C13

miercuri, 24 ianuarie 2024 23:24

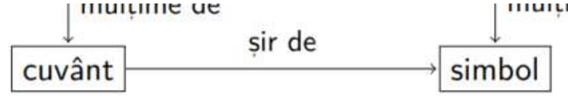
Sisteme cu comportament simplu: *automate*
un model pentru calcule cu *memorie finită*

Limbaje (mulțimi de șiruri) de o formă simplă:
concatenare, alternativă, repetiție

Alfabetul e o mulțime de *simboluri* (caractere)
{*a, b, c*} sau {0, 1} sau {0, 1, ..., 9}, ...

Cu simbolurile din alfabet putem forma *șiruri* (*cuvinte*, secvențe):
aba, 010010, 437, ...

Un *limbaj* e o *mulțime de cuvinte* (șiruri)
ca orice mulțime, definită explicit: {*a, ab, ac, abc*}
sau după o regulă: șiruri de *a*, *b*, încep cu *a*, mai mulți *a* decât *b*



schemă: http://web.stanford.edu/class/cs103/lectures/14/Small14.pdf

Fie un *alfabet* Σ: o mulțime de *simboluri* (ex. caractere)

Un *cuvânt* finit peste alfabetul Σ e un *șir de simboluri* din Σ

$a_1a_2 \ldots a_n \quad a_i \in \Sigma$ oricâte în orice ordine

Notăm cu Σ* mulțimea *tuturor* cuvintelor *finite* peste alfabetul Σ

$$\Sigma^* = \{a_1a_2 \ldots a_n \mid a_i \in \Sigma\}$$

* steaua Kleene: *repetiție* (*zero sau mai multe* apariții)
conține *șirul vid*: repetiție de zero ori

Important: Σ* are cuvinte de lungime *nelimitată*, dar nu *infinite*

Un *limbaj formal* L e o mulțime de cuvinte L ⊆ Σ*, definită după anumite *reguli*: automate, expresii regulate, gramatici, etc.

limbajul șirurilor de paranteze echibrate; al șirurilor palindrom; al șirurilor de 0 și 1 care nu au trei 0 consecutivi; etc.

Automat finit determinist (DFA)

Un automat e dat de: *simbolurile* de intrare
stări
tranziții (trecerile dintr-o stare în alta)
starea *inițială*
stările *acceptoare* (unde vrem să ajungem)

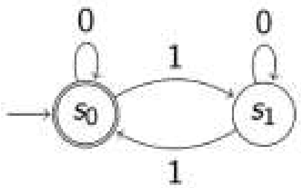
tranziții (trecerile dintr-o stare în alta)
starea *inițială*
stările *acceptoare* (unde vrem să ajungem)

Formal, un automat finit e un tuplu cu 5 elemente $(\Sigma, S, s_0, \delta, F)$

- Σ e un *alfabet* finit nevid de *simboluri* de intrare $\{a, 0, 1, \ldots\}$
- S e o mulțime finită nevidă de *stări*
- $s_0 \in S$ e *starea inițială* (*una*, în definiția uzuală)
- $\delta : S \times \Sigma \to S$ e *funcția de tranziție* $\xrightarrow{a}$
  determinist: la orice stare *și* intrare, o *unică* stare următoare
- $F \subseteq S$ e mulțimea stărilor *acceptoare*
  în final, vrem să fim aici dacă șirul e bun (din limbaj)

Exemplu de automat determinist (1)

automat de *paritate*: acceptă șiruri de 0 și 1 cu număr par de 1



sau ca tabelă de tranziții

	0	1
s_0	s_0	s_1
s_1	s_1	s_0

s_0 e stare inițială și acceptoare în același timp

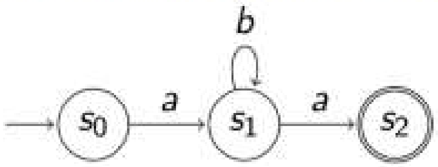
Stările acceptoare *pot* avea tranziții:

aici, din s_0 se iese la citirea lui 1
contează starea în care ajunge *când se termină* șirul

aici, din s_0 se iese la citirea lui 1
contează starea în care ajunge *când se termină* șirul

