# TEHNICI DE COMPILARE

CURS 2

Gianina Georgescu

# CUPRINSUL CURSULUI 2

## ANALIZA LEXICALĂ

- Descrieri lexicale. Definiție, exemple
- Algoritm de transformarea AFN$\lambda - \mathrm{AFD}$
- Algoritm de transformarea ExprReg $- \mathrm{AFD}$
  (funcțiile *nullable*, *firstpos*, *lastpos*, *followpos*, exemple)
- Translator finit – definiție, exemple, proprietăți

# DESCRIERE LEXICALĂ

Descrierea lexicală a unui limbaj de programare poate fi formalizată de o expresie regulată de forma:

$$e=(e_1|e_2|\ldots|e_n)^*$$

unde $e_1,e_2,\ldots,e_n$ sunt expresii regulate, numite termeni, peste alfabetul T (mulțimea caracterelor admise de limbaj). Expresia *e* se numește descriere lexicală, iar $e_i$ desemnează o clasă particulară de token-i.

# DESCRIERI LEXICALE

- Spunem că un text (program sursă) este **corect din punct de vedere lexical** dacă aparține limbajului descris de expresia regulată e peste T, notat cu L(e)
- Definiție. Fie $e=(e_1|e_2|\ldots|e_n)^*$ o d.lex. peste T. O **interpretare lexicală** a unui șir $w \in L(e)$ este o secvență de forma:

$$(x_1,m_1),\ldots,(x_k,m_k),$$

unde $k \geq 1$, $w=x_1\ldots x_k$, $x_i \in L(e_{m_i})$, $1 \leq m_1,\ldots,m_k \leq n$

- Spunem că d. lex. e este **ambiguă** dacă în L(e) există w care are cel puțin două interpretări distincte. În caz contrar, spunem că e este **neambiguă**.

# EXEMPLU DE DESCRIERE LEXICALĂ

P_sursa = (identifier|intcon|comment|slash|spaces|
semicolon|equals)*

identifier = letter(letter|digit|_)*

intcon = digit digit*

comment= “/*”(notstar|“*”notslash)*“*/”

slash = “/”

spaces = (“ “)*

semicolon= “;”

letter = a|b|…|z|A|B|…|Z

digit = 0|1|2|…|9

L(notstar) = T* - {“*”}

L(notslash)=T* - {“/”}

(caracterele cuprinse între “ ” apar ca atare în text)

# EXEMPLU DE DESCRIERE LEXICALĂ

Fie șirul ‘xyz’. Acesta are 4 interpretări:

(“xyz”,identifier)

(“xy”,identifier) (“z”,identifier)

(“x”,identifier) (“yz”,identifier)

(“x”,identifier) (“y”,identifier) (“z”,identifier)

Care este interpretarea cea mai naturală?

# DESCRIERI LEXICALE

Definiție. Fie $e=(e_1|e_2|\dots|e_n)^*$ o descr. lexicală, $w \in L(e)$.

O interpretare $(x_1,m_1),\dots,(x_k,m_k)$ a lui w este orientată dreapta

dacă pentru orice $i=1,\dots,k$, $x_i$ este cel mai lung șir din
$L(e_1|e_2|\dots|e_n) \cap PREFIX(x_i \dots x_k)$

Exemplul 1. int x2; x2 = 123;
(“int”,identifier) (“ “,spaces) (“x2”, identifier)
(“;”,semicolon) (“ “,spaces) (“x2”, identifier)
(“=“,equals) (“123”,intcon) (“;”,semicolon)

Exemplul 2. $e=(\overset{1}{a}|\overset{2}{ab}|\overset{3}{bc})$, w=abc. Unica interpretare pt w este:
(a,1) (bc,2), care nu este orientată dreapta

# DESCRIERI LEXICALE

Definiție. Spunem că descrierea lexicală $e=(e_1|e_2|\ldots|e_n)^*$ este bine formată dacă orice șir w din L(e) are o unică interpretare orientată dreapta.

