

Tehnici de Compilare

Bibliografie: Aho, Sethi, Lam, Ullman - Compilers: Principles, Techniques and Tools, 2006

Lab.: 50% (cu proiecte) - 2 proiecte

Seminar - se fac exerciții de examen, ieșitul la tablă crește nota

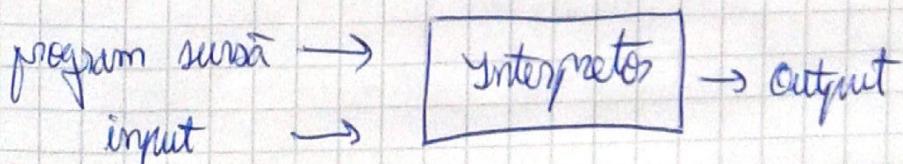
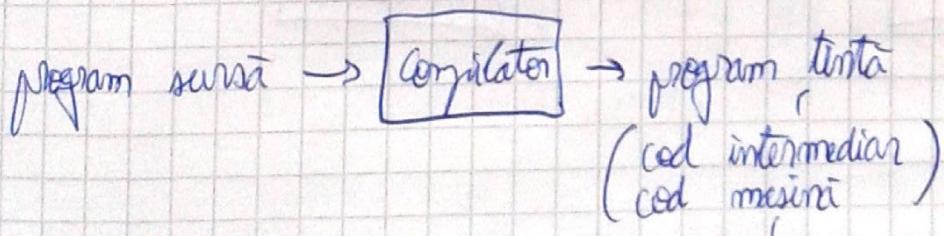
Examen cu cărți pe masă, numai exerciții: 50%.

19.02.2015

Curs 1

Procesare de limbaje

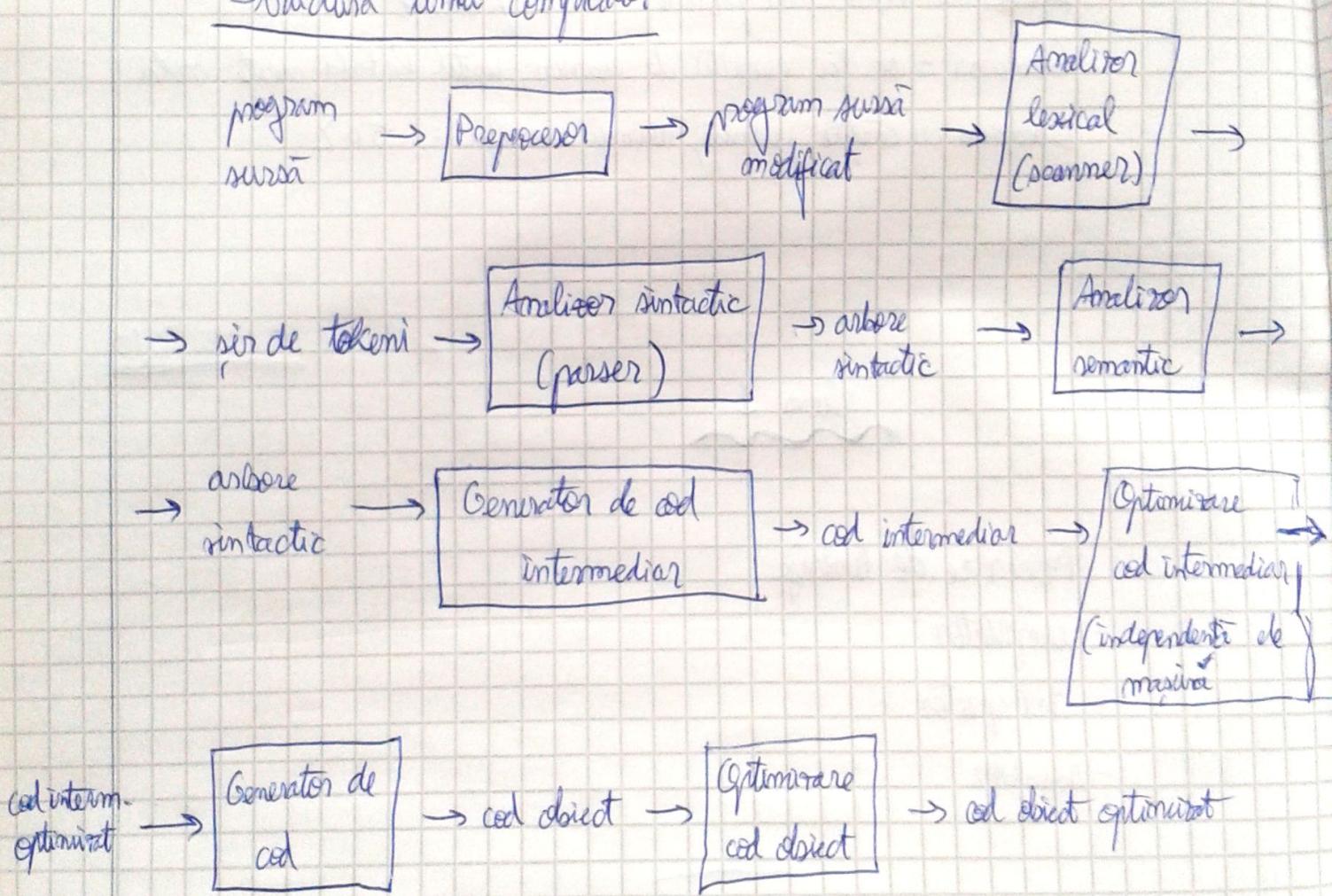
- Compilator
- Interpretor
- Asembler



program în limbaj de arambălare → Asamblator → cod mașină

Limbiile de ~~programare~~ generația 1 (cod mașină), 2 (assembly), 3 (lb. de
mig. simplă: FORTRAN, C++, C, Java), 4 (limbișe speciale: MySQL, SQL),
5 (PROLOG)

