

# **Limbaje formale si tehnici de compilare.**

## **Introducere**

2023-2024

# Translator. Compilator.

Translator: program

care traduce textul unui program, numit program sursa,  
intr-un alt program, numit program obiect.

Cele doua programe sunt echivalente din punct de vedere  
semantic.

Compilator: translator

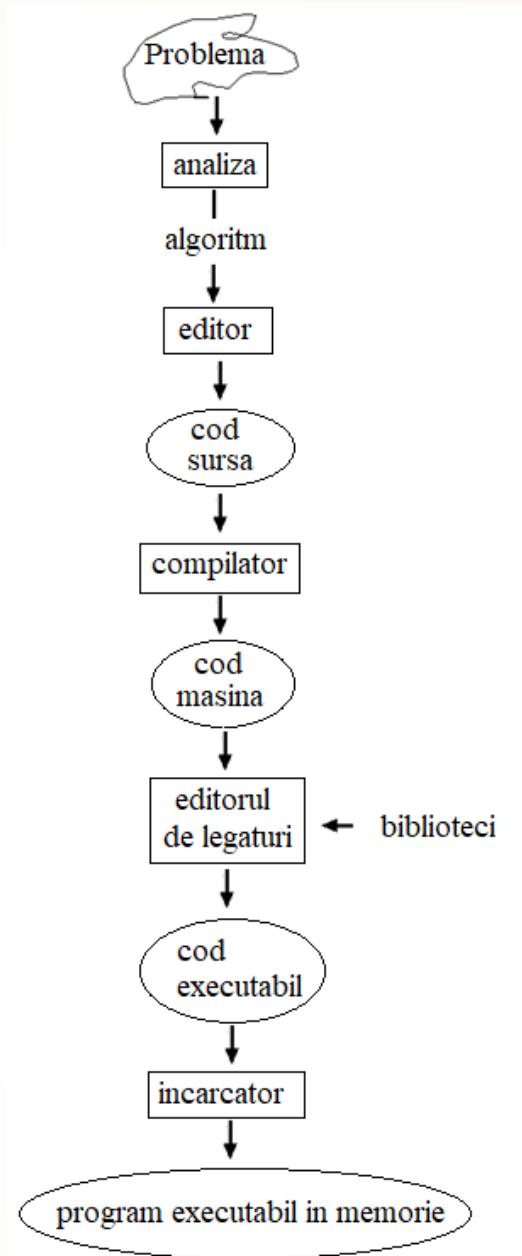
programul obiect este în cod (limbaj) masina  
(sau cod al unei masini virtuale)



- Ex.: tpc, bcc, gcc

# Compilator

- Locul compilatorului



# Compilator vs. interpretor

- Compilator
  - parcurge întregul program și il traduce în cod de mașină
- Interpretor:
  - (program)
  - la un moment dat
    - citește o singură linie de program,
    - o traduce în cod mașină
    - o execută

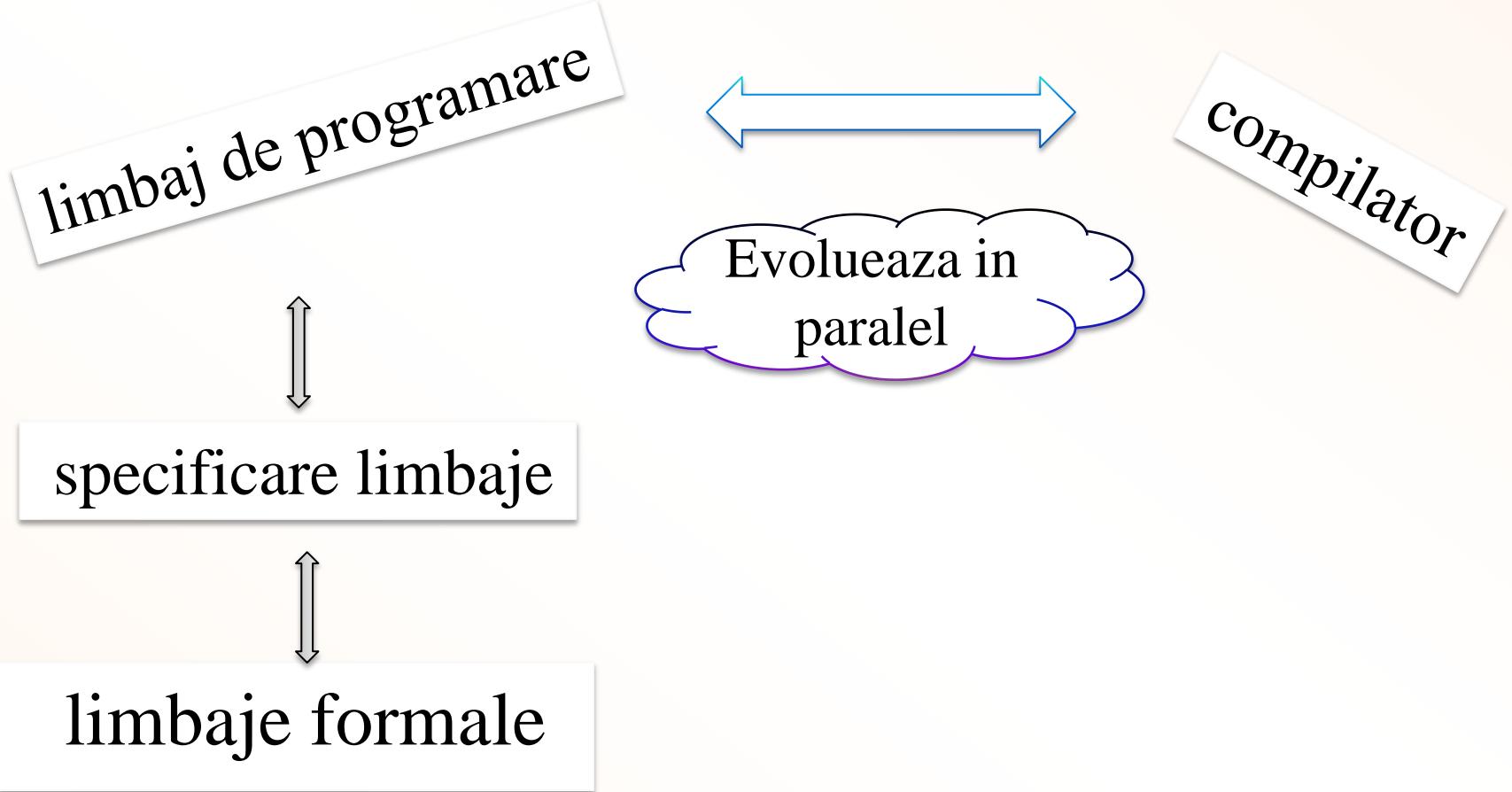


Un compilator este mai rapid decât un interpretor.

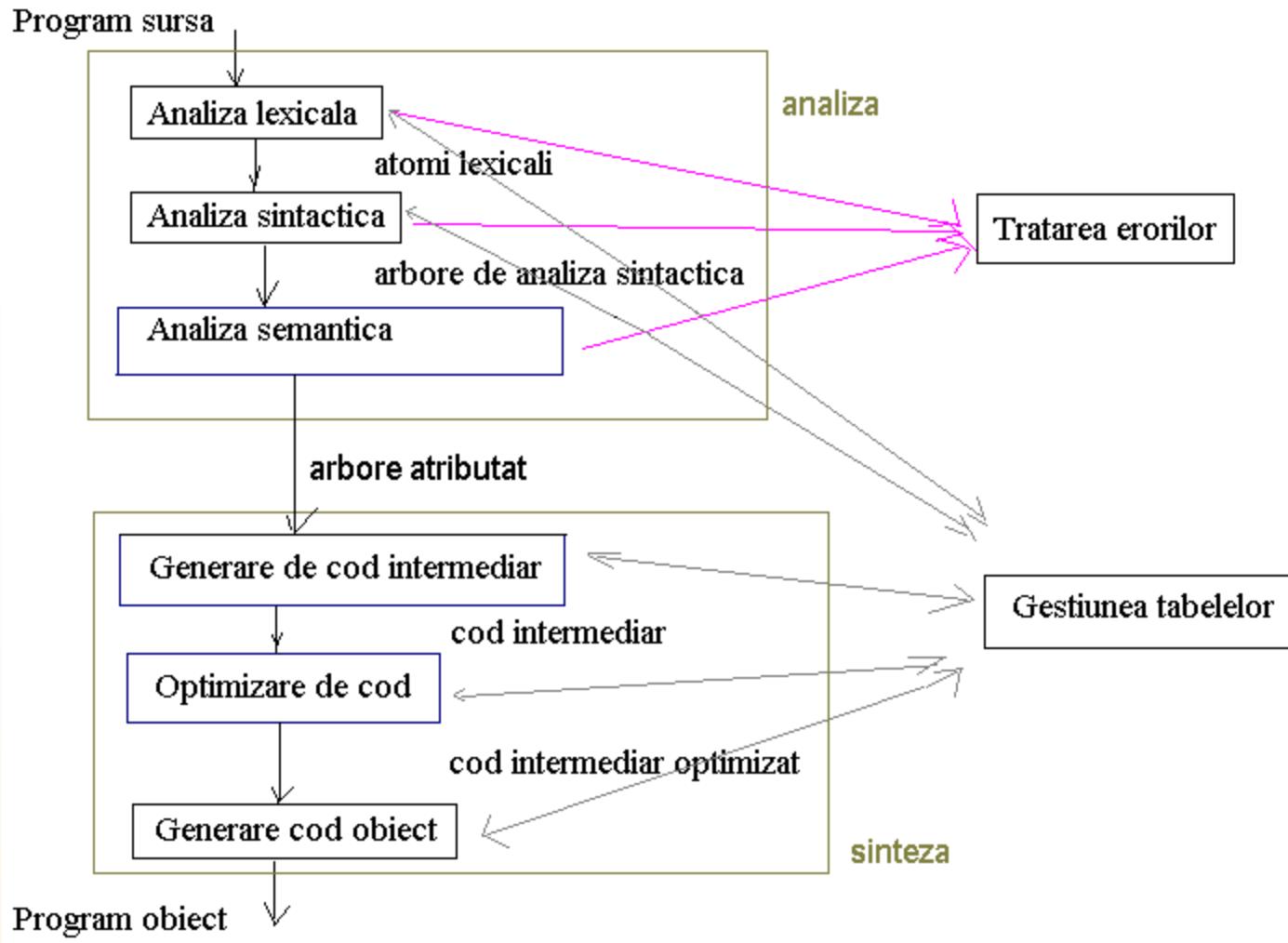
# Limbaj de programare

- De ce e importanta specificarea limbajului de programare ?  
Comunicare între:
  - proiectantul limbajului,
  - programatorii care scriu compilatoarele, ...
  - programatorii care utilizeaza compilatoarele
- Ce specificăm?
  - Reguli referitoare la corectitudinea textului ca limbaj de programare (semantică statică)
  - Semanticii programelor (numită și semantică de rulare)
- Ce mecanisme se folosesc pentru specificare?
  - ...

# Limbaj de programare



# Structura unui compilator



# Limbaje formale si tehnici de compilare. Continut

## Limbaj formal

Limbaj regulare:

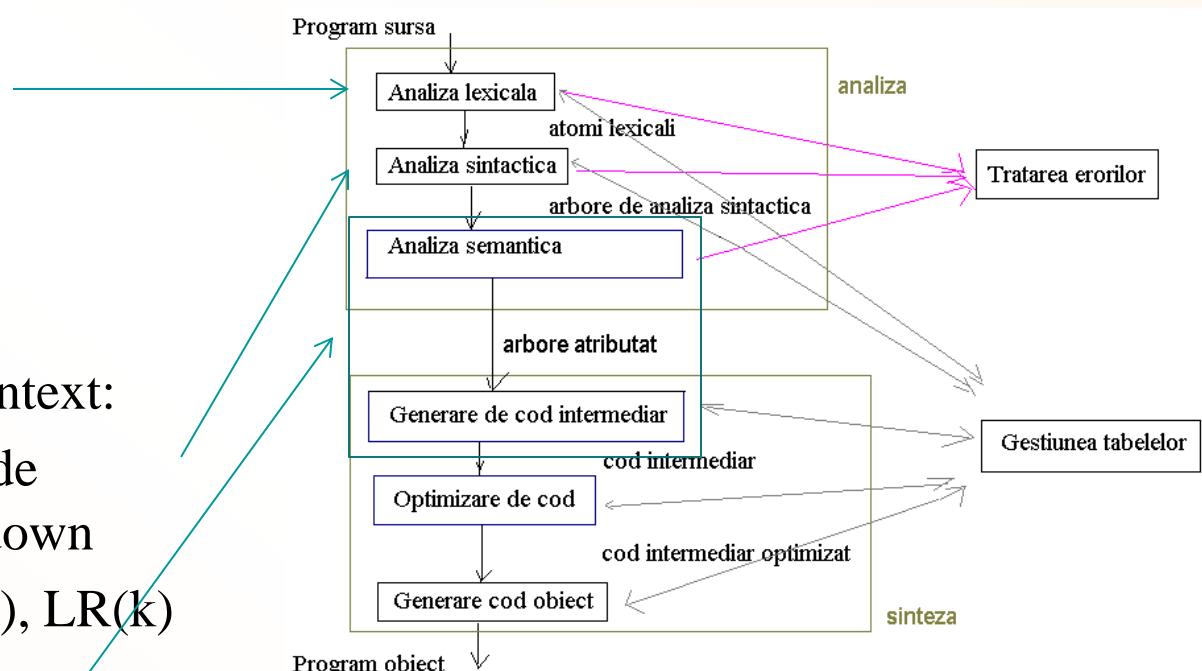
- gramatici regulare,
- automate finite,
- expresii regulare.

Limbaje independente de context:

- Gramatici independente de context, automate push-down
- Gramatici speciale: LL( $k$ ), LR( $k$ )

Gramatici de atrbute

## Structura unui compilator



# Notiuni de baza

- alfabet
- secventa
  - lungime
  - secv. vida
- concatenare
  - notatii
- subsecventa
  - prefix
  - sufix
- multimi speciale
- proprietati  $\Sigma^*$
- limbaj
  - cuvant
- tipuri de limbaje
- specificarea unui limbaj
- operatii cu limbaje

# Alfabet

- Def:

Alfabet = o multime finită și nevidată de elemente numite simboluri

- Notatie:  $\Sigma$

# Secventa

Def.

- **secventa peste  $\Sigma$**   
o succesiune finita de simboluri din  $\Sigma$
- **subsecventa**  
o succesiune de simboluri consecutive dintr-o secventa

# Lungimea unei secvențe

- def:  
nr. de simboluri din care este formată acea secvență
- notatie  
 $| \dots |$   
Ex:  $|abc| = 3$
- **Secv. vida** = secv. de lungime 0
- notatie:  $\varepsilon$       (unele surse  $\lambda$ )

# Concatenare

Dacă

$$x = a_1 a_2 \dots a_n$$

$$y = b_1 b_2 \dots b_m$$

atunci  $z = a_1 \dots a_n b_1 \dots b_m$

reprezintă **concatenarea** secvențelor  $x$  și  $y$

și se notează  $z = xy$

Notatii:  $aa\dots aa = a^n$

(de  $n$  ori)

Exemplu:  $a^2 = aa$

# Secventa

Def.

- **secventa peste  $\Sigma$**   
o succesiune finita de simboluri din  $\Sigma$
- **subsecventa**  
o succesiune de simboluri consecutive dintr-o secventa

w2 – subsecventa a lui w1

daca  $\exists$  secventele u, v    a.i.                 $w1 = u \ w2 \ v$

# Prefix, sufix

Fie **x**, **y**, **z** sunt secvențe peste alfabetul  $\Sigma$

- **x** este un **prefix** al secvenței **xy**
- **y** un **sufix** al secvenței **xy**
- **Prefix**: o subsecvență care
  - fie este vîdă
  - fie începe cu primul simbol al secvenței date
- **Sufix**: o subsecvență care
  - fie este vîdă
  - fie se termină cu ultimul simbol al secvenței date

# Multimi speciale

- $\Sigma^n = \{ w \mid w \text{ - secenta peste } \Sigma, |w| = n \}$
- $\Sigma^* = \{ w \mid w \text{ - secenta peste } \Sigma, 0 \leq |w| \}$  lungimea lui  $w \geq 0$
- $\Sigma^+ = \{ w \mid w \text{ - secenta peste } \Sigma, 0 < |w| \}$  lungimea lui  $w$  e mai mare strict decat 0
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

# Operatia \*

Denumiri:

- Operatia: \*, steaua lui Kleene
- Inchiderea lui Kleene

$\Sigma^*$  - inchiderea alfabetului

- multimea tuturor secventelor ce se pot obtine folosind secvențe din  $\Sigma$

( Similar pentru limbaje)

## $\Sigma^*$ - proprietati

- $\forall w_1, w_2 \in \Sigma^*$  avem:  $w_1 w_2 \in \Sigma^*$
- $\forall w_1, w_2, w_3 \in \Sigma^*$  avem:  $(w_1 w_2) w_3 = w_1 (w_2 w_3)$
- $\forall w \in \Sigma^*, \varepsilon w = w\varepsilon = w$

**$(\Sigma^*, \cdot)$  - monoid**

# Limbaj, cuvant

- def: (limbaj)

L – limbaj peste alfabetul  $\Sigma$

daca  $L \subseteq \Sigma^*$

- def: (cuvant)

Cuvant al unui limbaj – un element al limbajului

# Metode de specificare a unui limbaj

- enumerand elementele
- evidențierea unor proprietăți ale elementelor
  - folosind multimi și descrieri matematice
  - ...
- folosind gramatici, automate, expresii regulate
- ...

# Cateva tipuri de limbaje

Multime & descriere matematica a proprietatilor

- **teoretice**

$$L = \{a^n \mid n \in \mathbb{N}\}$$

limbaj peste  $\Sigma = \{a\}$

$$L = \{a^n b^n \mid n \in \mathbb{N}\}$$

limbaj peste  $\Sigma = \{a, b\}$

- **informatică**

limbajul identificatorilor

$$\Sigma = \{a, \dots, z, A, \dots, Z, \_, 0, \dots, 9\}$$

Enumerarea elementelor

$$L = \{a'w' \mid a' \in \{a, \dots, z, A, \dots, Z, \_\}, w' \in \Sigma^*\}$$

- **matematice**

ex: limbajul reprezentarii zecimală a numerelor naturale

# Operatii cu limbaje (1)

Fie:  $L_1$  – limbaj peste  $\Sigma_1$        $L_2$  – limbaj peste  $\Sigma_2$   
(operatii cu multimi)

- $L_1 \cup L_2$  limbaj peste  $\Sigma$  ales corespunzator;

de exemplu:  $\Sigma = \Sigma_1 \cup \Sigma_2$

- $L_1 \cap L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1 \cap \Sigma_2$ )

- $L_1 - L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1$ )

- $L_1 L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1 \cup \Sigma_2$ )

(operatii bazate pe concatenare)

- câtul la dreapta:  $L_1 / L_2 = \{ w \in \Sigma^* | \exists y \in L_2: wy \in L_1 \}$

- câtul la stânga:  $L_1 \setminus L_2 = \{ w \in \Sigma^* | \exists y \in L_2: yw \in L_1 \}$

# Operatii cu limbaje (2)

- L limbaj peste un alfabet  $\Sigma$
- complementara:  $\overline{L} = \{x \in \Sigma^* \mid x \notin L\}$

- Închiderea reflexivă și tranzitivă:

$$L^* = \bigcup_{n \geq 0} L^n \quad \text{unde } L^n = LL^{n-1}, \quad L^0 = \{\epsilon\};$$

- Închiderea tranzitivă:

$$L^+ = \bigcup_{n \geq 1} L^n \quad \text{sau } L^+ = LL^*, \quad L^* = L^+ \bigcup \{\epsilon\}$$

# Gramatica independenta de context

O gramatica independenta de context este un cvadruplu

$$\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$$

- $\mathbf{N}$  este un alfabet de simboluri **neterminale**

- $\Sigma$  este un alfabet de simboluri **terminale**

- $\mathbf{N} \cap \Sigma = \emptyset$

- $P \subseteq \mathbf{N} \times (\mathbf{N} \cup \Sigma)^*$

$P$  multime finită (multimea regulilor de productie)

- $S \in \mathbf{N}$  (simbolul de start - simbolul initial)

Notatie:

$(\alpha, \beta) \in P$  se noteaza:  $\alpha \rightarrow \beta$

( $\alpha$  se înlocuieste cu  $\beta$ )

Folosita pentru a specifica un limbaj

# Doua tipuri de gramatici

- Gramatica independenta de context:

reg. productie sunt de forma  $A \rightarrow \alpha$ ,  $A \in N$ ,  $\alpha \in (N \cup \Sigma)^*$

- Gramatica regulara:

reg. prod. sunt de forma

- $A \rightarrow aB$

- $A \rightarrow b$

unde  $A, B \in N$  si  $a, b \in \Sigma$

caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . In acest caz S nu apare în membrul drept al nici unei reguli de productie.

# Backus Naur Form

- primitive (terminale)
- variabile metalingvistice : intre paranteze unghiulare
- conective metalingvistice: ::= , |

<intreg\_fara\_semn> ::= <cifra> |  
                                  <cifra> <intreg\_fara\_semn>

<cifra> ::= 0 | 1 | ... | 9

# Extended Backus Naur Form

- primitive (terminale): intre “ghilimele”
- variabile metalingvistice
- conective metalingvistice: = , | ,  
[ optional ] , { de oricate ori }
- punctul este folosit ca sfarsit de regula

intreg\_fara\_semn = cifra { cifra } .

cifra = “0” | “1” | ... | “9” .

# Operatii cu limbaje (1)

Fie:  $L_1$  – limbaj peste  $\Sigma_1$        $L_2$  – limbaj peste  $\Sigma_2$   
(operatii cu multimi)

- $L_1 \cup L_2$  limbaj peste  $\Sigma$  **ales corespunzator**;

de exemplu:  $\Sigma = \Sigma_1 \cup \Sigma_2$

- $L_1 \cap L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1 \cap \Sigma_2$ )

- $L_1 - L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1$ )

- $L_1 L_2$  limbaj peste  $\Sigma$       ( $\Sigma = \Sigma_1 \cup \Sigma_2$ )

(operatii bazate pe concatenare)

- câtul la dreapta:  $L_1 / L_2 = \{ w \in \Sigma^* | \exists y \in L_2: wy \in L_1 \}$

- câtul la stanga:  $L_1 \setminus L_2 = \{ w \in \Sigma^* | \exists y \in L_2: yw \in L_1 \}$

# Operatii cu limbaje (2)

- L limbaj peste un alfabet  $\Sigma$
- complementara:  $\overline{L} = \{x \in \Sigma^* \mid x \notin L\}$

- Închiderea reflexivă și tranzitivă:

$$L^* = \bigcup_{n \geq 0} L^n \quad \text{unde } L^n = LL^{n-1}, \quad L^0 = \{\epsilon\};$$

- Închiderea tranzitivă:

$$L^+ = \bigcup_{n \geq 1} L^n \quad \text{sau } L^+ = LL^*, \quad L^* = L^+ \bigcup \{\epsilon\}$$

# Analiza lexicală

Atomi lexicali:

- identificatori
- constante (literali)
- cuvinte cheie (*cuvinte rezervate*)
- operatori              aritmetici, relationali, ...  
separatori

Clase de atomi  
lexicali

spatiile "albe"

cuvinte cheie - simboluri sintactice

cuvinte rezervate - nu pot fi folosite ca identificatori  
(semnificatii speciale)

# Analiza lexicală

## Analiza lexicală

- Date: fisier text
  - Rezultate:
    - FIP, TS
    - sau: mesaj de eroare
- 

### TS: tabela de simboluri

- informații despre identificatori și constante  
nume, tip, ...

### FIP: forma internă a programului

- Cod atom lexical
- Poziția în TS (unde e cazul)

# Analiza lexicala, FIP, TS – exemplu partial

if a < b then b:=a

- Care sunt elementele lexicale ?
- Descrieti sintaxa instructiunilor.
- Creati un alt exemplu care respecta sintaxa descrisa.

Cod	Atom lexical
0	ID
1	CONST
2	:=
3	<
4	if
5	then

FIP

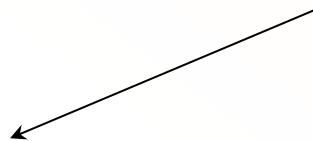
Cod atom	Pozitie in TS
4	
0	1
3	
0	2
5	
0	2
2	
0	1

TS

Pozitie	Atom lexical	Alte informatii
1	a	...
2	b	...

# Analiza lexicala (versiunea 1)

**detecteaza**



Pentru fiecare *atom* lexical din fisierul sursa

clasifica *atom*

codifica *atom*

Sf.pentru

# Analiza lexicala (versiunea 2)

Cat timp (mai exista caractere necitite in prg. sursa) ex.

detecteaza *atom*

clasifica *atom*

codifica *atom*

Sf.cattimp

# Analiza lexicala (versiunea 3)

## Clasifica

Cat timp (mai exista caractere necitite in prg. sursa) ex

detecteaza atom

**Daca** e cuvant-cheie sau operator sau separator atunci  
adauga in FIP codul corespunzator atomului

**Altfel**

**Daca** e identifier sau constanta

Daca nu exista deja in TS atunci adauga-l ; Sf. Daca  
adauga in FIP codul coresp. atomului si pozitia in TS

altfel

Eroare

Sf.Daca

Sf. Daca

**Sf.cattimp**

# Analiza lexicala (versiunea 4)

Cat timp (mai exista caractere necitite in prg. sursa) ex  
detecteaza *atom*

Daca *atom* e cuvant-cheie sau operator sau separator atunci  
adaugaFIP(cod(*atom*), 0)

Altfel

Daca *atom* e identificator atunci

indice:=poz(*atom*, TS)  
adaugaFIP((cod\_id, indice))

altfel Daca *atom* e constanta atunci

indice:=poz(*atom*, TS)  
adaugaFIP((cod\_const, indice))

altfel

MesajEroare:, indicatie asupra erorii

sfDaca

Sf.Daca

Sf. Daca

Sf.cattimp

Obs.: multe cautari in TS

++ urmatoarele faze !!

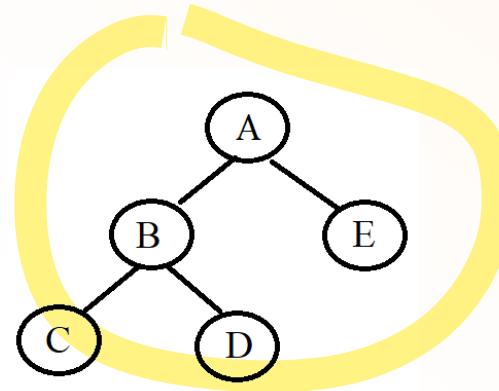
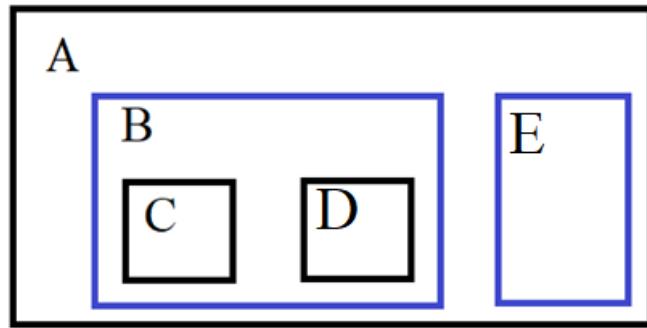
- TS fara organizare
  - cautare secventiala
- organizare pt. cautare mai rapida
  - ordonare (ordonata lexicografic)
    - tabel ordonat lexicografic
    - arbore binar de cautare echilibrat
  - tabela de dispersie

→cautare – fct. de dispersie  
(hash)



Probleme ?

