Limbaje formale, automate şi compilatoare

Curs 7

Limbaje formale şi automate

- Limbaje de tipul 3
  - Gramatici regulate
  - Automate finite
    - Deterministe
    - Nedeterministe
  - Expresii regulate
    - a, $a \in \sum$, ε, $\emptyset$
    - $E_1.E_2$, $E_1|E_2$, E_1^*, (E_1)
- Limbaje de tipul 2
  - Gramatici de tipul 2

Plan

- Istoric
- Paşii compilării
- Analiza lexicală
  - Descriere lexicală
  - Interpretare
  - Interpretare orientată dreapta
  - Descriere lexicală bine formată

Istoric – 1940

- Programe scrise în instrucțiuni procesor
- Calculatoare puține
- Programatori puțini

Istoric – 1950

- Fortran (1957):
  - Primul compilator (expresii aritmetice, instrucțiuni, proceduri)
  - Încă este folosit pentru aplicații complexe computațional sau pentru testarea performanței
- Algol (1958):
  - Gramatici BNF (Backus-Naur Normal Form), bloc de instrucțiuni, recursie
  - Precursorul sintaxei curente
- Lisp (1958)
  - Programare funcțională
  - Structuri aroborescente, gestiunea automată a spațiului de stocare, dynamic typing
- COBOL (1959)
  - Sintaxă similară limbii engleze
  - Business oriented
  - Pune accent pe citire și scriere de date in format text și numeric

