

Logică și structuri discrete  
Automate finite și expresii regulate

Marius Minea  
marius@cs.upt.ro

<http://www.cs.upt.ro/~marius/curs/lsd/>

27 noiembrie 2017

## În cursul de azi

Sisteme cu comportament simplu: *automate*

un model pentru calcule cu *memorie finită*

*Limbaje* (multimi de siruri) de o formă simplă:

concatenare, alternativă, repetiție

## Un exemplu: automatul de cafea

*acțiuni* (utilizator): introdu *fisă*, apasă *buton răspuns* (automat): *toarnă cafea*

După o acțiune *se întâmplă* ceva ?

<i>buton</i>	nu
<i>fisă</i>	nu imediat
<i>fisă buton</i>	da

*fisă* a avut un efect *intern*: automatul a trecut în altă *stare* (se *comportă altfel* la acțiunea *buton*)

<i>fisă fisă buton buton</i>	dă două cafele ?
dacă da, câte fise poate ține minte ?	

# Automate în practică

Multe sisteme se pot modela ca automate:

numărătoare, afișaje, control simplu pornit/oprit

*protocole* de comunicație: trimite, primește, așteaptă, ...

Automatele sunt un *model* pentru ce se poate *calcula*

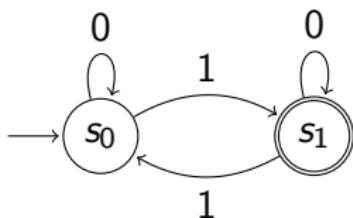
cu *memorie finită* (automate cu număr finit de stări)

Alte probleme legate de automate:

*Testarea* cu diverse *secvențe de intrări*: corespunde specificației?

Putem *învăța* (deduce) forma automatului din comportament?

# Un automat foarte simplu



începe în *starea*  $s_0$   
când primește 1, schimbă starea  
când primește 0, stă pe loc

După un sir cu număr *par de 1*, automatul va fi în  $s_0$

După un sir cu număr *impar de 1*, automatul va fi în  $s_1$

⇒ automatul poate *deosebi* cele două feluri de siruri

Dacă vrem un număr impar de 1, marcăm  $s_1$  ca *stare acceptoare sir acceptat*: doar dacă la final automatul e în stare acceptoare

⇒ automatul definește o *multime de siruri*, adică un *limbaj*

# Ce e un limbaj

*Alfabetul* e o mulțime de *simboluri* (caractere)

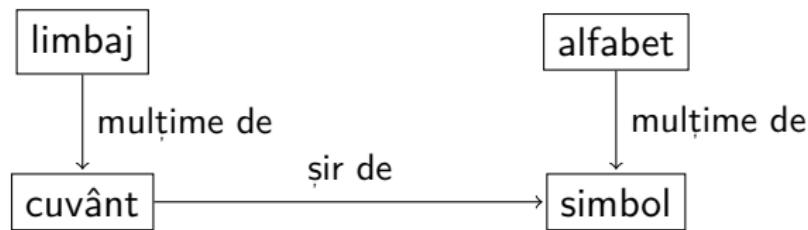
$\{a, b, c\}$  sau  $\{0, 1\}$  sau  $\{0, 1, \dots, 9\}$ , ...

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  $b$



## Ce e un limbaj (formal)

Fie un *alfabet*  $\Sigma$ : o mulțime de *simboluri* (ex. caractere)

Un *cuvânt* finit peste alfabetul  $\Sigma$  e un *șir de simboluri* din  $\Sigma$

$a_1 a_2 \dots a_n \quad a_i \in \Sigma \quad$  oricâte în orice ordine

Notăm cu  $\Sigma^*$  mulțimea *tuturor* cuvintelor *finite* peste alfabetul  $\Sigma$

$$\Sigma^* = \{a_1 a_2 \dots a_n \mid a_i \in \Sigma\}$$

\* steaua Kleene: *repetiție (zero sau mai multe apariții)*

conține *șirul vid*: repetiție de zero ori (vezi ciclul while)

Important:  $\Sigma^*$  are cuvinte de lungime *nelimitată*, dar nu *infinite*

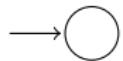
Un *limbaj formal*  $\mathcal{L}$  e o mulțime de cuvinte  $\mathcal{L} \subseteq \Sigma^*$ , definită după anumite *reguli*: automate, expresii regulate, gramatici, etc.

