apare acesta și obținem:

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

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

$B \rightarrow \lambda$

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

$D \rightarrow BAd \mid B$

▪ **Pas 2:** Se elimină λ -producțiile.

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

\rightarrow 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.

Pot exista două situații:

(i) Dacă neterminalul care are o λ -producție NU are și alte producții, atunci

- va fi eliminată producția lui

- toate producțiile care au ca membru drept un cuvânt de lungime minim 2 în care apare acest neterminal \rightarrow vor fi înlocuite prin eliminarea neterminalului din cuvinte

- dacă membrul drept avea lungime 1 (adică era doar acest neterminal), atunci se înlocuiește cu λ (dar și această producție va fi ulterior eliminată tot la „pasul 2”)

- (ii) Dacă neterminalul care are o λ -producție are și alte producții, atunci
- va fi eliminată doar λ -producția lui.
 - toate producțiile care au ca membru drept un cuvânt de lungime minim 2 în care apare acest neterminal \rightarrow vor fi înlocuite atât de varianta în care cuvântul conține neterminalul, cât și de varianta în care neterminalul este eliminat din cuvânt.
 - dacă membrul drept avea lungime 1 (adică era doar acest neterminal), atunci această producție este o redenumire și va fi eliminată la „pasul 3”.

• **Exemplu:**

Pas 2: Se elimină λ -producțiile.

Caz (i) \rightarrow Eliminăm $B \rightarrow \lambda$ și neterminalul $B \Rightarrow$ în toate producțiile în care apare B în membrul drept, **înlocuim** B cu λ (în cuvintele de lungime 1) sau îl **eliminăm** de tot pe B (în cuvintele de lungime ≥ 2).

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

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

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

$D \rightarrow Ad \mid \lambda$

Caz (ii)

\rightarrow Eliminăm $A \rightarrow \lambda \Rightarrow$ pentru toate producțiile în care apare A în membrul drept, **păstrăm** și variantele cu A (pentru că A are și alte producții non-lambda), dar **adăugăm** și producțiile în care înlocuim A cu λ (în cuvintele de lungime 1) sau îl eliminăm de tot pe A (în cuvintele de lungime ≥ 2).

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

$A \rightarrow bc \mid d$

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

$D \rightarrow Ad \mid d \mid \lambda$

\rightarrow Eliminăm $C \rightarrow \lambda \Rightarrow$ pentru toate producțiile în care apare C în membrul drept, **păstrăm** și variantele cu C (pentru că C are și alte producții non-lambda), dar **adăugăm** și producțiile în care înlocuim C cu λ (în cuvintele de lungime 1) sau îl eliminăm de tot pe C (în cuvintele de lungime ≥ 2).

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

$A \rightarrow bc \mid d$

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

$D \rightarrow Ad \mid d \mid \lambda$

\rightarrow Eliminăm $D \rightarrow \lambda \Rightarrow$ pentru toate producțiile în care apare D în membrul drept, **păstrăm** și variantele cu D (pentru că D are și alte producții non-lambda), dar **adăugăm** și producțiile în care înlocuim D cu λ (în cuvintele de lungime 1) sau îl eliminăm de tot pe D (în cuvintele de lungime ≥ 2).

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

$A \rightarrow bc \mid d$

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

$D \rightarrow Ad \mid d$

!! Gramatica trebuie să genereze λ (pentru că avem $S \rightarrow \lambda$), dar simbolul de start NU are voie să apară în membrul drept al nici unei producții (avem $C \rightarrow S$). \Rightarrow **Adăugăm un nou simbol de start** S' cu producțiile ($S' \rightarrow S \mid \lambda$).

\rightarrow Eliminăm $S \rightarrow \lambda \Rightarrow$ pentru toate producțiile în care apare S în membrul drept, **păstrăm** și variantele cu S (pentru că S are și alte producții non-lambda), dar **adăugăm** și producțiile în care înlocuim S cu λ (în cuvintele de lungime 1) sau îl eliminăm de tot pe S (în cuvintele de lungime ≥ 2).

- *Observăm* că ar fi reapărut λ -producția $C \rightarrow \lambda$, dar nu o mai introducem pentru că am avut-o deja și am eliminat-o, deci am obținut deja toate producțiile care rezultau din eliminarea ei.