# TS si programe cu mai multe blocuri



Obs.:  
noi vom lucra doar cu  
un singur bloc program

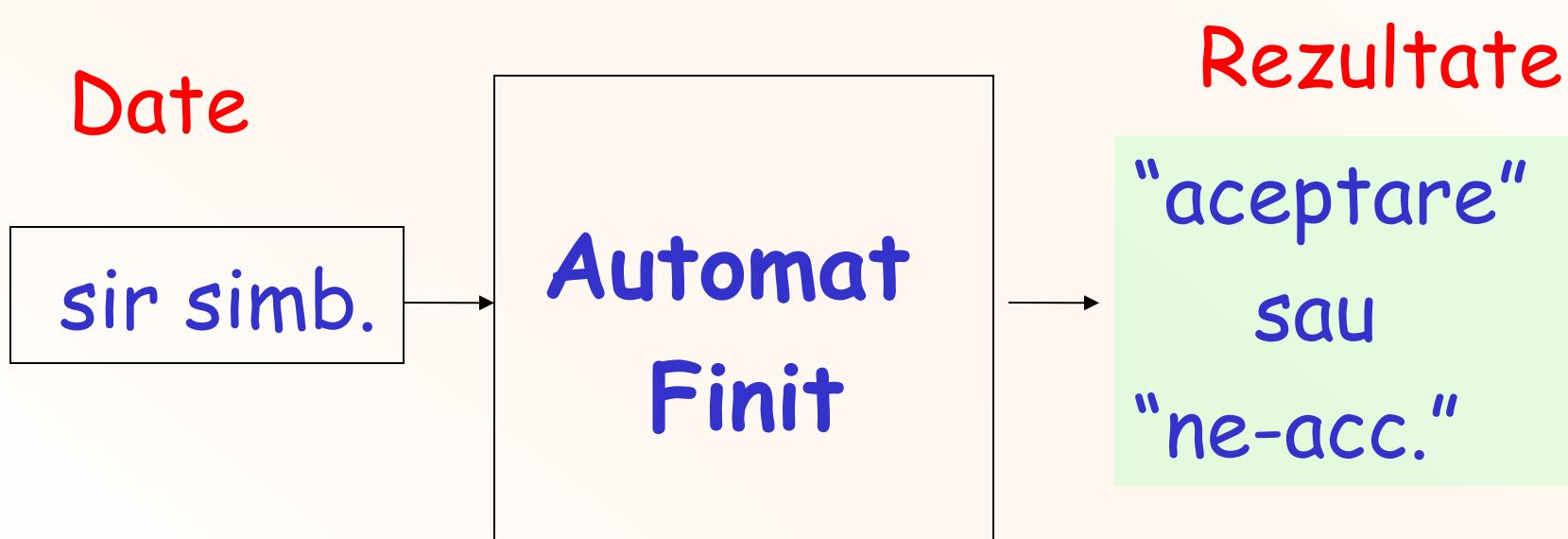
Se construieste cate o TS pt. fiecare bloc

Se construieste o tabela pentru incluziunea  
blocurilor

Cum se modifica  
algoritmul de cautare  
in TS?

nume bloc	parinte	"legatura" TS
A		TS.A
B	1	TS.B
C	2	TS.C
D	2	TS.D
E	1	TS.E

# Automat finit (AF)



# Automat finit: model fizic

banda de intrare



cap  
citire

directie de deplasare



stari

# Automat finit: model matematic

- Un *automat finit* este un ansamblu

$$M = (Q, \Sigma, \delta, q_0, F) :$$

- $Q$  – alfabetul starilor
- $\Sigma$  – alfabet de intrare
- $\delta : Q \times \Sigma \rightarrow P(Q)$  functie de tranzitie
- $q_0 \in Q$  - stare initială
- $F \subseteq Q$  multimea stărilor finale

# AF – reprezentare tabelara

$\delta$		$a_j$		

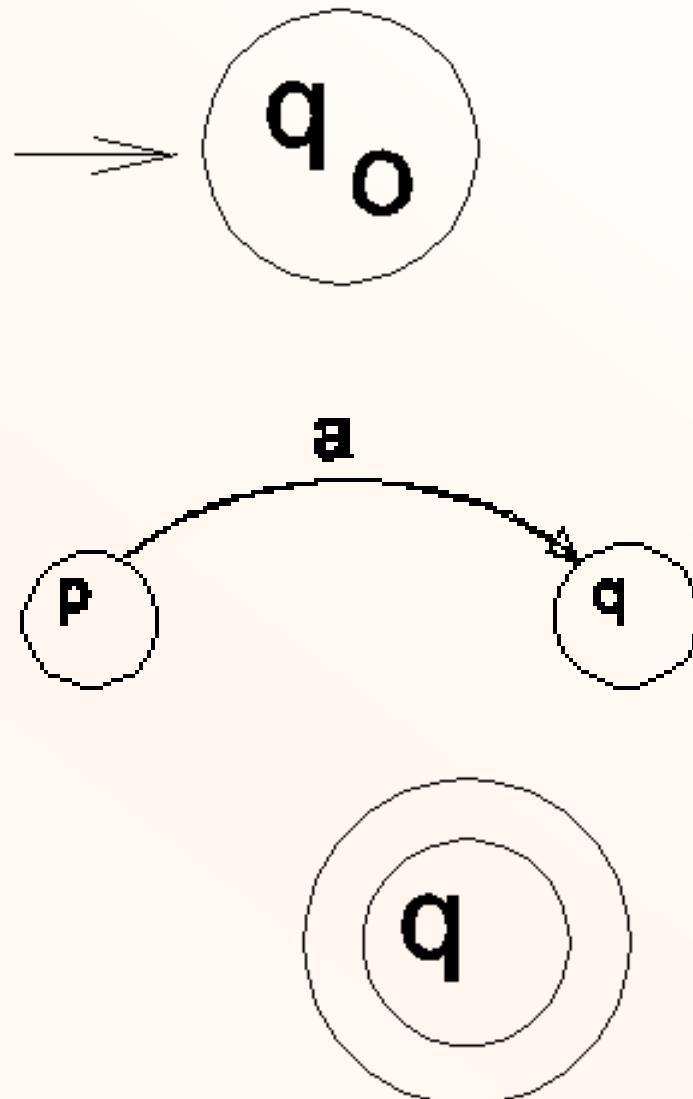
$z_i = \begin{cases} 0 & \text{daca } q_i \text{ nu e stare finala} \\ 1 & \text{daca } q_i \text{ este stare finala} \end{cases}$

# AF reprezentat tabelar; exemplu

$\delta$	0	1	
p	q	p	0
q	r	p	0
r	r	r	1

# AF – reprezentare sub forma de graf

- graf orientat
- cu noduri si arce etichetate
- (graf de tranzitii)



# Configuratii si relatii de tranzitie

$M = (Q, \Sigma, \delta, q_0, F)$ .

configuratie:  $(q, x) \in Q \times \Sigma^*$

tranzitie: element din  $(Q \times \Sigma^*) \times (Q \times \Sigma^*)$

$(p, aw) \vdash (q, w) \Leftrightarrow \delta(p, a) \ni q;$   
 $p, q \in Q, a \in \Sigma, w \in \Sigma^*$

- $\vdash$  tranzitie directa
- $\vdash^k$  k-tranzitie
- $\vdash^+$  +-tranzitie
- $\vdash^*$  \*-tranzitie

# Limbaj acceptat; autom. echivalente

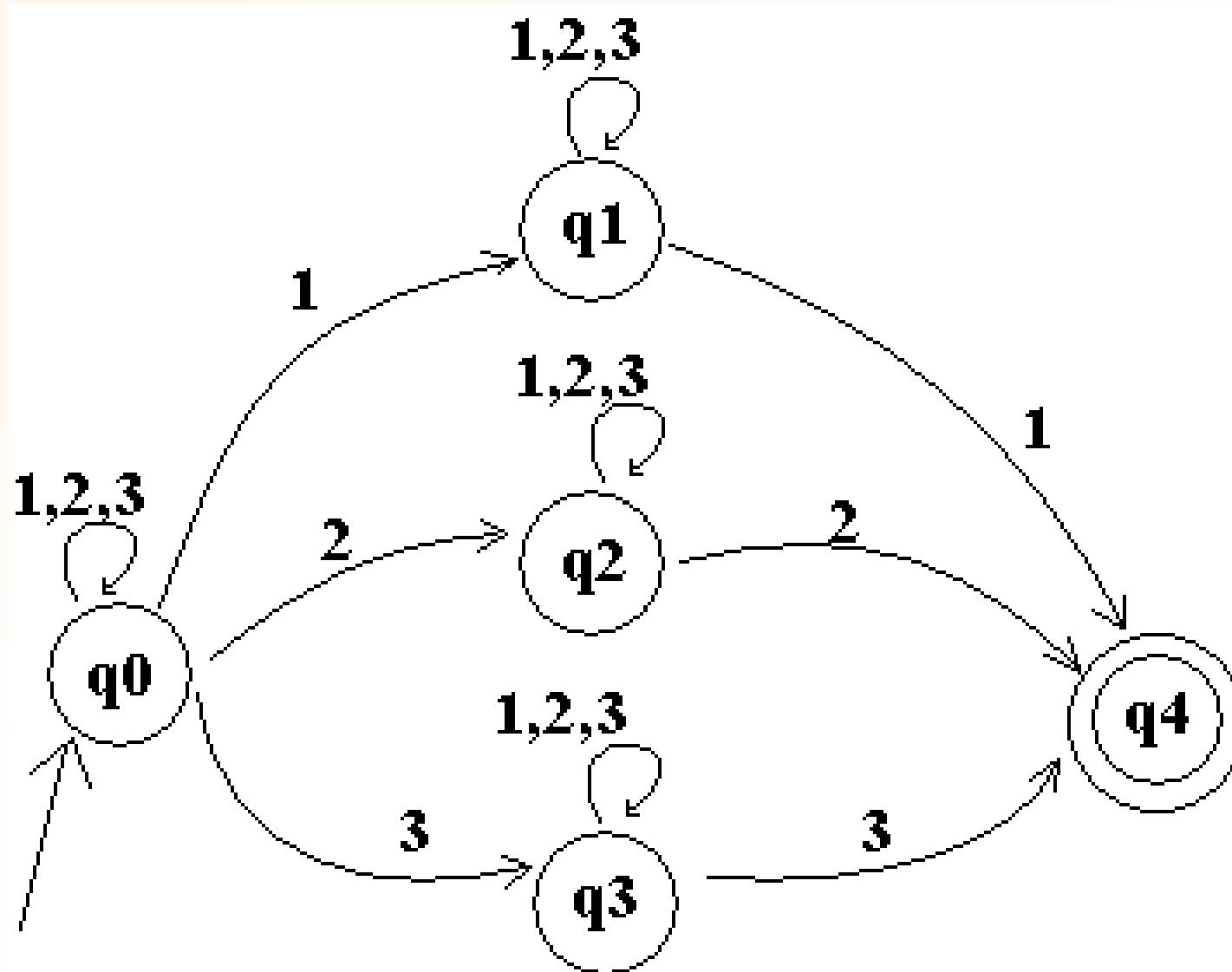
- Limbaj acceptat de automat

$$L(M) = \{w \mid w \in \Sigma^*, (q_0, w) \xrightarrow{*} (q_f, \epsilon), q_f \in F\}$$

- Automate echivalente

$M_1$  echivalent cu  $M_2$  daca:  $L(M_1) = L(M_2)$

# Automat finit - exemplu



# Determinism

- Automat finit determinist (AFD)

$$|\delta(q,a)| \leq 1 \quad \forall q \in Q, a \in \Sigma$$

- Automat finit nedeterminist (AFN)

$$\exists q \in Q, a \in \Sigma \text{ astfel incat } |\delta(q,a)| > 1$$

- Automat finit determinist complet definit

$$|\delta(q,a)| = 1 \quad \forall q \in Q, a \in \Sigma$$

# Echivalenta dintre AFD si AFN

Teorema:

- $\forall M_1 - \text{AFN} \quad \exists M_2 - \text{AFD} \text{ echivalent}$

Constructie (nu demonstratie!):

- Pornim cu:  $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$  – AFN oarecare
- Construim:  $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$  – AFD
  - pe baza lui  $M_1$
  - a.i.  $L(M_1) = L(M_2)$

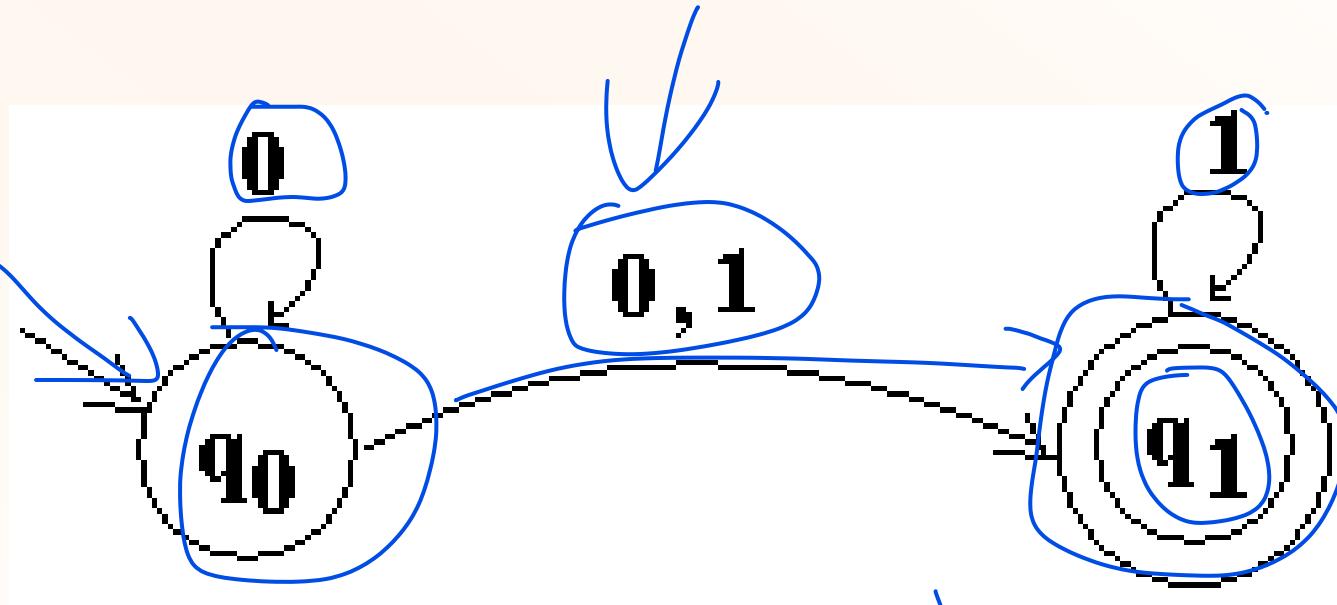
# Teor: $\forall M_1 - \text{AFN} \quad \exists M_2 - \text{AFD}$ echivalent

- $\Sigma_2 = \Sigma_1$
- $Q_2 = \mathcal{P}(Q_1)$
- $q_{02} = \{q_{01}\}$
- $F_2 = \{S \in \mathcal{P}(Q_1) \mid S \cap F_1 \neq \emptyset\}$
- $\delta_2(q, a) = \{r \in Q_1 \mid \exists q_1 \in q \text{ a.i. } r \in \delta_1(q_1, a)\}$

$$= \bigcup_{q_1 \in q} \delta(q_1, a)$$

**$M_2$  – determinist (?)**

# Problema: determinati AFD echiv. pt.



$L = h^0 \dots ^1 \dots$

0 0 0 0 1 1 1

0 1

# AF – stari care nu contribuie la acceptarea unui cuvant

- stare neproductiva – (nu e stare productiva)
- stare inaccesibila – (nu e stare accesibila)
- stare productiva:  $q \in Q$  a.i.  
 $\exists w \in \Sigma^* \text{ si } q_f \in F \text{ a.i. } (q, w) \vdash^*(q_f, \varepsilon)$
- stare accesibila:  $q \in Q$  a.i.  
 $\exists w \in \Sigma^* \text{ a.i. } (q_0, w) \vdash^*(q, \varepsilon)$

# Algoritm determin. starii accesibile

1.  $i := 0$

$A_0 := \{q_0\}$

2. **Repetă**

$i := i + 1$

$A_{i+1} := A_i \cup \{ q \in Q \mid \exists p \in A_i, \exists a \in \Sigma \text{ a.i. } q \in \delta(p, a) \}$

**pană cand**  $A_i = A_{i+1}$

Cu alte cuvinte, adaug toate starile accesibile din starile curente folosind o tranziție cu un simbol din alfabet.

$\{A_i - multimea starilor accesibile\}$

# Algoritm determin. starii productive

1.  $i := 0$

$A_0 := F$

2. Repeta

$i := i + 1$

$A_{i+1} = A_i \cup \{ q \in Q \mid \exists p \in A_i, \exists a \in \Sigma \text{ a.i. } p \in \delta(q, a) \}$

**pană cand**  $A_i = A_{i+1}$

e la fel

$\{A_i - multimea starilor productive\}$

Teorema:

$\forall M_1 - AF$  există  $M_2 - AF$  fără st. neproductive **echiv.**

Constructie (nu demonstratie!):

- Pornim cu:  $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$  – AF oarecare
- determinam  $A$  – multimea starilor productive (algoritmul anterior)
- Construim:  $M_2$  pe baza lui  $M_1$  (a.i.  $L(M_1) = L(M_2)$ )

$$M_2 = (A, \Sigma_1, \delta_{1/A}, q_{01}, F_1)$$
$$L(M_1) = L(M_2) !$$

Teorema:

$\forall M_1 - AF$  există  $M_2 - AF$  fără st. inaccesibile echiv.

Constructie (nu demonstratie!):

- ... analog ...

# ? alta metoda de determinare a AFD echivalent pentru un AFN dat

- $M_1 \Rightarrow ? M_2$

Idea:

1.  $\{q_{01}\} \in Q_2$   
pornim cu  $Q_2 = \{q_{01}\}$
2. adaugam la  $Q_2$  toate submultimile lui  $Q_1$  la care se poate ajunge prin functia de tranzitie, atunci cand se aplica unei stari  $q \in Q_2$  deja adaugata

# AFD complet definit

$M = (Q, \Sigma, \delta, q_0, F)$  (AFD)

- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  functie de tranzitie ;  $|\delta(q,a)| \leq 1$
- ...

Teor:  $\forall$  AFD  $\exists$  AFD complet definit echivalent

Constructie:

AFD  $\Rightarrow$  AFD complet definit:

- adaugam o stare (neproductiva)  $r$  si extindem  $\delta$  astfel:
- $\forall (q,a) \in Q \times \Sigma$  a.i.  $\delta(q,a) = \emptyset$  devine:  $\delta(q,a) = \{r\}$
- $\forall a \in \Sigma$   $\delta(r,a) = \{r\}$

# Minimizarea automatelor finite

Ce vrem:

**Automat determinist cu numar minim de stari !**

**Automat redus**

- AFD
- nu contine stari inaccesibile si neproductive
- nu contine perechi de stari echivalente

# Minimizarea AFD

- automat cu numar minim de stari
  - fara stari –inaccesibile, neproductive
  - mai putine stari ?

ideea: relatie de echivalenta; clase de echivalenta

- stari diferențiate
- stari k diferențiate
- stari echivalente
- stari k-echivalente

# Automatul redus

Fie  $M_1$  – un automat finit oarecare

- Determinam AFD echivalent
- Eliminam starile inaccesibile si neproductive
- Determinam AFD echivalent complet definit

Fie  $M = (Q, \Sigma, \delta, q_0, F)$  automatul rezultat.

- Determinam relatia  $\equiv$  (stari echivalente, clase de echivalenta)
- Pe baza relatiei  $\equiv$  determinam automatul:

$$M_{\equiv} = (Q/\equiv, \Sigma, \delta_{\equiv}, [q_0], F_{\equiv})$$

$Q/\equiv$  - multimea claselor de echivalenta

$$\delta_{\equiv}([q], a) = [\delta(q, a)]$$

$$F_{\equiv} = \{ [q] \mid q \in F \}$$

# Automat redus

- **Teorema**

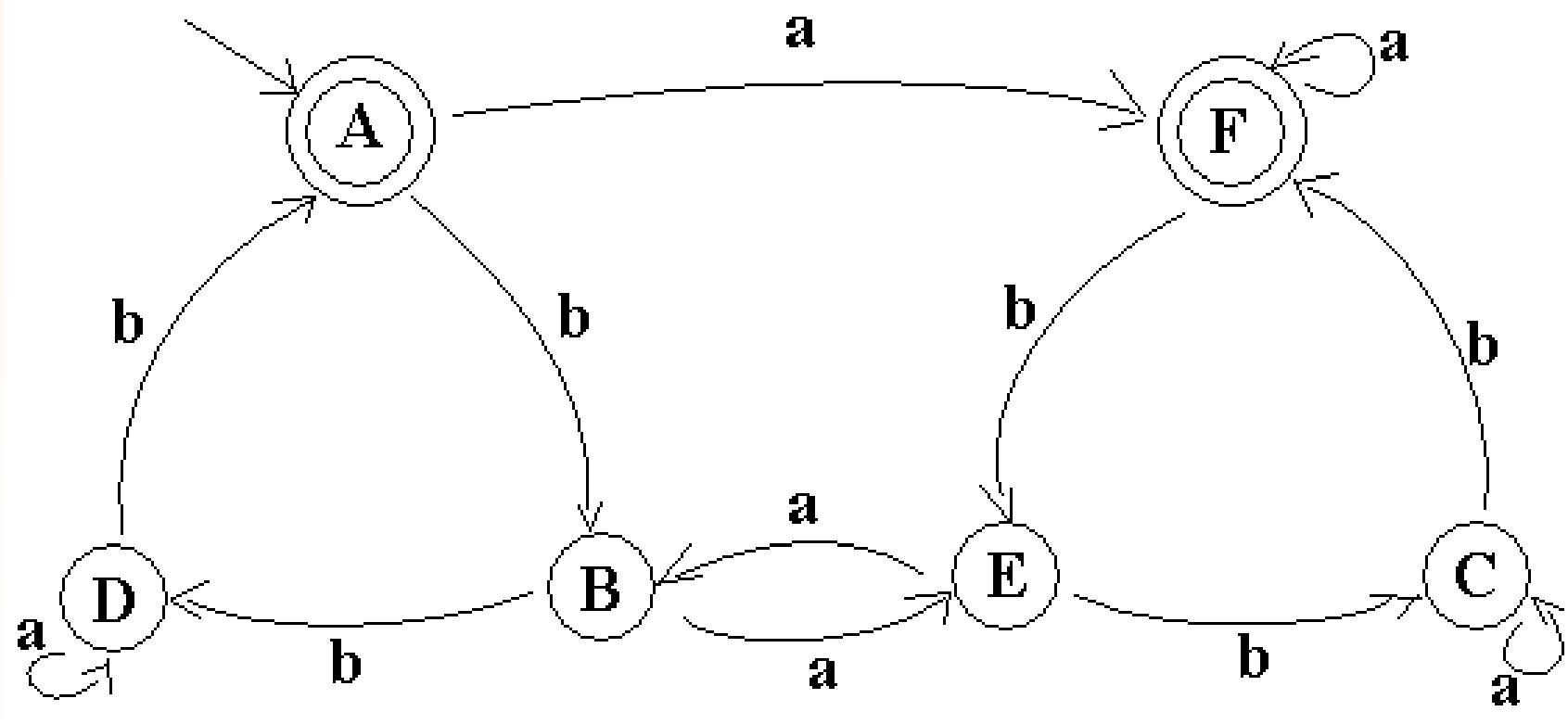
Automatul redus are numar minim de stari

dintre toate AFD echivalente

- **Teorema**

$\forall M_1 - AF \exists M_2 - \text{automat redus echivalent}$

# Determinati clasele de echivalenta ale starilor automatului de mai jos



determinati automatul redus !

# Stari diferențiate

- o alta exprimare

$q_1, q_2$  sunt *stari diferențiate* de  $x \in \Sigma^*$

daca  $\exists q_f \in F$  a.i.  $(q_1, x) \vdash^*(q_f, \varepsilon)$

si nu există nici un  $q \in F$  a.i.  $(q_2, x) \vdash^*(q, \varepsilon)$

sau

daca  $\exists q_f \in F$  a.i.  $(q_2, x) \vdash^*(q_f, \varepsilon)$

si nu există nici un  $q \in F$  a.i.  $(q_1, x) \vdash^*(q, \varepsilon)$

- $x$  (de mai sus) *diferențiază* pe  $q_1$  și  $q_2$

# Stari diferențiate

- o alta exprimare

PP. AFD complet definit

$q_1, q_2$  sunt *stari diferențiate* de  $x \in \Sigma^*$

daca  $\exists r_1, r_2$  astfel incat:  $(q_1, x) \vdash^*(r_1, \varepsilon)$

si  $(q_2, x) \vdash^*(r_2, \varepsilon)$

are loc una dintre:

1.  $r_1 \in F$         si         $r_2 \in Q-F$
2.  $r_1 \in Q-F$         si         $r_2 \in F$

# Relatii intre stari

- **$q_1, q_2$  - stari diferențiate**
  - cf. def. de mai sus
  - ( $\exists x \in \Sigma^*$  care să le diferențieze)
- **stari  $k$  diferențiate**
  - dacă  $\exists x \in \Sigma^*, |x| \leq k$  care să le diferențieze
- **stari echivalente ( $\equiv$ )**
  - dacă nu există  $x \in \Sigma^*$  care să le diferențieze
- **stari  $k$ -echivalente ( $\equiv^k$ )**
  - dacă nu există  $x \in \Sigma^*, |x| \leq k$ , care să le diferențieze

# Proprietati ale rel. de k-echivalenta ( $\equiv^k$ )

- $q_1 \equiv^0 q_2$  dacă ( $q_1, q_2 \in F$ ) sau ( $q_1, q_2 \in Q - F$ )
- $\equiv^0 \supseteq \equiv^1 \supseteq \equiv^2 \supseteq \dots \supseteq \equiv^n \supseteq \dots$
- Dacă  $(\equiv^k) = (\equiv^{k+1})$  atunci  $(\equiv^k) = \equiv$

Lema:

Pt. orice  $M$  există  $n \in \mathbf{N}$  a.i.  $q_1 \equiv^n q_2 \Rightarrow q_1 \equiv q_2$

- ideea: pot avea un nr. finit de relații distincte (max.  $|Q|$ )

# Curs 3

- Gramatici. Clasificarea Chomsky
  - Gramatici regulare
  - Limbaje regulare. Echivalente
    - Expresii regulare
- Proprietati de inchidere ale limbajelor regulare

# Gramatica

O gramatica este un cvadruplu  $\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$

- $\mathbf{N}$  este un alfabet de simboluri **neterminale**
  - $\Sigma$  este un alfabet de simboluri **terminale**
  - $\mathbf{N} \cap \Sigma = \emptyset$
  - $P \subseteq (\mathbf{N} \cup \Sigma)^* \mathbf{N} (\mathbf{N} \cup \Sigma)^* \times (\mathbf{N} \cup \Sigma)^*$
- $P$  multime finită (multimea regulilor de producție)
- $S \in \mathbf{N}$  (simbolul de start - simbolul initial)

Notatie:

$(\alpha, \beta) \in P$  se noteaza:  $\alpha \rightarrow \beta$   
( $\alpha$  se înlocuieste cu  $\beta$ )

# Notatii

- la nivel abstract (exemple matematice, specificari)
  - $\Sigma$ : a,b,... litere mici de la inceputul alfabetului
  - $N$ : A,B,.. litere mari de la inceputul alfabetului
  - $\Sigma$  sau  $N$ : X,Y,...litere mari de la sfarsitul alfabetului
  - $\Sigma^*$  : x,y,... litere mici de la sfarsitul alfabetului
  - $(\Sigma \cup N)^*$  :  $\alpha, \beta, \dots$  litere grecesti
- nu se folosesc spatii cand avem nevoie de mai multe caractere pentru a specifica un simbol (terminal sau neterminat)

# Relatii de derivare

relatii binare peste  $(\Sigma \cup N)^*$  adica  $(\Sigma \cup N)^* \times (\Sigma \cup N)^*$

- derivare directă

$\gamma \Rightarrow \delta \Leftrightarrow \exists \gamma_1, \gamma_2, \alpha, \beta \in (N \cup \Sigma)^*$

a.i.  $\gamma = \gamma_1 \alpha \gamma_2$ ,  $\delta = \gamma_1 \beta \gamma_2$ , iar  $(\alpha \rightarrow \beta) \in P$

- k-derivare

$\stackrel{k}{\Rightarrow}$

(o succesiune de  $k$  derivări directe)

- + derivare

$\stackrel{+}{\Rightarrow}$

dacă  $\exists k > 0$  a.i. cele 2 secvențe să fie într-o relație de "k derivare"

- \* derivare

$\stackrel{*}{\Rightarrow}$

dacă fie cele 2 secvențe sunt egale, fie între ele există o relație de +derivare

# Limbaj generat de o gramatica

- Limbaj generat gramatica  $G=(N, \Sigma, P, S)$

$$L(G) = \{ w \in \Sigma^* \mid S \xrightarrow{*} w \}$$

- Forma propositională

$$- \alpha \in (N \cup \Sigma)^* \text{ a.i. } S \xrightarrow{*} \alpha$$

- Propozitie (cuvant)

$$- \text{un element din } L(G)$$

- Gramatici echivalente

daca genereaza acelasi limbaj

# Tipuri de gramatici

- Gramatica monotonă
  - $\forall \alpha \rightarrow \beta \in P: |\alpha| \leq |\beta|$   $\alpha, \beta \in (N \cup \Sigma)^*$
  - caz special:  $S \rightarrow \epsilon$  poate să aparțină lui P. În acest caz S nu apare în membrul drept al nici unei reguli de producție.
- Gramatica dependenta de context  
reguli de producție sunt de forma:
$$\alpha A \beta \rightarrow \alpha \gamma \beta \quad A \in N$$
$$\alpha, \beta, \gamma \in (N \cup \Sigma)^*, \gamma \neq \epsilon$$
  - caz special:  $S \rightarrow \epsilon$  poate să aparțină lui P. În acest caz S nu apare în membrul drept al nici unei reguli de producție.

# Tipuri de gramatici

- **Gramatica regulară:**

reg. prod. sunt de forma

- $A \rightarrow aB$
- $A \rightarrow b$

unde  $A, B \in N$  și  $a, b \in \Sigma$

caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . În acest caz  $S$  nu apare în membrul drept al nici unei reguli de producție.

- **Gramatica independentă de context:**

reg. producție sunt de forma  $A \rightarrow \alpha$ ,  $A \in N$ ,  $\alpha \in (N \cup \Sigma)^*$

# Clasificarea Chomsky

- Gramatici de tip **0**  
nici o restrictie (*suplimentara*) referitoare la forma regulilor de productie
- Gramaticile de tip **1**  
*dependente de context*  $\Leftrightarrow$  *gramatici monotone*  
( *monotonic, non-contracting* )
- Gramaticile de tip **2**  
*gramatici independente de context*
- Gramaticile de tip **3**  
*gramatici regulate*  
=> Limbaje independente de context  
=> Limbaje regulate

# Ierarhia Chomsky

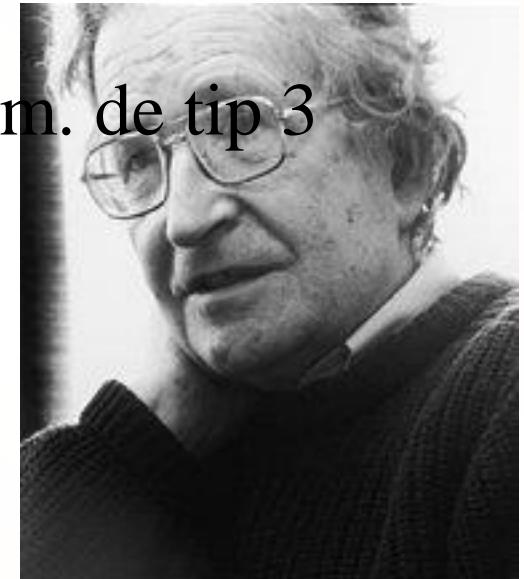
Fie

~ 1959-1963

- $\mathcal{L}_0$  - multimea limbajelor generate de gram. de tip 0
- $\mathcal{L}_1$  - multimea limbajelor generate de gram. de tip 1
- $\mathcal{L}_2$  - multimea limbajelor generate de gram. de tip 2
- $\mathcal{L}_3$  - multimea limbajelor generate de gram. de tip 3

Are loc:

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$$



# Ierarhia Chomsky: observatii

Teorema:

Fiecare dintre familiile de limbaje:

$$\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$$

este inchisa fata de operatia de reuniiune

# Limbaje regulare. Echivalente

- Limbaj regular
  - = limbaj generat de o gramatica regulara
- putere de exprimare
  - AF: AFN  $\leftrightarrow$  AFD
  - AF  $\leftrightarrow$  gr.regulare
  - AF  $\leftrightarrow$  (m.regulare  $\leftrightarrow$  expr.reg.)

# Multimi regulare

Fie  $\Sigma$  un alfabet.

Multimile regulare peste  $\Sigma$  se definesc recursiv astfel:

1.  $\Phi$  este o m. reg. peste  $\Sigma$
2.  $\{\varepsilon\}$
3.  $\{a\}$  daca:  $a \in \Sigma$
4.  $RUS$  daca  $R, S$  – multimi regulare peste  $\Sigma$  +
5.  $RS$  daca  $R, S$  – multimi regulare peste  $\Sigma$
6.  $R^*$  daca  $R$  – multime regulara peste  $\Sigma$
7. Orice alta multime regulara se obtine aplicand de un numar finit de ori reg. 1-6

# Multimi regulare si expresii regulare

- **Expresii regulare**

1.	$\Phi$	expr. reg. coresp. m.reg. $\Phi$	
2.	$\epsilon$		$\{\epsilon\}$
3.	a	daca: $a \in \Sigma$	$\{a\}$
4.	$r+s$	daca $r,s$ – expresii regulare	$r   s$
5.	$rs$	daca $r,s$ – expresii regulare	RS
6.	$r^*$	daca $r$ – expresie regulara	$R^*$
7.	Orice alta expr. reg. se obtine aplicand de un numar finit de ori reg. 1-6		

- **Expresii regulare echivalente:**

- mult. regulare reprezentate de acestea sunt egale

# Expresii regulare

- expresiile regulare – secv. obtinute prin concatenarea de simb. din

$$\Sigma \cup \{\Phi, \varepsilon, +, *, (, )\} \quad (\dots \text{prioritate} \dots)$$

- multimile regulare asociate expresiilor regulare sunt limbaje regulare

Deci: *Orice expresie regulară peste  $\Sigma$  descrie un limbaj regular peste  $\Sigma$*

# Proprietati de inchidere ale limbajelor regulare

**Teorema:**

*Daca*

$L_1, L_2$  sunt limbaje regulare peste alfabetul  $\Sigma$

*atunci:*

$L_1 \cup L_2, L_1 \cap L_2, L_1 L_2, L_1^*, \text{complement}(L_1)$

sunt limbaje regulare peste alfabetul  $\Sigma$

# Lema de pompare pt. limbaje regulare

- Daca  $L$  este un limbaj regular,
- atunci  $\exists p \in \mathbb{N}^*$  ( fix pt. un limbaj dat )  
(oricat de mare)
- astfel incat:  
 $\forall w \in L$  de lungime cel putin  $p$   
exista o descompunere de forma  $w=xyz$ ,  
unde  $0 < |y| \leq p$   
cu proprietatea ca:  $xy^iz \in L, \forall i \in \mathbb{N}$

# Lema de pompare pt. limbaje regulare

$(\forall L \subseteq \Sigma^*)$

$(\text{regular}(L) \Rightarrow$

$((\exists p \geq 1)((\forall w \in L)((|w| \geq p) \Rightarrow$

$((\exists x, y, z \in \Sigma^*)(w = xyz \wedge (|y| \geq 1 \wedge |y| \leq p \wedge (\forall n \geq 0)(xy^n z \in L)))))))$

(enunt formal al teoremei)

# Lema de pompare pt. limbaje regulare

(o alta versiune, mai “puternica”)

Daca  $L$  este un limbaj regular,

- atunci  $\exists p \in \mathbb{N}^*$  ( fix pt. un limbaj dat )  
(oricat de mare)
- astfel incat:  
 $\forall w \in L$  de lungime cel putin  $p$   
exista o descompunere de forma  $w=xyz$  astfel incat
  - $0 < |y|$
  - $|x y| \leq p$
  - $xy^i z \in L, \forall i \in \mathbb{N}$

# Lema de pompare pt. limbaje regulare

Observatii:

- Lema da o conditie necesara dar nu suficiente
- daca un limbaj satisface conditiile lemei nu inseamna ca este regular
- folosim negatia lemei de pompare pt. a dem. ca un limbaj nu este regular

# Lema de pompare pt. limbaje regulare

De ce se intampla asa:

- Daca  $L$  – limb. reg.

