Limbaje formale, automate şi compilatoare

Curs 7

Limbaje formale şi automate

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

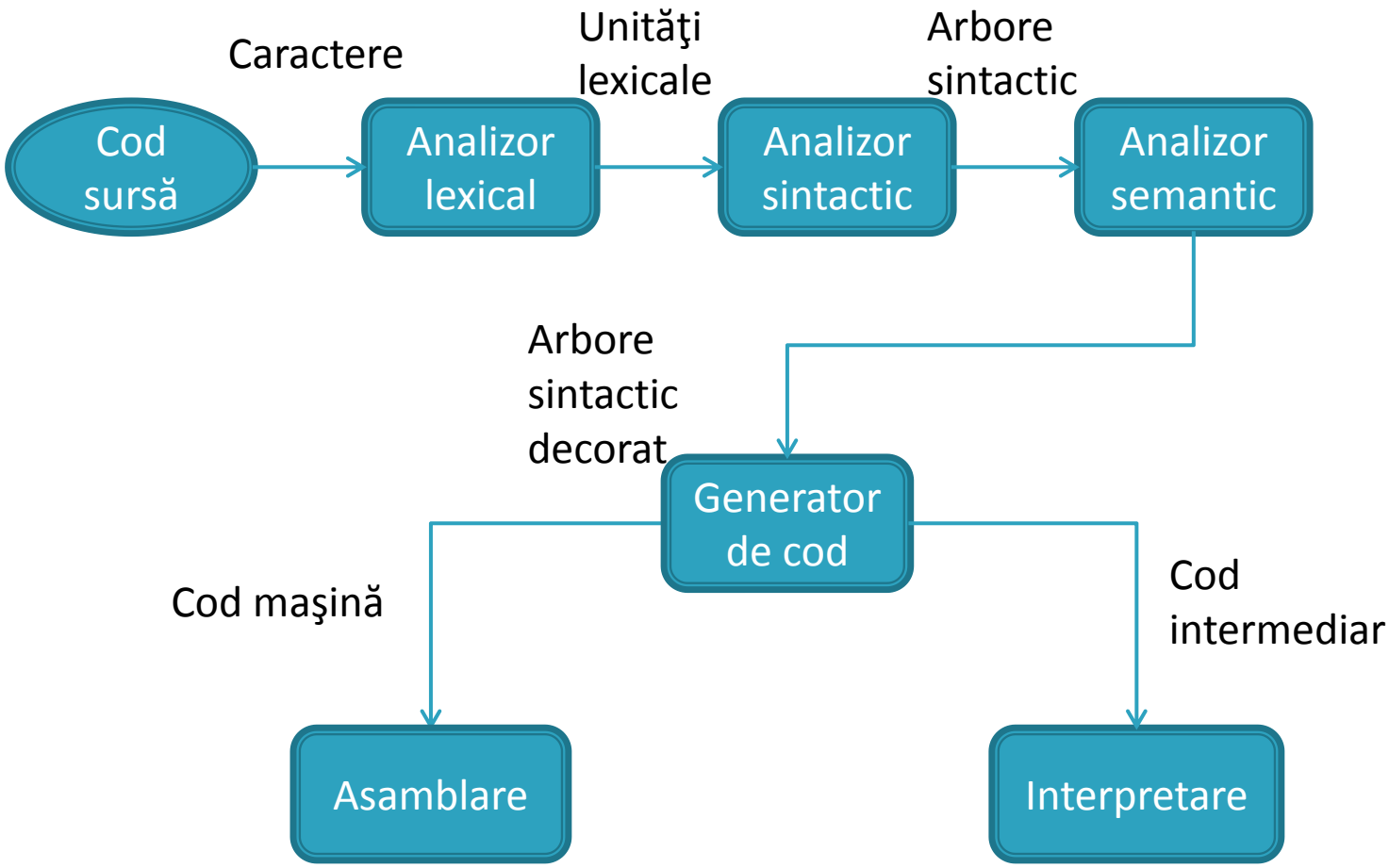
Plan

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

Istoric

- 1957 Fortran: primul compilator (expresii aritmetice, instrucţiuni, proceduri)
- 1960 Algol: prima utilizare a definiţiilor formale (gramatici, BNF, structura de bloc, recursie)
- 1970 Pascal: tipuri utilizator, maşini virtuale (P-code)
- 1972 C: variabilele dinamice, multitasking, gestionarea întreruperilor
- 1983 ADA: primul limbaj standardizat
- 1985 C++: orientare-obiect, excepţii, template-uri
- 1995 Java: just-in-time compilation
- 2000 C#: Tehnologia .NET