Observații

- Există un algoritm care decide dacă o descriere lexicală dată este bine formată
- Un analizor lexical (scanner) pentru o descriere lexicală bine formată $e=(e_1|e_2|\ldots|e_n)^*$ este un program care recunoaște L(e) și care pentru orice $w \in L(e)$ produce unica sa interpretare orientată dreapta

# TRANSFORMAREA UNEI EXPRESII REGULATE ÎN AFD – ALGORITM ÎN 2 PAȘI

Pasul 1. Transformăm expresia regulată în AFN$\lambda$ cu ajutorul algoritmului Thompson

Pasul 2. Transformăm AFN$\lambda$ în AFD echivalent

**Algoritm** de transformare AFN$\lambda$ în AFD

**Intrare** A=$(Q,\Sigma,\delta,s,F)$ AFN$\lambda$

**Ieșire** A'=$(Q',\Sigma,\delta',s',F')$ AFD cu L(A')=L(A)

Notatii: pentru $s \in Q$, $M \subseteq Q$, $a \in \Sigma$

$$\lambda\text{-closure}(s)=\{p \in Q \mid \exists\, s_0,s_1,\ldots,s_n \in Q,\ n \geq 0,\ s_0=s,\ s_n=p,\ s_i \in \delta(s_{i-1},\lambda),\ \forall\, i,\ 1 \leq i \leq n\} = \langle s \rangle$$

$$\lambda\text{-closure}(M) = \bigcup_{s \in M} \langle s \rangle = \langle M \rangle$$

$$\text{move}(M,a) = \{p \in Q \mid \exists\, s \in M,\ p \in \delta(s,a)\}$$

**ALGORITMUL**

$Q' \leftarrow \{<s>\}$, $<s>$ stare nemarcata

while ( exista $M \in Q'$ stare nemarcata)

{

marcheaza M;

for ($a \in \Sigma$)

{

$V \leftarrow <move(M,a)>$;

if ($V \notin Q'$) adauga V la Q' ca stare nemarcata;

$\delta'(M,a) \leftarrow V$;

}// end_for

}// end_while

$Q' \subseteq 2^{Q}$, $s' = <s>$, $F' = \{M \in Q' \mid M \cap F \neq \phi\}$

**ALGORITMUL** pentru <M>, $M \subseteq Q$

```
for ( fiecare p∈M)
            push (p) // se foloseste o stiva
while (stiva nu este vida)
    {
       pop p;
       for (u∈Q, u∈δ(p,λ))
           if (u ∉ <M>)
               {
                 <M> ← <M> U {u};
                 push u;
               } // end_if
    }// end_while
```

## CONSTRUCTIA DIRECTA A UNUI AFD PORNIND DE LA O EXPRESIE REGULATA

Fie $r$ expresie regulata peste $\Sigma$ si $\# \notin \Sigma$.

Numim extensia lui $r$ expresia $(r)\#$.

Vom construi un AFD care recunoaste $r$. Pentru aceasta construim arborele sintactic $T$ asociat lui $(r)\#$, apoi printr-o parcurgere in adancime a lui $T$ calculam 4 functii: *nullable*, *firstpos*, *lastpos*, *followpos*.

- Numerotam frunzele lui $T$ de la stanga la dreapta, cu numere de la 1 până la $n$ (n este asociat lui #). Aceste numere se numesc <u>poziții</u>.

- Nodurile interne ale lui $T$ sunt etichetate cu $|, \cdot$ sau $*$, iar cele externe sunt notate cu simboluri din $\Sigma \cup \{\lambda, \#\}$.

- Pentru un nod $n$ al lui $T$, care este radacina unui subarbore $T_1$, al lui $T$, calculam:

- $nullable(n) = true$ dacă și numai dacă $\lambda$ apartine limbajului descris de subexpresia regulata ce corespunde lui $T_1$, pe care o notam cu $r_1$
- $firstpos(n)$ reprezinta primele pozitii ce corespund primelor simboluri ale șirurilor din $L(r_1)$
- $lastpos(n)$ reprezinta ultimele pozitii ce corespund ultimelor simboluri ale șirurilor din $L(r_1)$
- $followpos(p)$, unde $p$ reprezinta o pozitie, $1 \leq p \leq n$, cuprinde toate pozitiile $q$, unde $1 \leq q \leq n$ astfel că în $L((r)\#)$ există un șir $w \in \Sigma^*$ de forma $w = w'a_1a_2w''$, $a_1, a_2 \in \Sigma$. Poziția corespunzătoare lui $a_1$ este $p$, iar a lui $a_2$ este $q$.

