Logică și Structuri Discrete -LSD

Cursul 13 – Automate. Expresii regulate

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

În cursul de azi

Sisteme cu comportament simplu: *automate*

un model pentru calcule cu *memorie finită*

Limbaje (mulțimi de șiruri) 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 ?

- *buton* — nu
- *fisă* — nu imediat
- *fisă buton* — da
  - *fisă* a avut un efect *intern*: automatul a trecut în altă *stare* (se *comportă altfel* la acțiunea *buton*)
- *fisă fisă buton buton* — dă două cafele ?
  - dacă da, câte fise poate ține minte ?
  - una sau mai multe, dar practic un număr *finit* ⇒ *stări finite*

Automate în practică

Multle sisteme se pot modela ca automate:

- numărătoare, afișaje, control simplu pornit/oprit
- *protocoale* de comunicație: trimite, primește, așteaptă, ...

Automate în practică

Multe sisteme se pot modela ca automate:

numărătoare, afișaje, control simplu pornit/oprit

protocoale de comunicație: trimite, primește, așteaptă, ...

Automatele sunt un *model* pentru ce se poate *calcula* cu *memorie finită* (automate cu număr finit de stări)

Automate în practică

Multle sisteme se pot modela ca automate:

numărătoare, afișaje, control simplu pornit/oprit

protocoale de comunicație: trimite, primește, așteaptă, ...

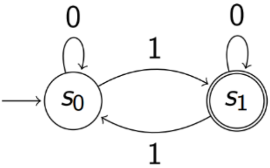
Automatele sunt un *model* pentru ce se poate *calcula*
cu *memorie finită* (automate cu număr finit de stări)

Alte probleme legate de automate:

Testarea cu diverse *secvențe de intrări*:

corespunde specificației?

Un automat foarte simplu

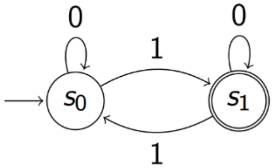


începe în *starea* s_0
când primește 1, schimbă starea
când primește 0, stă pe loc

După un șir cu număr *par de 1*, automatul va fi în s_0
După un șir cu număr *impar de 1*, automatul va fi în s_1

⇒ automatul poate *deosebi* cele două feluri de șiruri

Un automat foarte simplu



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

După un șir cu număr *par de 1*, automatul va fi în s_0
După un șir cu număr *impar de 1*, automatul va fi în s_1

⇒ automatul poate *deosebi* cele două feluri de șiruri

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

⇒ automatul definește o *mulțime de șiruri*, adică un *limbaj*

Ce e un limbaj

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

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

Ce e un limbaj

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, ...

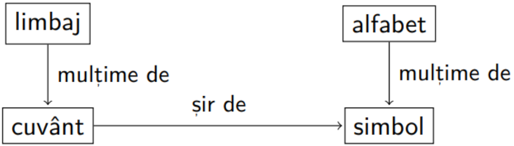
Un *limbaj* e o *mulțime de cuvinte* (șiruri)
ca orice mulțime, definită explicit: {*a, ab, ac, abc*}
sau după o regulă: șiruri de *a*, *b*, încep cu *a*, mai mulți *a* decât *b*

Ce e un limbaj

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

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

Ce e un limbaj (formal)

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

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

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

Ce e un limbaj (formal)

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

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

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

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

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

$*$ 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*

Ce e un limbaj (formal)

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

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

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

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

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

$*$ 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 $L \subseteq \Sigma^*$, 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.

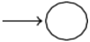
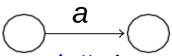
Automat finit determinist (DFA)

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

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)

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

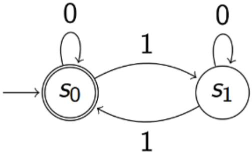
- Σ e un *alfabet* finit nevid de *simboluri* de intrare $\{a, 0, 1, \ldots\}$
- S e o mulțime finită nevidă de *stări*
- $s_0 \in S$ e *starea inițială* (*una*, în definiția uzuală) 
- $\delta : S \times \Sigma \to S$ e *funcția de tranziție* 

  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)

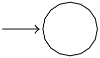
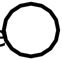
Exemplu de automat determinist (1)

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

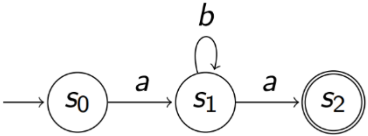
Stările acceptoare *pot* avea tranziții:

aici, din s_0 se iese la citirea lui 1

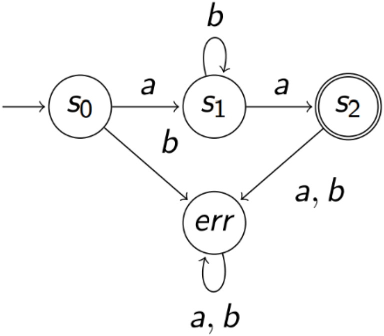
contează starea în care ajunge *când se termină* șirul

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

Limbajul acceptat de un automat

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

Definim o funcție de tranziție $\delta^* : S \times \Sigma^* \to S$ cu intrări *cuvinte*:

în ce stare ajunge automatul pentru un cuvânt dat la intrare?