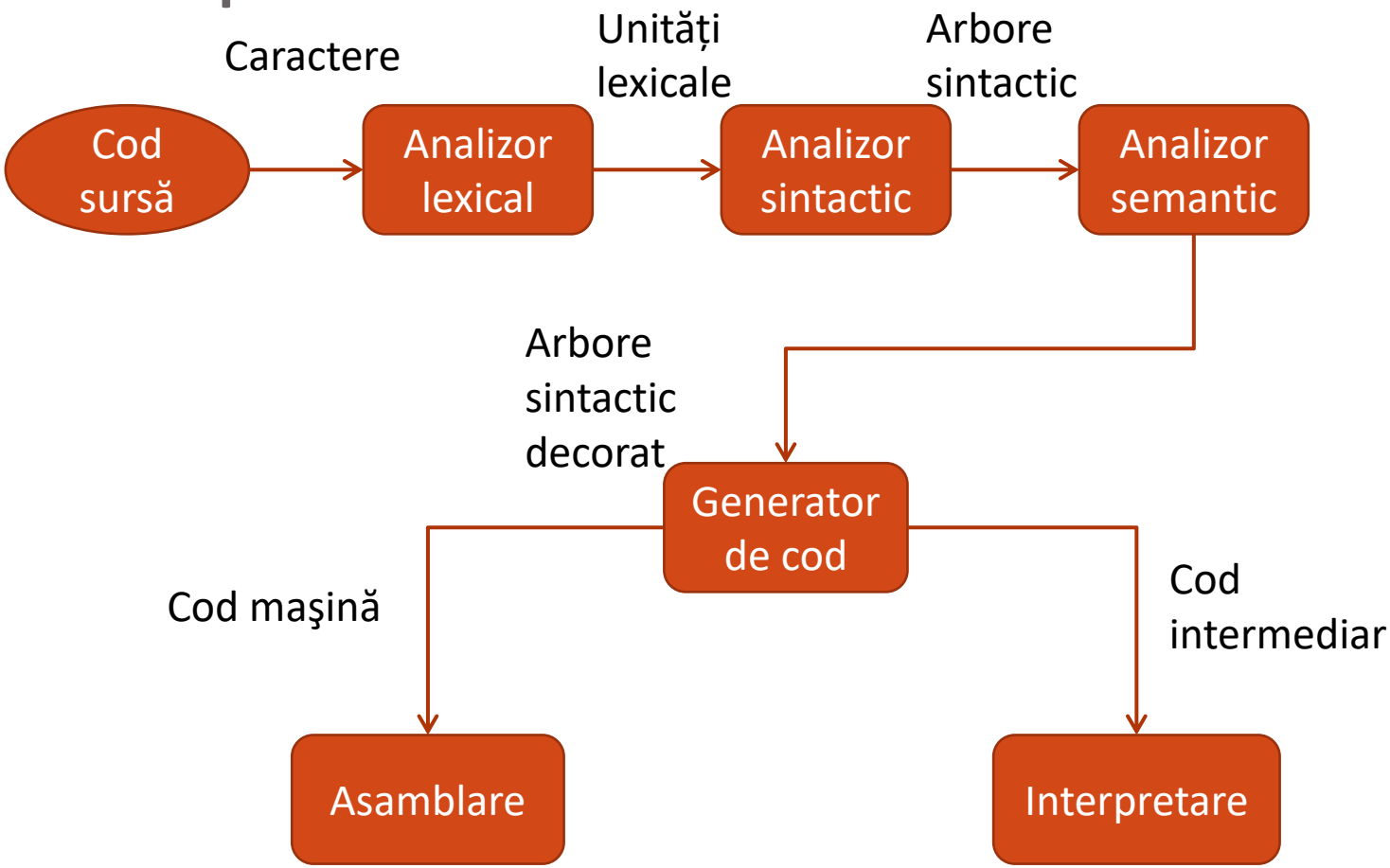
Istoric – 1960 - 1970

- Simula (1965)
  - Bazat pe ALGOL 60
  - Primul limbaj orientat obiect
  - Obiecte, clase, moștenire, funcții virtuale, etc.
- Programare structurată (1968)
  - Edsger Dijkstra – GOTO Considered Harmful
- Pascal (1970)
- C (1973)
  - IRQ, variabile dinamice, multitasking