$S' \rightarrow S \mid \lambda$

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

$A \rightarrow bc \mid d$

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

$D \rightarrow Ad \mid d$

▪ **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

- Dacă nu, atunci eliminăm toate producțiile pe care le avea.

- Dacă mai apare în cuvinte de lungime minim 2, arunci trebuie să i le păstrăm

Dacă apare în cuvânt de lungime 1, înseamnă că este tot o redenumire și va fi eliminată tot la „pasul 3”.

• **Exemplu:**

Pas 3: *Se elimină redenumirile.*

\rightarrow Eliminăm $C \rightarrow A \Rightarrow$ Pentru toate producțiile $A \rightarrow \alpha$, adăugăm producțiile $C \rightarrow \alpha$.

$S' \rightarrow S \mid \lambda$

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

$A \rightarrow bc \mid d$

$C \rightarrow bc \mid d \mid dcabb \mid S$

$D \rightarrow Ad \mid d$

\rightarrow Eliminăm $S \rightarrow D \Rightarrow$ Pentru toate producțiile $D \rightarrow \alpha$, adăugăm producțiile $S \rightarrow \alpha$.

$S' \rightarrow S \mid \lambda$

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

$A \rightarrow bc \mid d$

$C \rightarrow bc \mid d \mid dcabb \mid S$

$D \rightarrow Ad \mid d$

\rightarrow Eliminăm $S \rightarrow C \Rightarrow$ Pentru toate producțiile $C \rightarrow \alpha$, adăugăm producțiile $S \rightarrow \alpha$ (atenție, adăugăm doar producțiile care nu există deja).

Observăm că ar apărea și producția $S \rightarrow S$, dar nu o adăugăm pentru că e inutilă.

$S' \rightarrow S \mid \lambda$

$S \rightarrow aAa \mid aa \mid CD \mid bc \mid dcabb \mid Ad \mid d \mid bbAC \mid bbA \mid bbC \mid bb$

$A \rightarrow bc \mid d$

$C \rightarrow bc \mid d \mid dcabb \mid S$
 $D \rightarrow Ad \mid d$

→ Eliminăm $C \rightarrow S \Rightarrow$ Pentru toate producțiile $S \rightarrow \alpha$, adăugăm producțiile $C \rightarrow \alpha$ (care nu există deja).

$S' \rightarrow S \mid \lambda$
 $S \rightarrow aAa \mid aa \mid CD \mid bc \mid dcabb \mid Ad \mid d \mid bbAC \mid bbA \mid bbC \mid bb$
 $A \rightarrow bc \mid d$
 $C \rightarrow bc \mid d \mid dcabb \mid aAa \mid aa \mid CD \mid Ad \mid bbAC \mid bbA \mid bbC \mid bb$
 $D \rightarrow Ad \mid d$

Observație: sortăm alfabetic cuvintele de pe fiecare rând, pentru a fi mai ușor de urmărit în continuare.

$S' \rightarrow S \mid \lambda$
 $S \rightarrow aa \mid aAa \mid Ad \mid bb \mid bbA \mid bbAC \mid bbC \mid bc \mid CD \mid d \mid dcabb$
 $A \rightarrow bc \mid d$
 $C \rightarrow aa \mid aAa \mid Ad \mid bb \mid bbA \mid bbAC \mid bbC \mid bc \mid CD \mid d \mid dcabb$
 $D \rightarrow Ad \mid d$

Acum se vede clar că neterminalele S și C au exact aceleași producții, deci putem simplifica gramatica prin eliminarea neterminalului C și a producțiilor lui, și prin înlocuirea lui C cu S în toate celelalte apariții.

$S' \rightarrow S \mid \lambda$
 $S \rightarrow aa \mid aAa \mid Ad \mid bb \mid bbA \mid bbAS \mid bbS \mid bc \mid d \mid dcabb \mid SD$
 $A \rightarrow bc \mid d$
 $D \rightarrow Ad \mid d$

→ Eliminăm $S' \rightarrow S \Rightarrow$ Pentru toate producțiile $S \rightarrow \alpha$, adăugăm producțiile $S' \rightarrow \alpha$.

$S' \rightarrow aa \mid aAa \mid Ad \mid bb \mid bbA \mid bbAS \mid bbS \mid bc \mid d \mid dcabb \mid SD \mid \lambda$
 $S \rightarrow aa \mid aAa \mid Ad \mid bb \mid bbA \mid bbAS \mid bbS \mid bc \mid d \mid dcabb \mid SD$
 $A \rightarrow bc \mid d$
 $D \rightarrow Ad \mid d$

- **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.