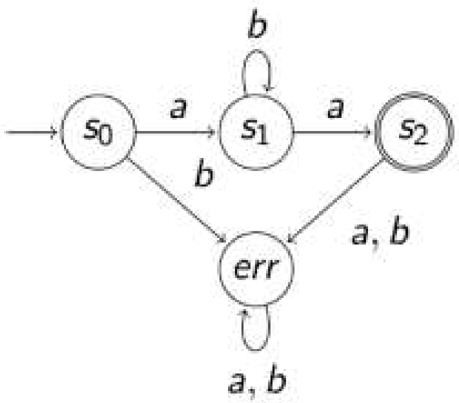
Exemplu de automat determinist (2)

automat care acceptă cuvinte cu oricâți de *b* (incl. 0) între doi *a*



ca δ să fie definită peste tot e necesară încă o stare *err* în practică se poate omite

dacă dintr-o stare nu e tranziție automatul s-a *blocat*, șirul nu e bun



Intersecția, reuniunea și complementul limbajelor

Un limbaj recunoscut de un automat se numește *limbaj regulat*
vom vedea că se poate exprima prin *expresii regulate*

Automatul pentru *intersectia* a două limbaje $L_1 \cap L_2$

vom vedea că se poate exprima prin *expresii regulate*

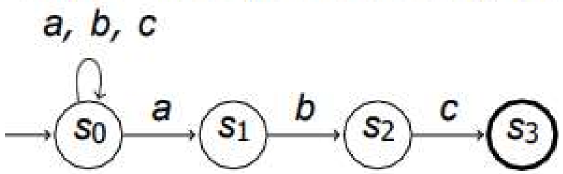
Automatul pentru *intersecția* a două limbaje $L_1 \cap L_2$
(numit uzual automatul produs)
are stări din *produsul cartezian* $S_1 \times S_2$ al stărilor
tranziționează *simultan* în ambele automate acceptă
dacă *ambele* acceptă

Automatul pentru *reuniunea* a două limbaje $L_1 \cup L_2$
tranziționează *simultan* în ambele automate (ca mai sus)
acceptă dacă *cel puțin unul* acceptă

Automatul pentru *complement* $\overline{L}$
acceptă dacă automatul original nu acceptă (complementăm $\overline{F}$)
mai întâi scriem automatul riguros complet (nu cu tranziții lipsă)

Automate finite nedeterministe (NFA): Exemplu (1)

Exemplu: toate șirurile de *a, b, c* care se *termină* în *abc*



Din s_0, primind simbolul *a*, automatul poate
– rămâne în s_0
– trece în s_1
⇒ automatul poate urma *una din mai multe* căi

Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare.

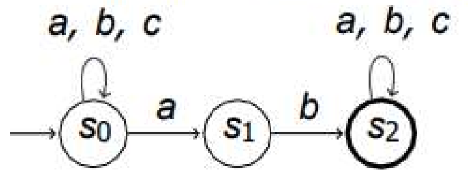
⇒ automatul poate urma *una din mai multe* căi

Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare.

Dacă pentru un șir ...*abc* alegem să trecem în s_1 la simbolul *a* (antepenultimul simbol), șirul va fi acceptat.

Automate finite nedeterministe (NFA): Exemplu (2)

Toate șirurile de *a, b, c* care *conțin* un subșir *ab*



Odată găsit *ab*, șirul e bun, oricum ar continua
tranzițiile din starea acceptoare trec tot în stare acceptoare

Avantajele NFA:

uneori se scrie mai ușor decât un automat determinist
(trebui să descriem calea acceptoare, nu toate celelalte)

e util când *specificăm* un sistem: putem lăsa deschise mai multe posibilități, ne permite o alegere la implementare

Comparație: automate deterministe și nedeterministe

Funcția de tranziție e acum $\delta : S \times \Sigma \to P(S)$

o *mulțime de stări* în care poate trece automatul (0 sau mai multe)

δ e echivalentă cu o *relație*: orice mulțime $\subseteq S \times \Sigma \times S$ de tranziții ($stare \xrightarrow{simbol} stare$) definește un automat nedeterminist

Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare. Acceptă șirul $a_1a_2\ldots a_n$ dacă *există* șirul de stări $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ cu $s_k \in \delta(s_{k-1}, a_k)$ $(k \geq 1)$ și $s_n \in F$ (acceptoare)

Un NFA poate să aibă tranziții lipsă: $\delta(s, a) = \emptyset$ (mulțimea vidă)

Nu afectează noțiunea de șir acceptat: ne interesează doar dacă *există* o cale acceptoare, chiar dacă se blochează pe altele.