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

$\delta^*(s, \varepsilon) = s$ cuvânt vid: nu face nimic

$\delta^*(s, a_1a_2 \ldots a_n) = \delta^*(\delta(s, a_1), a_2 \ldots a_n)$ pentru $n > 0$

Altfel spus, $\delta^*(s_0, a_1a_2 \ldots a_n) = \delta^*(s_1, a_2 \ldots a_n)$ cu $s_1 = \delta(s_0, a_1)$
obținem starea s_1 după intrarea a_1, și aplicăm δ^* pe șirul rămas

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

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 mulțime acceptoare F
  în plus: un *alfabet de ieșire* Ω și o *funcție de ieșire* g

automate de tip *Moore*
  ieșirea e funcție de *stare*: $g : S \to \Omega$

automate de tip *Mealy*
  ieșirea e funcție de *stare* și *intrare* $g : S \times \Sigma \to \Omega$

folosite pentru a modela *circuite secvențiale*

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*

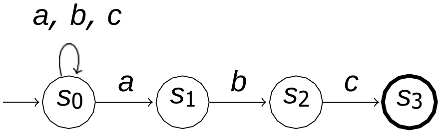
Automatul pentru *intersecția* a două limbaje $L_1 \cap L_2$
(numit uzual automatul produs)
are stări din *produsul cartezian* $S_1 \times S_2$ al stărilor
tranziționează *simultan* în ambele automate acceptă
dacă *ambele* acceptă

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 *terminǎ* în *abc*



Din s_0, primind simbolul *a*, automatul poate

– rămâne în s_0

– trece în s_1

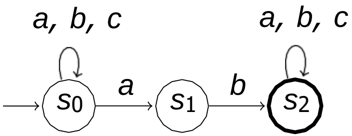
⇒ automatul poate urma *una din mai multe* căi

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

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 $S` = P(S)$

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 $q \in P(S)$,

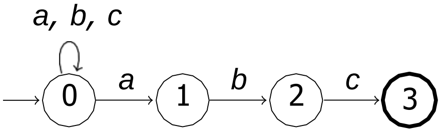
reunim mulțimile stărilor în care se ajunge pe simbolul a

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

mulțimea stărilor care au o stare acceptoare din F

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

Conversie NFA-DFA (exemplu)

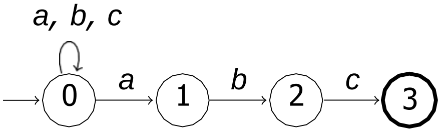


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

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

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

Conversie NFA-DFA (exemplu)

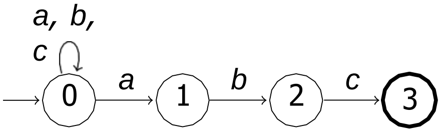


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

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

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

Conversie NFA-DFA (exemplu)

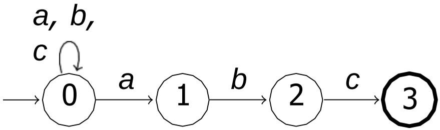


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

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

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

Conversie NFA-DFA (exemplu)



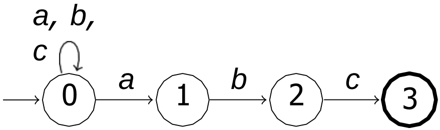
Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	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}

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

Fiecare mulțime obținută devine o stare în DFA-ul rezultat

Conversie NFA-DFA (exemplu)

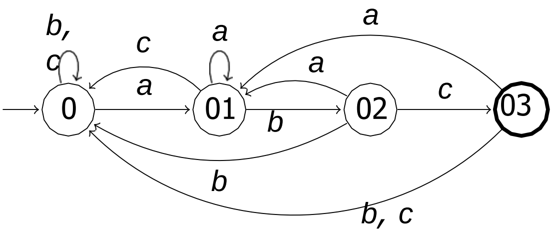


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	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}

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

Fiecare mulțime 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

stare inițială: 1

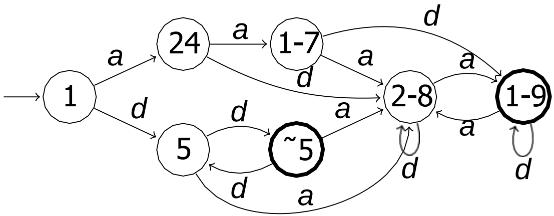
stare accept.: 9

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

a: mută adiacent

d : mută diagonal

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



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

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

~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 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
- ⇒ se pot scrie mai simplu ca *expresii regulate*

Operații 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)

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

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 $L \neq \emptyset$

Expresii regulate: definiție formală

O expresie regulată descrie un limbaj (regulat).

