Logică și structuri discrete

Limbaje și automate

Marius Minea
marius@cs.upt.ro

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

25 noiembrie 2013

Un exemplu: automatul de cafea

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

După o acțiune *se întâmplă* ceva ?

buton	nu
fisă	(aparent) nu
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*

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

Putem *învăța* comportamentul automatului (căutăm un *model*)

Testarea de conformanță și învățarea: importante în practică

Eliminarea comentariilor în C

Comentariu C: încadrat între /* și */
Sursa nu are voie să se termine înăuntrul unui comentariu

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

Intuitiv, trebuie să *reținem* unde ne aflăm (*starea*)
- în afara comentariului
- înăuntrul comentariului
- așteptând un posibil început (după /)
- așteptând un posibil sfârșit (după *)

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

Desigur, mai sunt multe alte reguli în afară de comentarii ...

Ce e un limbaj (formal)

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

Un *cuvânt* finit peste alfabetul Σ e un șir $a_1 a_2 \dots a_n$ de simboluri $a_i \in \Sigma$.

Mulțimea tuturor cuvintelor finite peste alfabetul Σ e notată Σ^*

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

steaua Kleene: zero sau mai multe apariții
(în *expresii regulate*, vezi ulterior)

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

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

Automat finit determinist (DFA)

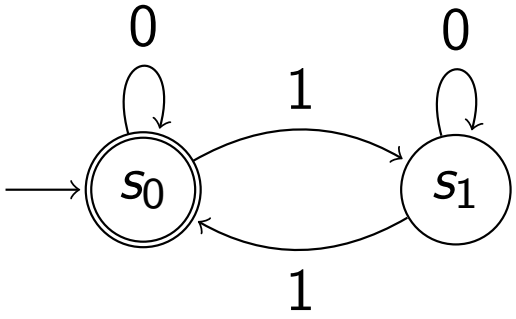
Automat finit determinist: o mulțime de *stări* (unele acceptoare), o *stare inițială*, și *tranziții* în funcție de *simbolurile* de intrare.

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

- Σ e un *alfabet* finit nevid de *simboluri* de intrare
- 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*

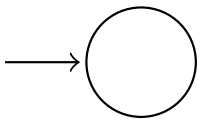
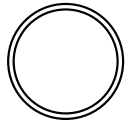
Exemple de automate deterministe

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

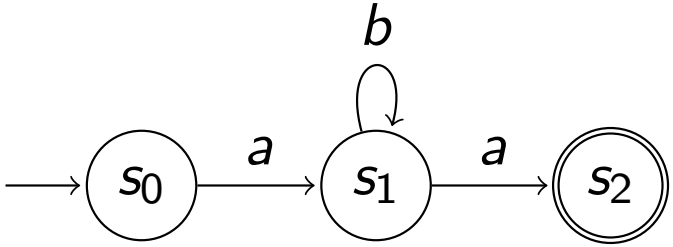


sau ca tabelă de tranziții

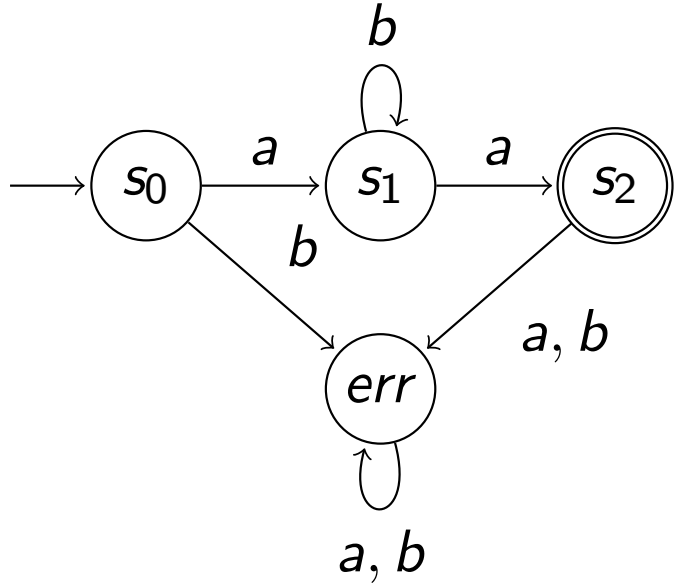
	0	1
s_0	s_0	s_1
s_1	s_1	s_0

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

Convențional, se notează $\epsilon \in \Sigma^*$ cuvântul *vid* (fără niciun simbol).

