Atomi lexicali:

– identificatori

spatiile "albe"

– constante (literali)

– cuvinte cheie (*cuvinte rezervate*)

– operatori aritmetici, relationali, …

separatori

cuvinte cheie - simboluri sintactice

cuvinte rezervate - nu pot fi folosite ca identificatori

(semnificatii speciale)

Analiza lexicala

detecteaza

Pentru fiecare *atom* lexical din fisierul sursa

clasifica *atom*

codifica *atom*

Sf.pentru

Analiza lexicala (versiunea 1)

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

detecteaza *atom*

clasifica *atom*

codifica *atom*

Sf.cattimp

Analiza lexicala (versiunea 2)

Analiza lexicala

- Date: fisier text
- Rezultate: FIP, TS

  sau: mesaj de eroare

TS: tabela de simboluri

- informatii despre identificatori si constante

  nume, tip, …

FIP: forma interna a programului

- Cod atom lexical
- Pozitia in TS (unde e cazul)

Analiza lexicala

Analiza lexicala, FIP, TS – exemplu partial

```
if  a< b then b:=a
```

- Care sunt elementelelexicale ?
- 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	…

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 identificator 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

Clasifica

Analiza lexicala (versiunea 3)

```
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
```

Analiza lexicala (versiunea 4)

Obs.: multe cautari in TS

++ urmatoarele faze !!

- TS fara organizare

  - cautare secventiala

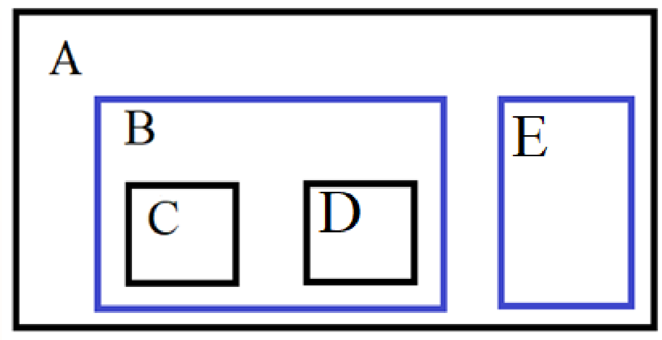
- organizare pt. cautare mai rapida
  - ordonare (ordonata lexicografic)
    - tabel ordonat lexicografic
    - arbore binar de cautare echilibrat
  - tabela de dispersie

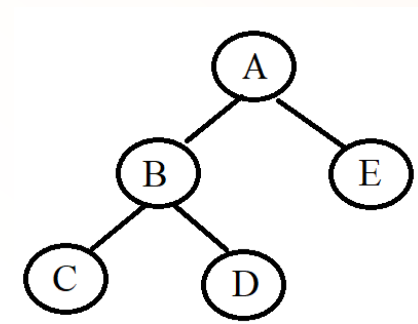
➔cautare – fct. de dispersie

(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	
1	A			TS.A
2	B	1		TS.B
3	C	2		TS.C
4	D	2		TS.D
5	E	1		TS.E

Notiuni de baza

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

Alfabet

- Def:

  Alfabet = o multime finitã si nevidã de elemente numite simboluri

- Notatie: Σ

Secventa

Def.

- **secventa peste Σ**

  o succesiune finita de simboluri din Σ

- subsecventa

  o succesiune de simboluri consecutive dintr-o secventa

Lungimea unei secvente

- def:

  nr. de simboluri din care este formata acea secventa
- notatie

  | … |

  Ex: |abc| = 3

- **Secv. vida** = secv. de lungime 0
- notatie: ε   (unele surse λ)

Concatenare

Dacã

$\mathbf{x=a_1a_2\ldots a_n}$

$\mathbf{y=b_1b_2\ldots b_m}$

atunci $\mathbf{z = a_1\ldots a_n b_1\ldots b_m}$

reprezintã **concatenarea** secventelor x si y

si se noteaza $\mathbf{z} = \mathbf{xy}$

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

(de n ori)

Exemplu: $a^2 = aa$

Secventa

Def.

- **secventa peste Σ**

  o succesiune finita de simboluri din Σ

- subsecventa

  o succesiune de simboluri consecutive dintr-o secventa

w2 – subsecventa a lui w1

daca ∃ secventele u, v a.i. w1=u w2 v

Prefix, sufix

Fie **x**, **y**, **z** sunt secvente peste alfabetul Σ

- **x** este un **prefix** al secventei **xy**
- **y** un **sufix** al secventei **xy**

- **Prefix**: o subsecventa care
  - fie este vida
  - fie incepe cu primul simbol al secventei date
- **Sufix**: o subsecventa care
  - fie este vida
  - fie se termina cu ultimul simbol al secventei date

Multimi speciale

- $\Sigma^n = \{w \mid w$ – secventa peste $\Sigma, |w| = n\}$
- $\Sigma^* = \{w \mid w$ – secventa peste $\Sigma, 0 <= |w| \}$
- $\Sigma^+ = \{w \mid w$ – secventa peste $\Sigma, 0 < |w| \}$

- $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Operatia *

Denumiri:

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

Σ^* - inchiderea alfabetului

- multimea tuturor secventelor ce se pot obtine folosind secvente din Σ

(Similar pentru limbaje)

Σ^* - proprietati

- $\forall\ w_1, w_2 \in \Sigma^*$ avem: $w_1w_2 \in \Sigma^*$
- $\forall\ w_1, w_2, w_3 \in \Sigma^*$ avem: $(w_1w_2)\,w_3 = w_1(w_2w_3)$
- $\forall\ w \in \Sigma^*$, $\varepsilon w = w\varepsilon = w$

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

Limbaj, cuvant

- def: (limbaj)

  L – limbaj peste alfabetul Σ

  daca $L \subseteq \Sigma^*$
- def: (cuvant)

  Cuvant al unui limbaj – un element al limbajului

Metode de specificare a unui limbaj

- enumerand elementele
- evidentierea unor proprietati ale elementelor
  - folosind multimi si descrieri matematice
  - …
- folosind gramatici, automate, expresii regulare
- …

Cateva tipuri de limbaje

- teoretice

  $L = \{a^n \mid n \in N\}$ limbaj peste $\Sigma = \{a\}$

  $L = \{a^n b^n \mid n \in N\}$ limbaj peste $\Sigma = \{a,b\}$

- matematice

  ex: limbajul reprezentarii zecimale a numerelor naturale

- informatice

  limbajul identificatorilor

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

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

Operatii cu limbaje (1)

Fie: L1 – limbaj peste Σ_1 L2 – limbaj peste Σ_2

(operatii cu multimi)

- $\mathbf{L_1 \cup L_2}$ limbaj peste **Σ ales corespunzator**;

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

- $\mathbf{L_1 \cap L_2}$ limbaj peste Σ ($\Sigma = \Sigma_1 \cap \Sigma_2$)
- $\mathbf{L_1 - L_2}$ limbaj peste Σ ($\Sigma = \Sigma_1$)
- $\mathbf{L_1 L_2}$ limbaj peste Σ ($\Sigma = \Sigma_1 \cup \Sigma_2$)

(operatii bazate pe concatenare)

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

Operatii cu limbaje (2)

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

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

- închiderea tranzitivã:

$$L^+ = \bigcup_{n \geq 1} L^n \quad \textbf{sau } L^+ = LL^* \qquad ,\ L^* = L^+ \bigcup \{\varepsilon\}$$

Gramatica

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

- **N** este un alfabet de simboluri ***neterminale***
- Σ este un alfabet de simboluri ***terminale***
- $N \cap \Sigma = \phi$
- $\mathbf{P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* x (N \cup \Sigma)^*}$

  P multime finitã (multimea regulilor de productie)
- **S** $\in$ **N** (simbolul de start - simbolul initial)

Notatie:

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

(α se înlocuieste cu β)

Notatii

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

Relatii de derivare

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

- derivare directa

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

  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* derivãri directe)

- \+ derivare $\stackrel{+}{\Rightarrow}$

  dacã $\exists$ k>0 a.i. cele 2 secvente sã fie într-o relatie de "k derivare"

- * derivare $\stackrel{*}{\Rightarrow}$

  daca fie cele 2 secvente sunt egale, fie intre ele exista o relatie de +derivare

Limbaj generat de o gramatica

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

  $L(G) = \{w \in \Sigma^* \mid S \overset{*}{=>} w\}$
- Forma propozitionala
  - $\alpha \in (N \cup \Sigma)^*$ a.i. $S \overset{*}{=>} \alpha$
- Propozitie (<u>cuvant</u>)
  - un element din L(G)
- Gramatici echivalente

  daca genereaza acelasi limbaj

- 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: $\mathbf{S} \longrightarrow \boldsymbol{\varepsilon}$ 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 $A \rightarrow \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^*$

Tipuri de gramatici

- Gramaticile monotona
  - $\forall\ \alpha \rightarrow \beta \in P: |\alpha| <= |\beta|$ $\alpha, \beta \in (N \cup \Sigma)^*$
  - caz special: $\mathbf{S} \longrightarrow \boldsymbol{\varepsilon}$ poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

- Gramatica dependenta de context

  reguli de productie sunt de forma:

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

  $\alpha, \beta, \gamma \in (N \cup \Sigma)^*, \gamma \neq \boldsymbol{\varepsilon}$
  - caz special: $\mathbf{S} \longrightarrow \boldsymbol{\varepsilon}$ poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Tipuri de gramatici

- Gramatici de tip **0**

  nici o restrictie *(suplimentara)* referitoare la forma regulilor de productie

- Gramaticile de tip **1**

  dependente de context ⇔ gramatici monotone

  (monotonic, non-contracting)

- Gramaticile de tip **2**

  gramatici independente de context

- Gramaticile de tip **3**

  gramatici regulare

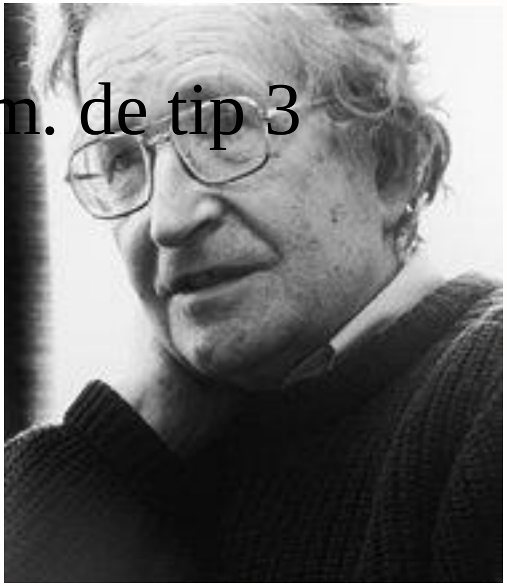
Clasificarea Chomsky

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 reuniune

Gramatica: exemplu

- $G = (N, \Sigma, P, S)$
  - $N = \{A\}$
  - $\Sigma = \{a\}$
  - $S : A$
  - P: $A \to aA$

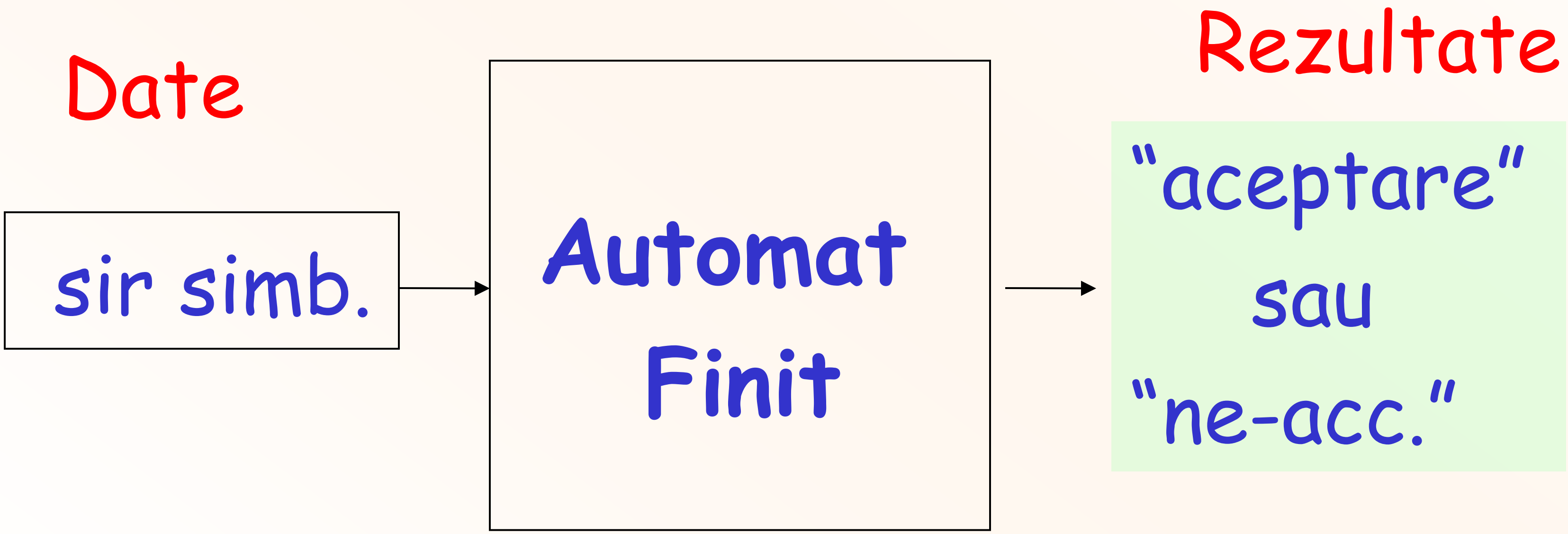
    $A \to a$

  $L(G) = ?$

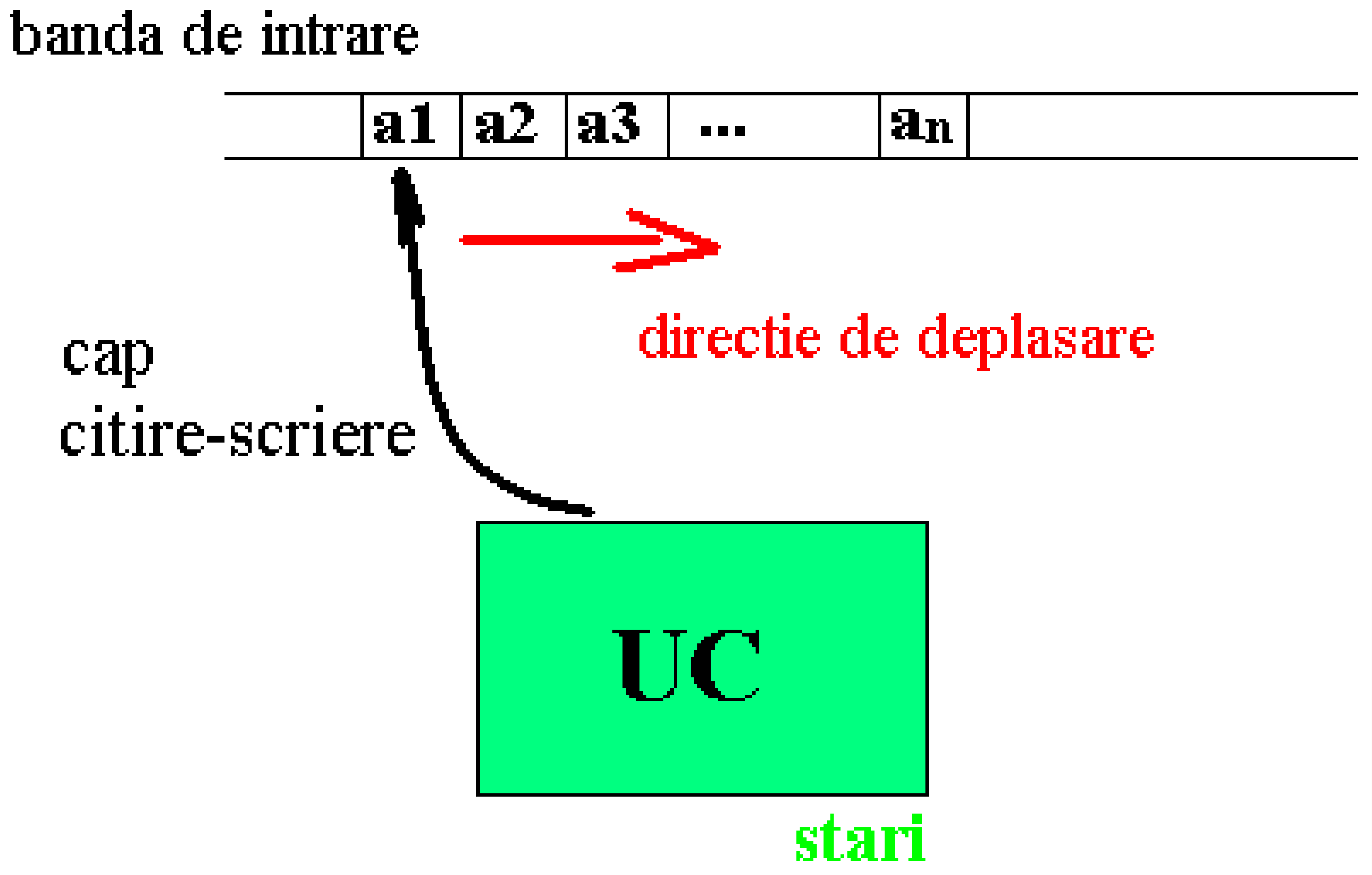
  $|L(G)| = ?$

Automat finit (AF)

•



Automat finit: model fizic



Automat finit: model matematic

- Un ***automat finit*** este un ansamblu

  $M = (Q, \Sigma, \delta, q_0, F)$:
- Q – alfabetul starilor
- Σ – alfabet de intrare
- $\delta : Q x \Sigma \rightarrow \mathcal{P}(Q)$ functie de tranzitie
- $q_0 \in Q$ - stare initialã
- $F \subseteq Q$ multimea stãrilor finale

AF – reprezentare tabelara

δ		a_j		

$z_i =$ 0 daca q_i nu e stare finala
1 daca q_i este stare finala

st. init.

q_0

F?

δ	0	1	
p	q	p	0
q	r	p	0
r	r	r	1

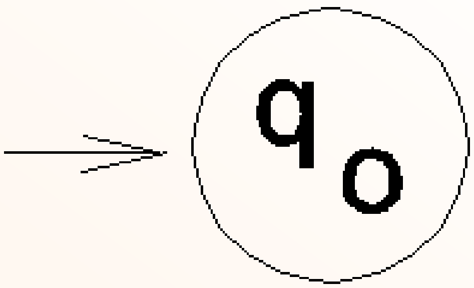
$Q \setminus F$

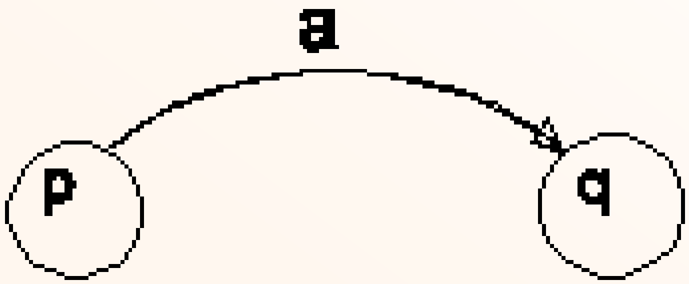
F

AF reprezentat tabelar; exemplu

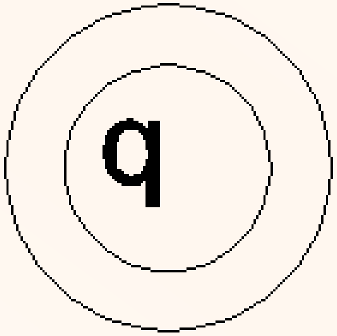
AF – reprezentare sub forma de graf

- graf orientat
- cu noduri si arce etichetate





- (graf de tranzitii)



Configuratii si relatii de tranzitie

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

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

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

$(p,aw) \vdash (q,w) \Leftrightarrow \delta(p,a) \ni q$;

$p,q \in Q$, $a \in \Sigma$, $w \in \Sigma^*$

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

Limbaj acceptat; autom. echivalente

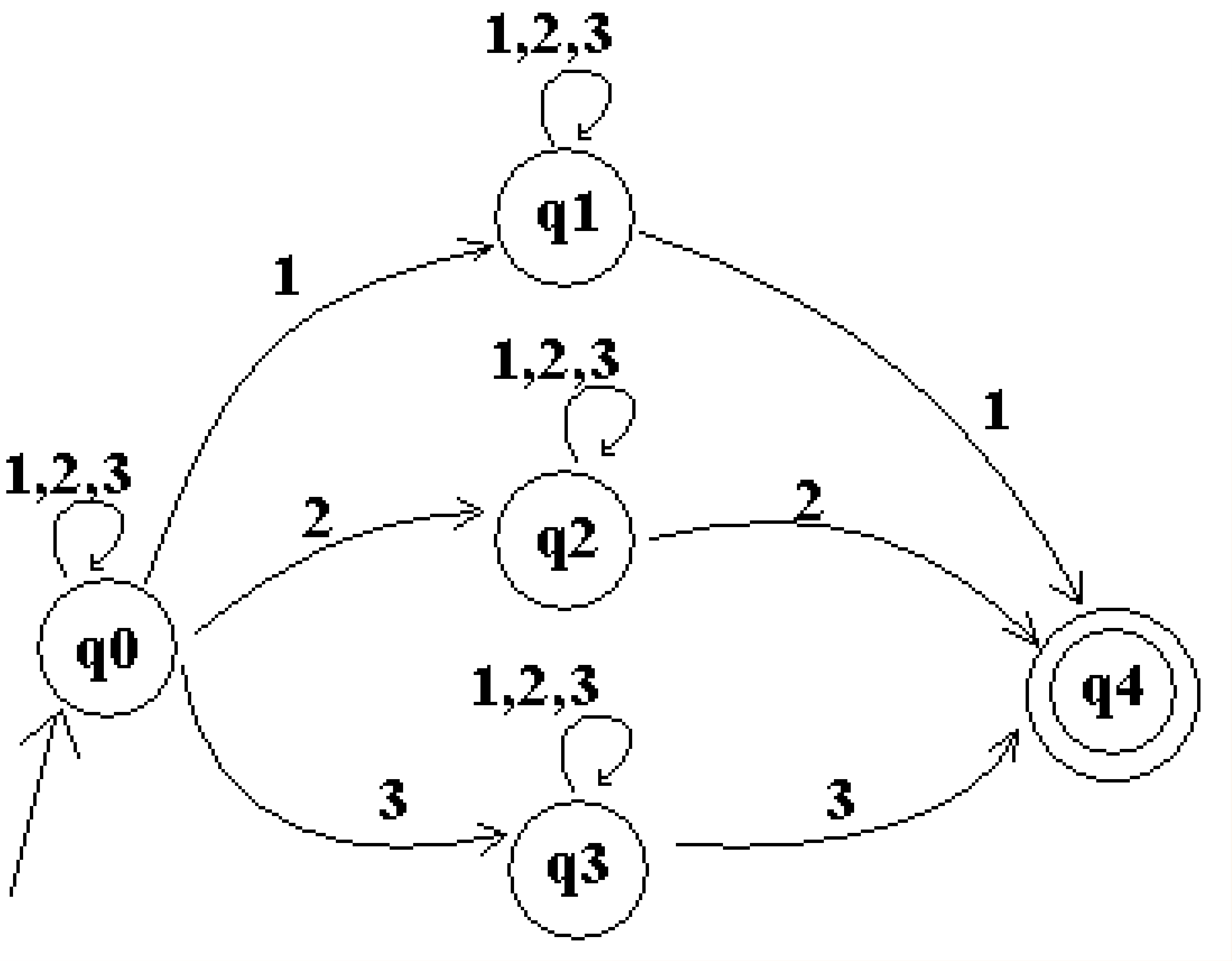
- Limbaj acceptat de automat

  $\mathbf{L(M) = \{w \mid w \in \Sigma^*, (q_0,w) \vdash^* (q_f,\varepsilon), q_f \in F\}}$

- Automate echivalente

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

până aici am făcut
în cursul 2



Automat finit - exemplu

Determinism

- Automat finit determinist (AFD)

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

- Automat finit nedeterminist (AFN)

  $\exists\, q \in Q,\, a \in \Sigma$ 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:

- $\forall\ M_1$ – AFN $\quad \exists\ M_2$ – AFD 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. $L(M_1) = L(M_2)$

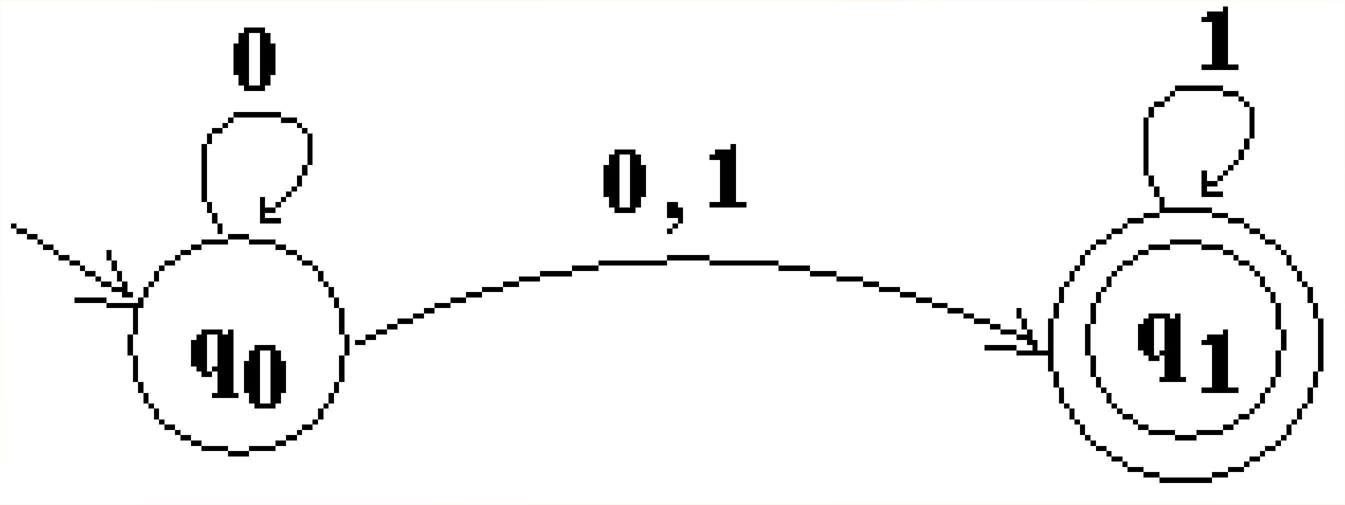
Teor: $\forall$ M_1 – AFN $\exists$ M_2 – AFD echivalent

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

$$= \bigcup_{q1 \in q} \delta(q_1,a)$$

M_2 – determinist (?)