Structura unui compilator



Exemplu: calcul rezultat instrucție:

$$pos = init + nata * 60$$

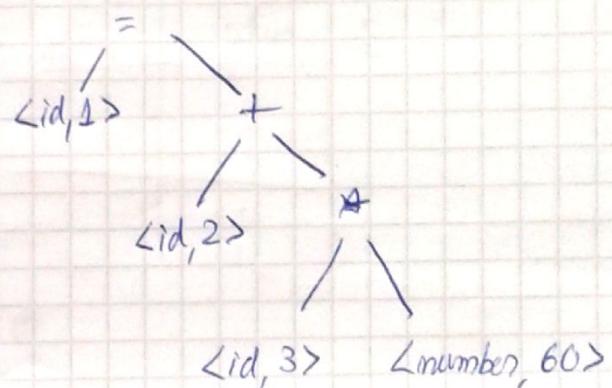
↓
Analizor lexical

$$\langle \text{id}, 1 \rangle \Rightarrow \langle \text{id}, 2 \rangle \langle + \rangle \langle \text{id}, 3 \rangle \langle * \rangle \langle \text{number}, 60 \rangle$$

↓

introduc în
tabelă de simboluri

Analizor sintactic → Arbore



→ Analizor semantic → int to float → $\langle \text{number}, 60 \rangle$

↓

Generator de cod
intermediar

$$t_1 = \text{int to float}(60)$$

$$t_2 = \text{id}_3 * t_1$$

$$t_3 = \text{id}_2 + t_2$$

$$\text{id}_1 = t_3$$

↓
Optimizator
de cod

$$\begin{cases} t_1 = \text{id}_3 * 60.0 \\ \text{id}_1 = \text{id}_2 + t_1 \end{cases}$$

↓
Generator de cod

{
LDF R₂, id₃
MULF R₂, R₂, #600
LDI R₁, id₂
ADAF R₁, R₁, R₂
STF id₁, R₁

Tabelă de simboluri

1	nos	- -
2	init	- -
3	nata	- -
	:	
	:	

Procesul de compilare poate fi împărțit în :

- Analiză (front end) : analiză lexicală, sintactică, semantică
- Sinteză (back end) : generația codului

Analiză lexicală

- expresii regulate
- automate și translatoare finite

Analiză sintactică

- gramatici independente de context
- automate și translatoare stivă

Analiză semantică: gramatica celor atribuite

Instrumente utile în dezv. unui compilator:

a) Generatoare de analizare lexicală

lex, flex, jflex

b) Generatoare de analiză sintactică

Yacc - Yet Another Compiler Compiler

bison

c) Generatoare de generator de cod

d) Toolkit pentru construcția unui compilator

• Limbi formale •

Σ alfabet (multime nevidată și finită)

Σ^* - toate cuvintele cu litere din Σ , incl. cuvantul vid

$$\Sigma = \{a, b\} \quad \Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, \dots\}$$

cuv. de lungime 0

$$L \subseteq \Sigma^*, \quad L^* = \{\lambda\} \cup L \cup L^2 \cup \dots \cup L^i \cup \dots$$

L - limbaj

$$i \geq 2 : L^i = \{w \in \Sigma^* \mid \exists w_1, w_2, \dots, w_i \in L, w = w_1 \dots w_i\}$$

↓ i cuvinte din L, concatenare

$$w \in \Sigma^*, \quad w = a_1 a_2 \dots a_m \quad a_i \in \Sigma, \quad 1 \leq i \leq m$$

|w| = m (nr. de litere din cuvantul w)

|w|_a = nr. de apariții ale lui a în w, $a \in \Sigma$

Gramatica Chomsky

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

N : alfabetul neterminabilor

Σ : — " — terminalibor

S : simbolul de start

$$P \subseteq (\Sigma \cup N)^* N (\Sigma \cup N)^*$$

multime finita

$$(x, y) \in P \quad x \rightarrow y \quad (\text{productie sau regula din } P)$$

$$x \xrightarrow{*} y \quad \text{daca} \quad x = y \quad (\text{derivare in 0 pasi})$$

sau $\exists x_1, \dots, x_m \in (N \cup \Sigma)^*$ a.i.

$$x_1 = x, x_m = y, x_i \Rightarrow_G x_{i+1}$$

$$1 \leq i \leq m-1, m \geq 2$$

$$x, y \in (N \cup \Sigma)^*$$

(derivare in m pasi)

L(G) = L(G'):

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

- Gramaticile se clasifică în funcție de productii (de forma pe care o ia productia)