$\Rightarrow$  exista  $G$  – gram. reg. a.i.  $L(G) = L$  (def.)

$\Rightarrow$  exista  $M$  – AF a.i.  $L(M) = L$  (teorema)

- Fie  $p$  – nr. de stari ale lui  $M$
- daca  $|w| \geq p$  si  $w$  – acceptat

$\Rightarrow \exists$  un drum in graful asociat lui  $M$  a.i. etichetele arcelor sunt simboluri din  $w$

$\Rightarrow$  drumul este de lungimea  $p$ ; adica trece prin  $p + 1$  noduri din graf

$\Rightarrow \exists$  un nod prin care se trece de cel putin 2 ori

$\Rightarrow$  ciclu/bucla – care se poate repeta de oricate ori !!

$\Rightarrow$  se poate repeta sirul etichetelor arcelor din bucla !!

(de 0 sau mai multe ori)

Exemplu:

Fie  $L$  - limbajul regular corespunzator expresiei regulare:

$aa^*b^*$

1) fie  $w = ab$  ;

Puteti identifica o descompunere  $w=xyz$  a.i.  $xy^iz$  in  $L$  ?

2) fie  $w = aa$  ;

Puteti identifica o descompunere  $w=xyz$  a.i.  $xy^iz$  in  $L$  ?

---

Analog pt.:  $a(ba)^*$

si  $w = aba$

---

Analog pt.:  $L=\{a,b\}$  si  $w = a$

# Proprietati de inchidere ale limbajelor regulate

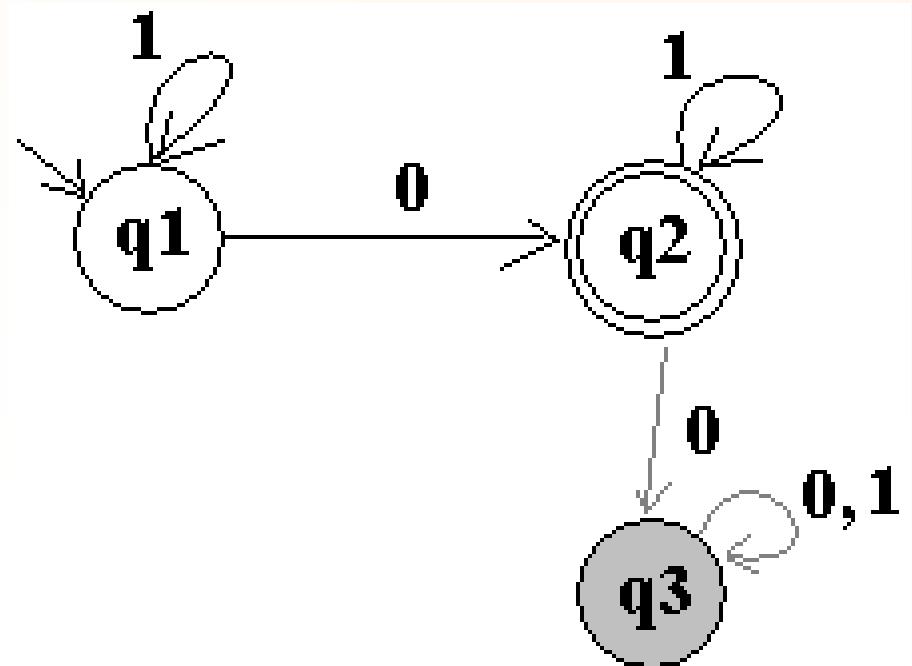
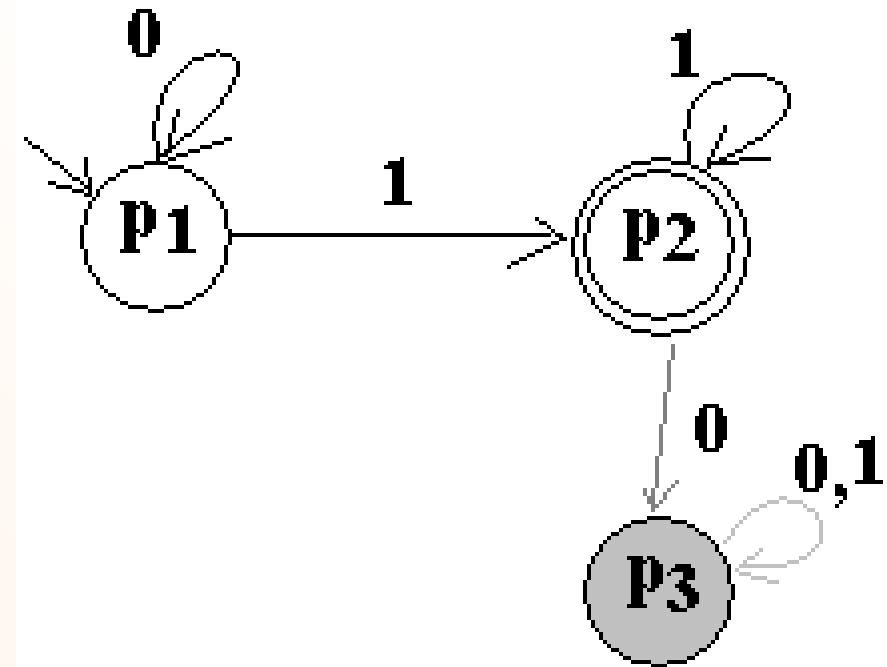
$L_1 \cap L_2$

- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
- $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- ?  $M = (Q, \Sigma, \delta, q_0, F)$

PP. ca aut.  $M_1$  si  $M_2$  sunt deterministe, complet definite !

(alg. de constr. !!)

- $M = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times F_2)$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$



# Proprietati de inchidere ale limbajelor regulate

complement( $L_1$ )

- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
- $?M = (Q, \Sigma, \delta, q_0, F)$

PP. ca aut.  $M_1$  este determinist complet definit !  
(alg. de constr.)

- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, Q_1 - F_1)$

# Curs 4

- AF cu  $\epsilon$ -miscari
- Expresii regulare - proprietati
- Legatura dintre AF si expresiile regulate
- Lex/flex - generator de analizor lexical

Lect. Dr. Lupsa Dana  
Universitatea Babes - Bolyai  
Facultatea de Matematica si Informatica  
2023 - 2024

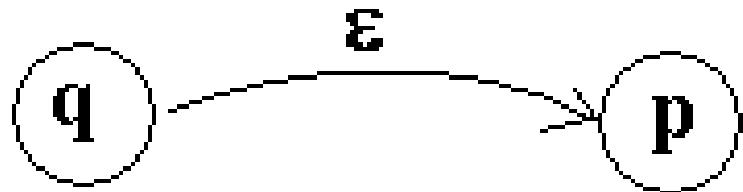
# Automate finite cu $\varepsilon$ -miscari

$M = (Q, \Sigma, \delta, q_0, F)$  :

...

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$  functia de tranzitie

Ideeă: putem avea si  $\varepsilon$ -tranzitii  
(automate cu  $\varepsilon$ -tranzitii)



Teorema:

Pentru orice automat finit cu  $\varepsilon$ -miscari  
exista un automat finit echivalent.

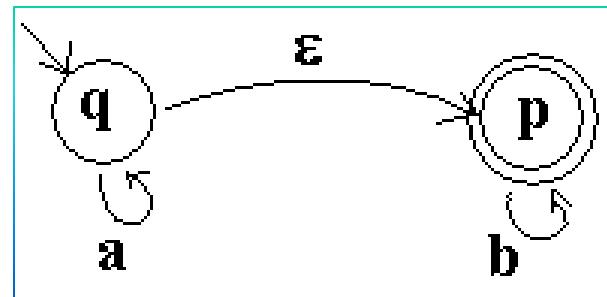
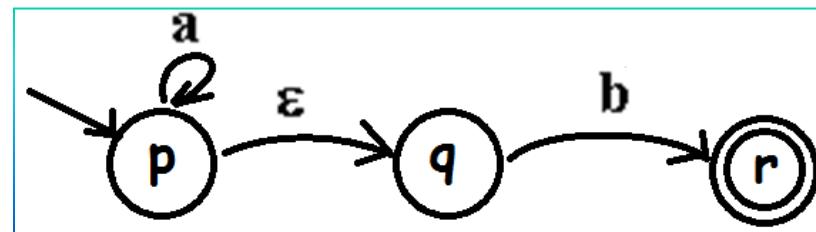
Obs. Conform definitiei pe care am dat-o (in cursul 2),  
automatele finite sunt fara  $\varepsilon$ -miscari

# Automate finite cu $\varepsilon$ -miscari

$M = (Q, \Sigma, \delta, q_0, F)$  :

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$  functie de tranzitie

*Care este  
automatul finit  
echivalent?*



# Echivalenta dintre expresiile

## regulare si limbajele acceptate de AF

Teorema:

Daca  $r$  este o expresie regulara, atunci exista un AF care accepta multimea secentelor reprezentate de aceasta expresie (multimea regulara). Si reciproc.

- Echivalenta:
  - constructia automatului echivalent pentru fiecare dintre constructiile de mai sus (nu vom face dem.)
  - constructia expresiei regulare ce descrie limbajul acceptat de un automat (nu vom face dem.)(→ ~ seminar)

# Expresie regulara

## => limbaj acceptat de AF

- **Expresii regulare**

- $\emptyset$
- $\epsilon$
- $a$       daca:  $a \in \Sigma$
- $r+s$      daca  $r,s$  – expresii regulare
- $rs$         daca  $r,s$  – expresii regulare
- $r^*$         daca  $r$  – expresie regulara

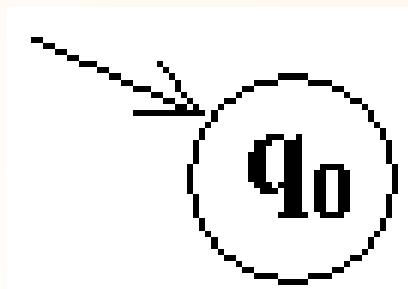


- **Constructia automatului echivalent**

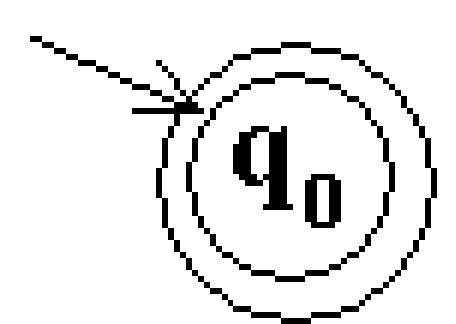
pentru fiecare dintre constructiile de mai sus

# Expresie regulara=> limbaj acceptat de AF

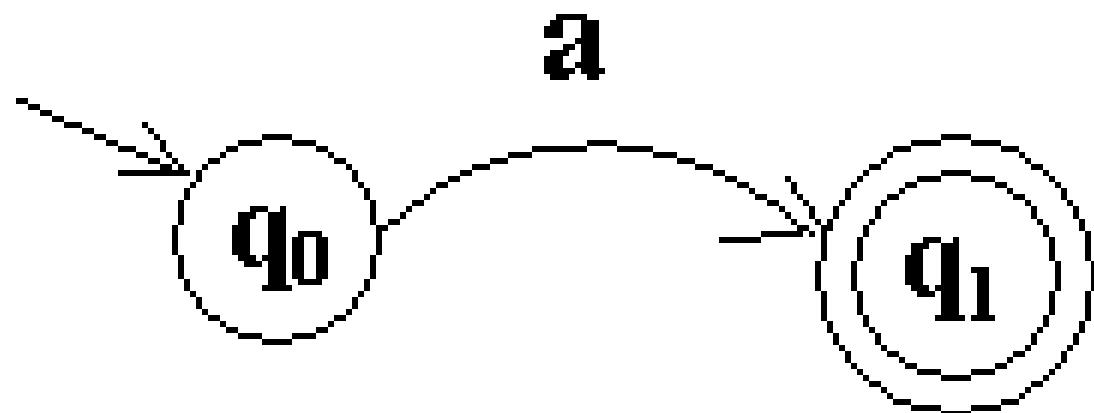
- Automatul ce accepta:  $\Phi$



- Automatul ce accepta:  $\epsilon$

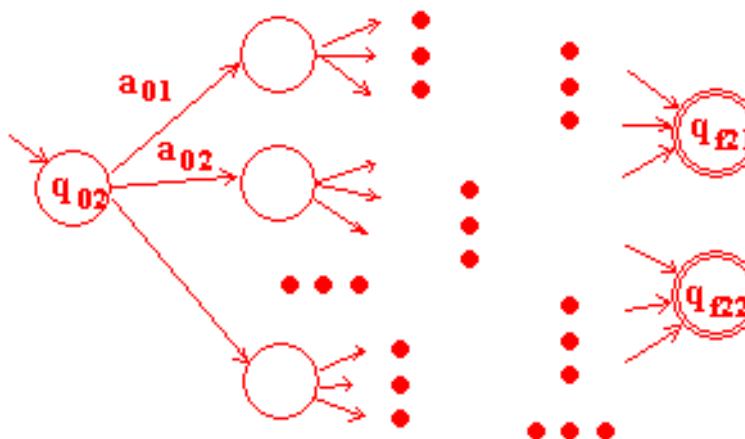
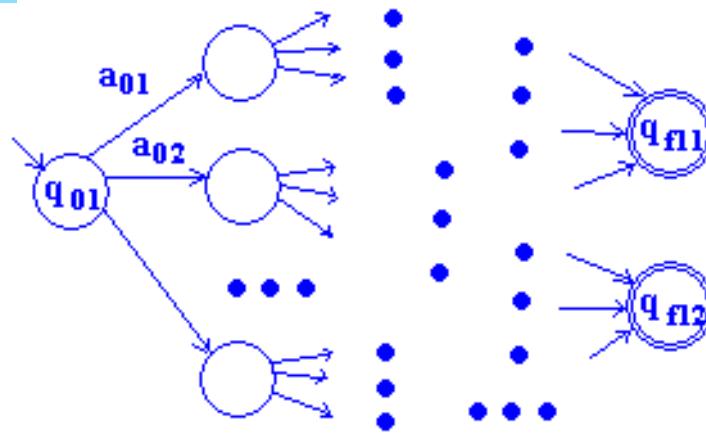


- Automatul ce accepta:  $a$  (daca:  $a \in \Sigma$ )



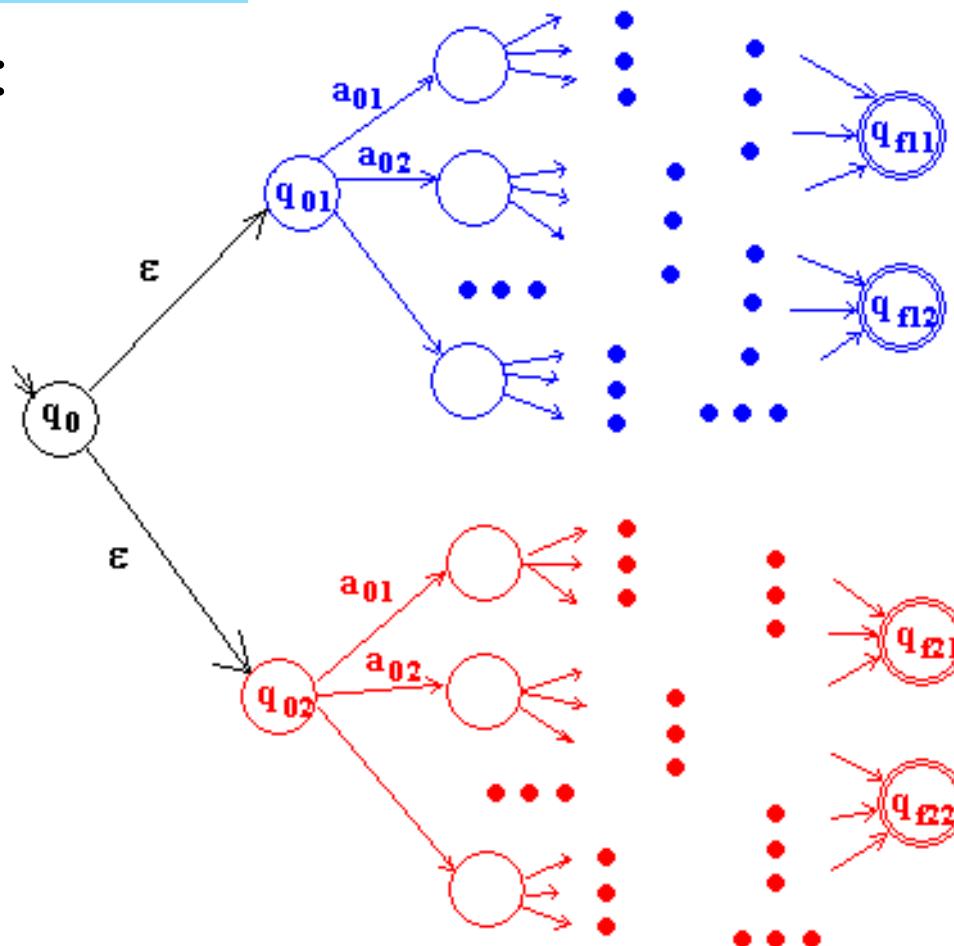
- Automatul ce acceptă reuniunea limbajelor acceptate de două automate date

– se dau:



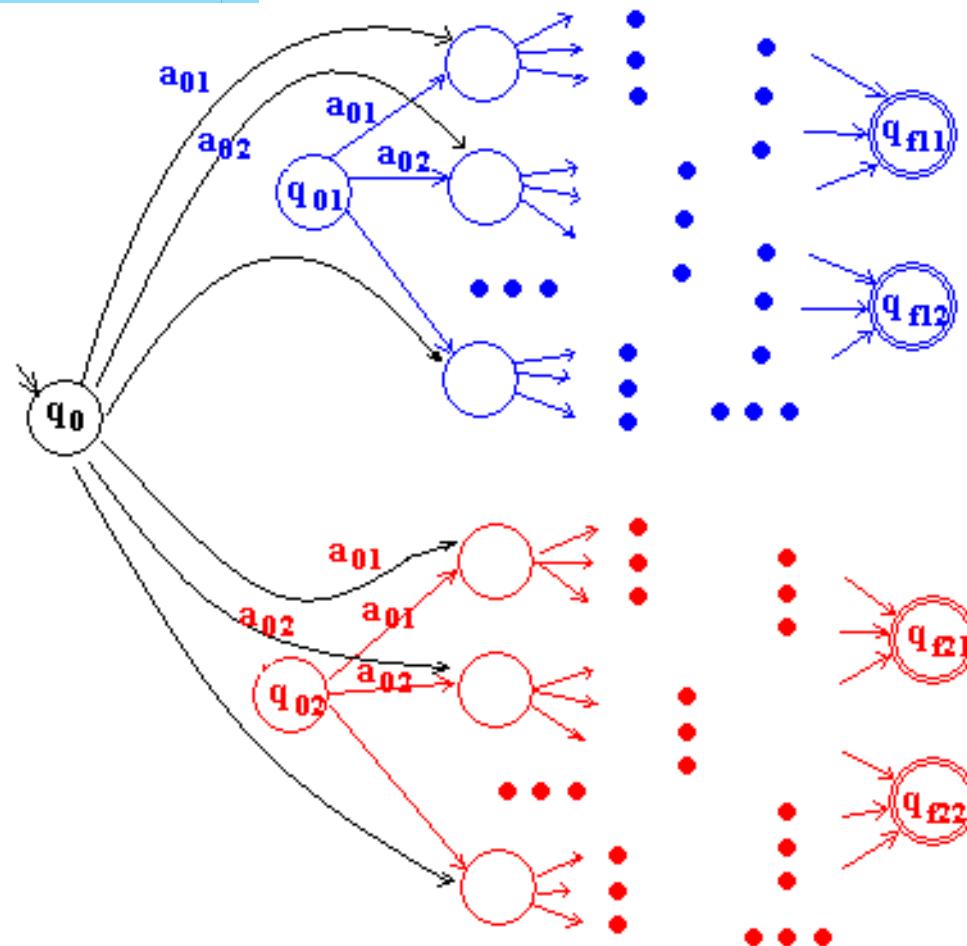
- Automatul ce acceptă reuniunea limbajelor acceptate de două automate date

– AF cu  $\epsilon$  tranz.:



- Automatul ce acceptă reuniunea limbajelor acceptate de două automate date

– AF

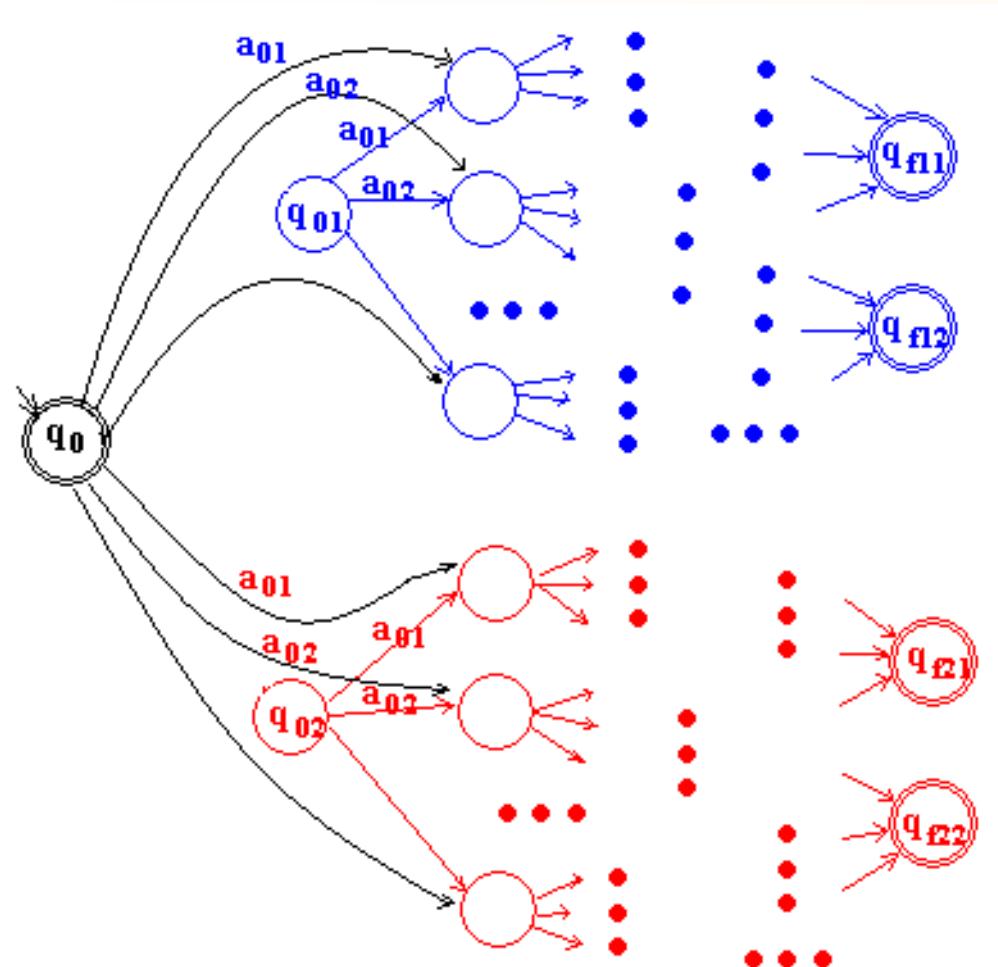


??! cel putin una  
dintre  $q_{01}$  sau  $q_{02}$   
e stare finala

- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date

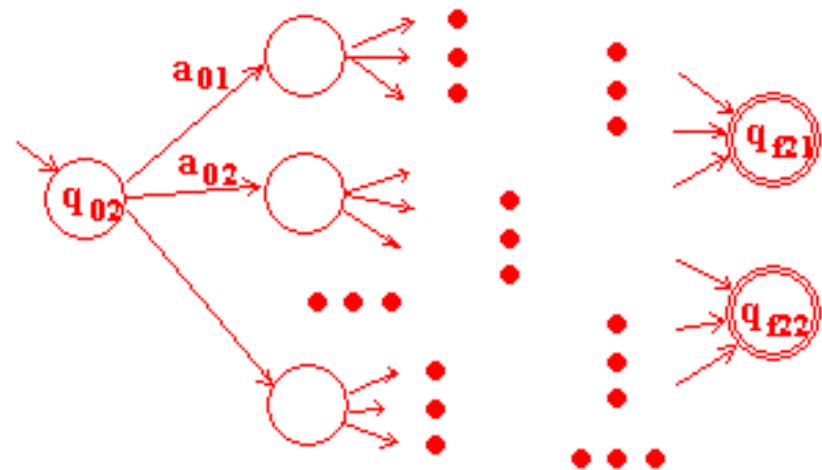
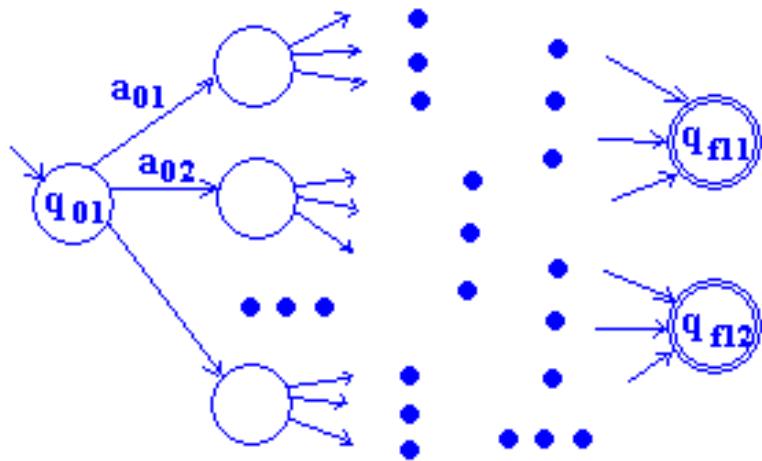
– AF

**Daca cel putin una  
dintre  $q_{01}$  sau  $q_{02}$   
este stare finala**

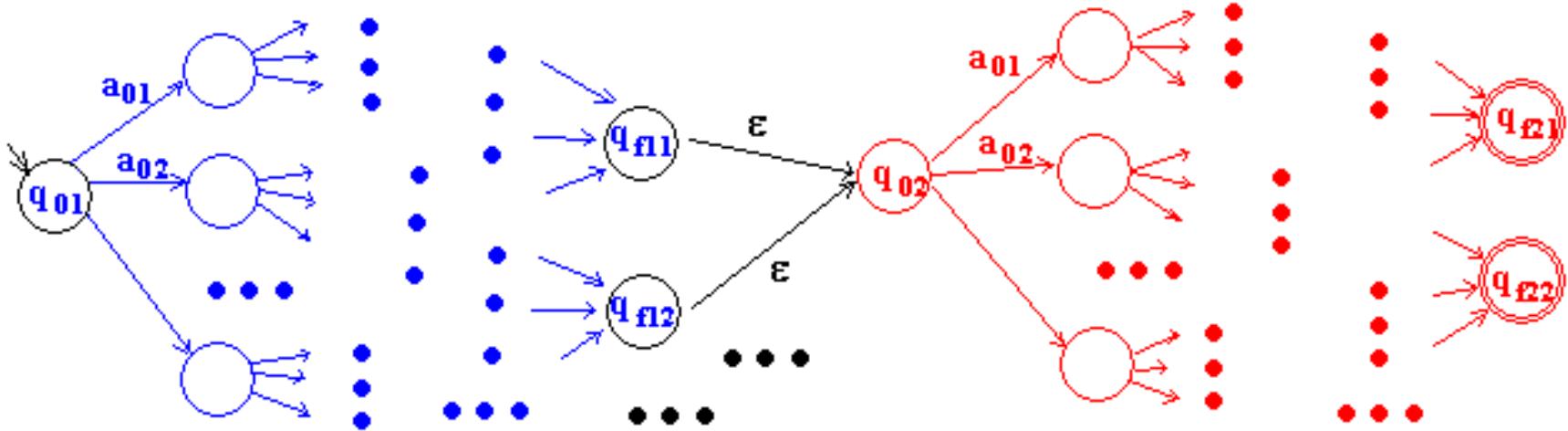


- Automatul ce acceptă concatenarea limbajelor acceptate de două automate date

– se dau

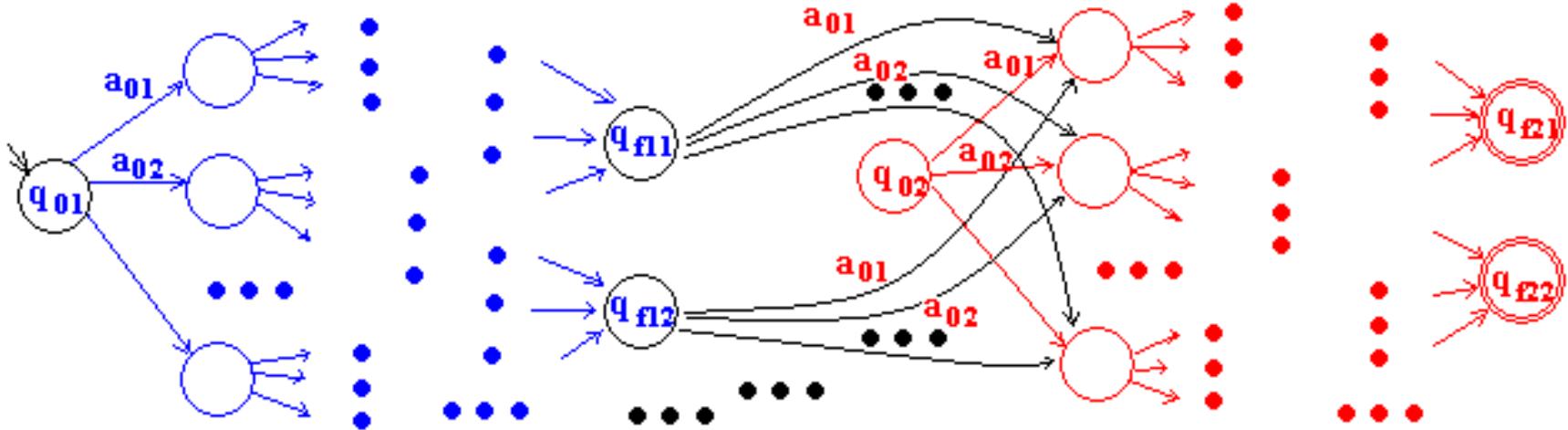


- Automatul ce acceptă concatenarea limbajelor acceptate de două automate date
  - AF cu  $\epsilon$  tranz.:



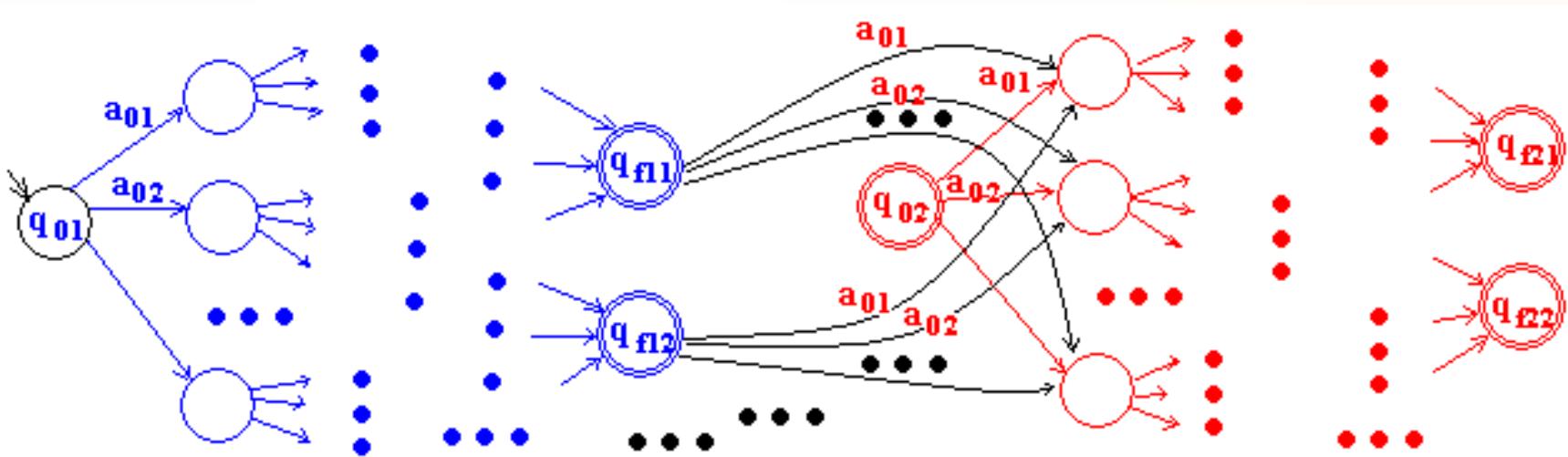
- Automatul ce acceptă concatenarea limbajelor acceptate de două automate date