Problema: determinati AFD echiv. pt.



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.

  $\exists\, w \in \Sigma^*$ si $q_f \in F$ a.i. $(q,w) \vdash^* (q_f,\varepsilon)$
- stare accesibila: $q \in Q$ a.i.

  $\exists\, w \in \Sigma^*$ a.i. $(q_0,w) \vdash^* (q,\varepsilon)$

Algoritm determin. stari accesibile

1. $i:=0$

   $A_0:=\{q_0\}$

2. **Repeta**

   $i:=i+1$

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

   pana cand $A_i=A_{i+1}$

$\{A_i$ – multimea starilor accesibile$\}$

Algoritm determin. stari productive

1. $i:=0$

   $A_0:=F$

2. **Repeta**

   $i:=i+1$

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

   pana cand $A_i=A_{i+1}$

$\{A_i$ – multimea starilor productive$\}$

Teorema:

$\forall$ M_1 – AF exista M_2 – AF fara 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. $L(M_1) = L(M_2)$)

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

$$L(M_1) = L(M_2) \quad !$$

Teorema:

$\forall$ M_1 – AF exista M_2 – AF fara st. inaccesibile **echiv**.

Constructie (nu demonstratie!):

- … analog …

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

- $M_1 \Rightarrow ?\ M_2$

Ideea:

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)**

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

<u>Teor</u>: $\forall$ AFD $\exists$ AFD complet definit echivalent

Constructie:

AFD => AFD complet definit:

- adaugam o stare (neproductiva) *r* si extindem δ astfel:
- $\forall (q,a) \in Q x \Sigma$ a.i. $\delta(q,a)=\phi$ devine: $\delta(q,a)= \{\mathbf{r}\}$
- $\forall \; a \in \Sigma$ $\delta(r,a)= \{\mathbf{r}\}$

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
- … mai tarziu … (**~ curs 13**)

Până aici am discutat

în cursul 3

Minimizarea AFD

- automat cu numar minim de stari

  - fara stari –inaccesibile, neproductive

  - mai putine stari ?

  ideea: relatie de echivalenta; clase de echivalenta
- stari diferentiate
- stari k diferentiate
- stari echivalente
- stari k-echivalente

Automat redus

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

Constructie:

1. AFD
2. elim. stari inaccesibile si neprod.
3. det. relatia $\equiv$ si clasele de echiv.
4. constr. aut. redus

Fie M1 – un automat finit oarecare

- Determinam AFD echivalent
- Eliminam starile inaccesibile si neproductive

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

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

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

Automatul redus

Automat redus

- **Teorema**

  Automatul redus are numar minim de stari dintre toate AFD echivalente

- **Teorema**

  $\forall$ M_1 – AF $\exists$ M2 – automat redus echivalent

Stari diferentiate

- o alta exprimare

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

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

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

- x (de mai sus) *diferentiaza* pe q_1 si q_2

Stari diferentiate

- o alta exprimare

PP. AFD complet definit

q_1,q_2 sunt *stari diferentiate* 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 diferentiate
  - cf. def. de mai sus
  - ($\exists$ x $\in \Sigma^*$ care sa le diferentieze)
- stari k diferentiate
  - daca $\exists$ x $\in \Sigma^*$, $|x| \leq k$ care sa le diferentieze
- stari echivalente ($\equiv$)
  - daca nu exista x $\in \Sigma^*$ care sa le diferentieze
- stari k-echivalente ($\equiv^k$)
  - daca nu exista x $\in \Sigma^*$, $|x| \leq k$, care sa le diferentieze

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

- $q_1 \equiv^0 q_2$ ddaca ($q_1, q_2 \in F$) sau ($q_1, q_2 \in Q\text{-}F$)

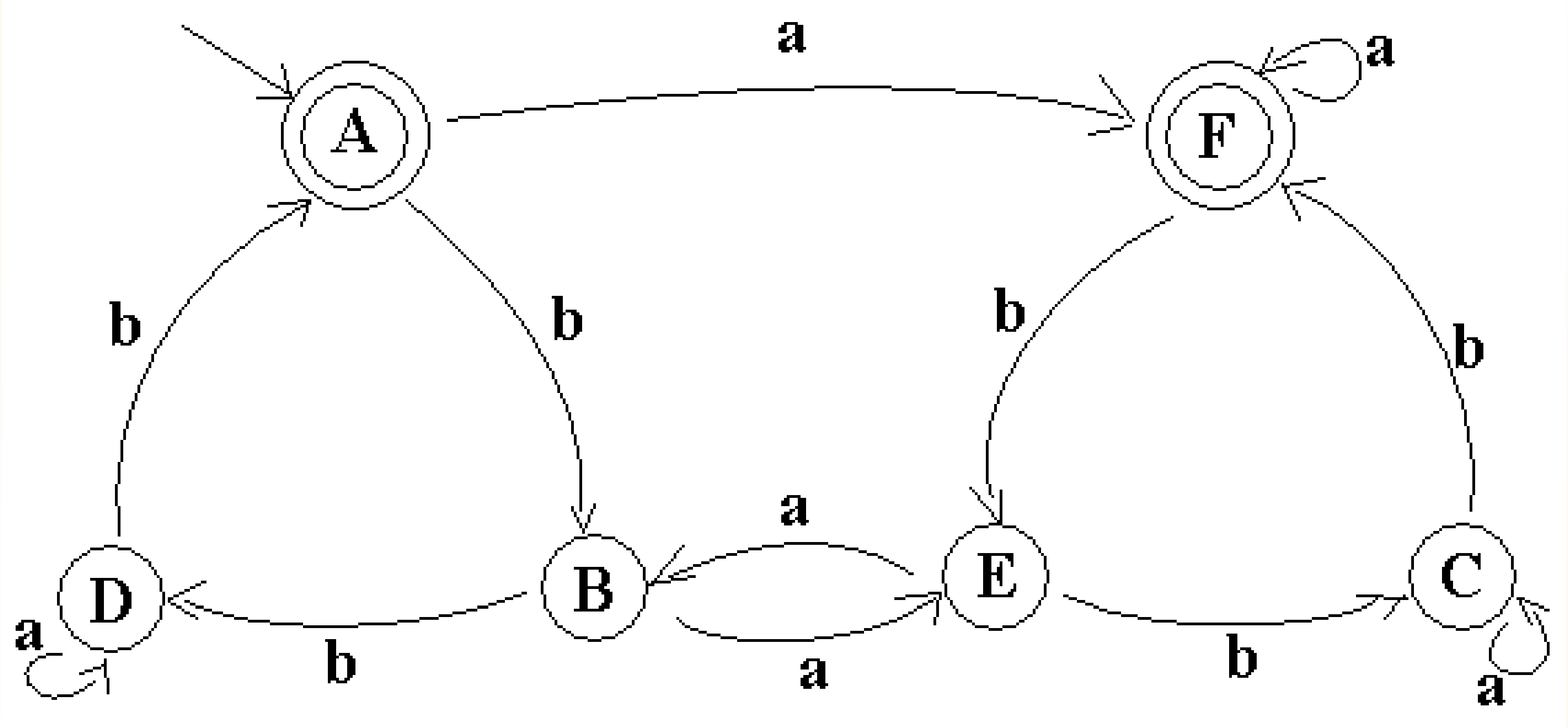
- $\equiv^0 \supseteq \equiv^1 \supseteq \equiv^2 \supseteq \cdots \supseteq \equiv^n \supseteq \cdots$
- Daca $(\equiv^k) = (\equiv^{k+1})$ atunci $(\equiv^k) = \equiv$

Lema:

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

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

Determinati clasele de echivalenta ale starilor automatului de mai jos



determinati automatul redus !

Limbaje regulare. Echivalente

- putere de exprimare

  AF: AFN $\Leftrightarrow$ AFD

  AF $\Leftrightarrow$ (m.regulare $\Leftrightarrow$ expr.reg.)

  AF $\Leftrightarrow$ gr.regulare

Obs.: vom studia aceste echivalente mai in detaliu pe parcursul semetrului

Gramatica

Cursul anterior

O gramatica este un cvadruplu **G = (N, Σ , P, S)**

- **N** este un alfabet de simboluri ***neterminale***
- Σ este un alfabet de simboluri ***terminale***
- $N \cap \Sigma = \phi$
- $\mathbf{P} \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

  P multime finitã (multimea regulilor de productie)
- $\mathbf{S} \in \mathbf{N}$ (simbolul de start - simbolul initial)

Notatie:

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

(α se înlocuieste cu β)

Notatii

Cursul anterior

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

Cursul anterior

- 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: $\mathbf{S} \rightarrow \boldsymbol{\varepsilon}$ poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

- …

Tipuri de gramatici

Multimi regulare

Fie Σ un alfabet.

Multimile regulare peste Σ se definesc recursiv astfel:

1. Φ este o m. reg. peste Σ
2. $\{\varepsilon\}$
3. $\{a\}$ daca: $a \in \Sigma$
4. $R \cup S$ daca R,S – multimi regulare peste Σ +
5. RS daca R,S – multimi regulare peste Σ
6. R^* daca R – multime regulara peste Σ
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. Φ expr. reg. coresp. m.reg. Φ
2. ε $\{\varepsilon\}$
3. a daca: $a \in \Sigma$ $\{a\}$
4. r+s daca r,s – expresii regulare **r | s** $R \cup S$
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 <u>echivalente</u>:
  - mult. regulare reprezentate de acestea sunt egale

Expresii regulare

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

  $\Sigma \cup \{\Phi, \varepsilon, +, *, (\ ,\ )\}$ **(… prioritate …)**
- multimile regulare asociate expresiilor regulare sunt limbaje regulare

Deci: ***Orice expresie regulara peste Σ descrie un limbaj regular peste Σ***

Proprietati de inchidere ale limbajelor regulare

Teorema:

Daca

L_1, L_2 sunt limbaje regulare peste alfabetul Σ

atunci:

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

sunt limbaje regulare peste alfabetul Σ

Lema de pompare pt. limbaje regulare

- Daca L este un limbaj regular,
- atunci $\exists$ p $\in$ $\mathbf{N}^*$ (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 $\in$ $\mathbf{N}$

Lema de pompare pt. limbaje regulare

$$
\begin{aligned}
&(\forall L \subseteq \Sigma^*)\\
&\quad (\mathbf{regular}(L) \Rightarrow\\
&\quad ((\exists p \geq 1)((\forall w \in L)((|w| \geq p) \Rightarrow\\
&\quad ((\exists x, y, z \in \Sigma^*)(w = xyz \wedge (|y| \geq 1 \wedge |\, y \,| \leq p \wedge (\forall n \geq 0)(xy^n z \in L))))))))
\end{aligned}
$$

(enunt formal al teoremei)

Lema de pompare pt. limbaje regulare

(o alta versiune, mai “puternica”)

Daca L este un limbaj regular,

- atunci $\exists\ p \in \mathbf{N}^*$ (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

  $0 < |y|$

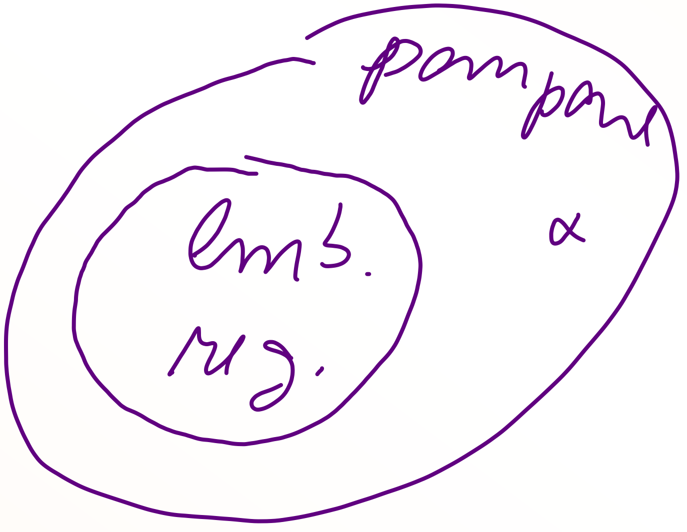
  $|\ x\ y\ | \leq p$

  $xy^iz \in L,\ \forall\ i \in \mathbf{N}$

Lema de pompare pt. limbaje regulare

Observatii:

- Lema da o conditie necesara dar nu suficienta
- 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|>=p si w –acceptat

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

$\Rightarrow$ drumul e de lung. 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

până aici cursul 3

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

Proprietati de inchidere ale limbajelor regulare

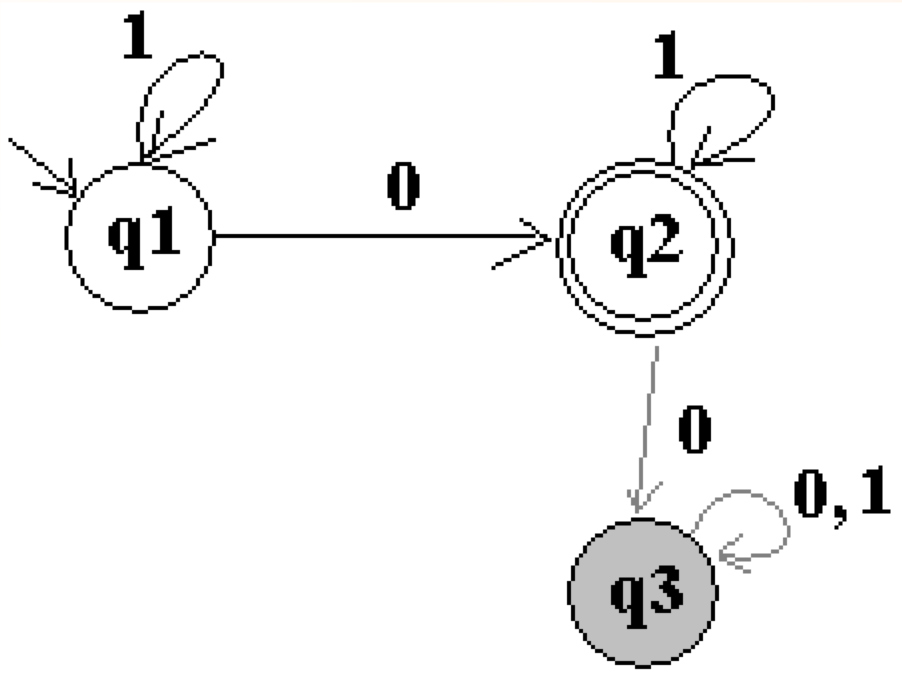
$L_1 \cap L_2$

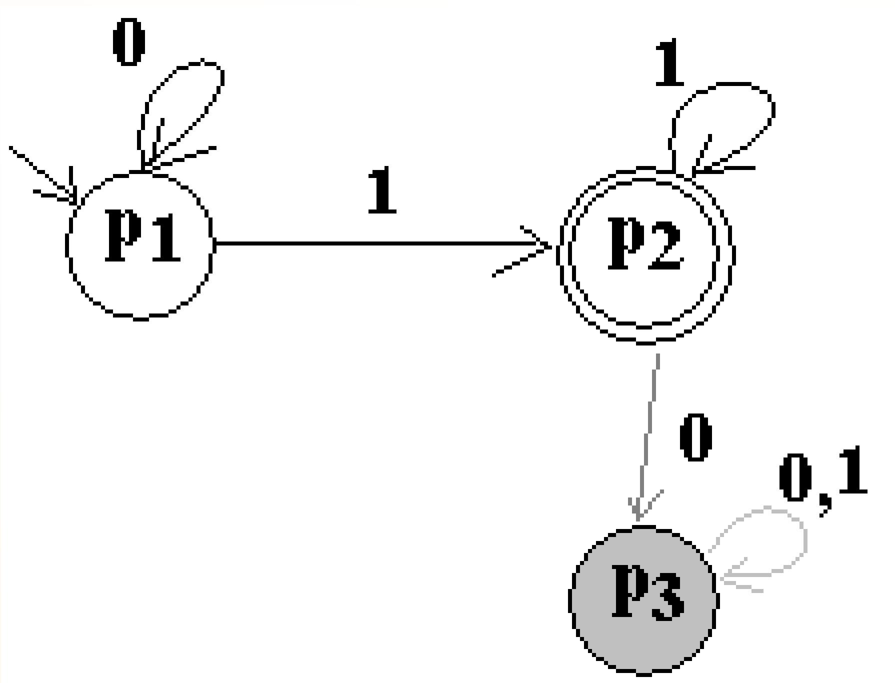
- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
- $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- ? $M = (Q, \Sigma, \delta, q_0, F)$

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

(alg. de constr. !!)

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


1
1
0
q1
q2
0
0, 1
q3


0
1
1
p1
p2
0
0,1
p3

Proprietati de inchidere ale limbajelor regulare

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.)

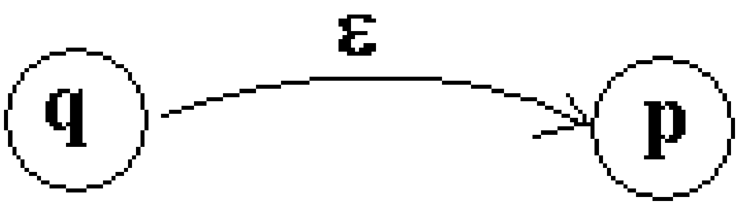
- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, Q_1\text{-}F_1)$

Automate finite cu ε-miscari

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

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

<u>Ideea</u>: putem avea si ε-tranzitii (automate cu ε-tranzitii)



<u>Teorema</u>:

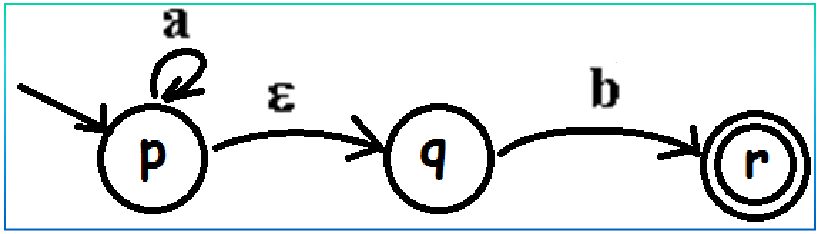
Pentru orice automat finit cu ε-miscari exista un automat finit echivalent.

