

~ Seminar 4 ~

➤ Expresii regulate (Regex)

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

- $\emptyset, \lambda \in \text{Regex}$ și $a \in \text{Regex}, \forall a \in \Sigma$.
- Dacă $e_1, e_2 \in \text{Regex}$, atunci $(e_1 + e_2) \in \text{Regex}$. (reuniune)
- Dacă $e_1, e_2 \in \text{Regex}$, atunci $(e_1 \cdot e_2) \in \text{Regex}$. (concatenare)
- Dacă $e \in \text{Regex}$, atunci $(e^*) \in \text{Regex}$. (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:

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



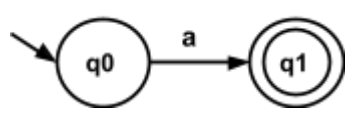
$$L = L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ și } w_2 \in L_2\}$$

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

➤ Algoritm: Transformarea Regex → AFN-λ

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

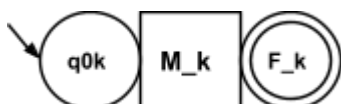
Caz i)

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

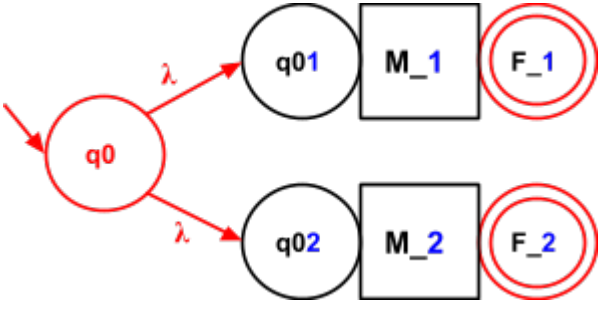
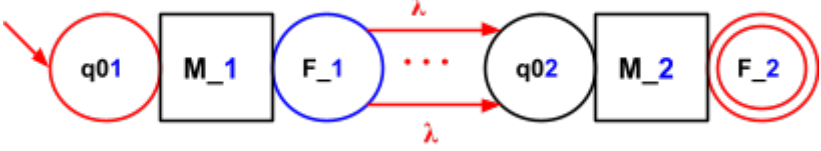
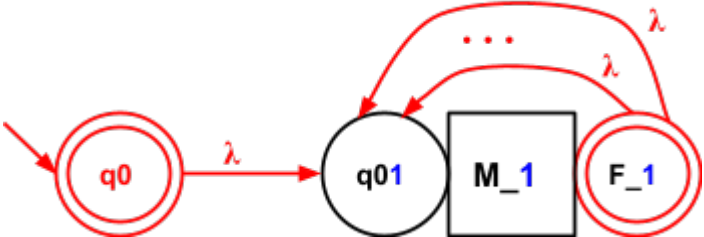
În cazurile ii), iii) și iv) presupunem că

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).

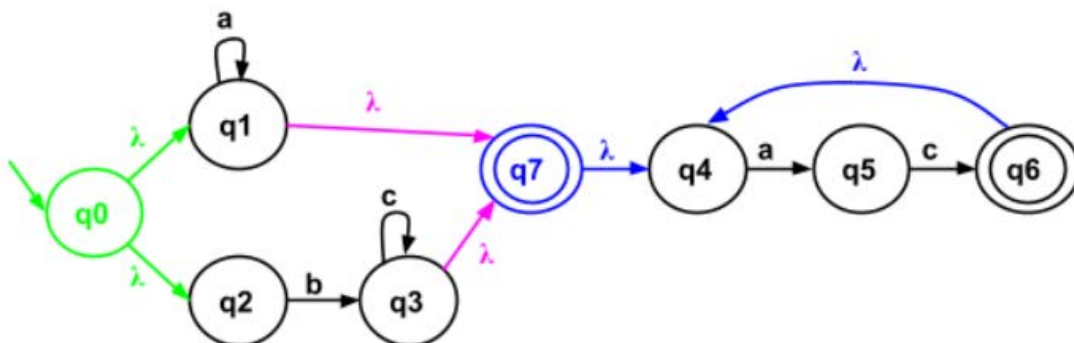
Desenăm schema unui automat punând în evidență starea inițială q_{0k} și mulțimea stărilor finale F_k . Dreptunghiul M_k include toate celelalte stări și tranzițiile automatului.