- **Exemplu:**

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

$$\begin{aligned} S' &\rightarrow X_1X_1 \mid X_1AX_1 \mid AX_4 \mid X_2X_2 \mid X_2X_2A \mid X_2X_2AS \mid X_2X_2S \mid X_2X_3 \mid d \mid X_4X_3X_1X_2X_2 \mid SD \mid \lambda \\ S &\rightarrow X_1X_1 \mid X_1AX_1 \mid AX_4 \mid X_2X_2 \mid X_2X_2A \mid X_2X_2AS \mid X_2X_2S \mid X_2X_3 \mid d \mid X_4X_3X_1X_2X_2 \mid SD \\ A &\rightarrow X_2X_3 \mid d \\ D &\rightarrow AX_4 \mid d \\ X_1 &\rightarrow a \quad ; \quad X_2 \rightarrow b \quad ; \quad X_3 \rightarrow c \quad ; \quad X_4 \rightarrow d \end{aligned}$$

- **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.

- **Exemplu:**

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

$$\begin{aligned} S' &\rightarrow X_1X_1 \mid X_1Y_1 \mid AX_4 \mid X_2X_2 \mid X_2Y_2 \mid X_2Y_3 \mid X_2Y_5 \mid X_2X_3 \mid d \mid X_4Y_6 \mid SD \mid \lambda \\ S &\rightarrow X_1X_1 \mid X_1Y_1 \mid AX_4 \mid X_2X_2 \mid X_2Y_2 \mid X_2Y_3 \mid X_2Y_5 \mid X_2X_3 \mid d \mid X_4Y_6 \mid SD \\ A &\rightarrow X_2X_3 \mid d \\ D &\rightarrow AX_4 \mid d \\ X_1 &\rightarrow a \quad ; \quad X_2 \rightarrow b \quad ; \quad X_3 \rightarrow c \quad ; \quad X_4 \rightarrow d \\ Y_1 &\rightarrow AX_1 \quad ; \quad Y_2 \rightarrow X_2A \quad ; \quad Y_3 \rightarrow X_2Y_4 \quad ; \quad Y_4 \rightarrow AS \quad ; \quad Y_5 \rightarrow X_2S \\ Y_6 &\rightarrow X_3Y_7 \quad ; \quad Y_7 \rightarrow X_1Y_8 \quad ; \quad Y_8 \rightarrow X_2X_2 \end{aligned}$$

~ Seminar 7 ~

➤ 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)

Σ alfabetul de intrare („Sigma”) (mulțime finită)

$q_0 \in Q$ starea inițială

$F \subseteq Q$ mulțimea de stări finale

Γ 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 Σ*) sau cuvântul vid λ

- **simbolul** aflat în vârful stivei (*element din mulțimea Γ*)

- 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 Γ^**)

(*stânga* cuvântului va fi vârful stivei)

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

În stivă ordinea de adăugare a literelor este de la dreapta spre stânga cuvântului.

Obs: Atenție, dacă ștergeți tot conținutul stivei, *automatul își oprește funcționarea*, pentru că nu poate aplica funcția delta dacă nu mai are parametrul al treilea (simbolul din vârful stivei). Deci asigurați-vă că nu se va întâmpla asta înainte de a fi citit tot cuvântul de intrare. În general e bine să-l păstrăm pe Z_0 la început și să-l eliminăm abia la final (eventual folosind o λ -tranziție).

• 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*).

- **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).

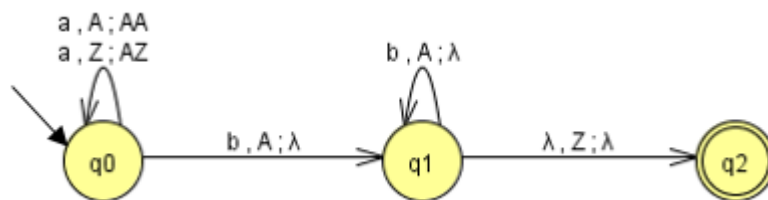
La fiecare pas, căutăm pe graf tranziția care

- pleacă din starea curentă,
- citește prima literă din cuvântul curent (sau eventual λ) 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 λ),
- iar vârful stivei se înlocuiește cu cuvântul spus de tranziție (cel de după “/” sau “;”), iar restul stivei se concatenează după.