Compilare


Caractere
Unităţi 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 Σ este o expresie regulată $E = (E_1 | E_2 | \ldots | E_n)^+$, unde n este numărul unităţilor lexicale, iar E_i descrie o unitate lexicală, $1 \leq i \leq n$.

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

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 Σ şi w $\in$ 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 $(\forall\ i)\ 1 \leq i \leq m$, 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 mulţimea 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

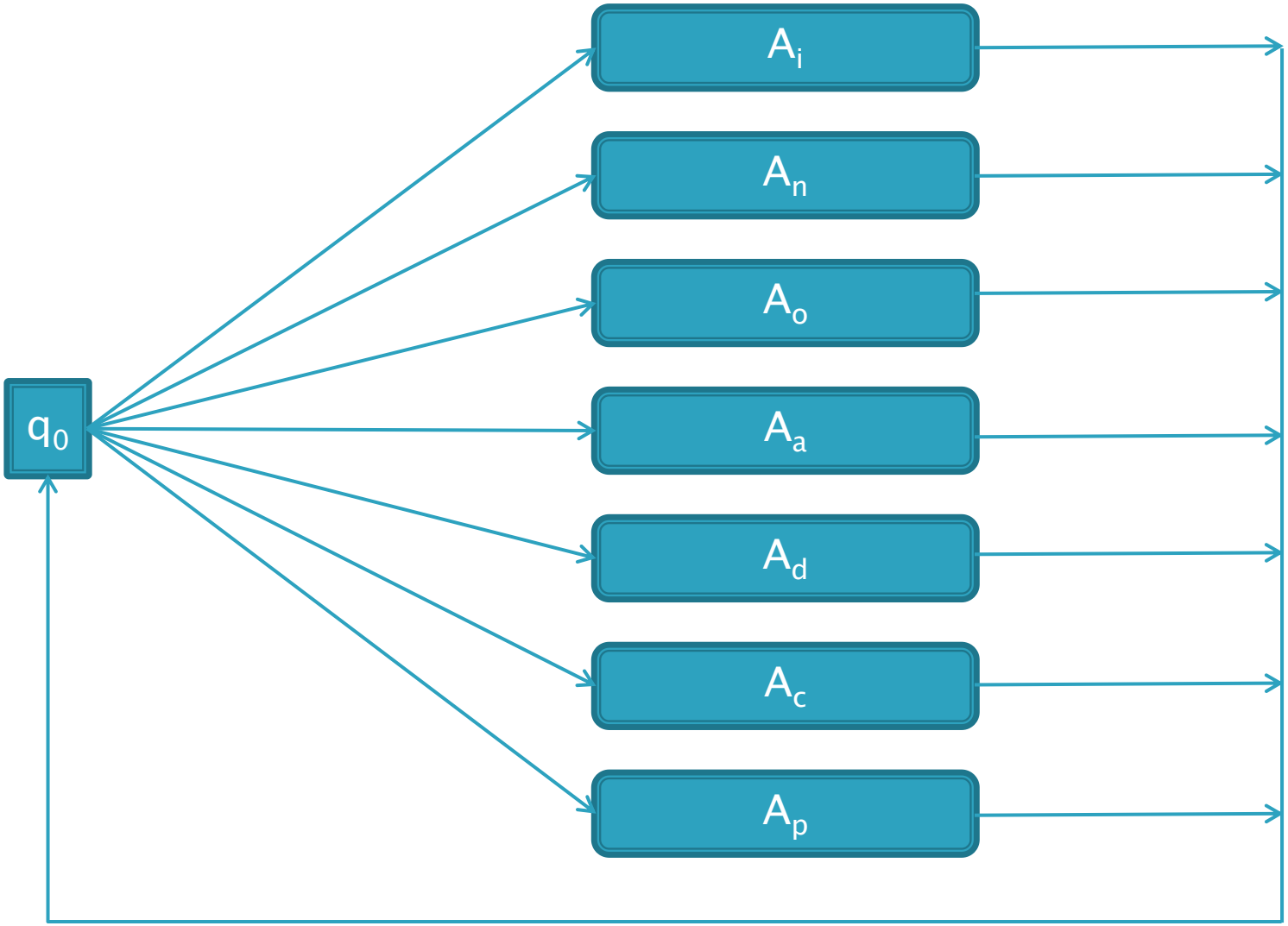
- **Def. 5** Fie E o descriere lexicală bine formată peste Σ. Un *analizor lexical* (*scanner*) pentru E este un program ce recunoaşte limbajul L(E) şi produce, pentru fiecare $w \in L(E)$, interpretarea sa drept-orientată.