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)

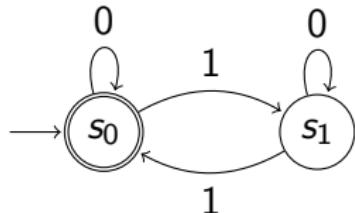
Un automat e dat de: *simbolurile* de intrare (de ex. un caracter, sau o acțiune externă, apăsarea unui buton)  
*stări*  
trecerile dintr-o stare în alta (*tranzitii*)  
starea *initială*  
unde vrem să ajungem (stările *acceptoare*)

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

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

## Exemplu de automat determinist (1)

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



sau ca tabelă de tranzitii

	0	1
$s_0$	$s_0$	$s_1$
$s_1$	$s_1$	$s_0$

$s_0$  e stare inițială  $\rightarrow$  și acceptoare  $\circlearrowleft$  în același timp

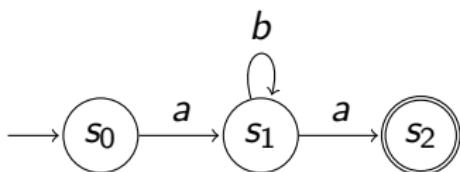
Stările acceptoare pot avea tranzitii:

aici, din  $s_0$  se ieșe la citirea lui 1

contează starea în care ajunge când se termină sirul

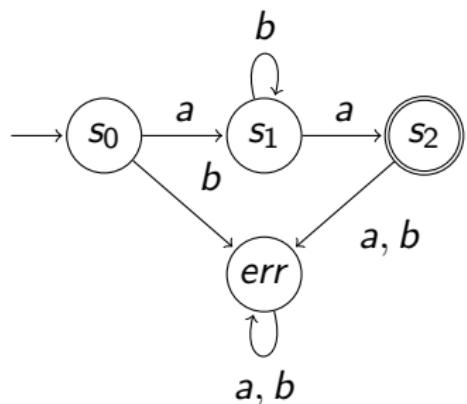
## Exemplu de automat determinist (2)

automat care acceptă cuvinte cu oricărăi de  $b$  (incl. 0) între doi  $a$



ca  $\delta$  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 *blockat*, sirul nu e bun



## Limbajul acceptat de un automat

Notăm  $\varepsilon \in \Sigma^*$  cuvântul *vid* (fără niciun simbol).

Definim o funcție de tranziție  $\delta^* : S \times \Sigma^* \rightarrow S$  cu intrări *cuvinte*:  
în ce stare ajunge automatul pentru un cuvânt dat la intrare?

Pentru orice stare  $s \in S$ , definim inductiv:

$$\begin{aligned}\delta^*(s, \varepsilon) &= s && \text{cuvânt vid: nu face nimic} \\ \delta^*(s, a_1 a_2 \dots a_n) &= \delta^*(\delta(s, a_1), a_2 \dots a_n) && \text{pentru } n > 0\end{aligned}$$

Altfel spus,  $\delta^*(s_0, a_1 a_2 \dots a_n) = \delta^*(s_1, a_2 \dots a_n)$  cu  $s_1 = \delta(s_0, a_1)$   
obținem starea  $s_1$  după intrarea  $a_1$ , și aplicăm  $\delta^*$  pe sirul rămas

Automatul *acceptă* cuvântul  $w \in \Sigma^*$  dacă și numai dacă  $\delta^*(s_0, w) \in F$   
(cuvântul *duce* automatul într-o stare *accepțoare*)

# Cum reprezentăm un automat ?

Matrice  $S \times \Sigma$  cu elemente din  $S$   
(pentru fiecare stare și intrare, starea următoare)  
reprezintă *explicit* fiecare combinație

	a	b
$s_0$	$s_0$	$s_1$
$s_1$	$s_1$	$s_0$

Sau: un *dicționar* care dă pentru fiecare stare funcția de tranziție  
reprezentată tot ca un *dicționar* (intrare, stare)

Dacă dintr-o stare multe simboluri duc în aceeași stare următoare,  
asociem fiecărei stări:

- un dicționar (intrare, stare)
- o stare următoare *implicită* (pentru celelalte intrări)

## Automate cu ieșiri

numite și *traductoare* (engl. *transducer*)

scopul: generează răspunsuri/ieșiri; nu au multime acceptoare  $F$

în plus: un *alfabet de ieșire*  $\Omega$  și o *funcție de ieșire*  $g$

automate de tip *Moore*

ieșirea e funcție de *stare*:  $g : S \rightarrow \Omega$

automate de tip *Mealy*

ieșirea e funcție de *stare* și *intrare*  $g : S \times \Sigma \rightarrow \Omega$

folosite pentru a modela *circuite sevențiale*

discutate la disciplina *Logică digitală*

## 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 *intersecția* a două limbaje  $\mathcal{L}_1 \cap \mathcal{L}_2$   
(numit ușual automatul produs)

are stări din *produsul cartezian*  $S_1 \times S_2$  al stărilor  
tranzitionează *simultan* în ambele automate  
acceptă dacă *ambele* acceptă:

