FUNDAMENTELE PROIECTĂRII COMPILATOARELOR

CURS 2

Gianina Georgescu

CUPRINSUL CURSULUI 2

ANALIZA LEXICALĂ

- Descrieri lexicale. Definiție, exemple
- Algoritmul *shunting yard*
- Algoritmul Thompson: ExpReg $\rightarrow AFN\lambda$
- Algoritm de transformare $AFN\lambda \rightarrow AFD$
- Analiza lexicală - exemple

ANALIZA LEXICALĂ
idei de bază

- Lexicul unui limbaj de programare: descris cu ajutorul expresiilor regulate
- Scanner-ul (analizorul lexical) este un program inplementat 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.)

DESCRIERE LEXICALĂ

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

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

unde $e_1, e_2, \dots e_n$ 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 $w \epsilon L(e)$ este o secvență de forma:

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

$$k \geq 1, w = x_1 \cdots x_k, x_i \epsilon L\left(e_{m_i}\right), 1 \leq m_1, \cdots, m_k \leq n$$

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

O interpretare $(x_1, m_1), \ldots, (x_k, m_k)$ a lui w se numește **orientată dreapta** dacă pentru orice $i = 1, \cdots, k,\ x_i$ este cel mai lung șir din $L(e_1|e_2|\ldots|e_n) \cap PREFIX(x_i \ldots x_k)$

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

1 2 3

Exemplul 2. $e = (a|ab|bc)^*, w = abc$. 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 $w \epsilon L(e)$ 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 AFNλ în AFD echivalent

Conversia unei expresii regulate în forma postfixată cu algoritmul Shunting-Yard (triajul dintr-o gară), folosit de alg Thompson

Algoritmul shunting yard (triajul din gară) a fost inventat de Edsger Dijkstra pentru a converti o expresie scrisă în formă infixată în forma postfixată.

- Folosim o **coadă pentru** ieșire și o **stivă pentru operatori**
- Dacă simbolul curent din intrare este o **literă sau λ**… îl adăugăm în **coada de ieșire.**
- Dacă simbolul curent din intrare este un **operator**… dacă există deja un operator în vârful **stivei de operatori** cu o precedență mai mare sau egală decât a simbolului din intrare, ștergem operatorul din vârful stivei de operatori și îl adăugăm în **coada de ieșire**. Repetăm aceasta până când: simbolul curent are precedența mai mare decât operatorul din vârful stivei; or simbolul din vf stivei este "("; or stiva este vidă, cazuri în care îl punem pe stivă.
- Dacă simbolul curent din intrare este un **operator ȘI** există o paranteză stângă în vârful stivei … introducem acel operator în vârful stivei.
- Dacă simbolul curent din intrare este '(' … o adăugăm în stiva de operatori
- Dacă simbolul curent din intrare este ')' … extragem toți operatorii din stivă și îi adăugăm în coada de ieșire până întâlnim '('. Apoi ștergem ambele paranteze și continuăm algoritmul.
- La final, scoatem în ieșire toți operatorii care au rămas pe stivă

while (există $token$ de citit)
{
read token_curent;
if (token_curent este)
- o *literă* sau λ:
introducem în coadă (ieșire)
- un *operator* o_1:
while (în vârful stivei există un operator $o_2 \neq$ ”(” **AND** (o_2 are precedența mai mare decat o_1) **OR** (o_2, o_1 au precedență egală și o_1 asociativ la stânga)):
pop o_2 din stiva de operatori și introdu o_2 în coadă (ieșire)
push o_1 în stiva de operatori
- o *paranteză stângă* "(":
push paranteza în stiva de operatori
- o *paranteză dreaptă* ")":
while (stiva este nevidă **AND** operatorul din vârful stivei ≠"("):
/* Dacă stiva devine vidă înseamnă că nu există potrivire de paranteză; exit */
pop operatorul din stivă **and push** în coadă (output)
{**presupunem** că există "(" în vârful stivei}
pop "(" din stivă /*o ștergem */
}/* while exista *token* */

/* După ciclul while, pop restul token-ilor din stiva de operatori și îi introducem în coadă (output). */

while (există operatori în stiva de operatori):

/* Dacă în vârful stivei este o paranteză, atunci există o nepotrivire de paranteze. */

{**presupunem** că operatorul din vârful stivei nu este o paranteză stângă}

pop operatorul din stivă **and push** în coadă (output)

Acest algoritm are complexitatea O(n), n fiind lungimea intrarii.