<u>Obs</u>. *Conform definitiei pe care am dat-o (in cursul 2), automatele finite sunt fara ε-miscari*

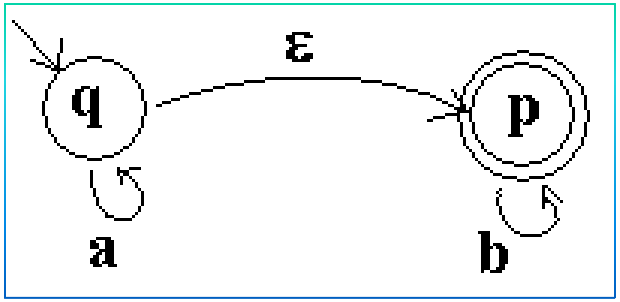
Automate finite cu ε-miscari

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

- $\delta : Qx(\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ 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 secventelor 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)**

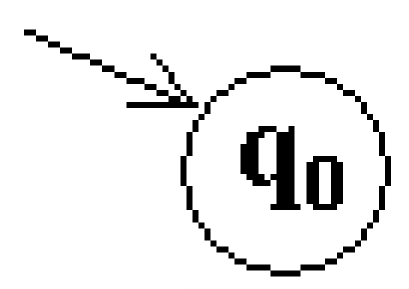
Expresie regulara => limbaj acceptat de AF

- Expresii regulare
  - $\emptyset$
  - ε
  - 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

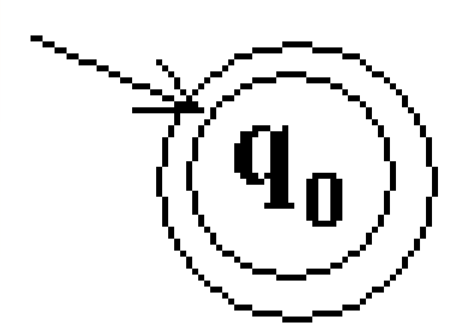
  pentru fiecare dintre constructiile de mai sus

Expresie regulara=> limbaj acceptat de AF

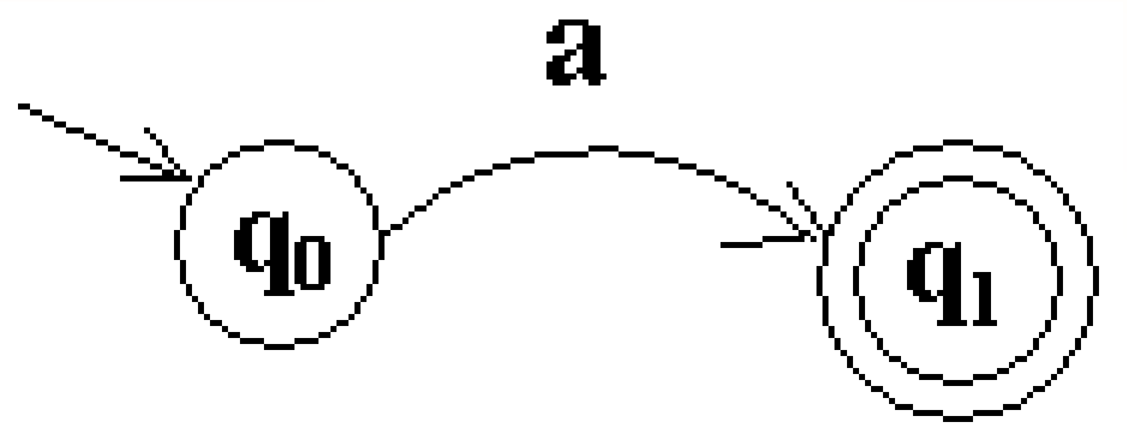
- Automatul ce accepta: Φ



- Automatul ce accepta: ε

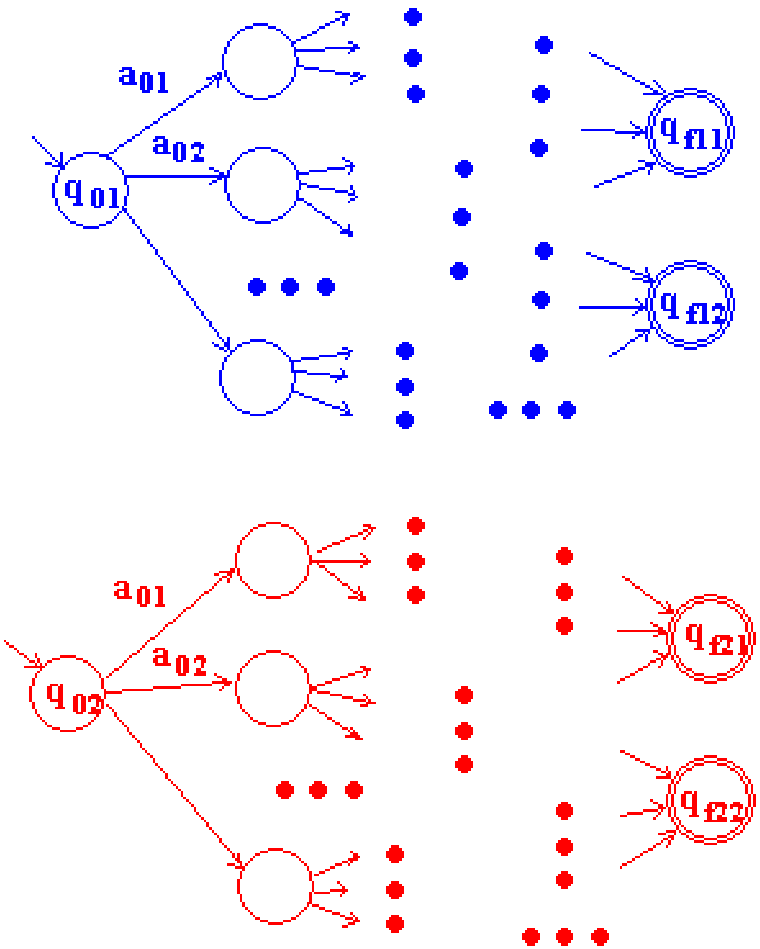


- Automatul ce accepta: a (daca: a ∈ Σ)



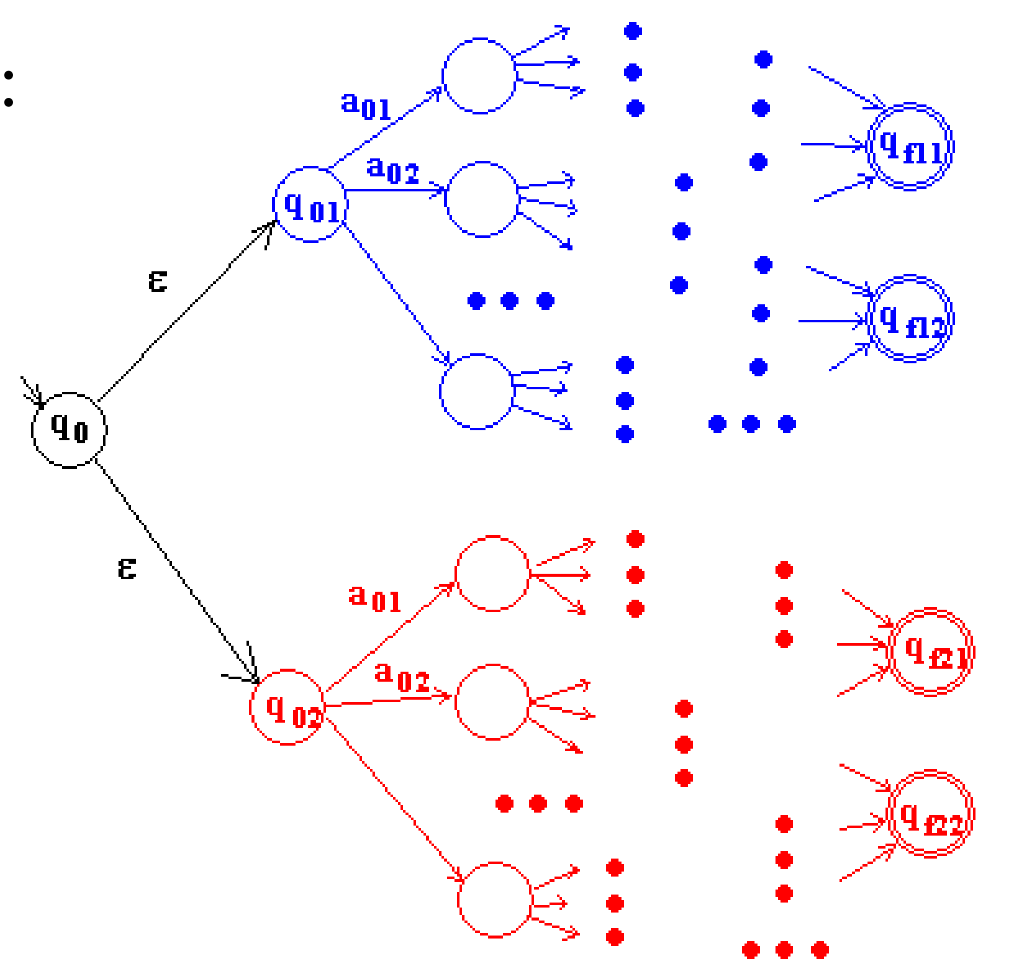
•

- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date
  - se dau:

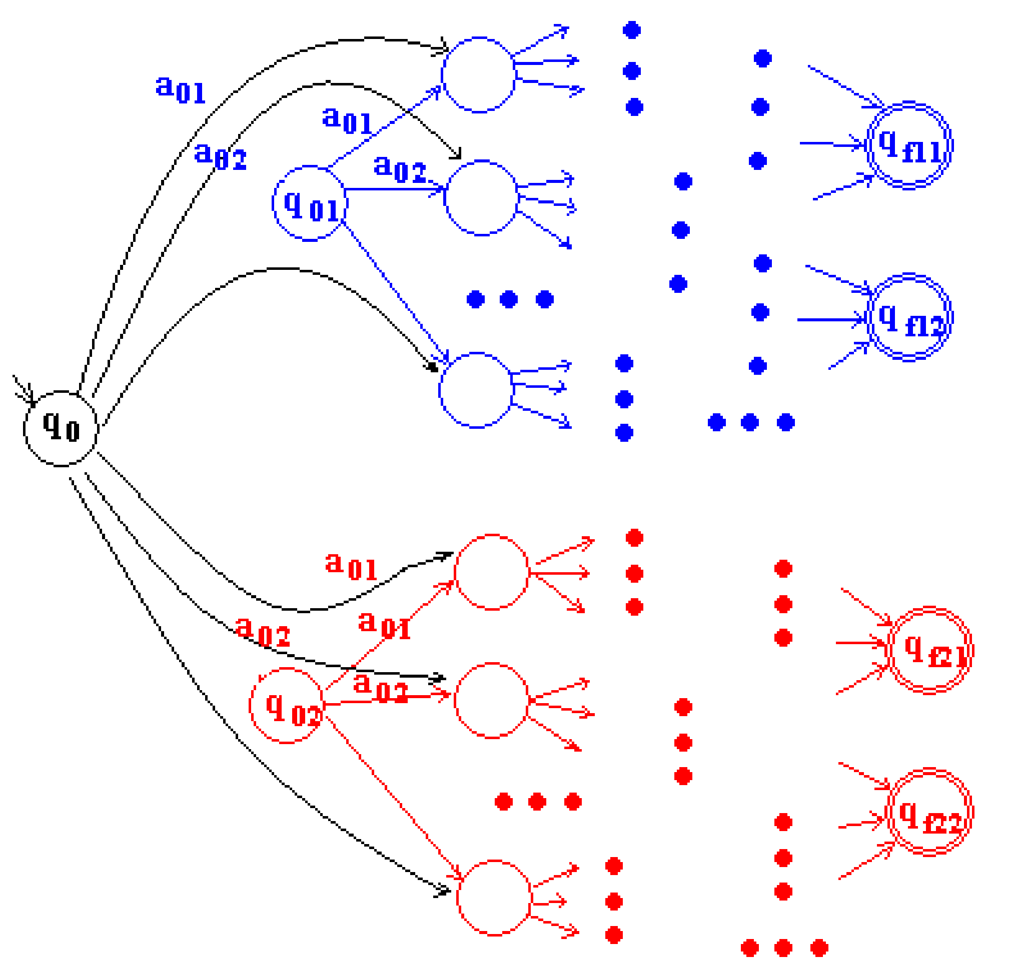


•

- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date
  - AF cu ε tranz.:



- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date
  - AF



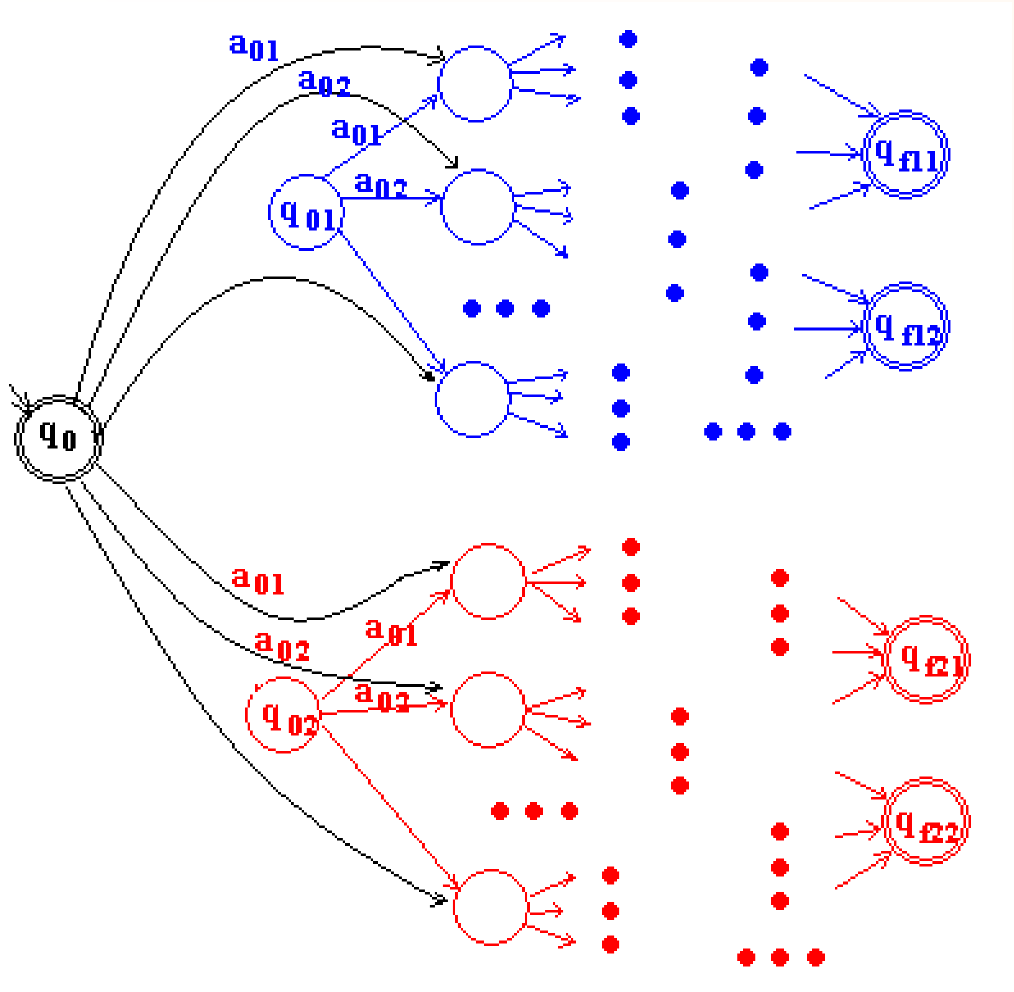
??! cel putin una dintre q_{01} sau q_{02}

e stare finala

•

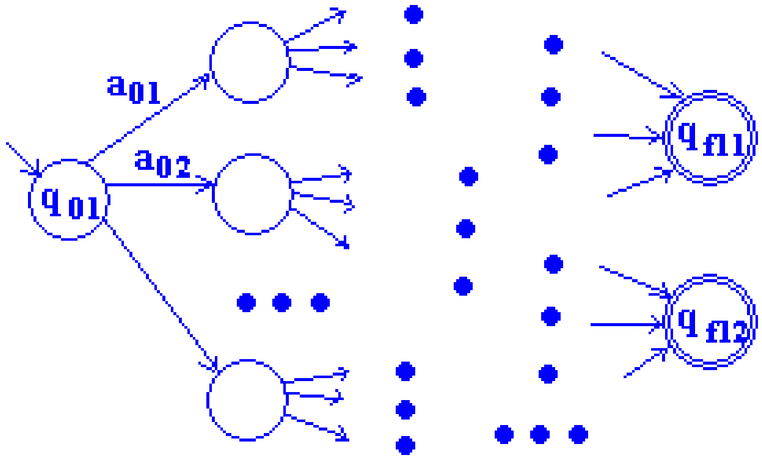
- Automatul ce accepta reuniunea limbajelor acceptate de doua automate date
  - AF

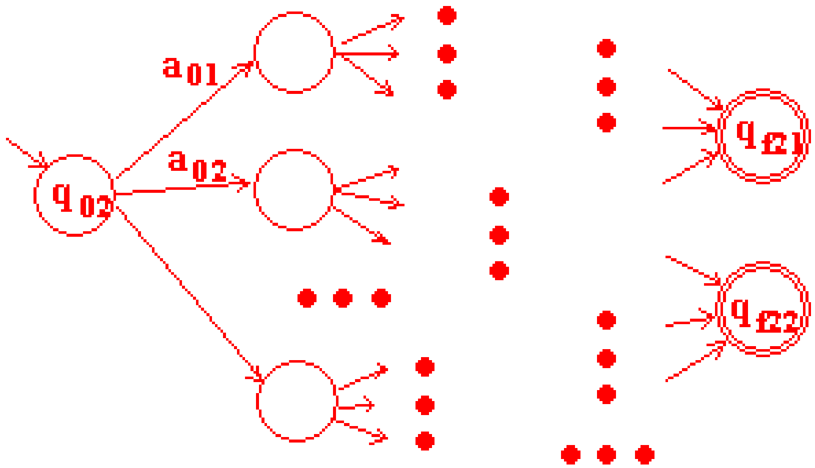
Daca cel putin una dintre q_{01} sau q_{02} este stare finala



...

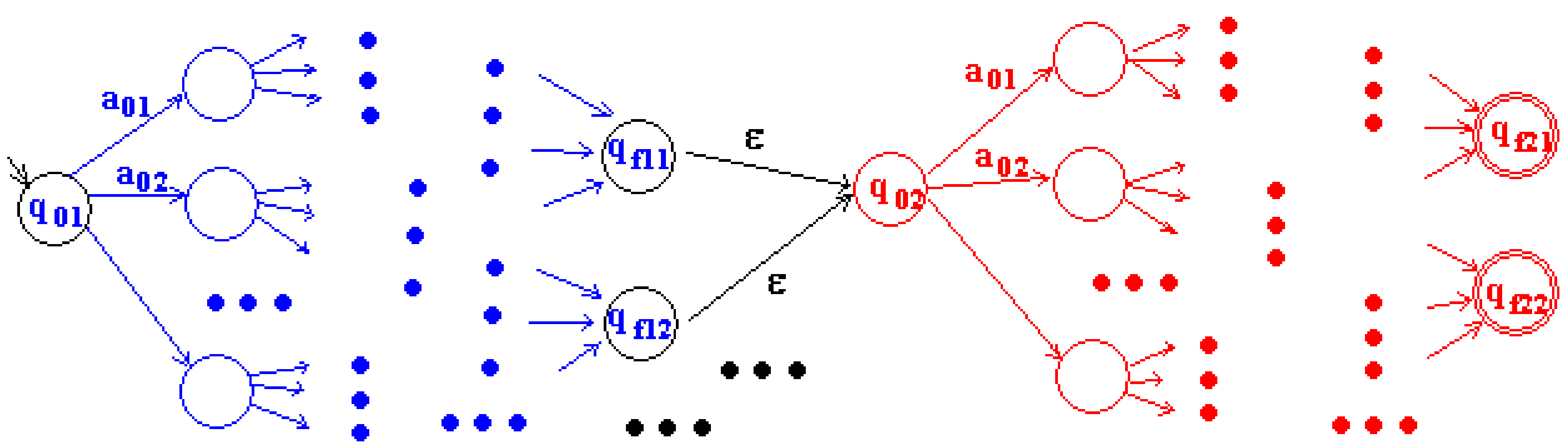
- Automatul ce accepta concatenarea limbajelor acceptate de doua automate date
  - se dau





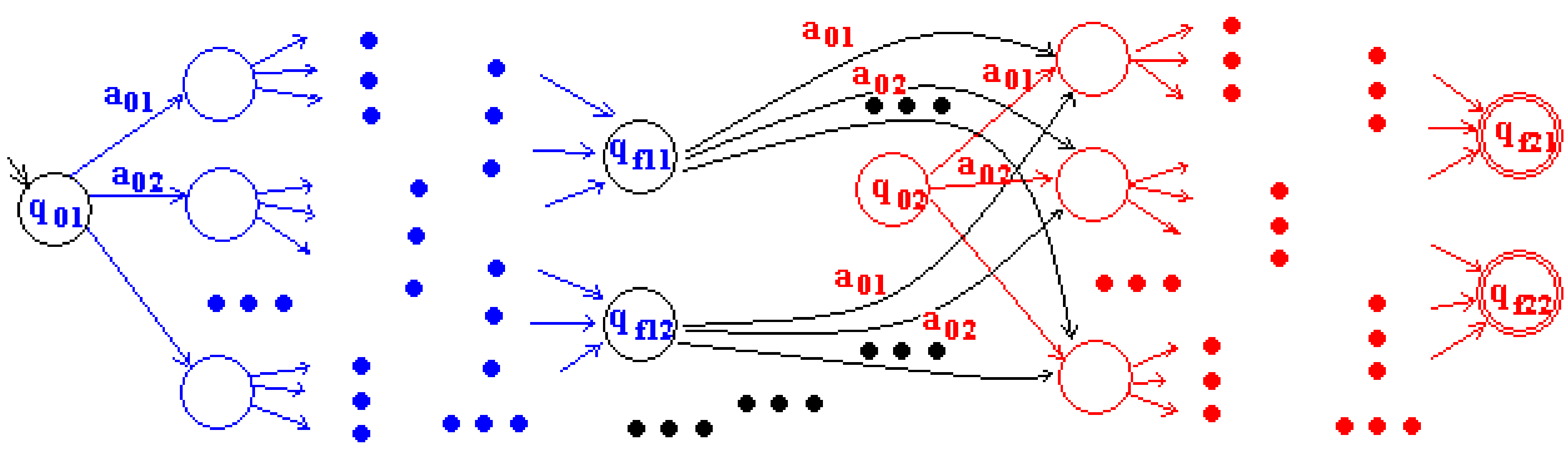
•

- Automatul ce accepta concatenarea limbajelor acceptate de doua automate date
  - AF cu ε tranz.:



•

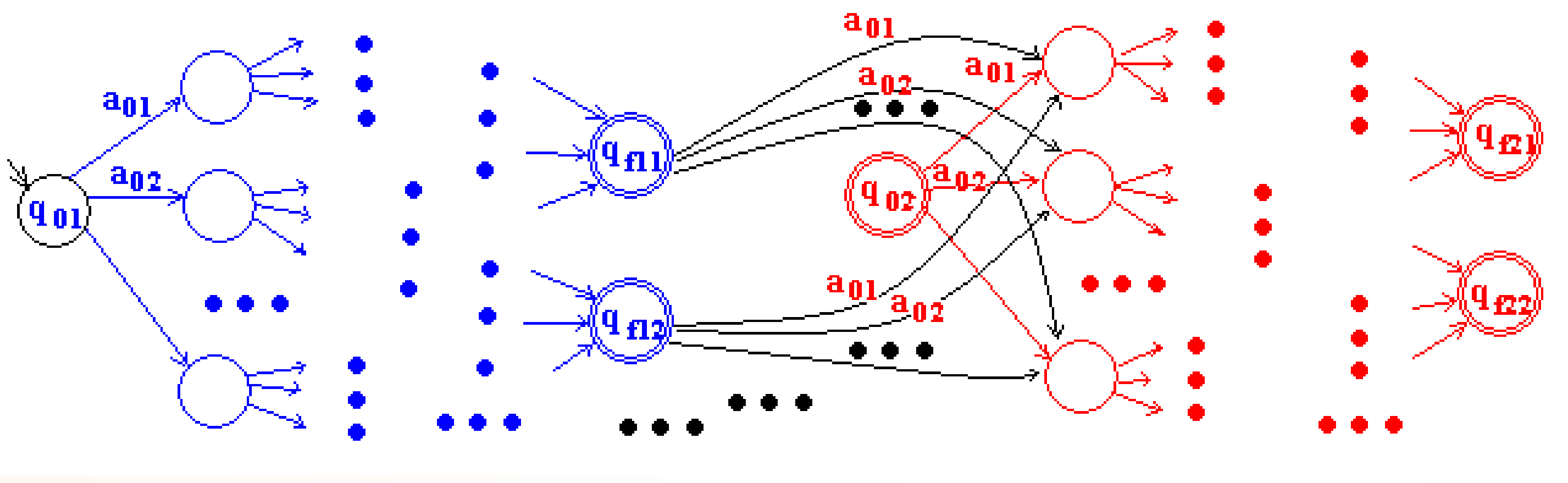
- Automatul ce accepta concatenarea limbajelor acceptate de doua automate date
  - AF