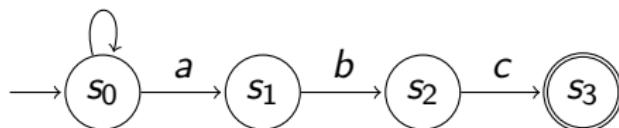
Automatul pentru *reuniunea* a două limbaje  $\mathcal{L}_1 \cup \mathcal{L}_2$   
tranzitionează *simultan* în ambele automate (ca mai sus)  
acceptă dacă *cel puțin unul* acceptă

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

## Automate finite nedeterministe (NFA): Exemplu (1)

Exemplu: toate sirurile de  $a, b, c$  care se termină în  $abc$

$a, b, c$



Din  $s_0$ , primind simbolul  $a$ , automatul poate

- rămâne în  $s_0$
- trece în  $s_1$

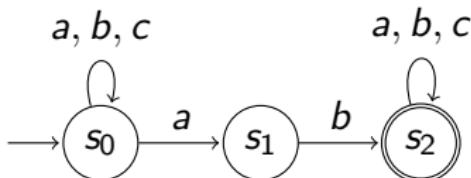
⇒ automatul poate urma *una din mai multe* căi

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

Dacă pentru un sir ... $\textcolor{red}{abc}$  alegem să trecem în  $s_1$  la simbolul  $\textcolor{red}{a}$  (antepenultimul simbol), sirul va fi acceptat.

## Automate finite nedeterministe (NFA): Exemplu (2)

Toate sirurile de  $a, b, c$  care *contin* un subșir  $ab$



Odată găsit  $ab$ , sirul e bun, oricum ar continua  
tranzitiiile din starea acceptoare trec tot în stare acceptoare

Avantaje:

- uneori se scrie mai ușor decât un automat determinist  
(trebuie 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  $\delta : S \times \Sigma \rightarrow \mathcal{P}(S)$

o *multime de stări* în care poate trece automatul (0 sau mai multe) e echivalentă cu o *relație*: orice mulțime  $\subseteq S \times \Sigma \times S$  de tranziții (*stare*  $\xrightarrow{\text{simbol}} \text{stare}$ ) definește un automat nedeterminist

Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare. Acceptă sirul  $a_1 a_2 \dots a_n$  dacă *există* sirul de stări  $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$  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 sir acceptat: ne interesează doar dacă *există* o cale acceptoare, chiar dacă se blochează pe altele.

*Orice automat nedeterminist are un automat determinist echivalent* (aceleși siruri). 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 *multime de stări* din automatul initial  
⇒ noua mulțime de stări va fi  $S' = \mathcal{P}(S)$   
poate fi exponential în dimensiunea inițială,  $|\mathcal{P}(S)| = 2^{|S|}$

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

$$S' = \mathcal{P}(S)$$

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

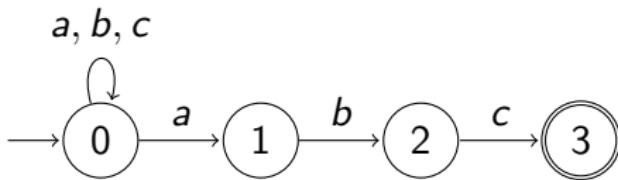
reunim multimile stărilor în care se ajunge pe simbolul  $a$

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

mulțimea stărilor care au o stare acceptoare din  $F$

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

## Conversie NFA-DFA (exemplu)

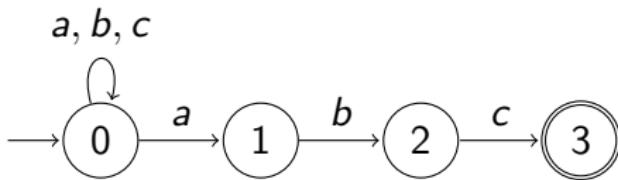


Când obținem o nouă multime  
**(roșu)** adăugăm o linie la tabel.

Scriem tabelul de tranziție  
cu multimea stărilor în care  
se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}

## Conversie NFA-DFA (exemplu)

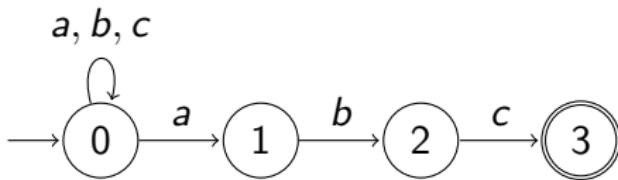


Când obținem o nouă multime (roșu) adăugăm o linie la tabel.