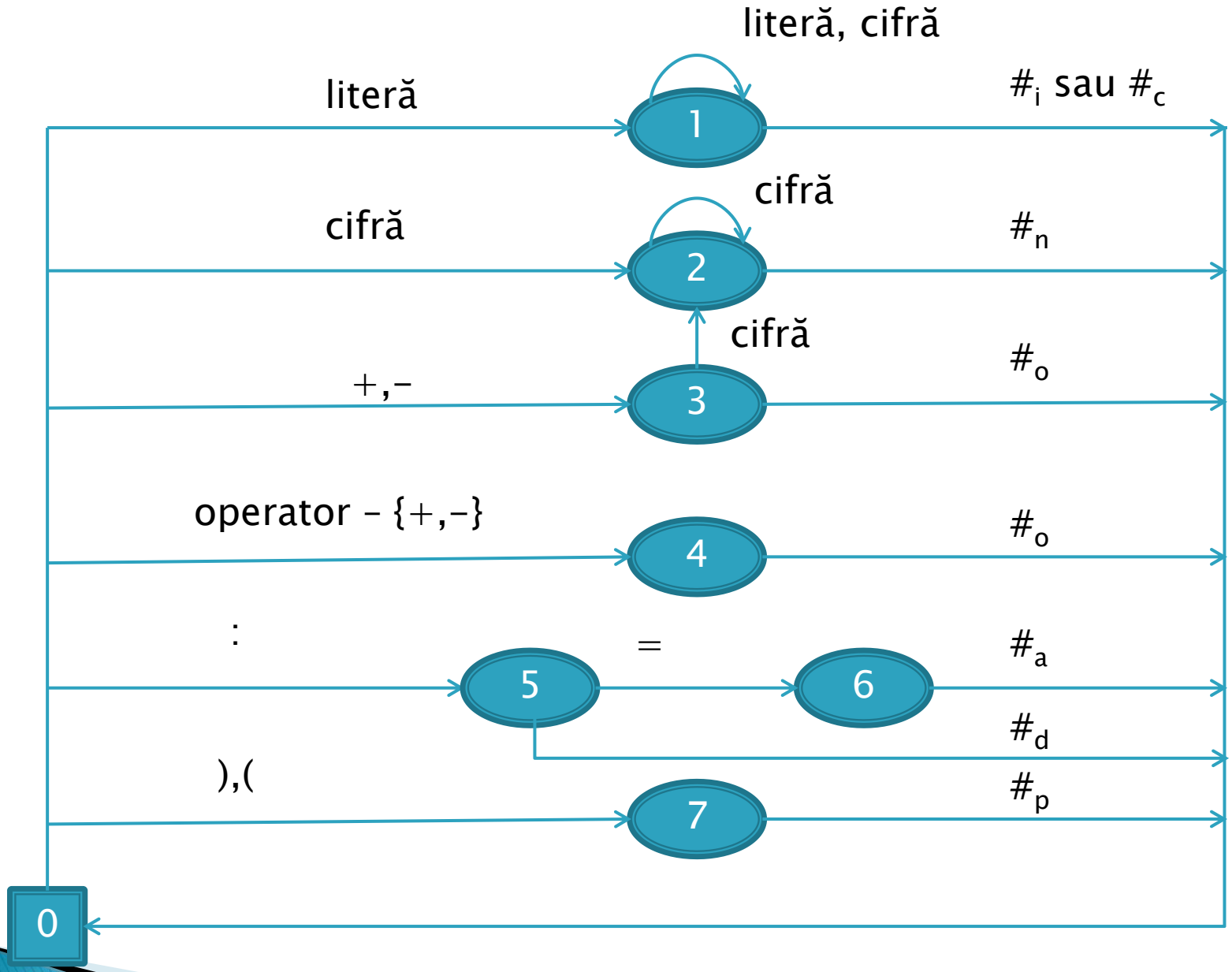
Analiza lexicală

- 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 obţine automatul determinist echivalent cu E, fie acesta A'.
  - 3. (Opţional) 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 →) | (

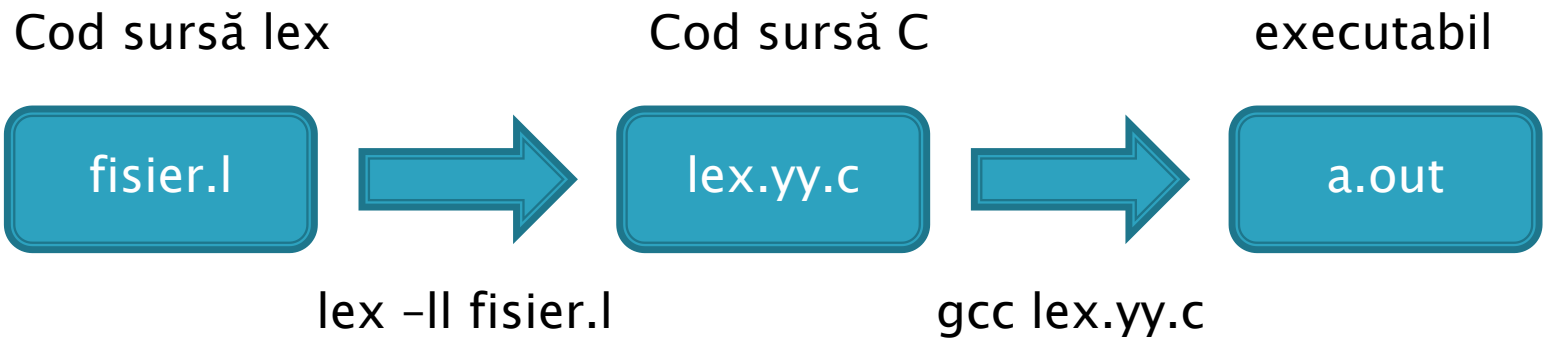

A_i
A_n
A_o
q_0
A_a
A_d
A_c
A_p


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

Lex

- Bell Laboratories 1975 M.E. Lesk şi E. Schmidt
- Standard în UNIX începând cu versiunea a 7-a
- Variante:
  - FLEX (Fast LEXical Analyzer Generator) http://flex.sourceforge.net/
  - PCLEX lansat de Abraxax Software Inc. (Windows)
  - YooLex (Yet another Object-Oriented Lex) http://yoolex.sourceforge.net/
  - Flex++: http://www.kohsuke.org/flex++bison++/ (variantele Bison, Flex care produc cod C++)

Lex – rulare


Cod sursă lex
Cod sursă C
executabil
fisier.l
lex.yy.c
a.out
lex –ll fisier.l
gcc lex.yy.c

Lex – sintaxă

- Trei segmente, separate de %%
  - Declaraţii
  - Reguli
  - Cod C

- Declaraţiile conţin
  - Declaraţii C, între secvenţele rezervate %{, %}
  - Definiţii Lex pentru segmentul de reguli