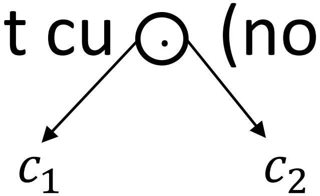
# Calculul pentru *nullable*, *firstpos*, *lastpos*

| Nodul $n$ | $nullable(n)$ | $firstpos(n)$ | $lastpos(n)$ |
|---|---|---|---|
| $n$ este frunză etichetată cu $\lambda$ | true | $\emptyset$ | $\emptyset$ |
| $n = a \in \Sigma$, frunza având asociată poziția $i$ | false | $\{i\}$ | $\{i\}$ |
| $n \mid$ <br> $\swarrow \searrow$ <br> $c_1 \quad c_2$ | $nullable(c_1)$ or $nullable(c_2)$ | $firstpos(c_1) \cup firstpos(c_2)$ | $lastpos(c_1) \cup lastpos(c_2)$ |
| $n \cdot$ <br> $\swarrow \searrow$ <br> $c_1 \quad c_2$ | $nullable(c_1)$ and $nullable(c_2)$ | if $(nullable(c_1))$ <br> $firstpos(c_1) \cup firstpos(c_2)$ <br> else <br> $firstpos(c_1)$ | if $(nullable(c_2))$ <br> $lastpos(c_1) \cup lastpos(c_2)$ <br> else <br> $lastpos(c_2)$ |
| $n \star$ <br> $\downarrow$ <br> $c$ | true | $firstpos(c)$ | $lastpos(c)$ |

# CALCULUL FUNCȚIEI *firstpos*

Funcția *firstpos* se calculează simultan cu funcțiile *firstpos, lastpos, nullable* prin parcurgerea în adâncime a arborelui sintactic $T$, doar pentru nodurile concatenare și iterație Kleene:

- Pentru nodul $n$ etichetat cu $\odot$ (nod concatenare cu descendenții $c_1, c_2$):




$$\forall i \in lastpos(c_1), followpos(i) \supseteq firstpos(c_2)$$

- Pentru un nod $n$ etichetat cu $*$ (iterație Kleene), având ca unic descendent pe $c$:

$$\forall i \in lastpos(c), followpos(i) \supseteq firstpos(c)$$

# Algoritmul ExpReg $\Rightarrow$ AFD

**Intrare:** $(r)\#$, unde $r$ expresie regulata peste $\Sigma$

**Ieșire:** AFD $A = (Q, \Sigma, \delta, s, F), L(A) = L(r)$

**Metoda:**

1. Se construieste arborele sintactic $T$ pentru $(r)\#$;
2. Se calculează funcțiile *nullable*, *firstpos*, *lastpos*, *followpos* ca în tabelul de mai sus, prin parcurgerea în adâncime a lui $T$.
3. $Q \subseteq 2^{\{1,2,\ldots,n\}}$, unde $1,2,\ldots,n$ reprezinta pozițiile asociate frunzelor lui $T$. Stările lui $A$ sunt mulțimi de poziții.