Orice automat nedeterminist are un automat determinist echivalent (acceptă aceleași șiruri). Prezentăm cum facem conversia.

Conversie automat nedeterminist → automat determinist

Fie un NFA $M = (\Sigma, S, s_0, \delta, F)$. Construim un DFA echivalent.

Reținem la orice pas mulțimea de stări în care s-ar putea afla M
- o stare în noul automat e o *mulțime de stări* din automatul inițial
- noua mulțime de stări va fi $S' = P(S)$

Reținem la orice pas mulțimea de stări în care s-ar putea afla M

o stare în noul automat e o *mulțime de stări* din automatul inițial

$\Rightarrow$ noua mulțime de stări va fi $S` = P(S)$

Poate fi exponențial în dimensiunea inițială, $|P(S)| = 2^{|S|}$

Obținem automatul *determinist* $M` = (\Sigma, S`, s_0, \delta`, F`)$ cu

$S` = P(S)$

$\delta`(q, a) = \bigcup_{s \in q} \delta(s, a)$ pentru fiecare stare $s \in q$ cu $q \in P(S)$,

reunim mulțimile stărilor în care se ajunge pe simbolul a

$F` = \{s \in S` \mid s \cap F \neq \emptyset\}$

mulțimea stărilor care au o stare acceptoare din F

acceptă dacă *există o cale* care duce în stare acceptoare

Un limbaj = o mulțime de cuvinte peste un alfabet

Adesea ne interesează cuvinte cu structură simplă, "regulată":

un *întreg*: o secvență de cifre, eventual cu semn

un *real*: parte întreagă + parte zecimală (una din ele opțională), exponent opțional

un *identificator*: litere, cifre, _ începând cu literă sau _

nume de fișiere: 01-*titlu*.mp3, 02-*alttitlu*.mp3, ...

Unele limbaje pot fi recunoscute eficient de *automate finite*

dar scrierea automatului ia efort

$\Rightarrow$ se pot scrie mai simplu ca *expresii regulate*

⇒ se pot scrie mai simplu ca *expresii regulate*

)perații pe limbaje

Reuniunea, intersecția și complementul limbajelor regulate sunt limbaje regulate

Concatenarea limbajelor

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}$$

orice cuvânt din L_1 urmat de orice cuvânt din L_2

Închiderea Kleene (repetiția)

$$L* = \{w \mid \exists n \in \mathrm{N}. w = w_1 w_2 \ldots w_n,\ w_i \in L\}$$

concatenarea *oricăror* șiruri din L, nu neapărat același șir

luând $n = 0$, rezultă *șirul vid* (niciun simbol, lungime 0) notăm șirul vid cu epsilon: $\varepsilon \in L*$ pentru orice $L \neq \emptyset$

Reguli de scriere și exemple

Reguli de scriere și exemple

Omitem paranteze când sunt clare din relațiile de precedență
cel mai prioritar: $*$, apoi concatenare și apoi reuniune $+$
punctul pentru concatenare se omite

În practică se mai folosesc abrevierile
$e?$ pentru $e + \varepsilon$ (e, opțional)
e^+ pentru $e^* \setminus \varepsilon$ (e, cel puțin o dată)

$(0 + 1)^*$ mulțimea tuturor șirurilor din 0 sau 1

$(0 + 1)^*0$ ca mai sus, încheiat cu 0 (numere pare în binar)

$1(0 + 1)^* + 0$ numere binare, fără zerouri inițiale inutile

Conversia în automat: Reuniune/alternativă

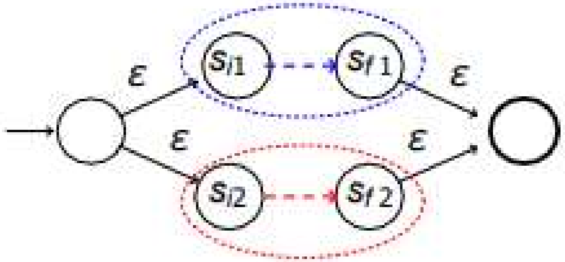
Combinăm automatele pentru cele două expresii regulate
(în oval pot fi alte stări și tranziții)

e_1 

e_2 

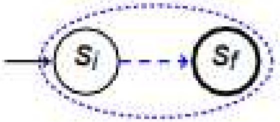
Starea inițială și finală au tranziții ε spre/din automatele originale, fără a consuma simboluri $\Rightarrow$ pot parcurge oricare din automate