Fie $G = (N, \Sigma, S, P)$ gram. Chomsky:

a) G este de tip 0 (nu există nicio restricție asupra produselor din G)

b) G monotonă, ($x \rightarrow y \in P, |x| \leq |y|$)

c) G dependentă de context: $x \xrightarrow{*} y \rightarrow x z y$
 $x, y, z \in (N \cup \Sigma)^*, z \neq \lambda, A \in N$

d) G semidependentă de context la stânga sau la dr.

$$xA \rightarrow xz, xz \in (N \cup \Sigma)^*, z \neq \lambda, A \in N$$

e) Gramatica de context

$$A \rightarrow x, A \in N, x \in (N \cup \Sigma)^*$$

f) G liniară: $A \rightarrow xBy$ sau $A \rightarrow z, x, y \in \Sigma^*, A, B \in N$

g) G semiliniară la stânga (la dreapta)

$$A \rightarrow Bx, t \rightarrow y, x, y \in \Sigma^*, A, B \in N$$

h) gram. regulate $A \rightarrow aB, A \rightarrow b, a, b \in \Sigma, A, B \in N$

e, f, g, h nu interesează mai mult

$$\mathcal{L}_3 \neq \mathcal{L}_{\text{lin}} \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$

$$\mathcal{L}_1 = \{a^n \mid n \geq 1\} \in \mathcal{L}_3$$

$$\mathcal{L}_2 = \{a^m b^n \mid m, n \geq 1\} \in \mathcal{L}_{\text{lin}} - \mathcal{L}_3$$

$$\mathcal{L}_3 = \{a^n b^m a^p b^p \mid n, p \geq 1\} \in \mathcal{L}_2 - \mathcal{L}_{\text{lin}}$$

$$\mathcal{L}_4 = \dots$$

Automat finit determinist

~~Automat finit~~, ~~determinist~~

$$A = (Q, \Sigma, \delta, q_0, F) \text{ AFD}$$

Q: mult. stări

Σ : alfabet automat

$\delta: Q \times \Sigma \rightarrow Q$

$q_0 \in Q$: stare initială

$F \subseteq Q$: stări finale

$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$

$$\hat{\delta}(q_1, \lambda) = q_2$$

a.i

Σ x_{i+1}

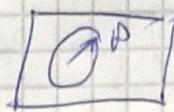
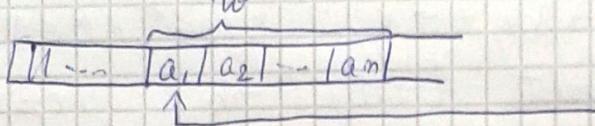
oia potrivită

or din G)

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a) \quad a \in \Sigma$$

Descriere instantanee a lui A

(p, w) $p \in Q$ starea curentă, $w \in \Sigma^*$



- Misare sau schimbarea de config. a lui A

$$(p, aw) \xrightarrow{a} (q, w) \text{ dacă } q : \delta(p, a) \quad p, q \in Q, a \in \Sigma$$

\vdash^* incluzarea self și trans. a relației \vdash

$$L(A) = \{ w \in \Sigma^* \mid (q_0, w) \vdash^* (q, \lambda), q \in F \}$$

Automat redeterminist: multe stări ...

19.02.2015

Laborator 1

2 proiecte \Rightarrow nota

proiectele se notează de la 4 la 10

prezență nu contează

termen: înainte de următoarele optimizări

verificarea lui Drăgușici!

★ Proiect 1: Analizor lexical pt. un limbaj visual (C, C++...) \rightarrow căută pe net specificația limbajului

Adică înțelegerea în tokeni ce tip și valoare cu un AFD

Trebuie reprodus algoritmul

Limbaj complet! Să recunoască tot tokenul

(automat finit determinist)

Programare POO conform teoriei - metoda Scanner cu obiectul token ...

1 (Scanner) analiză lexicală
Separarea textului în ~~țările~~ atomi lexicali ("cuvinte")

atom lexical → tip (la fel cum se stabilește "partea de vorbire" la
(token) ca
lb. nr.)

valoare: simbol de caracter

Un token poate contine spațiu etc.

tip: literali întregi / flotanti / char, bloc de spatii abe, identificator,
de tokeni operator, comentariu, separator
(clase de tokeni)

2 (Parser) Analiză ca "parte de proprietate" ⇒ arbore de derivare (Parserul se
mută peste lista tipurilor de tokeni, nu se
mută la ~~aceea~~ de exemplu, dacă o variabilă a
fost declarată sau nu)

- Gramatică independentă de context
- nu se mută la ce a fost scris în
rest

3 (Analiză semantică
(constrâneri))

- Gramatică independentă de context
(gramatică atributată)
- Tabelă de simboluri

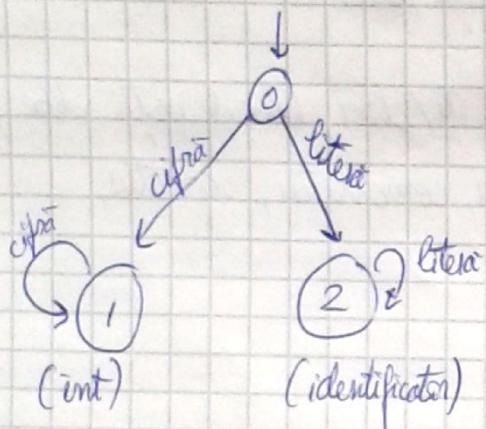
⇒ arbore cu decorări (conține
toate informațiile codului)

4 Generarea de cod

5 Optimizare cod

În practică 1-5 se fac în paralel într-un compușor.