O expresie regulată peste un alfabet Σ e fie:

3 cazuri de bază:

$\emptyset$ — limbajul vid

ε — limbajul $\{\varepsilon\}$ (cu șirul 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 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

Orice expresie regulată e recunoscută de un automat

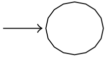
Construcție dată de Ken Thompson (creatorul UNIX, premiul Turing 1983)

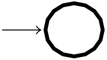
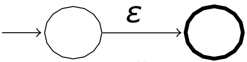
Definim prin *inducție structurală*

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

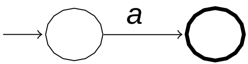
cum *combinăm* automatele în cele 3 *cazuri recursive*

$\Rightarrow$ descompunând, convertim *orice expresie regulată* în automat

$\emptyset$  nu are stare acceptoare

ε  starea inițială e acceptoare sau 

nu consumă simbol

a 

acceptă simbolul *a*

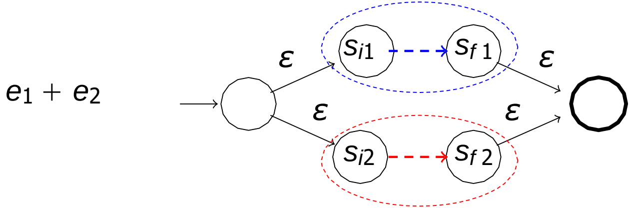
în cele trei cazuri recursive, *combinăm* automatele limbajelor date

$\Rightarrow$ *automat finit nedeterminist* cu tranziții ε (nu consumă simbol)

Conversia în automat: Reuniune/alternativă

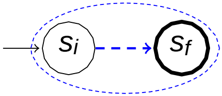
Combinăm automatele pentru cele două expresii regulate
(în oval pot fi alte stări și tranziții)

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



Conversia în automat: Închiderea Kleene

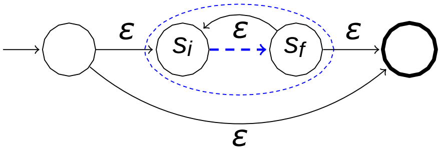
e



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

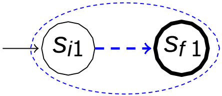
- închid ciclul stare finală $\xrightarrow{\varepsilon}$ inițială în automatul pentru e
- trec direct din starea inițială în cea finală (șirul vid, 0 iterații)
- leagă noua stare inițială și finală de cele ale expresiei interioare

e^*

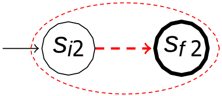


Conversia în automat: Concatenare

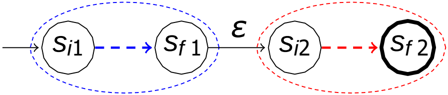
e_1



e_2



e_1e_2

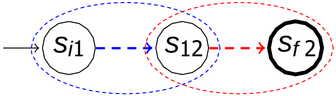


Construcțiile de până acum asigură:

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

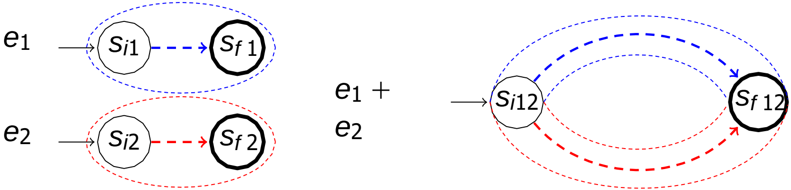
Atunci putem contopi la concatenare capetele lui e_1 și e_2.