Scriem tabelul de tranziție cu multimea stărilor în care se trece pe fiecare simbol

	a	b	c
$\{0\}$	$\{0, 1\}$	$\{0\}$	$\{0\}$
$\{0, 1\}$	$\{0, 1\}$	$\{0, 2\}$	$\{0\}$

## Conversie NFA-DFA (exemplu)

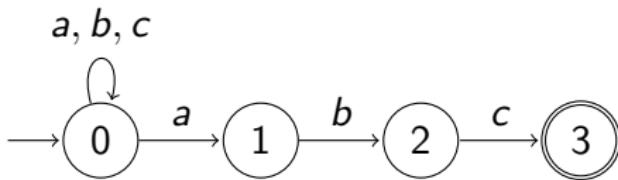


Când obținem o nouă multime (roșu) adăugăm o linie la tabel.

Scriem tabelul de tranziție cu multimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}

## Conversie NFA-DFA (exemplu)

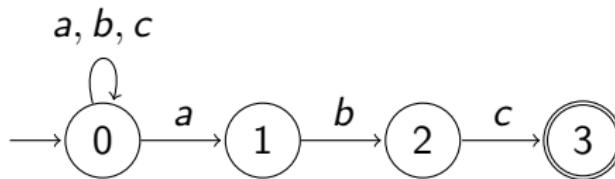


Când obținem o nouă multime (roșu) adăugăm o linie la tabel.

Scriem tabelul de tranziție cu multimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}
{0, 3}	{0, 1}	{0}	{0}

## Conversie NFA-DFA (exemplu)

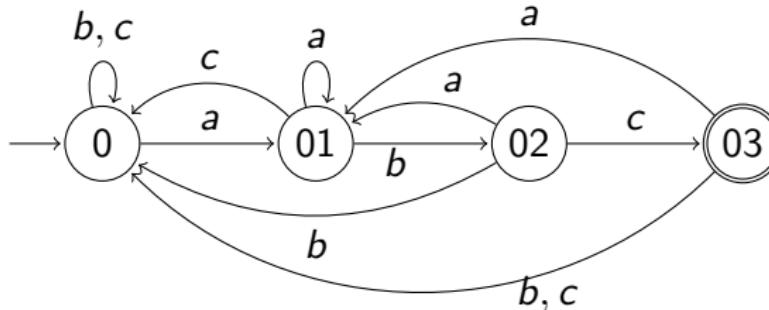


Când obținem o nouă multime (roșu) adăugăm o linie la tabel.

Scriem tabelul de tranziție cu multimea stărilor în care se trece pe fiecare simbol

	a	b	c
{0}	{0, 1}	{0}	{0}
{0, 1}	{0, 1}	{0, 2}	{0}
{0, 2}	{0, 1}	{0}	{0, 3}
{0, 3}	{0, 1}	{0}	{0}

Fiecare multime obținută devine o stare în DFA-ul rezultat



Stările acceptoare sunt cele care conțin o stare acceptoare din automatul inițial.

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

	a	d
{1}	{2, 4}	{5}

stare initială: 1

stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}

stare initială: 1

stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare initială: 1

stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{2, 4, 6, 8}	{1, 3, 5, 7, 9}	{2, 4, 6, 8}

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$\Sigma = \{a, d\}$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{2, 4, 6, 8}	{1, 3, 5, 7, 9}	{2, 4, 6, 8}
{1, 3, 7, 9}	{2, 4, 6, 8}	{5}

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{2, 4, 6, 8}	{1, 3, 5, 7, 9}	{2, 4, 6, 8}
{1, 3, 7, 9}	{2, 4, 6, 8}	{5}
{1, 3, 5, 7, 9}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}

## Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare initială: 1

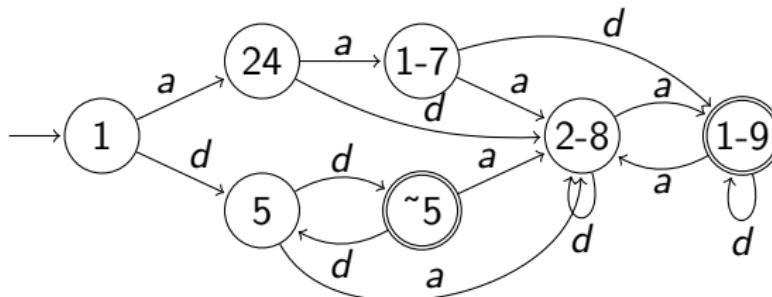
stare accept.: 9

$$\Sigma = \{a, d\}$$

a: mută adiacent

d: mută diagonal

	a	d
{1}	{2, 4}	{5}
{2, 4}	{1, 3, 5, 7}	{2, 4, 6, 8}
{5}	{2, 4, 6, 8}	{1, 3, 7, 9}
{1, 3, 5, 7}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}
{2, 4, 6, 8}	{1, 3, 5, 7, 9}	{2, 4, 6, 8}
{1, 3, 7, 9}	{2, 4, 6, 8}	{5}
{1, 3, 5, 7, 9}	{2, 4, 6, 8}	{1, 3, 5, 7, 9}



