

# Expresii regulate

March 27, 2023

# Outline

Gramatici regulate 2. Expresii regulate

# Regular expression - expresii regulate

Expresiile regulate descriu limbajele regulate

Fie  $V$  un vocabular si simbolurile  $E, \varepsilon, +, *, (, ) \notin V$ .

Un string  $\rho$  peste  $V \cup \{E, \varepsilon, +, *, (, )\}$  este o **expresie regulata** peste  $V$  daca

1.  $\rho$  este un simbol peste  $V$  sau unul dintre simbolurile  $E, \varepsilon$ , sau
2.  $\rho$  este de forma  $(X + Y)$ ,  $(XY)$ ,  $(X)^*$ , unde  $X$  si  $Y$  sunt expresii regulate.

# Descriere expresii regulate

- ▶  $E = \emptyset$  este limbajul empty
- ▶  $\varepsilon = \{\varepsilon\}$  este limbajul format din stringul empty
- ▶  $v, v \in V$  descrie limbajul  $\{v\}$
- ▶  $(X + Y) = \{w | w \in X \text{ sau } w \in Y\}$
- ▶  $XY = \{\chi\gamma | \chi \in X \text{ si } \gamma \in Y\}$
- ▶ operatorul  $*$  inchidere (Kleene closure):

$$X^* = \varepsilon + X + XX + XXX + \dots$$

## expresii regulate 2

- ▶ Parantezele se pot omite
- ▶  $*$  este operator unar, cu prioritate mai mare decat oricare operator binar
- ▶  $+$  are prioritate mai mica decat concatenarea

$W + XY^*$  este echivalent cu  $(W + (X (Y^*) ) )$

## Exemple expresii regulate

►  $01 = \{01\}$

## Exemple expresii regulate

- ▶  $01 = \{01\}$
- ▶  $01 + 0 = \{01, 0\}$  - in lex |

## Exemple expresii regulate

- ▶  $01 = \{01\}$
- ▶  $01 + 0 = \{01, 0\}$  - in lex |
- ▶  $0(1 + 0) = \{01, 00\}$



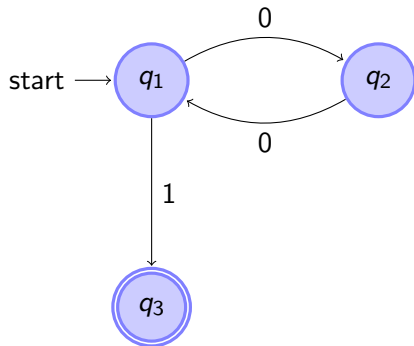
# Exemple expresii regulate

- ▶  $01 = \{01\}$
- ▶  $01 + 0 = \{01, 0\}$  - in lex |
- ▶  $0(1 + 0) = \{01, 00\}$
- ▶  $0^* = \{\varepsilon, 0, 00, 000, \dots\}$

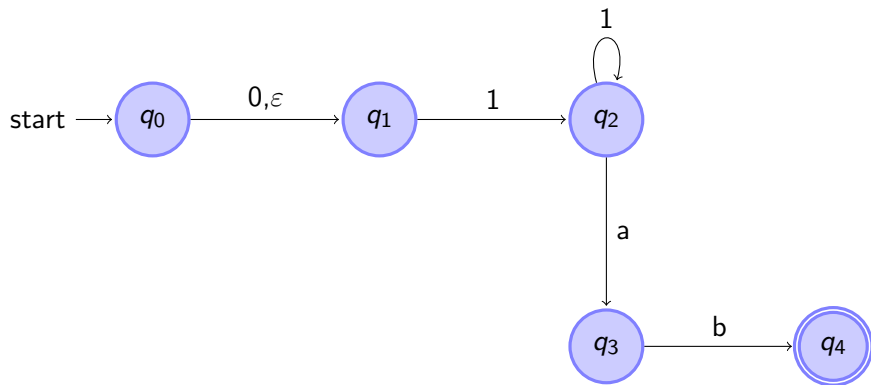
# Exemple expresii regulate

- ▶  $01 = \{01\}$
- ▶  $01 + 0 = \{01, 0\}$  - in lex |
- ▶  $0(1 + 0) = \{01, 00\}$
- ▶  $0^* = \{\varepsilon, 0, 00, 000, \dots\}$
- ▶  $(0 + 10)^*(\varepsilon + 1)$  Toate stringurile de 0 si 1 fara doua consecutive 1

## Exemplu $(00)^*1$



# Exemplu $0?1^+ab$



# Operatori aditionali: ? +

Nu permit definirea unor limbaje aditionale, dar permit exprimarea mai usoara a expresiilor regulate

- ▶ Operatorul optional: ?