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



Verificare acceptare cuvânt

Fie $n = 3$ deci avem cuvântul $a^3 b^3$.

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

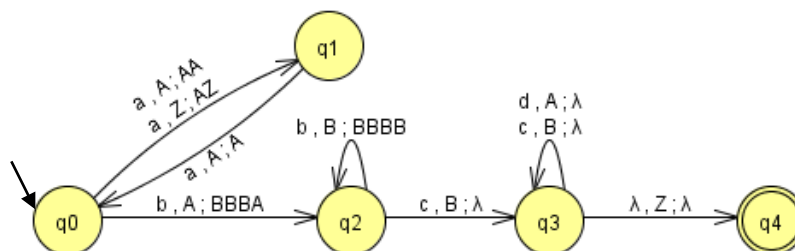
$\vdash (q_1, b^2, AAZ_0) \vdash (q_1, b, AZ_0) \vdash (q_1, \lambda, Z_0) \vdash (q_2, \lambda, \lambda), q_2 \in F \Rightarrow a^3 b^3$ cuvânt acceptat.

- **Exemple:**

$$L2 = \{a^{2n} b^k c^{3k} d^n \mid n \geq 1, k \geq 1\}$$

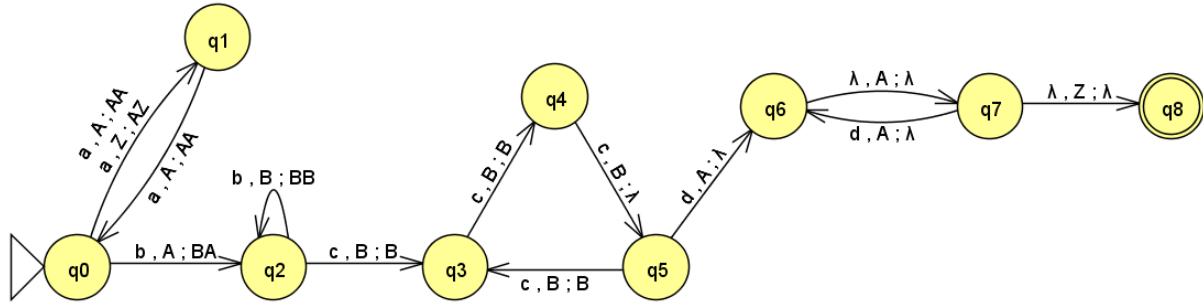
Idee rezolvare 1:

- 1) Scriem n de A în stivă: pentru fiecare 2 de a , primul adaugă un A în stivă, al doilea nu modifică stiva.
- 2) Scriem $3k$ de B în stivă: fiecare b adaugă câte 3 de B în stivă.
- 3) Ștergem $3k$ de B din stivă: fiecare c elimină câte un B din stivă.
- 4) Ștergem n de A din stivă: fiecare d elimină câte un A din stivă.



Idee rezolvare 2:

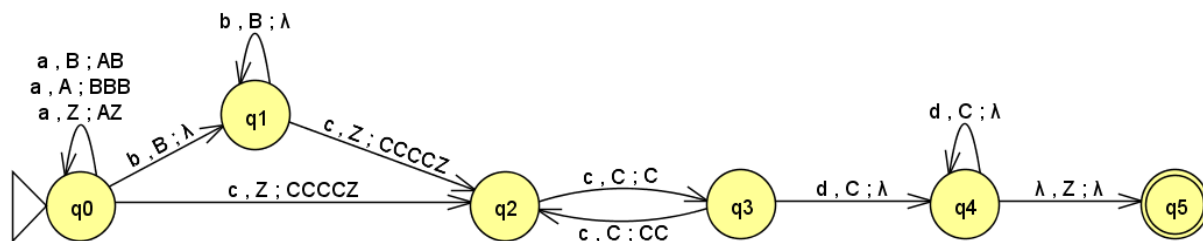
- 1) Scriem $2n$ de A în stivă: pentru fiecare 2 de a, ambii adaugă câte un A în stivă.
- 2) Scriem k de B în stivă: fiecare b adaugă câte un B în stivă.
- 3) Ștergem k de B din stivă: pentru fiecare 3 de c, primii 2 de c nu modifică stiva, iar al 3-lea c elimină un B din stivă.
- 4) Ștergem $2n$ de A din stivă: pentru fiecare 2 de A, un d elimină un A din stivă, iar o lambda-tranziție elimină al 2-lea A din stivă.



