

# TEHNICI DE COMPILARE

CURS 1

Gianina Georgescu

# SCOPUL CURSULUI

- Să învățați despre structura unui compilator
- Să aprofundați unele cunoștințe de limbaje formale care constituie baza realizării unui compilator
- Să dobândeți abilități care să vă permită să proiectați un compilator
- Să folosiți cunoștințele dobândite în realizarea unui compilator sau a unei mari părți din acesta

Tehnicile de bază care vor fi învățate în timpul cursului pot fi utilizate în:

- construcția compilatoarelor
- arhitectura calculatoarelor
- teoria limbajelor
- algoritmică
- software engineering
- etc.

# STRUCTURA SĂPTĂMÂNALĂ A CURSULUI

- Nr. ore/săptămână: 4 (curs = 2; laborator = 2 la 2 săptămâni; seminar = 2 la 2 săptămâni)
- Semestrul: 6 / anul III de studiu: 10 cursuri
- Forma de examinare: examen
- Credite: 5
- **EVALUARE:** 50% laborator, 50% examen

# STRUCTURA CURSULUI

- 1. Motivație, scurt istoric. Structura unui compilator. Exemple.
- 2. Analiza lexicală. Descrieri lexicale cu ajutorul expresiilor regulate. Implementarea analizorului lexical.
- 3. Metode generale de analiză sintactică. Analiza sintactică top down, algoritmul top-down general.
- 4. Gramatici si limbaje LL(k). Gramatici si limbaje LL(k) tari. Mulțimile FIRST, FOLLOW. Recursivitatea la stânga. Factorizarea stângă.
- 5. Proprietăți ale gramaticilor LL(k). Echivalența dintre gramaticile LL(1) tari și LL(1). Parserul recursiv descendent – algoritm.

- 6. Parser predictiv pentru gramatici LL(k) tari – algoritm. Demonstrarea validitatii algoritmului pentru gramatici LL(k) tari.
- 7. Algoritmul Earley. Analiza sintactică bottom up - metoda generală. Gramatici și limbaje LR(k), definiții, proprietăți.
- 8. Parser de tip deplasare-reducere pentru gramatici LR(1) – algoritm. Demonstrarea validității algoritmului pentru gramatici LR(1).
- 9. Parser SLR(1) – algoritm. Parser LALR(1) – algoritm. Revenirea din eroare în parsere de tip LR.
- 10. Analiza semantică. Gramatici cu attribute, attribute sintetizate și attribute moștenite. Exemple.

# BIBLIOGRAFIE

- A. Aho, M. Lam, R. Sethi, J. Ullman, *Compilers: Principles, Techniques & Tools*, 2007, Addison Wesley
- A. Aaby, *Compiler Construction using Flex and Bison*, 2004,
- Bruno Preiss, *Lexical Analysis and Parsing using C++*, 2004

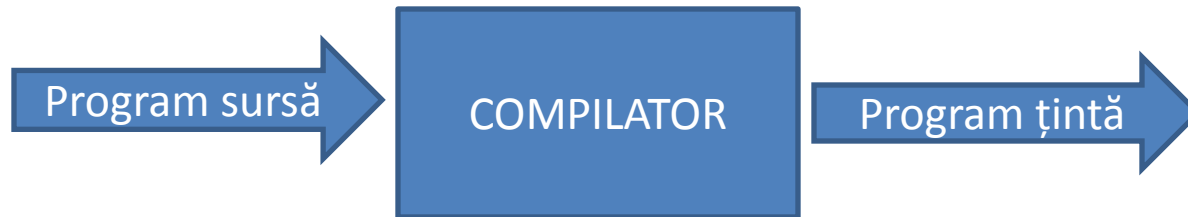
# LIMBAJE PENTRU CALCULATOARE

- Limbaj cod-mașină (nivel 0)
  - adresele, numerele, instrucțiunile: scrise în binar
  - foarte greu de folosit
- Limbaje de asamblare (nivel 1)
  - mnemonici pentru instrucțiuni, reprezentări în hexazecimal, referiri la adrese, regiștri etc.
- Limbaje de programare (nivel 2)
  - sunt independente de mașină, oferă facilități de prelucrare, învățare, depanare
- Limbaje specializate (pentru domenii restrânse)



# PROCESOARE DE LIMBAJE

- **COMPILATORUL:** translatează un program scris într-un limbaj (de nivel înalt, specializat) într-o formă care poate fi executată de calculator (cod-mașină sau cod intermediar)



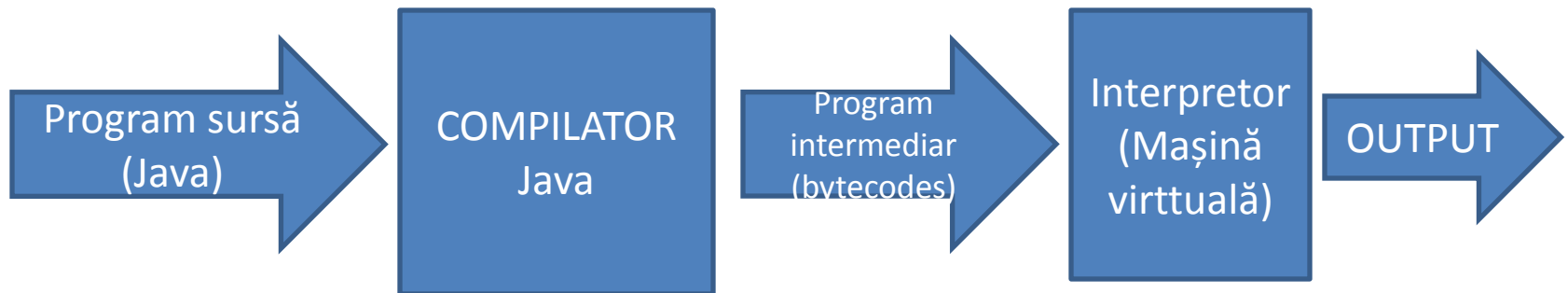
# PROCESOARE DE LIMBAJE

- ASAMBLORUL: translatează un program scris în limbaj de asamblare în cod-mașină
- INTERPRETORUL: nu produce un program țintă, ci execută direct instrucțiunile din programul sursă