Daca  $R$  este o expresie regulata,  $R? = \varepsilon + R$

- ▶ Operatorul +

Daca  $R$  este o expresie regulata  $R^+ = RR^*$ :

$$L(R^+) = L(R) \cup L(RR) \cup L(RRR) \cup \dots$$

# Proprietati algebrice ale expresiilor regulate

$$X + Y = Y + X$$

*comutativitate*

$$(X + Y) + Z = Z + (Y + Z)$$

*asociativitate*

$$X(YZ) = (XY)Z$$

$$X(Y + Z) = XY + XZ$$

*distributivitate*

$$(X + Y)Z = XZ + YZ$$

$$X + \emptyset = \emptyset + X = X$$

*identitate*

$$X\varepsilon = \varepsilon X$$

$$X\emptyset = \emptyset X = X$$

*zero*

$$X + X = X$$

*idempotentă*

$$(X^*)^* = X^*$$

$$X^* = \varepsilon + XX^*$$

$$X^* = X + X^*$$

$$\varepsilon^* = \varepsilon$$

$$\emptyset^* = \varepsilon$$

# Echivalenta expresii regulate - automate finite

Fie  $R$  o expresie regulata care descrie un subset  $S \subseteq T^*$ .

Exista un automat finit determinist  $A = (T, Q, P, q_0, F)$  a.i.  
 $L(A) = S$ .

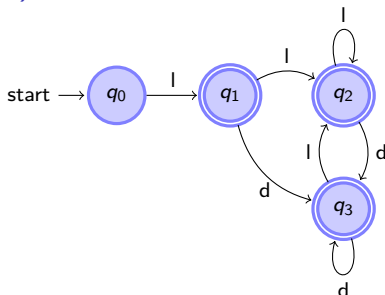
# Construire automat

1.  $R = l(l + d)^*$
2.  $R' = 1(2 + 3)^*$  - o noua expresie in care se inlocuiesc elementele lui  $T$  din  $R$  cu simboluri distincte  
Aparitii multiple ale aceluiasi element - simboluri diferite
3.  $R' = 01(2 + 3)^*$  - se adauga un prefix (un simbol distinct)  
Daca  $R = E$  atunci  $R'$  este doar simbolul de start
4. starile automatului corespund submultimilor setului de simboluri.
5. Se inspecteaza pe rand starile lui  $Q$  si daca e necesar, se adauga stari noi:  
*pentru  $\forall q \in Q$  si  $\forall t \in T$ , fie  $q'$  corespondentul setului de simboluri din  $R'$ :*
  - ▶ care inlocuiesc pe  $t$  si
  - ▶ urmeaza unui simbol din setul corespunzator lui  $q$Daca setul corespunzator lui  $q'$  nu e vid, se adauga  $qt \rightarrow q'$  la  $P$  si se include  $q'$  in  $Q$ .
6. setul  $F$  de starile finale = toate starile care includ un simbol final posibil al lui  $R'$



$$I(I + d)^*. \quad R' = 01(2 + 3)^*$$

	I	d	
$q_0$	$q_1$		$\{0\}$
$q_1$	$q_2$	$q_3$	$\{1\}$
$q_2$	$q_2$	$q_3$	$\{2\}$
$q_3$	$q_2$	$q_3$	$\{3\}$