Proiect 1 - emit token atunci când automatul se blochează într-o stare finală sau se termină codul cursă



123a → va se deca în
(123, int)

a
(a, idf)

nu în (1, int), (2, int), (3, int),
(a, idf)

Să ignoreți blank spaceuri fără să le lițiere
Tablă de stânguri

Scanner()

main()

{ getToken();

}

Curs 2

- Def. 1: Expresie regulată peste Σ

ϕ este expresie regulată care descrie limbajul ϕ
 λ
 $a \in \Sigma$
 $\{\lambda\}$
 $\{a\}$

Fie $n \geq 1$ expr. regulate peste Σ care descriu lb. $L(n), L(\Delta)$

Astăzi :

$n | \Delta$ este expr. reg. care descrie lb. $L(\Delta) \cup L(n)$
 $n \cdot \Delta$
 n^*
 (n)
 ~~$L(n) \cup L(\Delta)$~~
 $(L(n))^*$

- Def. 2: Expr. Reg \rightarrow Expr. Reg. | Termen

→ Termen

Termen \rightarrow Termen · Factor

→ Factor

Factor \rightarrow Factor *

→ Rest

Rest $\rightarrow a \quad a \in \Sigma$

Descriere lexicală

- Def.: Neamim descrierea lexicală peste Σ o expr. reg. de forma
 $e = (e_1 | e_2 | \dots | e_m)^*$, unde e_i este expr. reg. peste Σ care descrie o categorie de atomi lexicali.

Fie $w \in L(e)$. Vom imita interpretarea lui w ca un sir de forma $(w_1, m_1), \dots, (w_t, m_t)$, unde $1 \leq m_1, \dots, m_t \leq n$, $w_1, w_2, \dots, w_t = w$, $w_i \in L(\text{elem}_i)$, $i = 1, \dots, t$.

Suntem cai este ambiguă da. Fie $w \in L(e)$ care are 2 interpretări distincte.

Exemplu: $P_{\text{source}} = (\text{id} | \text{int} | \text{comment} | \text{slash} | \text{spaces} | \text{semicolon} | \text{equal})^*$

$\text{id} = \text{letter} (\text{letter} | \text{digit})^*$

$\text{int} = \text{digit digit}^*$

$\text{comment} = "/*" / ("*" ("not slash")*)^* "*/^*$

$\text{slash} = "/"$

$\text{spaces} = (" " \cup "\n")^*$

$\text{semicolon} = ";"$

$\text{letter} = "a" | \dots | "z"$

$\text{digit} = "0" | \dots | "9"$

$L(\text{not slash}) = T - \{"/"\}$

$L(\text{not slash}) = T - \{"/\", "\{\", "\}\}$

T - alfabetul caracterelor permise

abc

("a", id) ("b", id) ("c", id)
 ("ab", id) ("c", id)
 ("a", id) ("bc", id)
 ("abc", id)

Def.: Suntem cai interpretarea $(w_1, m_1), \dots, (w_t, m_t)$ a lui $w \in L(e)$ este orientată dreapta dacă pt. orice i , $1 \leq i \leq t$, w_i este cel

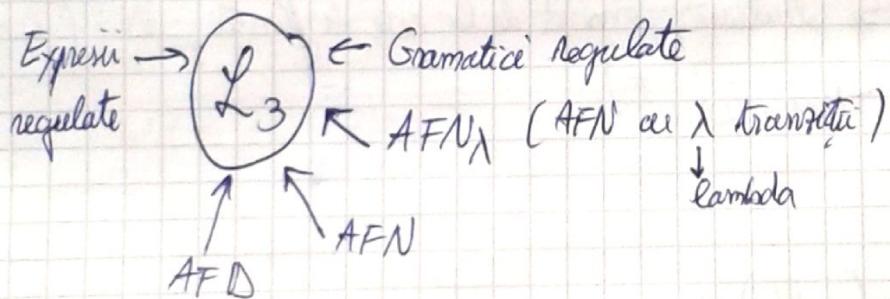
mai lung decât den $L(e, // \dots / em)$ și PREFIX(w_i ... w_j)

Def.: Suntem că descr. lexicală e este bine formată dacă și $w \in L(e)$ are o unică ~~aceeași~~ interpretare orientată dreapta.

ex.: $e = (a / ab / bc)$

$w = abc$ ambiguitate între 1 și 2
(“a”, 1) (‘bc’, 3)

Ameliorare lexicală: program care recunoaște lă. descris de un sir de exp. regulate



Transformarea unei expresii regulate într-un AFD (automat finit deterministic):

1) Cu ajutorul unui generator de ameliorare lexicală lex, flex, iflex

termi laboator

2) $Exp\ Reg \xrightarrow{2.1} AFN_\lambda \xrightarrow{} AFD$

AFN → AFD

22) $Exp\ Reg \rightarrow AFD$

2.2) Exp Reg \rightarrow AFD

Algorithm :

