~ Seminar 3 ~

Automatul minimal (metoda 1: cu teorema Myhill-Nerode)

Pentru explicații și exemplu → **vezi CURS 5**

Automatul minimal (metoda 2: cu partiționarea mulțimii stărilor)

Se pornește de la un AFD complet definit (adică pentru orice stare din mulțimea Q și pentru orice caracter din alfabetul Σ, funcția δ este definită), fără stări inaccesibile. Se cere un AFD echivalent (care acceptă același limbaj), care să aibă număr minim de stări.

Ideea algoritmului de minimizare a unui AFD 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.
  $\forall p,q \in Q, p \equiv q \Leftrightarrow [\forall w \in \Sigma^*, \delta(p,w) \in F \leftrightarrow \delta(q,w) \in F]$
- Două stări sunt „**separabile**" dacă există un cuvânt pentru care plecând din cele două stări ajungem într-o stare finală și într-una nefinală.
  $\forall p,q \in Q, p \not\equiv_w q \Leftrightarrow [\exists w \in \Sigma^*, \delta(p,w) \in F \leftrightarrow \delta(q,w) \notin F]$

Algoritmul de minimizare a unui AFD (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.

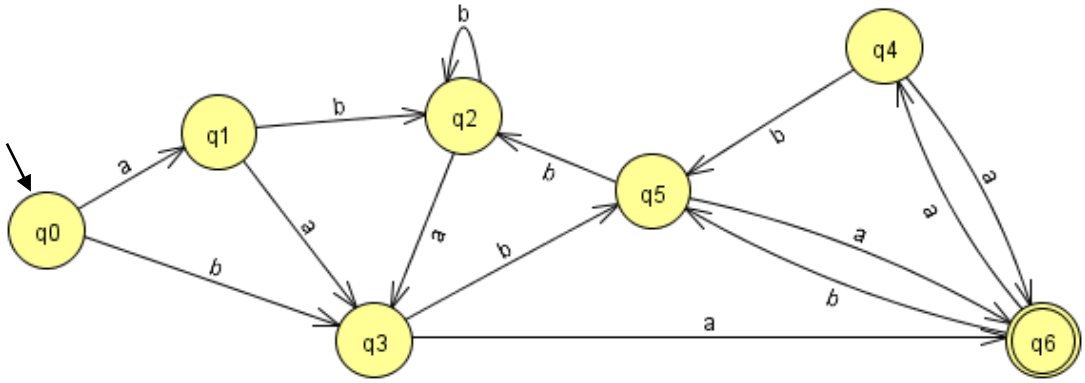
Pas 0: Împărțim mulțimea Q în două partiții, una care conține stările nefinale (A_0 = Q\F) și una care conține stările finale (B_0 = F). Orice stare din A_0 este separabilă de orice stare din B_0 prin cuvântul λ de lungime 0.
Pas k $\in \{1, 2, \dots\}$**:**
a) În cadrul fiecărei partiții X_{k-1} (cu $|X_{k-1}| \geq 2$, X∈{A, B, C,…}), verificăm pentru orice pereche de două stări ($q_i \in X_{k-1}$ *si* $q_j \in X_{k-1}$) 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 \ 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”).

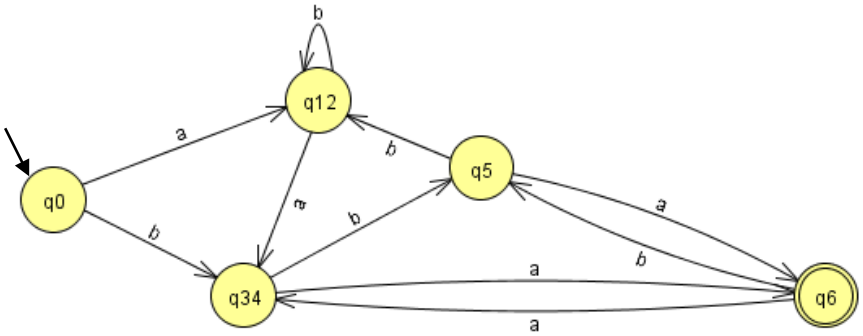
→ Partiția C_1 conține o singură stare, deci aici nu avem ce compara.