$$1-7 = \{1, 3, 5, 7\}$$

$$2-8 = \{2, 4, 6, 8\}$$

$$\sim 5 = \{1, 3, 7, 9\}$$

$$1-9 = \{1, 3, 5, 7, 9\}$$

# Putem exprima mai concis definiția unui limbaj?

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 optională), exponent optional

un *identifier*: litere, cifre, \_ începând cu literă sau \_

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

Unele limbi pot fi recunoscute eficient de *automate finite*

dar scrierea automatului ia efort

⇒ se pot scrie mai simplu ca *expresii regulate*

# Operații pe limbaje

*Reuniunea, intersecția și complementul* limbajelor regulate  
sunt limbaje regulate

Mai putem defini:

*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)

$$L^* = \{w \mid \exists n \in \mathbb{N}. w = w_1 w_2 \dots w_n, w_i \in L\}$$

concatenarea *oricărora* siruri din  $L$ , nu neapărat același sir

luând  $n = 0$ , rezultă *sirul vid* (niciun simbol, lungime 0)

notăm sirul vid cu epsilon:  $\varepsilon \in L^*$  pentru orice  $L \neq \emptyset$

## Expresii regulate: definiție formală

O expresie regulată descrie un limbaj (regulat).

O expresie regulată peste un alfabet  $\Sigma$  e fie:

3 cazuri de bază:

$\emptyset$	limbajul vid
$\varepsilon$	limbajul $\{\varepsilon\}$ (cu sirul vid)
$a$	limbajul $\{a\}$ cu $a \in \Sigma$ (un cuvânt de o literă)

3 cazuri recursive: date  $e_1, e_2$  expresii regulate, putem forma:

$e_1 + e_2$  reuniunea limbajelor

în practică, notată adesea  $e_1 | e_2$  (alternativă, "sau")

$e_1 \cdot e_2$  concatenarea limbajelor

$e_1^*$  închiderea Kleene a limbajului

## Reguli de scriere și exemple

Omitem paranteze cănd 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, optional)

$e^+$  pentru  $e^* \setminus \varepsilon$  (e, cel puțin o dată)

$(0 + 1)^*$  mulțimea tuturor sirurilor 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 initiale inutile

# Orice expresie regulată e recunoscută de un automat

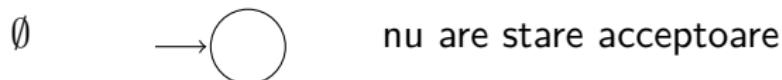
Construcție dată de Ken Thompson (creatorul UNIX, premiul Turing 1983)

Definim prin *inducție structurală*

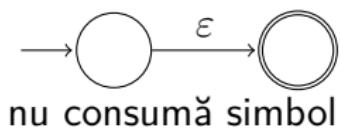
cum traducem cele 3 *cazuri de bază* de expresie regulată

cum *combinăm* automatele în cele 3 *cazuri recursive*

⇒ descompunând, convertim *orice expresie regulată* în automat



sau



în cele trei cazuri recursive, *combinăm* automatele limbajelor date

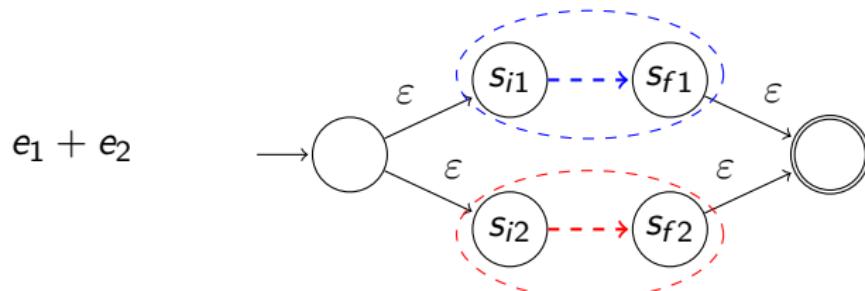
⇒ *automat finit nedeterminist* cu tranziții  $\varepsilon$  (nu consumă simbol)

## Conversia în automat: Reuniune/alternativă

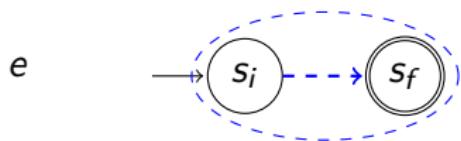
Combinăm automatele pentru cele două expresii regulate  
(în oval pot fi alte stări și tranzitii)