Input: Exp Reg proto \mathcal{E}, r

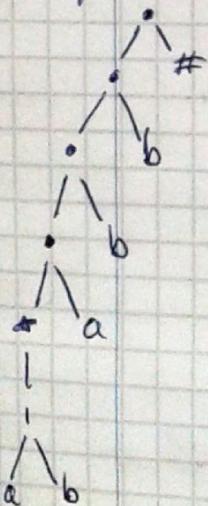
Output: AF(D), D, a.i. L(D) = L(n)

- 1) Se extinde expresia r la $(r)\#$, unde $\#$ este un simbol nou, $\# \notin \Sigma$
 - 2) Se construiește arborele sintactic T , asociat expresiei extinse $(r)\#$
 - 3) Se numerotează frunzele lui T de la st. la dr. cu numere din \mathbb{Z} începând cu 1 (poziții)
 - 4) Se calculează, privind-o pe răcirea în adâncime a lui T , funcțiile: $\text{nullable}(n)$, $\text{firstpos}(n)$, $\text{lastpos}(n)$, ~~$\text{followpos}(n)$~~ , $\text{followpos}(i)$, unde n este nod al lui T și poziție
 - 5) Se calculează, pornind de la cele 4 funcții, $D = (D\text{ states}, \Sigma, \text{tran}, q_0, F)$

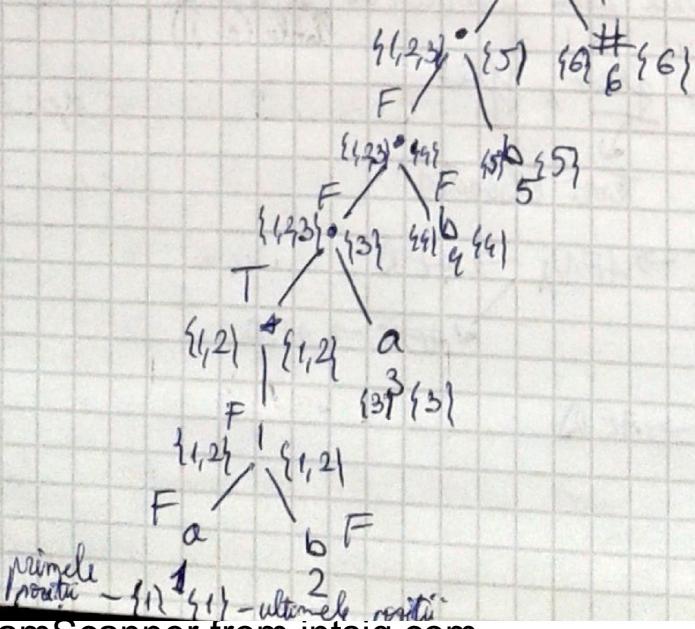
$$\text{ext. : } r = (a/b) * abt$$

expr. reg.: $((a/b) * abb) \#$

simply :



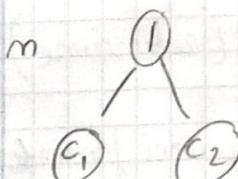
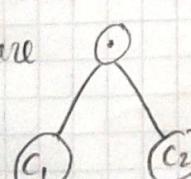
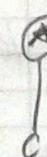
arborele sintactic : $\{1, 2, 3\}^*$ \{6\}



follow pos	
1	1, 2, 3
2	1, 2, 3
3	4
4	5
5	6
6	—

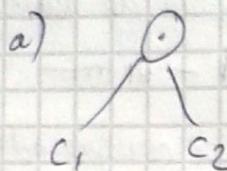
- $\text{mutable}(m)$: funcție booleană care este true dacă lb. descriș de subarborele de rădăcină m conține simbolul vid, λ .
- $\text{firstpos}(m)$: mult. pozițiilor care corespund primele litere ale porunilor din lb. descriș de subarborele de rădăcină m
- $\text{lastpos}(m)$: — " — ultimele " —
- $\text{followpos}(i)$: mult. pozițiilor ce corespund unei litere de la $i+1$ și pînă la finalul cuvântului care are înaintea o literă (din Σ^*) ce corespunde pozitiei i

ex.: $w \in L(r)$, $w \in \Sigma^*$, $w = a_1 a_2 \dots a_i \dots a_{i+1} \dots a_p$
 pozitii $\uparrow \quad \uparrow \quad t$
 $t = \text{followpos}(i)$ (corect?)

Nodul m	$\text{mutable}(m)$	$\text{firstpos}(m)$	$\text{lastpos}(m)$
m nod etichetat cu λ	true	\emptyset	\emptyset
m nod etichetat cu a și pozitia i , $a \in \Sigma$	false	{ i }	{ i }
m 	$\text{mutable}(c_1)$ or $\text{mutable}(c_2)$	$\text{firstpos}(c_1) \cup$ $\text{firstpos}(c_2)$	$\text{lastpos}(c_1) \cup$ $\text{lastpos}(c_2)$
m nod concatenare 	— " — and — " —	$\text{if } (\text{mutable}(c_1)) \text{ if } (\text{mutable}(c_2))$ $\text{firstpos}(c_1) \cup \text{firstpos}(c_2)$ $\text{else } \text{lastpos}(c_1)$	$\text{lastpos}(c_1) \cup \text{lastpos}(c_2)$ $\text{else } \text{lastpos}(c_2)$
m nod iteratie 	true	$\text{firstpos}(c)$	$\text{lastpos}(c)$

follower_{pos}(i), i pozitie - cum se calculeaza:

- Se parcurge in adancime T
- Se considera doar nodurile • i^*



a) At. fiecare $i \in \text{lastpos}(c_1)$, orice pozitie $j \in \text{firstpos}(c_2)$ este in $\text{follower}_{pos}(i)$

b)
At. fiecare $i \in \text{lastpos}(c)$, fiecare $j \in \text{firstpos}(c)$ este in $\text{follower}_{pos}(i)$

Starea initială a automatului e $q_0 \leftarrow \text{firstpos}(m_0)$, m_0 radacina a arborelui T.