Exemplu. Aplicăm algoritmul pentru expresia $\boldsymbol{a}(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{b}$

Vom adăuga operatorul concatenare în mod explicit: $\boldsymbol{a} \cdot (\boldsymbol{a}|\boldsymbol{b})^* \cdot \boldsymbol{b}$

Precedența operatorilor		Asociativitatea
()	4	
*	3	
·	2	stângă
|	1	stângă

Token	Acțiune	Stivă de operatori cu vîrful la stânga	Output
a	Output a	Ø	a
·	Push token	·	a
(	Push token	(·	a
a	Output a	(·	aa
|	Push token	|(·	aa
b	Output b	|(·	aab
)	Pop stack (|) to output Pop stack	(· ·	aab|
*	Push token (* prioritar față de ·)	*·	aab|
·	Pop *, Output * (· prioritate ≤ *) Pop ·, Output · (·=·) Push token	· Ø ·	aab|* aab|*·
b	Output b	·	aab|*·b
	Output stack	Ø	aab|*·b·

ALGORITMUL THOMPSON

Este un algoritm datorat lui Ken Thompson prin care se obține un AFN_λ (vom nota șirul vid prin λ) care recunoaste limbajul descris de o expresie regulată dată.

Algoritmul se aplică recursiv prin impartirea unei expresii în subexpresiile sale constituente, din care se poate construi AFN-ul folosind un set de reguli. Mai precis, de la o expresie regulată E, automatul obținut A cu ajutorul funcției de tranziție δ respectă următoarele proprietăți:

- A are exact o stare inițială q_0, care nu este accesibilă din nicio altă stare. Adică, pentru orice stare q și orice simbol a, $\delta(q, a)$ nu conține q_0.
- A are exact o stare finală q_f, din care nu se poate ajunge în nicio altă stare. Adică, pentru orice simbol $a, \delta\ (q_f\ , a\ )\ =\ \emptyset$

REGULI

Următoarele reguli sunt descrise potrivit Aho et al. (1986)