Starea initială și finală au tranzitii  $\varepsilon$  spre/din automatele originale,  
fără a consuma simboluri  $\Rightarrow$  pot parurge oricare din automate

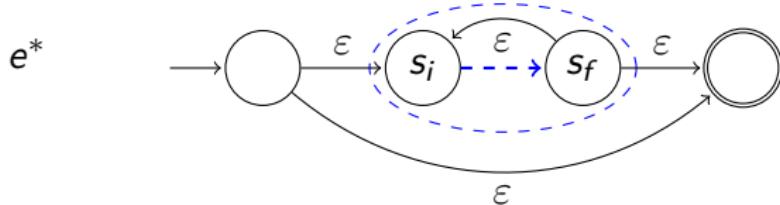


## Conversia în automat: Închiderea Kleene

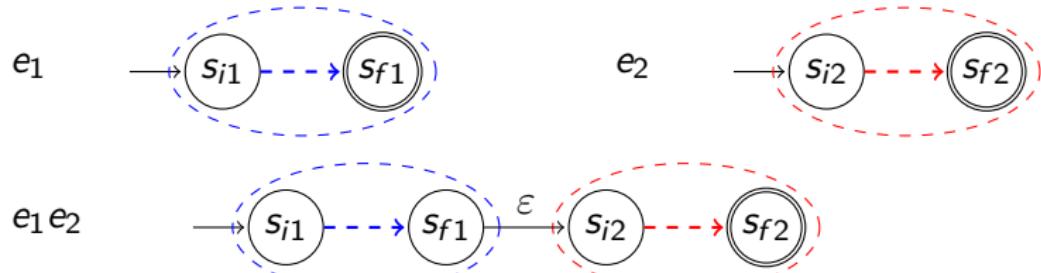


Adăugăm tranziții  $\varepsilon$  (șirul vid) care nu consumă niciun un simbol:

- închid ciclul stare finală  $\xrightarrow{\varepsilon}$  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



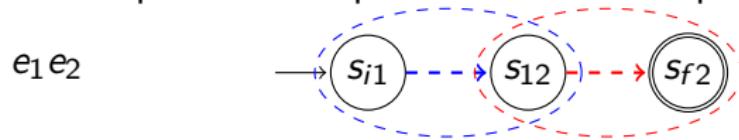
## Conversia în automat: Concatenare



Construcțiile de până acum asigură:

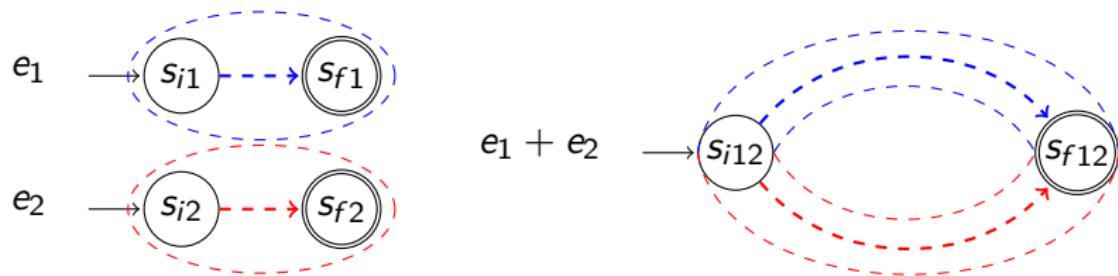
- o *unică* stare inițială, în care nu se revine
- o *unică* stare acceptoare, din care nu ies tranziții

Atunci putem contopi la concatenare capetele lui  $e_1$  și  $e_2$ .