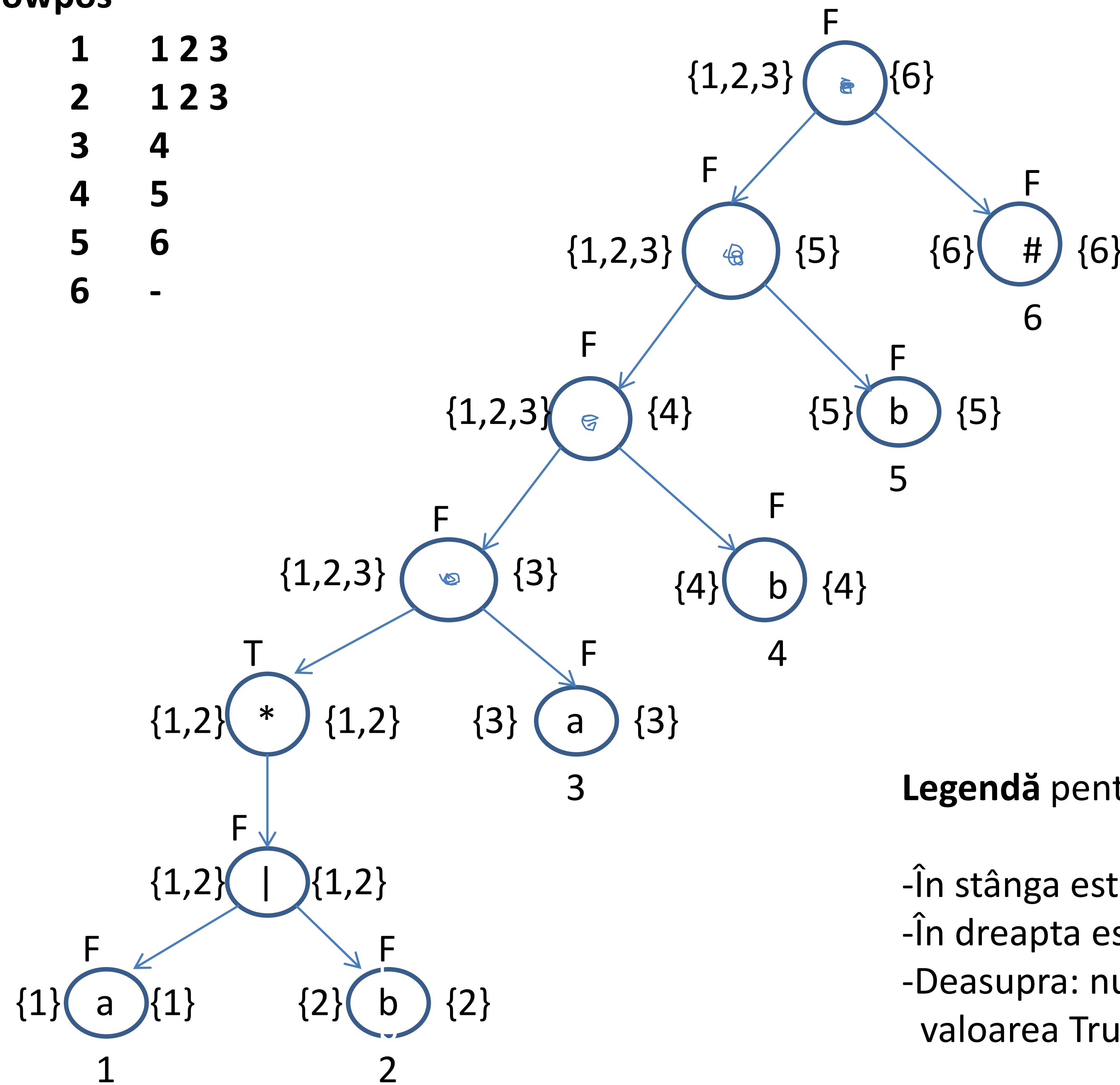
# Algoritmul ExpReg ⇒ AFD

{
$s \leftarrow firstpos(rad)$;  // *rad* este radacina lui $T$
$Q \leftarrow \{s\}$, $s$ stare nemarcata;
while (exista $M \in Q$ nemarcata)
  {
    marcheaza $M$;
    for ($a \in \Sigma$){
      $V \leftarrow \{q | \exists p \in M, p$ pozitie ce corespunde lui
        $a, q \in followpos(p)\}$;
      if($V \notin Q$) $Q \leftarrow Q \cup \{V\}$;
      $\delta(M, a) \leftarrow V$;
    } \\end_for
  } \\end_while
$F \leftarrow \{M \in Q | M$ contine pozitia lui #$\}$;
}