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.

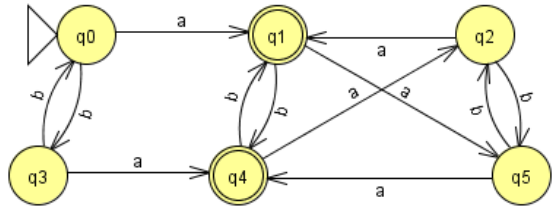
Nicio partiție nu s-a mai modificat, deci algoritmul se termină cu concluzia stărilor echivalente: $q_1 \equiv q_2$ și $q_3 \equiv q_4$.

Automatul AFD minimal obţinut va avea Q = {q_0, q_{12}, q_{34}, q_5, q_6} şi F = {q_6}.
Tranziţiile le desenăm conform automatului iniţial, dar ţinând cont de această grupare a stărilor.

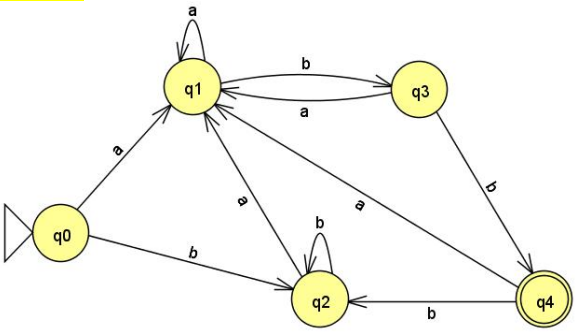


EXERCIȚII: Să se minimizeze următoarele AFD-uri.

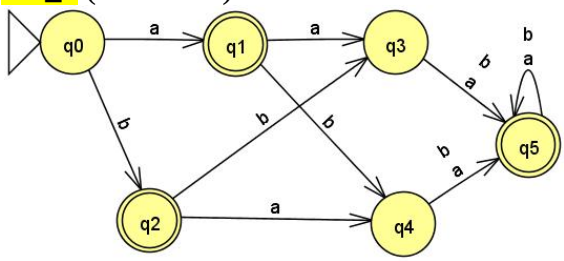
EX_1:



EX_2:



EX_3 (din curs 5):



După minimizare, pentru exemplele 1 și 3, specificați care este limbajul acceptat de acele automate obținute.

Expresii regulate

Se numeşte familia expresiilor regulate peste Σ şi se notează REX(Σ) mulţimea de cuvinte peste alfabetul $\Sigma \cup \{(,),+,\cdot,*,\varnothing,\lambda\}$ definită recursiv astfel:

i) $\varnothing, \lambda \in REX$; $a \in REX$ *pentru* $\forall a \in \Sigma$

ii) $e_1, e_2 \in REX \Rightarrow (e_1 + e_2) \in REX$

iii) $e_1, e_2 \in REX \Rightarrow (e_1 \cdot e_2) \in REX$

iv) $e \in REX \Rightarrow (e^*) \in REX$

Precedenţa operaţiilor $*, \cdot, +$

Obs: Atenţie la ordinea în care se evaluează operaţiile: întâi stelarea, apoi concatenarea şi apoi reuniunea. *(Dacă vreţi să fiţi siguri că nu le încurcaţi, puteţi să faceţi o analogie cu operaţiile aritmetice, unde se evaluează întâi ridicarea la putere, apoi înmulţirea şi apoi adunarea.)*

Operaţii cu limbaje

Reuniune $L = L_1 \bigcup L_2$

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

Exemplu:
L_1 = {a, ab, abc}
L_2 = {a, bd, cd}
→ L = {a, ab, abc, bd, cd}

Concatenare $L = L_1 \cdot L_2$

$L = \{w_i w_j \mid w_i \in L_1 \text{ şi } w_j \in L_2\}$

Obs: Atenție, concatenarea NU este comutativă!
Exemplu:
L_1 = {a, ab, abc}
L_2 = {a, bd, cd}
→ L = {aa, abd, acd, aba, abbd, abcd, abca, abcbd, abccd}