– AF



## • Automatul care acceptă concatenarea limbajelor acceptate de două automate date

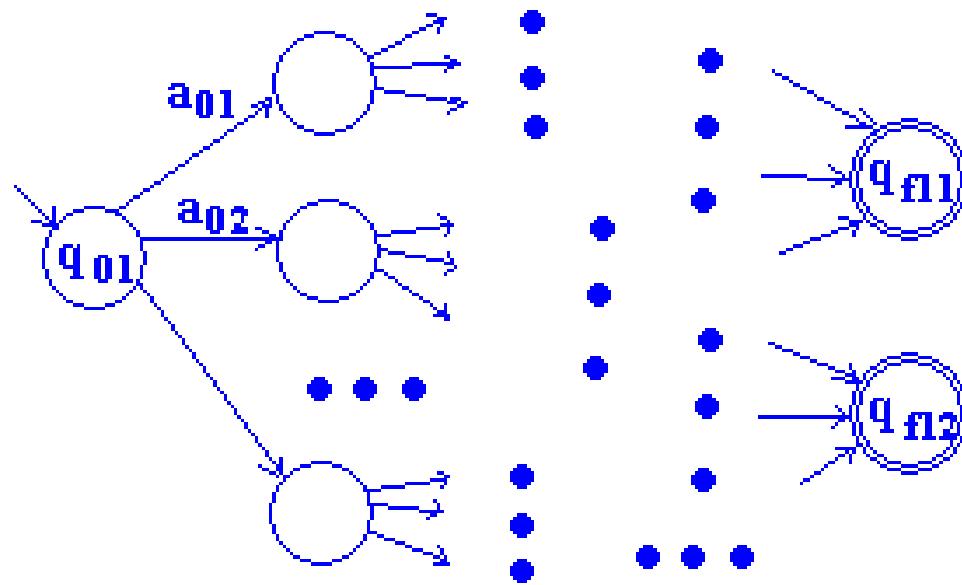
– AF



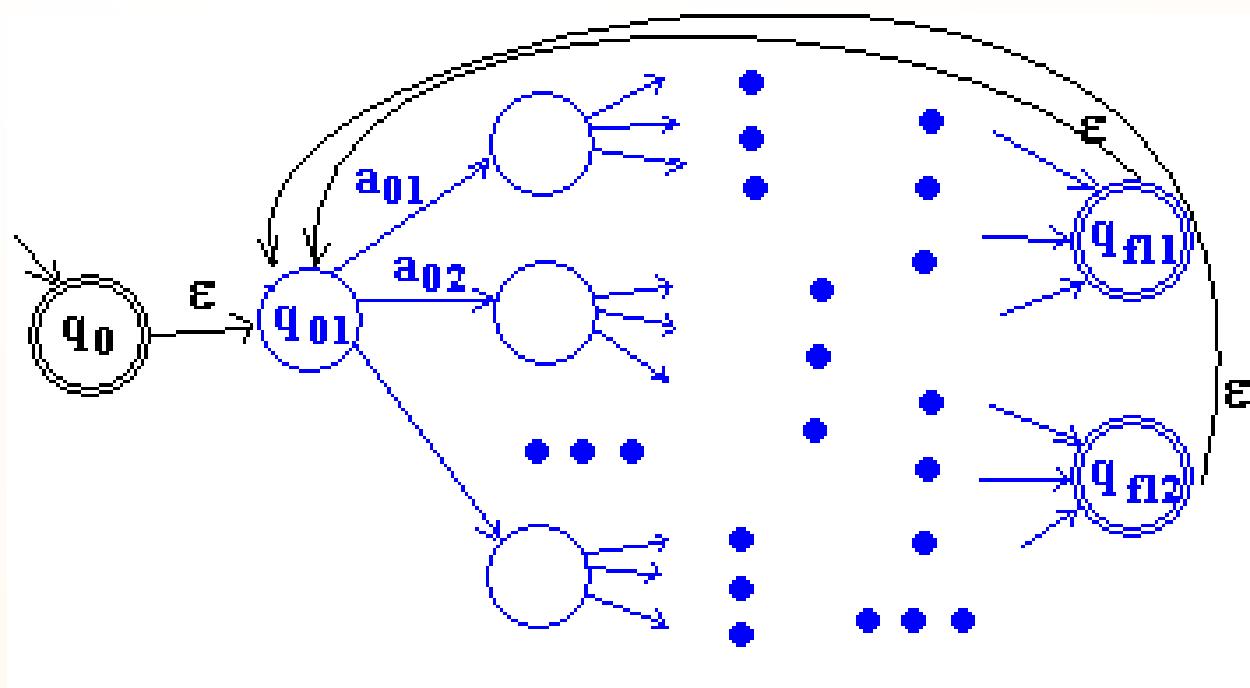
Daca  $q_{02}$  stare finala

- Automatul ce acceptă orice secvență peste limbajul acceptat de un automat dat

– se da:

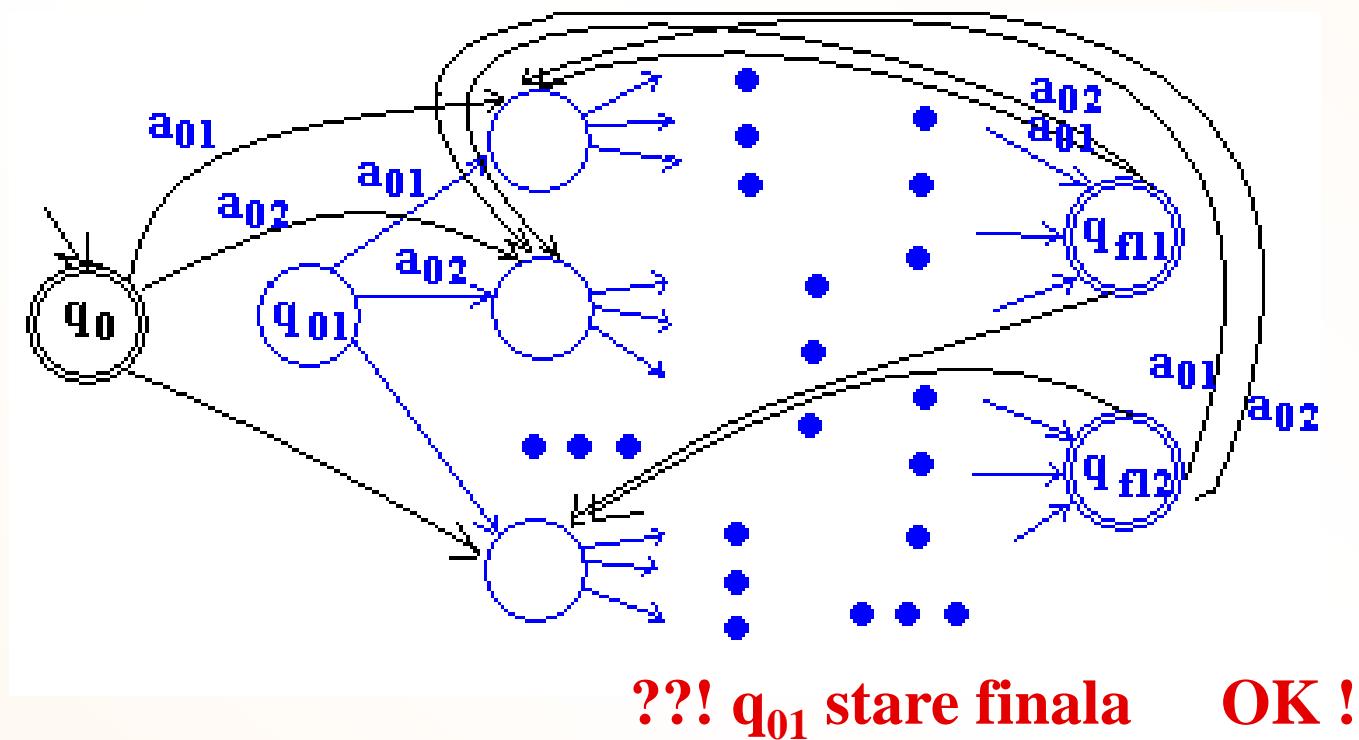


- Automatul ce acceptă orice secvență peste limbajul acceptat de un automat dat
  - AF cu  $\epsilon$  tranz.:



- Automatul ce accepta orice secventa peste limbajul acceptat de un automat dat

– AF:



# Exercitii:

## Expresie regulara=> limbaj acceptat de AF

- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date
  - se considera AF pt.:  
 $aa^*$   
 $bb^*$
  - se considera AF pt.:  
 $a^{2n}$                      $n$  – nr. natural;  $n \geq 0$   
 $b^{2n+1}$                      $n$  – nr. natural;  $n \geq 0$

## Exercitii:

### Expresie regulara=> limbaj acceptat de AF

Construiti automatul ce accepta concatenarea  
limbajelor acceptate de doua automate date.

a) Se considera AF pt.:      a      si      b

b) Se considera AF pt.:      a\*      si      b\*

c) Se considera AF pt.:      a      si      b\*

## Exercitii:

### Expresie regulara=> limbaj acceptat de AF

- Automatul ce accepta orice secventa peste limbajul acceptat de un automat dat
  - se considera AF pt.: a

# Multimi regulare

Fie  $\Sigma$  un alfabet.

Multimile regulare peste  $\Sigma$  se definesc recursiv astfel:

1.  $\Phi$  - multime reg. peste  $\Sigma$
2.  $\{\varepsilon\}$  ...
3.  $\{a\}$  daca:  $a \in \Sigma$
4.  $RUS$  daca  $R, S$  – multimile regulare peste  $\Sigma$  +
5.  $RS$  daca  $R, S$  – multimile regulare peste  $\Sigma$
6.  $R^*$  daca  $R$  – multime regulara peste  $\Sigma$
7. Orice alta multime regulara se obtine aplicand de un numar finit de ori reg. 1-6

# Multimi regulare si expresii regulate

- Expresii regulate

1.	$\Phi$	expr. reg. coresp. m.reg.	$\Phi$
2.	$\epsilon$		$\{\epsilon\}$
3.	a	daca: $a \in \Sigma$	$\{a\}$
4.	$r+s$	daca $r,s$ – expresii regulate	$r   s$
5.	$rs$	daca $r,s$ – expresii regulate	RS
6.	$r^*$	daca $r$ – expresie regulara	$R^*$
7.	Orice alta expr. reg. se obtine aplicand de un numar finit de ori reg. 1-6		

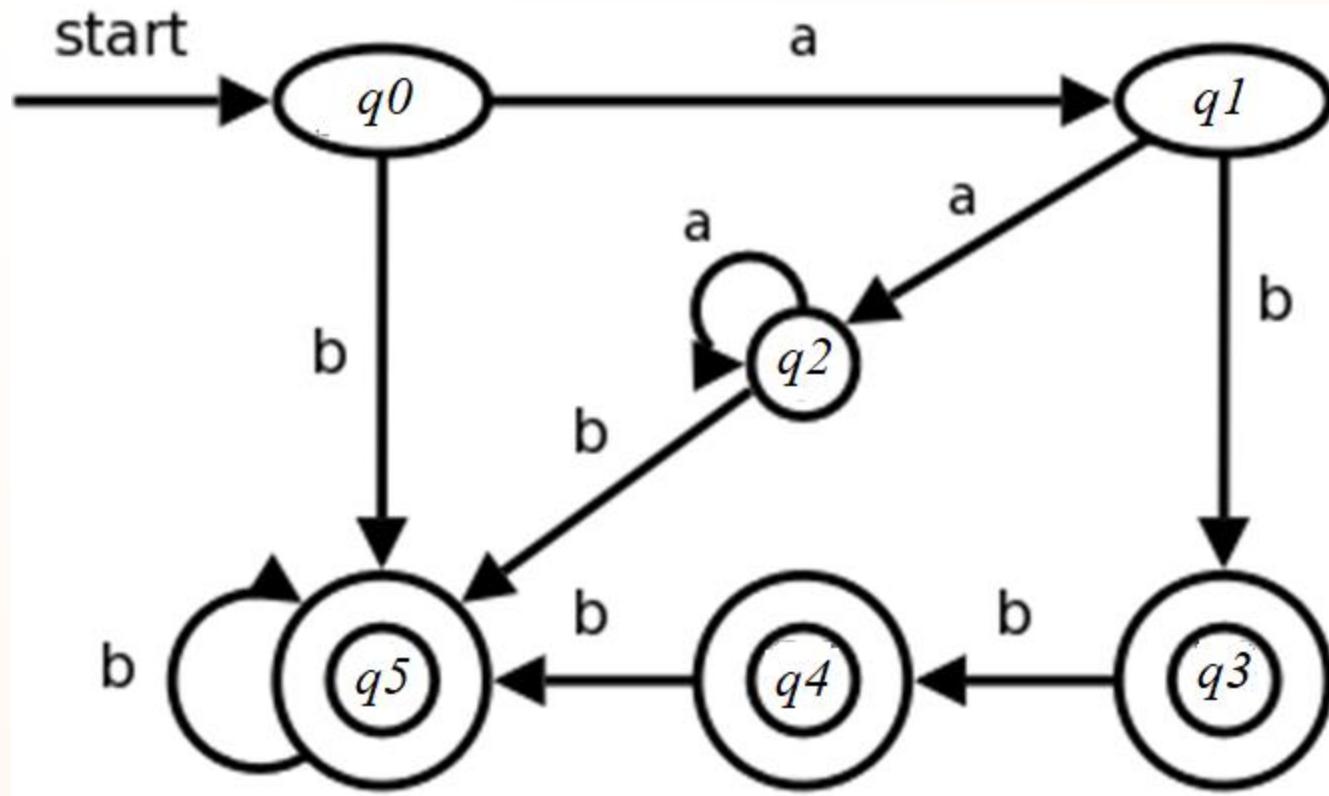
- Expresii regulate echivalente:

- mult. regulate reprezentate de acestea sunt egale

# Expresii regulare

- expresiile regulare – secv. obtinute prin concatenarea de simb. din
$$\Sigma \cup \{\Phi, \varepsilon, +, *, (, )\} \quad (\dots \text{prioritate} \dots)$$
  - multimile regulare asociate expresiilor regulare sunt limbaje regulare
- => orice expresie regulară peste  $\Sigma$  este un limbaj regular

# Expresii regulate si AF (exemplu)



*Care este expresia regulară ce îl descrie pe  $L(M)$ ?*

# Proprietati: expresii regulare echivalente

- “ = “ noteaza relatia dintre 2 expresii regulare echivalente

(reuniune si concaten.)

$$r + s = s + r$$

$$(r+s)+t = r+(s+t)$$

$$(rs)t = r(st)$$

$$(r+s)t = rt+st$$

$$r(s+t) = rs+rt$$

(utilizarea lui  $\Phi$  si  $\epsilon$ )

$$\Phi + r = r + \Phi = r$$

$$\epsilon r = r \epsilon = r$$

$$\Phi r = r \Phi = \Phi$$

$$\Phi^* = \epsilon$$

$$r^* + \epsilon = \epsilon + r^* = r^*$$

$$(\epsilon + r)^* = r^*$$

$$(r^*)^* = r^*$$

$$(r^*s^*)^* = (r+s)^*$$

# Expresii regulare

Exercitiu:

Fie  $r, s$  – expresii regulare oarecare

Demonstrati ca:

- $r^* r^* = r^*$
- $(r^*)^* = r^*$
- $(r^*s^*)^* = (r + s)^*$

# LEX / FLEX

**LEX -> generator de analizor LEXical**

~ 1975      Mike Lesk , Eric Schmidt

**FLEX - Fast LEX**

~ 1987      Vern Paxson

**doc:** “*pattern-matching techniques used by flex:  
deterministic finite automata*”  
(DFA)

# Expresii regulare

x

se potriveste cu caracterul *x*

r\*

zero sau mai multi *r*

r+

unul sau mai multi *r*

r?

zero sau un *r* (optional)

rs

concatenarea lui *r* cu *s*

r|s

sau *r* sau *s*

r{2,5}

doi, trei, patru sau cinci *r*

r{2,}

doi sau mai multi *r*

r{4}

exact patru *r*

*“character class”*

[xyz]

se potriveste cu oricare dintre 'x', 'y', 'z'.

[abj-oZ]

se potriveste cu oricare dintre 'a', 'b', orice de la 'j' la 'o', sau 'Z'

[^A-Z]

*“negated character class”*

se potriveste cu orice caracter cu exceptia celor specificate

.

orice caracter cu exceptia newline

{ name }

se potriveste cu definitia "name"

## *Exemplu:*

DIGIT	[0-9]
ID	[a-z] [a-z0-9]*

%%

```
if|then|while|do|begin|end
"+|-*"/
[ \t\r\n]           /* white spaces */
{DIGIT}+
{DIGIT}+."{DIGIT}*
{ID}
```

%

#include &lt;stdio.h&gt;

%

alpha [A-Za-z]

digit [0-9]

alphanum [A-Za-z0-9]

%%

[ \t\r\n]

"if"

"then"

"else"

"while"

"do"

"begin"

"end"

{alpha}{alphanum}\*{digit}+

.

%%

/\* ignore white spaces \*/  
printf("if \n");  
printf("then \n");  
printf("else \n");  
printf("while \n");  
printf("do \n");  
printf("begin \n");  
printf("end \n");  
printf("identifier \n");  
printf("number \n");  
/\* ignore other character\*/

# Formatul fisierului de specificatii

```
% {  
    // text copiat in fisierul generat  
    // - incluziuni de fisiere, variabile globale  
}  
definitii  
%%%  
reguli  
%%%  
cod utilizator
```

# Sectiunea de definitii

declaratii de *definitii de nume* simple, de forma  
nume definitie

- **nume** - este un cuvant compus din una sau mai multe litere, cifre, '\_' sau '-'  
primul caracter trebuie sa fie litera sau '\_' si sa se afle pe prima pozitie a liniei.
- **definitie** este o expresie regulara si se considera ca incepe cu primul caracter non-spatiu de dupa nume si tine pana la sfarsitul liniei.

Scop: pentru a simplifica specificarea regulilor

# Sectiunea de reguli

Scopul principal al acestei sectiuni este acela de a asocia

- expresii regulare (consideram ca o expr. regulara descrie un atom)
- actiuni *semantice* (cod C definit de utilizator).

Pentru aceasta se utilizeaza o constructie de forma:

## **sablon actiune**

unde:

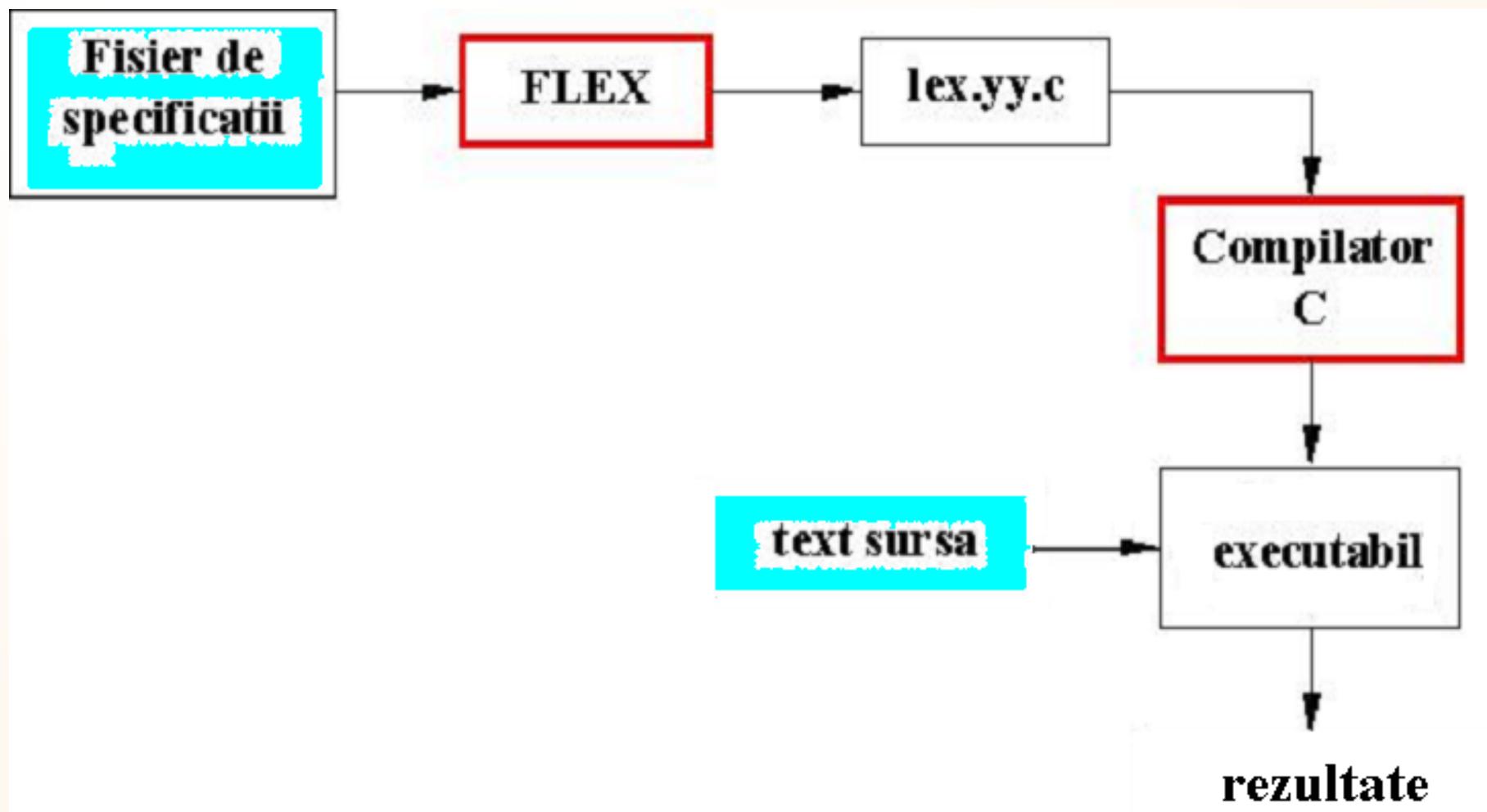
- **sablon** este o expresie regulara
- **actiune** este o secventa formata din 1 sau mai multe instructiuni C

Acestea trebuie sa inceapa pe aceeasi linie cu sablonul.

Daca dorim sa scriem instructiuni pe mai multe linii,  
acestea se vor inchide intre accolade.

In particular, actiunea poate fi si vida.

# Cum se lucreaza cu FLEX



# Executie analizor generat cu flex

## Lansarea în execuție flex

`flex [optiuni] [nume_fisier_specificatii]`

- unde *nume\_fisier\_specificatii* e fisierul de intrare
  - text sursa (implicit `stdin`)

e.g. (S.O. linux)

`$ flex spec.lxi`  $\Rightarrow$  `lex.yy.c`

`$ gcc lex.yy.c -o exe`  $\Rightarrow$  `exe`

`$ ./exe` (`<stdin>`)

`$ ./exe <fisier_intrare`

# Fisierul de specificatii

- fisier text
- Nume:  
poate avea orice nume acceptat de sistemul de operare. Din considerente istorice se prefera ca fisierul de specificatie sa aiba extensia **.lxi** sau **.l**
- Contine:
  - descrierea analizorului lexical
  - ... cod C

# Sectiunea de cod utilizator

- furnizam functia main
- specificam “pornirea analizorului lexical”,
  - apelam functia yylex
- de exemplu, in sectiunea cod utilizator:

...

**%%**

**int main()**

{

**yylex();**

}

# Sectiunea de cod utilizator

## Functia: int yywrap()

- spune ce facem dupa terminarea procesarii unui fisier  
(primește indicatorul EOF)
  - $\text{yywrap} \neq 0$  : dorim sa terminam procesarea
  - $\text{yywrap} = 0$  : dorim sa mai procesam ...
- de exemplu, in sectiunea cod utilizator:

...

**%%**

**int yywrap(){return 1;}**

- Optiuni echivalente cu: yywrap returneaza 1  
**%option noyywrap**

# Cateva variabile utile

***char \*yytext*** - reprezinta adresa zonei in care se depun caracterele atomului curent ;

***int yylen*** - reprezinta lungimea atomului curent;

***FILE \*yyin*** - desemneaza fisierul care contine textul sursa de analizat;

## Mod de functionare:

- pentru sabloanele recunoscute se aplica actiunile corespunzatoare din specificatie (sectiunea de reguli),
  - genereaza text specific sau “*pot returna coduri lexicale*”;
- textul din fisierul sursa care nu se potriveste cu nici un sablon este copiat automat in *yyout* (in cazul in care *yyout* nu a fost redefinit, el este implicit *stdout*);
- daca pentru un text se potrivesc mai multe sabloane, se alege potrivirea cea mai lunga;
- daca pentru un text se potrivesc mai multe sabloane de aceeasi lungime, se alege prima potrivire din specificatie, in ordine textuala;
- textul corespunzator potrivirii este copiat in variabila globala *yytext* iar lungimea acestuia este memorata in variabila globala *yyleng*.
- Cand analizorul intalneste marcajul EOF, el verifica rezultatul functiei *yywrap()*. Rezultat fals (zero) inseamna ca analiza trebuie sa continue cu noul fisier desemnat de *yyin*. Un rezultat nenul al functiei *yywrap()* determina sfarsitul analizei si returnarea valorii 0 spre apelantul lui *yylex()*.

# Gramatici independente de context (GIC) (CFG – context free grammars)

derivari, arbori de derivare,  
tipuri de gramatici independente de context,  
gramatici echivalente - constructii

# Ne reamintim: Gramatica

O gramatica este un cvadruplu  $\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$

- $\mathbf{N}$  este un alfabet de simboluri **neterminale**
- $\Sigma$  este un alfabet de simboluri **terminale**
- $\mathbf{N} \cap \Sigma = \emptyset$
- $P \subseteq (\mathbf{N} \cup \Sigma)^* \mathbf{N} (\mathbf{N} \cup \Sigma)^* \times (\mathbf{N} \cup \Sigma)^*$ 
  - $\mathbf{P}$  multime finită (multimea regulilor de productie)
  - $S \in \mathbf{N}$  (simbolul de start - simbolul initial)

Notatie:

$(\alpha, \beta) \in P$  se noteaza:  $\alpha \rightarrow \beta$   
( $\alpha$  se înlocuieste cu  $\beta$ )

# Ne reamintim: clasificarea Chomsky

- Gramatici de tip 0:  
nici o restrictie (*suplimentara*) referitoare la forma regulilor de productie
- Gramaticile de tip 1  
(gramatici monotone)
  - $\forall \alpha \rightarrow \beta \in P: |\alpha| \leq |\beta|$
  - caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . In acest caz S nu apare în membrul drept al nici unei reguli de productie.

## • Gramatici independente de context:

**reg. productie sunt de forma  $A \rightarrow \alpha$ ,  $A \in N$ ,  $\alpha \in (N \cup \Sigma)^*$**   
**(gramatici de tip 2)**

- Gramaticile de tip 3:  
reg. prod. sunt de forma
  - $A \rightarrow aB$
  - $A \rightarrow b$unde  $A, B \in N$  si  $a, b \in \Sigma$   
caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . In acest caz S nu apare în membrul drept al nici unei reguli de productie.

# derivarile de stanga/ dreapta

- ***derivare de stânga***  $\Rightarrow_{st}$   
o derivare directă în care se înlocuiește cel mai din  
stânga neterminal
- ***derivare de dreapta***  $\Rightarrow_{dr}$   
o derivare directă în care se înlocuiește cel mai din  
dreapta neterminal

# Analiza sintactica

- ***analiză sintactică*** pt. cuvantul w

succesiunea de derivări directe:

- $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = w$

altfel spus: reprezintă o derivare pentru cuvântul w

- ***analiză sintactică descendente***

dacă această succesiune de derivări directe se obtine pornind de la S și terminand cu w

- ***analiză sintactică ascendentă***

dacă această succesiune de derivări directe se obtine pornind de la w și terminand cu S

# Arbore de derivare

- Fie  $G = (N, \Sigma, P, S)$  o gramatică independentă de context. Numim *arbore de derivare* sau *arbore de analiză sintactică* un arbore cu radacina, ordonat, cu urmatoarele proprietati:
  1. Orice nod interior - o eticheta din  $N$ ;
  2. Orice nod frunza - o *etichetă* din  $\Sigma \cup \{\varepsilon\}$
  3. Eticheta rădăcinii este  $S$ ;
  4. Dacă un nod are eticheta  $A$  iar nodurile succesoare acestuia, în ordine de la stânga la dreapta sunt etichetate cu  $X_1, X_2, \dots, X_n$  atunci  $A \rightarrow X_1 X_2 \dots X_n$  trebuie să fie o producție din  $P$ .

# Arbore de derivare

- **frontiera (frontul)**: nodurile terminale, în ordine de la stânga la dreapta
- etichetele lor formează o secvență peste  $\Sigma^*$
- obs: denumirea de frontiera (front) se folosește și pentru a denumi succesiunea etichetelor nodurilor terminale

## Teoremă.

Fie  $G = (N, \Sigma, P, S)$  o gramatică independentă de context. Un cuvânt  $w$  peste alfabetul  $\Sigma$ , deci din  $\Sigma^*$ , aparține limbajului generat de  $G$ , adică  $w \in L(G)$ , dacă și numai dacă  $w$  este frontul unui arbore de analiză sintactică.

# Gramatica ambiguă

O gramatică  $\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$  independentă de context este *ambiguă*

dacă și numai dacă există cel puțin un cuvânt  $w$  care admite doi arbori de derivare distincti; în caz contrar gramatica este *neambiguă*.