Definim inductiv o funcție de tranziție δ^* cu intrări *cuvinte*

pentru orice $s \in S$:

$\delta^*(s, \epsilon) = s$ n $= 0$ simboluri

$\delta^*(s, a_0 a_1 \ldots a_n) = \delta^*(\delta(s_0, a_0), a_1 \ldots a_n)$ pentru $n > 0$

Atunci, automatul *acceptă* un cuvânt $w \in \Sigma^*$ 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)

necesită reprezentarea *explicită* a fiecărei combinații

Adesea, multe simboluri nu au niciun efect într-o stare (starea rămâne la fel)

$\Rightarrow$ pentru fiecare stare, un *dicționar* (intrare, stare)

nu trebuie reprezentare intrările care nu au efect

Automate cu ieșiri

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

scopul nu mai e de a *accepta*, dispare mulțimea 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*

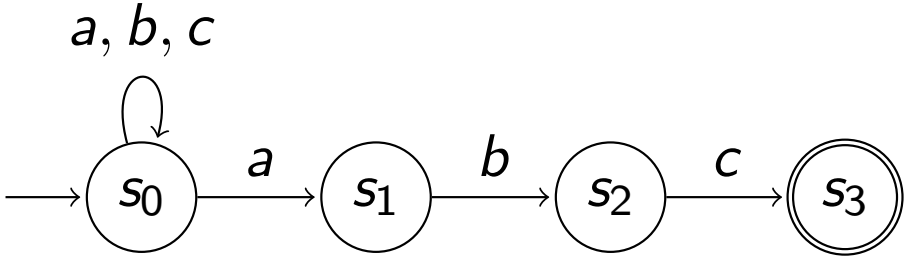
Automatul pentru *intersecția* a două limbaje (automatul produs)
tranziționează *simultan* în ambele automate
acceptă dacă *ambele* acceptă

Automatul pentru *reuniunea* a două limbaje
ca mai sus, dar acceptă dacă *cel puțin unul* acceptă

Automatul pentru *complement*
acceptă dacă automatul original nu acceptă

Automate finite nedeterministe (NFA)

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



Din s_0, primind a, automatul poate rămâne în s_0 sau trece în s_1
$\Rightarrow$ automatul poate urma *una din mai multe* căi
Un NFA acceptă dacă *există* o alegere ducând în stare acceptoare

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

Avantaje:
uneori se scrie mai ușor (permite să "ghicim" tranziția bună)
când *specificăm* un sistem, permite o alegere la implementare

Conversie NFA-DFA

Fie un NFA $M_1 = (\Sigma, S_1, s_0, \delta_1, F_1)$. Construim un DFA echivalent.

Noua mulțime de stări va fi $S_2 = \mathcal{P}(S_1)$

reținem *mulțimea de stări* în care s-ar putea afla M_1

în cel mai rău caz, poate fi exponențial în dimensiunea inițială

Obținem $M_2 = (\Sigma, S_2, s_0, \delta_2, F_2)$ cu

$S_2 = \mathcal{P}(S_1)$

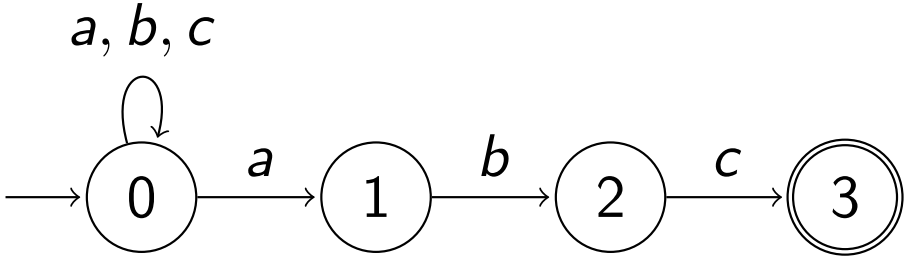
$\delta_2(s, a) = \bigcup_{q \in s} \delta_1(q, a)$

(mulțimea stărilor în care s-ar putea ajunge pe simbolul a

$F_2 = \{s \in S_2 \mid s \cap F_1 \neq \emptyset\}$

(mulțimea stărilor care au o stare acceptoare din F_1)