$e_1 + e_2$



Conversia în automat: Închiderea Kleene

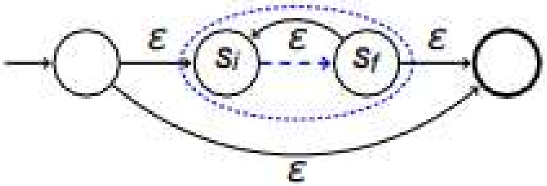
e



Adăugăm tranziții ε (șirul vid) care nu consumă niciun un simbol:

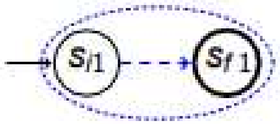
- închid ciclul stare finală $\rightarrow^{\varepsilon}$ inițială în automatul pentru e
- trec direct din starea inițială în cea finală (șirul vid, 0 iterații)
- leagă noua stare inițială și finală de cele ale expresiei interioare

e^*

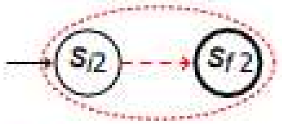


Conversia în automat: Concatenare

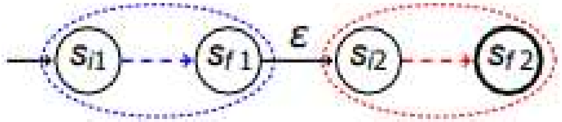
e_1



e_2



e_1e_2



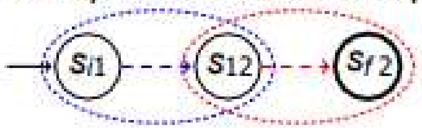
Construcțiile de până acum asigură:

o *unică* stare inițială, în care nu se revine

o *unică* stare acceptoare, din care nu ies tranziții

Atunci putem contopi la concatenare capetele lui e_1 și e_2.

e_1e_2

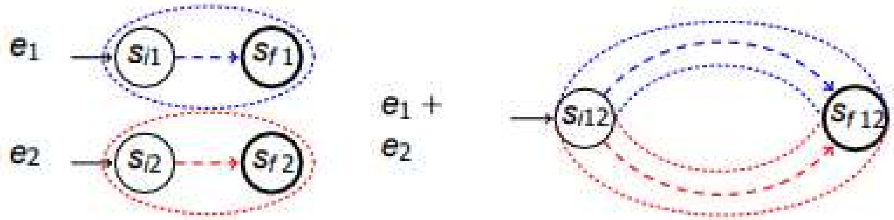


Construcții simplificate

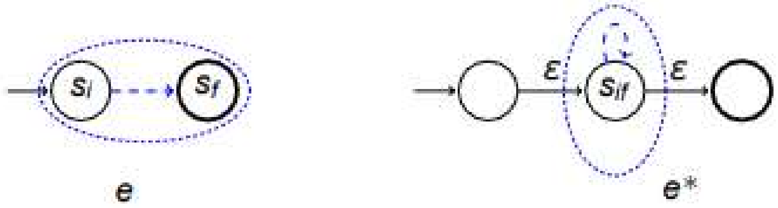
Neavând tranziții *spre* starea inițială și *din* cea acceptoare, simplificăm:

Construcții simplificate

Neavând tranziții *spre* starea inițială și *din* cea acceptoare, simplificăm:
La *alternativă*, comasăm cele două stări inițiale și finale