??! q_{02} stare finala

•

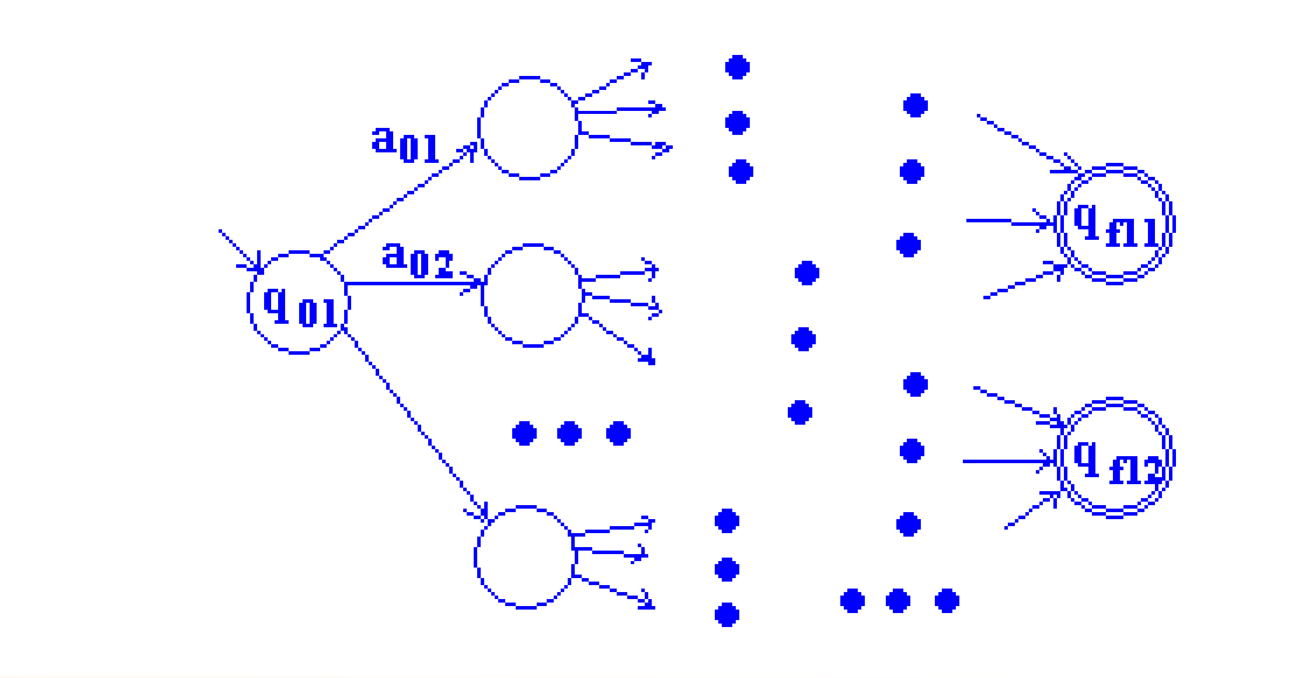
- Automatul ce accepta concatenarea limbajelor acceptate de doua automate date
  - AF



Daca q_{02} stare finala

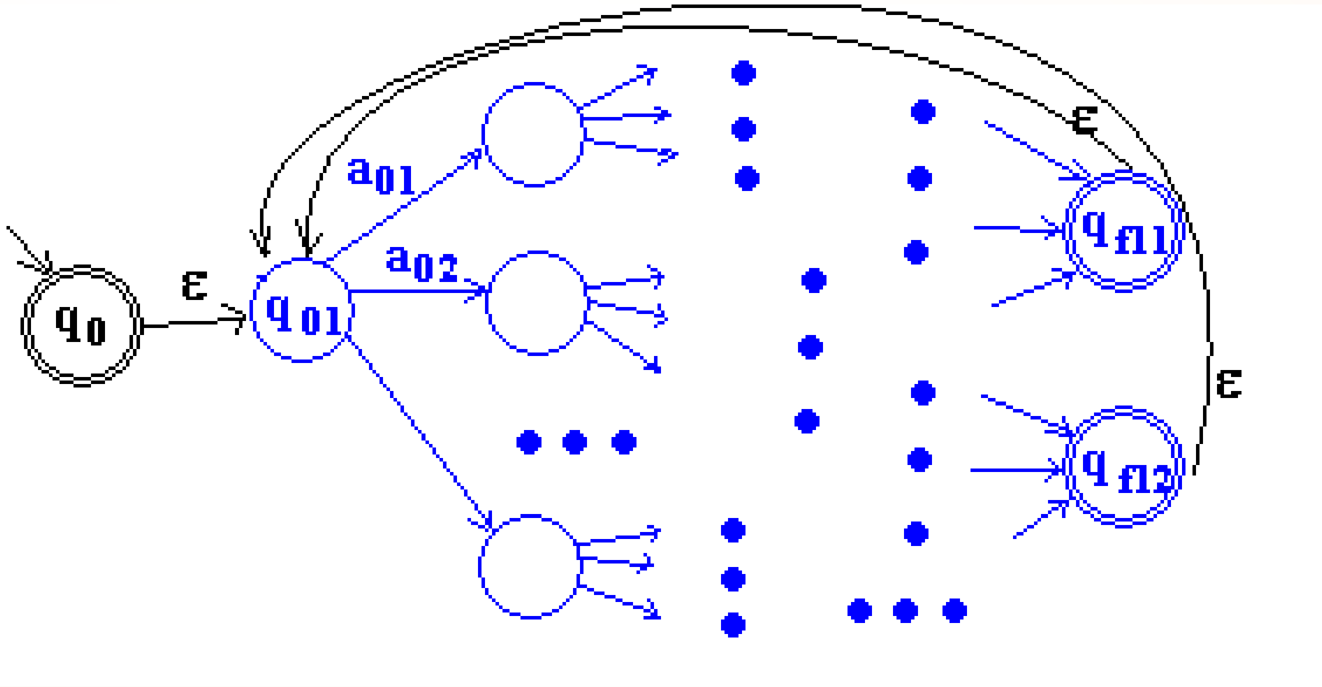
•

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



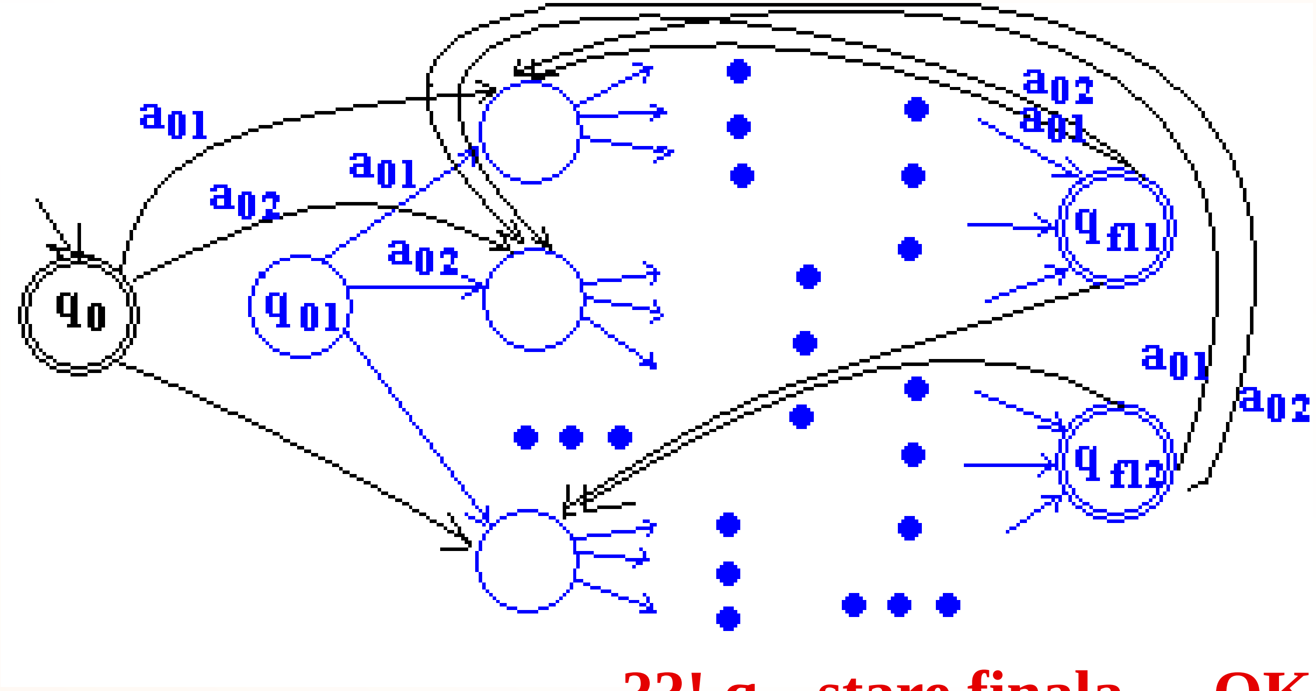
•

- Automatul ce accepta orice secventa peste limbajul acceptat de un automat dat
  - AF cu ε tranz.:



•

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



??! q_{01} stare finala OK !

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>=0

    b^{2n+1} n – nr. natural; n>=0

Exercitii:
Expresie regulara=> limbaj acceptat de AF

- Automatul ce accepta concatenarea limbajelor acceptate de doua automate date
  - se considera AF pt.:

    a^*

    b*
  - se considera AF pt.:

    a

    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

Cursul anterior

Fie Σ un alfabet.

Multimile regulare peste Σ se definesc recursiv astfel:

1. Φ - multime reg. peste Σ
2. {ε} …
3. {a} daca: a ∈ Σ
4. RUS daca R,S – multimi regulare peste Σ +
5. RS daca R,S – multimi regulare peste Σ
6. R* daca R – multime regulara peste Σ
7. Orice alta multime regulara se obtine aplicand de un numar finit de ori reg. 1-6

Multimi regulare si expresii regulare

Cursul anterior

- Expresii regulare

1. Φ expr. reg. coresp. m.reg. Φ
2. ε $\{\varepsilon\}$
3. a daca: $a \in \Sigma$ $\{a\}$
4. r+s daca r,s – expresii regulare **r | s** RUS
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

Cursul anterior

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

  $\Sigma \cup \{\Phi, \varepsilon, +, *, (,)\}$ **(… prioritate …)**
- multimile regulare asociate expresiilor regulare sunt limbaje regulare

=> orice expresie regulara peste Σ

este un limbaj regular

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 Φ si ε)

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

$$\Phi^* = \varepsilon$$
$$r^* + \varepsilon = \varepsilon + r^* = r^*$$
$$(\varepsilon + r)^* = r^*$$
$$(r^*)^* = r^*$$
$$(r^* s^*)^* = (r+s)^*$$

Expresii regulare

Exercitiu:

Fie r, s – expresii regulare oarecare

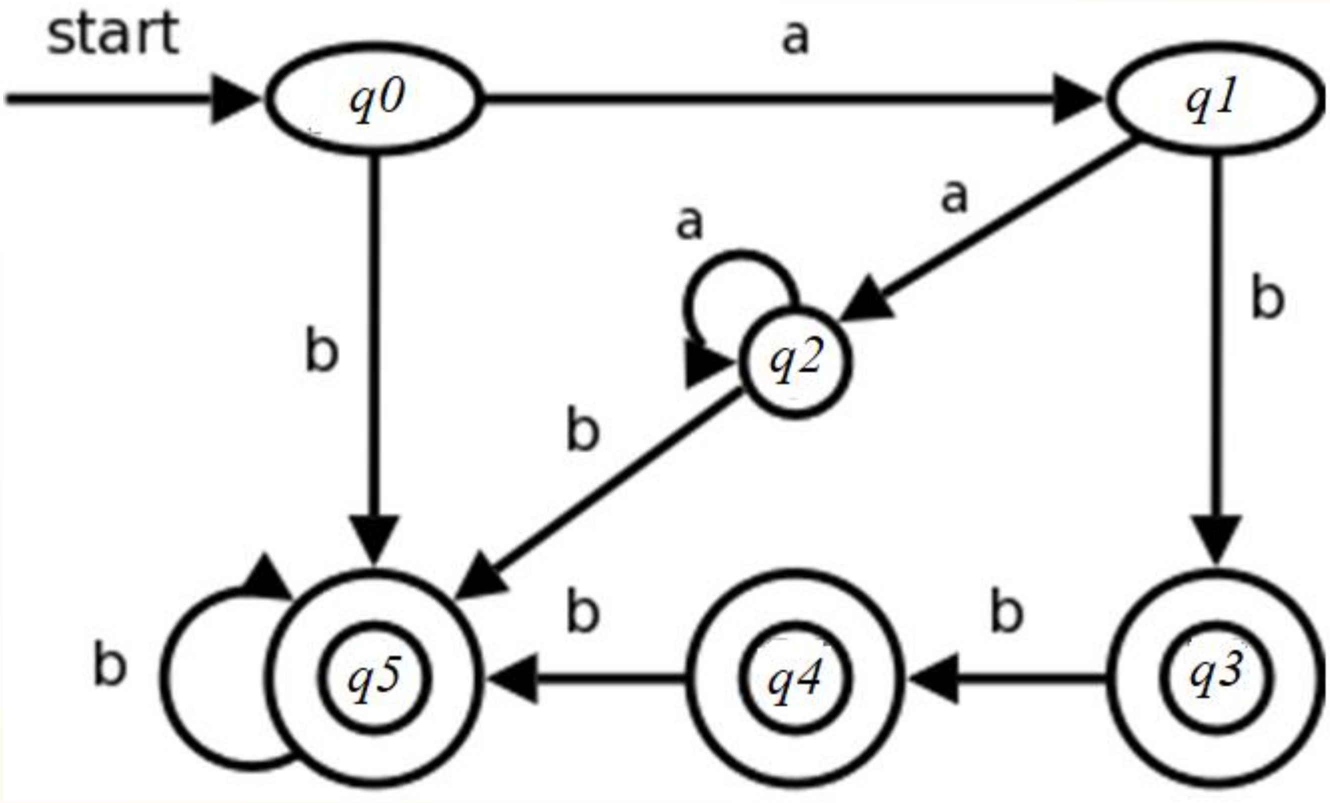
PP ca am demonstrat ca:

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

Demonstrati ca:

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

Expresii regulare si AF (exemplu)



Cine este L(M)?

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

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

Ne reamintim: Gramatica

O gramatica este un cvadruplu **G = (N, Σ , P, S)**

- **N** este un alfabet de simboluri ***neterminale***
- Σ este un alfabet de simboluri ***terminale***
- $N \cap \Sigma = \phi$
- $\mathbf{P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*}$

  P multime finitã **(**multimea regulilor de productie**)**
- $\mathbf{S \in N}$ (simbolul de start - simbolul initial)

Notatie:

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

(α se înlocuieste cu β)

Ne reamintim: clasificarea Chomsky

- Gramatici de tip 0:
  nici o restrictie *(suplimentara)* referitoare la forma regulilor de productie
- Gramaticile de tip 1 (gramatici monotone)
  - $\forall\ \alpha \to \beta \in P$: $|\alpha| <= |\beta|$
  - caz special: $\mathbf{S} \to \boldsymbol{\varepsilon}$ 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 $A \to \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^*$

(gramatici de tip 2)

- Gramaticile de tip 3:
  reg. prod. sunt de forma
  - $A \to aB$
  - $A \to b$ unde $A,B \in N$ si $a,b \in \Sigma$

caz special: $\mathbf{S} \to \boldsymbol{\varepsilon}$ poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

derivari de stanga/ dreapta

- ***derivare de stânga*** $\Rightarrow_{st}$

  o derivare directã in care se înlocuieste cel mai din stânga neterminal

- ***derivare de dreapta*** $\Rightarrow_{dr}$

  o derivare directã in care se înlocuieste cel mai din dreapta neterminal

Analiza sintactica

- ***analizã sintacticã*** pt. cuvantul w

  succesiunea de derivãri directe:

  - $\mathbf{S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = w}$

  altfel spus: reprezintã o derivare pentru cuvântul **w**

- ***analizã sintacticã descendentã***

  dacã aceastã succesiune de derivãri directe se obtine pornind de la **S** si terminand cu w

- ***analizã sintacticã ascendentã***

  dacã aceastã succesiune de derivãri directe se obtine pornind de la w si terminand cu **S**

Arbore de derivare

- **Fie G = (N,Σ,P,S)** o gramaticã independentã de context. Numim ***arbore de derivare*** sau ***arbore de analizã sintacticã*** un arbore cu radacina, ordonat,

  cu urmãtoarele proprietãþi:

1. Orice nod interior - o eticheta din ***N*;**
2. Orice nod frunza - o ***etichetã*** din Σ ∪ {ε}
3. Eticheta rãdãcinii este **S;**
4. Dacã un nod are eticheta A iar nodurile succesoare acestuia, în ordine de la stânga la dreapta sunt etichetate cu $\mathbf{X_1, X_2, \ldots X_n}$ atunci $\mathbf{A \rightarrow X_1X_2 \ldots X_n}$ trebuie sã fie o productie din **P.**

Arbore de derivare

- **frontiera (frontul)**: nodurile terminale, în ordine de la stânga la dreapta
- etichetele lor formeaza o secventa peste Σ^*
- obs: denumirea de frontiera (front) se foloseste si pentru a denumi succesiunea etichetelor nodurilor terminale

<u>Teoremã</u>.

Fie **G = (N, Σ, P, S)** o gramaticã independentã de context. Un cuvânt **w** peste alfabetul **Σ**, deci din **Σ***, apartine limbajului generat de **G**, adicã **w**$\in$ **L(G),** dacã si numai dacã **w** este frontul unui arbore de analizã sintacticã.

Gramatica ambigua

O gramaticã **G = (N, Σ, P, S)** independentã de context este ***ambiguã***

dacã si numai dacã existã cel putin un cuvânt w care admite doi arbori de derivare distincti; în caz contrar gramatica este ***neambiguã***.

- ⇔ ∃ 2 analize sintactice care folosesc numai derivari de stanga, diferite
- ⇔ ∃ 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

ddaca $L(G_1)=L(G_2)$

ε-productii si gram. ε-independente

- ***ε-productie*** : o productie de forma $A \to \varepsilon$
- Gramatica $G = (N, \Sigma, P, S)$ este **ε-independentã** daca:

  a) dacã $\varepsilon \notin L(G)$ atunci G nu are ε-productii

  b) dacã $\varepsilon \in L(G)$ atunci avem o singurã productie $S \to \varepsilon$ iar celelalte productii nu-l contin în membrul drept pe S

- <u>Teorema</u>

  $\forall$ **G= (N, Σ, P, S)**

  $\exists$ **G' = (N', Σ', P', S)** echivalentã, ε-independentã

Forma normalã Chomsky

O gramaticã independentã de context **G = (N, Σ , P, S)** este în ***forma normalã Chomsky*** (***FNC***)

dacã orice regula de productie din **P** este de una din formele:

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

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

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.

<u>Teoremã</u>. Oricare ar fi **G=(N, S, P, S)** o gramaticã independentã de context, întotdeauna existã o gramaticã în forma normalã Chomsky **G'**, astfel încât L(G) = L(G').

Forma normalã Greibach

O gramaticã **G = (N, Σ, P, S)**

este în ***forma normalã Greibach*** (***FNG***)

dacã **P** are productii numai de forma:

$A \rightarrow a\alpha, \qquad A \in \mathbf{N},\ a \in \Sigma,\ \alpha \in \mathbf{N}^*;$

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.

Teoremã. Oricare ar fi **G = (N, Σ, P, S)** o gramaticã independentã de context, întotdeauna existã o gramaticã în forma normalã Greibach, astfel încât **L(G)=L(G')**.

Simplificarea GIC

- ***simbol neproductiv***

  Un simbol $A \in N$ este ***neproductiv*** dacã nu existã nici o derivare de forma $A =^{*}> \mathbf{x}$ $(\mathbf{x} \in \Sigma^{*})$

- în caz contrar **A** este ***simbol productiv***

- Teorema

  $\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã, fãrã simboluri neproductive

Determinarea simbolurilor productive

$A \rightarrow BC$
$B \rightarrow bB$
$C \rightarrow c$

≈ algoritm AF determ. stari productive

1. $i:=0$; $V_0 := \Phi$
2. **Repeta**

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

   $i:=i+1$

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

   {V_i – multimea simbolurilor productive}

Simplificarea GIC

(transformari echivalente)

- ***simbol inaccesibil***

  Un simbol $X \in N \bigcup \Sigma$ este ***simbol inaccesibil*** dacã nu existã nici o * derivare: $S \Rightarrow^{*} \alpha X \beta$ $\quad (\alpha, \beta \in (N \bigcup \Sigma)^{*})$

- în caz contrar simbolul este ***accesibil***
- <u>Teorema</u>

  $\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã, fãrã simboluri inaccesibile

Determinarea simbolurilor accesibile

≈ algoritm AF determ. stari accesibile

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

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

   $i:=i+1$

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

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

* analog pentru simboluri terminale accesibile

Simplificarea GIC

- Un simbol este **neutilizabil** dacã el este fie inaccesibil, fie neproductiv
- <u>Teorema</u>

  $\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã simboluri neutilizabile

•

Observatii:

Fie **G= (N, Σ, P, S)** o gramatica independenta de context:

- Fie un simbol neterminal 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

Pana aici in cursul 5

Eliminarea ε-productiilor

1. Construim multimea N_ε care are ca elemente acele neterminale care prin derivare conduc la ε adicã :

- $N_\varepsilon = \{A \mid A \in N, A \Rightarrow^* \varepsilon\}$

  alg. ≈ 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_ε

1. $i:=0$;

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

2. **Repeta**

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

   $i:=i+1$

   pana 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 ε-productiilor astfel:

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

unde: $B_i \in N_\varepsilon$

α_j nu contine simb. din N_ε

Se inlocuieste cu:

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

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

Ce lipseste ???

determinam noile reguli de productie

- continuare

Dacã $\varepsilon \in L(G)$ trebuie sa avem o ε-productie

“*atunci avem productia $S \to \varepsilon$ si S nu apare în membrul drept al nici unei productii*”

(gram. ε-independenta)

- adaugam un nou simbol de start S' si productiile

$S' \to \varepsilon \mid S$

Redenumiri. Cicluri.

- ***redenumire:*** reg.prod. de forma $\mathbf{A \rightarrow B}$
- ***Gramatica fãrã redenumiri***: fara r.p. de redenumire

  Teorema:

  $\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã redenumiri
- ***ciclu***: o * derivare de forma $\mathbf{A \Rightarrow^* B}$
- ***Gramatica fãrã cicluri***: nu se pot obtine cicluri (la derivare)

  Teorema:

  $\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã cicluri

Eliminarea redenumirilor

PP. G – ε-independenta (daca nu , luam gr.echiv. ε-ind.)

Pentru fiecare **A∈ N**

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

- construieste multimile $N_A = \{B \mid A \Rightarrow^* B\}$;

  (≈ det. simb. accesibile)

- determinam noile reguli de productie

Construieste $N_A = \{D \mid A \overset{*}{\Rightarrow} D\}$

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 $A \rightarrow \alpha \in P$ **executa**
- **daca** α 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:

$E \rightarrow E+T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow a$

Gramatica fara cicluri

Teorema:

$\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã cicluri

Daca G – ε-independenta si fara redenumiri atunci este fara cicluri

Gramatica proprie:

este o gramatica

- fara simb. neutilizabile
- **ε-independenta**
- fara cicluri

<u>Teorema</u>:

$\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** proprie echiv.

Recursivitate

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

Reg. prod. recursive la stanga

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

<u>Teorema</u>:

$\forall$ **G= (N, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã reg.prod. recursive la stanga

- **PP. G – gr. proprie**

  (daca nu este, det. gr. proprie echiv. si lucram cu ea)
- Obs.: vom obtine tot o gramatica proprie

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

  $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_r$ (r.p. recursive)

  $A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_s$ (r.p. ne-recursive)

- r.p. se transforma astfel:

  $A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_s \mid \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_s A'$

  $A' \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_r \mid \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_r A'$

  (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-> Sa

S->a

Recursivitate

- **neterminal recursiv la stanga**:
  $A \in N$ daca $\exists$ o derivare de forma: $A \stackrel{+}{\Rightarrow} A\alpha$
- **neterminal recursiv la dreapta**:
  $A \in N$ daca $\exists$ o derivare de forma: $A \stackrel{+}{\Rightarrow} \alpha A$
- **neterminal recursiv**:
  $A \in N$ daca $\exists$ o derivare de forma: $A \stackrel{+}{\Rightarrow} \alpha A \beta$
- **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, Σ, P, S)** $\exists$ **G' = (N', Σ', P', S)** echivalentã fãrã neterminale recursive la stanga

- PP. G – gr. proprie
  (daca nu este, det. gr. proprie echiv. si lucram cu ea)

- impunem o ordine asupra neterminalelor
  $N = \{A_1, A_2, \ldots, A_n\}$
  si apoi modific r.p. a.i. sa nu existe $A_i \rightarrow A_j\alpha$ cu $j \le i$
  de aici => 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 $\boldsymbol{A_i} \rightarrow A_j\alpha$ *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 β 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)

$A \rightarrow a \mid aB$

$B \rightarrow AC \mid b$

$C \rightarrow BA \mid c$

Lema de pompare pentru limbaje independente de context

Fie **L** un limbaj independent de context. Existã atunci o constantã **p** dependentã numai de **L** astfel cã dacã **$z \in$ L si $|z| >= p$,** atunci avem descompunerea **z=uvwxy** cu proprietãtile:

a) **$|vx| >= 1$**,

b) **$|vwx| <= p$**,

c) **$uv^iwx^iy \in L$** $\forall\ i \in$ **N**

?