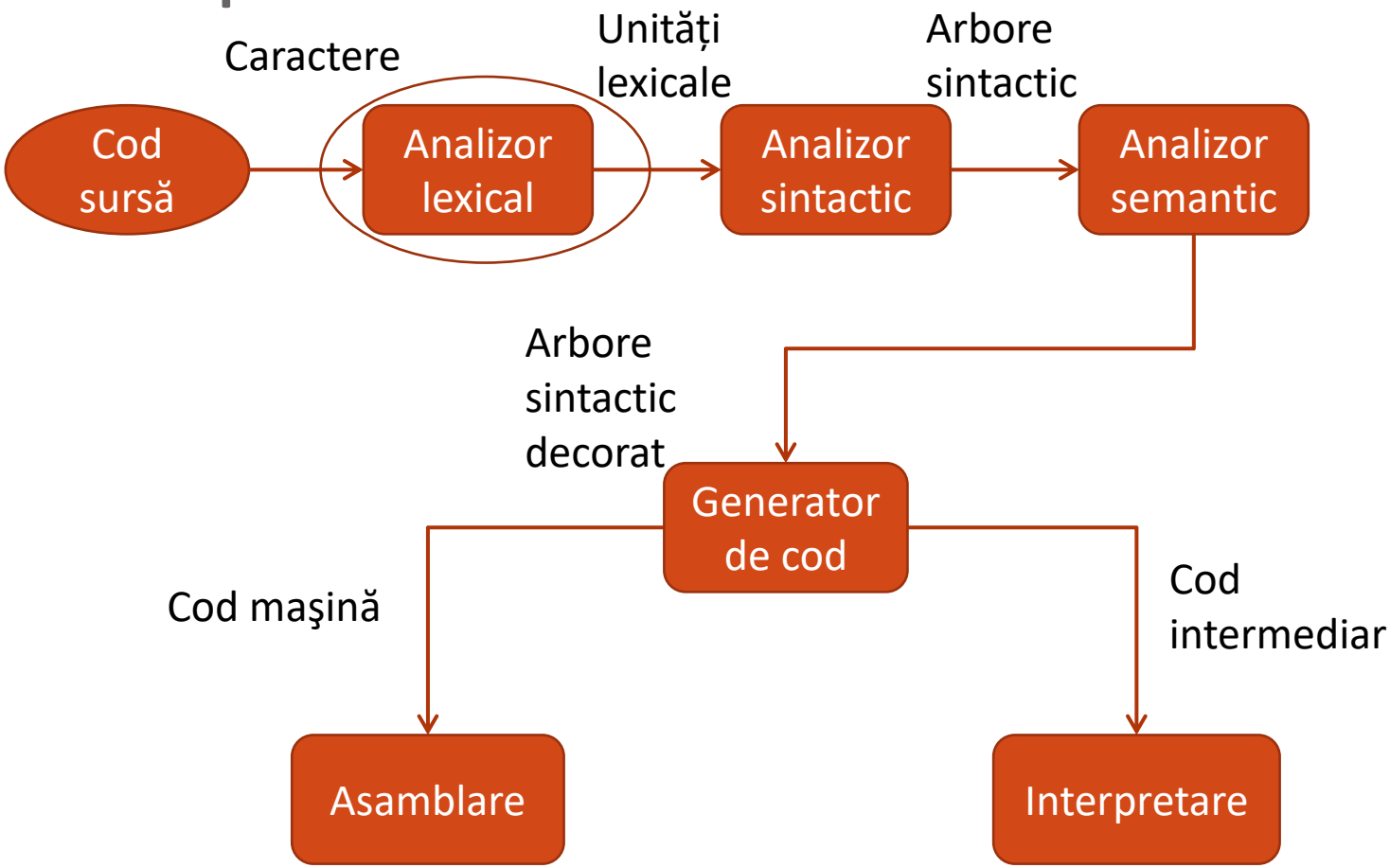
Istoric – 1980 - prezent

- ADA (1980)
  - primul limbaj standardizat
- Objective C (1984)
  - Inspirat de Smalltalk
  - Orientare obiect
- C++ (1985)
  - *C with Classes*;
  - Orientare-obiect, excepții, template-uri
  - Inspirat de Simula
- Java (1995)
  - just-in-time compilation
- C# (2000)
  - Tehnologia .NET

Compilare


Caractere
Unități lexicale
Arbore sintactic
Cod sursă
Analizor lexical
Analizor sintactic
Analizor semantic
Arbore sintactic decorat
Generator de cod
Cod maşină
Cod intermediar
Asamblare
Interpretare

Compilare


Caractere
Unități lexicale
Arbore sintactic
Cod sursă
Analizor lexical
Analizor sintactic
Analizor semantic
Arbore sintactic decorat
Generator de cod
Cod maşină
Cod intermediar
Asamblare
Interpretare

Analiza lexicală

- **Def. 1** Fie Σ un alfabet (al unui limbaj de programare). O *descriere lexicală* peste $\sum$ este o expresie regulată $E = (E_1| E_2|\ldots| E_n)^+$, unde n este numărul unităților lexicale, iar E_i descrie o unitate lexicală, $1 \leq i \leq n$.

- **Def. 2** Fie E o descriere lexicală peste Σ ce conține n unități lexicale şi $w \in \sum^+$. Cuvântul w este *corect relativ la descrierea* E dacă $w \in L(E)$. O *interpretare* a cuvântului w $\in L(E)$ este o secvență de perechi $(u_1, k_1), (u_2, k_2), \ldots, (u_m, k_m)$, unde $w = u_1u_2\ldots u_m$, $u_i \in L(E_{ki})$ $1 \leq i \leq m$, $1 \leq ki \leq n$.

Exemplu

- w = `alpha := beta = 542`
- Interpretări ale cuvântului w:
  - (`alpha`, Id), (`:=`, Asignare), (`beta`, Id), (=, Egal), (`542`, Intreg)
  - (`alp`, Id), (`ha`, Id), (`:=`, Asignare), (`beta`, Id), (=, Egal), (`542`, Intreg)
  - (`alpha`, Id), (`:`, Dp), (`=`, Egal), (`beta`, Id), (=, Egal), (`542`, Intreg)

Analiza lexicală

- **Def. 3** Fie E o descriere lexicală peste $\sum$ şi w ∈ L(E). O interpretare a cuvântului w, $(u_1, k_1)(u_2, k_2), \ldots(u_m, k_m)$, este *interpretare drept -orientată* dacă (∀) $1 \leq i \leq m$, are loc:

  $|u_i| = \max\{|v|, v \in L(E_1| E_2|\ldots| E_n) \cap Pref(u_i u_{i+1}\ldots u_m)\}$.

  (unde Pref(w) este mulțimea prefixelor cuvântului w).

- Există descrieri lexicale E în care nu orice cuvânt din L(E) admite o interpretare drept-orientată.
- $E = (a \mid ab \mid bc)^+$ şi w = abc.

Analiza lexicală

- **Def. 4** O descriere lexicală E este *bine -formată* dacă orice cuvânt w din limbajul L(E) are exact o interpretare drept-orientată.

- **Teoremă** Dată o descriere lexicală E este decidabil dacă E este bine formată.