# PROCESOARE DE LIMBAJE

- COMPILATORUL HIBRID: este o combinație între un compilator și un interpretor



# STRUCTURA GENERALĂ A UNUI COMPILATOR

Șir de caractere  
(programul sursă)

Analizor  
lexical  
(scanner)

Sir de token-i

Analizor  
sintactic  
(parser)

Arbore sintactic

Analizor  
semantic  
(constrainer)

Arbore modificat

FRONT  
END

Generator de  
cod  
intermediar

Reprezentare în cod intermediar

Optimizator de  
cod (indep de  
mașină)

Cod intermediar modificat

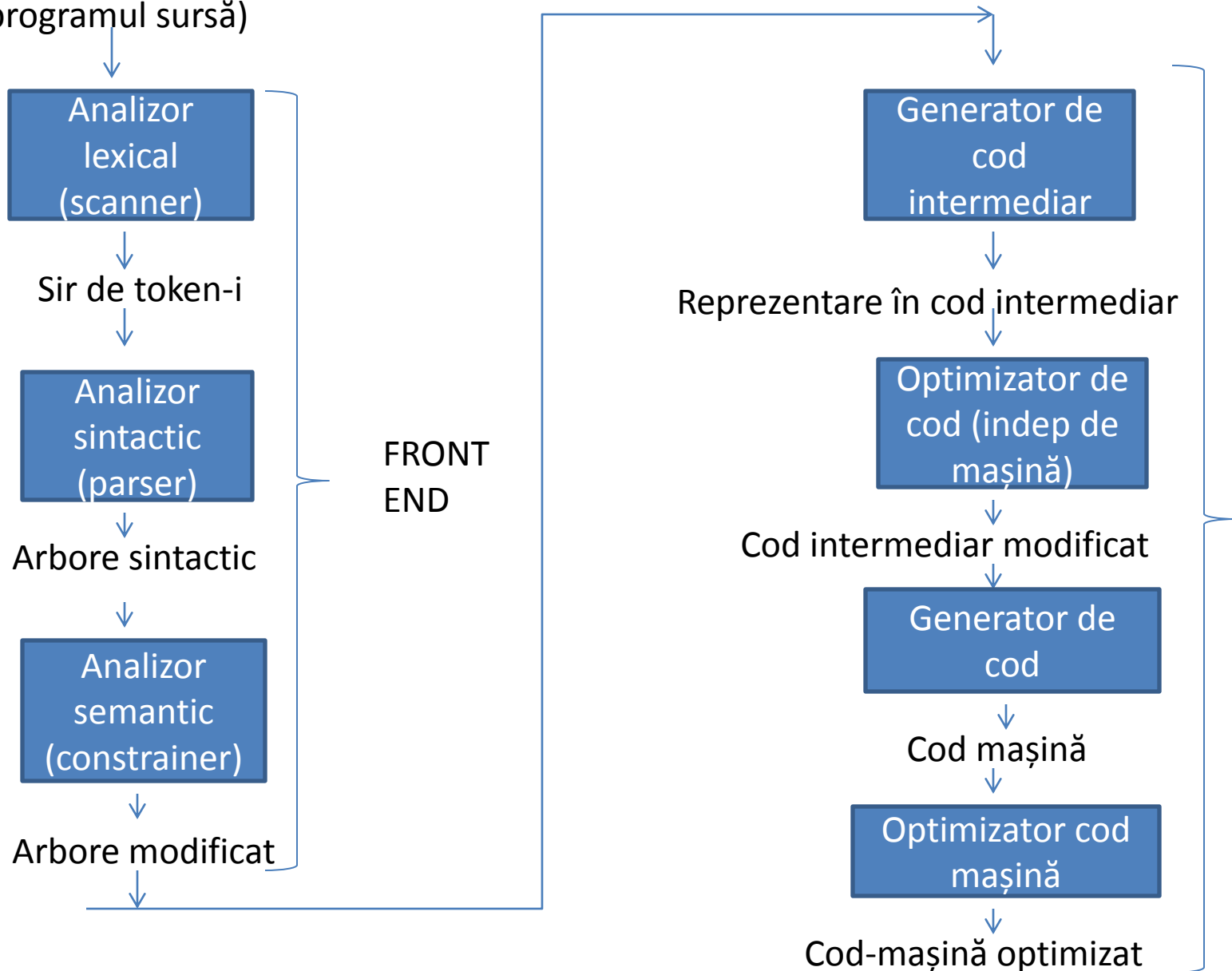
Generator de  
cod

Cod mașină