S -> 0A1
A -> 0S
A -> a

Lema de pompare pentru limbaje independente de context

- *pentru claritate: sa vedem si definitia formala*

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

Proprietãti de închidere ale limbajelor independente de context

<u>Teoremã</u>.

Dacã L_1si L_2 sunt limbaje independente de context atunci:

$L_1 \cup L_2$, L_1L_2 , L_1^*

sunt limbaje independente de context.

<u>Observatie</u>:

$L1 \cap L2$, compl(L1) - nu sunt neaparat l.i.c.

Exercitii

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

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

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

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

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

$L_1 \cup L_2$, $L_1 L_2$, 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 ddaca:
  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: $nr_a(\mathbf{z})=nr_b(\mathbf{z})=nr_c(\mathbf{z})$)

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 satisfice
  - |**z**|>=**p**
  - $\exists$ o descompunere **z** = **uvwxy** astfel incat: $\mathbf{uv^iwx^iy} \in L$, $\forall$ i $\in$ N
    si |**vx**| **>=1**
    si |**vwx**|<=**p**

Dem., Versiunea 1:

Alegem **z** cu |**z**|>=**p** (satisface cond. de mai sus)

- $\exists$ **n** a.i. $|a^n b^n c^n|$ >=**p** ;   z $\in$ L => $z = a^n b^n c^n$ si |**z**|>=**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)**

<u>**cazul 1**</u>: *(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:
$uv^2wx^2y = u\ a^{k1}\ \underline{\mathbf{b}^{\mathbf{k2}}\ \mathbf{a}^{\mathbf{k1}}}\ b^{k2}\ wx^2y \in L$,
atunci cand k1>0 si k2>0 *(cf.* ***rel.1)***
ar insemna ca simb. **b** pot sa apara inaintea simb. **a**
ceea ce nu e adevarat pentru cuvintele din L
(observatia (a.)(i.))
=> 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 $uv^2wx^2y \in L$
... => 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 $uv^2wx^2y \in L$
... => contradictie

cazul 2: *(dintre cazurile posibile pentru cazul 2 aleg unul dintre ele si dem. pt. el)*
fie: $v = a^{k1}$ k1>=0
$x = a^{k2}$ k2>=0

Stim ca: $|\mathbf{vx}| \mathbf{>=1}$
⇔ $|\mathbf{a}^{k1}\mathbf{a}^{k2}| \mathbf{>=1}$
⇔ k1+ k2 > 0 ***(rel.2)***
(k1, k2 – nu sunt simultan 0)
atunci: $u = a^{k3}$, k3>=0
$w = a^{k4}$, k4>=0
$y = a^{n-k1-k2-k3-k4}b^nc^n$, n-k1-k2-k3-k4 >=0
fie i =2: cf. lemei: $uv^2wx^2y \in L$
$uv^2wx^2y = a^{k3}\ a^{2*k1}\ a^{k4}\ a^{2*k2}\ a^{n-k1-k2-k3-k4}b^nc^n$
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
=> n+k1+k2 = n
=> k1+k2 = 0
dar *(cf.* ***rel.2)*** : k1+k2>0
=> 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 $nr_a(z')=nr_b(z')=nr_c(z')$
=> contradictie

cazul 3: *(dintre cazurile posibile pentru cazul 3 aleg unul dintre ele si dem. pt. el)*
fie: $v = a^{k1}$, k1>0 (rel.4)
$x = b^{k2}$, k2>0 (rel.5)
atunci: $u = a^{k3}$, k3>=0

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

fie $i =2$; atunci $uv^2wx^2y \in L$
$uv^2wx^2y = a^{k3}\ a^{2*k1}\ a^{n-k1-k2}b^{n-k2-k4}\ b^{2*k2}\ b^{k4}c^n$
$z' = uv^2wx^2y \in L \Rightarrow nr_a(z')=nr_b(z')=nr_c(z')$
$k3+2*k1+n-k1-k3 = n-k2-k4 + 2*k2 +k4 = n$
$\Rightarrow n+k1 = n+k2 = n$
$\Rightarrow k1=0$ contrad cu (rel.4)
($\Rightarrow k2=0$, contrad. cu (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 $nr_a(z')=nr_b(z')=nr_c(z')$
=> <u>contradictie</u>

<u>**cazurile posibile pt. cazul 1**</u>
$z = a^nb^nc^n$, $z = uvwxy$
cel putin unul dintre v si x contin cel putin 2 simboluri (dintre a,b,c) diferite;
$v = a^{k1}b^{k2}$, $k1>0, k2>0$ si nu specificam ce poate contine x
$v = a^{k1}b^{k2}\ c^{k3}$, $k1>0, k2>0, k3>0$ si nu specificam ce poate contine x
$v = b^{k2}\ c^{k3}$, $k2>0, k3>0$ si nu specificam ce poate contine x
daca v contine un singur acelasi simbol, ne situam in cazul 1 daca:
$x = a^{k1}b^{k2}$, $k1>0, k2>0$
$x = a^{k1}b^{k2}\ c^{k3}$, $k1>0, k2>0, k3>0$
$x = b^{k2}\ c^{k3}$, $k2>0, k3>0$

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

<u>Exercitiu</u>:
descrieti cazurile posibile pt. cazul 2 si cazul 3

Dem., Versiunea 2 (scurta ☺):

Alegem **z** cu |**z**|>=**p** (satisface cond. de mai sus)

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

- => |**z**|>=**p**
- z = uvwxy descompunerea din lema de pompare

- astfel incat: $\mathbf{uv^iwx^iy} \in L, \forall i \in N$

  si |**vx**| **>=1**

  si |**vwx**|<=**p**

Pentru ca |**vwx**|<=**p** : secventa **vwx** contine maxim 2 simboluri dintre a, b,c.
Astfel, in secventa $\mathbf{uv^iwx^iy}$ exista cel putin un simbol care nu este “pompat” si cel putin unul care este “pompat” ; astfel se pierde egalitatea dintre numarul de aparitii ale celor doua 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;
- Σ alfabetul de intrare;
- Γ alfabetul memoriei stivă; ;
- $q_o \in Q$ stare iniţială;
- $Z_o \in \Gamma$ simbolul de start al memoriei stivă;
- $F \subseteq Q$ mulţimea stărilor finale;
- $\delta: Q x (\Sigma \cup \{\varepsilon\}) x \Gamma \rightarrow \mathcal{P}(Q x \Gamma^*)$ funcţia de tranziţie

Reprezentare

- enumerare
- tabelara
- sub forma de graf

Reprezentare folosind enumerare

Exemplu:

- $M=(\{q_0,q_1\},\{a,b\},\{A,Z\}, \delta,q_0,Z,\{q_0\})$
- δ:

$\delta(q_0,a,Z) = \{(q_0,AZ)\}$

$\delta(q_0,a,A) = \{(q_0,AA)\}$

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

$\delta(q_1,b,A) = \{(q_1,\varepsilon)\}$

$\delta(q_0,\varepsilon,Z) = \{(q_0,\varepsilon)\}$

$\delta(q_1,\varepsilon,Z) = \{(q_0,\varepsilon)\}$ … si Φ in celelalte cazuri

Reprezentare tabelara

Exemplu:

		a	b	ε	
q_0	Z	(q_0, AZ)		(q_0, ε)	1
	A	(q_0, AA)	(q_1, ε)		
q_1	Z			(q_1, ε)	0
	A		(q_1, ε)		

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 găseşte în starea **q**, pe banda de intrare urmează să se citească (accepte) secvenţa **x**, iar în memoria stivă avem secvenţa **α**
- configuratie initiala

$$(q_0,w,Z_0)$$

Tranzitii

- $\vdash$ ***tranziţie directă***

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

  sau

  $(q,aw,Z\alpha) \vdash (p,aw,\gamma\alpha) \Leftrightarrow \delta(q,\varepsilon,Z) \ni (p,\gamma)$ **(ε-tranziţie)**

  – unde $\mathbf{p,q} \in \mathbf{Q}$, $\mathbf{a} \in \mathbf{\Sigma}$, $\mathbf{Z} \in \Gamma$, $w \in \Sigma^*$, $\boldsymbol{\alpha,\gamma}$ din $\mathbf{\Gamma^*}$
- $\vdash^{k}$ ***k* tranziţia** (k tranziţii directe) ~AF
- $\vdash^{+}$ + ***tranziţia*** ~AF
- $\vdash^{*}$ * ***tranziţia*** ~AF

Secventa acceptata de automat

- dupa criteriul stivei vide

  $L_\varepsilon(M) = \{w |\ w \in \Sigma^*, (q_o,w,Z_o) \vdash^* (q,\varepsilon,\varepsilon), q \in Q\}$

  – $(q,\varepsilon,\varepsilon)$ - configuraţia finală după criteriul stivei vide
- dupa criteriul starii 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,ε,γ), $q \in F$ configuraţie finală după criteriul stării finale

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

		a	b	ε	
q_0	**Z**	**(q_0, AZ)**		**(q_0, ε)**	**1**
	A	**(q_0, AA)**		**(q_1, A)**	
q_1	**Z**			**(q_1, ε)**	**0**
	A		**(q_1, ε)**		

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 secventa 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

Pornind de la:

- un APD care accepta un limbaj dupa criteriul stivei vide

construiti

- un APD echivalent

  care obtine acelasi limbaj dupa criteriul starii finale

Teoreme de echivalenta

<u>Teoremă</u>.

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

<u>Teoremă</u>.

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

si reciproc.

G.I.C. => APD echivalent

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

Cine este M - APD astfel incat $L(G)=L_{\varepsilon}(M)$ **?**

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. $\delta(q,a,a) = \{(q,\varepsilon)\}\ \forall\ a \in \Sigma$;
3. $\delta(.,.,.) = \Phi$ în celelalte cazuri.

Determinism

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

este ***determinist*** ddacă:

$\forall \mathbf{Z} \in \Gamma, \forall \mathbf{q} \in \mathbf{Q}, \forall \mathbf{a} \in \Sigma$

1) $|\boldsymbol{\delta}(\mathbf{q},\varepsilon,\mathbf{Z})| = \mathbf{0}$ si $|\boldsymbol{\delta}(\mathbf{q},\mathbf{a},\mathbf{Z})| <= \mathbf{1}$

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

in caz contrar, automatul nu este determinist

- multimea limbajelor acceptate de APD nedeterministe este strict mai larga decat multimea limbajelor acceptate de APD deterministe

Analizorul descendent cu reveniri

- Automat: configuratii, tranzitii
- Se poate folosi pentru

  gramatici nerecursive la stanga

Analizorul descendent cu reveniri

Configuratie:

(s,i,α,β)

- 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 secventa de intrare
- α – stiva de lucru: istoria r.p. aplicate
- β – stiva de intrare: partea inca neprelucrata

- configuratie initiala: $(q,1,\varepsilon,S)$
- Tranzitii:
  - expandare: $(q,i,\alpha,A\beta) \vdash (q, i, \alpha A_1, \gamma_1\beta)$
  - avans: $(q,i,\alpha,a_i\beta) \vdash (q, i+1, \alpha a_i, \beta)$
  - insucces de moment: $(q,i,\alpha,a\beta) \vdash (r,i,\alpha,a\beta)$ $\quad a<>a_i$
  - succes: $(q,n+1,\alpha,\varepsilon) \vdash (t,n+1,\alpha,\varepsilon)$
  - revenire: $(r,i,\alpha a,\beta) \vdash (r,i-1,\alpha,a\beta)$
  - alta incercare: $(r,i, \alpha A_j, \gamma_j\beta) \vdash \ldots$

    daca $\exists\ A_{j+1}\rightarrow\gamma_{j+1}$ $\quad \vdash (q,i,\alpha A_{j+1},\gamma_{j+1}\beta)$

    altfel daca $i=1, A=S$ $\quad \vdash (e,i,\alpha,A\beta)$

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

    altfel $\quad \vdash (r,i,\alpha,A\beta)$

Analizorul descendent cu reveniri

Observatii:

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

Analizorul descendent cu reveniri

Gramatici recursive $\rightarrow$ curs 5

Analiza sintactica descendenta. Gramatici si analiza LL(K)

•

$FIRST_k$

- $FIRST_k: (N \cup \Sigma)^* \rightarrow Part(\Sigma^*)$
- $FIRST_k(\alpha) = \{u \in \Sigma^* \mid (\alpha \overset{*}{\Rightarrow} ux, |u| = k)$
  $\text{sau } (\alpha \overset{*}{\Rightarrow} u, |u| < k) \}$

FOLLOW$_k$

$FOLLOW_k: (N \cup \Sigma)^* \rightarrow Part(\Sigma^*)$

$FOLLOW_k(\alpha) = \{u \in \Sigma^* \mid S =^{*}> \gamma\alpha\beta,\ u \in FIRST_k(\beta)\}$

Operatia $\oplus_k$

Fie L1, L2 – 2 limbaje peste Σ

$L1 \oplus_k L2 = \{w \mid \exists\ x \in L1, y \in L2$ astfel incat:

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

fie: $|w| < k$ si $xy = w$

$\}$

<u>Exercitii</u>: $L1 = \{a, ab, abb, abbb, \ldots\}$

$L2 = \{c, cc, ccc, cccc, \ldots\}$

$L1 \oplus_3 L2 = \ldots$

$L1 \oplus_1 L2 = \ldots$

Determinarea lui $FIRST_1$

- pentru fiecare $A \in N$:

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

  sf.pentru

  $i = 0$

- repeta

  $i = i+1$

  pentru fiecare $A \in N$:

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

  sf. pentru

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

- $FIRST_1 = F_i$

<u>Obs</u>:

$\forall a \in \Sigma$: $FIRST_1(a) = \{a\}$

$\forall A \in N$: $FIRST_1(A)$ se determina pe baza reg. productie: $A \to \dots$

$FOLLOW_1$

- FOLL (S) = {\$}
  FOLL (X) = Φ (X<>S)
- repeta

  pentru fiecare B: $A \rightarrow \alpha B \beta$ executa

  FOLL (B) = FOLL (B) $\cup$ ($FIRST_1(\beta) - \{\varepsilon\}$)

  Daca $\varepsilon \in FIRST_1(\beta)$ atunci

  FOLL (B) = FOLL (B) $\cup$ FOLL (A)

  sf. daca

  sf. pentru

  pana cand FOLL nu se mai modifica

  $FOLLOW_1$ = FOLL

Proprietati:

$FIRST_1(\alpha\beta) = FIRST_1(\alpha) \oplus_1 FIRST_1(\beta)$

$FIRST_1(X_1 \ldots X_n) =$ … **?**

- $A \to \alpha$

  $FIRST_1(A) \supset FIRST_1(\alpha)$

- $A \to \alpha\ X\ \beta$

  $(FIRST_1(\beta) - \{\varepsilon\}) \subset FOLLOW_1(X)$

- $A \to \alpha\ X$

  $FOLLOW_1(A) \subset FOLLOW_1(X)$

Exercitii: $FIRST_1$ si $FOLLOW_1$

Problema:

- determinati $FIRST_1$ si $FOLLOW_1$ pentru toate terminalele si neterminalele gramaticii cu regulile de productie:

$S \rightarrow BA$	(1)
$A \rightarrow +BA$	(2)
$A \rightarrow \varepsilon$	(3)
$B \rightarrow DC$	(4)
$C \rightarrow *DC$	(5)
$C \rightarrow \varepsilon$	(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

Gramatici LL(k)

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

  ? $w = a_1 a_2 \ldots a_n \in L(G)$

  $S \Rightarrow_{st} \ldots \Rightarrow_{st} a_1 a_2 \ldots a_i\ A\ \alpha$

  (derivari de stanga)
- LL(k) : alegerea r.p. pentru a-l retranscrie pe A este unic determinata de: $a_{i+1} a_{i+2} \ldots a_{i+k}$
- terminologie:
  - parte inchisa: $a_1 a_2 \ldots a_i$
  - predictia (de lungime k): $a_{i+1} a_{i+2} \ldots a_{i+k}$
  - simbol de retranscris: A

Gramatici LL(k)

Definitie:

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

- Daca:
  - $S =^{*}>_{st} wA\alpha =>_{st} w\,\beta\,\alpha =^{*}> w\,x$
  - $S =^{*}>_{st} wA\alpha =>_{st} w\,\gamma\,\alpha =^{*}> w\,y$
  - $FIRST_k(x) = FIRST_k(y)$
- atunci:

$$\beta=\gamma$$

k=1

•

Gramatici LL(1)

- ? $w=a_1a_2\ldots a_ia_{i+1}\ \ldots\ a_n \in L(G)$
  $S =>_{st} \ldots =>_{st} a_1a_2\ldots a_i\ A\ \alpha$
- LL(1): rescrierea lui A este unic determin. de $\mathbf{a_{i+1}}$

De exemplu:

Aleg:	**atunci cand:**
$\mathbf{A\to\gamma}$	$a_{i+1} \in FIRST_1(\gamma)$
$\mathbf{A\to\varepsilon}$	$a_{i+1} \in FOLLOW_1\ (A)$
$\mathbf{A\to\delta\quad (=^{*}>\varepsilon)}$	$a_{i+1} \in FOLLOW_1\ (A)$

Gramatici LL(1)

Teorema:

G – este de tip LL(1)

ddaca:

$\forall A \in N$

$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_m$

- $FIRST_1(\alpha_i) \cap FIRST_1(\alpha_j) = \Phi$, i<>j
- daca $\exists$ i a.i. $\alpha_i \overset{*}{=}> \varepsilon$ atunci: **$\varepsilon \in FIRST_1(\alpha_i)$**

$FIRST_1(\alpha_j) \cap FOLLOW_1(A) = \Phi$, i<>j

Analiza sintactica LL(1)

- analizorul sintactic LL(1)

  ~ modelare ca un **automat**

- se construieste tabelul de analiza LL(1)
  - alegerea lui $A \rightarrow \alpha_i$ este indicata in tabel

---

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 => (membrul drept al reg.prod , nr. reg.prod.)

Tabelul de analiza LL(1)

M(X,a) =

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

Analizorul LL(1)

- Automat: (α,β,Π)
  - banda de intrare: α (stiva de intrare)
  - stiva β (stiva de lucru)
  - banda de iesire Π => sirul regulilor de productie
- config. initiala: (w\$, S\$, ε)
- config. finala: (\$, \$, *Π*)
- tranzitii
  - push (**a**x\$, **A**β, Π) $\vdash$ (ax\$,αβ,Π**i**) dc.: M(A,a) = (α,i)
  - pop (**a**x\$, **a**β, Π) $\vdash$ (x\$, β, Π)
  - acc (\$, \$, Π) $\vdash$ acc
  - err in celelalte cazuri

Transformari echivalente

- => gram. LL(1) echiv. ?
- factorizare la stanga

…

Gramatici LR(k)

- analizoare LR(k)

  - analiza sintactica ascendenta

  - 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:

$S \rightarrow AB$ (1)

$A \rightarrow a$ (2)

$B \rightarrow b$ (3)

si intrarea: ab ?

Gramatica LR(K)

Analizoarele sintactice LR(k) lucreaza cu gramatica imbogatita:

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

$(S' \notin N)$

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 $k>=0$

ddaca din:

- $S' =^{*}>_{dr} \alpha\ A\ w =>_{dr} \alpha\ \beta\ w$
- $S' =^{*}>_{dr} \gamma\ B\ x =>_{dr} \alpha\ \beta\ y$
- $FIRST_k(w) = FIRST_k(y)$

rezulta ca:

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

Gramatici LR(K) - terminologie

Prefix viabil

Fie: $S \overset{*}{\Rightarrow}_{dr} \alpha A w \Rightarrow_{dr} \alpha \beta w$

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

Element de analiza LR(k)

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

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

Element de analiza valid

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

- $S \overset{*}{\Rightarrow}_{dr} \gamma A w \Rightarrow_{dr} \gamma \alpha \beta w$
- $u = FIRST_k(w)$

Analizor sintactic LR(K)

- la multimea cuv. de analizat se adauga la sfarsit $
  – $ - marcator de sfarsit de cuvant

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

Colectia canonica LR(K)

$\mathcal{C}$ = {I_i-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)

Terminologie: I_i – stare a automatului

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

Constructia colectiei canonice LR(k)

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

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

- $I_0 = \mathit{Closure}([S' \rightarrow .S , \ldots])$
- $\mathcal{C} = \{ I_0 \}$
- repeta

    pentru toti I_i din $\mathcal{C}$, $X \in (N \cup \Sigma)$ executa

        $\mathcal{C} = \mathcal{C} \cup \mathit{goto}(I_i, X)$

    sf. pentru

  pana cand $\mathcal{C}$ nu se mai modifica

K=0: LR(0)

Gramatica imbogatita

- se adauga S’
  - nou simbol de start
  - S’→S

Colectia canonica:

- In I_0 avem: [S’ → .S]
- …

Functia *Closure* LR(0)

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

daca e = [A $\rightarrow \alpha$. Bβ]

atunci $\forall$ B$\rightarrow \delta \in$ P: [B $\rightarrow$. δ] $\in$ *Closure* (e)

Functia *goto* LR(0)

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

Tabelul de analiza LR(0)

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

- $T(I_i, \text{actiune}) =$
  - s (shift, deplasare)

    daca: $[A \to \alpha.\beta] \in I_i$, $\beta <> \varepsilon$

    si: $T(I_i, X) = I_j$, daca $I_j = goto(I_i, X)$
  - L (reducere cu r.p. nr. L)

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

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

    si: $T(I_i, X)$ nu se completeaza
  - acc daca: $[S' \to S.] \in I_i$

  Toate celelalte cazuri se considera eroare .