$\Delta_{\text{states}} \leftarrow \{ q_0 = \text{firstpos}(m_0) \}$, q_0 stare marcată

while (există $M \in \Delta_{\text{states}}$ stare marcată)

{ marchează M ;

for (fiecare $a \in \Sigma$)

{ $V \leftarrow \bigcup_{p \text{ pozitie}} \text{follower}_{pos}(p)$ (recenzie)

p pozitie în M ce corespunde unei frunze etichetată

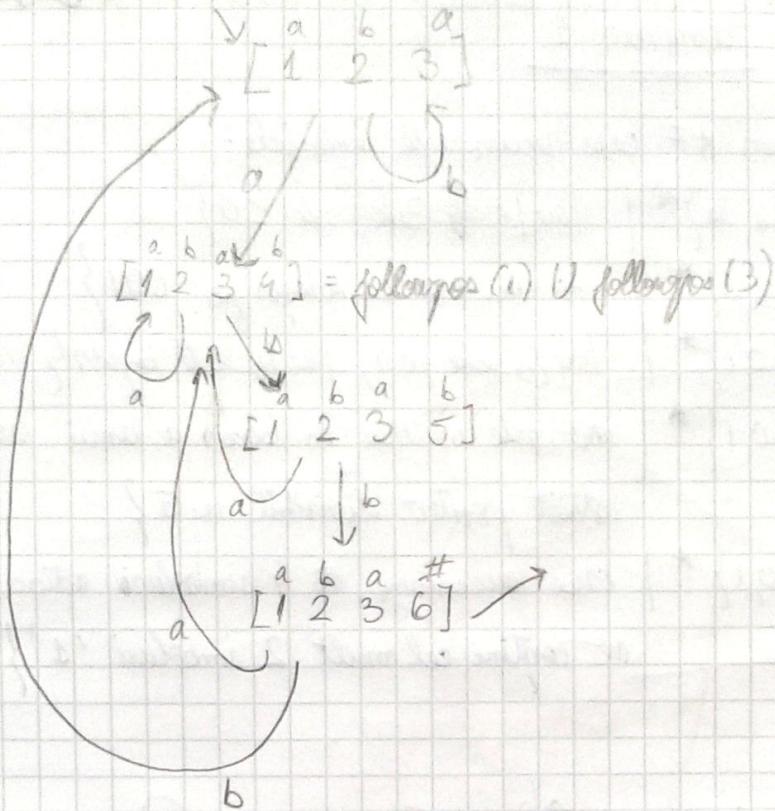
if ($V \notin \Delta_{\text{states}}$) adaugă V la Δ_{states} ca stare marcată;

$\Delta_{\text{tran}}[M, a] \leftarrow V$

}

$F = \{ M \in \Delta_{\text{states}} \mid M \text{ contine pozitia marcatului } \#\}$

cu $\#$ reprezentând stările finale ale automatului



Obs.: AFD D obtinut ca alg. de mai sus nu este nesupravant minimal.

2.1.) Algoritm ExpReg \rightarrow AFM \rightarrow AFD.

$$A \text{ AFM } A = \{Q, \Sigma, \delta, q_0, F\}$$

$$D \text{ AFD } D = \{Q', \Sigma, \delta', q_0', F'\}$$

$$L(D) = L(A) \quad q_0' = \langle q_0 \rangle \text{ (submărginime ale lui } q_0)$$

$$Q' \subseteq 2^Q$$

Seminarul 1

Ex.: Să se scrie un AF care recunoaște limbajele:

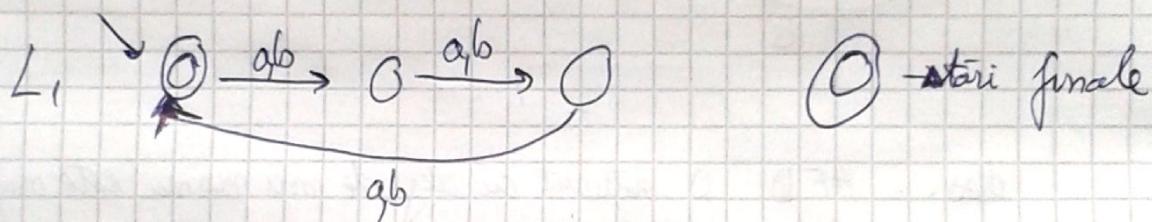
$$L_1 = \{w \in \{a, b\}^* \mid |w| = 3k, k \geq 0\}$$