# Exemplu pentru expresia (a|b)*abb, cu extensia ((a|b)*abb)#

**followpos**

| | |
|---|---|
| **1** | **1 2 3** |
| **2** | **1 2 3** |
| **3** | **4** |
| **4** | **5** |
| **5** | **6** |
| **6** | **-** |

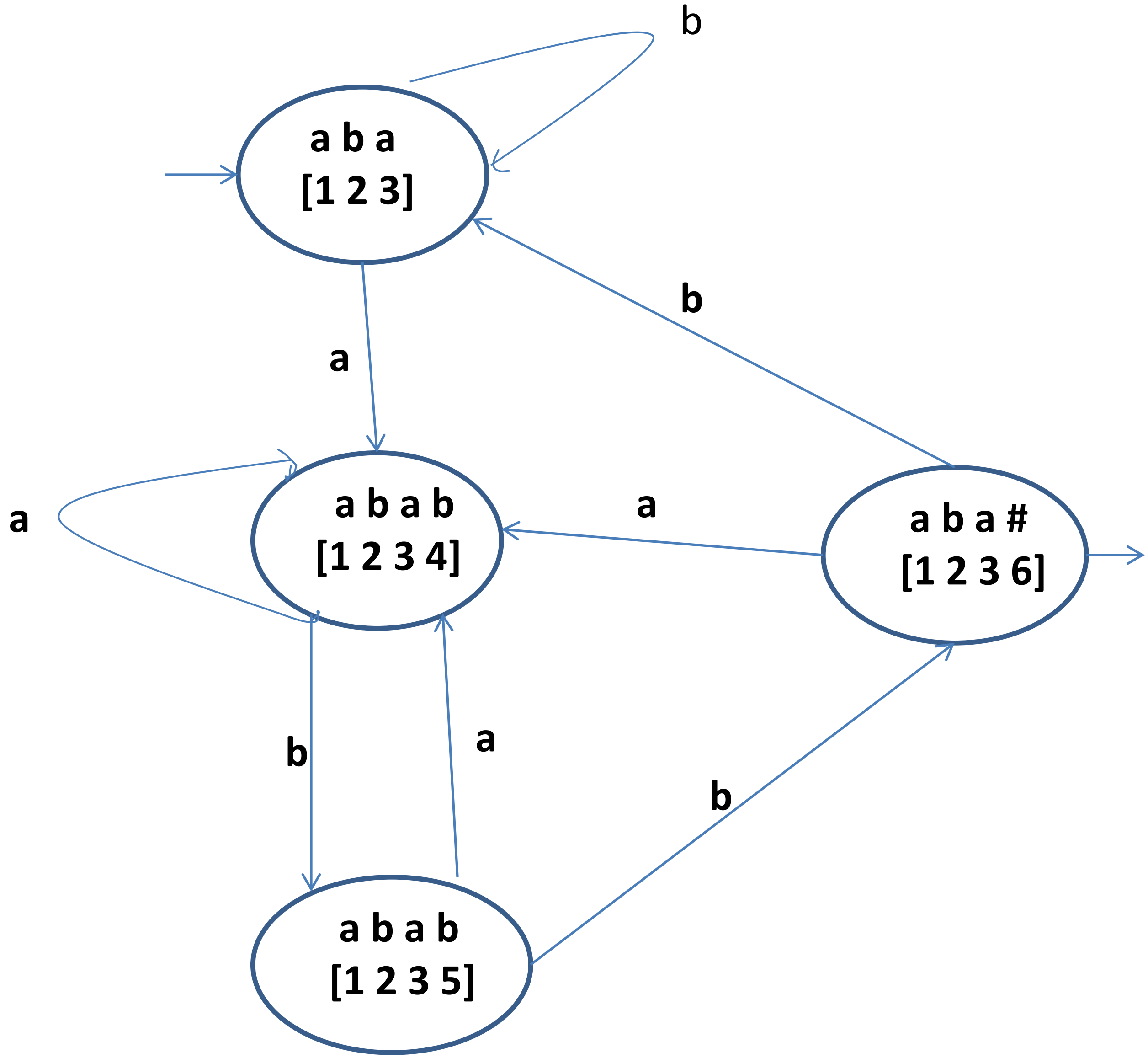



**Legendă** pentru un nod n:

-În stânga este figurat firstpos(n)
-În dreapta este figurat lastpos(n)
-Deasupra: nullable(n) cu valoarea True sau False