Tabelul de analiza LR(0)

Automatul LR(0) – model matematic

- configuratie:

  (α,β,Π)

  (stiva_de_lucru, banda_de_intrare, banda_de_iesire)
- pe stiva: prefixe viabile, stari ale analizorului

- config. initiala: ($0,w$, ε)
- config. finala: ($0S I_{acc} , $, Π)

Tranzitii

- deplasare:

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

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

- reducere

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

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

  si: $A \rightarrow X_p\ldots\ X_k$ – r.p. cu nr. $\boldsymbol{L}$

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

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

- acceptare: $(\ \$\ 0S\ s_{acc}\ ,\ \$\ ,\ \Pi\ )\ \vdash$ acc.
- eroare: *orice alta situatie*

Automatul LR(0) – model matematic

- 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:

  $[A\rightarrow\alpha.\beta, u]$

  $u = FOLLOW_1(A)$

  $|u| = 1$

- SLR: tine cont de predictie numai pentru reducere

Analiza sintactica SLR

- constructia colectiei canonice (~LR(0))
  - $[A \rightarrow \alpha.\beta, u]$, $u = FOLLOW_1(A)$
- constructia tabelului de analiza SLR
  - actiunea de reducere depinde de predictia u

=>reducerea va avea o coloana pentru fiecare $a \in \Sigma$

tabelul: linii: elementele colectiei canonice

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

celula: $s_{stare}, r_{nr.r.p}$, acc

- analizorul ~ analizorul pt. LR(0)

automat: configuratii si tranzitii

Tabelul de analiza SLR

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

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

linii: elementele colectiei canonice

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

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

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

- *Closure*

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

$$B \to \gamma \qquad\qquad \forall\ b \in FIRST_1(\beta a)$$

- *goto*

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

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

Tabelul LR(1)

	stare	shift + reduce $N \cup \Sigma \cup \$$
	I_0	
(I_{acc})	I_1	
	I_2	
	⋮	

Construirea tab. de analiza LR(1)

- $[A \to \alpha.X\beta,b] \in I_i$: *goto* $(I_i,X) = I_{\mathbf{j}}$ <= functia *goto*

  $action(I_i,X) = sj$
- $[A \to \alpha. , a] \in I_i$ $action(I_i,a) = rL$

  L – nr. reg. de productie: $A \to \alpha$

  A <> S'
- $[S' \to S., \$] \in I_i$ $action(I_i,\$) = acc$

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

- $[\underbrace{A \longrightarrow \alpha.\beta}_{\text{nucleu}}, a]$

- colectia canonica LR(1)
- fuzioneaza elementele de analiza cu nuclee identice si care nu creeaza conflicte
- predictia: reuniunea predictiilor

- tabelul LALR & analiza : similar LR(1)

LR (1 –uri)

- Conflict:

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

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

$[\ A \rightarrow \alpha_1.\ \ ,\ \ a\ ]$ reducere-reducere

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

Bibliografie:

S. MOTOGNA

Metode de proiectare a compilatoarelor, 2006

	descendent	ascendent
cu reveniri / recursiv	descendent cu reveniri	*ascendent cu reveniri*
liniar	LL(k)	LR(k)

Analizoare sintactice

LL(k)

- **L**: “**L**eft-to-right” (scaneaza intrarea)
- **L**: “**L**eft-most derivation”
- k simboluri de lookahead

descendent

~ *dezvoltare* in pre-ordine a arborelui de analiza sintactica

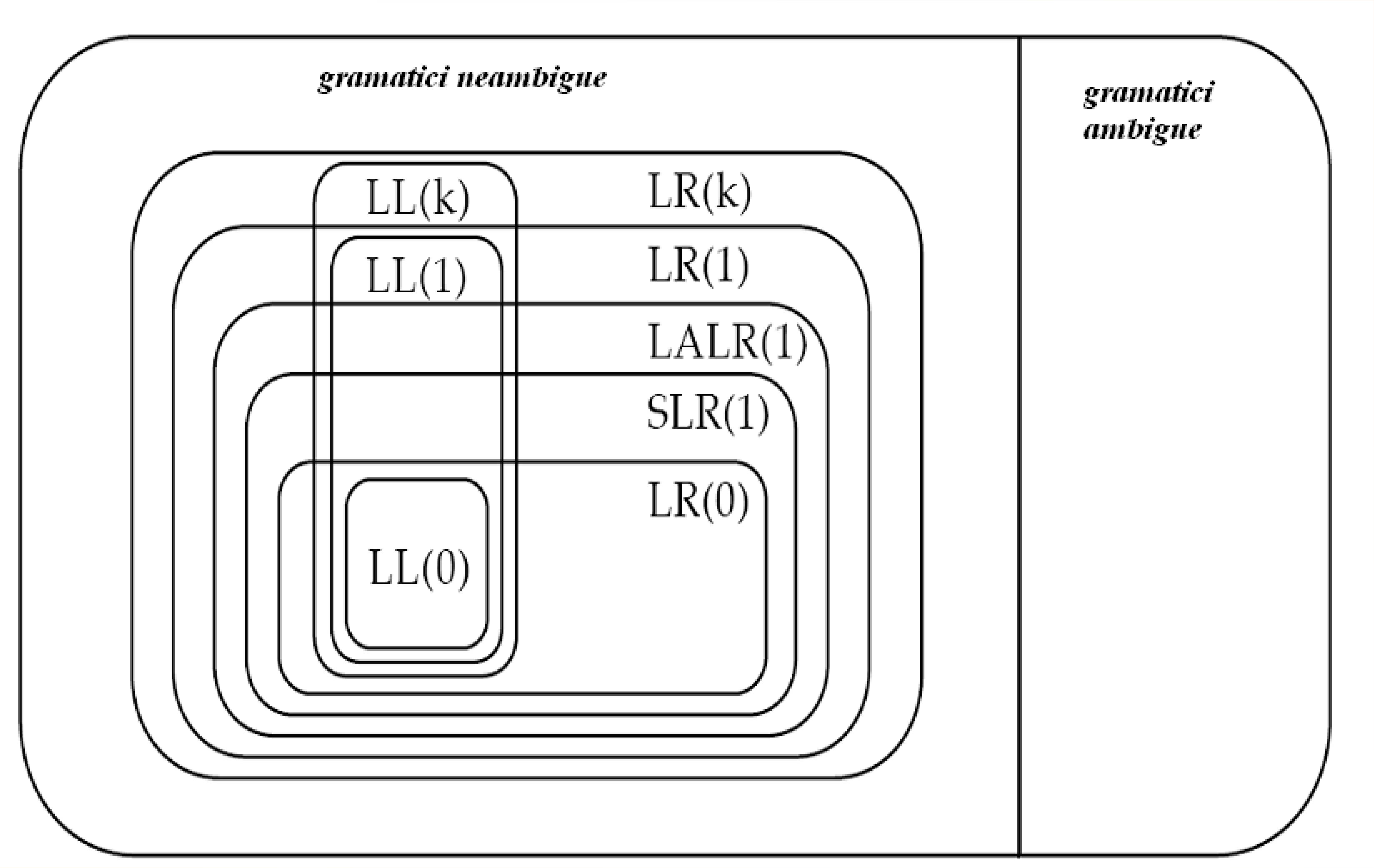
LR(k)

- **L**: Scaneaza intrarea “**L**eft-to-right”
- **R**: “**R**ight-most derivation”
- k simboluri de lookahead

ascendent

~ *traversare* in post-ordine a arborelui de analiza sintactica

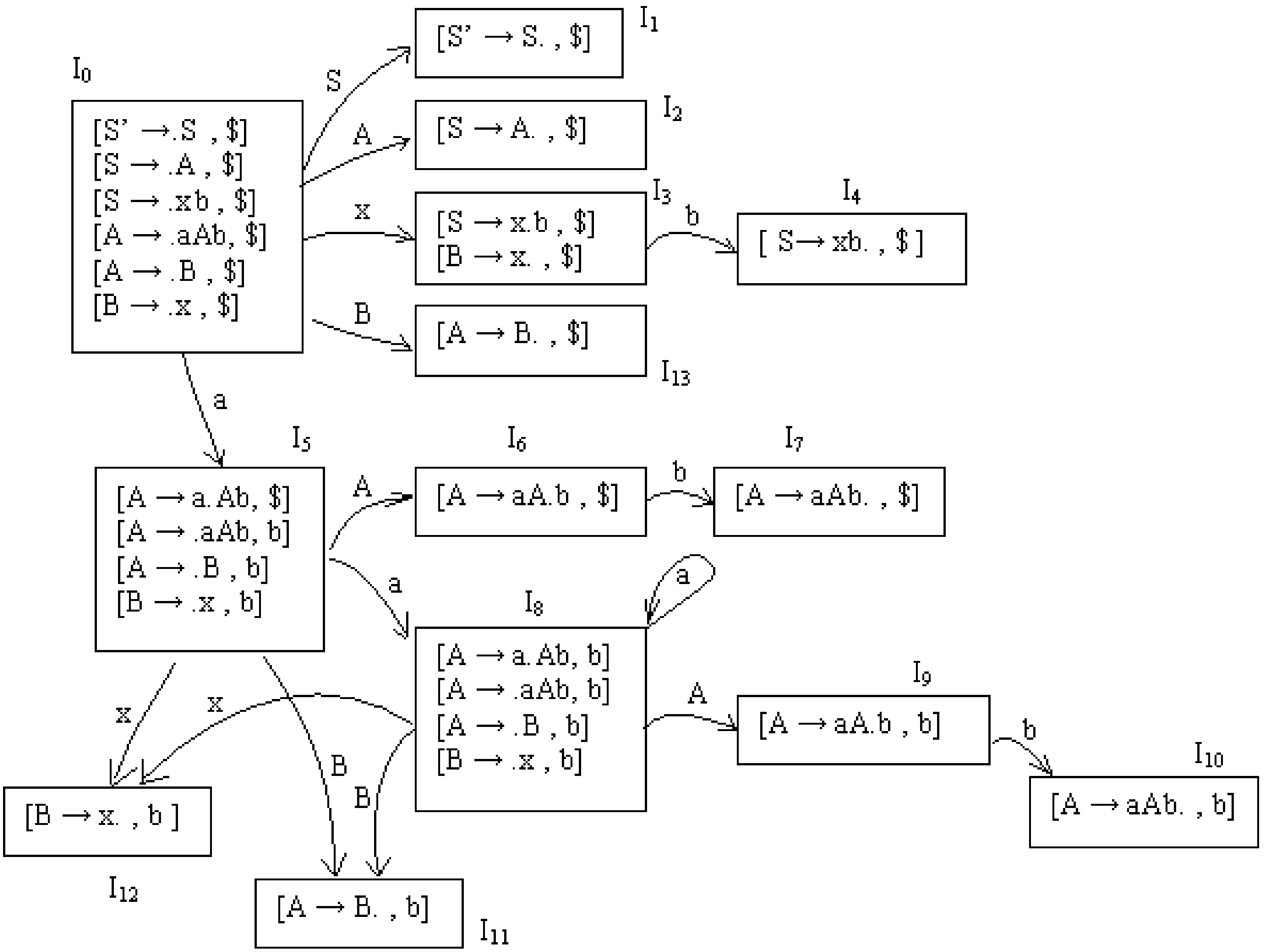
LL vs LR



Clase de limbaje si relatia dintre ele

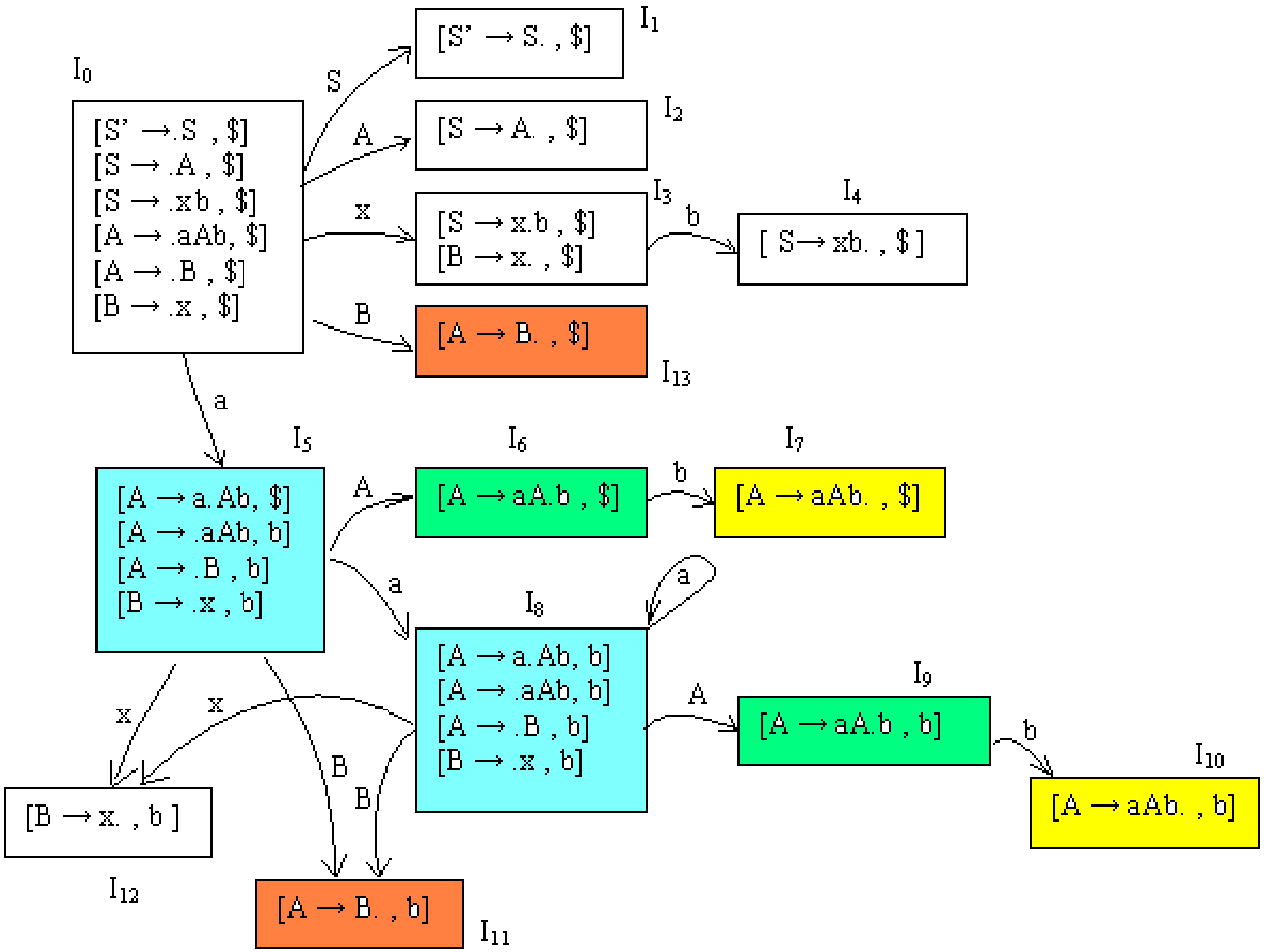
Analiza LR(1)

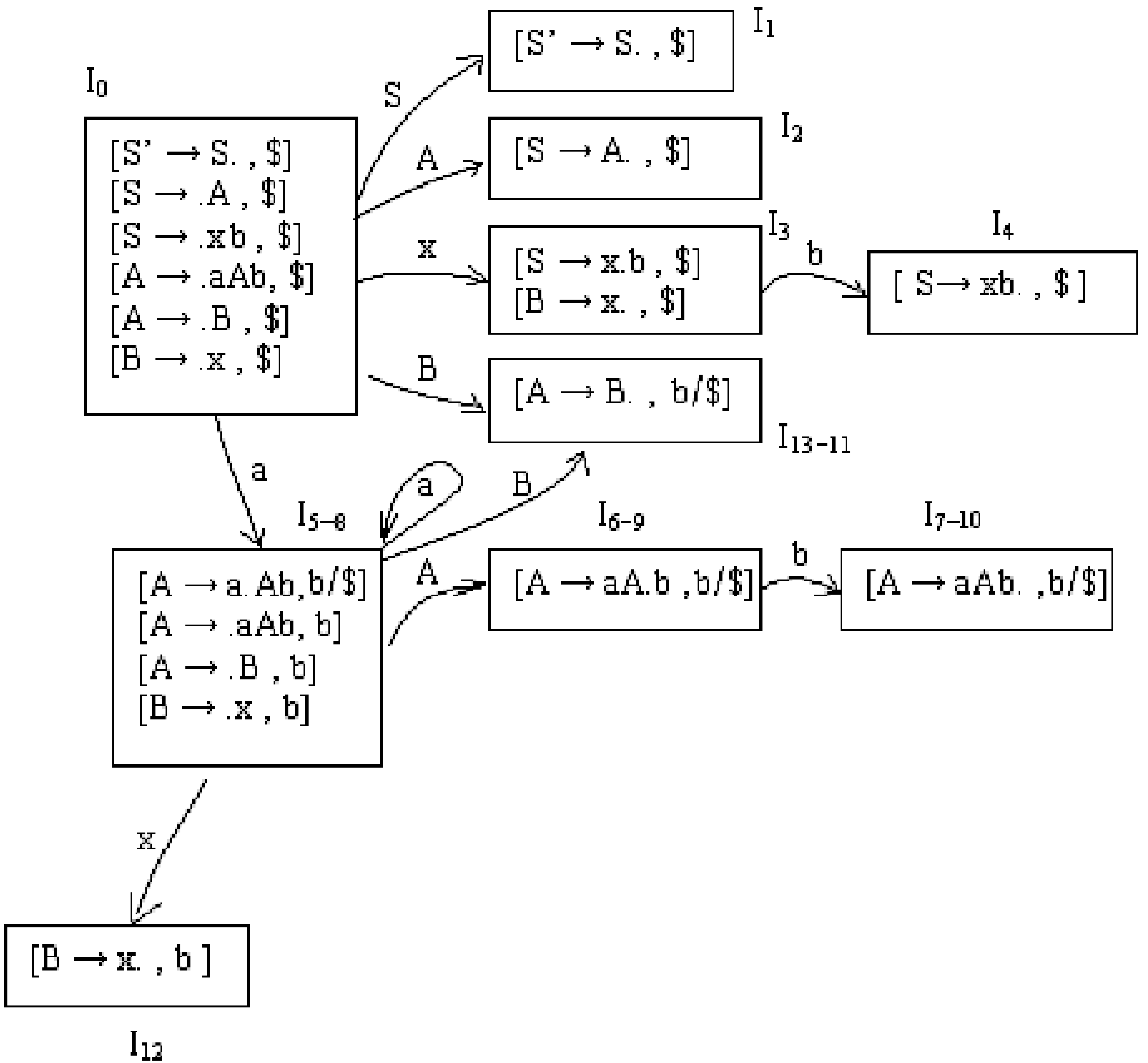
$S' \rightarrow S$	0
$S \rightarrow A$	1
$S \rightarrow xb$	2
$A \rightarrow aAb$	3
$A \rightarrow B$	4
$B \rightarrow x$	5


I0
[S' → .S , $]
[S → .A , $]
[S → .xb , $]
[A → .aAb, $]
[A → .B , $]
[B → .x , $]
S
I1
[S' → S. , $]
A
I2
[S → A. , $]
x
I3
[S → x.b , $]
[B → x. , $]
b
I4
[S→ xb. , $]
B
[A → B. , $]
I13
a
I5
[A → a. Ab, $]
[A → .aAb, b]
[A → .B , b]
[B → .x , b]
A
I6
[A → aA.b , $]
b
I7
[A → aAb. , $]
a
I8
a
[A → a. Ab, b]
[A → .aAb, b]
[A → .B , b]
[B → .x , b]
A
I9
[A → aA.b , b]
b
I10
[A → aAb. , b]
x
x
[B → x. , b]
I12
B
B
[A → B. , b]
I11

	S	A	B	a	b	x	$
I_0	s1	s2	s13	s5		s3	
I_1							acc
I_2							r1
I_3					s4		r5
I_4							r2
I_5		s6	s11	s8		s12	
I_6					s7		
I_7							r3
I_8		s9	s11	s8		s12	
I_9					s10		
I_{10}					r3		
I_{11}					r4		
I_{12}					r5		
I_{13}							r4

Analiza LALR


I0
[S' → .S , $]
[S → .A , $]
[S → .xb , $]
[A → .aAb, $]
[A → .B , $]
[B → .x , $]
S
I1
[S' → S. , $]
A
I2
[S → A. , $]
x
I3
[S → x.b , $]
[B → x. , $]
b
I4
[S→ xb. , $]
B
[A → B. , $]
I13
a
I5
[A → a. Ab, $]
[A → .aAb, b]
[A → .B , b]
[B → .x , b]
A
I6
[A → aA.b , $]
b
I7
[A → aAb. , $]
a
a
I8
[A → a. Ab, b]
[A → .aAb, b]
[A → .B , b]
[B → .x , b]
A
I9
[A → aA.b , b]
b
I10
[A → aAb. , b]
x
x
[B → x. , b]
I12
B
B
[A → B. , b]
I11



I0
[S' → .S , $]
[S → .A , $]
[S → .xb , $]
[A → .aAb, $]
[A → .B , $]
[B → .x , $]
S
A
x
B
a
I1
[S' → S. , $]
I2
[S → A. , $]
I3
[S → x.b , $]
[B → x. , $]
b
I4
[S→ xb. , $]
[A → B. , b/$]
I13-11
a
B
I5-8
[A → a. Ab,b/$]
[A → .aAb, b]
[A → .B , b]
[B → .x , b]
A
I6-9
[A → aA.b ,b/$]
b
I7-10
[A → aAb. ,b/$]
x
[B → x. , b]
I12

	S	A	B	a	b	x	$
I_0	s1	s2	s11-13	s5-6		s3	
I_1							acc
I_2							r1
I_3					s4		r5
I_4							r2
$I_{5\text{-}8}$		s6-9	s11-13	s5-8		s12	
$I_{6\text{-}9}$					s7-10		
$I_{7\text{-}10}$					r3		r3
$I_{11\text{-}13}$					r4		r4
I_{12}					r5		

bibliografie:
Motogna, cap. 4.5

G – gramatica de atribute

G = (N, Σ, 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 atribute

  : multime finita de atribute

- $\mathcal{R} = \bigcup_{p \in P} \mathcal{R}(p)$
  - fiecarei reguli de productie i se asociaza
    o multime finita de expresii ale atributelor
    asociate simbolurilor regulii de productie

  => reguli de evaluare ale atributelor

Atribute

asocierile atribut - valoare

sunt definite numai peste o “analiza sintactica”

un arbore de derivare

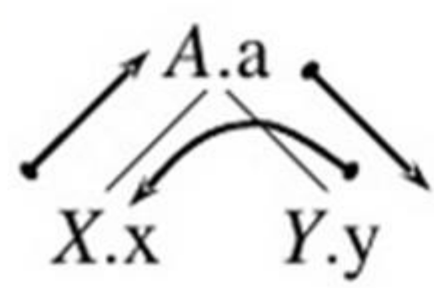
Evaluator de atribute

- calculeaza valori & propaga valorile calculate
- traverseaza arborele de derivare

- strategie de traversare a arborelui

  si propagare a valorilor

•



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

Dandu-se o gram. de atribute, este graful necircular pentru orice arbore de derivare?

- ! 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

Evaluarea atributelor

Metode de evaluare

- metode bazate pe arborele de derivare

    determina ordinea de evaluare

    pe baza sortarii topologice a grafului de dependenta

    construit pentru arborele de derivare

    pentru fiecare secventa de intrare

- metode bazate pe reguli

    ordinea de evaluare este determinata / fixata

    la nivelul la care se definesc regulile semantice

- metode bazate pe o ordine pre-fixata

    ordinea de evaluare este fixata si regulile semantice trebuie

    definite astfel incat sa respecte ordinea data

Atribute

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

$A \rightarrow X1 \ldots Xk$

- atribut sintetizat:
  - un atribut al lui A
  - regula de evaluare atribuie valoare atributului lui A
- atribut mostenit:
  - atribut al lui Xi
  - regula de evaluare atribuie valoare atributului lui Xi

El depinde de valorile parintilor si fratilor.