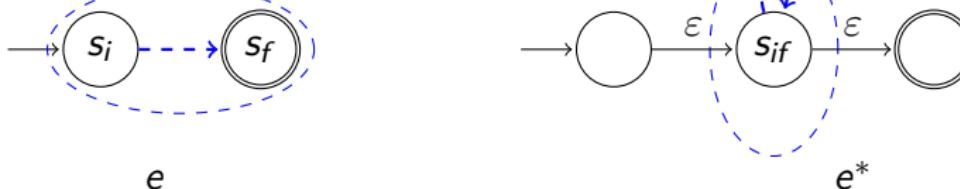


# Construcții simplificate

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

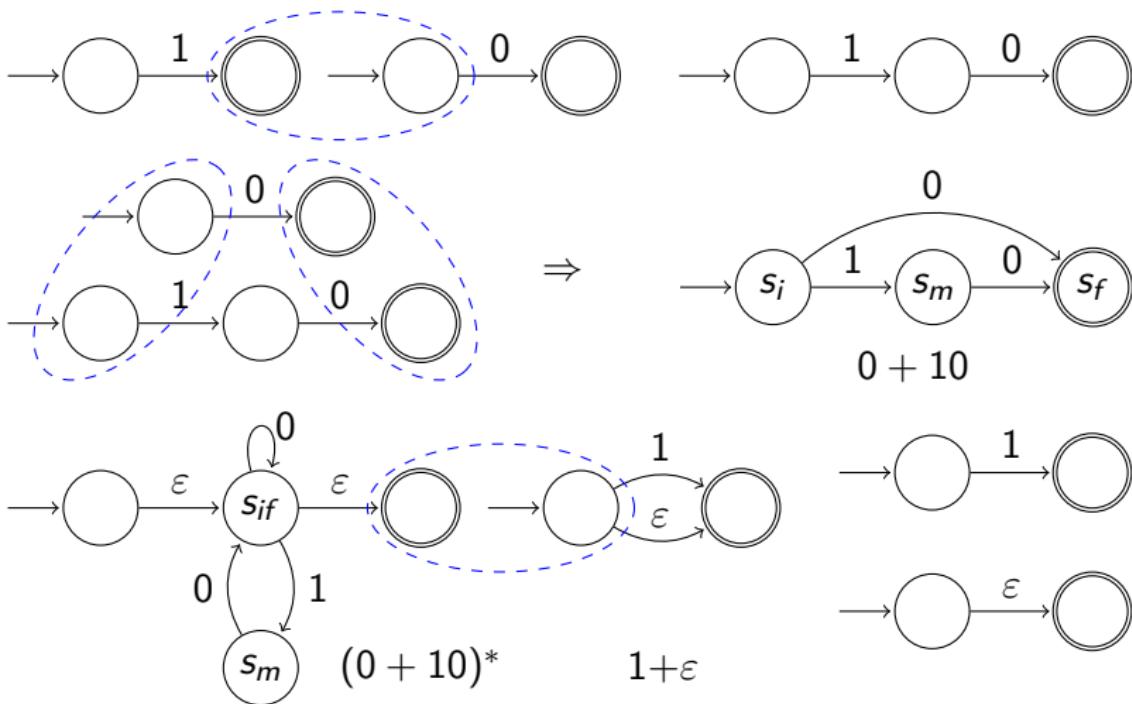


La închiderea Kleene, comasăm starea inițială cu cea finală;  
cei doi  $\varepsilon$  consecutivi ne dau cazul cu zero repetiții

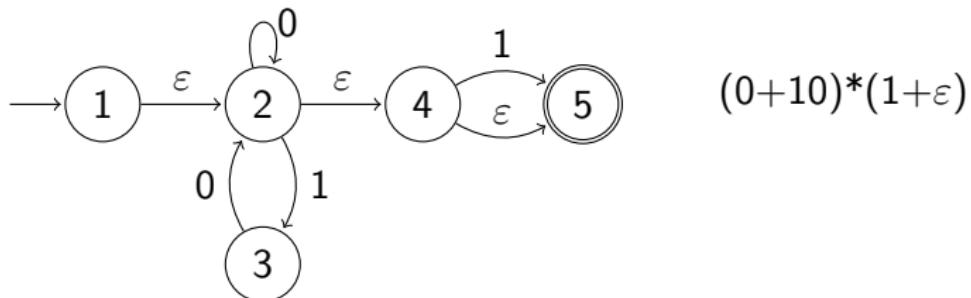


## Exemplu: Conversie din expresie regulată în automat

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



## Exemplu: Conversie din expresie regulată în automat (2)



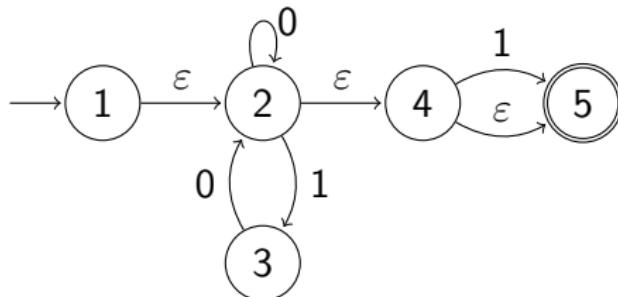
O *tranzitie  $\epsilon$*  se face spontan, fără a consuma un simbol de intrare  
⇒ din starea  $s$  se ajunge în orice  $s'$  legată prin oricâte  $\epsilon$ -tranzitii  
(*închiderea tranzitivă* a relației definite de  $\epsilon$ )

Ajuns în 1, s-ar putea afla și în 2, 4 sau 5

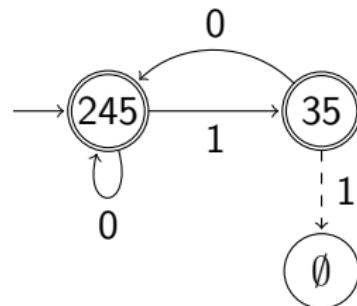
⇒ starea inițială e de fapt multimea  $\{1, 2, 4, 5\}$

Ajuns în 2, s-ar putea afla și în 4 sau 5 ⇒ multimea  $\{2, 4, 5\}$ , etc.

