Logică și structuri discrete

Automate finite și expresii regulate

Marius Minea
marius@cs.upt.ro

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

28 noiembrie 2016

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* $\Rightarrow$ *stări finite*

Important în practică:
Testăm cu diverse *secvențe de intrări*: corespunde specificației?
Putem *învăța* structura automatului (din comportament)

Eliminarea comentariilor în C

Comentariu C: încadrat între `/*` și `*/` sau toată linia după `//`
Sursa nu are voie să se termine înăuntrul unui comentariu

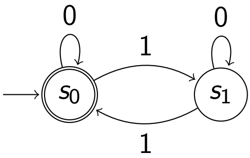
Cum *recunoaștem* un comentariu ?
Cum *decidem* dacă să *acceptăm* o sursă corectă ?

Trebuie să *reținem* unde ne aflăm în prelucrare (*starea*)
caracterele "interesante" la care reacționăm depind de stare:
- în afara comentariului
- înăuntrul comentariului
- așteptând un posibil început de comentariu (după `/`)
- așteptând un posibil sfârșit de comentariu (după `*`)

Acceptăm programul dacă în final, suntem în afara comentariului
altfel *respingem* (nu acceptăm) programul

Nu toate regulile de sintaxă pot fi descrise prin automate
în general, se folosesc *gramatici* (cursul următor)

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

$\Rightarrow$ automatul poate *deosebi* cele două feluri de șiruri

Dacă ne trebuie un număr par de 1, marcăm s_0 ca *stare acceptoare*
șir acceptat: doar dacă automatul se oprește în stare acceptoare

$\Rightarrow$ automatul definește o *mulțime de șiruri*, adică un *limbaj*

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

$$a_1 a_2 \ldots a_n \qquad a_i \in \Sigma$$

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)
(în *expresii regulate*, vezi ulterior)

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

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

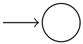
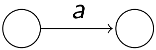
limbajul șirurilor de paranteze echibrate; al șirurilor palindrom; al șirurilor de 0 și 1 care nu au trei 0 consecutivi; etc.

Automat finit determinist (DFA)

Într-un automat trebuie să definim stările, trecerile dintr-o stare în alta (tranzițiile), starea inițială, și unde dorim să ajungem.

Automat finit determinist: o mulțime de *stări* (unele acceptoare), o *stare inițială*, și *tranziții* în funcție de *simbolurile* de intrare.

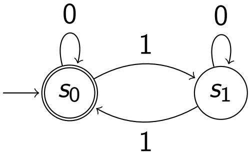
Un automat finit e un tuplu cu 5 elemente (cvintuplu) $(\Sigma, S, s_0, \delta, F)$

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

  (pentru fiecare stare și intrare, dă starea următoare)
- $F \subseteq S$ e mulțimea stărilor *acceptoare* 
  (unde dorim să ajungem)

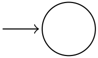
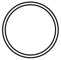
Exemple de automate deterministe

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

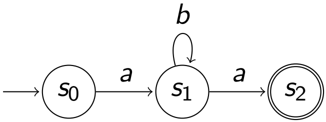


sau ca tabelă de tranziții

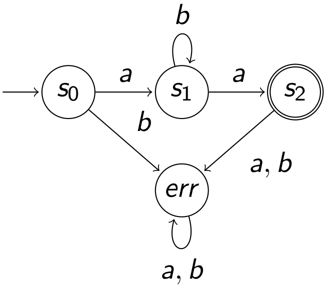
	0	1
s_0	s_0	s_1
s_1	s_1	s_0

s_0 e stare inițială  și acceptoare  în același timp

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*
uneori în practică se omite



Limbajul acceptat de un automat

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

Definim inductiv o funcție de tranziție δ^* cu intrări *cuvinte*:
în ce stare ajunge automatul pentru un cuvânt dat la intrare?

pentru orice stare $s \in S$:

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

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

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 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

	0	1
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*

Intersecția, reuniunea și complementul limbajelor

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

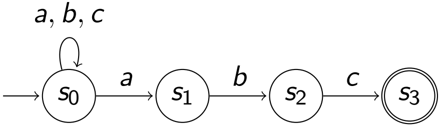
Automatul pentru *intersecția* a două limbaje $\mathcal{L}_1 \cap \mathcal{L}_2$
(numit uzual automatul produs)
tranziționează *simultan* în ambele automate
acceptă dacă *ambele* acceptă:

Automatul pentru *reuniunea* a două limbaje $\mathcal{L}_1 \cup \mathcal{L}_2$
tranziționează *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

Automate finite nedeterministe (NFA)

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



Din s_0, primind a, automatul poate
– încerca sǎ vadǎ dacǎ vor mai fi exact 3 simboluri (trece în s_1)
– rǎmâne în s_0 (prespunând cǎ urmeazǎ mai mult de 3)

$\Rightarrow$ automatul poate urma *una din mai multe* cǎi

Un NFA acceptǎ dacǎ *existǎ* o alegere ducând în stare acceptoare.

Funcția de tranziție e acum $\delta : S \times \Sigma \to \mathcal{P}(S)$
dǎ o *mulțime de stǎri în care poate trece automatul*

Avantaje:
uneori se scrie mai ușor (permite sǎ "ghicim" tranziția bunǎ)
când *specificǎm* un sistem, permite o alegere la implementare

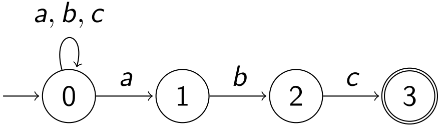
Conversie NFA-DFA

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
$\Rightarrow$ noua mulțime de stări va fi $S' = \mathcal{P}(S)$
în cel mai rău caz, poate fi exponențial în dimensiunea inițială
$|\mathcal{P}(S)| = 2^{|S|}$

Obținem $M' = (\Sigma, S', s_0, \delta', F')$ cu
$S' = \mathcal{P}(S)$
$\delta'(q, a) = \bigcup_{s \in q} \delta(s, a)$ (pentru fiecare stare $s \in q$ cu $q \in \mathcal{P}(S)$,
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)

Conversie NFA-DFA (exemplu)

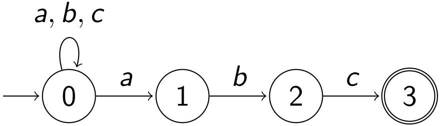


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

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

Când obținem o nouă mulțime adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

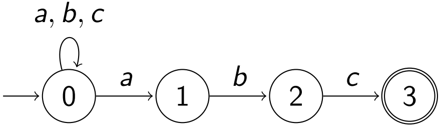


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

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

Când obținem o nouă mulțime adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

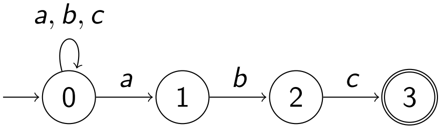


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	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 adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

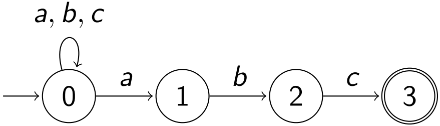


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	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 adăugăm o linie la tabel.

Conversie NFA-DFA (exemplu)

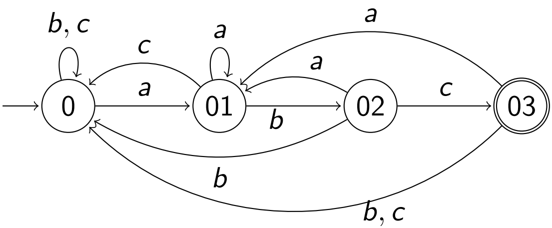


Scriem tabelul de tranziție cu mulțimea stărilor în care se trece pe fiecare simbol

	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 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 finală: 9

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

a: mută adiacent

d: mută diagonal

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

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare finală: 9

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

a: mută adiacent

d: mută diagonal

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

Un alt exemplu: mutări după o regulă

1	2	3
4	5	6
7	8	9

stare inițială: 1

stare finală: 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 finală: 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 finală: 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 finală: 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 finală: 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 inițială: 1

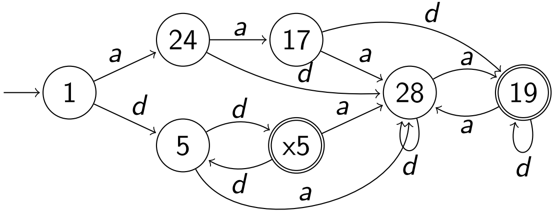
stare finală: 9

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

a: mută adiacent

d: mută diagonal

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



$17 = \{1,3,5,7\}$
$28 = \{2,4,6,8\}$
$x5 = \{1,3,7,9\}$
$19 = \{1,3,5,7,9\}$

Putem exprima mai concis definiția unui limbaj?