Gramatica S-atributata

Def:

exista doar atribute sintetizate

si 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-> X1 X2 … Xn

- un atribut mostenit a lui Xi depinde de atribute mostenite ale lui A si de atribute ale lui X1, X2, …, Xi-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 descendent Xi : (X1 , X2, …, Xn)
                evalueaza atributele mostenite ale lui Xi
                viziteaza (Xi)
        sf. pentru
        evalueaza atributele sintetizate ale lui A
endSubalg.
```

GA, DDS

Gramatică de atribute (GA)

- gramatica independenta de context
- atribute + expresii ale atributelor

in unele surse se foloseste acelasi termen

Definiţii dirijate de sintaxă (DDS)

(EN: Syntax directed definition)

- gramatica independenta de context
- atribute + reguli de calcul ale atributelor
  - apeluri de proceduri sau fragmente de program
  - pot avea efecte laterale

(Daca nu au, atunci sunt gramatici de atribute)

Scheme de traducere: **(vezi tema de laborator 6)**

(EN: Syntax directed translation)

definitie orientata sintaxa

\+ alte actiuni ex.: fragment de program

se execută atunci când este întâlnit în parcurgerea arborelui

EX: A -> α { print('x')} β

se va afişa caracterul 'x' după ce se vizitează subarborele α

şi înainte de traversarea subarborelui β.

Gramatica de atribute. Exemplu

S -> A	A.a <- A.x
A_0 -> 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
A -> b	A.y <- A.a A.x <- 1
A -> bb	A.x <- A.b A.y <- 1

Modele de calcul

Masini de calcul abstracte idealizate, simplificate

organizarea masinii

– actiuni (instructiuni) + valorile cu care operam (operanzii)

Masina cu registri

– registri: valorile cu care operam
– instructiuni:

operanzii: registri de intrare , registru de iesire

Masina cu stiva *calculator de tip stiva*

– operanzii – in stiva
– instructiuni: opereaza asupra varfului stivei

Masina cu registri

Operatori pentru exemplele cu care vom lucra:
load, add, sub, mul, div

Exemplu:
Fie expresia aritmetica

A+B

Codul pentru masina cu registri echivalent expresiei aritmetice date este:

```
load A, r0
load B, r1
add  r0, r1, r2
```

Ce stim sa facem?
Dandu-se o expresie aritmetica
cu operatorii: … (add, sub, mul, div), stim sa determinam:
codul pentru masina cu registri

Masina cu stiva

Modelul se bazeaza pe faptul ca orice expresie poate fi scrisa in forma poloneza postfixata.

In exemplele cu care vom lucra consideram o **masina cu stiva** si 6 operatii: push, pop, add, sub, mul, div

Ce stim sa facem?
Dandu-se o expresie aritmetica, stim sa determinam:
a) forma poloneza postfixata
b) codul pentru masina cu stiva

Exemplu:
Fie expresia aritmetica

B + C - D

Forma poloneza postfixata este

B C + D -

Codul pentru masina cu stiva echivalent expresiei aritmetice date este:

```
push B
push C
add
push D
sub
```

Masina cu stiva / masina cu registri

Exemplu: 2+3

- Masina cu stiva

  Push 2
  Push 3
  Add

- Masina cu registri

  Load 2, r0
  Load 3, r1
  Add r0, r1, r2

Alte modele de calcul

+ combinatii

- Masina cu acumulator

  acumulator : un registru special

  - operand implicit

  - rezultate stocate in acumulator

  + …
- Masina cu acumulator si stiva

  stiva + acumulator (registru special)

…   …

Java bytecode : exemplu