$$L3 = \{a^{2n}b^{3n}c^{2k}d^{k+3} \mid n \geq 0, k \geq 1\}$$

Idee rezolvare:

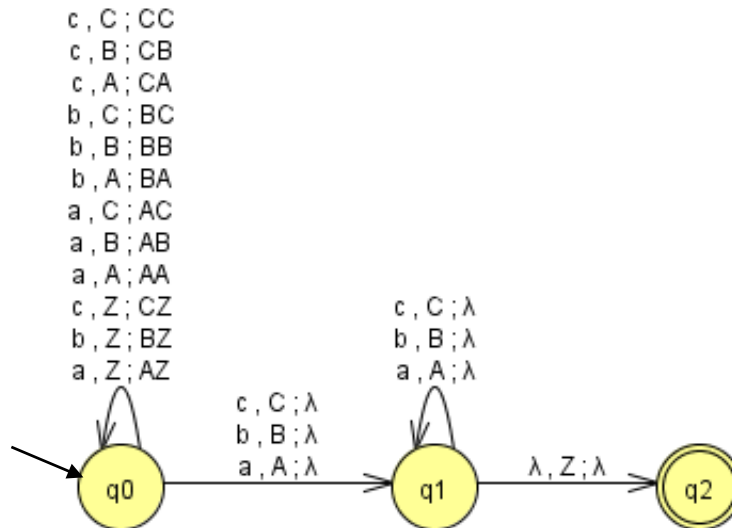
- 1) (Pentru verificarea parității a-urilor nu vom folosi circuit între 2 stări diferite, ci vom avea buclă pe aceeași stare dar vom folosi simboluri diferite în stivă.)
Scriem $3n$ de B în stivă: pentru fiecare 2 de a, primul a adaugă un A în stivă, al doilea a elimină A din stivă și adaugă 3 de B.
- 2) Ștergem $3n$ de B din stivă: fiecare b elimină câte un B din stivă.
- 3) Scriem $k+3$ de C în stivă: pentru fiecare 2 de c, primul c din pereche adaugă un C în stivă
(*excepție:* primul c din prima pereche adaugă $1+3=4$ de C), al doilea c din pereche nu modifică stiva.
- 4) Ștergem $k+3$ de C din stivă: fiecare d elimină câte un C din stivă.



$$L4 = \{w \cdot w^R \mid w \in \{a, b, c\}^+\}$$

Idee rezolvare:

- 1) Cât timp citim prima parte din cuvânt îl salvăm în stivă: fiecare literă mică a/b/c adaugă în stivă litera corespunzătoare A/B/C.
- 2) Când presupunem că am ajuns la mijlocul cuvântului (schimbăm starea), cât timp citim a doua parte din cuvânt verificăm oglindirea cu cuvântul din stivă: fiecare literă mică a/b/c trebuie să șteargă din stivă exact aceeași literă A/B/C.



➤ **Lema de pompare pentru limbaje 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 α din limbajul L astfel încât $|\alpha| \geq p$, $\forall p \in \mathbb{N}$. Este important ca pentru acel α 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 α în cele 5 componente (altfel spus, trebuie să poziționăm vwx în toate modurile posibile în α). 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 α , nu doar pe unele cazuri.*)

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

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

$$L2 = \{a^n b^m c^n d^m \mid n \geq 0, m \geq 0\} \quad [\text{făcut la seminar}]$$

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

$$L4 = \{a^n b^m c^r \mid n > m > r \geq 0\} \quad [\text{făcut la seminar}]$$

$$L5 = \{a^r \mid r \text{ este număr prim}\}$$

✓ **Demonstrație pentru $L1 = \{a^n b^n c^n \mid n \geq 0\} \notin CFL$**

Presupunem prin reducere la absurd că L1 este limbaj independent de context (CFL). Atunci $\exists p \in \mathbb{N}$ și putem aplica lema de pompare. (*În continuare negăm afirmația lemei.*)

Alegem cuvântul $\alpha = a^p b^p c^p \in L1$, cu $|\alpha| = 3p \geq p, \forall p \in \mathbb{N}$. Conform lemei, cuvântul poate fi scris sub forma $\alpha = u \cdot v \cdot w \cdot x \cdot y$ astfel încât $|v \cdot w \cdot x| \leq p$ și $|v \cdot x| \geq 1$, deci $1 \leq |vx| \leq p$ (*).