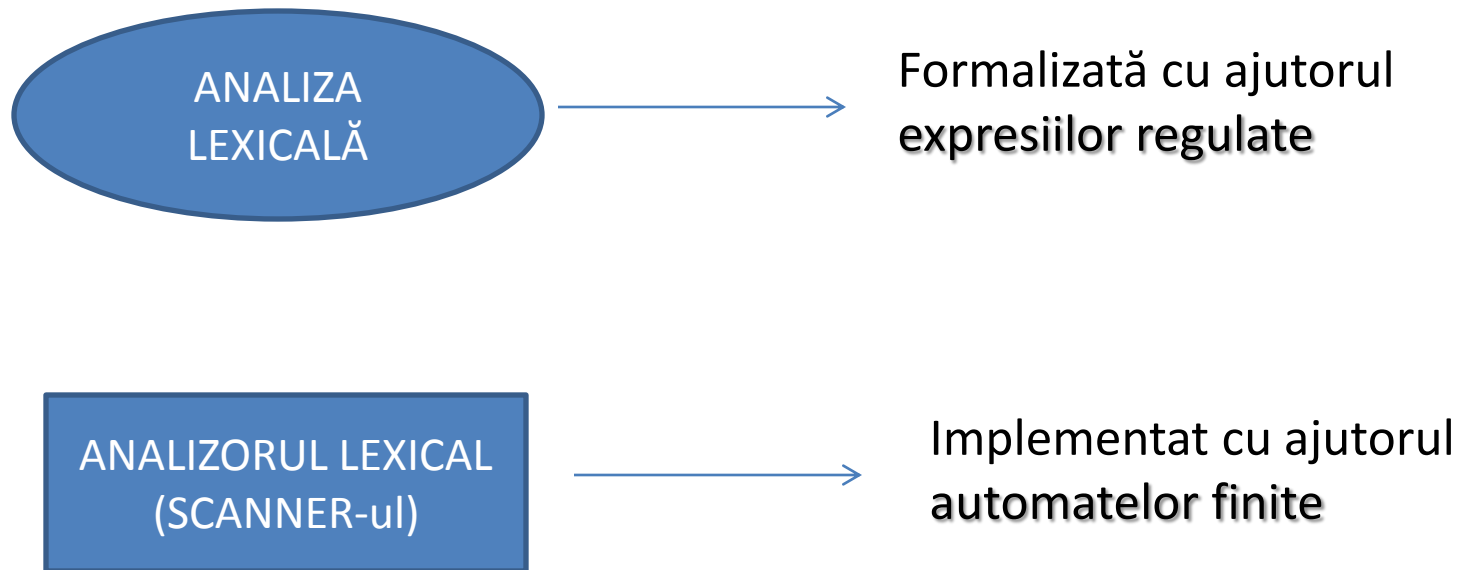
Optimizator cod  
mașină

Cod-mașină optimizat

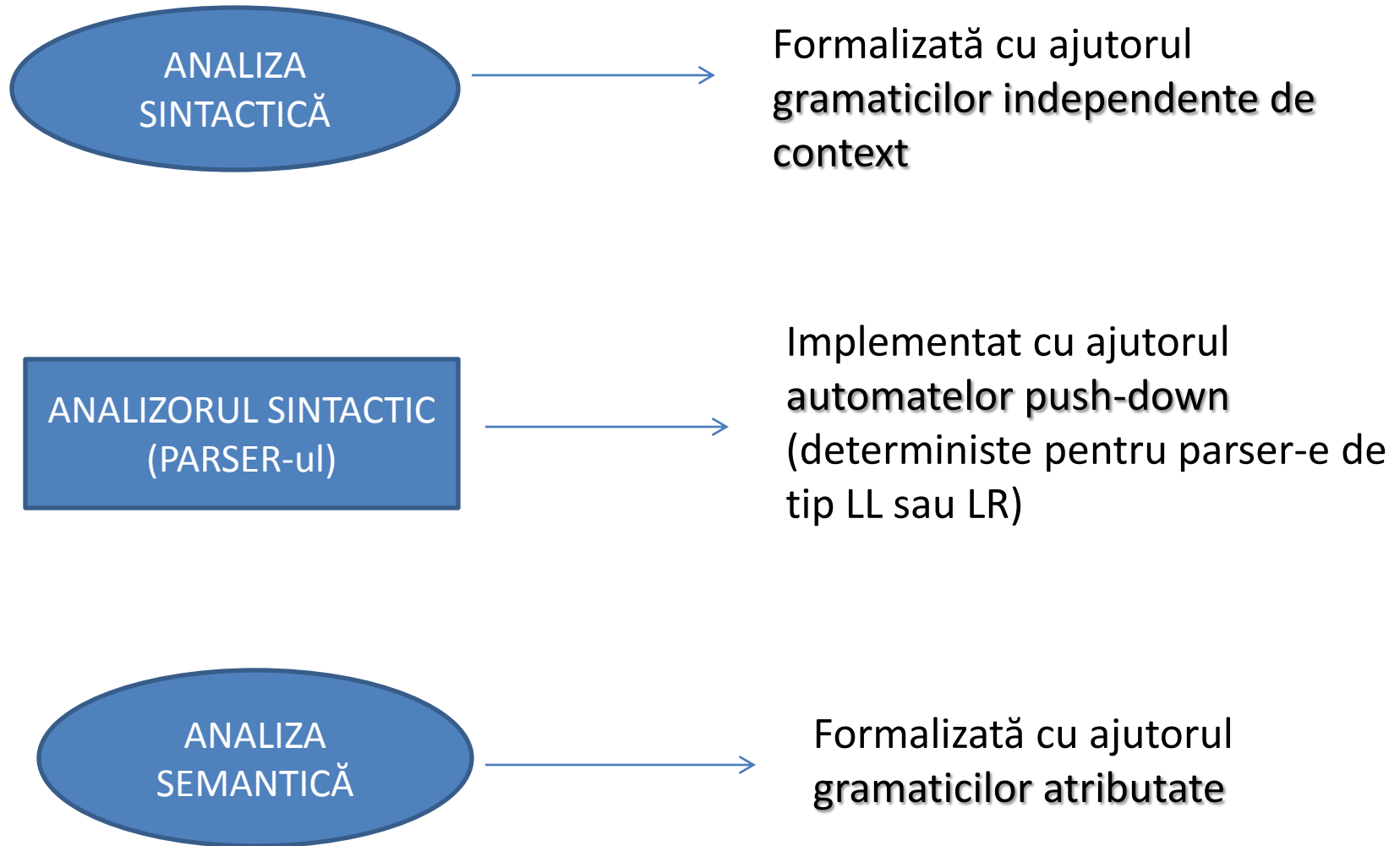
BACK  
END



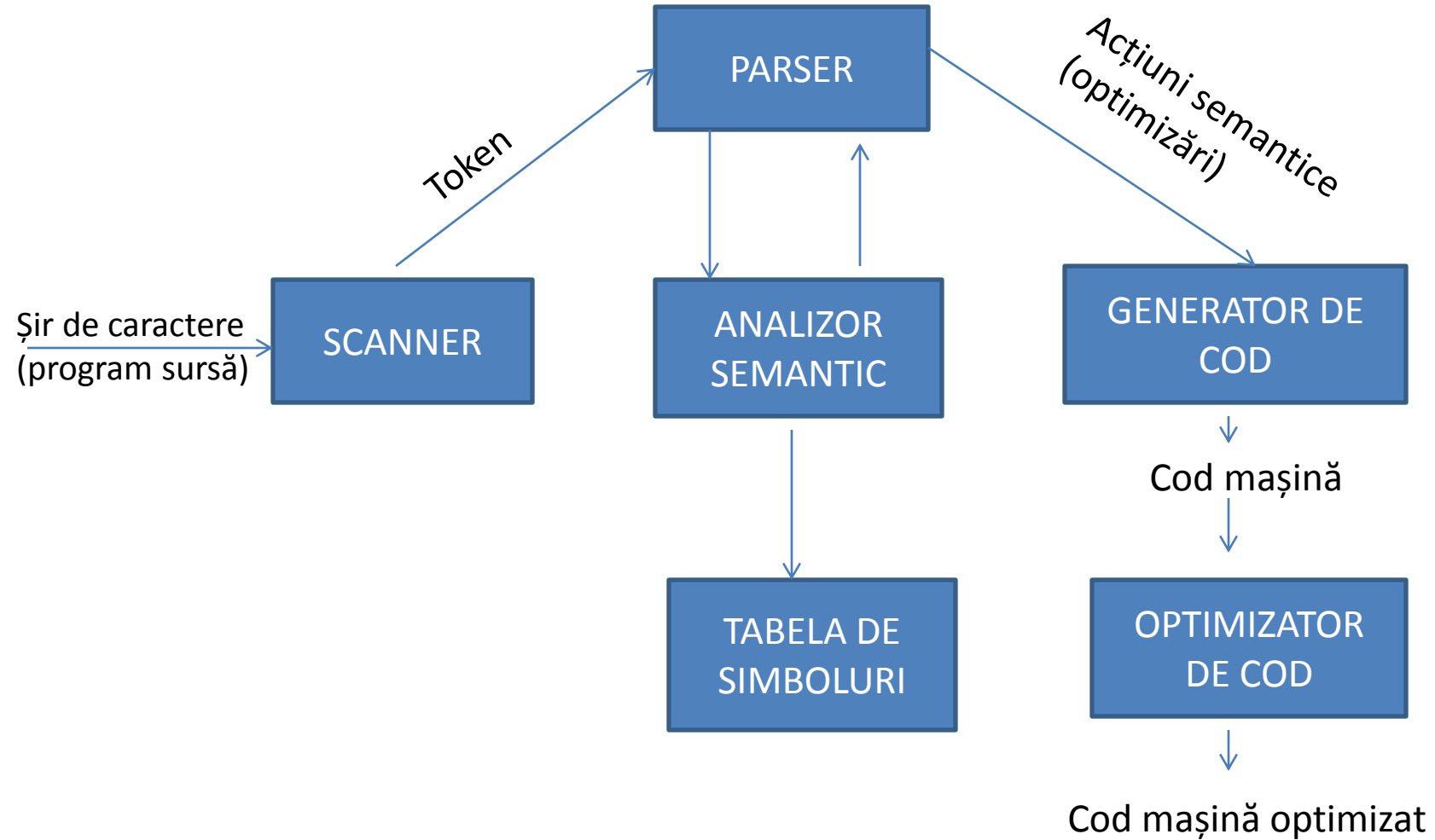
# LEGĂTURA DINTRE TEORIA COMPILĂRII ȘI LIMBAJELE FORMALE



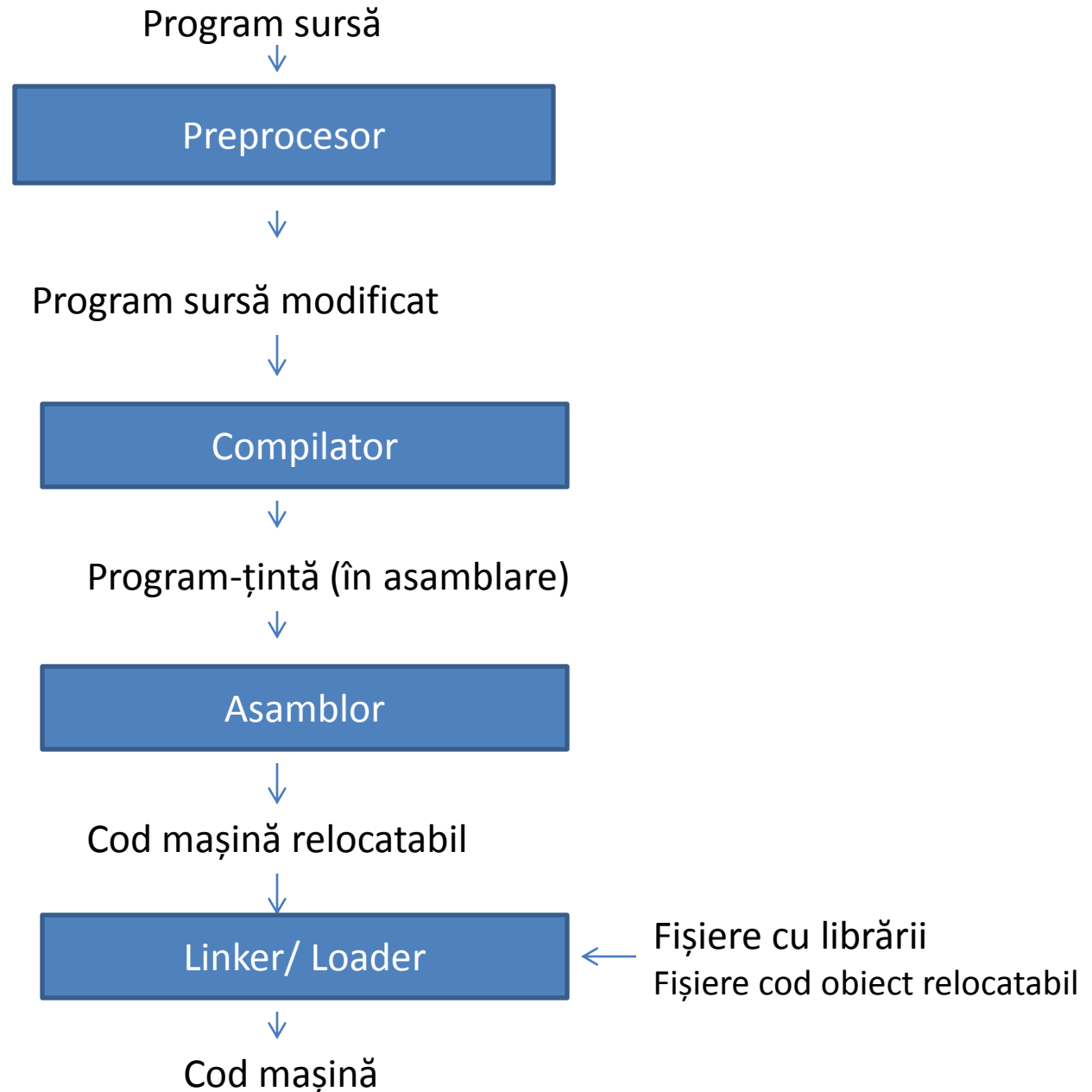
# LEGĂTURA DINTRE TEORIA COMPILĂRII ȘI LIMBAJELE FORMALE



# STRUCTURA UNUI COMPILATOR



# SISTEM DE PROCESARE A UNUI LIMBAJ





# EXEMPLU

Fie instrucțiunea  $\text{poz} = \text{init} + \text{rata} * 60$

$\text{poz}$  :token <id,1>

// id - tipul token-ului;

// 1 – poziția în tabela de simboluri;