- $\Leftrightarrow \exists 2$  analize sintactice care folosesc numai derivari de stanga, diferite
- $\Leftrightarrow \exists 2$  analize sintactice care folosesc numai derivari de dreapta, diferite

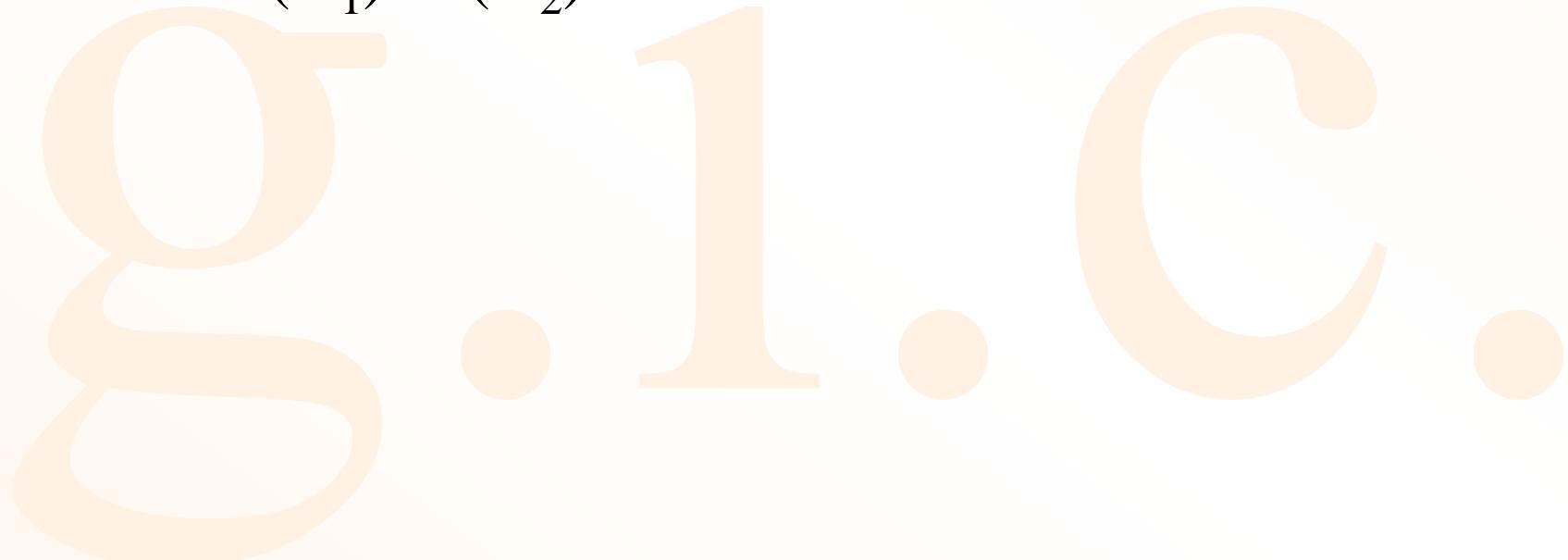
# Descrieri echivalente. Forme normale

**Ne reamintim :**

O gramatica  $G_1$

este echivalenta cu gramatica  $G_2$

daca  $L(G_1) = L(G_2)$



# $\epsilon$ -productii si gram. $\epsilon$ -independente

- **$\epsilon$ -productie** : o productie de forma  $A \rightarrow \epsilon$
- Gramatica  $G = (N, \Sigma, P, S)$  este  **$\epsilon$ -independentă** daca:
  - a) dacă  $\epsilon \notin L(G)$  atunci  $G$  nu are  $\epsilon$ -productii
  - b) dacă  $\epsilon \in L(G)$  atunci avem o singură productie  $S \rightarrow \epsilon$  iar celelalte productii nu-l contin în membrul drept pe  $S$
- Teorema
$$\forall G = (N, \Sigma, P, S)$$
$$\exists G' = (N', \Sigma', P', S) \text{ echivalentă, } \epsilon\text{-independentă}$$

# Redenumiri. Cicluri.

- ***redenumire***: reg.prod. de forma  $A \rightarrow B$
- ***Gramatica fără redenumiri***: fara r.p. de redenumire

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără redenumiri

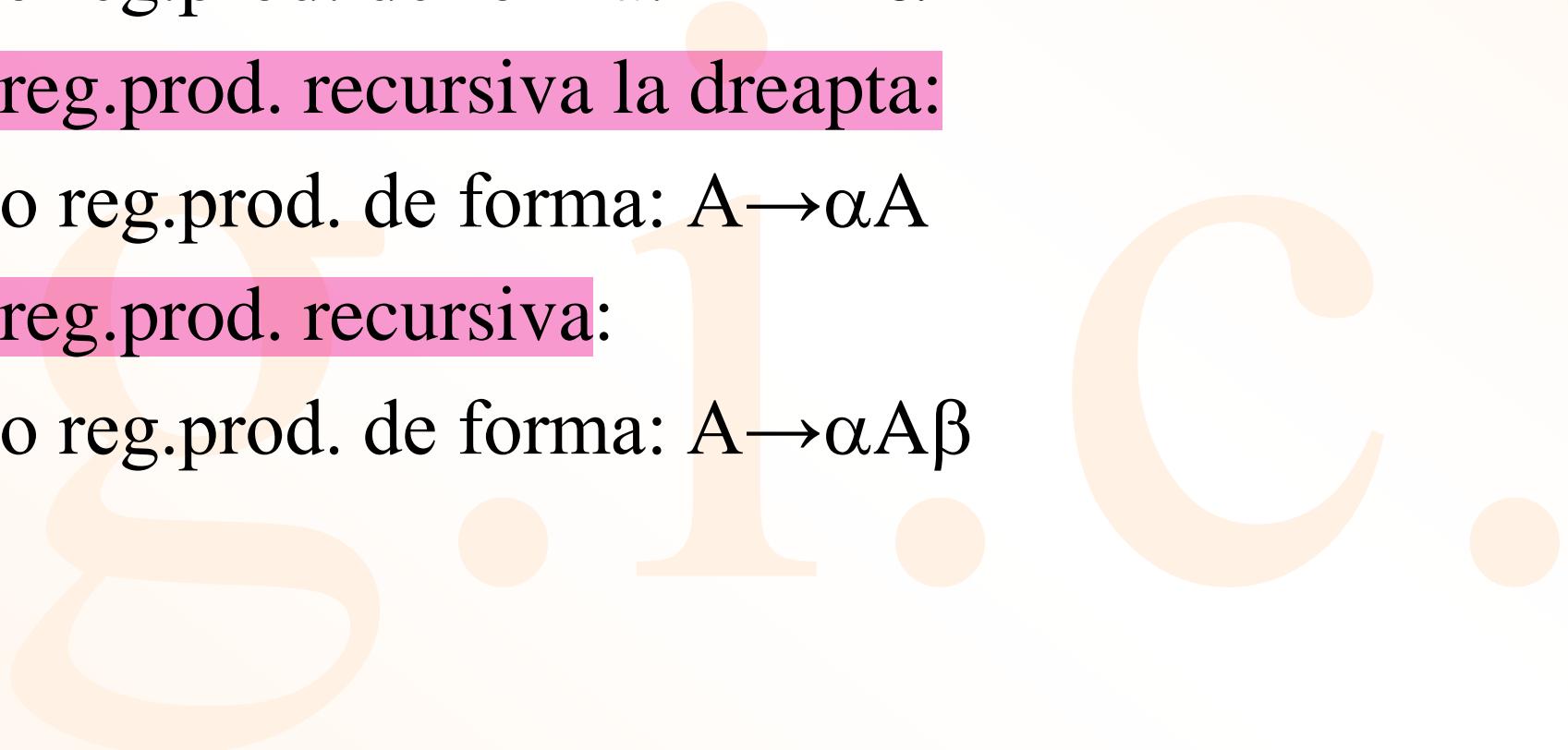
- ***ciclu***: o  $*$  derivare de forma  $A \Rightarrow^* B$
- ***Gramatica fără cicluri***: nu se pot obtine cicluri (la derivare)

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără cicluri

# Recursivitate

- reg.prod. recursiva la stanga:
  - o reg.prod. de forma:  $A \rightarrow A\alpha$
- reg.prod. recursiva la dreapta:
  - o reg.prod. de forma:  $A \rightarrow \alpha A$
- reg.prod. recursiva:
  - o reg.prod. de forma:  $A \rightarrow \alpha A \beta$



# Recursivitate

- **neterminal recursiv la stanga:**

$A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> A\alpha$

- **neterminal recursiv la dreapta:**

$A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> \alpha A$

- **neterminal recursiv:**

$A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> \alpha A \beta$

- **gramatica recursiva la stanga:**

are cel putin un neterminal recursiv la stanga

- **gramatica recursiva la dreapta:**

...

# Forma normală Chomsky

O gramatică independentă de context  $G = (N, \Sigma, P, S)$  este în **forma normală Chomsky (FNC)**

dacă orice regula de producție din  $P$  este de una din formele:

a)  $A \rightarrow BC$        $A, B, C \in N;$

b)  $A \rightarrow a$        $a \in \Sigma, A \in N;$

Si un caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . În acest caz  $S$  nu apare în membrul drept al nici unei reguli de producție.

**Teoremă.** Oricare ar fi  $G = (N, S, P, S)$  o gramatică independentă de context, întotdeauna există o gramatică în forma normală Chomsky  $G'$ , astfel încât  $L(G) = L(G')$ .

# Forma normală Greibach

O gramatică  $G = (N, \Sigma, P, S)$

este în *forma normală Greibach (FNG)*

dacă  $P$  are productii numai de forma:

$$A \rightarrow a\alpha, \quad A \in N, a \in \Sigma, \alpha \in N^*;$$

Si un caz special:  $S \rightarrow \varepsilon$  poate  $\in P$ . In acest caz  $S$  nu apare în membrul drept al nici unei reguli de productie.

**Teorema.** Oricare ar fi  $G = (N, \Sigma, P, S)$  o gramatică independentă de context, întotdeauna există o gramatică în forma normală Greibach, astfel încât  $L(G) = L(G')$ .

# Simplificarea GIC

- ***simbol neproductiv***

Un simbol  $A \in N$  este ***neproductiv*** dacă nu există nici o derivare de forma  $A =^* > x \quad (x \in \Sigma^*)$

- În caz contrar  $A$  este ***simbol productiv***

- **Teorema**

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă, fără simboluri neproductive

# Simplificarea GIC

(transformari echivalente)

- ***simbol inaccessible***

Un simbol  $X \in N \cup \Sigma$  este ***simbol inaccessible*** dacă nu există nici o  $*$  derivare:  $S \Rightarrow^* \alpha X \beta \quad (\alpha, \beta \in (N \cup \Sigma)^*)$

- În caz contrar simbolul este ***accessible***
- **Teorema**

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă, fără simboluri inaccessible

# Determinarea simbolurilor productive

≈ algoritm AF determ. stari productive

$$\begin{aligned} A &\rightarrow BC \\ B &\rightarrow bB \\ C &\rightarrow c \end{aligned}$$

1.  $i := 0 ; V_0 := \Phi$

2. Repeta

$$V_{i+1} := V_i \cup \{A \in N \mid \exists A \rightarrow \alpha \in P, \alpha \in (V_i \cup \Sigma)^*\}$$

$$i := i + 1$$

pana cand  $V_i = V_{i-1}$

$\{V_i - multimea simbolurilor productive\}$

# Determinarea simbolurilor accesibile

≈ algoritm AF determ. stari accesibile

1.  $i := 0 ; V_0 := \{S\}$

2. Repeta

$$V_{i+1} := V_i \cup \{ B \in N \mid \exists A \in V_i, \alpha, \beta \in (N \cup \Sigma)^* \text{ a.i. } A \rightarrow \alpha B \beta \in P \}$$

$i := i + 1$

**pana cand**  $V_i = V_{i-1}$

$\{V_i - multimea simbolurilor neterminale accesibile\}$

\* analog pentru simboluri terminale accesibile

# Simplificarea GIC

- Un simbol este **neutilizabil** dacă el este fie inaccesibil, fie neproductiv

- Teorema

$$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$$
 echivalentă fără simboluri neutilizabile



## Observatii:

Fie  $G = (N, \Sigma, P, S)$  o gramatica independenta de context:

- Fie un simbol neterminat A al gramaticii G.

Daca nu exista o regula de productie  $A \rightarrow \alpha$  in P  
atunci A este neproductiv

- Fie un simbol terminal a al gramaticii G.

Daca nu exista o regula de productie de forma  
 $B \rightarrow \alpha a \beta$  in P  
atunci a este inaccesibil

# *Eliminarea $\varepsilon$ -productiilor*

1. Construim multimea  $N_\varepsilon$  care are ca elemente acele neterminale care prin derivare conduc la  $\varepsilon$  adică :

- $N_\varepsilon = \{A \mid A \in N, A \Rightarrow^* \varepsilon\}$   
alg.  $\approx$  determinarea simb. productive

2. Determinam noile reguli de productie

- astfel incat productiile de forma  $A \rightarrow \varepsilon$  se elimina
- dar, daca  $\varepsilon \in L(G)$ , atunci  $\exists S \rightarrow \varepsilon$  si  $S$  nu apare în membrul drept al nici unei productii

# Determinarea lui $N_\varepsilon$

1.  $i := 0 ;$

$$V_0 := \{A \in N \mid \exists A \rightarrow \varepsilon \in P\}$$

2. **Repetă**

$$V_{i+1} := V_i \cup \{A \in N \mid \exists A \rightarrow \alpha \in P, \alpha \in (V_i)^*\}$$

$$i := i + 1$$

**pană cand**  $V_i = V_{i-1}$

# determinam noile reguli de productie

- productiile de forma  $A \rightarrow \varepsilon$  se elimina
- celelalte r.p. se rescriu astfel incat sa “suplineasca” eliminarea  $\varepsilon$ -productiilor astfel:

Fie r.p.  $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \dots B_k \alpha_k$

unde:  $B_i \in N_\varepsilon$

$\alpha_j$  nu contine simb. din  $N_\varepsilon$

Se inlocuieste cu:

$A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \dots X_k \alpha_k$

unde  $X_i = \begin{cases} B_i & \text{este unul dintre } B_i \text{ sau } \varepsilon \\ \varepsilon & \text{(se fac toate inlocuirile posibile)} \end{cases}$

**Ce lipseste ???**

# determinam noile reguli de productie

- continuare

Dacă  $\varepsilon \in L(G)$  trebuie să avem o  $\varepsilon$ -productie

“atunci avem productia  $S \xrightarrow{\varepsilon}$  și  $S$  nu apare în membrul drept al nici unei productii”

(gram.  $\varepsilon$ -independenta)

- adaugam un nou simbol de start  $S'$  și productiile  
 $S' \xrightarrow{\varepsilon | S}$

# Redenumiri. Cicluri.

- *redenumire*: reg.prod. de forma  $A \rightarrow B$
- *Gramatica fără redenumiri*: fara r.p. de redenumire

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără redenumiri

- *ciclu*: o \* derivare de forma  $A \Rightarrow^* B$
- *Gramatica fără cicluri*: nu se pot obtine cicluri (la derivare)

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără cicluri

# Eliminarea redenumirilor

**PP. G –  $\varepsilon$ -independenta** (daca nu , luam gr.echiv.  $\varepsilon$ -ind.)

Pentru fiecare  $A \in N$

se elimina redenumirile de forma  $A \rightarrow B$  ( $\forall B \in N$ )

- construieste multimile  $N_A = \{B \mid A =^* > B\}$ ;  
 $(\approx$  det. simb. accesibile)
- determinam noile reguli de productie

# Construieste $N_A = \{D \mid A =^* > D\}$

1.  $i := 0$  ;

$V_0 := \{A\}$

2. Repeta

$V_{i+1} := V_i \cup \{C \mid (B \rightarrow C) \in P, B \in V_i\}$

$i := i + 1$

pana cand  $V_i = V_{i-1}$

$N_A := V_i$

# determinam noile reguli de productie

$A \in N:$

- **pentru** fiecare  $A \rightarrow \alpha \in P$  execută
- **daca**  $\alpha$  e format dintr-un singur neterminal  
**atunci**
  - il excludem din mult. noilor reg.prod
- **altfel**
  - adaugam:  $B \rightarrow \alpha, \forall B \in N$  a.i.  $A \in N_B$
- **sf.daca**
- **sf.pentru**

# Eliminarea redenumirilor

## Exercitiu:

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow a$$

1.C

# Gramatica fara cicluri

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă  
fără cicluri

Daca  $G$  –  $\varepsilon$ -independenta si fara redenumiri  
atunci este fara cicluri

# Gramatica propre:

este o gramatica

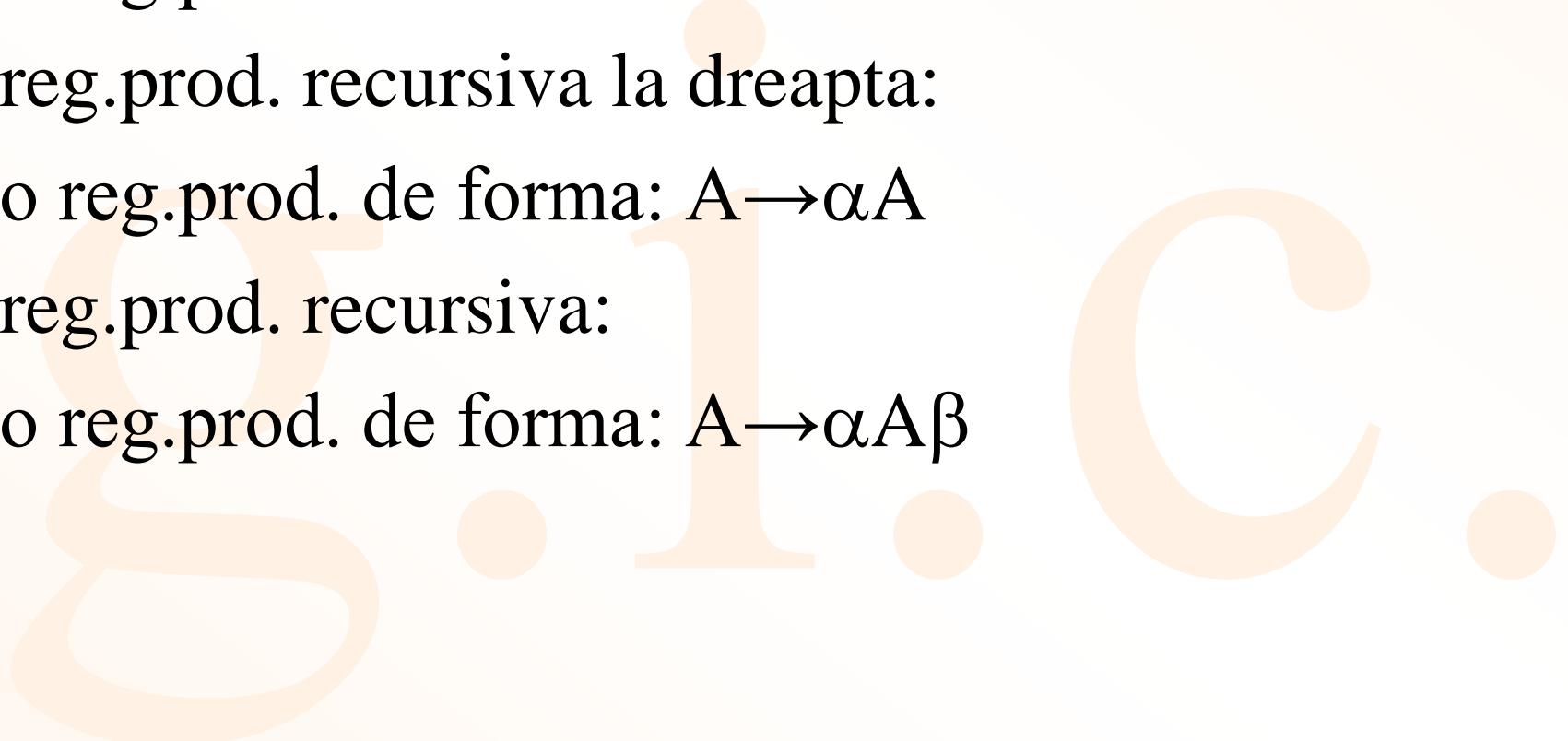
- fara simb. neutilizabile
- **$\epsilon$ -independenta**
- fara cicluri

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  propri echiv.

# Recursivitate

- reg.prod. recursiva la stanga:
  - reg.prod. de forma:  $A \rightarrow A\alpha$
- reg.prod. recursiva la dreapta:
  - reg.prod. de forma:  $A \rightarrow \alpha A$
- reg.prod. recursiva:
  - reg.prod. de forma:  $A \rightarrow \alpha A \beta$



# Reg. prod. recursive la stanga

- reg.prod. recursiva la stanga:  $A \rightarrow A\alpha$

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără reg.prod. recursive la stanga

- **PP. G – gr. propre**  
(daca nu este, det. gr. propre echiv. si lucram cu ea)
- Obs.: vom obtine tot o gramatica propre

# Eliminarea r.p. recursive la stanga

pentru fiecare  $A \in N$ : reg.prod.cu m.s. A

- grupam r.p. in recursive la stng. si nerec. la stanga

$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_r$  (r.p. recursive)

$A \rightarrow \beta_1 | \beta_2 | \dots | \beta_s$  (r.p. ne-recursive)

- r.p. se transforma astfel:

$A \rightarrow \beta_1 | \beta_2 | \dots | \beta_s | \beta_1 A' | \beta_2 A' | \dots | \beta_s A'$

$A' \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_r | \alpha_1 A' | \alpha_2 A' | \dots | \alpha_r A'$

(a fost introdus un net. nou:  $A'$ )

# Eliminarea r.p. recursive la stanga

## Observatii:

- Recursivitatea nu se poate elimina.
- Recursivitatea la stanga a fost transformata în recursivitate la dreapta.

Exercitiu

Eliminati recursivitatea la stanga:

$S \rightarrow Sa$

$S \rightarrow a$

# Recursivitate

- **neterminal recursiv la stanga:**  
 $A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> A\alpha$
- **neterminal recursiv la dreapta:**  
 $A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> \alpha A$
- **neterminal recursiv:**  
 $A \in N$  daca  $\exists$  o derivare de forma:  $A =^+> \alpha A \beta$
- **gramatica recursiva la stanga:**  
are cel putin un neterminal recursiv la stanga
- **gramatica recursiva la dreapta:** ...

# Eliminarea recurs. la stg. a neterm.

Teorema:

$\forall G = (N, \Sigma, P, S) \exists G' = (N', \Sigma', P', S)$  echivalentă fără neterminale recursive la stanga

- PP.  $G$  – gr. propriie  
(daca nu este, det. gr. propriie echiv. si lucram cu ea)
- impunem o ordine asupra neterminalelor  
 $N = \{A_1, A_2, \dots, A_n\}$   
si apoi modific r.p. a.i. sa nu existe  $A_i \rightarrow A_j \alpha$  cu  $j <= i$   
de aici  $\Rightarrow$  nu va exista recursivitate la stanga

# Eliminarea recurs. la stg. a neterm.

**pentru**  $A_i$  de la  $A_1$  la  $A_n$  **executa**

//se elimina r.p. de forma  $A_i \rightarrow A_j \alpha$  cu  $j <= i$  astfel:

\***repeta**

**pentru**  $j := 1, i - 1$  **executa**

\*  $A_i \rightarrow A_j \alpha$  ( $j < i$ ) se inlocuieste cu:  $A_i \rightarrow \beta \alpha$   
cu toti  $\beta$  cu proprietatea  $A_j \rightarrow \beta \in P_{inloc}$

**sf.pentru**

\* se elimina r.p. de forma  $A_i \rightarrow A_i \alpha$

(se inlocuiesc cf.alg. de elim.r.p.rec.stg)

\***pana cand** toate r.p. cu  $A_i$  in m.s. respecta:  $\nexists j < i : A_i \rightarrow A_j \alpha$

**sf.pentru**

# Eliminarea recurs. la stg. a neterm.

Exercitii:

(1)

$$A \rightarrow BC \mid a$$

$$B \rightarrow CA \mid b$$

$$C \rightarrow AB \mid c$$

(2)

$$A \rightarrow a \mid aB$$

$$B \rightarrow AC \mid b$$

$$C \rightarrow BA \mid c$$

# **Proprietati ale limbajelor independente de context**



# Lema de pompare pentru limbaje independente de context

Fie  $L$  un limbaj independent de context. Există atunci o constantă  $p$  dependentă numai de  $L$  astfel că dacă  $z \in L$  și  $|z| \geq p$ , atunci avem descompunerea  $z = uvwxy$  cu proprietătile:

- a)  $|vx| \geq 1$ ,
- b)  $|vwx| \leq p$ ,
- c)  $uv^iwx^i y \in L \quad \forall i \in \mathbb{N}$

?

$S \rightarrow 0A1$   
 $A \rightarrow 0S$   
 $A \rightarrow a$

# Lema de pompare pentru limbaje independente de context

- *definitia formală:*

$$\begin{aligned} \forall L \in GIC \quad \exists p \in \mathbf{N}^* \quad \forall z \in L \quad |z| \geq p \\ \rightarrow \\ (\exists u, v, w, x, y \quad z = uvwxy \\ \quad \wedge \quad |vwx| \leq p \quad \wedge \quad |vx| \geq 1 \\ \quad \wedge \quad (\forall n \in \mathbf{N} : uv^n w x^n y \in L)) \\ ) \end{aligned}$$

# Proprietăți de închidere ale limbajelor independente de context

## Teorema.

Dacă  $L_1$  și  $L_2$  sunt limbaje independente de context atunci:

$$L_1 \cup L_2, L_1 L_2, L_1^*$$

sunt limbaje independente de context.

## Observatie:

$L_1 \cap L_2$ ,  $\text{compl}(L_1)$  - nu sunt neapărat l.i.c.

# Exercitii

Pentru urmatoarele limbaje, scrieti cate o gramatica independenta de context care le genereaza:

$$L_1 = \{ a^n b^n \mid n \in \mathbb{N} \}$$

$$L_2 = \{ c^n \mid n \in \mathbb{N} \}$$

$$L_3 = \{ a^n b^n c^m \mid m, n \in \mathbb{N} \}$$

$$L_4 = \{ a^n b^m c^m \mid m, n \in \mathbb{N} \}$$

$$L_1 \cup L_2, L_1 L_2, L_1^*$$

## Problema 1:

Fie limbajul:

$$L = \{a^n b^n c^n \mid n \in N\}$$

Este independent de context?

### Rezolvare:

- Facem **observatia** ca:  $z \in L$  daca:
  - a. ordinea simb. este data de regulile:
    - i. simb. **a** apar inaintea simb. **b** si **c**
    - ii. simb. **b** apar inaintea simb. **c**
  - b. nr. simb. **a** este egal cu nr. simb. **b** este egal cu nr. simb. **c**  
(si notam:  $nr_a(z) = nr_b(z) = nr_c(z)$ )

Vom dem. ca nu este independent de context, prin reducere la absurd, folosind lema de pompare pentru limbaje independente de context.

- PP. ca este independent de context.  
Atunci au loc conditiile din lema de pompare

De aici rezulta ca  $\exists p \in N^*$  astfel incat:

$\forall z \in L$  care satisface

- $|z| \geq p$
- $\exists$  o descompunere  $z = uvwxy$  astfel incat:  $uv^iwx^i y \in L, \forall i \in N$ 
  - si  $|vx| \geq 1$
  - si  $|vwx| \leq p$

### Dem., Versiunea 1:

Alegem  $z$  cu  $|z| \geq p$  (satisfac cond. de mai sus)

- $\exists n$  a.i.  $|a^n b^n c^n| \geq p$ ;  $z \in L \Rightarrow z = a^n b^n c^n$  si  $|z| \geq p$
- $z = uvwxy$  descompunerea din lema de pompare  
ne aflam in unul din urmatoarele cazuri generale:
  1. cel putin unul dintre **v** si **x** contin cel putin 2 simboluri (dintre a,b,c) diferite; **(cazul 1)**
  2. **v** si **x** contin un singur simbol de oricate ori (o sau mai multe) dar acelasi simbol (sau a, sau b, sau c)  
dar **v** si **x** nu pot fi ambele vide **(cazul 2)**
  3. **v** si **x** contin un simbol (a, sau b, sau c) de oricate ori,  
dar nu pot fi vide,  
dar **v** si **x** nu contin acelasi simbol **(cazul 3)**

**cazul 1:** (vezi cazurile posibile pentru cazul 1; aleg unul dintre ele si dem. pt. el;  
pentru celelalte demonstratia se face analog)

fie:  $v = a^{k1} b^{k2}$ ,  $k1 > 0, k2 > 0$  (**rel.1**) (oricare x)  
fie  $i = 2$

cf. Lemei de pompare:  $uv^2wx^2y \in L$

adica:

$$uv^2wx^2y = u a^{k1} \underline{b^{k2}} \underline{a^{k1}} b^{k2} wx^2y \in L,$$

atunci cand  $k1 > 0$  si  $k2 > 0$  (cf. rel.1)

ar insemana ca simb. **b** pot sa apara inaintea simb. **a**

ceea ce nu e adevarat pentru cuvintele din L

(observatia (a.)(i.))

$\Rightarrow$  contradictie

Se poate dem. in mod **analog** ca:

- pentru oricare doua (sau trei) simboluri distincte ar fi format v,  $v^2$  nu va mai pastra ordinea simbolurilor care este necesara pt.ca  $uv^2wx^2y \in L$

...  $\Rightarrow$  contradictie

- pentru oricare doua (sau trei) simboluri distincte ar fi format x,  $x^2$  nu va mai pastra ordinea simbolurilor care este necesara pt.ca  $uv^2wx^2y \in L$

...  $\Rightarrow$  contradictie

**cazul 2:** (dintre cazurile posibile pentru cazul 2 aleg unul dintre ele si dem. pt. el)

$$\begin{aligned} \text{fie: } v &= a^{k1} & k1 >= 0 \\ x &= a^{k2} & k2 >= 0 \end{aligned}$$

Stim ca:  $|vx| >= 1$

$$\Leftrightarrow |a^{k1}a^{k2}| >= 1$$

$$\Leftrightarrow k1 + k2 > 0 \quad (\text{rel.2})$$

( $k1, k2$  – nu sunt simultan 0)

atunci:  $u = a^{k3}, k3 >= 0$

$w = a^{k4}, k4 >= 0$

$y = a^{n-k1-k2-k3-k4}b^n c^n, n-k1-k2-k3-k4 >= 0$

fie  $i = 2$ : cf. lemei:  $uv^2wx^2y \in L$

$$uv^2wx^2y = a^{k3} a^{2*k1} a^{k4} a^{2*k2} a^{n-k1-k2-k3-k4} b^n c^n$$

dar:  $uv^2wx^2y \in L \Rightarrow nr_a(z') = nr_b(z') = nr_c(z')$

$$k3 + 2*k1 + k4 + 2*k2 + n - k1 - k2 - k3 - k4 = n = n$$

$$\Rightarrow n + k1 + k2 = n$$

$$\Rightarrow k1 + k2 = 0$$

dar (cf. rel.2) :  $k1 + k2 > 0$

$\Rightarrow$  contradictie

Se dem. analog pt. orice alte combinatii posibile atunci cand

si **y** si **u** contin un acelasi simbol (**a**, sau **b**, sau **c**),

ca in  $z' = uv^2wx^2y$  nu are loc relatia  $nr_a(z') = nr_b(z') = nr_c(z')$

$\Rightarrow$  contradictie

**cazul 3:** (dintre cazurile posibile pentru cazul 3 aleg unul dintre ele si dem. pt. el)

$$\begin{aligned} \text{fie: } v &= a^{k1}, k1 > 0 & (\text{rel.4}) \\ x &= b^{k2}, k2 > 0 & (\text{rel.5}) \end{aligned}$$

atunci:  $u = a^{k3}, k3 >= 0$

$$y = b^{k_4}c^n, k_4 >= 0$$

$$w = a^{n-k_1-k_3}b^{n-k_2-k_4}, n-k_1-k_2 >= 0; n-k_2-k_4 >= 0$$

fie  $i = 2$ ; atunci  $uv^2wx^2y \in L$

$$uv^2wx^2y = a^{k_3} a^{2*k_1} a^{n-k_1-k_2} b^{n-k_2-k_4} b^{2*k_2} b^{k_4} c^n$$

$$z' = uv^2wx^2y \in L \Rightarrow nr_a(z') = nr_b(z') = nr_c(z')$$

$$k_3 + 2*k_1 + n - k_1 - k_3 = n - k_2 - k_4 + 2*k_2 + k_4 = n$$

$$\Rightarrow n + k_1 = n + k_2 = n$$

$$\Rightarrow k_1 = 0 \text{ contradicție (rel.4)}$$

$$(\Rightarrow k_2 = 0, \text{ contradicție (rel.5)})$$

Se dem. analog pt. orice alte combinatii posibile atunci cand  
 si v si x contin cate un simbol (a, sau b, sau c), dar nu acelasi  
 ca in  $z' = uv^2wx^2y$  nu are loc relatia  $nr_a(z') = nr_b(z') = nr_c(z')$   
 $\Rightarrow \underline{\text{contradicție}}$

---

### **cazurile posibile pt. cazul 1**

$$z = a^n b^n c^n, z = uvwxy$$

cel putin unul dintre v si x contin cel putin 2 simboluri (dintre a,b,c) diferite;

$$v = a^{k_1} b^{k_2}, k_1 > 0, k_2 > 0 \text{ si nu specificam ce poate contina x}$$

$$v = a^{k_1} b^{k_2} c^{k_3}, k_1 > 0, k_2 > 0, k_3 > 0 \text{ si nu specificam ce poate contina x}$$

$$v = b^{k_2} c^{k_3}, k_2 > 0, k_3 > 0 \text{ si nu specificam ce poate contina x}$$

daca v contin un singur acelasi simbol, ne situam in cazul 1 daca:

$$x = a^{k_1} b^{k_2}, k_1 > 0, k_2 > 0$$

$$x = a^{k_1} b^{k_2} c^{k_3}, k_1 > 0, k_2 > 0, k_3 > 0$$

$$x = b^{k_2} c^{k_3}, k_2 > 0, k_3 > 0$$

analog se face dem. pt. fiecare dintre cazurile de mai sus (ajunge la o contradictie)

Exercitiu:

descrieti cazurile posibile pt. cazul 2 si cazul 3

---

### **Dem., Versiunea 2 ( scurta ☺ ):**

Alegem  $z$  cu  $|z| \geq p$  (satisfacă condiția de mai sus)

$$z = a^p b^p c^p$$

- $\Rightarrow |z| \geq p$
- $z = uvwxy$  descompunerea din lema de pompare
  - astfel încât:  $uv^iwx^i y \in L, \forall i \in N$ 
    - si  $|vx| \geq 1$
    - si  $|vwx| \leq p$

Pentru ca  $|vwx| \leq p$  : secvența  $vwx$  conține maxim 2 simboluri dintre a, b, c.