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

<p>Caz ii) (reuniune)</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) (concatenare)</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 $AF(L(e_1 \cdot e_2)) = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, q_{01}, F_2, \delta_1 \cup \delta_2 \cup \{\delta(q_{f_1}, \lambda) \ni q_{02} \mid \forall f_1 \in F_1\})$</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 algoritmii 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- $\lambda \rightarrow \text{RegEx}$

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* $et: Q \times Q \rightarrow \text{RegEx}(\Sigma)$.

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.

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

Pas 2: Dacă 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.)

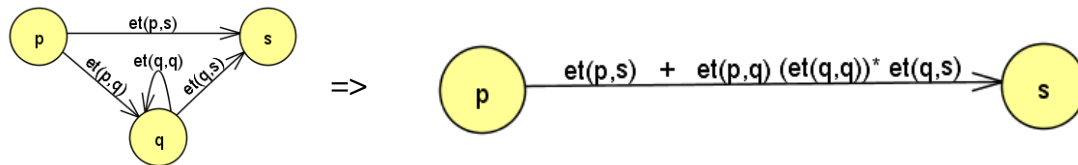
Pas 5: Atunci când rămân doar două stări, expresia obținută între starea inițială și cea finală este răspunsul final (o expresie regulată echivalentă cu automatul finit dat).

• ***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 dintre expresii conține reuniune (“+”), atunci o includem 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.

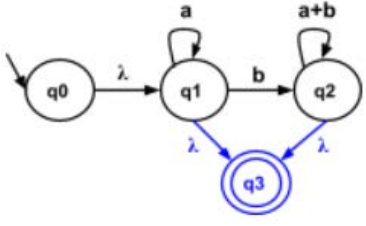
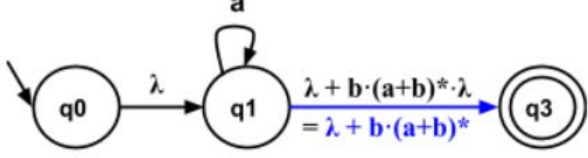
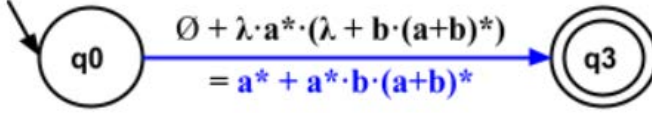
• Câteva formule utile

- (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:

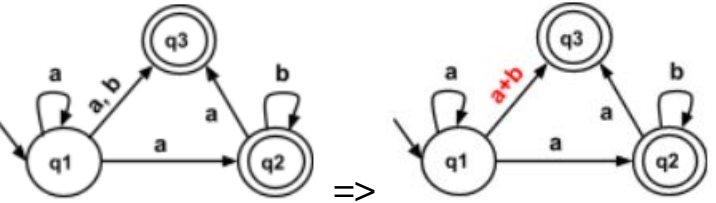
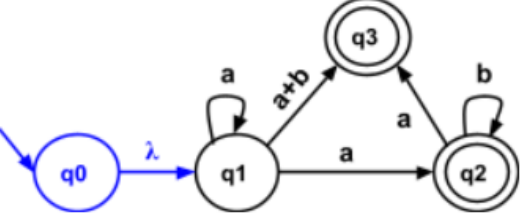
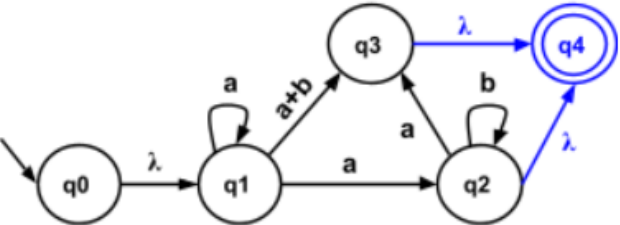
Să se transforme următorul automat finit într-o expresie regulată echivalentă.