```
i = j + k;
if (i == 3)
    k = 0;
else
   j = j – 1;
```

```
        ILOAD j     // i = j + k
        ILOAD k
        IADD
        ISTORE i
        ILOAD i     // if (i == 3)
        BIPUSH 3
        IF_ICMPEQ L1
        ILOAD j     // j = j – 1
        BIPUSH 1
        ISUB
        ISTORE j
        GOTO L2
L1:     BIPUSH 0        // k = 0
        ISTORE k
L2:
```

Generarea codului intermediar

Cod intermediar cu trei adrese.
Exemple.

Generarea codului intermediar

In urmatoarele exemple,

pentru generarea codului intermediar vom atributa gramatica ce descrie structurile sintactice.

Vom folosi ca atribute:

- nume de variabile (*pt. expresii*)
- cod (*cod cu trei adrese*)
- nume de etichete

Vom folosi niste functii speciale

- *newVarName* - creeaza (nume de) variabile
  (*newTemp …)*
- *newLabel* - creeaza nume de etichete

Exemplu:

Fie secventa: a-b+c

O gramatica independenta de context, simplificata, care descrie sintaxa expresiei artimetice date:

$E \rightarrow E + ID$

$E \rightarrow E - ID$

$E \rightarrow ID$

$ID \rightarrow a$

$ID \rightarrow b$

$ID \rightarrow c$

Vrem sa obtinem: codul cu trei adrese echivalent semantic

T1 := **a - b**

T2 := T1 + **c**

$E \rightarrow E_1 + ID$ $\quad$ $E.varn = newVarName()$

$E.cod = E_1.cod \;||$

$E.varn := E_1.varn \; + ID.varn$

$E \rightarrow E_1 - ID$ $\quad$ $E.varn = newVarName()$

$E.cod = E_1.cod \;||$

$E.varn := E_1.varn \; - ID.varn$

$E \rightarrow ID$ $\quad$ E.cod= “ ”

E.varn = *ID.varn*

$ID \rightarrow a$ $\quad$ ID.varn= a

$ID \rightarrow b$ $\quad$ ID.varn= b

$ID \rightarrow c$ $\quad$ ID.varn= c

Gramatica atributata

Exercitiu:

Dati o gramatica independenta de context (forma simplificata) care descrie sintaxa expresiilor aritmetice , a.i.

a+b-c

sa fie un cuvant al gramaticii.

(Gramatica va respecta structurile lexicale, sintactice si semantice!)

a) atributati gramatica:

- introduceti (cel putin) atributul cod
  - cod - cod intermediar cu 3 adrese, reprezentare cvadruple
- definiti reg. evaluare pt. atributul cod
- folositi si alte atribute daca e nevoie

b) dati arborele de derivare

c) descrieti evaluarea atributului cod pentru exemplul dat

Exercitiu:

Dati o gramatica independenta de context (forma simplificata) care descrie sintaxa instructiunii if , a.i.

if a > b then max:=a else max:=b

este un cuvant al gramaticii.

(Gramatica va respecta structurile lexicale, sintactice si semantice!)

a) atributati gramatica:

- introduceti (cel putin) atributul cod
  - cod - cod intermediar cu 3 adrese, reprezentare cvadruple
- definiti reg. evaluare pt. atributul cod
- folositi si alte atribute daca e nevoie

b) dati arborele de derivare

c) descrieti evaluarea atributului cod pentru exemplul dat

Exercitiu:

Se cere:

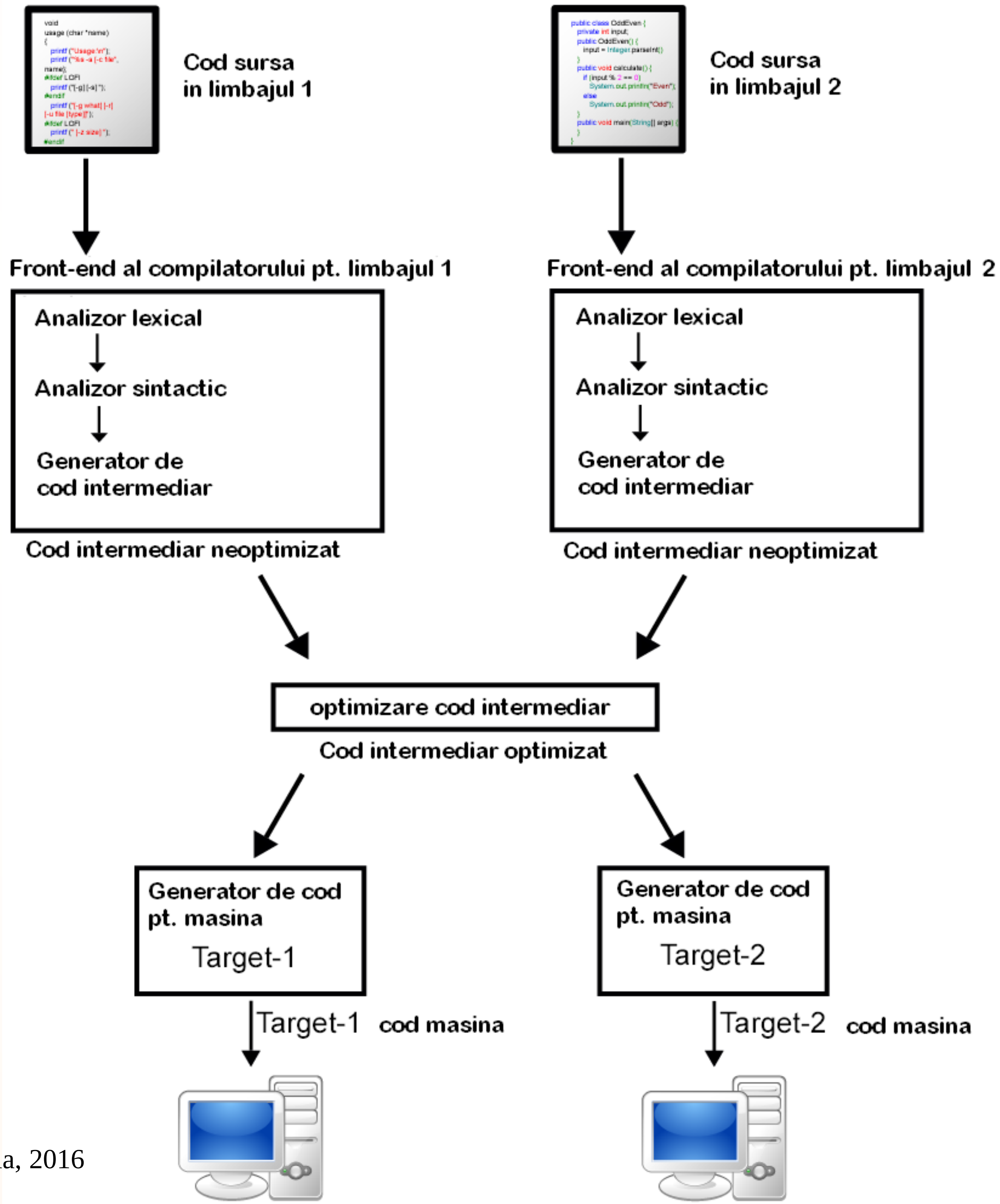
cod echivalent cu trei adrese,

reprezentare cvadruple, pentru:

while a>=b do a:=a-b

Compilator: multi-limbaj, multi-target (*masina*)

in stransa legatura cu separarea fazelor



Sursa imagine: Wikipedia, 2016

Exemple:

CIL **- Common Intermediate Language**

– Microsoft .Net

anterior cunoscut sub numele MSIL

(Microsoft Intermediate Language)

RTL - register transfer language

- GNU Compiler Collection
- multe alte compilatoare

Codul intermediar

- limbaj intermediar:
  - usor de transcris din arborele sintactic
  - usor de translatat in cod masina
  - proiectat inclusiv pt. a fi inteles/utilizat de oameni
  - mai apropiat de limbajul procesorului decat limbajul sursa

Reprezentari intermediare:

"*intre*" arborele de analiza sintactica si ASM

- high-level: mentine structura limbajului
- mid-level: independent de limbaj si masina ***(tinde sa fie)***
- low-level: dependent de masina

Codul intermediar

- Arbore sintactic abstract
- Forma poloneza postfixata
- Cod intermediar cu 3 adrese

Reprezentari:

- cvadruple
- triplete
- triplete indirecte

Arbore sintactic abstract

“***Abstract syntax trees***”

Alte denumiri **(intalnite in literatura de specialitate, in lb/ romana)**

arbore de sintaxa abstracta, arbore sintactic

- reprezentare apropiata de structura sintactica a programelor
- nu este arborele de derivare, ci o varianta simplificata

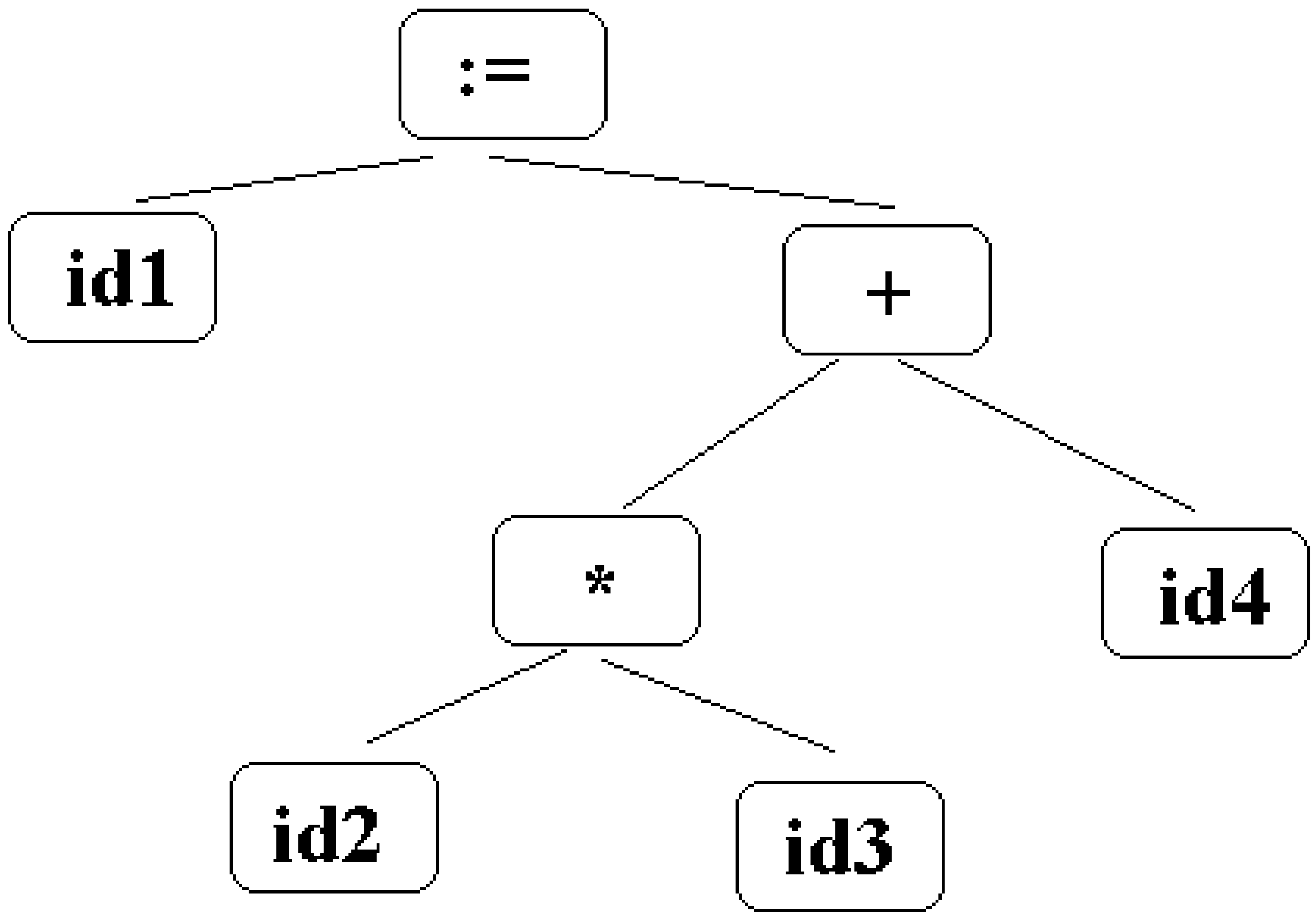
Proprietati:

- nodurile interioare sunt operatori
- descendentii unui nod sint *operanzii* lui

➔ fiecare subarbore formeaza o "*unitate logica*"

Arbore sintactic abstract

Exemplu: id1:=id2*id3+id4



Arbore sintactic abstract

<u>Exercitii propuse</u>:

Descrieti arborele sintactic abstract pentru urmatoarele instructiuni:

- if id1>id2 then id3:=id2
  else id3:=id1
- while id1>id2 do
  id1:=id2-id1

Forma poloneza

- aceeasi idee ca si la:

  forma poloneza postfixata pentru expr. aritmetice

<u>Exemplu</u>:

id1 := id2 * id3 + id4   =>   id1 id2 id3 * id4 + :=

<u>Proprietate</u>:

- operatorii apar in ordinea in care se executa operatiile

<u>Avantaj</u>:

evaluarea se face parcurgand o singura data expresia si executand operatiile tinand cont de aritatea lor

Codul intermediar

- limbaj intermediar:
  - mai apropiat de limbajul *procesorului* decat limbajul sursa

? limbaj de asamblare

Ne amintim: in mod uzual, o masina de tipul x86 are:

- Registri
- Memorie, incluzand: stiva , programul propriu-zis
- Instructiunile limbajului masina, incluzand instr. de
  - mutare a datelor intre registri si memorie
  - calcul: aritmetice ,…

Cod intermediar cu 3 adrese

- secventa de instructiuni cu forma generala:

  <rezultat> := <arg1> <operator> <arg2>
- operatii:
  – binare
  – unare – se reprezinta doar un operand
- reprezentare
  – cvadruple
  – triplete
  – triplete indirecte

Pentru triplete si triplete indirecte vom lucra numai cu exemple cu atribuiri si expresii aritmetice.

3 adrese – reprezentare cvadruple

- structura tip inregistrare ce contine 4 campuri:

operator	arg1	arg2	rez

Exemplu:

A := B*(C+D)

operator	arg1	arg2	rez
…	…	…	…
+	C	D	T1
*	B	T1	T2
:=	T2		A

3 adrese – reprezentare triplete

- structura tip inregistrare ce contine 3 campuri:

operator	arg1	arg2

- se renunta la introducerea numelor temporare ce stocheza rezultate intermediare
- se considera ca instructiunea care calculeaza o valoare temporara retine acea valoare

3 adrese – reprezentare triplete

Exemplu:

A := B*(C+D)

	operator	arg1	arg2
…	…	…	…
(51)	+	C	D
(52)	*	B	(51)
(53)	:=	A	(52)

3 adrese – reprez. triplete indirecte

- codul contine instructiunile intr-o ordine oarecare
- pentru a obtine ordinea in care se executa operatiile, se foloseste un tabel suplimentar cu 2 campuri:

nr. de ordine a operatiei	nr. operatiei propriu-zise

3 adrese – reprez. triplete indirecte

nr. de ordine a operatiei	nr. operatiei propriu-zise
51	131
52	132
53	133

	operator	arg1	arg2
…	…	…	…
(131)	+	C	D
(132)	*	B	(131)
(133)	:=	(132)	

Cvadruple. Conventii cu care vom lucra noi

- operanzi: constanta numerica
  valoarea unei variabile
- **operanzi speciali**
  - **@ adresa variabilei**
  - **^ variabila de la adresa indicata de valoarea variabilei**
- operatii
  - aritmetice binare: +, * , …
  - aritmetice unare: -
  - de atribuire (copiere): :=
  - relatii < , > , …
  - salt neconditionat goto et
  - salt conditionat g<operlogic> exp1 exp2 et

Optimizare cod intermediar

rearanjarea codului intermediar in vederea obtinerii unui program mai eficient

optimizari locale

Exemple:

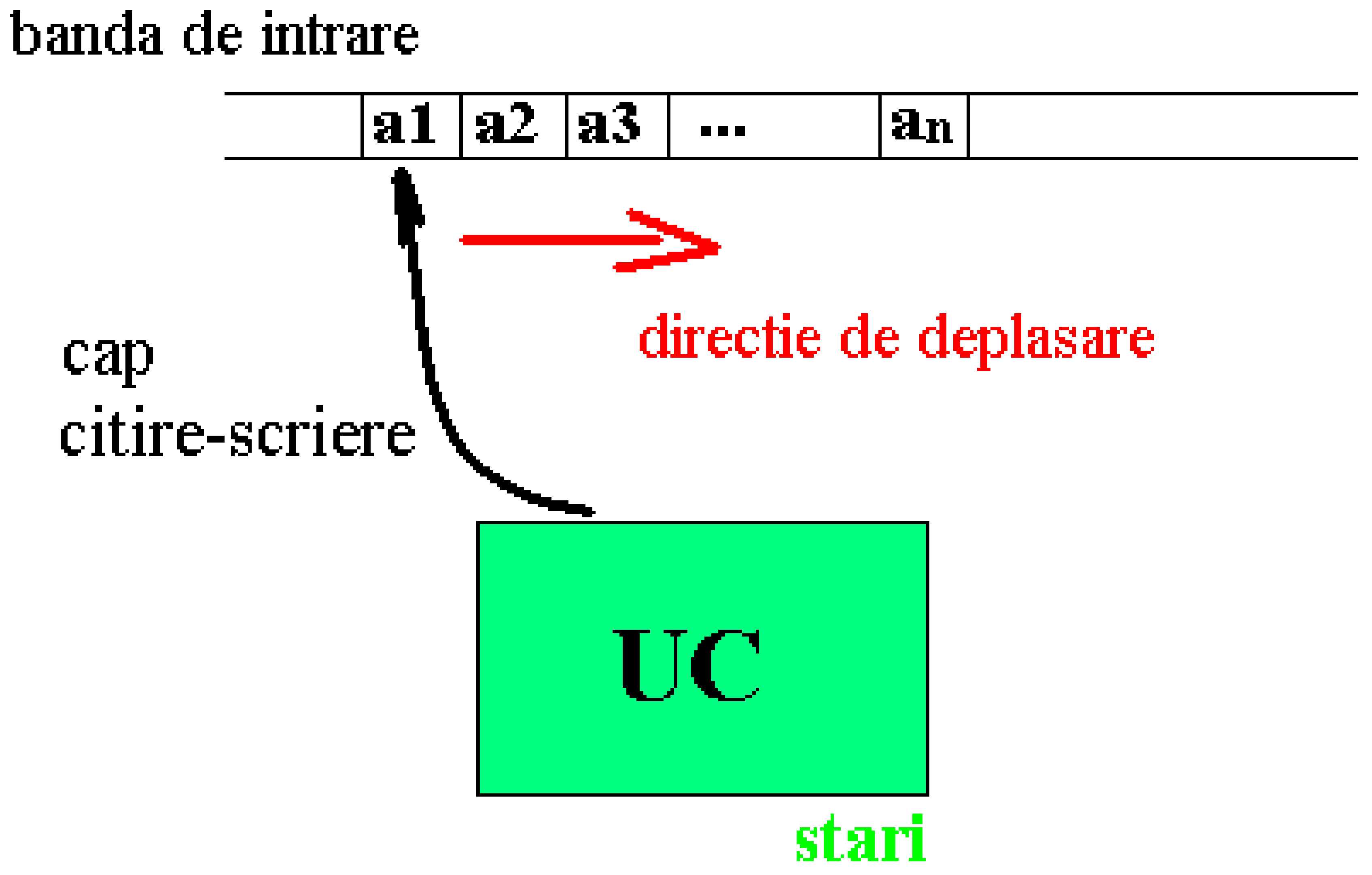
1. realizarea unor calcule in mom. compilarii
2. eliminarea operatiilor redundante si a expresiilor comune
3. eliminarea codului inaccesibil *(secvente moarte)*
4. scurtcircuitarea expresiilor logice
5. factorizarea invariantilor de cicluri

optimizarea ciclurilor

Automate

- Recapitulare
- Exemple
- Aplicatii
- Translatoare, masini Turing

Automat finit: model fizic



Limbaje regulare. Echivalente

- putere de exprimare

  AF: AFN ⇔ AFD

  AF ⇔ (m.regulare ⇔ expr.reg.)

  AF ⇔ gr.regulare

Proprietati de inchidere

Cursul 3, constructii

Teorema:

Daca

L_1, L_2 sunt limbaje regulare peste alfabetul Σ

atunci:

$\mathbf{L_1 \cup L_2, L_1 \cap L_2, L_1L_2, L_1^*}$, **complement**$\mathbf{(L_1)}$

sunt limbaje regulare peste alfabetul Σ

Proprietãti de închidere ale limbajelor independente de context

<u>Teoremã</u>.

Dacã L_1si L_2 sunt limbaje independente de context atunci:

$$L_1 \cup L_2 \;,\; L_1L_2 \;,\; L_1^*$$

sunt limbaje independente de context.

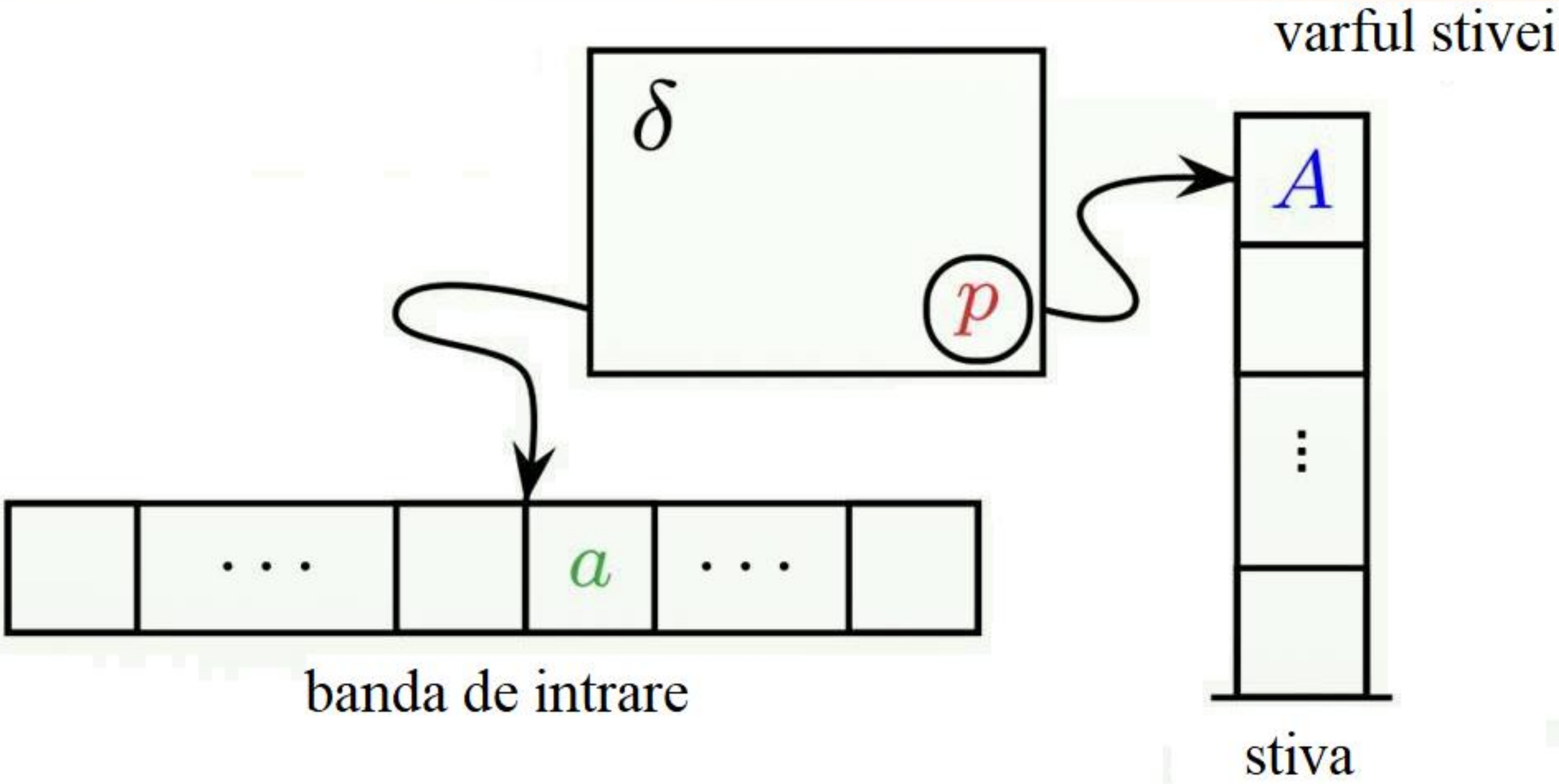
<u>Observatie</u>:

$L1 \cap L2$, compl(L1) :

nu sunt neaparat limbaje independente de context

constructie

Automat push-down (APD)



https://en.wikipedia.org/wiki/File:Pushdown-overview.svg

Automat push-down (APD)

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

- Q alfabetul stărilor;
- Σ alfabetul de intrare;
- Γ alfabetul memoriei stivă;
- $q_o \in Q$ stare iniţială;
- $Z_o \in \Gamma$ simbolul de start al memoriei stivă;
- $F \subseteq Q$ mulţimea stărilor finale;
- $\delta : Q x (\Sigma \cup \{\varepsilon\}) x \Gamma \rightarrow \mathcal{P}(Q x \Gamma^*)$ funcţia de tranziţie
  - are ca valori submulţimi finite din $Q x \Gamma^*$ (posibil multimea vida)

Determinism

$\mathbf{M = (Q, \Sigma, \Gamma, \delta, q_o, Z_o, F)}$ este ***determinist*** dacă:

$\forall \mathbf{Z} \in \Gamma, \forall \mathbf{q} \in \mathbf{Q}, \forall \mathbf{a} \in \Sigma$

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

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

In caz contrar, automatul nu este determinist

Multimea limbajelor acceptate de APD nedeterministe este strict mai larga decat multimea limbajelor acceptate de APD deterministe

exemplu

Limbaje independente de context. Teoreme de echivalenta

constructii

<u>Teoremă</u>.

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

<u>Teoremă</u>.

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

si reciproc.

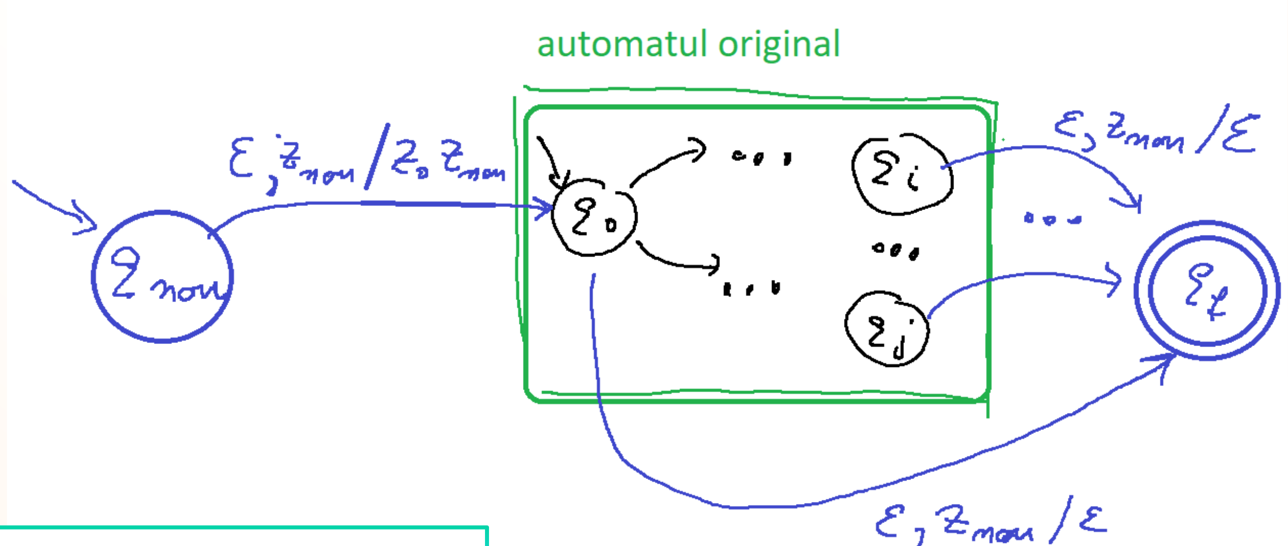
Transformarea acceptarii dupa criteriul stivei vide in acceptare dupa criteriul starii finale

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 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

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

- Q alfabetul stărilor;
- Σ alfabetul de intrare;
- D alfabetul de iesire;
- $q_o \in Q$ stare iniţială;
- $F \subseteq Q$ mulţimea stărilor finale;
- $\delta: Q x(\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}_0 (Q \text{ x } D^*)$

multimea partilor finite

Translator finit

Exemplu:

$M = (\quad Q, \qquad \Sigma, \quad D, \quad \delta, \quad q_o, \quad F \ )$

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

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

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

Translatarea definita de M:

$T(M)=\{(x,y) | \ x\in\Sigma^*\ , \ y\in D^*, \ (q_0,x,\varepsilon) \ |\!-^* \ (q,\varepsilon,y), \ q\in F)$

Translator push-down

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

- Q alfabetul stărilor;
- Σ alfabetul de intrare;
- Γ alfabetul memoriei stivă;
- D alfabetul de iesire;
- $q_o \in Q$ stare iniţială;
- $Z_o \in \Gamma$ simbolul de start al memoriei stivă;
- $F \subseteq Q$ mulţimea 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

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

$Q = \{q\}$

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

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

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

$q_o = q$

$Z_o = E$

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

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

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

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

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

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: (α,β,Π)
  - banda de intrare: α
  - stiva β (stiva de lucru)
  - banda de iesire Π => sirul regulilor de productie
- config. initiala: (w\$, S\$, ε)
- config. finala: (\$, \$, Π)
- tranzitii
  - push (**a**x\$, **A**β, Π) $\vdash$ (ax\$,$\alpha\beta$,Π**i**) dc.: M(A,a) = (α,i)
  - pop (**a**x\$, **a**β, Π) $\vdash$ (x\$, β, Π)
  - acc (\$, \$, Π) $\vdash$ acc
  - err in celelalte cazuri

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

[*Moldovan*]

Translatorul push-down modificat este:

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

Σ- alfabet de intrare şi este acelaşi cu alfabetul Σ din G;

D – alfabet de ieşire, $D = \{1, 2, \ldots, m\}$

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

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

q - starea internă (stare iniţială)

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

δ – funcţia de tranziţie modificată

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

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

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

dacă $(\exists) i : A \to x$ *şi dacă* $a \in \varphi(x)$, *sau*

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

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

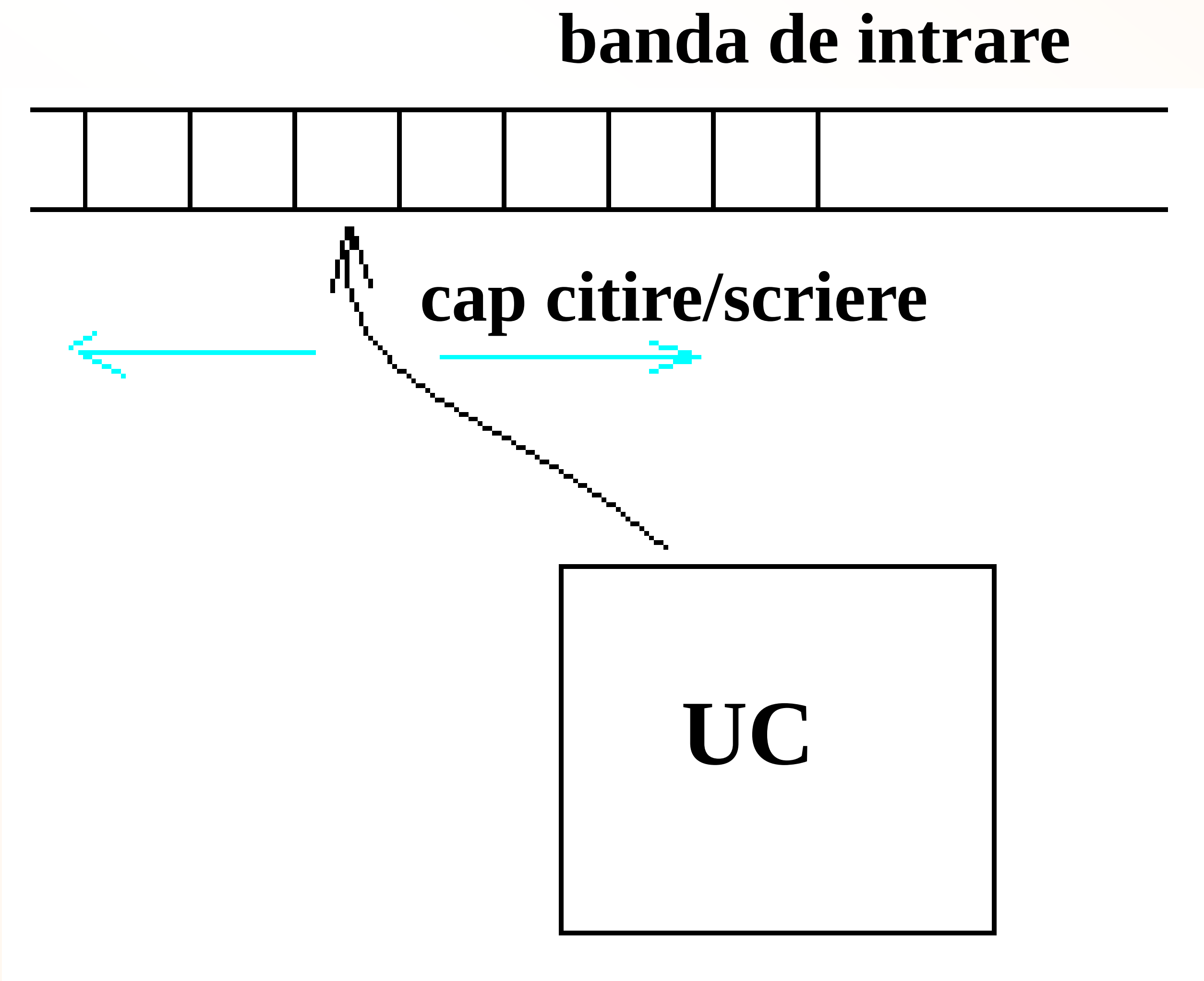
1 : semnifică înaintarea benzii cu o poziţie;

0 : semnifică staţionarea benzii.

Limbajul acceptat se defineste dupa criteriul stivei vide.

Masini Turing

- 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
- Γ – multimea simbolurilor benzii
- # - un simbol din Γ, numit simbolul *blanc*
- Σ – o submultime a lui $\Gamma - \{\#\}$
- δ – 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

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

- tranzitie

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

$$X_1X_2\ldots X_{i-1}qX_i\,X_{i+1}\ldots X_n \vdash X_1X_2 \ldots X_{i-2}\,p\,X_{i-1}YX_{i+1}\ldots X_n$$

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

$$X_1X_2\ldots X_{i-1}qX_i\,X_{i+1}\ldots X_n \vdash X_1X_2 \ldots X_{i-1}Y\,p\,X_{i+1}\ldots X_n$$

- limbaj acceptat

$$\{w \in \Sigma^* \mid q_0w \vdash \alpha_1q\alpha_2 \,,\, q \in F,\, \alpha_1\alpha_2 \in \Gamma^*\}$$

Exemplu: masina Turing

Functia de tranzitie

0011 ?

	Σ		$\Gamma - \Sigma$			
	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 mașină Turing cu k benzi

  nu este mai puternică decât o mașină Turing standard
- Mașină Turing deterministă (MTD)

  versus mașină Turing nedeterministă (MTND)

  Cele două sunt computațional echivalente,

  adică orice MTND se poate transforma într-o MTD

  (și invers).

Masini Turing

Teza lui Church

- Logicianul Alonzo Church a emis ipoteza că maşina Turing este modelul cel mai general de calcul care poate fi propus.
    - maşina 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 aceleaşi lucruri
- Acest enunţ nu este demonstrabil în sens matematic.

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

- **Timpul** de calcul pentru un şir dat la intrare: este numărul de mutări făcut de maşina Turing înainte de a intra în starea ``terminat'';
- **Spaţiul** consumat pentru un şir de intrare: este numărul de căsuţe de pe bandă pe care algoritmul le foloseşte în timpul execuţiei sale.

Gramatici

- Recapitulare
- Exemple
- Aplicatii
- Algoritmul CYK

Limbaje formale si tehnici de compilare

Limbaj formal

Limbaj regulare:

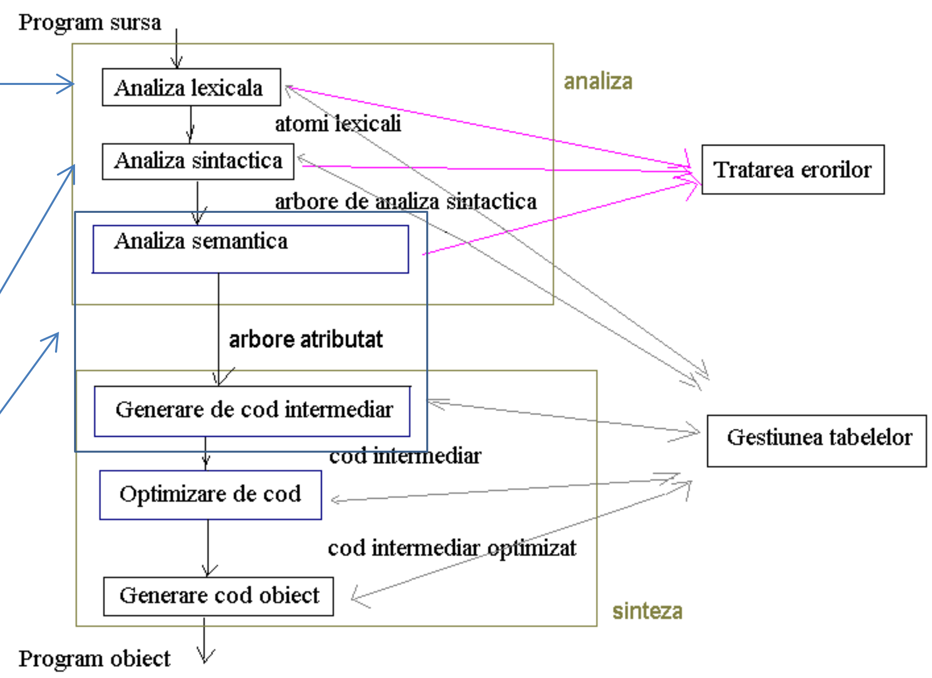
- gramatici regulare,
- automate finite,
- expresii regulare.

Limbaje independente de context:

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

Gramatici de atribute

Structura unui compilator



Gramatica

O gramatica este un cvadruplu **G = (N, Σ , P, S)**

- **N** este un alfabet de simboluri ***neterminale***
- Σ este un alfabet de simboluri ***terminale***
- **N $\cap \Sigma = \phi$**
- **P $\subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$**
  P multime finitã **(**multimea regulilor de productie**)**
- **S$\in$ N** (simbolul de start - simbolul initial)

Notatie:

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

(α se înlocuieste cu β)

- Gramatici de tip **0**

  nici o restrictie *(suplimentara)* referitoare la forma regulilor de productie
- Gramaticile de tip **1**

  dependente de context ⇔ ***gramatici monotone***

  (monotonic, non-contracting)
- Gramaticile de tip **2**

  gramatici independente de context
- Gramaticile de tip **3**

  gramatici regulare

Curs 2

Clasificarea Chomsky

Ierarhia Chomsky

~ 1959-1963

Fie

- $\mathcal{L}0$ - 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:

$$\mathcal{L}0 \supset \mathcal{L}1 \supset \mathcal{L}2 \supset \mathcal{L}3$$

- Gramaticile monotona
  - $\forall\ \alpha \to \beta \in$ P: $|\alpha| <= |\beta|$ $\alpha, \beta \in (N \cup \Sigma)^*$
  - caz special: **S→ ε** poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

- Gramatica dependenta de context

  reguli de productie sunt de forma:

  $\alpha\ A\ \beta \to \alpha\ \gamma\ \beta$ $A \in N$

  $\alpha, \beta, \gamma \in (N \cup \Sigma)^*$, $\gamma \neq \boldsymbol{\varepsilon}$

  - caz special: **S→ ε** poate $\in$. P In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Tipuri de gramatici

- 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→ ε** 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 $A \rightarrow \alpha$, $A \in N$, $\alpha \in (N \cup \Sigma)^*$

Tipuri de gramatici

Despre cum se definesc gramaticile regulare si liniare in diverse surse. (cateva aspecte)

Noi vom folosi definitiile de pe slide-urile anterioare.

Gramatica regulara la dreapta

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

- unele surse accepta si: **$A \rightarrow \varepsilon$**

- alte resurse nu accepta deloc **ε**

Gramatica regulara la stanga

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

Gramatica liniara la dreapta (gr.regulara la dreapta extinsa)

- $A \rightarrow a1\ a2\ \ldots\ an\quad B$
- $A \rightarrow b1\ b2\ \ldots\ bm$ ...

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

Curs 5

- Echivalente

  gram. ε -independenta

  eliminarea regulilor de productie de redenumire

  forma normala Greibach

  forma normala Chomsky (FNC)

Forma normala Chomsky

Gramatica in forma normala Chomsky

reg. prod. sunt de forma

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

caz special:

S→ ε poate $\in$ P

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. ε –independenta, fara redenumiri

Folosim transformari de forma:

$$A \to X_1X_2 \ldots X_n \quad : \quad \begin{array}{l} A \to X_1Z_1 \\ Z_1 \to X_2Z_2 \\ Z_{n-2} \to X_{n-1}X_n \end{array}$$

Forma normala Greibach

Gramatica in forma normala Greibach

- reg. prod. sunt de forma
  - $A \rightarrow a\ \alpha$ unde $\alpha \in (N \cup \Sigma)^*$ si $a \in \Sigma$

- caz special:

  S→ ε poate $\in$ P

  In acest caz S nu apare în membrul drept al nici unei reguli de productie.

Teorema:

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

Constructie:

Fie: gram. ε –independenta, fara redenumiri

1. (similar cu ce facem pt. eliminare recursivitate stanga)
   - impunem o ordine asupra neterminalelor: $N = \{A_1, A_2, \ldots, A_n\}$
     si apoi modific r.p. a.i. sa nu existe $A_i \rightarrow A_j\alpha$ cu $i<j$

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

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

.

Forma normala Greibach

Analizorul CYK (Coucke-Younger-Kasami)

- Se da: G – gramatica independenta de context in Forma Normala Chomsky (FNC)

  $w \in \Sigma^*$
- Se cere sa se verifice daca $w \in L(G)$

PP. $w = a_1a_2 \ldots a_n$

Analizorul CYK (Coucke-Younger-Kasami)

$t_{ij}=\{A \in N \mid A \Rightarrow^* a_i \ldots a_{i+j-1}\}$

Constructia tabelului $T = (t_{ij})$

- $t_{i1} = \{A \in N \mid A \rightarrow a_i \}$
- $t_{ij} = \cup_{k=1,j-1}\{A \in N \mid A \rightarrow BC,\ B \in t_{ik},\ C \in t_{i+k,j-k}\}$

$S \in t_{1n}$?

Analizorul CYK (Coucke-Younger-Kasami)

Fie gramatica cu r.p.:

$S \rightarrow AA \mid AS \mid b$

$A \rightarrow SA \mid AS \mid a$

$w = abaab$

? $w \in L(G)$

Analizorul CYK (Coucke-Younger-Kasami)

Avem T

$S \in T_{1,n}$

? sirul regulilor de productie aplicate ?

Subalg. recursiv;

- in specif.: pun in evidenta numai param. care se modifica
- foloseste G,T, $w=a_1 \ldots a_n$ (date) si obtine r.p. (rezultate)

Subalg. genprod(i,j,A)

- genereaza r.p. ce genereaza substringul $a_i..a_{i+j-1}$ din w
- pornind de la neterminalul A din $T_{i,j}$

Analizorul CYK (Coucke-Younger-Kasami)

Subalg. genprod(i,j,A)

Daca j=1 atunci

*cautam $A \rightarrow a_j \in P$

*rez.partial: r.p.: A→aj

altfel

*cautam cel mai mic k a.i. $A \rightarrow BC \in P$

$B \in t_{ik}$, $C \in t_{i+k,j-k}$

*rez.partial : r.p. A→BC

genprod(i,k,B)

genprod(i+k,j-k,C)

sf.daca

Sf.subalg.genprod