## Conversie din NFA cu tranziții $\varepsilon$ în DFA



	0	1
1245	245	35
245	245	35
35	245	$\emptyset$



Tranzițiile pe 0 ne duc direct în 2, apoi prin  $\varepsilon$  în 4 și 5.

Liniile 1 și 2 au destinații identice  $\Rightarrow$  stările sunt echivalente.

$\Rightarrow$  automat cu doar două stări (ignorând starea de eroare  $\emptyset$ )

Ambele conțin pe 5  $\Rightarrow$  sunt acceptoare.

## Conversia din automat în expresie regulată

Vrem să rămânem doar cu *două noduri* (initial și acceptor), cu tranzițiile etichetate de *șiruri* (părți din expresia regulată). (extindem notația de automat doar în cadrul acelei 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  $\varepsilon$  spre el

*Eliminăm* pe rând *fiecare nod* în afara de cel initial și acceptor:

pentru orice nod intermediar  $i$  de eliminat

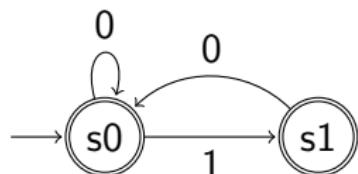
pentru orice pereche de noduri  $(s, d)$

adăugă la muchia  $s \rightarrow d$  limbajul  $L_{si}L_{ij}^*L_{id}$

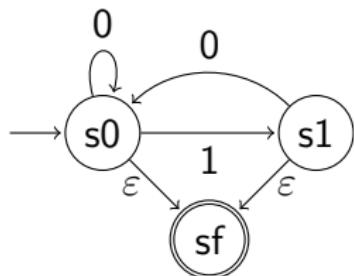
(tranziționăm din  $s \rightarrow i$ , repetăm indefinit  $i \rightarrow i$ , apoi  $i \rightarrow d$ )

## Exemplu: Conversie din automat în expresie regulată

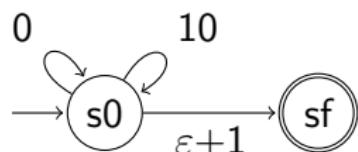
Şiruri de 0 și 1 care nu au doi 1 consecutivi  
pe 1, trece în starea  $s_1$  cu tranziție doar pe 0



Ambele stări sunt acceptoare  $\Rightarrow$  adăugăm o unică stare acceptoare



Eliminăm  $s_1$ :  
 $s_0 \xrightarrow{10} s_0$   
 $s_0 \xrightarrow{1\varepsilon} sf$



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 celalătă, nu

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

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.

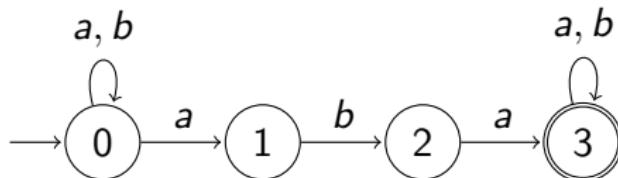
Diverse *algoritmi de minimizare* (ex. Hopcroft-Ullman, Moore)

initial, partiție cu 2 blocuri:  $F, S \setminus F$  (stări acceptoare sau nu)  
(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)

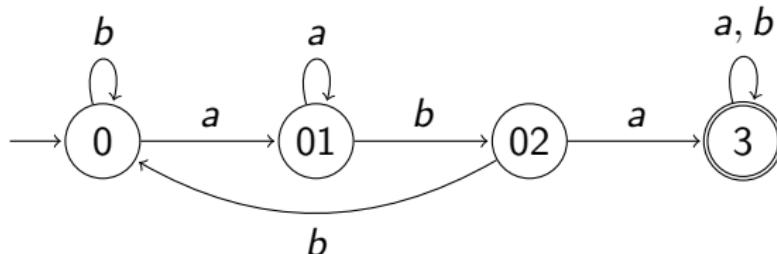
Cuvinte din  $a, b$  cu subșir  $aba$ : “ghicim” când începe subșirul dorit



	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 **accepțoare**.

Aici, ele trec tot timpul în stări accepțoare, deci sunt **echivalente** (caz simplu), și le putem comoda 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 exponential

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

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