Observăm că subșirul $vw x$ (de lungime maxim p) nu poate conține simultan litere de a , b și c (pentru că lungimea ar depăși cele p litere de b) $\Rightarrow vwx$ poate conține doar 1 sau 2 tipuri de litere, deci există 5 cazuri:

Caz I: $vw x \in a^* \Rightarrow vx = a^k$, din (*) $\Rightarrow 1 \leq k \leq p$.

Alegem $i = 2 \Rightarrow \beta = u \cdot v^2 \cdot w \cdot x^2 \cdot y = a^{p+k} b^p c^p \in L1$

$\Leftrightarrow |\beta|_a = |\beta|_b \Leftrightarrow p + k = p \Leftrightarrow k = 0$, contradicție cu $1 \leq k$ (rel 1).

Caz II: $vw x \in b^* \Rightarrow vx = b^k$ și

Caz III: $vw x \in c^* \Rightarrow vx = c^k$

se tratează analog cu Caz I \Rightarrow contradicții (rel 2 și rel 3).

Caz IV: $vw x \in a^* b^* \Rightarrow vx = a^k b^s$, din (*) $\Rightarrow 1 \leq k + s \leq p$.

Alegem $i = 0 \Rightarrow \beta = u \cdot v^0 \cdot w \cdot x^0 \cdot y = u \cdot w \cdot y = a^{p-k} b^{p-s} c^p \in L1$

$\Leftrightarrow |\beta|_a = |\beta|_b = |\beta|_c \Leftrightarrow p - k = p - s = p$

$\Leftrightarrow k = s = 0$, contradicție cu $1 \leq k + s$ (rel 4).

Caz V: $vw x \in b^* c^* \Rightarrow vx = b^k c^s$

se tratează analog cu Caz IV \Rightarrow contradicție (rel 5).

Concluzie: Din relațiile 1 – 5 (am reușit să obținem contradicții pentru *toate* 5 cazurile de descompunere) rezultă că presupunerea făcută este falsă $\Rightarrow L1 \notin CFL$. ■

✓ Demonstratie pentru $L3 = \{z \cdot z \mid z \in \{a, b\}^*\} \notin CFL$

Presupunem că $L3$ este limbaj independent de context, rezultă că există acel număr natural p din lema (numit lungimea de pompă).

Alegem $\alpha = a^p b^p a^p b^p = \underbrace{a \dots a}_p \underbrace{b \dots b}_p \underbrace{a \dots a}_p \underbrace{b \dots b}_p \in L3 \Rightarrow |\alpha| = 4p \geq p, \forall p \in \mathbb{N}$.

Avem $\alpha = u \cdot v \cdot w \cdot x \cdot y$ astfel încât $|v \cdot w \cdot x| \leq p$ și $|v \cdot x| \geq 1$,
 $\Rightarrow 1 \leq |vx| \leq p$ (pentru că w are voie să fie inclusiv cuvântul vid).

Caz I: Dacă $vw x$ este în prima jumătate a lui α , atunci alegem $i = 2$ și rezultă după pompă cuvântul $\beta = u \cdot v^2 \cdot w \cdot x^2 \cdot y \Rightarrow |\beta| = |\alpha| + |vx| \Rightarrow |\beta|$ este mai mare decât $|\alpha|$ cu maxim p litere (și minim o literă) \Rightarrow jumătatea cuvântului se mută spre stânga cu maxim $p/2$ poziții (și minim $1/2$), deci nu va mai fi între un “b” și un “a”, ci va fi între doi de “b” (indiferent dacă pompăm doar a-uri, doar b-uri, sau amândouă în prima jumătate a cuvântului). Rezultă că cuvântul β începe cu litera “a”, dar prima literă din a doua lui jumătate este “b”, deci cuvântul $\beta \notin L3$ (pentru că nu este de forma $z \cdot z$), contradicție cu proprietatea (3) din lema. (rel. 1)