Astfel, în secvența  $uv^iwx^i y$  există cel puțin un simbol care nu este "pompat" și cel puțin unul care este "pompat" ; astfel se pierde egalitatea dintre numărul de aparitii ale celor două simboluri.

---

# Automate push-down (APD)

---

# Automat Push Down (APD)

## Definitie:

Un automat push-down (APD) este un ansamblu

$$M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)$$

, unde:

- $Q$  alfabetul starilor;
- $\Sigma$  alfabetul de intrare;
- $\Gamma$  alfabetul memoriei stivă; ;
- $q_o \in Q$  stare inițială;
- $Z_o \in \Gamma$  simbolul de start al memoriei stivă;
- $F \subseteq Q$  multimea stărilor finale;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  funcția de tranziție

# Reprezentare

- enumerare
- tabelara
- sub forma de graf

# Reprezentare folosind enumerare

Exemplu:

- $M = (\{q_0, q_1\}, \{a, b\}, \{A, Z\}, \delta, q_0, Z, \{q_0\})$
- $\delta$ :
  - $\delta(q_0, a, Z) = \{(q_0, AZ)\}$
  - $\delta(q_0, a, A) = \{(q_0, AA)\}$
  - $\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$
  - $\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$
  - $\delta(q_0, \varepsilon, Z) = \{(q_0, \varepsilon)\}$
  - $\delta(q_1, \varepsilon, Z) = \{(q_1, \varepsilon)\} \dots$  si  $\Phi$  in celelalte cazuri

# Reprezentare tabelara

Exemplu:

		a	b	$\varepsilon$	
$q_0$	Z	$(q_0, AZ)$		$(q_0, \varepsilon)$	1
	A	$(q_0, AA)$	$(q_1, \varepsilon)$		
$q_1$	Z			$(q_1, \varepsilon)$	0
	A		$(q_1, \varepsilon)$		

Care este limbajul acceptat dupa criteriul stivei vide?

Dar dupa criteriul starii finale?

Dar daca starea finala ar fi q1?

# Configuratie

- **formal:**

$$(q, x, \alpha) \in Q \times \Sigma^* \times \Gamma^*$$

- automatul se găsește în starea  $q$ , pe banda de intrare urmează să se citească (acalte) secvența  $x$ , iar în memoria stivă avem secvența  $\alpha$
- configuratie initiala

$$(q_0, w, Z_0)$$

# Tranzitii

- $\vdash$  *tranzitie directă*

$$(q, aw, Z\alpha) \vdash (p, w, \gamma\alpha) \Leftrightarrow \delta(q, a, Z)\ni(p, \gamma)$$

sau

$$(q, aw, Z\alpha) \vdash (p, aw, \gamma\alpha) \Leftrightarrow \delta(q, \varepsilon, Z)\ni(p, \gamma) \quad (\underline{\varepsilon\text{-tranzitie}})$$

– unde  $p, q \in Q$ ,  $a \in \Sigma$ ,  $Z \in \Gamma$ ,  $w \in \Sigma^*$ ,  $\alpha, \gamma$  din  $\Gamma^*$

- $\vdash^k$  –  **$k$  tranzitii** ( $k$  tranzitii directe) ~AF
- $\vdash^+$  – **+ tranzitii** ~AF
- $\vdash^*$  – **\* tranzitii** ~AF

# Secvența acceptată de automat

- după criteriul stivei vide

$L_\varepsilon(M) = \{ w \mid w \in \Sigma^*, (q_o, w, Z_o) \vdash^* (q, \varepsilon, \varepsilon), q \in Q \}$   
–  $(q, \varepsilon, \varepsilon)$  - configurația finală după criteriul stivei vide

- după criteriul stării finale

$L_f(M) = \{ w \mid w \in \Sigma^*, (q_o, w, Z_o) \vdash^* (q, \varepsilon, \gamma), q \in F, \gamma \in \Gamma^* \}$   
–  $(q, \varepsilon, \gamma)$ ,  $q \in F$  configurație finală după criteriul stării finale

$$L_{\varepsilon}(M) = \{ a^n b^n \mid n \in \mathbb{N} \}$$

De ce? (justificare:)

in  $q_0$  – se accepta oricate simboluri a

cu ramanere in  $q_0$  si adaugare cate un A in stiva

**adica: la fiecare citire de a adaug in stiva un A** (1)

sau – se trece in starea  $q_1$  (dupa ce am citit cel putin un a, adica am A in stiva)

obs.: se poate trece in  $q_1$  oricand, fara modificarea stivei (2)

sau - se scoate Z din stiva

(acest lucru se poate intampla numai inainte de citirea unui simb)

**=> se accepta seventa vida**

in  $q_1$  – cand in varful stivei este un A, se citeste un b

**adica: fiecare citire de b scoate un A din stiva** (3)

sau: daca in varful stivei este un Z, acesta se scoate (goleste stiva)

din (1) , (2) , (3) => nr(a) = nr(b)

(4)  $q_0$  citeste a (oricati)

(5)  $q_1$  citeste numai b; nu se poate trece inapoi in  $q_0$

din (2), (4) , (5) => simb. a citite inaintea simb b

		a	b	$\varepsilon$	
$q_0$	Z	$(q_0, AZ)$		$(q_0, \varepsilon)$	1
	A	$(q_0, AA)$		$(q_1, A)$	
$q_1$	Z			$(q_1, \varepsilon)$	0
	A			$(q_1, \varepsilon)$	

# Teoreme de echivalenta

## Teoremă.

Fie automatul push-down  $M$ . Există întotdeauna un automat push-down  $M'$  astfel încât  $L_\varepsilon(M') = L_f(M)$ ; și reciproc.

## Teoremă.

Oricare ar fi  $G$  – o gramatica independentă de context, există un automat push-down  $M$  astfel încât  $L_\varepsilon(M) = L(G)$ ;  
și reciproc.

# G.I.C. => APD echivalent

Fie:  $\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$  – gram. independentă de context

Cine este  $M$  - APD astfel încât  $L(G) = L_\varepsilon(M)$  ?

**constructia:**

$$M = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \Phi)$$

1. dacă  $(A \rightarrow \alpha) \in P$  atunci  $(q, \alpha) \in \delta(q, \varepsilon, A)$ ;
2.  $\delta(q, a, a) = \{(q, \varepsilon)\} \quad \forall a \in \Sigma$  ;
3.  $\delta(\dots) = \Phi$  în celelalte cazuri.

# Determinism

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

este **determinist** dacă:

$$\forall Z \in \Gamma, \forall q \in Q, \forall a \in \Sigma$$

$$1) |\delta(q, \varepsilon, Z)| = 0 \text{ si } |\delta(q, a, Z)| \leq 1$$

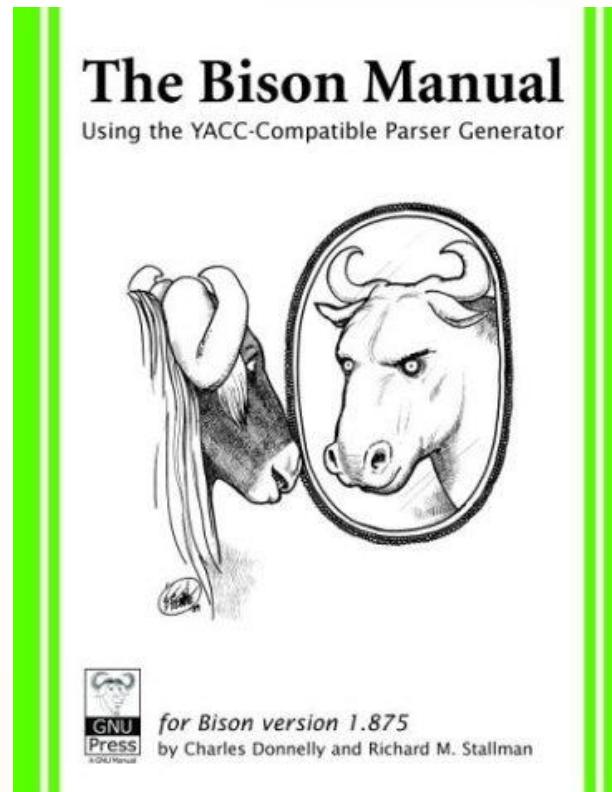
$$2) |\delta(q, \varepsilon, Z)| = 1 \text{ si } |\delta(q, a, Z)| = 0$$

în caz contrar, automatul nu este determinist

- multimea limbajelor acceptate de APD nedeterministe este strict mai largă decât multimea limbajelor acceptate de APD deterministe

# yacc / bison

## generatoare de analizoare sintactice



- **yacc** = Yet Another Compiler Compiler  
**1970 - 1975 , Stephen C. Johnson**
- **bison** =  
“ GNU Project parser generator (yacc replacement)”  
**1988 - 1990 Robert Corbett, Richard Stallman**  
  
“general-purpose parser generator  
that converts a context-free grammar ...  
into a C program to parse that grammar “

<http://dinosaur.compilertools.net/bison/index.html>

<http://www.gnu.org/software/bison/manual/bison.html>



- **yacc** = Yet Another Compiler Compiler
- **bison** =  
“GNU Project parser generator (yacc replacement)”

**gramatică bazată pe LR(1) → cod C**

sub UNIX

\$ info bison

\$ bison -V

bison (GNU Bison) 2.5 2.4.1

Written by Robert Corbett and Richard Stallman

~ 1988 – 1990

•

# Structura fișierului de intrare

```
% {  
declarații C  
% }
```

*declarații*

```
%%  
reguli ale gramaticii  
%%
```

*Cod C*

## Secțiunea *reguli ale gramaticii*:

- regulile multiple pt. același *rezultat* pot fi scrise separat sau pot fi unificate prin “|”
- pentru fiecare componentă pot fi specificate *acțiuni* (semantica regulii) {*instrucțiuni C*}
- actiunea poate fi vida

*rezultat:*      *regula1-componente...* {*instr. C 1*}

                  | *regula2-componente...* {*instr. C 2*}

      ...

;

# Observatii :

- presupune ca exista functiile:

**int yylex()**

- returneaza codul unui atom  
**( <= %token DIGIT )**

**yyerror**

- apelata atunci cand se detecteaza o eroare de sintaxa

- genereaza functia

- **int yyparse(void)**

- nu genereaza functia main

# Restul fisierului

- yacc generează o funcție `yyparse()`
- erorile sintactice sunt raportate apelând `yyerror()`

```
%%
```

```
yylex()
```

```
{
```

```
    ...
```

```
}
```

```
main()
```

```
{
```

```
    yyparse();
```

```
}
```

```
yyerror()
```

```
{
```

```
    printf("syntax error\n");
```

```
    exit(1);
```

```
}
```

# Folosire bison

- \$ bison fisier.y  
=> fisier.tab.c  
... gcc ...

# bison + flex pot fi folosite impreuna

Bison poate folosi functia yylex generata de flex.

---

- ?? constantele asociate atomilor lexicali
- \$ bison -d fisier.y
  - => fisier.tab.h
  - => fisier.tab.c
- fisierul \*.lxi (ce urmeaza sa fie compilat cu flex)  
    #include “fisier.tab.h”

# Valori semantice ale neterminalelor

- permite ca actiunea semantica asociata unei reguli de productie sa fie descrisa in functie de valorile semantice ale neterminalelor

- exemplu:

expr: expr '+' expr { \$\$ = \$1 + \$3; }

- **\$\$** - valoarea semantica asociata net. din membrul stang

- **\$i** - valoarea semantica asociata  
celui de-al i-lea net. din membrul drept

- **yylval** - valoarea semantica a unui atom

**<= yylex !!**

# Legatura cu yylex

- yacc apelează `yylex()` pt. a obține următorul atom
- “valoarea” unui atom se memoreaza în var. **globală yylval**
- tipul implicit este `int`

```
%%
yylex()
{
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return DIGIT;
    }
    return c;
}
```

# Interpreter expresii

```
%%
line : expr '\n'          { printf("%d\n", $1); }
;
expr : expr '+' expr     { $$ = $1 + $3; }
| expr '*' expr         { $$ = $1 * $3; }
| '(' expr ')'          { $$ = $2; }
| DIGIT                  { $$ = $1; }
;
%%
```

## Gramatica ambigua !

- atunci cand construieste arborele de derivare – care alegere se face?
- eroare: “conflict shift/reduce”

# Precedență operatorilor

prioritate  
de sus  
(mică)  
în jos  
(mare)

```
%token DIGIT
%left '+'
%left '*'
%%

line : expr '\n'      { printf("%d\n", $1); }

expr : expr '+' expr   { $$ = $1 + $3; }
      | expr '*' expr    { $$ = $1 * $3; }
      | '(' expr ')'     { $$ = $2; }
      | DIGIT             { $$ = $1; }

%%
```

declaratie de atom !!

# Analizorul descendente cu reveniri

- Automat: configuratii, tranzitii
- Se poate folosi pentru gramatici nerecursive la stanga

# Analizorul descentent cu reveniri

Configuratie:

$(s, i, \alpha, \beta)$

- **s - starea automatului**
  - q – stare normala
  - r – stare de revenire (sau b – back)
  - t – stare de terminare (terminare cu succes)
  - e – stare de eroare
- **i – pozitia (urmatoare) in sevenita de intrare**
- **$\alpha$  – stiva de lucru (de istorie): istoria r.p. aplicate**
- **$\beta$  – stiva de intrare: partea inca neprelucrata**

# Analizorul descendente cu reveniri

- configuratie initiala:  $(q, 1, \varepsilon, S)$

## Tranzitii:

- expandare:
- avans:
- insucces de moment:

- succes:
- revenire:  $(r, i, \alpha a, \beta)$
- alta incercare:  $(r, i, \alpha A_j, \gamma_j \beta)$

daca  $\exists A_{j+1} \rightarrow \gamma_{j+1}$   
 altfel daca  $i = 1, A = S$

altfel

$$\begin{array}{ll}
 (q, i, \alpha, A\beta) \xrightarrow{} & (q, i, \alpha A_1, \gamma_1 \beta) \\
 (q, i, \alpha, a_i \beta) \xrightarrow{} & (q, i+1, \alpha a_i, \beta) \\
 (q, i, \alpha, a \beta) \xrightarrow{} & (r, i, \alpha, a \beta), a <> a_i \\
 (q, i, \alpha, \varepsilon) \xrightarrow{} & (r, i, \alpha, \varepsilon), i \neq n+1 \\
 (q, n+1, \alpha, \varepsilon) \xrightarrow{} & (t, n+1, \alpha, \varepsilon)
 \end{array}$$

$$\xrightarrow{} (r, i-1, \alpha, a \beta)$$

$$\xrightarrow{} \dots$$

$$\begin{array}{l}
 \xrightarrow{} (q, i, \alpha A_{j+1}, \gamma_{j+1} \beta) \\
 \xrightarrow{} (e, i, \alpha, A \beta)
 \end{array}$$

$$\alpha = \varepsilon, \beta = \varepsilon$$

$$\xrightarrow{} (r, i, \alpha, A \beta)$$

Orice altceva: blocare (eroare)

# Analizorul descendant cu reveniri

## Observatii:

- Se numeroteaza regulile de productie cu acelasi membru stang
- Stiva de lucru contine informatiile referitoare la regulile de productie aplicate

# Schita subalgoritm ASDR

## Subalgoritmul ASDR(G, x, sir\_prod)

```
s:=q; i:=1; α:=ε; β:=S; //config initiala
Cattimp ((s≠t) si (s ≠e)) execută
daca (s=q) atunci
    daca (β=ε) atunci
        daca (i=n+1) atunci
            s:=t
        altfel
            s:=r
        sf_daca
    altfel
        daca (varf(β) = un neterminal A) atunci
            pop(β, A);
            push(α, A1);
            push(β, Y1);
        altfel
            daca (varf(β) = terminalul xi) atunci
                i:=i+1;
                pop(β, xi);
                push(α, xi);
            altfel s:=r;
            sf_daca
        sf_daca
    altfel
        // ...
```

```
daca(s=r) atunci
    daca (varf(α) = un terminal a) atunci
        i:=i-1;
        pop(α, a);
        push(β, a);
    altfel // varf(α) - un Aj oarecare (indicatie pt. A→Yj)
        daca (exista r.p. A→Yj+1 ) atunci
            s:=q;
            pop(α, Aj);
            push(α, Aj+1);
            pop(β, Yj );
            push(β, Yj+1 );
    altfel
        daca (i=1) si (A=S) atunci s:=e;
        altfel
            pop(α, Aj );
            push(β, Aj );
        sf_daca
        sf_daca
    Sf_daca
    Sf_daca
    Sf_daca
    Sf_cattimp
    Daca s=e atunci
    altfel
        tipareste "Eroare"
        tipareste "Seventa este acceptata"
        constructie_sir_prod(G, α);
    Sf_daca
    Sf_subalgoritm
```

•

### Subalgoritmul constructie\_sir\_prod( $G, \alpha$ ) este:

sir\_prod:=“”;

Cattimp not  $vida(\alpha)$  executa

    daca  $varf(\alpha)$  este de forma  $A_j$  atunci

        sir\_prod:= sir\_prod +  $indicativ\_rp(varf(\alpha))$ ;

    sf\_daca

    pop( $\alpha$ );

Sf\_Cattimp

Sf\_subalgoritm

Gramatici recursive → curs 5

# **Analiza sintactica descendenta.**

## **Gramatici si analiza LL(K)**

.

# FIRST<sub>k</sub>

- FIRST<sub>k</sub>: (N U Σ)\* → Part(Σ\*)
- FIRST<sub>k</sub>(α) = {u ∈ Σ\* | (α =\*> ux, |u| = k)  
sau (α =\*> u, |u| < k) }

# FOLLOW<sub>k</sub>

FOLLOW<sub>k</sub>: (N ∪ Σ)\* → Part(Σ\*)

FOLLOW<sub>k</sub>(α) = {u ∈ Σ\* | S = \* > γαβ, u ∈ FIRST<sub>k</sub>(β)}

.

# Operatia $\oplus_k$

=> ca putem descompune sirul w în două pari, x și y, astfel încât să putem concatena x cu orice z pentru a obține înapoi w.

Fie  $L_1, L_2 - 2$  limbaje peste  $\Sigma$

$$L_1 \oplus_k L_2 = \{w \mid \exists x \in L_1, y \in L_2 \text{ astfel încât:}$$

ex.  $w=acc$ ;  $x=a$ ,  $y=cc$ ,  $z=eps$   
deci  $xy=wz$ , deoarece  $a^*cc=acc$

fie:  $|w| = k$  și  $\exists z: xy = wz$

fie:  $|w| < k$  și  $xy = w$

}

---

$$\underline{\text{Exercitii:}} \quad L_1 = \{a, ab, abb, abbb, \dots\}$$

$$L_2 = \{c, cc, ccc, cccc, \dots\}$$

$$L_1 \oplus_3 L_2 = \dots \{acc, abcc, abbcc, abbbcc \dots\}$$

$$L_1 \oplus_1 L_2 = \dots$$

# Determinarea lui FIRST<sub>1</sub>

- pentru fiecare  $A \in N$ :

$$F_0(A) = \{a \in \Sigma \mid A \rightarrow a\alpha \in P\} \cup \{\varepsilon \mid A \rightarrow \varepsilon\}$$

sf.pentru

$i = 0$

- repeta

$$i = i + 1$$

pentru fiecare  $A \in N$ :

$$F_i(A) = \{a \in \Sigma \cup \{\varepsilon\} \mid A \rightarrow X_1 \dots X_k, a \in F_{i-1}(X_1) \oplus_1 \dots \oplus_1 F_{i-1}(X_k)\}$$

sf. pentru

pana cand  $F_i = F_{i-1}, \forall A \in N$

- FIRST<sub>1</sub> = F<sub>i</sub>

Obs:

$$\forall a \in \Sigma : \text{FIRST}_1(a) = \{a\}$$

$\forall A \in N$ : FIRST<sub>1</sub>(A) se determina pe baza reg. productie:  $A \rightarrow \dots$

# FOLLOW<sub>1</sub>

- $\text{FOLL}(\text{S}) = \{\$\}$   
 $\text{FOLL}(\text{X}) = \Phi \quad (\text{X} \neq \text{S})$
  - repeta
    - pentru fiecare  $B: A \rightarrow \alpha B \beta$  executa  
 $\text{FOLL}(B) = \text{FOLL}(B) \cup (\text{FIRST}_1(\beta) - \{\varepsilon\})$   
Daca  $\varepsilon \in \text{FIRST}_1(\beta)$  atunci  
 $\text{FOLL}(B) = \text{FOLL}(B) \cup \text{FOLL}(A)$   
sf. daca
    - sf. pentru
- pana cand FOLL nu se mai modifica
- $\text{FOLLOW}_1 = \text{FOLL}$

# Proprietati:

$$\text{FIRST}_1(\alpha\beta) = \text{FIRST}_1(\alpha) \oplus_1 \text{FIRST}_1(\beta)$$

$$\text{FIRST}_1(X_1 \dots X_n) = \dots ?$$

- $A \rightarrow \alpha$   
 $\text{FIRST}_1(A) \supset \text{FIRST}_1(\alpha)$
- $A \rightarrow \alpha X \beta$   
 $(\text{FIRST}_1(\beta) - \{\varepsilon\}) \subset \text{FOLLOW}_1(X)$
- $A \rightarrow \alpha X$   
 $\text{FOLLOW}_1(A) \subset \text{FOLLOW}_1(X)$

[... ]

# Exercitii: FIRST<sub>1</sub> si FOLLOW<sub>1</sub>

Problema:

- determinati FIRST<sub>1</sub> si FOLLOW<sub>1</sub> pentru toate terminalele si neterminalele gramaticii cu regulile de productie:

$S \rightarrow BA$	(1)
$A \rightarrow +BA$	(2)
$A \rightarrow \epsilon$	(3)
$B \rightarrow DC$	(4)
$C \rightarrow *DC$	(5)
$C \rightarrow \epsilon$	(6)
$D \rightarrow (S)$	(7)
$D \rightarrow a$	(8)

# Gramatici LL(k)

- analizoare LL(k)
  - analiza sintactica descendenta
  - secv. de intrare este citita de la stanga spre dreapta
  - se folosesc derivari de stanga

# Gramatichi LL(k)

- $G = (N, \Sigma, P, S)$   
?  $w=a_1a_2\dots a_n \in L(G)$   
 $S \Rightarrow_{st} \dots \Rightarrow_{st} a_1a_2\dots a_i A \alpha$   
(derivari de stanga)
- LL(k) : alegerea r.p. pentru a-l retranscrie pe A este unic determinata de:  $a_{i+1}a_{i+2}\dots a_{i+k}$
- terminologie:
  - parte inchisa:  $a_1a_2\dots a_i$
  - predictia (de lungime k):  $a_{i+1}a_{i+2}\dots a_{i+k}$
  - simbol de retranscris: A

# Gramatichi LL(k)

Definitie:

$G = (N, \Sigma, P, S)$  este LL(K):

- Daca:

- $S =^*_{st} w A \alpha \Rightarrow_{st} w \beta \alpha =^* w x$
- $S =^*_{st} w A \alpha \Rightarrow_{st} w \gamma \alpha =^* w y$
- $\text{FIRST}_k(x) = \text{FIRST}_k(y)$

- atunci:

$$\beta = \gamma$$

**k=1**

•

# Gramatici LL(1)

- ?  $w = a_1 a_2 \dots a_i a_{i+1} \dots a_n \in L(G)$   
 $S \Rightarrow_{st} \dots \Rightarrow_{st} a_1 a_2 \dots a_i A \alpha$
- LL(1): rescrierea lui A este unic determin. de  $a_{i+1}$

**De exemplu:**

**Aleg:**

$$A \rightarrow \gamma$$

$$A \rightarrow \epsilon$$

$$A \rightarrow \delta \quad (=^* > \epsilon)$$

**atunci cand:**

$$a_{i+1} \in \text{FIRST}_1(\gamma)$$

$$a_{i+1} \in \text{FOLLOW}_1(A)$$

$$a_{i+1} \in \text{FOLLOW}_1(A)$$

# Gramatici LL(1)

## Teorema:

$G$  – este de tip LL(1)

daca:

$$\forall A \in N \quad A - \text{neterminal}$$

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \dots \mid \alpha_m$$
 sunt diferite alternative ale productiilor neterminalului  $A$

- $\text{FIRST}_1(\alpha_i) \cap \text{FIRST}_1(\alpha_j) = \Phi$ ,  $i <> j$
- daca  $\exists i$  a.i.  $\alpha_i =^* \varepsilon$  atunci:  $\varepsilon \in \text{FIRST}_1(\alpha_i)$   
 $\text{FIRST}_1(\alpha_j) \cap \text{FOLLOW}_1(A) = \Phi$ ,  $i <> j$

# Analiza sintactica LL(1)

- se construieste tabelul de analiza LL(1)
  - alegerea lui  $A \rightarrow \alpha_i$  este indicata in tabel
- analizorul sintactic LL(1)
  - ~ modelare ca un **automat**

---

## Conventie:

**\$** : la orice cuvant din limbaj se adauga \$:

marcator de sfarsit de cuvant

## Observatie:

O gramatica este de tip LL(1) daca tabelul de analiza nu contine conflicte (nu exista mai mult de o valoare intr-o celula din tabel)

# Tabelul de analiza LL(1)

indica actiunea posibila la un moment dat

- coloane:  $\Sigma \cup \{\$\}$
- linii:  $N \cup \Sigma \cup \{\$\}$
- celula  $\Rightarrow$  (membrul drept al reg.prod , nr. reg.prod.)

# Tabelul de analiza LL(1)

$M(X,a) =$

- $(\alpha,i)$     daca  $X \rightarrow \alpha \in P$ ,  $a \in FIRST_1(\alpha)$   
                       $X \rightarrow \alpha - a$  i-a regula de productie
- $(\alpha,i)$     daca  $\epsilon \in FIRST_1(\alpha)$ ,  $a \in FOLLOW_1(X)$   
                       $X \rightarrow \alpha - a$  i-a regula de productie
- **pop**           $X = a$
- **acc**           $X = \$$ ,  $a = \$$
- **err**          in toate celelalte cazuri

# Analizorul LL(1)

- Automat:  $(\alpha, \beta, \Pi)$ 
  - banda de intrare:  $\alpha$  (stiva de intrare)
  - stiva  $\beta$  (stiva de lucru)
  - banda de iesire  $\Pi \Rightarrow$  sirul regulilor de productie
- config. initiala:  $(w\$, \$\$, \epsilon)$
- config. finala:  $(\$, \$\$, \Pi)$
- tranzitii
  - push  $(ax\$, A\beta, \Pi) \vdash (ax\$, \alpha\beta, \Pi i)$  dc.:  $M(A, a) = (\alpha, i)$
  - pop  $(ax\$, a\beta, \Pi) \vdash (x\$, \beta, \Pi)$
  - acc  $(\$, \$\$, \Pi) \vdash \text{acc}$
  - err in celelalte cazuri

# Transformari echivalente

- $\Rightarrow$  gram. LL(1) echiv. ?
- factorizare la stanga

...

	descendent	ascendent
cu reveniri / recursiv	descendent cu reveniri	<i>ascendent cu reveniri</i>
liniar	LL(k)	LR(k)

# Analizoare sintactice

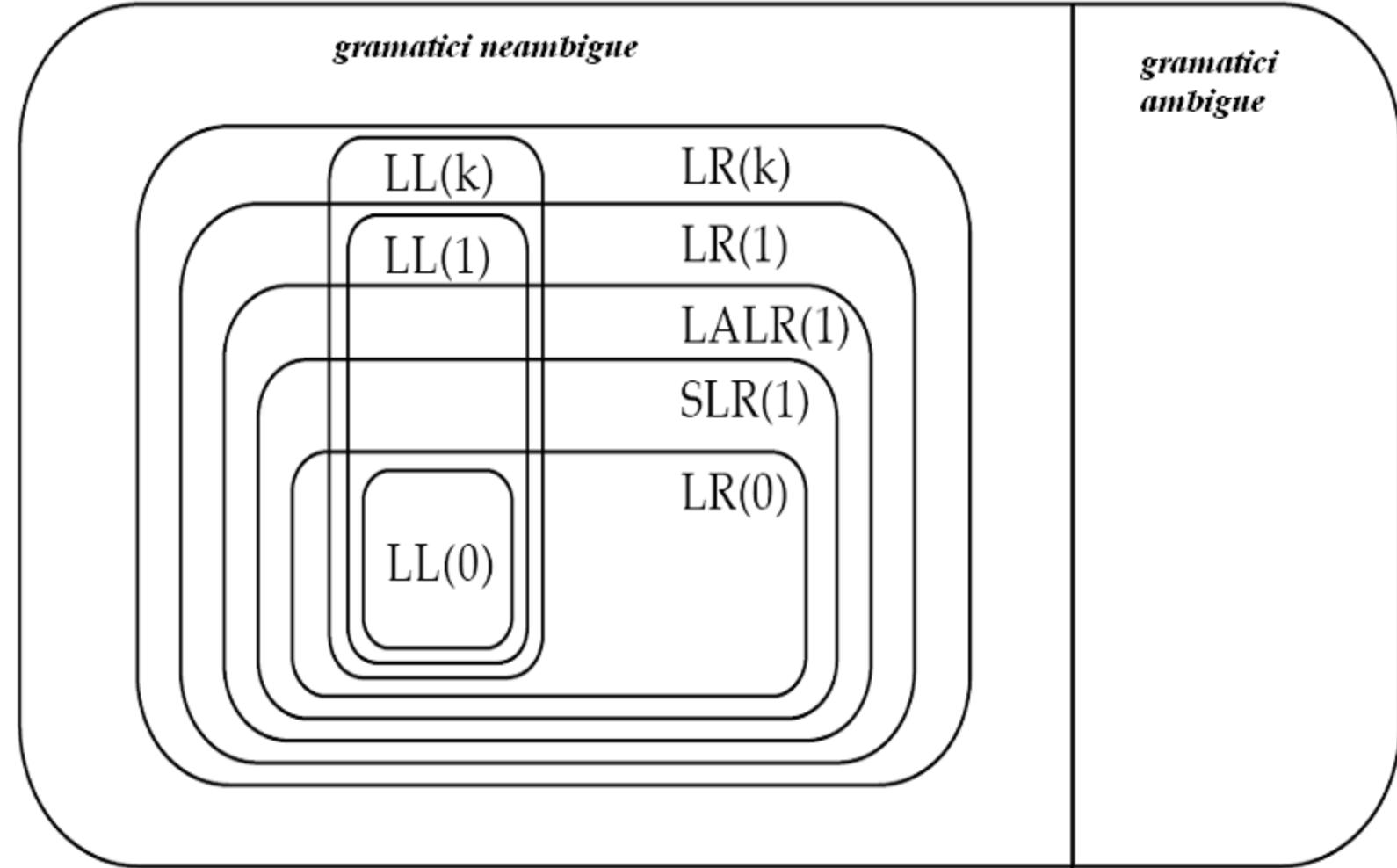
## LL(k)

- L: “Left-to-right” (scaneaza intrarea)
- L: “Left-most derivation”
- k simboluri de lookahead *descendent*  
~ *dezvoltare* in pre-ordine a arborelui de analiza sintactica

## LR(k)

- L: Scaneaza intrarea “Left-to-right”
- R: “Right-most derivation” *ascendent*
- k simboluri de lookahead  
~ *traversare* in post-ordine a arborelui de analiza sintactica

# LL vs LR



# Clase de lenguaje si relativos dintre ele

# Gramatici si relatii de precedenta

- Gramatici de precedenta simpla

- analiza sintactica ascendenta, determinista
- foloseste actiuni de tipul deplasare-reducere

Se folosesc niste relatii numite relatii de precedenta pentru a depista limita dreapta si a celei stanga pentru a face o reducere

- automatul este similar cu automatul LR

# Gramatici de precedenta simpla

**Definitie:**

gramatica de precedenta simpla

este o gramatica independenta de context propri

**(inclusiv  $\epsilon$ -independenta)**

unic invertibila:

- nu exista 2 reguli de productie cu acelasi membru drept
- intre oricare 2 simboluri exista cel mult o relatie de precedenta

# Gramatichi de precedenta simpla

Relatii de precedenta Wirth-Weber:

$X \stackrel{\bullet}{=} Y :$        $A \rightarrow \alpha XY\gamma \in P$

$X < \bullet Y :$        $A \rightarrow \alpha XB\gamma \in P , B = >^+ Y\gamma$

$X \bullet > a :$        $A \rightarrow \alpha BY\gamma \in P , B = >^+ \gamma X , Y = >^* a\delta$

$\$ < \bullet X :$        $S = >^+ X\alpha$

$X \bullet > \$ :$        $S = >^+ \alpha X$

$$R_{<\bullet} \subset (N \cup \Sigma \cup \{ \$ \}) \times (N \cup \Sigma \cup \{ \$ \ })$$

$$R_{\stackrel{\bullet}{=}} \subset (N \cup \Sigma) \times (N \cup \Sigma)$$

$$R_{\bullet >} \subset (N \cup \Sigma \cup \{ \$ \ }) \times (\Sigma \cup \{ \$ \ })$$

Adevarat  
sau fals?