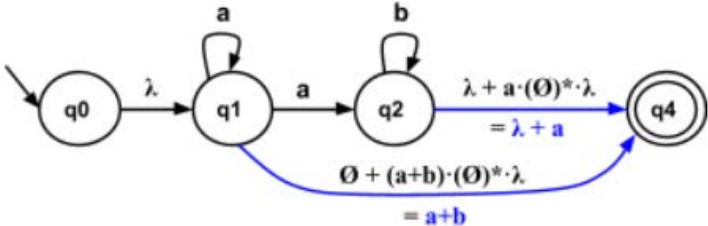
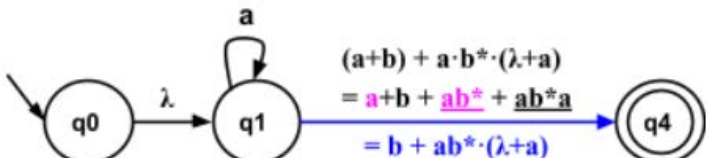
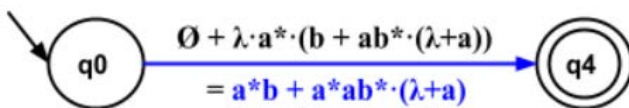
<p>Pas 1 (AF \rightarrow AFE):</p> <p>\Rightarrow Reunim tranzițiile aflate pe aceeași săgeată.</p>	
<p>Pas 2 (Verificăm <i>starea inițială</i>):</p> <ul style="list-style-type: none"> - să nu fie stare finală și - să nu vină săgeți către ea <p>\Rightarrow adăugăm o nouă stare inițială cu λ către fosta stare inițială.</p>	

<p>Pas 3 (Verificăm <i>starea finală</i>):</p> <ul style="list-style-type: none"> - să fie unica finală și - să nu plece săgeți din ea) <p>=> adăugăm o nouă unică stare finală spre care vin λ din fostele stări finale.</p>	
<p>Pas 4 (eliminăm <i>q2</i>):</p> <p>=> $et(q_1, q_3) = et(q_1, q_3) + et(q_1, q_2) \cdot (et(q_2, q_2))^* \cdot et(q_2, q_3)$</p>	
<p>Pas 4 (eliminăm <i>q1</i>):</p> <p>=> $et(q_0, q_3) = et(q_0, q_3) + et(q_0, q_1) \cdot (et(q_1, q_1))^* \cdot et(q_1, q_3)$</p>	

• **Exemplu:** [discutat la seminar]

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

<p>Pas 1 (AF \rightarrow AFE):</p> <p>=> Reunim tranzițiile aflate pe aceeași săgeată.</p>	
<p>Pas 2 (Verificăm <i>starea inițială</i>):</p> <ul style="list-style-type: none"> - să nu fie stare finală și - să nu vină săgeți către ea) <p>=> adăugăm o nouă stare inițială cu λ către fosta stare inițială.</p>	
<p>Pas 3 (Verificăm <i>starea finală</i>):</p> <ul style="list-style-type: none"> - să fie unica finală și - să nu plece săgeți din ea) <p>=> adăugăm o nouă unică stare finală spre care vin λ din fostele stări finale.</p>	

<p>Pas 4 (eliminăm q3):</p> $\Rightarrow et(q_2, q_4) = et(q_2, q_4) + et(q_2, q_3) \cdot (et(q_3, q_3))^* \cdot et(q_3, q_4)$ $\Rightarrow et(q_1, q_4) = et(q_1, q_4) + et(q_1, q_3) \cdot (et(q_3, q_3))^* \cdot et(q_3, q_4)$	
<p>Pas 4 (eliminăm q2):</p> $\Rightarrow et(q_1, q_4) = et(q_1, q_4) + et(q_1, q_2) \cdot (et(q_2, q_2))^* \cdot et(q_2, q_4)$	
<p>Pas 4 (eliminăm q1):</p> $\Rightarrow et(q_0, q_4) = et(q_0, q_4) + et(q_0, q_1) \cdot (et(q_1, q_1))^* \cdot et(q_1, q_4)$	

➤ Minimizarea AFD

[definiție extrasă din curs 7.2, pag 4]

Def: Echivalență pe cuvinte

Pentru un limbaj $L \subseteq \Sigma^*$ definim \equiv_L astfel:

$$x \equiv_L y \Leftrightarrow [\forall z \in \Sigma^* \text{ avem } xz \in L \Leftrightarrow yz \in L]$$

Obs: Cuvintele x și y nu sunt echivalente conform L dacă există un cuvânt z astfel încât *exact unul* dintre cuvintele xz și yz aparține limbajului L și celălalt nu aparține lui L .

$$x \not\equiv_L y \Leftrightarrow [\exists z \in \Sigma^* \text{ avem } xz \in L \Leftrightarrow yz \notin L]$$

➤ *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.