= : token <=>

init :token <id,2>

+ : token <+>

rata:token <id,3>

\* : token <\*>

60 : token <60>

poz = init + rata \* 60



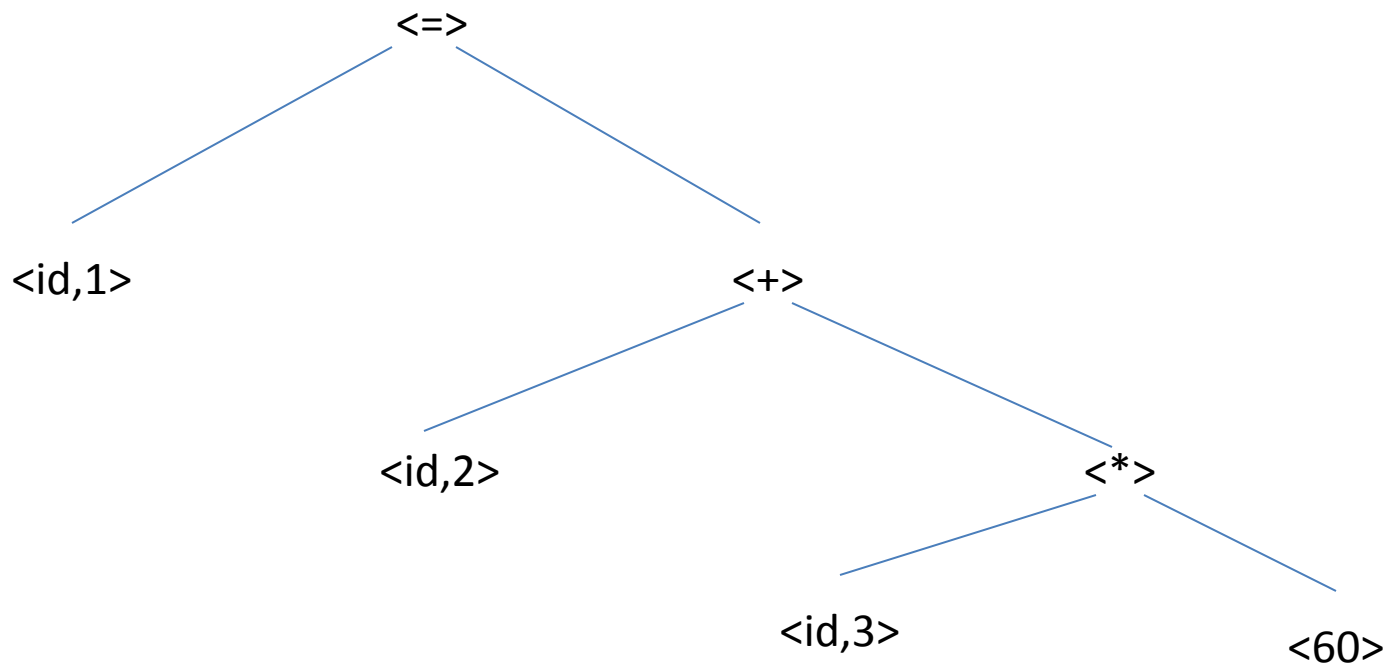
Analizor lexical

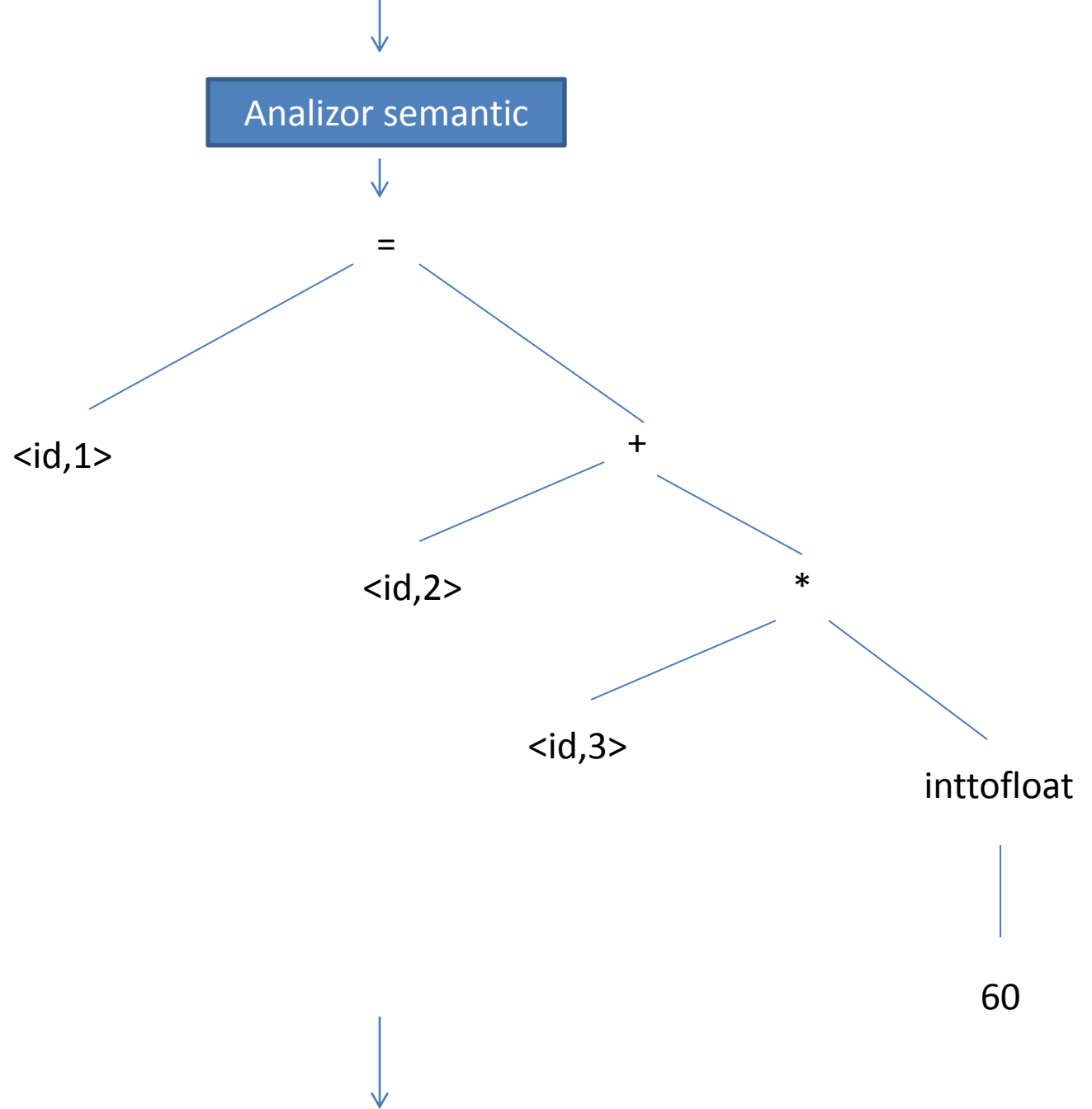


<id,1> <=> <id,2> <+> <id,3> <\*> <60>




Analizor sintactic








Generator cod intermediar




```
t1 = intofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```



Optimizator de cod