**followpos**

| | |
|---|---|
| **1** | **1 2 3** |
| **2** | **1 2 3** |
| **3** | **4** |
| **4** | **5** |
| **5** | **6** |
| **6** | **-** |




**AFD echivalent cu (a|b)*abb**
**(nu este neaparat minimal)**

# TRANSLATOARE FINITE

Sunt ca și automatele finite, cu deosebirea că la fiecare tranziție produc o ieșire. Formal:

**Definiție.** Un translator finit nedeterminist cu $\lambda$-tranziții este o structură de forma:

$T = (Q, V_i, V_e, \delta, s, F)$, unde:

- $Q$ este mulțimea stărilor
- $V_i$ este alfabetul de intrare
- $V_e$ este alfabetul de ieșire
- $\delta: Q \times (V_i \cup \{\lambda\}) \to \mathcal{P}_{fin}(Q \times V_e^*)$

  ↑ mulțimea părților finite ale lui $(Q \times V_e^*)$
- $s \in Q$ este starea initiala a lui $T$
- $F \subseteq Q$ mulțimea stărilor finale

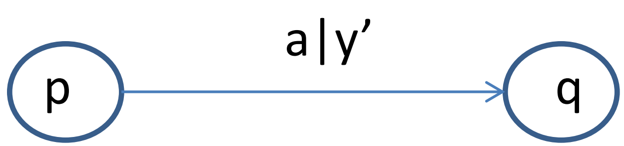
# TRANSLATOARE FINITE

**Descriere instantanee** (configurație, instanță) a lui $T$: triplet de forma $(q, x, y)$, unde:

- $q \in Q$ starea curentă a translatorului
- $x \in V_i^*$ șirul curent scanat din intrare
- $y \in V_e^*$ șirul de ieșire curent

**Mișcare a lui $T$**:

- $(p, ax, y) \vdash (q, x, yy')$ ddacă $(q, y') \in \delta(p, a)$, $p, q \in Q,\ a \in V_i \cup \{\lambda\},\ x \in V_i^*,\ y, y' \in V_e^*$




**Închiderea reflexivă și tranzitivă** a relației $\vdash$ este notată cu $\vdash^*$ (reprezintă 0 sau mai multe mișcări)