$$L_2 = \{w \in \{0, 1\}^* \mid w \text{ nu are ca sufix pe } 001\}$$

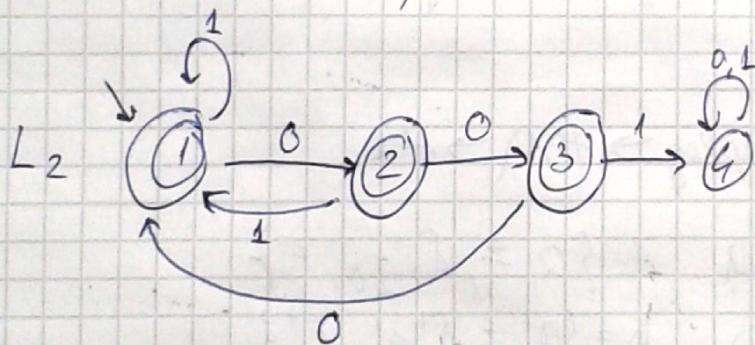
$$L_3 = \{w \in \{0, 1\}^* \mid |w|_0 \text{ este par}, |w|_1 \text{ este impar}\}$$

$$L_4 = \{w \in \{0, 1\}^* \mid w \text{ este scrierea în binar a unui nr. decimal strict pozitiv divizibil cu } 5\}$$

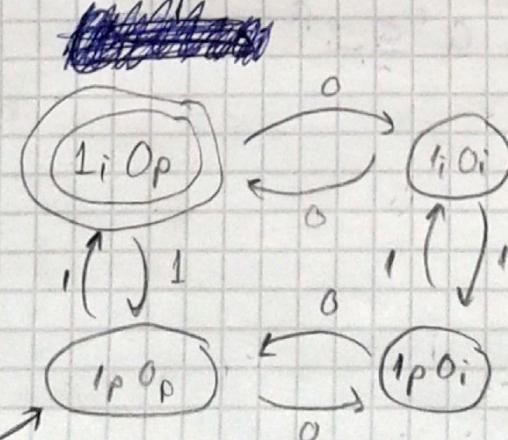
$$L_5 = \{w \in \{0, 1\}^* \mid \text{orice succesiune de 4 simboluri adiacente din } w \text{ conține cel mult 2 simboluri '1'}\}$$



○ → stări finale



L_3



1iOp - 1pOo 0 impar

$$L_4: \frac{x}{5} \Rightarrow x = \overline{a_1 a_2 \dots 0} \text{ unde } x = a_1 a_2 \dots 5$$

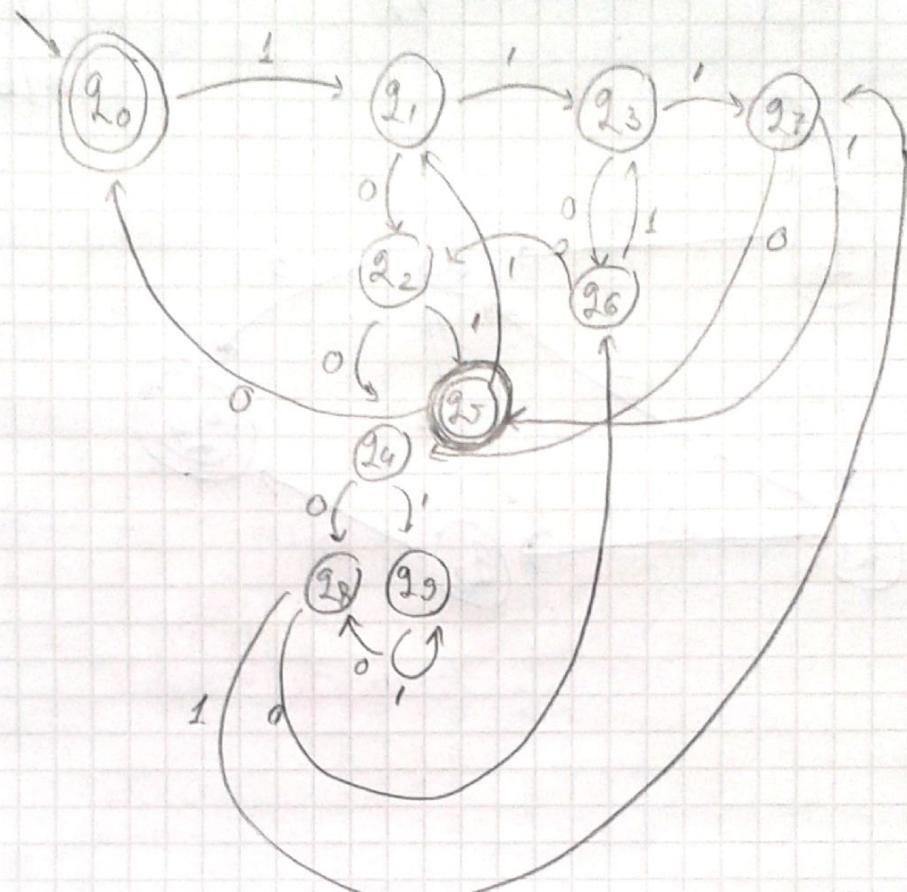
$$x_{\text{binary}} = \overline{1 a_1 a_2 \dots a_n}$$