```
t1 = id3 * 60.0
id1 = id2 + t1
```





Generator de cod



```
LDF    R2,id3
MULF   R2,R2,#60
LDF    R1,id1
ADDF   R1,R1,R2
STF    id1,R1
```

LDF R2, id3 - încarcă valoarea de tip float de la adresa lui id3 în registrul R2

MULF R2,R2,#60 - înmulțește ca valori de tip float numărul de la adresa conținută de R2 cu 60 și pune rezultatul la adresa R2

ADDF - adunare de numere de tip float....

STF id1,R1 stochează la adresa lui id1 ceea ce găsești la adresa conținută de R1, ca float

# ANALIZA LEXICALĂ

## idei de bază

- Lexicul unui limbaj de programare: descris cu ajutorul expresiilor regulate
- Scanner-ul (analizorul lexical) este un program implementat ca o subrutină ce funcționează pe baza AFD corespunzător expresiei regulate ce formalizează lexicul limbajului
- Lexicul unui limbaj de programare constă din mulțimea atomilor lexicali (șiruri cu un înțeles bine stabilit, sau altfel spus "cuvintele" limbajului respectiv):  
identificatori, cuvinte cheie, constante de diferite tipuri, operatori, delimitatori, comentarii

# ANALIZORUL LEXICAL (scanner-ul)

- Implementat ca o funcție ce întoarce token-ul curent (tipul, șirul de caractere ce îi corespunde, eventual lungimea acestui șir și numărul liniei pe care se află în fișierul sursă)
- Scanează programul sursă, sărind peste spații albe, comentarii, linii noi.
- Depistează token-ul curent pe care îl transmite parser-ului sub forma: (*nume-token*, *atribut*), unde *nume-token* este un simbol abstract utilizat în tabela sintactică (de exemplu *id*), iar *atribut* indică intrarea în tabela de analiză sintactică pentru acel token.
- Depistează eventualele erori lexicale (caractere nepermise, comentarii scrise greșit etc.)

# RECAPITULARE

## Expresii regulate

- Fie  $\Sigma$  un alfabet. Definim o **expresie regulata** astfel:
  - (i)  $\phi$  este exp reg peste  $\Sigma$  care descrie limbajul vid,  $\phi$
  - (ii)  $\lambda$  este exp reg peste  $\Sigma$  care descrie limbajul  $\{\lambda\}$
  - (iii)  $\forall a \in \Sigma, a$  este exp reg peste  $\Sigma$  care descrie limbajul  $\{a\}$

Fie  $p, q$  expresii regulate peste  $\Sigma$  care descriu respectiv limbajele  $P, Q$ . Atunci:

- (iv)  $p|q, pq, p^*$  sunt expresii regulate care descriu respectiv limbajele  $P \cup Q, PQ, P^*$
- (v)  $(p)$  este exp reg peste  $\Sigma$  care descrie limbajul  $P$



# AUTOMATE FINITE

## AUTOMATE FINITE NEDETERMINISTE CU $\lambda$ -tranzitii

- Numim automat finit nedeterminist cu  $\lambda$ -tranzitii ( $AFN_\lambda$ ) o structura de forma:

$A=(Q,\Sigma,\delta,s,F)$ , unde

$Q$  multimea starilor (finita, nevida)

$\Sigma$  alfabetul automatului

$\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$

$s \in Q$  starea initiala a automatului

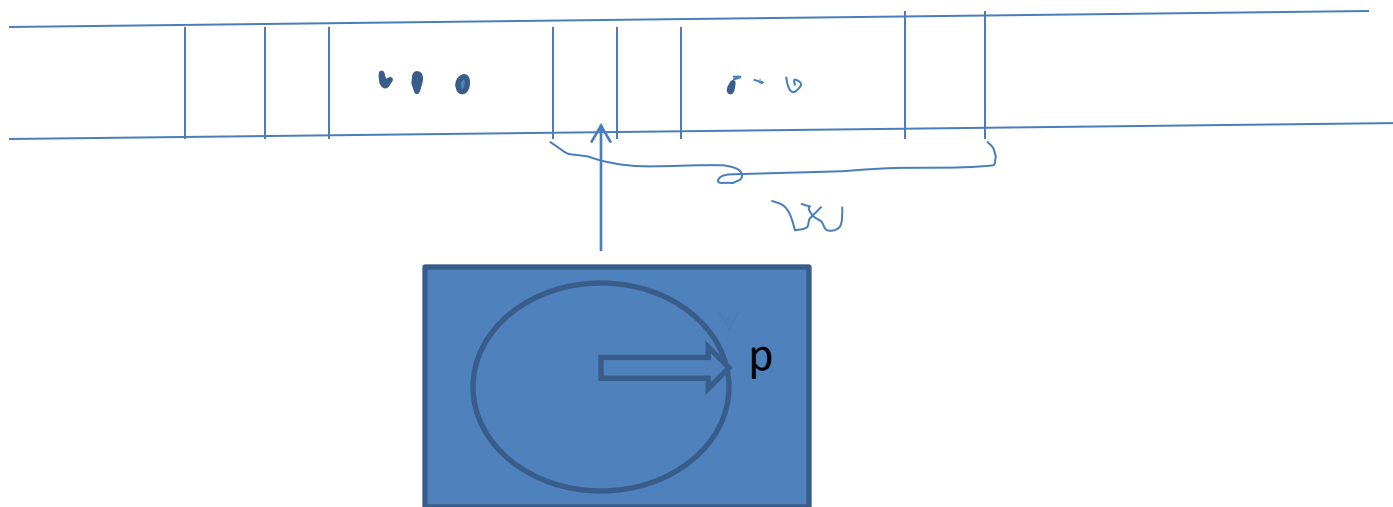
$F \subseteq Q$  multimea starilor finale

# AUTOMATE FINITE NEDETERMINISTE CU $\lambda$ -tranzitii

- Descriere instantanee (instantă a lui A)

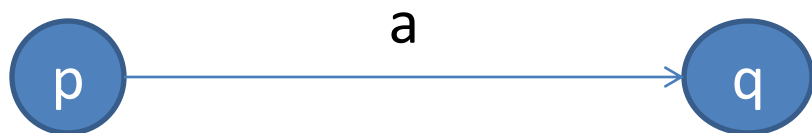
$(p, w)$   $p \in Q$  starea curentă

$w \in \Sigma^*$  sirul curent pe banda de intrare