# Gramatici de precedenta simpla

Exemplu:

$$S \rightarrow aSSb$$

$$S \rightarrow c$$

---

$$? accb \in L(G)$$

	S	a	b	c	\$
S	= .	<.	= .	<.	
a	= .	<.		<.	
b		›	›	›	›
c		›	›	›	›
\$		<.		<.	

Problema este programata a fi rezolvata complet la seminar.

# Gramatici LR(k)

- analizoare LR(k)
  - analiza sintactica ascendentă
  - secv. de intrare este citita de la stanga spre dreapta
  - se folosesc derivari de dreapta

metoda: deplasare - reducere

# Analiza sintactica ascendenta

Exemplu:

Cum “arata” analiza sintactica ascendenta pentru gramatica:

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow a \quad (2)$$

$$B \rightarrow b \quad (3)$$

si intrarea: ab ?

# Gramatica LR(K)

Analizoarele sintactice LR(k) lucreaza cu  
gramatica imbogatita:

$$G' = (N \cup \{S'\}, \Sigma, P \cup \{ S' \rightarrow S \} , S')$$
$$(S' \notin N)$$

pentru a evita ca simbolul de start sa apara in  
membrul drept al unei reguli de productie.

# Gramatica LR(K)

O gramatica  $G = (N, \Sigma, P, S)$

este de tip LR(k) pentru  $k \geq 0$

daca din:

- $S' \stackrel{*}{\Rightarrow}_{dr} \alpha A w \Rightarrow_{dr} \alpha \beta w$
- $S' \stackrel{*}{\Rightarrow}_{dr} \gamma B x \Rightarrow_{dr} \alpha \beta y$
- $\text{FIRST}_k(w) = \text{FIRST}_k(y)$

rezulta ca:

- $A = B$
- $x = y$
- $\alpha = \gamma$

# Gramatici LR(K) - terminologie

## Prefix viabil

Fie:  $S =^* \Rightarrow_{dr} \alpha A w \Rightarrow_{dr} \alpha \beta w$

Orice prefix al lui  $\alpha\beta$  se numeste prefix viabil

## Element de analiza LR(k)

se defineste ca fiind:  $[A \rightarrow \alpha.\beta, u]$

unde  $A \rightarrow \alpha\beta \in P$  si  $u \in \Sigma^k$  u-predictie

## Element de analiza valid

$[A \rightarrow \alpha.\beta, u]$  valid pentru prefixul viabil  $\gamma\alpha$  daca:

- $S =^* \Rightarrow_{dr} \gamma A w \Rightarrow_{dr} \gamma \alpha \beta w$
- $u = \text{FIRST}_k(w)$

# Analizor sintactic LR(K)

Vom studia:

- LR(0)
- SLR
- LR(1)
- LALR

Pasi in analiza LR(k):

- gramatica imbogatita
- constructia colectiei canonice
- constructia tabelului de analiza
- analiza: → automat

Vom lucra astfel:

la multimea cuv. de analizat se adauga la sfarsit \$

\$ - marcator de sfarsit de cuvant

# Colectia canonica LR(K)

$C = \{I_i - \text{elementele de analiza pentru un prefix viabil}\}$

- in  $I_0$  avem un prim element de analiza
- am cel putin un element in  $I_j$  *(pentru fiecare)*  
=> adaug altele: functia *Closure*
- am o multime  $I_j$  *(pentru fiecare)*  
=> construiesc multimile *goto*( $I_i, X$ )

---

Observatie:  $I_i$  corespunde unei stari a automatului

Notatie:  $E$  – multimea elementelor de analiza

# Constructia colectiei canonice LR(k)

$C = \{ I_i \text{-elementele de analiza pentru un prefix viabil} \}$

in  $I_0$  avem:  $[S' \rightarrow .S, \dots]$

- $I_0 = Closure([S' \rightarrow .S, \dots])$
- $C = \{ I_0 \}$
- repeta
  - pentru toti  $I_i$  din  $C$ ,  $X \in (N \cup \Sigma)$  executa
$$C = C \cup goto(I_i, X)$$
  - sf. pentru
  - pana cand  $C$  nu se mai modifica

**K=0:**

**LR(0)**

# Gramatica imbogatita

- se adauga  $S'$ 
    - nou simbol de start
    - $S' \rightarrow S$
- 

## Colectia canonica:

- In  $I_0$  avem: [  $S' \rightarrow .S$  ]
- ...

Notam:  $\mathcal{E}$  – multimea elementelor de analiza

- $\text{Closure} : \text{Part}(\mathcal{E}) \rightarrow \text{Part}(\mathcal{E})$
- Fie:  $e \in \mathcal{E}$

daca  $e = [A \rightarrow \alpha . B\beta]$

atunci  $\forall B \rightarrow \delta \in P: [B \rightarrow . \delta] \in \text{Closure}(e)$

# Functia *goto*

**LR(0)**

- $goto : \text{Part}(E) \times (N \cup \Sigma) \rightarrow \text{Part}(E)$
- $goto(I, X) = \text{Closure}(\{[A \rightarrow \alpha X. \beta] \mid [A \rightarrow \alpha . X \beta] \in I\})$

# Tabelul de analiza LR(0)

		goto $N \cup \Sigma$
actiune		
$I_0$	deplasare (s)	
$I_1$	reducere (nr. r.p.)	
:	acceptare (acc)	
	eroare	

# Tabelul de analiza LR(0)

$T(I_i, \text{actiune}) =$

– s (shift, deplasare)

daca:  $[A \rightarrow \alpha.\beta] \in I_i, \beta \not\sim \epsilon$

si:  $T(I_i, X) = I_j$ , daca  $I_j = goto(I_i, X)$

– L (reducere cu r.p. nr. L)

daca  $[A \rightarrow \alpha.] \in I_i$

$A \rightarrow \alpha \in P$  : regula de prod. cu numarul L

si:  $T(I_i, X)$  nu se completeaza

– acc daca:  $[S' \rightarrow S.] \in I_i$

Toate celelalte cazuri se considera eroare .

# Automatul LR(0) – model matematic

- configuratie:  
 $(\alpha, \beta, \Pi)$   
(stiva\_de\_lucru, banda\_de\_intrare, banda\_de\_iesire)
- pe stiva: prefixe viabile, stari ale analizorului
- config. initiala:  $(\$0, w \$, \varepsilon)$
- config. finala:  $(\$0S I_{acc}, \$, \Pi)$

# Automatul LR(0) – model matematic

## Tranzitii

- **deplasare:**

$$(\$, \gamma s_k, a_i..a_n \$, \Pi) \vdash (\$, \gamma s_k a_i s_m, a_{i+1}..a_n \$, \Pi)$$

daca:  $T(s_k, \text{actiune}) = s$  si  $T(s_k, a_i) = s_m$

- **reducere:**

$$(\$, \gamma s_{p-1} X_p s_p ... X_k s_k, a_i..a_n \$, \Pi) \vdash (\$, \gamma s_{p-1} A s_m, a_i..a_n \$, L\Pi)$$

daca:  $T(s_k, \text{action}) = L$

si:  $A \rightarrow X_p ... X_k$  – r.p. cu nr.  $L$

$T(s_{p-1}, \text{actiune}) = s$

$T(s_{p-1}, A) = s_m$

- **acceptare:**

$$(\$, 0S s_{\text{acc}}, \$, \Pi) \vdash \text{acc.}$$

- **eroare:**

*orice alta situatie*

# Gramatica LR(0)

O gramatica este LR(0)

daca tabelul de analiza nu contine conflicte.

---

Gramatica data prin urmatoarele r.p. este LR(0) ?

$$S \rightarrow Ax$$

$$S \rightarrow By$$

$$A \rightarrow a$$

$$B \rightarrow a$$

**K=1: SLR, LR(1), LALR**

# Analiza sintactica SLR

- SLR = Simple LR
- element de analiza SLR:  
 $[A \rightarrow \alpha.\beta, u]$   
 $u = FOLLOW_1(A)$   
 $|u| = 1$
- constructia colectiei canonice ( $\sim LR(0)$ )
  - $[A \rightarrow \alpha.\beta, u]$  ,  $u = FOLLOW_1(A)$
- SLR tine cont de predictie numai pentru reducere

# Analiza sintactica SLR

- constructia tabelului de analiza SLR
  - actiunea de reducere depinde de predictia  $u$   
 $\Rightarrow$  reducerea va avea o coloana pentru fiecare  $a \in \Sigma$
- tabelul:
  - linii: elementele colectiei canonice
  - coloane:  $N \cup \Sigma \cup \{\$\}$
  - celula:  $s_{\text{stare}}, r_{\text{nr.r.p}}, \text{acc}$
- analizorul ~ analizorul pt. LR(0)
  - automat: configuratii si tranzitii

# Analiza sintactica SLR

In tabelul de analiza SLR vom avea:

actiune: **reducere** + **deplasare**  
(**goto**)

$X \in \Sigma \cup \{\$\}$        $X \in N \cup \Sigma$

linii: elementele colectiei canonice

coloane:  $N \cup \Sigma \cup \{\$\}$

# Gramatica SLR

O gramatica este SLR

daca tabelul de analiza nu contine conflicte.

---

Gramatica data prin urmatoarele r.p. este SLR ?

$S \rightarrow A$	1
$S \rightarrow xb$	2
$A \rightarrow aAb$	3
$A \rightarrow B$	4
$B \rightarrow x$	5

- Gramatica data prin urmatoarele r.p. este SLR ?

$S \rightarrow A$	1
$S \rightarrow xb$	2
$A \rightarrow aAb$	3
$A \rightarrow B$	4
$B \rightarrow x$	5

# Analizor sintactic LR(1)

- imbogatirea gramaticii
- constructia colectiei canonice

element de analiza LR(1):

- $[A \rightarrow \alpha.\beta, u], |u| = 1$
- constructia tabelului de analiza
- analiza: automat

# Colectia canonica LR(1)

- elem. initial

$$[S' \rightarrow .S, \$]$$

- *Closure*

$$[A \rightarrow \alpha.B\beta, a] \Rightarrow [B \rightarrow .\gamma, b] \in \text{Closure}([A \rightarrow \alpha.B\beta, a])$$

$$B \rightarrow \gamma \quad \forall b \in \text{FIRST}_1(\beta a)$$

- *goto*

$$\text{goto}(I, X) =$$

$$\text{Closure}(\{[A \rightarrow \alpha X. \beta, a] \mid [A \rightarrow \alpha X \beta, a] \in I\})$$

# Tabelul LR(1)

**shift + reduce**

**stare**

**N ∪ Σ ∪ \$**

	I <sub>0</sub>	
(I <sub>acc</sub> )	I <sub>1</sub>	
	I <sub>2</sub>	
	:	
	:	

# Construirea tab. de analiza LR(1)

- $[A \rightarrow \alpha.X\beta, b] \in I_i$  :  $goto(I_i, X) = I_j \leqslant$  functia  $goto$   
 $action(I_i, X) = s_j$
- $[A \rightarrow \alpha. . , a] \in I_i$   $action(I_i, a) = r_L$   
 $L$  – nr. reg. de productie:  $A \rightarrow \alpha$   
 $A \diamond S'$
- $[S' \rightarrow S. . , \$] \in I_i$   $action(I_i, \$) = acc$

Obs: o gram. este LR(...) daca tabelul de analiza nu contine conflicte; si reciproc

# Analizorul LR(1)

pe baza tabelului de analiza  
→ similar LR(0), SLR

# Analizor sintactic LALR

- $[A \xrightarrow{\text{nucleu}} \alpha.\beta, a]$
- colectia canonica LR(1)
- fuzioneaza elementele de analiza cu nuclee identice si care nu creeaza conflicte
- predictia: reunirea predictiilor
- tabelul LALR & analiza : similar LR(1)

# LR (1 –uri)

- Conflict:

[  $A \rightarrow \alpha_1.a\alpha_2$  , u ]

deplasare-reducere

[  $B \rightarrow \beta_1. \quad , \quad a$  ]

[  $A \rightarrow \alpha_1. \quad , \quad a$  ]

reducere-reducere

[  $B \rightarrow \beta_1. \quad , \quad a$  ]



Vezi și resursa bibliografică:

S. MOTOGNA

Metode de proiectare a compilatoarelor, 2006

# Analiza LR(1)

$S' \rightarrow S$  0

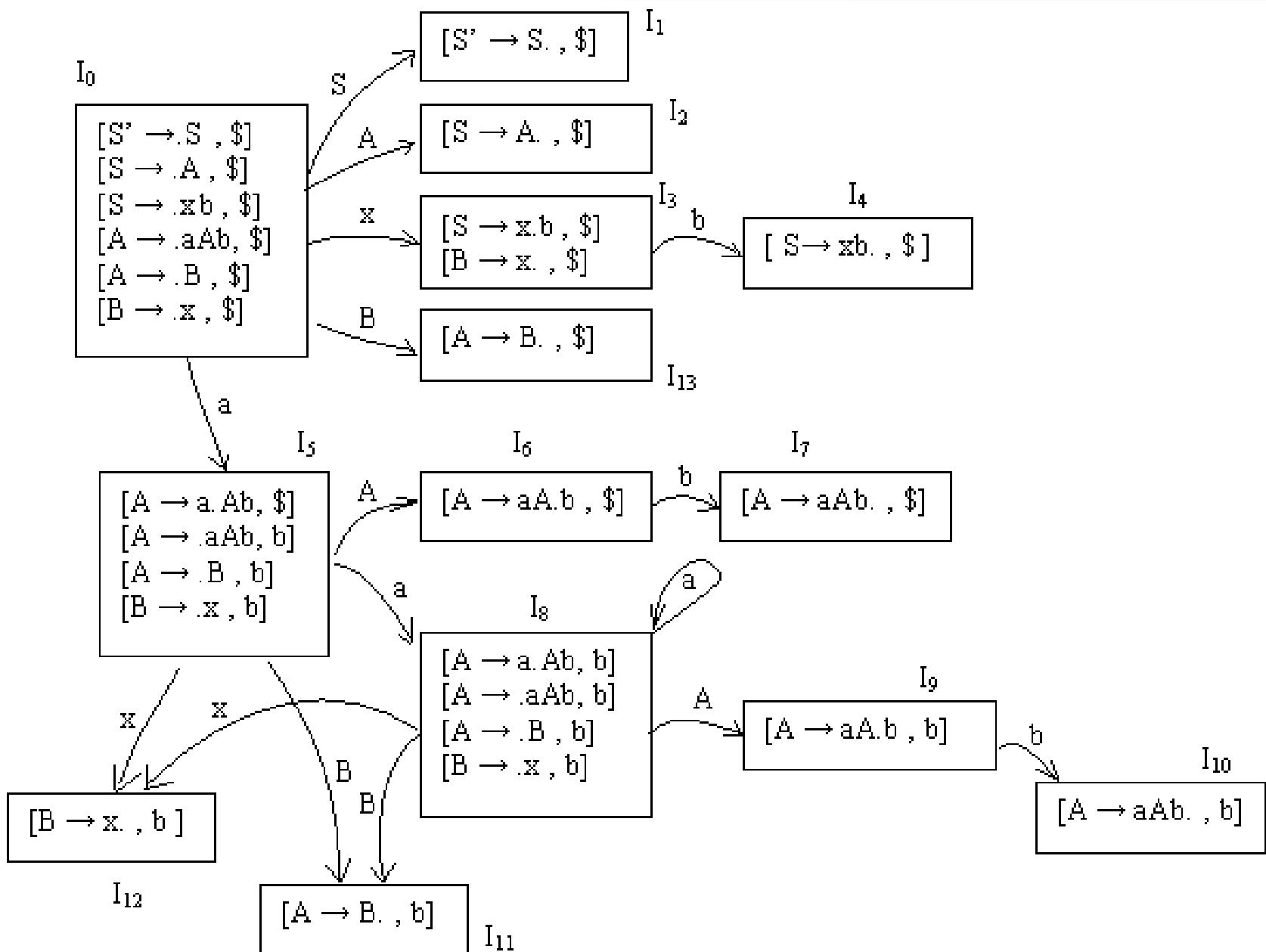
$S \rightarrow A$  1

$S \rightarrow xb$  2

$A \rightarrow aAb$  3

$A \rightarrow B$  4

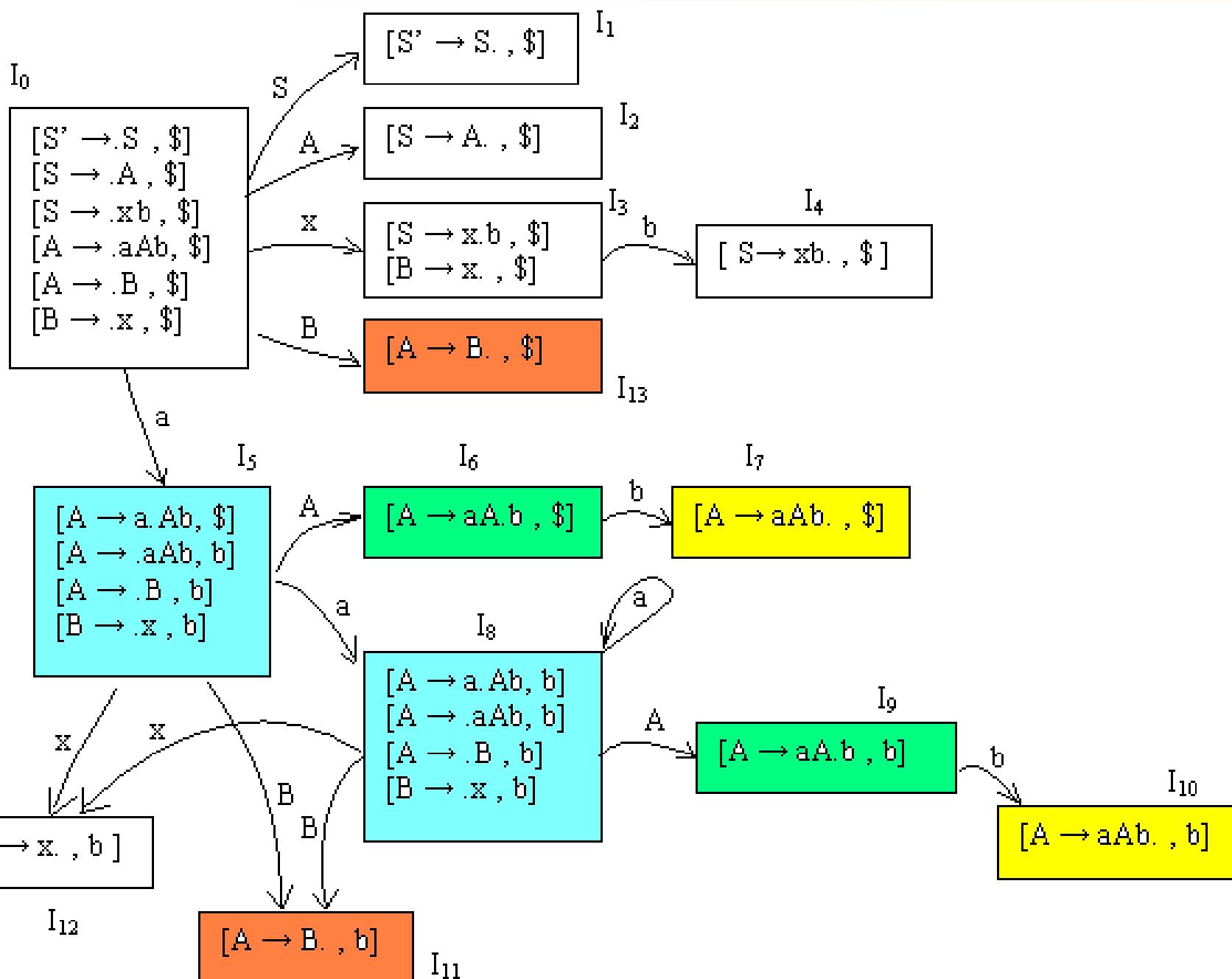
$B \rightarrow x$  5

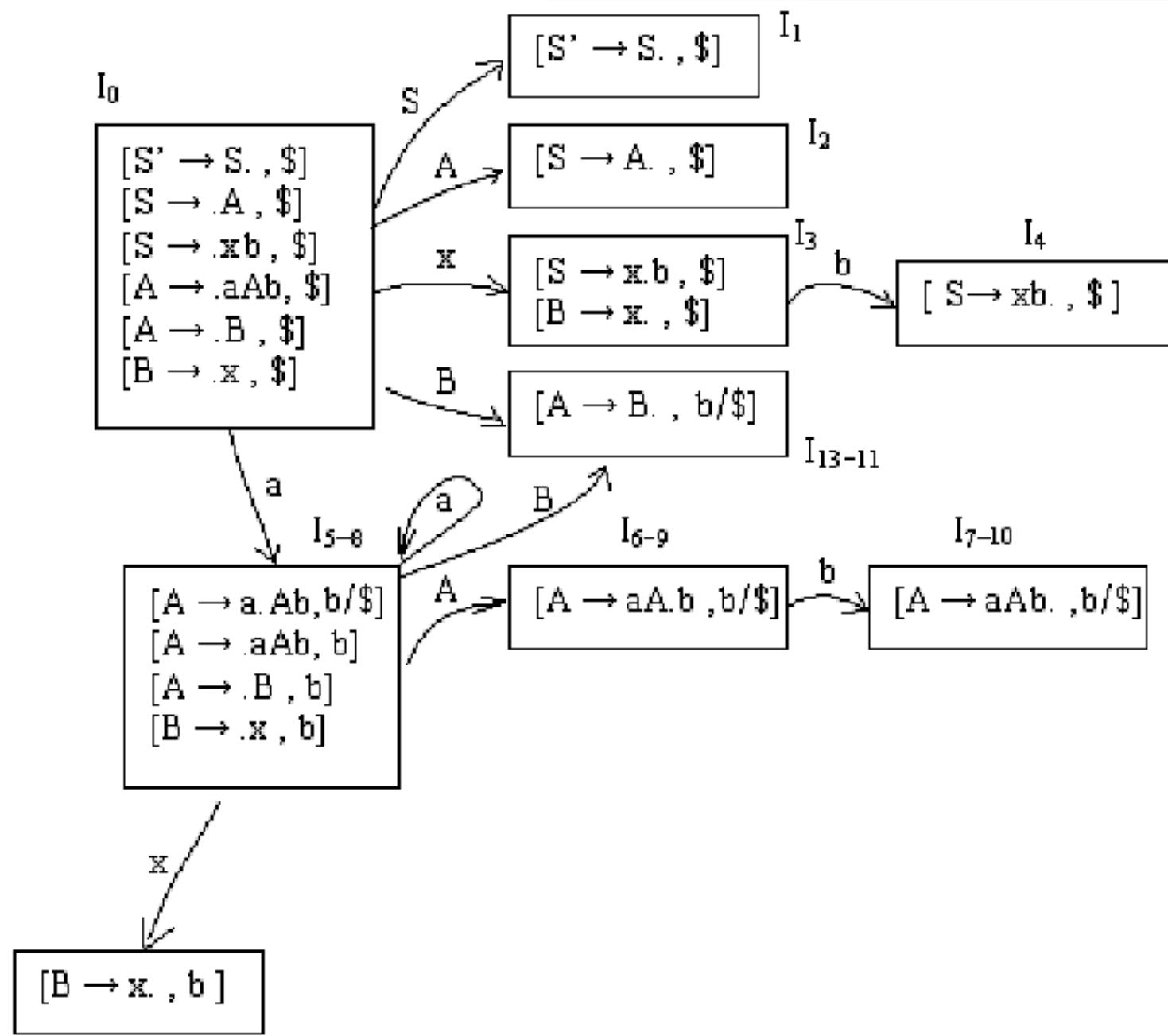


	<i>S</i>	<i>A</i>	<i>B</i>	<i>a</i>	<i>b</i>	<i>x</i>	<i>s</i>
I <sub>0</sub>	s1	s2	s13	s5		s3	
I <sub>1</sub>							acc
I <sub>2</sub>							r1
I <sub>3</sub>					s4		r5
I <sub>4</sub>							r2
I <sub>5</sub>		s6	s11	s8		s12	
I <sub>6</sub>					s7		
I <sub>7</sub>							r3
I <sub>8</sub>		s9	s11	s8		s12	
I <sub>9</sub>					s10		
I <sub>10</sub>						r3	
I <sub>11</sub>						r4	
I <sub>12</sub>						r5	
I <sub>13</sub>							r4

# Analiza LALR

→





	S	A	B	a	b	x	\$
I <sub>0</sub>	s1	s2	s11-13	s5-6		s3	
I <sub>1</sub>							acc
I <sub>2</sub>							r1
I <sub>3</sub>					s4		r5
I <sub>4</sub>							r2
I <sub>5-8</sub>		s6-9	s11-13	s5-8		s12	
I <sub>6-9</sub>					s7-10		
I <sub>7-10</sub>					r3		r3
I <sub>11-13</sub>					r4		r4
I <sub>12</sub>					r5		

# G – gramatica de atrbute

$G = (N, \Sigma, P, S)$  – gram. independenta de context

- $\mathcal{A} = \bigcup_{X \in N \cup \Sigma} \mathcal{A}(X)$ 
  - fiecarui simbol al gramaticii  
i se asociaza 0 sau mai multe atrbute  
: multime finita de atrbute
- $\mathcal{R} = \bigcup_{p \in P} \mathcal{R}(p)$ 
  - fiecarei reguli de productie i se asociaza  
o multime finita de expresii ale atrbutelor  
asociate simbolurilor regulii de productie  
=> reguli de evaluare ale atrbutelor

# Atribute

asocierile atribut - valoare

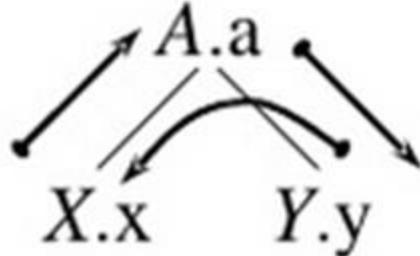
sunt definite numai peste o “analiza sintactica”

un arbore de derivare

## Evaluator de atribute

- calculeaza valori & propaga valorile calculate
- traverseaza arborele de derivare
- strategie de traversare a arborelui  
si propagare a valorilor
-

# Evaluarea atributelor



$$\begin{aligned}A.a &= f(X.x) \\X.x &= f(Y.y) \\Y.y &= f(A.a)\end{aligned}$$

Dandu-se o gram. de attribute, ce se intampla daca exista arbori de derivare pentru care graful este circular?

- ! restrictionari pt. regulile de calcul ale atributelor

Dacă un atribut **b** depinde de un alt atribut **c**, atunci regula semantică pentru calculul atributului **b** trebuie să fie evaluată după regula semantică care îl produce pe **c**

→ Graful de dependenta

(sortare topologica)

# Evaluarea atributelor

## Metode de evaluare

- metode bazate pe arborele de derivare
  - determină ordinea de evaluare
  - pe baza sortării topologice a grafului de dependență construit pentru arborele de derivare
  - pentru fiecare secvență de intrare
- metode bazate pe reguli
  - ordinea de evaluare este determinată / fixată la nivelul la care se definesc regulile semantice
- metode bazate pe o ordine pre-fixată
  - ordinea de evaluare este fixată și regulile semantice trebuie definite astfel încât să respecte ordinea data

# Atribute

Fie: regulile de evaluare a atributelor asociate urmatoarei reguli de productie:

$$A \rightarrow X_1 \dots X_k$$

- **atribut sintetizat:**
  - un atribut al lui A
  - regula de evaluare atribuie valoare atributului lui A
- **atribut mostenit:**
  - atribut al lui  $X_i$
  - regula de evaluare atribuie valoare atributului lui  $X_i$

El depinde de valorile parintilor si fratilor.

# Gramatica S-atributata

**Def:**

există doar attribute sintetizate  
și acestea depind de valorile atributelor copiilor

**Evaluarea atributelor**

- parcurgere "in sus" a arborelui de analiza sintact.
- analizor sintactic ascendent

# Gramatica L-atributata

## Def:

Pentru orice regula de productie:  $A \rightarrow X_1 X_2 \dots X_n$

- un atribut mostenit a lui  $X_i$  depinde de atribute mostenite ale lui A si de atribute ale lui  $X_1, X_2, \dots, X_{i-1}$
- orice atribut sintetizat al lui A nu depinde de alte atribute sintetizate ale lui A

## Evaluarea atributelor

- in stransa legatura cu parcurgerea arborelui de derivare

Subalg. viziteaza(A)

pentru fiecare descendant  $X_i : (X_1, X_2, \dots, X_n)$   
evaluateaza atributele mostenite ale lui  $X_i$   
viziteaza ( $X_i$ )

sf. pentru  
evaluateaza atributele sintetizate ale lui A

endSubalg.

## Gramatică de attribute (GA)

- gramatica independentă de context
- attribute + expresii ale atributelor



in unele surse se foloseste acelasi termen

## Definiții dirijate de sintaxă (DDS)

(EN: Syntax directed definition)

- gramatica independentă de context
- attribute + reguli de calcul ale atributelor

➤ pot avea efecte laterale

(apeluri de proceduri sau fragmente de program)

## Scheme de traducere:

(EN: Syntax directed translation)

definire orientată sintaxă

+ alte acțiuni

ex.: fragment de program

se execută atunci când este întâlnit în parcurgerea arborelui

EX:  $A \rightarrow \alpha \{ \text{print('x')} \} \beta$

se va afișa caracterul 'x' după ce se vizitează subarborele  $\alpha$   
și înainte de traversarea subarborelui  $\beta$ .