Caz II: Dacă $vw x$ este în a doua jumătate a lui α , atunci alegem $i = 2$ și rezultă după pompă cuvântul $\beta = u \cdot v^2 \cdot w \cdot x^2 \cdot y \Rightarrow |\beta| = |\alpha| + |vx| \Rightarrow |\beta|$ este mai mare decât $|\alpha|$ cu maxim p litere (și minim o literă) \Rightarrow jumătatea cuvântului se mută spre dreapta cu maxim $p/2$ poziții (și minim $1/2$), deci nu va mai fi între un “b” și un “a”, ci va fi între doi de “a” (indiferent dacă pompăm doar a-uri, doar b-uri, sau amândouă în a doua jumătate a cuvântului). Rezultă că cuvântul β se termină cu litera “b”, dar ultima literă din prima lui jumătate este “a”, deci cuvântul $\beta \notin L3$ (pentru că nu este de forma $z \cdot z$), contradicție cu proprietatea (3) din lema. (rel. 2)

Caz III: Dacă $vw x$ intersectează mijlocul cuvântului α ($vw x$ conține cel puțin una din cele 2 litere din mijloc), atunci alegem $i = 0$ și rezultă după pompă cuvântul $\beta = u \cdot v^0 \cdot w \cdot x^0 \cdot y = u \cdot w \cdot y$ care este de forma $a^p b^{p-s} a^{p-r} b^p$, cu $1 \leq s + r \leq p$. Rezultă că cuvântul β are mai puțini de “b” în prima parte decât la final sau are mai puțini de “a” în a doua parte decât la început (sau ambele), deci nu este de forma $z \cdot z \Rightarrow \beta \notin L3$, contradicție cu proprietatea (3) din lema. (rel. 3)

Concluzie: Din relațiile 1 – 3 (am reușit să obținem contradicții pentru toate 3 cazurile de descompunere) rezultă că presupunerea făcută este falsă $\Rightarrow L3 \notin CFL$. ■

✓ Demonstrație pentru $L5 = \{a^r \mid r \text{ este număr prim}\} \notin CFL$

Presupunem că $L5$ este limbaj independent de context, rezultă că există acel număr natural p din lema.

Alegem $\alpha = a^n$, unde $n = \text{nr. prim}$, $n \geq p + 2$ (cel mai mic număr natural este 0, dar cel mai mic număr prim este 2) $\Rightarrow |\alpha| = n \geq p + 2 \geq p, \forall p \in \mathbb{N}$.

Avem $\alpha = u \cdot v \cdot w \cdot x \cdot y$ astfel încât $|v \cdot w \cdot x| \leq p$ și $|v \cdot x| \geq 1$.

Fie $vx = a^k \Rightarrow |vx| = k$.

Din relațiile de mai sus rezultă $1 \leq k = |vx| \leq |vwx| \leq p \leq n - 2$ (rel. *).

Trebuie să alegem un număr natural i pentru care după pompare rezultă cuvântul $\beta = u \cdot v^i \cdot w \cdot x^i \cdot y$ despre care trebuie să demonstrăm că NU aparține limbajului $L5$, adică să arătăm că lungimea cuvântului β **nu poate fi un număr prim**.

$$\begin{aligned} \text{Avem } |\beta| &= |u \cdot v^i \cdot w \cdot x^i \cdot y| = |u \cdot v^1 \cdot w \cdot x^1 \cdot y| + |v^{i-1} \cdot x^{i-1}| \\ &= |uvwxy| + (i - 1) * |vx| = |\alpha| + (i - 1) * |vx| = n + (i - 1) * k = n - k + i * k \end{aligned}$$

Pentru că avem nevoie să dăm un factor comun în această expresie, observăm că putem alege $i = n + 1 \Rightarrow |\beta| = n + (n + 1 - 1) * k = n + n * k = n * (k + 1)$.

Dar știm că $n = \text{nr. prim}$, deci $n \geq 2$, iar conform (*) rezultă $k + 1 \geq 2$.

Pentru că $|\beta|$ este egal cu produsul a două numere $\geq 2 \Rightarrow |\beta| \neq \text{nr. prim} \Rightarrow \beta \notin L5$, contradicție cu proprietatea (3) din lema.

Concluzie: am reușit să obținem contradicție pentru singurul caz posibil de descompunere, rezultă că presupunerea făcută este falsă $\Rightarrow L5 \notin CFL$. ■

Observație:

Am fi putut alege și $i = n - k \Rightarrow |\beta| = n - k + (n - k) * k = (n - k) * (k + 1)$, care conform inegalităților din relația (*) este tot un produs de două numere ≥ 2 .