# AUTOMATE FINITE NEDETERMINISTE CU $\lambda$ -tranzitii

- Mișcare a lui A  
 $(p, aw) \rightarrow (q, w)$  dacă și numai dacă  
 $q \in \delta(p, a)$ , unde  $p, q \in Q$ ,  $a \in \Sigma \cup \{\lambda\}$ ,  $w \in \Sigma^*$
- Notăție grafică pentru  $q \in \delta(p, a)$



- Notăm cu  $\rightarrow^*$  închiderea reflexivă și tranzitivă a relației  $\rightarrow$

# AUTOMATE FINITE NEDETERMINISTE CU $\lambda$ -tranzitii

- $\rightarrow^*$  înseamnă 0 sau mai multe mișcări ale automatului A

- Limbajul recunoscut de A este:

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

# AUTOMATE FINITE NEDETERMINISTE

- Definiție. Numim automat finit nedeterminist (AFN) o structura de forma:

$$A=(Q,\Sigma,\delta,s,F), \text{ unde}$$

$Q$  multimea starilor (finita, nevida)

$\Sigma$  alfabetul automatului

$$\delta: Q \times \Sigma \rightarrow 2^Q$$

$s \in Q$  starea initiala a automatului

$F \subseteq Q$  multimea starilor finale

# AUTOMATE FINITE NEDETERMINISTE

- Mișcare a lui A  
 $(p, aw) \rightarrow (q, w)$  dacă și numai dacă  
 $q \in \delta(p, a)$ , unde  $p, q \in Q$ ,  $a \in \Sigma$ ,  $w \in \Sigma^*$
- Notăm cu  $\rightarrow^*$  închiderea reflexivă și tranzitivă a relației  $\rightarrow$
- Limbajul recunoscut de A este:  
$$L(A) = \{w \in \Sigma^* \mid (s, w) \rightarrow^* (q, \lambda), q \in F\}$$

- Care este deosebirea dintre un  $AFN_\lambda$  și un  $AFN$ ?
- Care este relația dintre cele 2 tipuri de automate?

# AUTOMATE FINITE DETERMINISTE (AFD)

- Definiție. Numim automat finit determinist (AFN) o structura de forma:

$$A=(Q,\Sigma,\delta,s,F), \text{ unde}$$

$Q$  multimea starilor (finita, nevida)

$\Sigma$  alfabetul automatului

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

$s \in Q$  starea initiala a automatului

$F \subseteq Q$  multimea starilor finale



# AUTOMATE FINITE DETERMINISTE

- Spunem că AFD  $A$  este total dacă
$$\delta: Q \times \Sigma \rightarrow Q \text{ definită total (ca funcție)}$$
- Pe mulțimea instanțelor lui  $A$  definim:
$$(p, aw) \rightarrow (q, w) \text{ dacă și numai dacă}$$
$$q = \delta(p, a), \text{ unde } p, q \in Q, a \in \Sigma, w \in \Sigma^*$$
- Notăm cu  $\rightarrow^*$  închiderea reflexivă și tranzitivă a relației  $\rightarrow$
- Limbajul recunoscut de  $A$  se definește ca în cazurile anterioare

# AUTOMATE FINITE DETERMINISTE

- Mișcare a lui A  
 $(p, aw) \rightarrow (q, w)$  dacă și numai dacă  
 $q = \delta(p, a)$ , unde  $p, q \in Q$ ,  $a \in \Sigma$ ,  $w \in \Sigma^*$
- Notăm cu  $\rightarrow^*$  închiderea reflexivă și tranzitivă a relației  $\rightarrow$
- Limbajul recunoscut de A este:  
$$L(A) = \{w \in \Sigma^* \mid (s, w) \rightarrow^* (q, \lambda), q \in F\}$$

- Care este deosebirea dintre un AFN și un AFD?
- Dar între un AFD și un AFD total?
- Care este relația dintre un AFD și un AFD total? Dar între un AFD și un  $\text{AFN}_\lambda$  ?

# RELAȚIILE DINTRE FAMILIILE DE LIMBAJE RECUNOSCUTE DE DIFERITELE TIPURI DE AUTOMATE FINITE ȘI CELE DESCRISE DE EXPRESIILE REGULATE

$$L_{AFN\lambda} = L_{AFN} = L_{AFD} = L_{ExpReg}$$