# Gramatica de atribute. Exemplu

$$S \rightarrow A \qquad \qquad A.a = A.x$$

$$\begin{array}{ll} A_0 \rightarrow A_1.a & A_1.a = A_0.a \\ & A_1.b = A_1.y \\ & A_0.x = A_1.x \\ & A_0.y = 1 \end{array}$$

$$\begin{array}{ll} A \rightarrow b & A.y = A.a \\ & A.x = 1 \end{array}$$

$$\begin{array}{ll} A \rightarrow bb & A.x = A.b \\ & A.y = 1 \end{array}$$

# Automate

- Recapitulare, Exemple, Aplicatii
- Translatoare
- Masini Turing

# Automat finit: model fizic

banda de intrare

	a1	a2	a3	...	an	
--	----	----	----	-----	----	--



stari

# Automat finit: model matematic

- Un *automat finit* este un ansamblu  
 $M = (Q, \Sigma, \delta, q_0, F)$  :
- $Q$  – alfabetul starilor
- $\Sigma$  – alfabet de intrare
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  functie de tranzitie
- $q_0 \in Q$  - stare initială
- $F \subseteq Q$  multimea stărilor finale



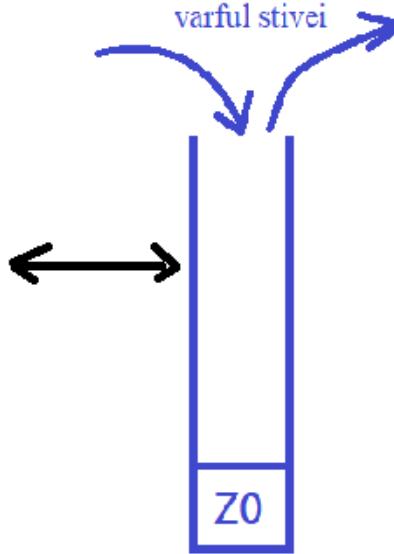
# Automat push-down (APD)

banda de intrare



cap de  
citire

directie de deplasare



stari

$z_0$

stiva

# Automat push-down (APD)

$$M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)$$

- $Q$  alfabetul stărilor;
- $\Sigma$  alfabetul de intrare;
- $\Gamma$  alfabetul memoriei stivă;
- $q_o \in Q$  stare inițială;
- $Z_o \in \Gamma$  simbolul de start al memoriei stivă;
- $F \subseteq Q$  multimea stărilor finale;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$  funcția de tranziție
  - are ca valori submultimi finite din  $Q \times \Gamma^*$  (posibil multimea vida)

# Transformarea acceptarii dupa criteriul stivei vide in acceptare dupa criteriul starii finale

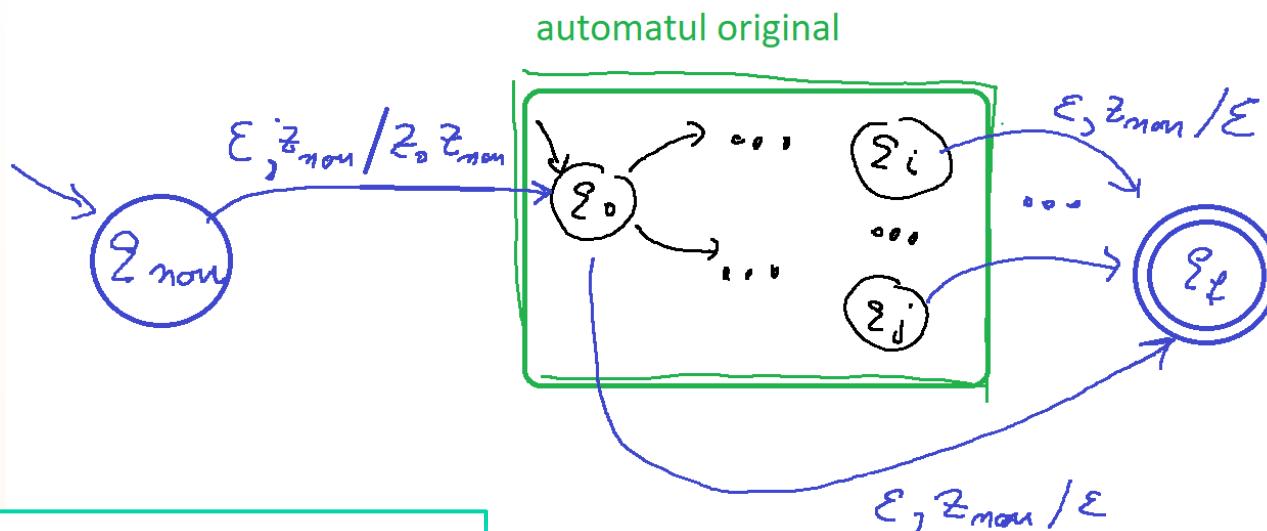
Fie: M – APD care accepta L dupa criteriul stivei vide

Dorim sa construim M' care accepta L dupa criteriul starii finale

M' – contine tot ce contine M , dar mai adaugam:

- $q_{nou}$  o noua stare initiala
- $q_f$  o noua starea finala
- $Z_{nou}$  noul simbol initial al stivei

Adaugam si tranzitiile corespunzatoare noilor stari, asa cum este schitat mai jos:



In mod analog se poate face transformarea acceptarii dupa criteriul starii finale in acceptare dupa criteriul stivei vide.

# Translator finit

$$M = (Q, \Sigma, D, \delta, q_0, F)$$

- $Q$  alfabetul stărilor;
- $\Sigma$  alfabetul de intrare;
- $D$  alfabetul de ieșire;
- $q_0 \in Q$  stare inițială;
- $F \subseteq Q$  mulțimea stărilor finale;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}_0(Q \times D^*)$  multimea partilor finite

# Translator finit

banda de intrare



cap de  
citire

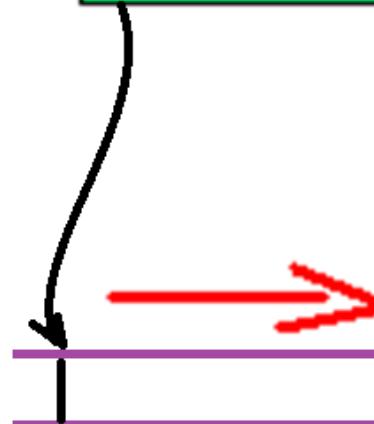


directie de deplasare

UC

stari

cap de  
scriere



banda de iesire

# Translator finit

Exemplu:

$$M = (Q, \Sigma, D, \delta, q_0, F)$$

$$M = (\{q_0, q_1\}, \{a\}, \{b\}, \delta, q_0, \{q_1\})$$

$$\delta(q_0, a) = \{q_1, \{b\}\}$$

$$d(q_1, \varepsilon) = \{q_1, \{b\}\}$$

---

Translatarea definită de M:

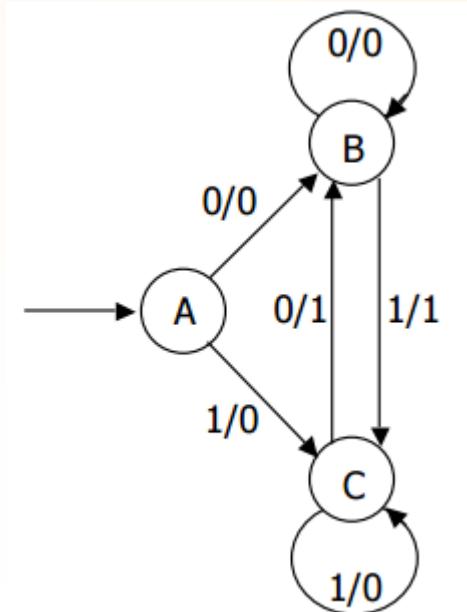
$$T(M) = \{(x, y) \mid x \in \Sigma^*, y \in D^*, (q_0, x, \varepsilon) \xrightarrow{*} (q, \varepsilon, y), q \in F\}$$

# Alte automate

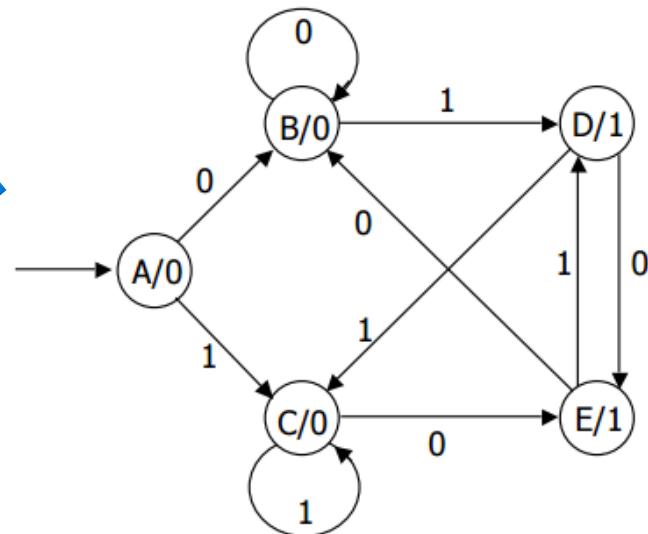
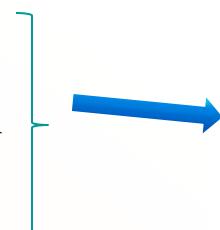
In literatura mai gasim:

(intre multe alte tipuri)

- **Automatele Mealy**: la fiecare tranzitie se produce un simbol de ieșire.



- **Automatele Moore**: fiecarei stari (intrare intr-o stare) i se asociaza un simbol de ieșire.

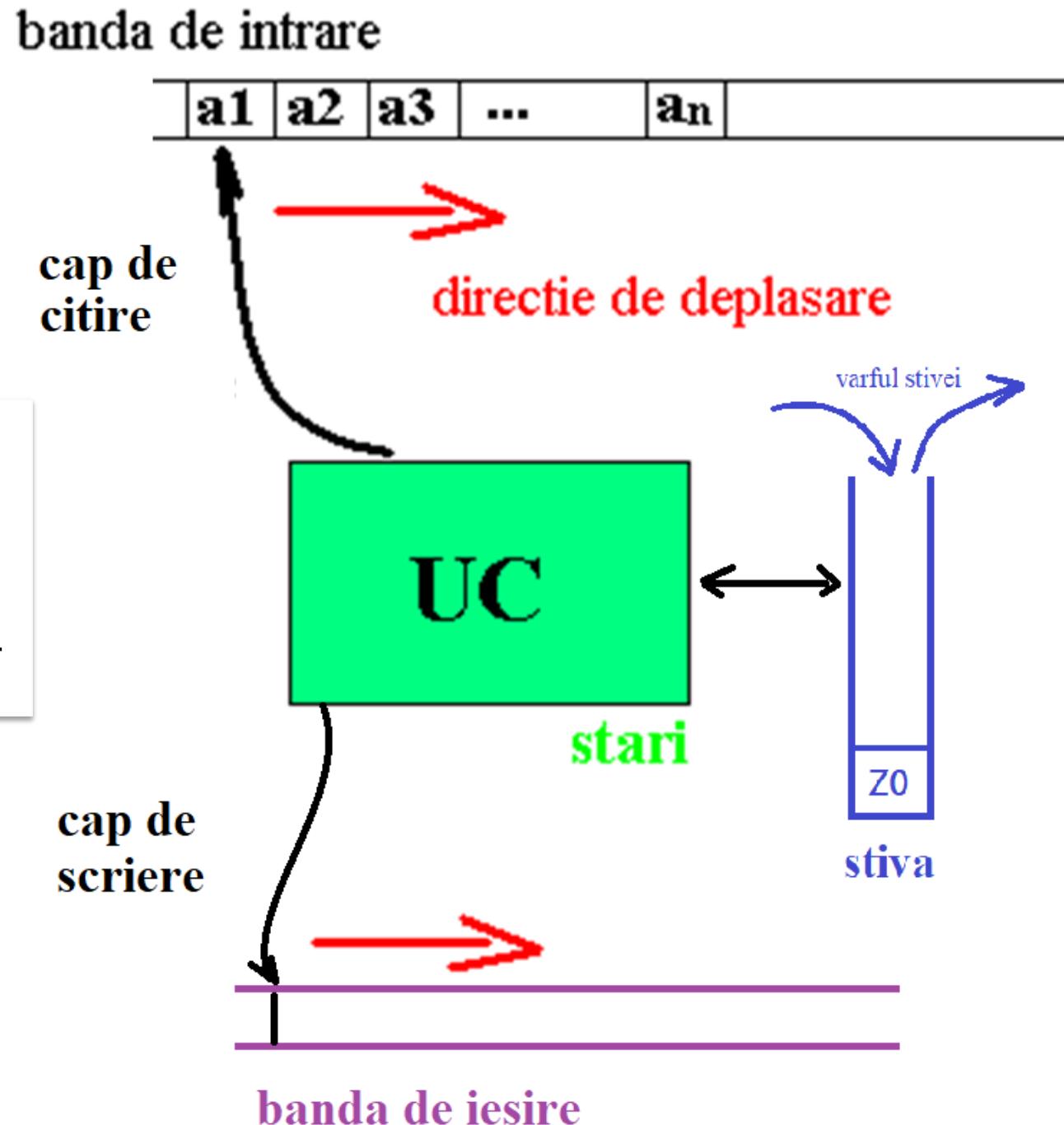


# Translator push-down

$$M = (Q, \Sigma, \Gamma, D, \delta, q_0, Z_0, F)$$

- $Q$  alfabetul stărilor;
- $\Sigma$  alfabetul de intrare;
- $\Gamma$  alfabetul memoriei stivă;
- $D$  alfabetul de ieșire;
- $q_0 \in Q$  stare inițială;
- $Z_0 \in \Gamma$  simbolul de start al memoriei stivă;
- $F \subseteq Q$  multimea stărilor finale;
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_0(Q \times \Gamma^* \times D^*)$  multimea partilor finite

# Translator push-down



# Translator push-down

$$M = (Q, \Sigma, \Gamma, D, \delta, q_0, Z_0, F)$$

$$Q = \{q\}$$

$$\Sigma = \{a, +, *\}$$

$$\Gamma = \{E, +, *\}$$

$$D = \{a, +, *\}$$

$$q_0 = q$$

$$Z_0 = E$$

$$\delta(q, a, E) = \{(q, \varepsilon, a)\}$$

$$\delta(q, +, E) = \{(q, EE+, \varepsilon)\}$$

$$\delta(q, *, E) = \{(q, EE^*, \varepsilon)\}$$

$$\delta(q, \varepsilon, +) = \{(q, \varepsilon, +)\}$$

$$\delta(q, \varepsilon, *) = \{(q, \varepsilon, *)\}$$

Considerand criteriul stivei vide,  
descrieti translatarea pe care acesta o defineste .

... am lucrat  
si cu alte  
translatoare



Vezi:

LL(1)

LR(\*)

# Ne reamintim: Analizorul LL(1)

- Automat:  $(\alpha, \beta, \Pi)$ 
  - banda de intrare:  $\alpha$
  - stiva  $\beta$  (stiva de lucru)
  - banda de iesire  $\Pi \Rightarrow$  sirul regulilor de productie
- config. initiala:  $(w\$, \$\$, \varepsilon)$
- config. finala:  $(\$, \$, \Pi)$
- tranzitii
  - push  $(ax\$, A\beta, \Pi) \vdash (ax\$, \alpha\beta, \Pi i)$  dc.:  $M(A, a) = (\alpha, i)$
  - pop  $(ax\$, a\beta, \Pi) \vdash (x\$, \beta, \Pi)$
  - acc  $(\$, \$, \Pi) \vdash \text{acc}$
  - err in celelalte cazuri

# Automatul LL(1) ca translator push-down (modificat)

**Translatorul push-down modificat** este:

[Moldovan]

$$M = (\{q\}, \Sigma, \Gamma, D, \delta, q, S, \emptyset)$$

$\Sigma$ - alfabet de intrare și este același cu alfabetul  $\Sigma$  din  $G$ ;

$D$  - alfabet de ieșire,  $D = \{1, 2, \dots, m\}$

$i \in D$  reprezintă nr. de ordine al producțiilor din  $P$ ,  $i : A \rightarrow \alpha$ ,  $i = \overline{1, m}$

$\Gamma$  - alfabetul memoriei push-down  $\Gamma = N \cup \Sigma$

$q$  - starea internă (stare inițială)

$S$  – simbolul de start în memoria push-down,  $S \in \Gamma$

$\delta$  – funcția de tranziție modificată

$$\delta : \Sigma \times \Gamma \rightarrow P(\Gamma^* \times (D \cup \{\varepsilon\}) \times \{0,1\})$$

$$\delta(a, a) = \{(\varepsilon, \varepsilon, 1)\} \quad \forall a \in \Sigma$$

$$\delta(a, A) = \{(x, i, 0)\}, a \in \Sigma, A \in N$$

dacă  $(\exists)i : A \rightarrow x$  și dacă  $a \in \varphi(x)$ , sau

dacă  $\varepsilon \in \varphi(x) \Rightarrow a \in \varphi(A)$

În celelalte cazuri  $\delta(., .) = \emptyset$

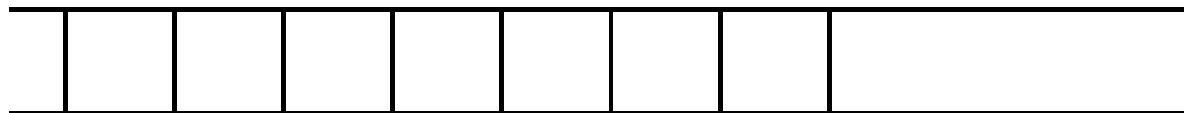
1 : semnifică înaintarea benzii cu o poziție;

0 : semnifică staționarea benzii.

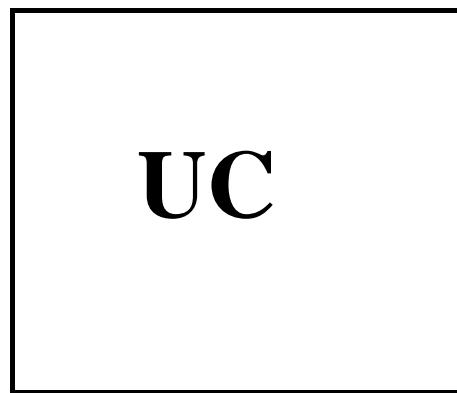
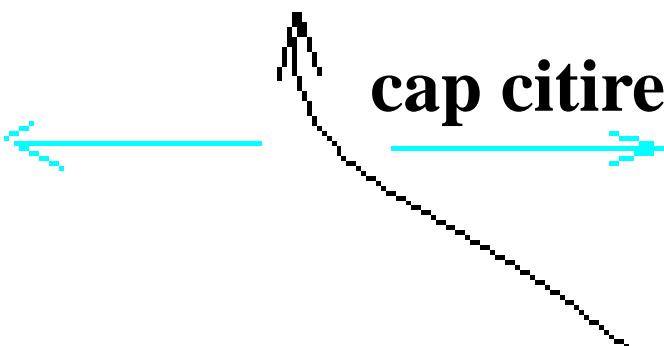
Limbajul acceptat  
se defineste dupa  
criteriul stivei vide.

# Masini Turing

banda de intrare



cap citire/scriere



- infinita
- finita la stanga
- ...

# Masini Turing

O miscare a masinii Turing consta din:

- se schimba starea
- se inlocuieste simbolul curent de pe banda de intrare
- capul citire scriere se muta cu o pozitie la stanga sau la dreapta

# Masina Turing cu banda infinita

O masina Turing este:  $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$

- $Q$  – multime finita de stari
- $\Gamma$  – multimea simbolurilor benzii
- $\#$  - un simbol din  $\Gamma$ , numit simbolul *blanc*
- $\Sigma$  – o submultime a lui  $\Gamma - \{\#\}$
- $\delta$  – este functia de tranzitie

$$\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- $q_0$  – starea initiala
- $F \subset Q$  – multimea starilor finale

# Masina Turing cu banda infinita

- configuratie

$\alpha_1 q \alpha_2$        $\alpha_1 \alpha_2 \in \Gamma^*$ , separate de capul de citire  
pana la cel mai din stanga/dreapta simbol ne-blank

- tranzitie

$(p, Y, L) \in \delta(q, X_i)$

$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 X_2 \dots X_{i-2} p X_{i-1} Y X_{i+1} \dots X_n$

$(p, Y, R) \in \delta(q, X_i)$

$X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n \vdash X_1 X_2 \dots X_{i-1} Y p X_{i+1} \dots X_n$

- limbaj acceptat

$\{ w \in \Sigma^* \mid q_0 w \vdash \alpha_1 q \alpha_2, q \in F, \alpha_1 \alpha_2 \in \Gamma^* \}$

# Exemplu: masina Turing

Functia de tranzitie

0011 ?

	$\Sigma$		$\Gamma - \Sigma$			
	0	1	X	Y	#	
$q_0$	( $q_1, X, R$ )			( $q_3, Y, R$ )		0
$q_1$	( $q_1, 0, R$ )	( $q_2, Y, L$ )		( $q_1, Y, R$ )		0
$q_2$	( $q_2, 0, L$ )		( $q_0, X, R$ )	( $q_2, Y, L$ )		0
$q_3$				( $q_3, Y, R$ )	( $q_4, \#, R$ )	0
$q_4$						1

# Masini Turing

- Masina Turing cu o singura banda
  - versus            Masina Turing cu mai multe benzi
  - O maşină Turing cu k benzi
  - nu este mai puternică decât o maşină Turing standard
- Maşină Turing deterministă (MTD)
  - versus            maşină Turing nedeterministă (MTND)
  - Cele două sunt computaţional echivalente,  
adică orice MTND se poate transforma într-o MTD  
(şi invers).

# Masini Turing

## *Teza lui Church*

- Logicianul Alonzo Church a emis ipoteza că mașina Turing este modelul cel mai general de calcul care poate fi propus.
  - mașina Turing este la fel de puternică ca orice alt model de calcul
  - nu înseamnă că poate calcula la fel de repede ca orice alt model de calcul, ci că poate calcula aceleasi lucruri
- Acest enunț nu este demonstrabil în sens matematic.

Dacă avem un model de calcul, putem defini precis ce înțelegem prin complexitate:

- **Timpul** de calcul pentru un sir dat la intrare: este numărul de mutări făcute de mașina Turing înainte de a intra în starea ``terminat'';
- **Spațiul** consumat pentru un sir de intrare: este numărul de căsuțe de pe bandă pe care algoritmul le folosește în timpul execuției sale.

# Potrivirea sirurilor de caractere (pattern matching) folosind AF

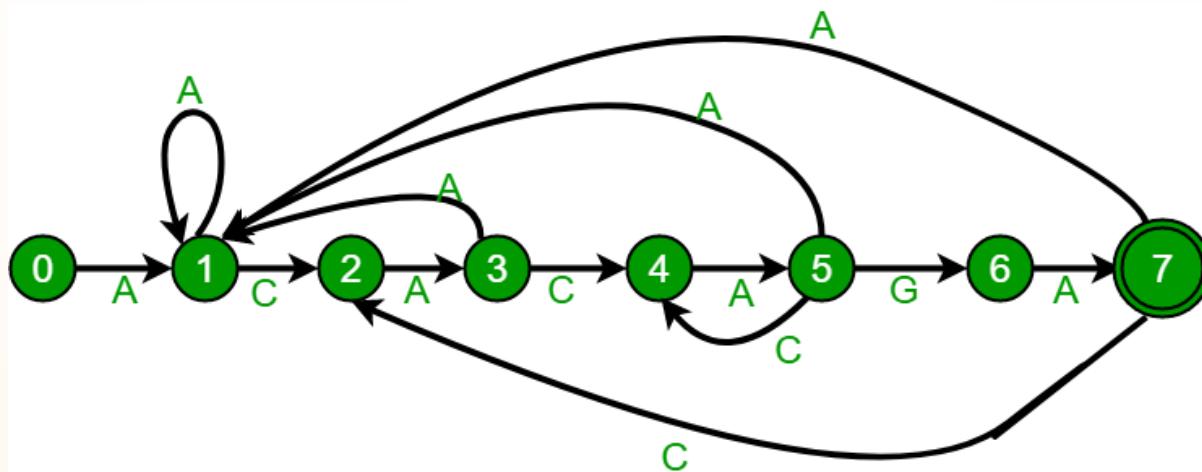
## Exercitiu:

Gasiti subsecventa:

ACACAGA

in textul:

ACACACAGA



# Gramatici

- Recapitulare, Exemple, Aplicatii

# Limbaje formale si tehnici de compilare

## Limbaj formal

Limbaj regulare:

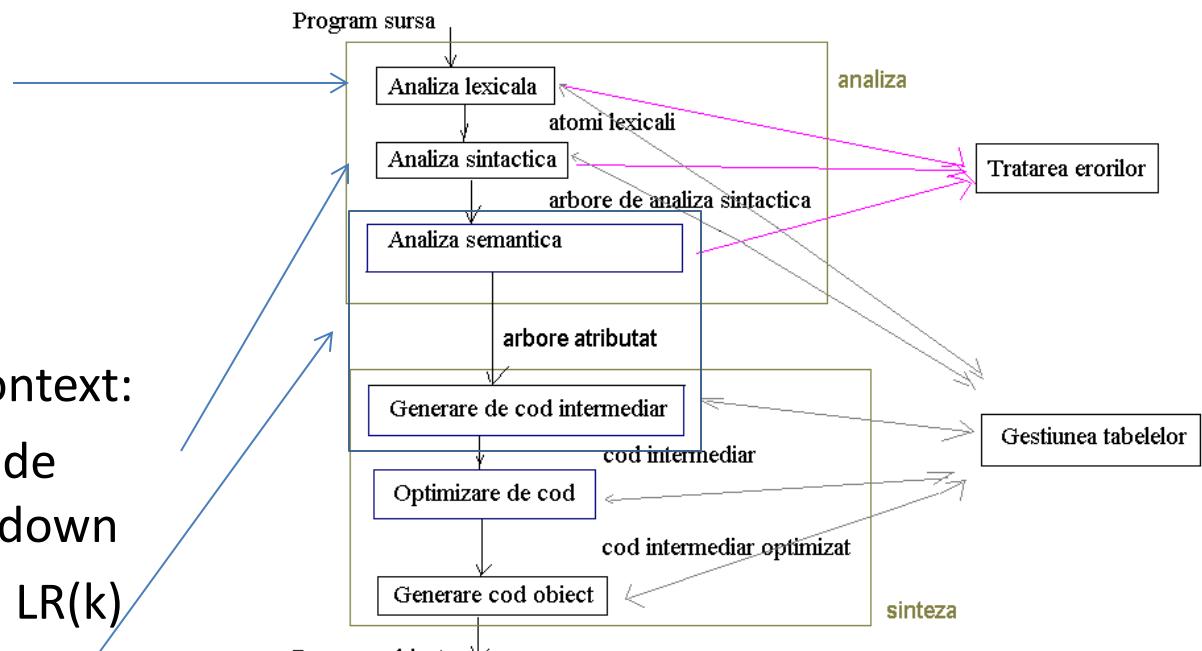
- gramatici regulare,
- automate finite,
- expresii regulare.

Limbaje independente de context:

- Gramatici independente de context, automate push-down
- Gramatici speciale: LL(k), LR(k)

Gramatici de atrbute

## Structura unui compilator



# Gramatica

O gramatica este un cvadruplu  $\mathbf{G} = (\mathbf{N}, \Sigma, P, S)$

- $\mathbf{N}$  este un alfabet de simboluri **neterminale**
- $\Sigma$  este un alfabet de simboluri **terminale**
- $\mathbf{N} \cap \Sigma = \emptyset$
- $P \subseteq (\mathbf{N} \cup \Sigma)^* \mathbf{N} (\mathbf{N} \cup \Sigma)^* \times (\mathbf{N} \cup \Sigma)^*$   
 $P$  multime finită (multimea regulilor de productie)
- $S \in \mathbf{N}$  (simbolul de start - simbolul initial)

Notatie:

$(\alpha, \beta) \in P$  se noteaza:  $\alpha \rightarrow \beta$   
( $\alpha$  se înlocuieste cu  $\beta$ )

- Gramatici de tip **0**  
nici o restrictie (*suplimentara*) referitoare la forma regulilor de productie
- Gramaticile de tip **1**  
***dependente de context***  $\Leftrightarrow$  ***gramatici monotone***  
( *monotonic, non-contracting* )
- Gramaticile de tip **2**  
***gramatici independente de context***
- Gramaticile de tip **3**  
***gramatici regulare***

- Gramaticile monotonă
  - $\forall \alpha \rightarrow \beta \in P: |\alpha| \leq |\beta| \quad \alpha, \beta \in (N \cup \Sigma)^*$
  - caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . În acest caz S nu apare în membrul drept al unei reguli de producție.
- Gramatica dependenta de context  
reguli de producție sunt de forma:  

$$\alpha A \beta \rightarrow \alpha \gamma \beta \quad A \in N$$

$$\alpha, \beta, \gamma \in (N \cup \Sigma)^*, \gamma \neq \epsilon$$
  - caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . În acest caz S nu apare în membrul drept al unei reguli de producție.

# Transformarea gramaticilor monotone in gramatici dependente de context

Ideea:

Pentru fiecare regula de productie de forma:  $X_1 \dots X_m \rightarrow Y_1 \dots Y_n$  , ( stim ca  $m \geq n$ ,) introducem:

$$X_1 \ X_2 \ \dots \ X_{m-1} \ X_m \quad \rightarrow \quad Z_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m$$

$$Z_1 \ Z_2 \ \dots \ X_{m-1} \ X_m \quad \rightarrow \quad Z_1 \ Z_2 \ \dots \ X_{m-1} \ X_m$$

...

$$Z_1 \ Z_2 \ \dots \ X_{m-1} \ X_m \quad \rightarrow \quad Z_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m$$

$$Z_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m \quad \rightarrow \quad Z_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n$$

$$Z_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n \rightarrow Y_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n$$

$$Y_1 \ Z_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n \rightarrow Y_1 \ Y_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n$$

...

$$Y_1 \ Y_2 \ \dots \ Z_{m-1} \ Z_m \ Y_{m+1} \dots Y_n \rightarrow Y_1 \ Y_2 \ \dots \ Y_{m-1} \ Z_m \ Y_{m+1} \dots Y_n$$

$$Y_1 \ Y_2 \ \dots \ Y_{m-1} \ Z_m \ Y_{m+1} \dots Y_n \rightarrow Y_1 \ Y_2 \ \dots \ Y_{m-1} \ Y_m \ Y_{m+1} \dots Y_n$$

- unde  $Z_i$  – sunt neterminale nou introduse, distincte pentru fiecare regula de productie

- Gramatica regulara:

reg. prod. sunt de forma

- $A \rightarrow aB$

- $A \rightarrow b$  unde  $A, B \in N$  si  $a, b \in \Sigma$

caz special:  $S \rightarrow \epsilon$  poate  $\in P$ . In acest caz S nu apare în membrul drept al nici unei reguli de productie.

- Gramatica independenta de context:

reg. productie sunt de forma  $A \rightarrow \alpha$ ,  $A \in N$ ,  $\alpha \in (N \cup \Sigma)^*$

# Gramaticile regulare si liniare in diverse surse.

Gramatica regulara la dreapta

- $A \rightarrow aB$
- $A \rightarrow b$

Discutie: unele surse accepta si:  $A \rightarrow \epsilon$

alte surse nu accepta deloc  $\epsilon$ -productii

Conventiile folosite in  
cadrul acestui curs:  
vezi definitiile  
de pe slide-urile anterioare.

Gramatica regulara la stanga

- $A \rightarrow B a$
- $A \rightarrow b \dots$

Gramatica liniara la dreapta

(gr.regulara la dreapta extinsa)

- $A \rightarrow a_1 a_2 \dots a_n B$
- $A \rightarrow b_1 b_2 \dots b_m \dots$

Gramatica liniara la stanga

(gr.regulara la dreapta extinsa)

...

Gramatica liniara:

- are cel mult un neterminal in membrul drept al regulii de productie

e.g.:  $S \rightarrow aSb$   
 $S \rightarrow ab$

# Gramatici independente de context

*Ne reamintim:*

- gram.  $\epsilon$ -independenta
  - eliminarea regulilor de productie de redenumire
  - forma normala Greibach
  - forma normala Chomsky (FNC)
- [...] si echivalente

# Forma normală Greibach

## Gramatica în forma normală Greibach

- reg. prod. sunt de forma
  - $A \rightarrow a \alpha$  unde  $\alpha \in (N \cup \Sigma)^*$  și  $a \in \Sigma$
- caz special:  
 **$S \rightarrow \epsilon$  poate  $\in P$**   
În acest caz S nu apare în membrul drept al nici unei reguli de producție.

# Forma normala Greibach

## Teorema:

Pentru orice gramatica independenta de context exista o gramatica in forma normala Greibach echivalenta.

## Constructie:

Fie: gram.  $\varepsilon$  –independenta, fara redenumiri

1. (similar cu algoritmul pt. eliminarea recursivitatii stanga)

- impunem o ordine asupra neterminalelor:  $N = \{A_1, A_2, \dots, A_n\}$   
si apoi modific r.p. a.i. sa nu existe  $A_i \rightarrow A_j \alpha$  cu  $i < j$

2. Pentru  $k = n-1, \dots, 1$

avem neterminalul  $A_k$

pentru toate r.p.  $A_k \rightarrow A_j \alpha$  ( $k < j$ )

se inlocuiesc cu:  $A_k \rightarrow \beta \alpha$ ,

pentru toti  $\beta$  a.i. :  $A_j \rightarrow \beta$  (in toate modurile posibile)

# Forma normala Chomsky

Gramatica in forma normala Chomsky

Regulile de productie sunt de forma:

- $A \rightarrow BC$
- $A \rightarrow d$  unde  $A, B, C \in N$  si  $d \in \Sigma$

caz special:

**S → ε** poate sa apara in multimea regulilor de productie,  
dar in acest caz S nu apare în membrul drept al nici unei reguli de  
productie.

# Forma normala Chomsky

## Teorema:

Pentru orice gramatica independenta de context  
exista o gramatica in forma normala Chomsky echivalenta

## Constructie:

Fie: gram.  $\epsilon$  –independenta, fara redenumiri

Folosim transformari de forma:

$$A \rightarrow X_1 X_2 \dots X_n$$

:

$$A \rightarrow X_1 Z_1$$

$$Z_1 \rightarrow X_2 Z_2$$

$$Z_{n-2} \rightarrow X_{n-1} X_n$$