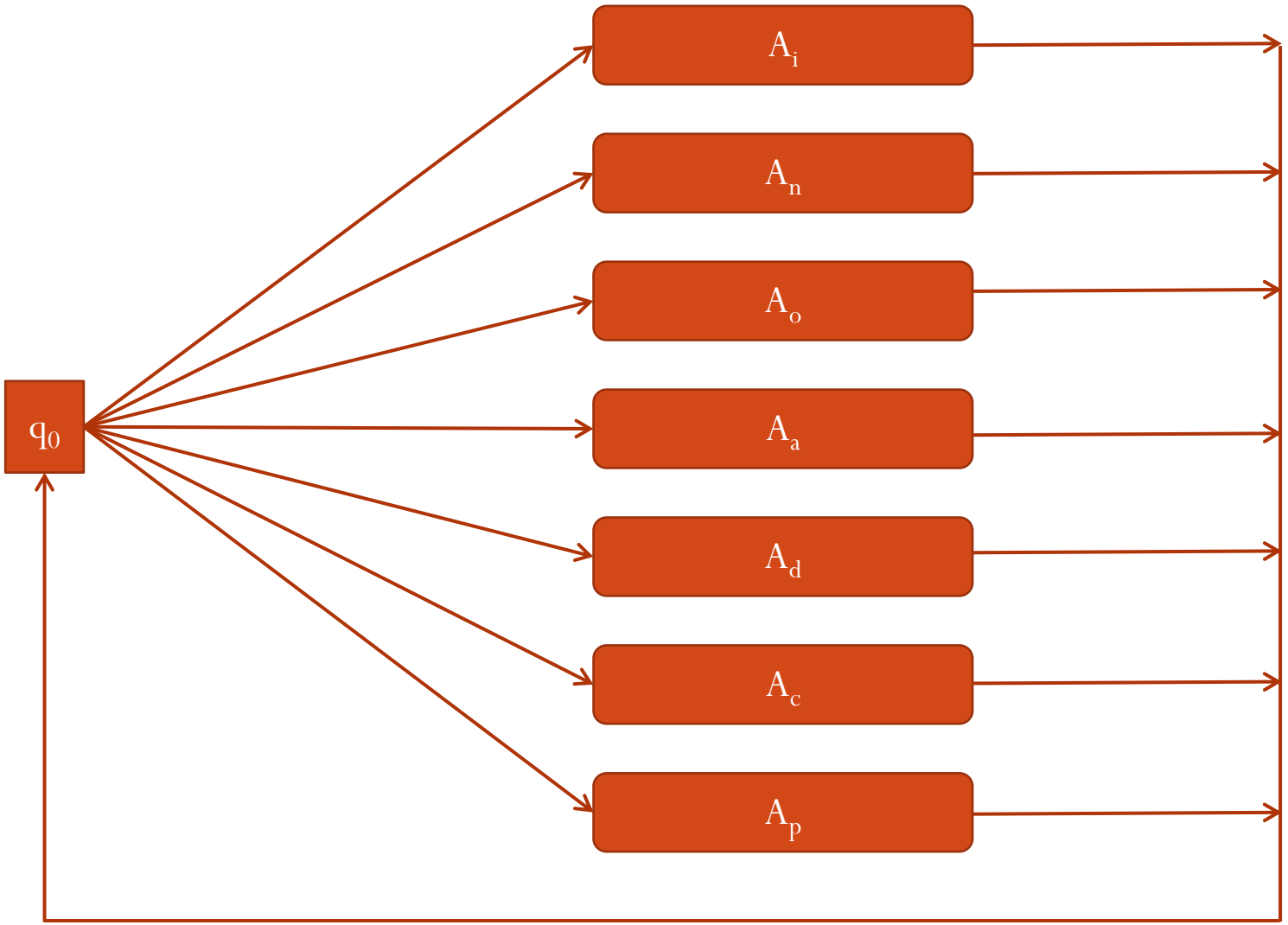
- **Def. 5** Fie E o descriere lexicală bine formată peste $\sum$. Un *analizor lexical* (*scanner*) pentru E este un program ce recunoaşte limbajul L(E) şi produce, pentru fiecare $w \in L(E)$, interpretarea sa drept-orientată.