Stelare $L = L_1^* = \bigcup_{n \geq 0} \left(L_1^n\right)$

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

Exemplu:
L_1 = {a, ab, abc}
→ $(L_1)^*$ = {λ ; a, ab, abc ; aa, aab, aabc, aba, abab, ababc, abca, abcab, abcabc ; …}

Obs: Cuvântul vid λ va aparţine limbajului stelat, indiferent dacă înainte aparţinea sau nu limbajului iniţial.

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

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

Obs: Atenţie la ridicarea la putere a cuvintelor! *(Să nu distribuiţi puterea ca la înmulţire, ci concatenaţi cuvântul cu el însuşi de câte ori spune puterea.)*
$(abc)^2 = abcabc$; $(abc)^2 \neq aabbcc = a^2b^2c^2$

Transformarea expresie_regulată → automat_finit

Expresie regulată	Limbaj	Automat finit
$\varnothing$	$\varnothing$	$Q=\{q_0\}, F=\varnothing, \delta=\varnothing$
λ	$\{\lambda\}$	$Q=\{q_0\}, F=\{q_0\}, \delta=\varnothing$
a	$\{a\}, a\in\Sigma$	$Q=\{q_0,q_1\}, F=\{q_1\}, \delta(q_0,a)=q_1$
e_1+e_2	$L(e_1+e_2)$ $=L(e_1)\cup L(e_2)$	$Q=Q_1\cup Q_2\cup\{q_0\}; F=F_1\cup F_2; V=V_1\cup V_2;$ $\delta=\delta_1\cup\delta_2\cup\{\delta(q_0,\lambda)=\{q_{01},q_{02}\}\}$
$e_1\cdot e_2$	$L(e_1\cdot e_2)$ $=L(e_1)\cdot L(e_2)$	$Q=Q_1\cup Q_2, q_0=q_{01}; F=F_2; V=V_1\cup V_2;$ $\delta=\delta_1\cup\delta_2\cup\{\delta(q_{f1},\lambda)=q_{02}, pentru\forall q_{f1}\in F_1\}$
$(e_1)^*$	$L(e_1^*)=(L(e_1))^*$	$Q=Q_1\cup\{q_0\}; F=\{q_0\}; V=V_1;$ $\delta=\delta_1\cup\{\delta(q_0,\lambda)=q_{01}\}\cup\{\delta(q_{f1},\lambda)=q_0, pentru\forall q_{f1}\in F_1\}$

Descrierea în cuvinte a ultimelor trei linii ale tabelului o găsiţi la secțiunea „compunerea automatelor finite” de mai jos.
Obs: La construirea automatelor compuse se ţine cont de precedenţa operaţiilor (întâi stelarea, apoi concatenarea, apoi reuniunea) şi de parantezele din expresie.

“Compunerea” automatelor finite

Reuniune

Dacă avem deja construite două automate finite (având stările iniţiale q_{01} şi q_{02} şi mulţimile stărilor finale F1 şi F2) care acceptă respectiv limbajele L1 şi L2 şi dorim să obţinem un automat finit care să accepte limbajul $L=L_1\bigcup L_2$:

- introducem o nouă stare q_0 care va fi noua stare iniţială
- adăugăm două λ-tranziţii de la q_0 spre q_{01} şi de la q_0 spre q_{02}
- noua mulţime de stări finale va fi $F=F_1\bigcup F_2$

Concatenare

Dacă avem deja construite două automate finite (având stările iniţiale q_{01} şi q_{02} şi mulţimile stărilor finale F1 şi F2) care acceptă respectiv limbajele L1 şi L2 şi dorim să obţinem un automat finit care să accepte limbajul $L=L_1\cdot L_2$:

- noua stare iniţială va fi q_{01}, starea iniţială a primului automat
- adăugăm λ-tranziţii de la fiecare stare finală din F1 spre q_{02}
- noua mulţime de stări finale va fi F = F2