numerical

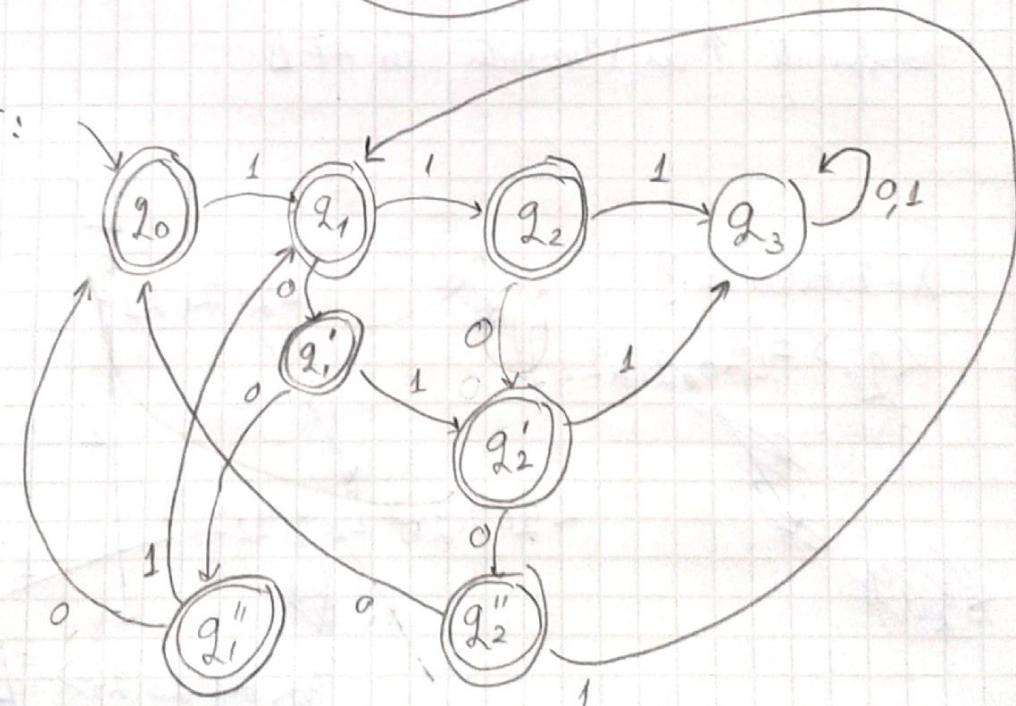
starii =

ultima cifra

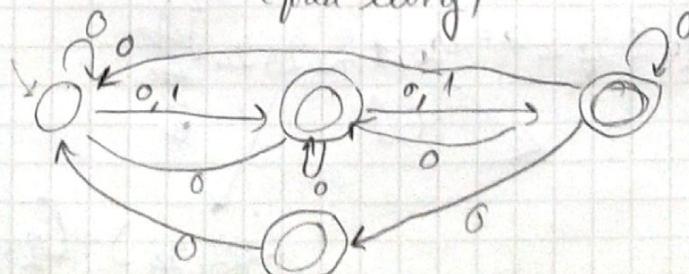
in decimal



L5:



(prelung)

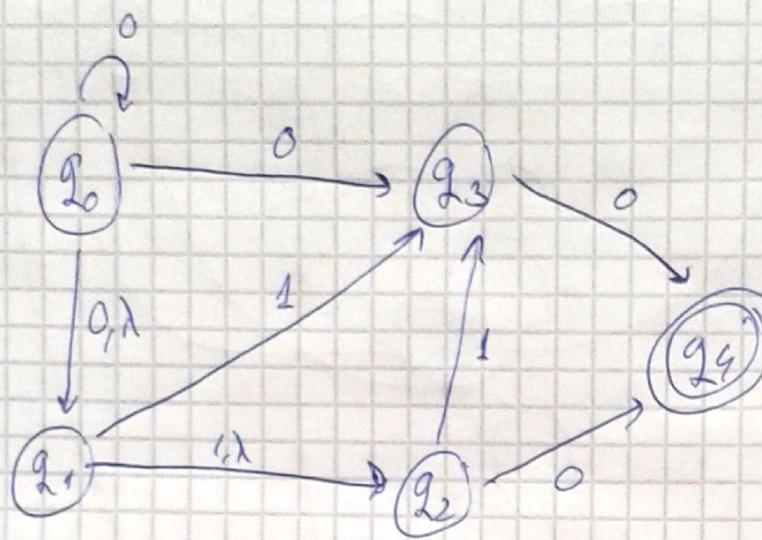


ex.: M_2 și M_3 exprimă regulate

$$M_3 : ((a/b)(a/b)(a/b))^*$$

/ înseamnă „scenă”

$$a/b = a \text{ scenă } b$$



Transformați 1 cu λ-tranzitii în AFD

λ -încideri = ?

$$\langle q_0 \rangle = [q_0, q_1, q_2]$$

~~q0, q1, q2~~

$$(\langle q_0 \rangle = \lambda\text{-încidere})$$

$$\langle q_0, q_1, q_2, q_3 \rangle = [q_0, q_1, q_2, q_3]$$

$$\langle q_0, q_1, q_2, q_3 \rangle = [q_0, q_1, q_2, q_3]$$

$$[q_0, q_1, q_2]$$

$$[q_2, q_3, q_4]$$

$$[q_4]$$

10

$$\langle q_3, q_4 \rangle = [q_3, q_4]$$