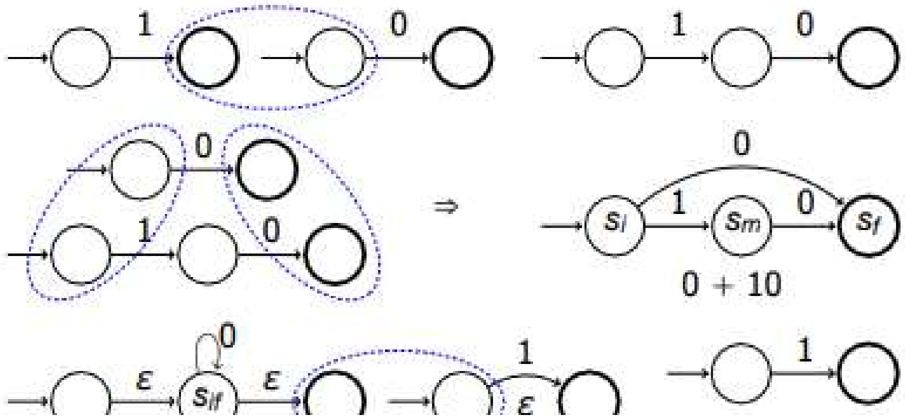


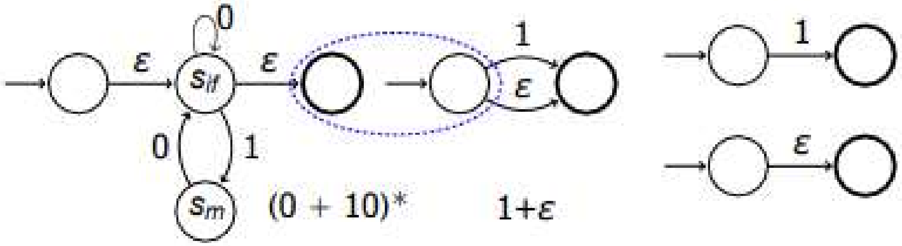
La *închiderea Kleene*, comasăm starea inițială cu cea finală;
cei doi ε consecutivi ne dau cazul cu zero repetiții



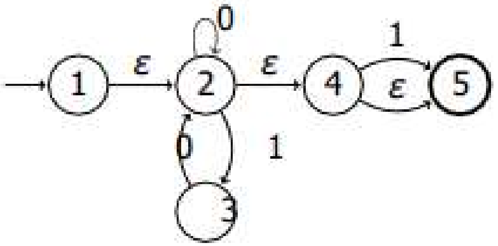
Exemplu: Conversie din expresie regulată în automat

Fie expresia regulată $(0+10)^*(1+\varepsilon)$. Construim pas cu pas:





Exemplu: Conversie din expresie regulată în automat (2)

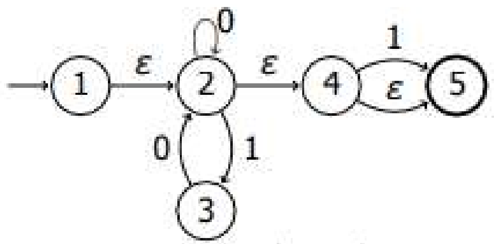


$(0+10)^*(1+\varepsilon)$

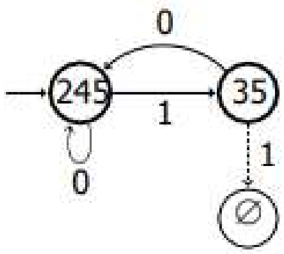
O *tranziție ε* se face spontan, fără a consuma un simbol de intrare
⇒ din starea s se ajunge în orice s' legată prin oricâte ε-tranziții
(*închiderea tranzitivă* a relației definite de ε)

Ajuns în 1, s-ar putea afla și în 2, 4 sau 5
⇒ starea inițială e de fapt mulțimea {1, 2, 4, 5}
Ajuns în 2, s-ar putea afla și în 4 sau 5 ⇒ mulțimea {2, 4, 5}, etc.

Conversie din NFA cu tranziții ε în DFA



	0	1
1245	245	35
245	245	35
35	245	∅



Tranzițiile pe 0 ne duc direct în 2, apoi prin ε în 4 și 5.

Liniile 1 și 2 au destinații identice ⇒ stările sunt echivalente.

⇒ automat cu doar două stări (ignorând starea de eroare ∅) Ambele conțin pe 5 ⇒ sunt acceptoare.

Conversia din automat în expresie regulată

Vrem să rămânem doar cu *două noduri* (inițial și acceptor), cu tranzițiile etichetate de *șiruri* (părți din expresia regulată).

(*extindem* notația de automat *doar* în cadrul acestei construcții; riguros automatele consumă doar *un* simbol pe tranziție)

Dacă sunt > 1 noduri acceptoare, *adaugăm un nod acceptor unic* și ducem din fiecare stare acceptoare tranziții ε spre el

Eliminăm pe rând *fiecare nod* în afară de cel inițial și acceptor:

pentru orice nod intermediar i de eliminat

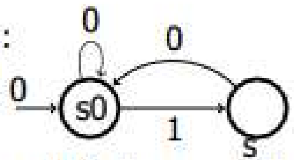
pentru orice pereche de noduri (s,d) adaugă la muchia $s \to d$ limbajul $L_{si} L^*_{ii} L_{id}$