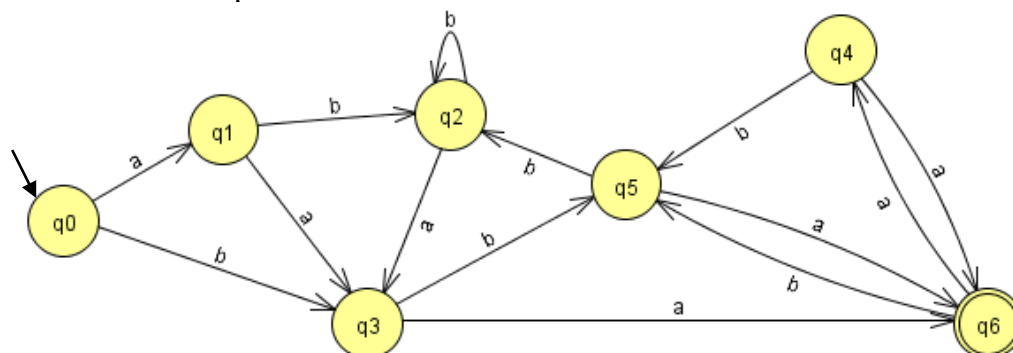
Atenție! La pasul k a) nu verificăm niciodată două stări aflate deja în partiții diferite (știm deja că stările sunt separabile), ci doar pe cele aflate în *aceeași* partiție.

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).

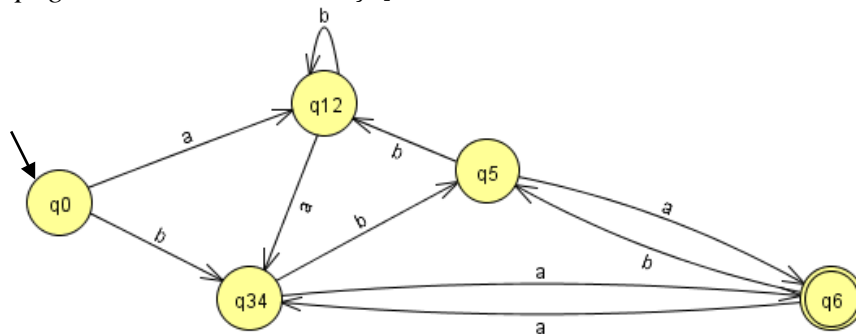
→ Verificăm perechea de stări din partiția C_2 și observăm că q_3 și q_4 rămân echivalente (pentru fiecare coloană în parte, cele două stări din pereche duc către aceeași partiție).

→ 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 10, 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 diagonala principală) pentru fiecare pereche de stări (q_i, q_j) , având $i > j$, cu un cuvânt prin care cele două stări sunt separabile. Dacă nu vom găsi un astfel de cuvânt atunci acele stări sunt echivalente.
- Vom completa tabelul căutând cuvintele recursiv, în ordinea crescătoare a lungimii lor. Cuvintele de lungime k le vom obține cu ajutorul celor de lungime $k-1$ calculate anterior.

→ **Pas 0:** Două stări sunt separabile prin cuvântul vid 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.

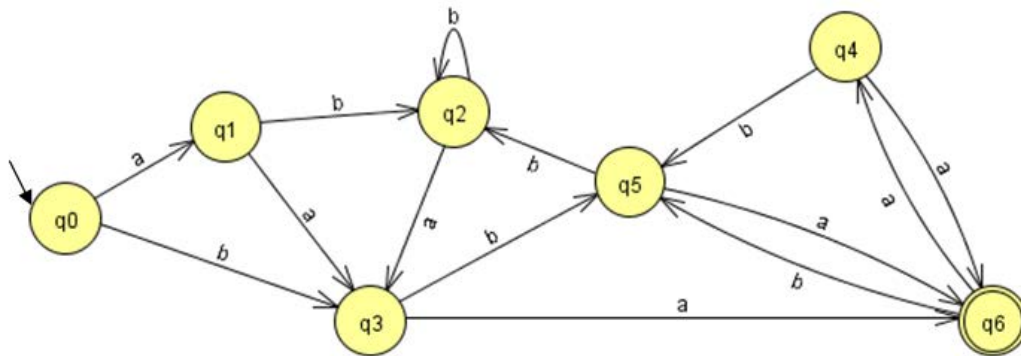
- Apoi desenăm AFD-ul minimal astfel:
 - Desenăm întâi toate stările, grupându-le pe cele echivalente între ele într-o singură stare a AFD-ului minimal.
 - **Starea inițială** este grupul format din stările echivalente cu fosta stare inițială q_0 .
 - **Stările finale** sunt grupurile formate din stări 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.