$q_0$  I:  $\{1, 2\}$ , dar numai 1 urmeaza lui 0

$q_0$  d:  $\{3\}$ , dar nu urmeaza lui 0

$q_1$  I:  $\{1, 2\}$ , dar numai 2 urmeaza lui 1

$q_1$  d:  $\{3\}$  si 3 urmeaza lui 1

$q_2$  I:  $\{1, 2\}$ , dar numai 2 urmeaza lui 2

$q_2$  d:  $\{3\}$ , si 3 urmeaza lui 2

$q_3$  I:  $\{1, 2\}$ , dar numai 2 urmeaza lui 3 -

$q_3$  d:  $\{3\}$ , si 3 urmeaza lui 3 -

deci  $q_0 I \rightarrow q_1$

deci  $q_1 d$  nu face parte din

deci  $q_1 I \rightarrow q_2$

deci  $q_1 d \rightarrow q_3$

deci  $q_2 I \rightarrow q_2$

deci  $q_2 d \rightarrow q_3$

deci  $q_3 I \rightarrow q_2$

deci  $q_3 d \rightarrow q_3$

Stari finale:  $q_1$ ,  $q_2$  si  $q_3$

# Conversie expresie regulata - automat finit determinist - continuare

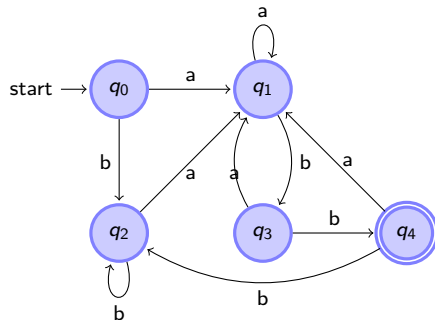
$(a + b) * abb$  becomes  $0(1 + 2)^*345$

	a	b		a	follow	b	follow
$q_0$	$q_1$	$q_2$	0	1,3	1,3	2,4,5	2
$q_1$	$q_1$	$q_3$	1,3	1,3	1,3	2,4,5	2,4
$q_2$	$q_1$	$q_2$	2	1,3	1,3	2,4,5	2
$q_3$	$q_1$	$q_4$	2,4	1,3	1,3	2,4,5	2,5
$q_4$	$q_1$	$q_2$	2,5	1,3	1,3	2,4,5	2

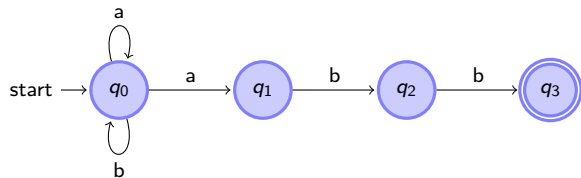
Test online: [dfa\\_toolbox](#), [dfa\\_nfa\\_pda](#)

# DFA NFA: $(a+b)^*abb$

DFA



NFA



# Alternativa

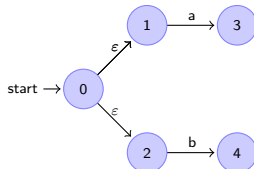
1.  $a$



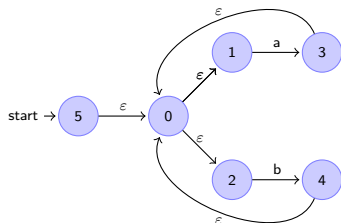
2.  $b$



3.  $a + b$



4.  $(a + b)^*$



5.  $ab$



# Expresii regulate - backtracking recursiv vs automate finite deterministe

fast vs simple

Test regex in python vs lex for a string of length  $n$

$$(a?)^n a \{n\}$$

Lex:

```
%%  
(a?){100} a{100}          printf("Easy with DFA");  
.
```

Python:

```
import re  
xx='a'*30  
print(re.match('(a?){30}a{30}',xx) )
```

test for  $a^{30}$  for python and 100 for lex

## why? Concluzii

- ▶ Assuming that the **language** has been described by a **grammar**, we are interested in techniques for **automatically** generating a **recognizer from that grammar**. There are two reasons for this requirement:
  - ▶ It provides a guarantee that the language recognized by the compiler is identical to that defined by the grammar.
  - ▶ It simplifies the task of the compiler writer.

Expresiile regulate = un mod algebric de a descrie limbajele  
Descriu limbajele regulate

$$a^n b^n$$

NU e un limbaj regulat: nu exista niciun automat finit care sa-l aiba ca limbaj

Dar limbajul *aaabbb*?

# LEX - analiza lexicala

Caractere operator

" \ [ ] ^ - ? . \* + | ( ) \$ / { } % < >

Folosirea lor drept caractere text: precedate de \ sau intre "".

xyz" ++"

"xyz ++"

xyz \ + \ +



# Expresii regulate in LEX

1. Clase de caractere  $[a - z0 - 9 <>]$ ,  $[-0 - 9]$ ,  $^abc$
2. Caracter arbitrar  $.$
3. Element optional  $ab?c$
4. Repetitii  $a^*$ ,  $a^+$ ,  $[a - z]^+$
5. Alternare  $ab|cd$   $a(b|d)$
6. Doar la inceput/final de linie  $^abc$ ,  $abc\$$
7. Context  $ab/cd$  ( $ab$  daca e urmat de  $cd$ )
8. Operator  $\{\}$ :  $a\{1, 5\}$ ,  $a\{2, \}$