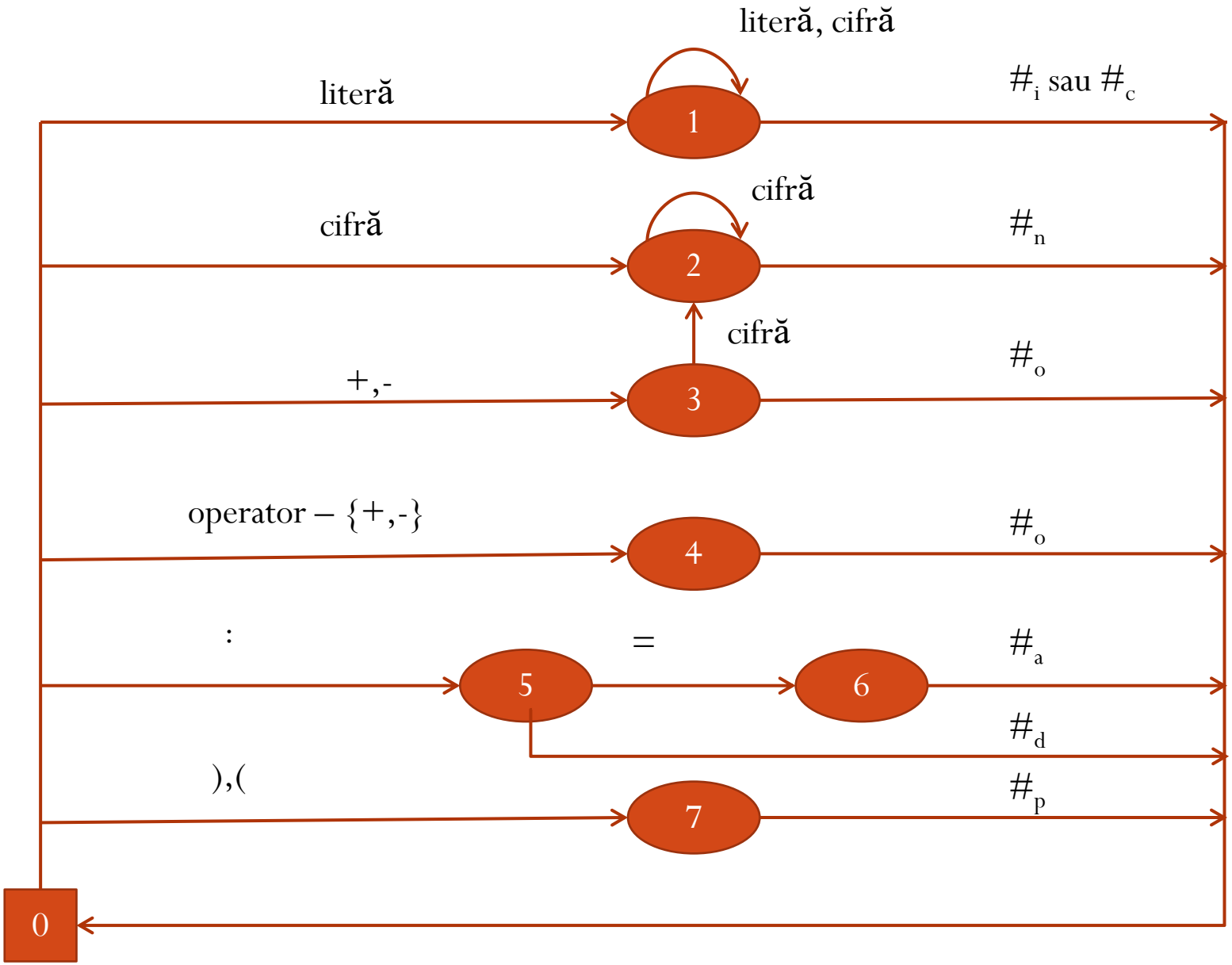
Analiza lexicală

- Fie o descriere lexicală E peste Σ. Crearea unui analizor lexical pentru E inseamnă:
  - 1. Se construieşte automatul finit echivalent A
  - 2. Din A se obține automatul determinist echivalent cu E, fie acesta A'.
  - 3. (Opțional) Automatul minimal echivalent cu A'.
  - 4. Implementarea automatului A'.

Exemplu de analizor lexical

- Fie descrierea lexicală:
  - litera → a | b |…|z
  - cifra → 0 | 1 |…| 9
  - identificator → litera (litera| cifra)*
  - semn → + | -
  - numar →(semn | ε) cifra+
  - operator → + | -| * | / | < | > | <= | >= | < >
  - asignare → :=
  - doua_puncte → :
  - cuvinte_rezervate → if| then|else
  - paranteze →) | (


A_i
A_n
A_o
q_0
A_a
A_d
A_c
A_p


literă, cifră
literă
#i sau #c
1
cifră
cifră
#n
2
cifră
#o
+,-
3
operator – {+,-}
#o
4
:
=
#a
5
6
#d
),(
#p
7
0

Bibliografie

- G. Grigoraş, *Construcția compilatoarelor. Algoritmi fundamentali*, Editura Universității "Alexandru Ioan Cuza", Iaşi, 2005