Un limbaj = o mulțime de cuvinte peste un alfabet

Adesea suntem interesați în cuvinte cu structură simplă:
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 _
fișiere cu numele `01-`*titlu*`.mp3`, `02-`*alttitlu*`.mp3`, ...

Unele limbaje pot fi recunoscute eficient de *automate finite*
dar scrierea automatului ia efort
$\Rightarrow$ se poate face mai simplu ?

Operații pe limbaje

Reuniunea, intersecția și complementul limbajelor regulate
sunt limbaje regulate

Mai putem defini:
Concatenarea limbajelor
$$L1 \cdot L2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

Închiderea Kleene (repetiția)
$$L^* = \{w \mid \exists n \in \mathbb{N}. w = w_1 w_2 \ldots w_n, w_i \in L\}$$
nu repetiția ~~aceluiași~~ șir, ci concatenarea *oricăror* șiruri
luând $n = 0$, rezultă $\epsilon \in L^*$ pentru orice $L \neq \emptyset$
ϵ reprezintă *șirul vid* (niciun simbol, lungime 0)

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	denotă limbajul $\{\epsilon\}$ (cu șirul vid)
a	denotă limbajul $\{a\}$ cu $a \in \Sigma$

3 cazuri recursive: date e_1, e_2 expresii regulate, și următoarele sunt:

$(e_1 + e_2)$	reuniunea limbajelor
	în practică, notată adesea $e_1 \vert e_2$ (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 $e + \epsilon$ (e, opțional)
  e^+ pentru $e^* \setminus \epsilon$ (e, cel puțin o dată)

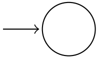
$(0+1)^*$ mulțimea tuturor șirurilor din 0 sau 1
$(0+1)^*0$ ca mai sus, încheiat cu 0 (numere pare în binar)
$1(0+1)^*+0$ numere binare, fără zerouri inițiale inutile

Orice expresie regulată e recunoscută de un automat

Construim prin *inducție structurală*

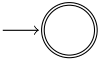
Pentru cazurile de bază

$\emptyset$



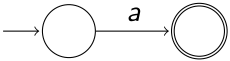
nu are stare acceptoare

ϵ



starea inițială e acceptoare

a



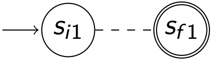
acceptă simbolul *a*

în celelalte trei cazuri, *combinăm* automatele limbajelor date
rezultă un *automat finit nedeterminist* cu tranzițiiϵ

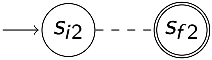
Conversia expresie regulată → automat

Construcția pentru reuniune

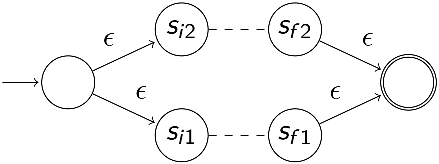
e_1



e_2

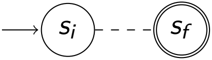


$e_1 + e_2$

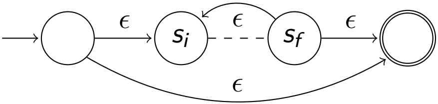


Construcția pentru închiderea Kleene

e

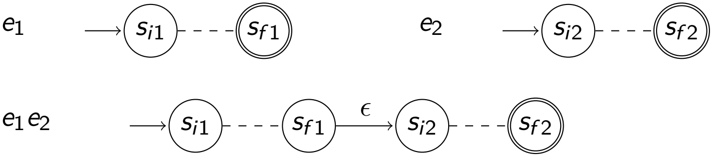


e^*



Conversia expresie regulată $\rightarrow$ automat

Construcția pentru concatenare



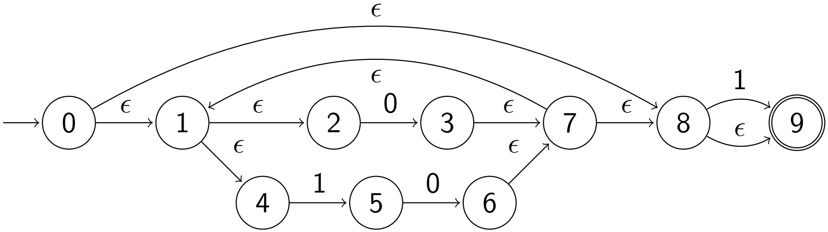
În general, nu putem contopi s_{f1} și s_{i2}: ajunși în s_{f1} nu avem voie să revenim în e_1.
Însă construcțiile de până acum asigură:

- 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.