e_1e_2

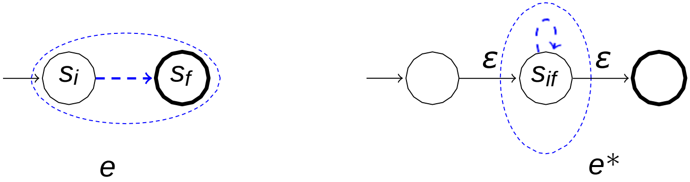


Construcții simplificate

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

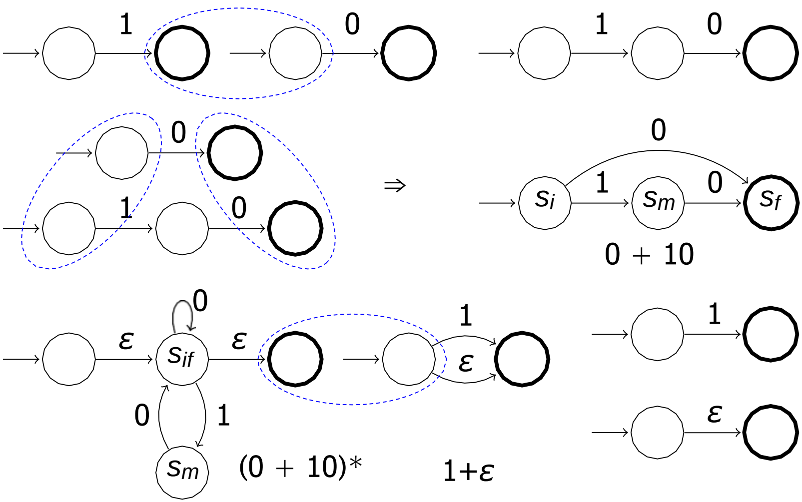


La *închiderea Kleene*, comasăm starea inițială cu cea finală;
cei doi ε consecutivi ne dau cazul cu zero repetiții

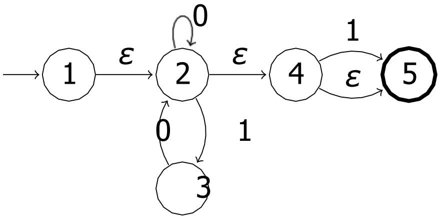


Exemplu: Conversie din expresie regulată în automat

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



Exemplu: Conversie din expresie regulată în automat (2)

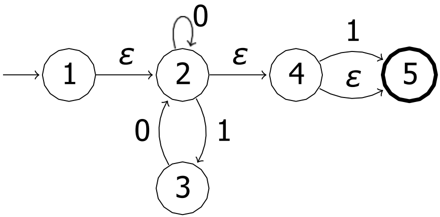


$(0+10)^*(1+\varepsilon)$

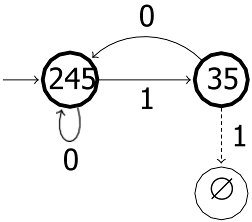
O *tranziție ε* se face spontan, fără a consuma un simbol de intrare
⇒ din starea *s* se ajunge în orice s^j legată prin oricâte ε-tranziții
(*închiderea tranzitivă* a relației definite de ε)

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	$\varnothing$



Tranzițiile pe 0 ne duc direct în 2, apoi prin ε 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 $\varnothing$) Ambele conțin pe 5 $\Rightarrow$ 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

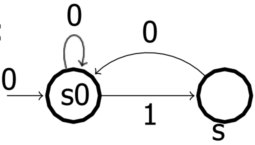
pentru orice pereche de noduri (s, d) adaugă la muchia $s \rightarrow d$ limbajul $L_{si} L^*_{ii} L_{id}$

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

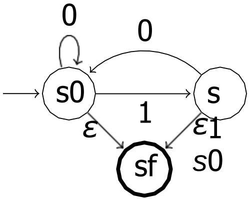
Exemplu: Conversie din automat în expresie regulată

șiruri de 0 și 1 care nu au doi 1 consecutivi:

pe 1, trece în starea s_1 cu tranziție doar pe 0



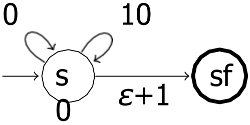
Ambele stări sunt acceptoare ⇒ adăugăm o unică stare acceptoare



Eliminăm s1:

$s0 \xrightarrow{10}$

$s0 \xrightarrow{1\varepsilon} sf$



Obținem astfel limbajul (0+10)*(1+ε)

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

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

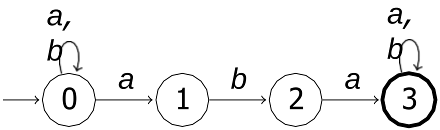
Două stări care nu pot fi deosebite sunt *echivalente*
 $\Rightarrow$ pot fi înlocuite cu o singură stare

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)

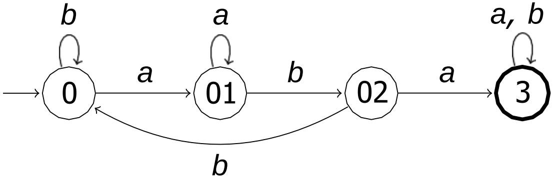
Conversie NFA-DFA și minimizare (exemplu)

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).

Vă mulțumesc!

Bibliografie

Conținutul cursului se bazează pe materialele de anii trecuți de la cursul de LSD, predat de conf. dr. ing.Marius Minea și ș.l. dr. ing. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)