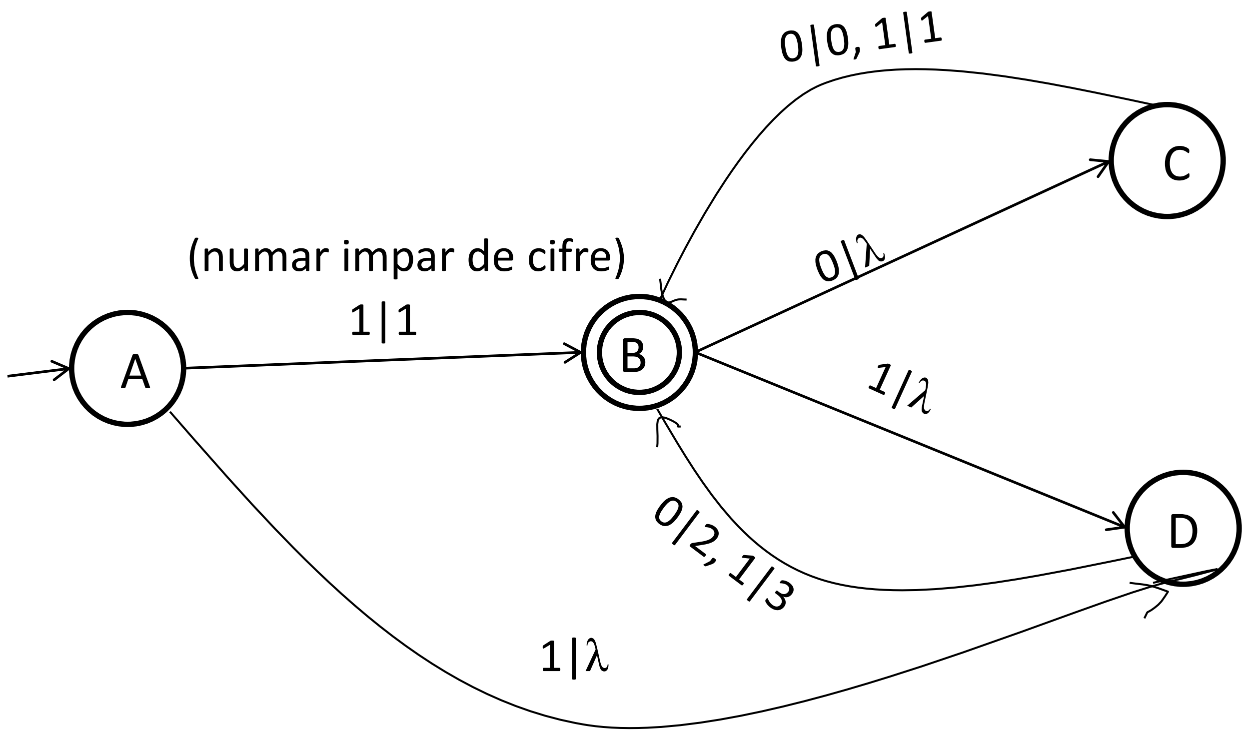
# TRANSLATAREA DEFINITĂ DE TRANSLATORUL *T*

- Pentru un șir $x \in V_i^*$:
  $T(x) = \{ y \in V_e^* | (s, x, \lambda) \vdash^* (q, \lambda, y), q \in F \}$
- Pentru un limbaj $L \subseteq V_i^*$:
  $$T(L) = \bigcup_{x \in L} T(x)$$
- Translatarea definita de $T$ la modul global:
  $\tau(T) = \{(x, y) | x \in V_i^*, y \in V_e^*, y \in T(x)\}$

# EXEMPLU

Translator care translatează orice șir $w \in \{0,1\}^*$ care începe cu '1' în șirul echivalent din baza 4.
Astfel: $100110010 \to 10302$, $101100 \to 230$

**T=({A,B,C,D}, {0,1}, {0,1,2,3},δ,{B})**




(ramura pentru siruri cu numar par de cifre)

(A,101100,λ)→
(B,01100, 1)→
(C,1100,1) →
(B,100,11) →
(C,00,11) →
(B,0,112) →
(C, λ,112),
C nefinala

(A,101100,λ)→
(D,01100, λ)→
(B,1100,2) →
(D,100,2) →
(B,00,23) →
(C,0,23) →
(B, λ,230)
B finala

T(101100)={230}

# PROPRIETĂȚI ALE TRANSLATOARELOR FINITE

**Propoziția 1.** Familia limbajelor regulate, $\mathcal{L}_{REG}$, este închisă la translatări finite.
Aceasta înseamnă că dacă $L \in \mathcal{L}_{REG}$ și $T$ este un translator finit, atunci $T(L) \in \mathcal{L}_{REG}$, unde $L \subseteq \Sigma^*$ iar alfabetul de intrare al lui $T$ este $\Sigma$.

**Propoziția 2.** Familia limbajelor independente de context, $\mathcal{L}_{CF}$, este închisă la translatări finite.
Aceasta înseamnă că dacă $L \in \mathcal{L}_{CF}$ și $T$ este un translator finit, atunci $T(L) \in \mathcal{L}_{CF}$, unde $L \subseteq \Sigma^*$ iar alfabetul de intrare al lui $T$ este $\Sigma$.