Conversie din expresie regulată în automat

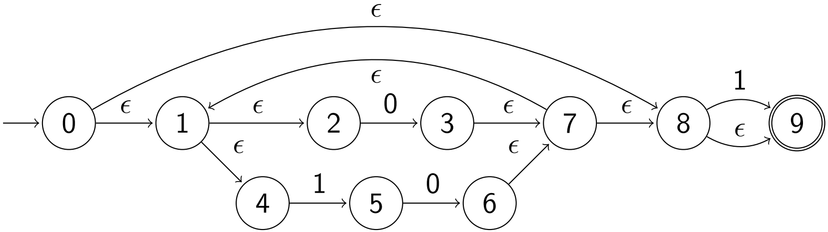
Fie expresia (0+10)*(1+ϵ). Construim (comasând pașii triviali):



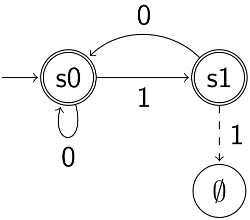
O *tranziție ϵ* se face spontan, fără a consuma un simbol de intrare
$\Rightarrow$ ajuns într-o stare s, e ca și ajuns în orice stare s' legată de s doar prin ϵ-tranziții (*închiderea tranzitivă* a relației definite de ϵ)

Deci, aflat în 0, e ca și cum s-ar afla în 1, 2, 4, 8 sau 9
Aflat în 7, e ca și cum s-ar afla în 1, 2, 4, 8, 9

Conversie din NFA cu tranziții ϵ în DFA



	0	1
012489	1234789	59
1234789	1234789	59
59	1246789	$\emptyset$
1246789	1234789	59



Liniile 1, 2 și 4 au destinații identice $\Rightarrow$ stările sunt echivalente.

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

Conversia din automat în expresie regulată

Dacă sunt mai multe noduri acceptoare, adaugăm un nod acceptor unic (cu tranziții ϵ spre el)

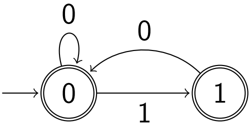
Eliminăm pe rând fiecare nod în afară de cel inițial și acceptor:

pentru orice nod intermediar i de eliminat
  pentru orice pereche (s, d)
    adaugă la muchia $s \rightarrow d$ limbajul $L_{si}L_{ii}^*L_{id}$

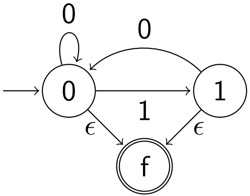
Conversie din automat în expresie regulată

Șiruri de 0 și 1 care nu au doi 1 consecutivi

pe 1, trece în stare cu tranziție doar pe 0



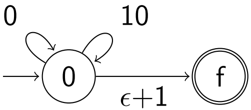
Ambele stări sunt acceptoare ⇒ adăugăm o unică stare acceptoare



Eliminăm 1:

$0 \xrightarrow{10} 0$

$0 \xrightarrow{1\epsilon} f$



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

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

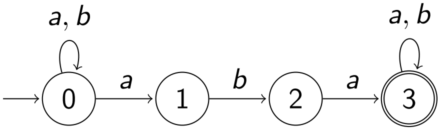
Două stări care nu pot fi deosebite sunt *echivalente*
$\Rightarrow$ pot fi înlocuite cu o singură stare

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 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)

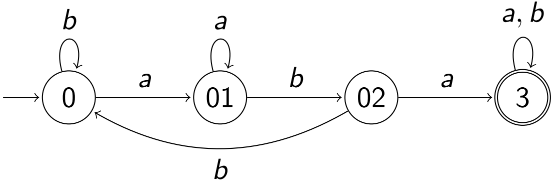
Conversie NFA-DFA și minimizare (exemplu)

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 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 complementarea limbajelor regulate
produc limbaje regulate
(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).