# Analizor lexical controlat prin automat finit 3.6.2

Fie:

D [0-9]

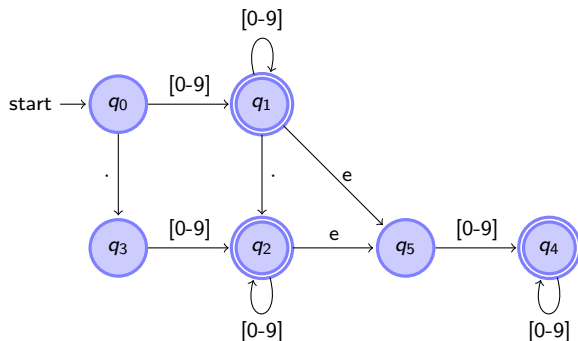
%%

{D}+

({D}\*|{D}\*\.{D}+|{D}+\.{D}\*) (e{D}+)?

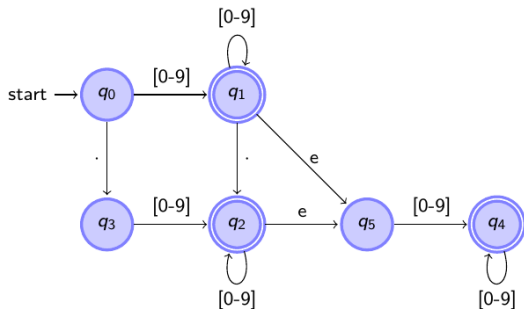
return ICON;

return FCON;



## Tabel de tranzitie

Stare curenta	caracter examinat inainte (lookahead) caracter de intrare		actiune la acceptare
	.	0-9 e	
0	3	1 -	-
1	2	1 5	return ICON;
2	-	2 5	return FCON;
3	-	2 -	-
4	-	4 -	return FCON;
5	-	4 -	-



## Algoritmul utilizat de LEX (greedy) 3.6.3

```
stare_curenta = 0;
stare_acceptoare_vazuta_anterior = nimic_vazut;
if (caracter lookahead este end_of_input)
    return 0;
while(caracter lookahead nu este end_of_input){
    if (exista tranzitie din starea curenta cu caracterul
        lookahead curent){
        stare_curenta = acea stare;
        avanseaza in intrare;
        if (starea curenta este o stare acceptoare){
            memoreaza pozitia curenta in intrare
            si actiunea asociata starii curente;
        }
    }
    else{
        if (nu a fost vazuta nicio stare acceptoare){
            exista o eroare:
                descarca lexemul curent si caracterul de intrare
                curent;
                stare_curenta = 0;
        }
        else {
            salveaza intrarea in pozitia in care se afla cand a
            vazut ultima stare acceptoare; realizeaza actiunea
            asociate acelei stari acceptoare;
        }
    }
}
```

Exemplu:  $stare\_urmatoare = array[stare\_curenta][intrare]$

Stare curenta	caracter examinat inainte (lookahead) caracter de intrare		actiune la acceptare
	.	0-9 e	
0	3	1 -	-
1	2	1 5	return ICON;
2	-	2 5	return FCON;
3	-	2 -	-
4	-	4 -	return FCON;
5	-	4 -	-

Stare	Intrare	Ultima stare acceptoare	actiune	pozitie in intrare
0	1.2e4			
1	.2e4	1	ICON	.
2	2e4	2	FCON	2
2	e4	2	FCON	e
5	4			
4		4	FCON	

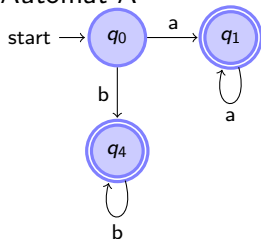
Exemplu:  $stare\_urmatoare = array[stare\_curenta][intrare]$

Stare curenta	caracter examinat inainte (lookahead) caracter de intrare		actiune la acceptare
	.	0-9 e	
0	3	1 -	-
1	2	1 5	return ICON;
2	-	2 5	return FCON;
3	-	2 -	-
4	-	4 -	return FCON;
5	-	4 -	-

Stare	Intrare	Ultima stare acceptoare	actiune	pozitie in intrare
0	1.2e			
1	.2e	1	ICON	.
2	2e	2	FCON	2
2	e	2	FCON	e
5				
	e			

## Gramatici regulate 2. Expresii regulate

Automat A



Automat B