(tranziționăm din $s \to i$, repetăm indefinit $i \to i$, apoi $i \to d$)

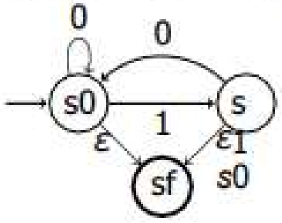
pentru orice pereche de noduri (ș*d*) adaugă la muchia $s \to d$ limbajul $L_{si} L^*_{ii} L_{id}$
(tranziționăm din $s \to i$, repetăm indefinit $i \to i$, apoi $i \to d$)

Exemplu: Conversie din automat în expresie regulată

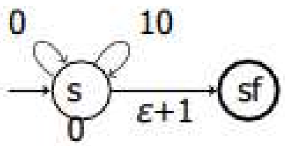
șiruri de 0 și 1 care nu au doi 1 consecutivi:
pe 1, trece în starea s_1 cu tranziție doar pe 0



Ambele stări sunt acceptoare ⇒ adăugăm o unică stare acceptoare



Eliminăm s1:
$s0 \xrightarrow{10}$
$s0 \xrightarrow{1\varepsilon} sf$



Obținem astfel limbajul $(0+10)^*(1+\varepsilon)$

Minimizarea automatelor

Două stări s_1 și s_2 pot fi *deosebite* dacă există un cuvânt *w* care dintr-una din stări conduce la o stare acceptoare, și din cealaltă, nu

$$\delta^*(s_1, w) \in F \neq \delta^*(s_2, w) \notin F$$

dintr-una din stări conduce la o stare acceptoare, și din cealaltă, nu

$$\delta^*(s_1, w) \in F \quad \neq \delta^*(s_2, w) \notin F$$

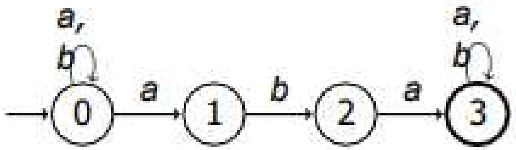
Două stări care nu pot fi deosebite sunt *echivalente*
⇒ pot fi înlocuite cu o singură stare

Un DFA e *minimal* dacă nu există un automat cu mai puține stări care acceptă același limbaj.

Diverși *algoritmi de minimizare* (ex. Hopcroft-Ullman, Moore)
inițial, partiție cu 2 blocuri: F, $S \setminus F$ (stări acceptoare sau nu) (o
(o împărțire în *potențiale* clase de echivalență)
desparte un bloc din partiție dacă pe un simbol, stările nu trec toate în același bloc din partiție (pot fi deosebite)

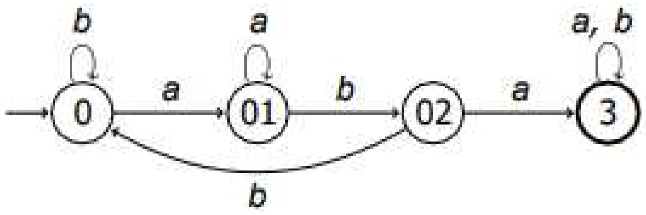
Conversie NFA-DFA și minimizare (exemplu)

Cuvinte din a, b cu subșir aba: "ghicim" când începe subșirul dorit



	a	b
0	01	0
01	01	02
02	013	0
013	013	023
023	013	03
03	013	03

Stările care conțin 3 (stare acceptoare) sunt acceptoare.
Aici, ele trec tot timpul în stări acceptoare, deci sunt *echivalente* (caz simplu), și le putem comasa într-o singură stare (numită 3).



Recapitulare

Un *automat* finit determinist definește un *limbaj acceptat*.
Un astfel de limbaj se numește *limbaj regulat*.
El poate fi exprimat și printr-o *expresie regulată*.

Intersecția, reuniunea, și complementul limbajelor regulate produc *limbaje regulate*, la fel concatenarea și închiderea Kleene.
deci pot fi *recunoscute de automate finite*

Automatele finite *nedeterministe* se pot transforma în *deterministe*
deci recunosc tot limbaje regulate
dar numărul de stări poate crește exponențial

Automatele finite pot fi *minimizate*, comasând *stările echivalente*.

Automatele deterministe și nedeterministe și expresiile regulate au *aceeași putere expresivă* (descriu limbaje regulate).