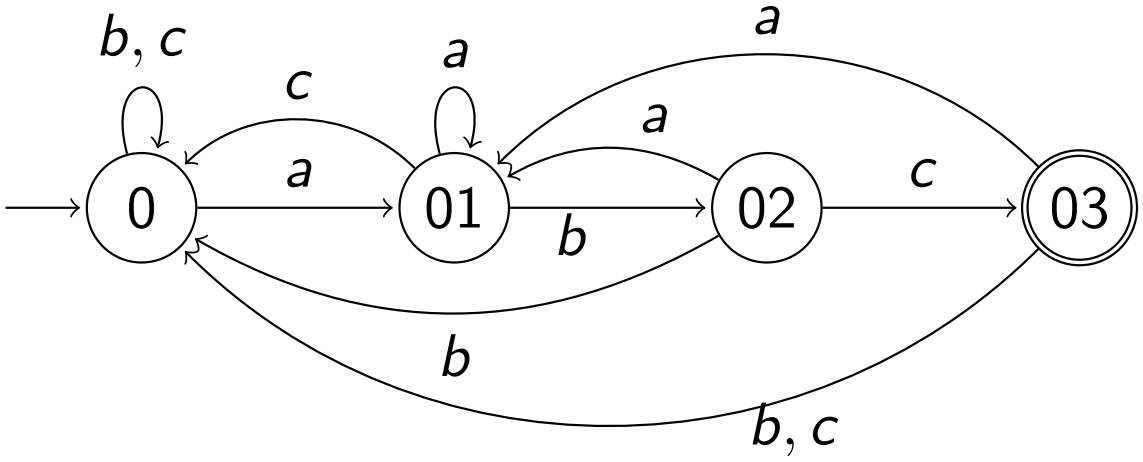
Conversie NFA-DFA (exemplu)



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

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

Fiecare mulțime obținută devine o stare în DFA-ul rezultat



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

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

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.

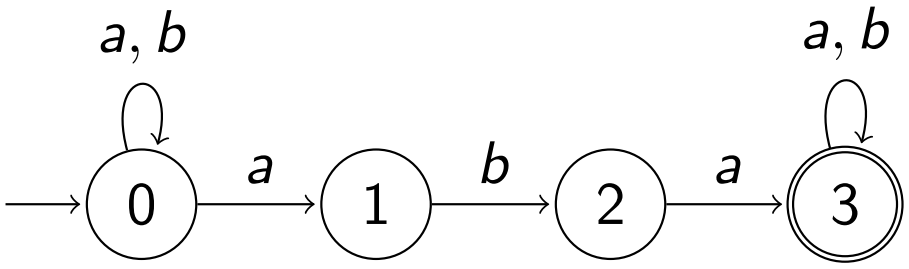
Diverși *algoritmi de minimizare* (ex. Hopcroft-Ullman, Moore)

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)

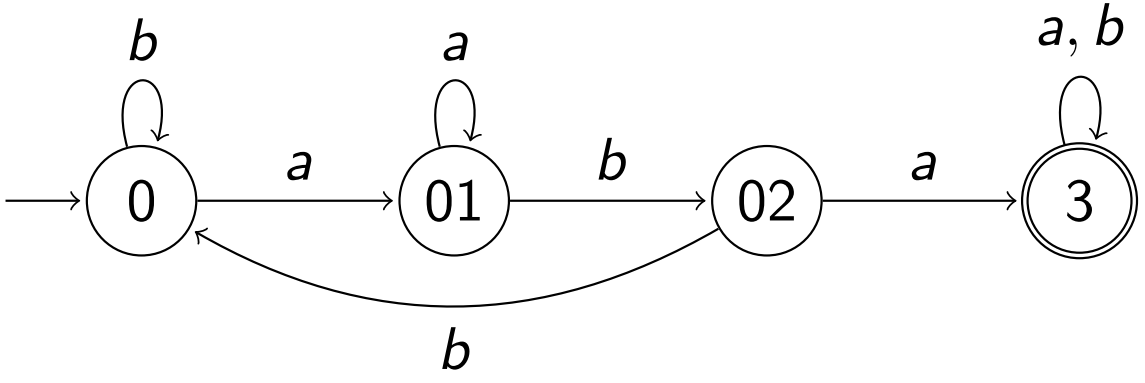
Conversie NFA-DFA și minimizare (exemplu)

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*.

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*.