Stelare

Dacă avem deja construit un automat finit (având starea iniţială q_{01} şi mulţimea stărilor finale F1) care acceptă limbajul L1 şi dorim să obţinem un automat finit care să accepte limbajul $L=L_1^*$:

- introducem o nouă stare q_0 care va fi noua stare iniţială
- adăugăm o λ-tranziţie de la q_0 spre q_{01}
- adăugăm λ-tranziţii de la fiecare stare finală din F1 spre q_0
- noua mulţime de stări finale va fi F = $\{q_0\}$ *(sau putem lăsa* $F=\{q_0\}\bigcup F_1$ *)*

EX_4: 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 $L_4=(a^*+bc^*)\cdot(ac)^*=(L_1+L_2)\cdot(L_3)^*$.

Transformarea automat_finit → expresie_regulată

Se numeşte ***AFE (automat finit extins)***, $M = (Q, \Sigma, et, q_0, F)$, unde, la fel ca la celelalte automate finite, Q este mulţimea stărilor, Σ este alfabetul, q_0 este starea iniţială, F este mulţimea stărilor finale. Aici avem *funcţia de etichetare* $et : Q \times Q \to REX(\Sigma)$.

Notăm $et(p,q)$ prin e_{pq}.

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

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

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

Pas 2: Dacă există săgeţi care ajung către starea iniţială, atunci se adaugă la automat o nouă stare care va fi iniţială şi va avea o săgeată cu expresia λ către fosta stare iniţială.

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

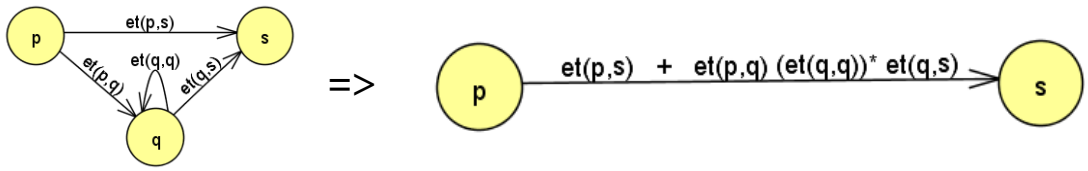
Pas 4: În orice ordine, se elimină pe rând, una câte una, toate stările în afară de cea iniţială şi cea finală. Presupunem că vrem să eliminăm starea q_k şi că există etichetele $et(q_i, q_k)$, $et(q_k, q_j)$ şi eventual bucla $et(q_k, q_k)$. Atunci obţinem noua etichetă între stările q_i şi q_j reunind [(fosta etichetă directă de la q_i la q_j), sau nimic (∅) dacă nu există drum direct] cu [(eticheta de la q_i la q_k) concatenată cu (stelarea etichetei buclei de la q_k la q_k, sau λ dacă bucla nu există) concatenată cu (eticheta de la q_k la q_j)]. *(Vezi desenul de mai jos.)*

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

Observaţii:

(1) La pas4, pentru starea q_k pe care dorim să o eliminăm (împreună cu toate săgeţile lipite de ea), trebuie să găsim orice "predecesor" $q_i \neq q_k$ (adică există o săgeată de la q_i la q_k) şi orice "succesor" $q_j \neq q_k$ (adică există săgeată de la q_k la q_j). Deci făcând abstracţie de eventuala buclă a lui q_k, căutăm şi *grupăm orice săgeată care intră* spre q_k *cu orice săgeată care iese* din q_k şi astfel obţinem expresia de pe săgeata de la q_i la q_j cu formula explicată mai sus. Atenţie, dacă $q_i = q_j$, înseamnă că veţi obţine o buclă.

→ Vrem să eliminăm q şi avem un p ("predecesor") şi un s („succesor").



(2) Dacă una din expresii conţine reuniune ("+"), *o includeţi între paranteze*, pentru a se executa întâi acea reuniune şi abia apoi concatenarea cu expresiile de pe alte săgeţi. Fiecare expresie obţinută între q_i şi q_j încercăm să o *simplificăm* cât mai mult folosind formulele de mai jos.