Lex – sintaxa

- O definiţie Lex are forma
  - <nume> <expresie_regulată>
  - Expresiile regulate sunt construite pornind de la orice caracter şi folosind operatorii
    - " \[ ] ^ -? . * + | () / {} % <>

  - `cifra  [0-9]`
  - `litera [a-zA-Z]`

Lex – simboluri rezervate

Simbol	Descriere
.	orice caracter cu excepţia newline
\	secvenţă escape
*	zero sau mai multe copii ale expresiei precedente
+	una sau mai multe copii ale expresiei precedente
?	zero sau o copie a expresiei precedente
^	negaţie
a|b	a sau b
()	grupare de caractere
a+b	literalul"a+b“
[]	clasa de caractere

Lex – expresii

Expresia	Candidaţi ce se potrivesc
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	unul dintre caracterele: a, b, c
[a-z]	orice literă mică
[a\-z]	unul din caracterele: a, -, z
[-az]	unul din caracterele: -, a, z
[A-Za-z0-9]+	unul sau mai multe caractere alfanumerice
[\t\n]+	spaţii
[^ab]	orice cu excepţia caracterelor: a, b
[a^b]	unul din caracterele : a, ^, b
[a|b]	unul din caracterele : a, |, b
a|b	unul din caracterele : a, b

Lex – sintaxa

- Secţiunea de reguli
  - exp_1 {Acţiune_1}
  - exp_2 {Acţiune_2}
  - .
  - .
  - .
  - exp_n {Acţiune_n}
- Regulile sunt aplicate în ordinea scrierii
- Prima regulă care acceptă cuvântul este aleasă

Lex – elemente predefinite

Nume	Descriere
int yylex(void)	Apelul către analizor
char *yytext	pointer la cuvântul găsit
yyleng	lungimea cuvântului găsit
yylval	valoarea asociată cuvântului
FILE *yyout	fişierul de ieşire
FILE *yyin	fişierul de intrare

Exemplul 1

```
%{
int yylineno;
%}
%%
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);
%%
int main(int argc, char *argv[]) {
  yyin = fopen(argv[1], "r");
  yylex();
  fclose(yyin);
}
```

Exemplul 2

```
letter [A-Za-z]
digit [0-9]
%{
int count;
%}
%%
{letter}({letter}|{digit})* {count++;}
.{}
%%
int main(void) {
  yylex();
  printf("number of identifiers = %d\n", count);
  return 0;
}
```

Exemplul 3

```
%{
int nchar, nword, nline;
%}
%%
\n { nline++; nchar++; }
[^ \t\n]+ { nword++, nchar += yyleng; }
. { nchar++; }
%%
int main(void) {
  yylex();
  printf("%d\t%d\t%d\n", nchar, nword, nline);
  return 0;
}
```

Exemplul 4

```
%{
# include <stdio.h>
%}
litera [a-zA-Z]
cifra [0-9]
cifre ({cifra})+
semn [+-]
operator [+*/<>=-]
spatiu [' ''\t''\n']
%%
"if" | "then" | "else"          {printf("%s cuvant rezervat\n", yytext);}
({litera})({litera}|{cifra})*   {printf("%s identificator\n", yytext);}
{cifre}|({semn})({cifre})       {printf("%s numar intreg\n", yytext);}
{operator}                      {printf("%c operator\n", yytext[0]);}
\:\=                            {printf("%s asignare\n", yytext);}
\:                              {printf("%c doua puncte\n", yytext[0]);}
(\()|(\))                       {printf("%c paranteza\n", yytext[0]);}
{spatiu}                        {}
.                               {printf("%c caracter ilegal\n", yytext[0]);}
%%
int main( ){
   yylex( );
   return 0;
}
```

Bibliografie

- Grigoraş Gh., *Construcţia compilatoarelor. Algoritmi fundamentali*, Editura Universităţii "Alexandru Ioan Cuza", Iaşi, 2005