(3) În funcţie de *ordinea* în care alegeţi să eliminaţi stările la pasul 4, veţi obţine o anumită expresie, dar toate sunt echivalente între ele. ***Sfat:*** În general, eliminaţi starea care are momentan cele mai puţine săgeţi pentru a calcula cât mai puţine drumuri.

Câteva formule utile

(A) $e \cdot \emptyset = \emptyset$; $\emptyset \cdot e = \emptyset$ ($\emptyset$ este pentru concatenare cum este 0 pentru înmulţire)

(B) $e \cdot \lambda = e$; $\lambda \cdot e = e$ (λ este pentru concatenare cum este 1 pentru înmulţire)

(C) $e^* \cdot e = e^+$; $e \cdot e^* = e^+$ *(asta nu va fi folosită în REX, pt că nu respectă definiţia lor)*

(D) $\{e_1, e_2\}^* = (e_1 + e_2)^* = (e_1^* \cdot e_2^*)^*$ *(formulă valabilă pentru oricâte expresii, nu doar 2)*

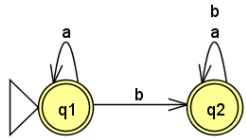
(E) $e_1 \cdot (e_2 + e_3) = (e_1 \cdot e_2) + (e_1 \cdot e_3)$; $(e_1 + e_2) \cdot e_3 = (e_1 \cdot e_3) + (e_2 \cdot e_3)$

Atenţie să nu confundaţi semnul "+" dintre expresii (folosit pentru *reuniunea* lor) cu semnul "+" pus la putere (folosit pentru *concatenare repetată*, cel puţin puterea 1).

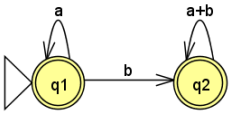
Obs: Algoritmul de mai sus descoperă și *reunește expresiile regulate corespunzătoare tuturor drumurilor de la starea inițială la o stare finală*. Puteți verifica asta pe exemplele următoare, comparând automatul finit dat cu expresia regulată obținută la finalul algoritmului.

Exemplul 1: Să se transforme următorul automat finit într-o expresie regulată echivalentă.
… TO DO …
… se vor obtine expresii diferite (dar echivalente) in functie de ordinea eliminarii starilor

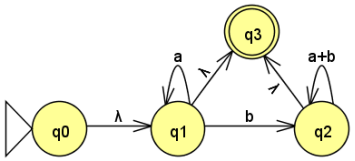
Exemplul 2: Să se transforme următorul automat finit într-o expresie regulată echivalentă.



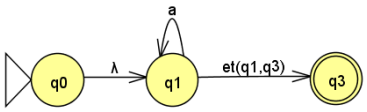
$Pas\ 1 \Rightarrow AFE:$



$Pas\ 2+3 \Rightarrow$

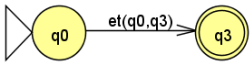


$Pas\ 4 \Rightarrow Elimin\ q_2$:



$et(q_1, q_3) = \lambda + b \cdot (a+b)^* \cdot \lambda \overset{(B)}{=} \lambda + b \cdot (a+b)^*$

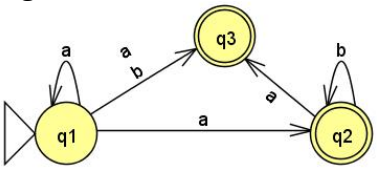
$Pas\ 4 \Rightarrow Elimin\ q_1$:



$et(q_0, q_3) = \emptyset + \lambda \cdot a^* \cdot (\lambda + b \cdot (a+b)^*) \overset{(B,E)}{=}$

$= a^* + a^* \cdot b \cdot (a+b)^*$ (***Obs:*** drum q1→q1 + drum q1→q2)

EX_5: Transformați următorul automat finit într-o expresie regulată echivalentă, folosind algoritmul de mai sus.



Concluzii pentru capitolul "Limbaje regulate"

… TO DO ….