Expresia vidă λ este convertită la

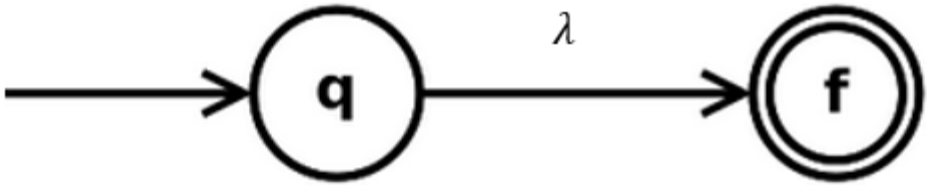


FIGURA 1

Un **simbol** *a* din alfabetul de intrare Σ este convertit la

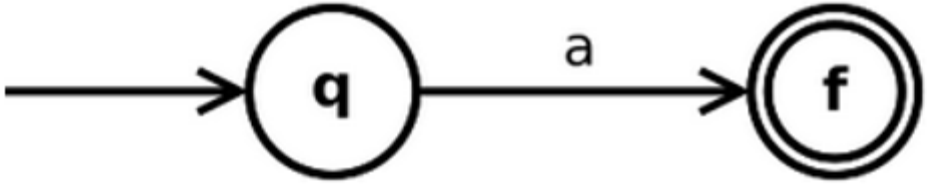


FIGURA 2

Expresia reuniune (alternanță) $s|t$ este convertită la

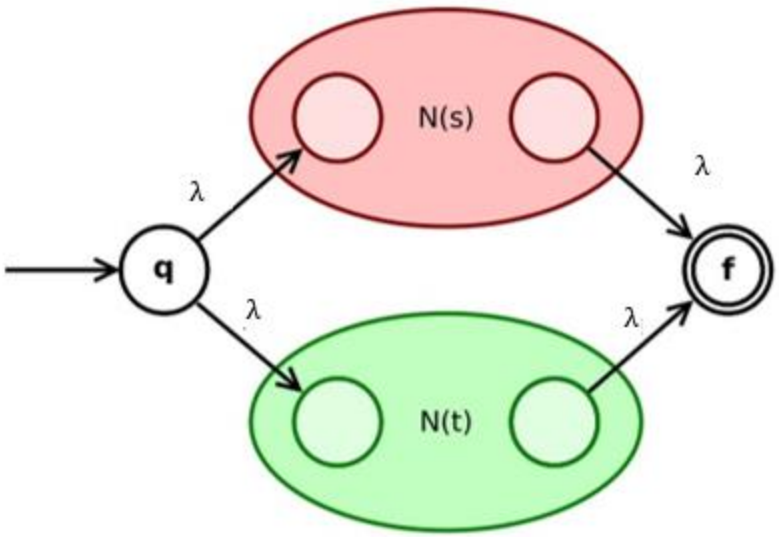


FIGURA 3

Expresia concatenare *st* este convertită la

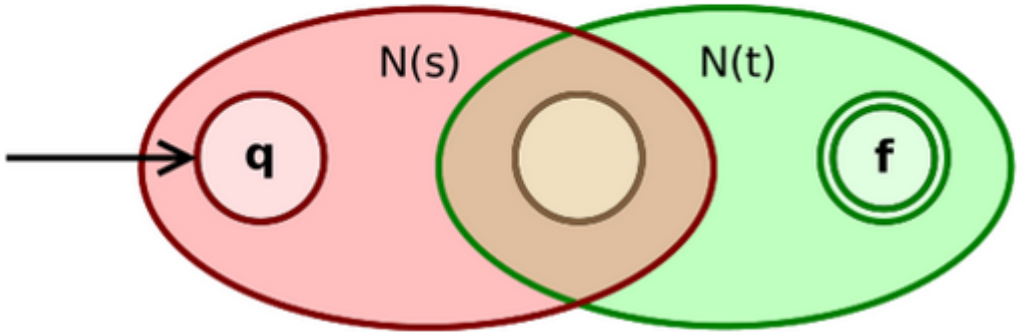


FIGURA 4

Starea inițială a lui $N(s)$ este starea inițială a întregului AFN. Starea finală a lui $N(s)$ devine starea inițială a lui $N(t)$. Starea finală a lui $N(t)$ este starea finală a întregului AFN.

Expresia inchidere Kleene s^* este convertită la

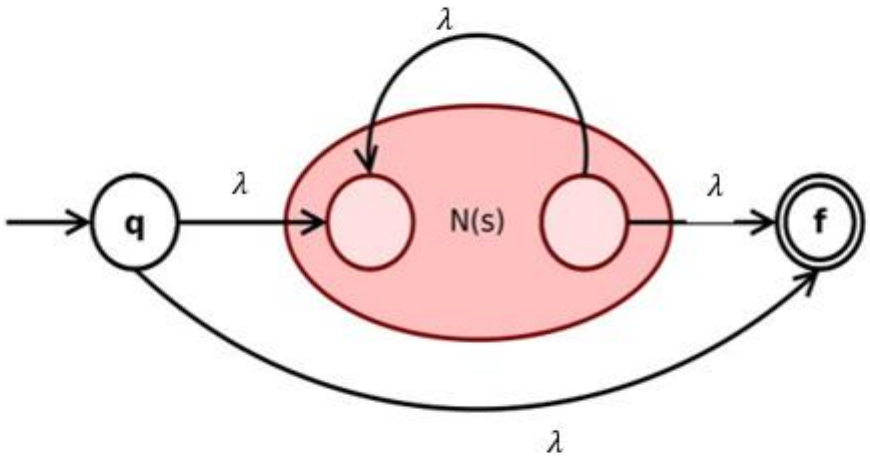


FIGURA 5

ALGORITM DE TRANSFORMARE A UNEI EXPRESII REGULATE ÎN AFN_λ ECHIVALENT

INPUT: o expresie regulată r în forma postfixată
OUTPUT: un AFN_λ A astfel încât $L(A) = L(r)$

while (există *token* de citit)
{
read *token_curent*;
if (*token_curent* este)
- λ: introducem pe stiva automatul ca in Fig 1
- o litera a: introducem pe stiva automatul ca in Fig 2
- un operator o:
  if (o este):
  - '|': scoate de pe stiva A_1, A_2 si pune pe stiva automatul ca in Fig 3
  - '·': scoate de pe stiva A_1, A_2 si pune pe stiva automatul ca in Fig 4 in ordinea A_2, A_1
  - '*': scoate de pe stiva A si pune pe stiva automatul ca in Fig 5

} /* while exista *token* */
Automatul (unic, dacă expresia în forma postfixată este corectă) care rămâne pe stivă este cel cerut

TRANSFORMAREA UNEI EXPRESII REGULATE ÎN AFD – ALGORITM ÎN 2 PAȘI

Pasul 2. Transformăm AFNλ în AFD echivalent

Algoritm de transformare AFNλ în AFD

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

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

$$Q' \subseteq 2^Q$$

TRANSFORMAREA UNEI EXPRESII REGULATE ÎN AFD – ALGORITM ÎN 2 PAȘI

Notații pentru $p \epsilon Q, M \subseteq Q$:

$$\lambda - closure(p) = \{q \epsilon Q | \exists\, p_0, p_1, \ldots, p_n \epsilon Q, n \geq 0, p_0 = p, p_n = q, p_i \epsilon \delta(p_{i-1}, \lambda), 1 \leq i \leq n\} = < p >$$

$$\lambda - closure(M) = U_{s \epsilon M} < s > = < M >$$

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

Proprietăți pentru $p \epsilon Q, M \subseteq Q$:

$$\ll p \gg = < p >$$

$$\ll M \gg = < M >$$

$$p \in < p >$$

$$M \subseteq < M >$$

$$< \emptyset > = \emptyset$$

ALGORITMUL

$Q' \leftarrow \{< s >\}, < s >$ stare nemarcata // s starea inițială a AFN_λ

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

{

marcheaza M;

for $(ac\Sigma)$

{

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

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

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

}// end_for

}// end_while

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

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

$< M > \leftarrow M$

for (fiecare $p \in M$)

push (p) // se foloseste o stiva

while (stiva nu este vida)

{

pop p;

for ($u \in Q$, $u \in \delta(p,\lambda)$)

if ($u \notin <M>$)

{

$<M> \leftarrow <M> \cup \{u\}$;

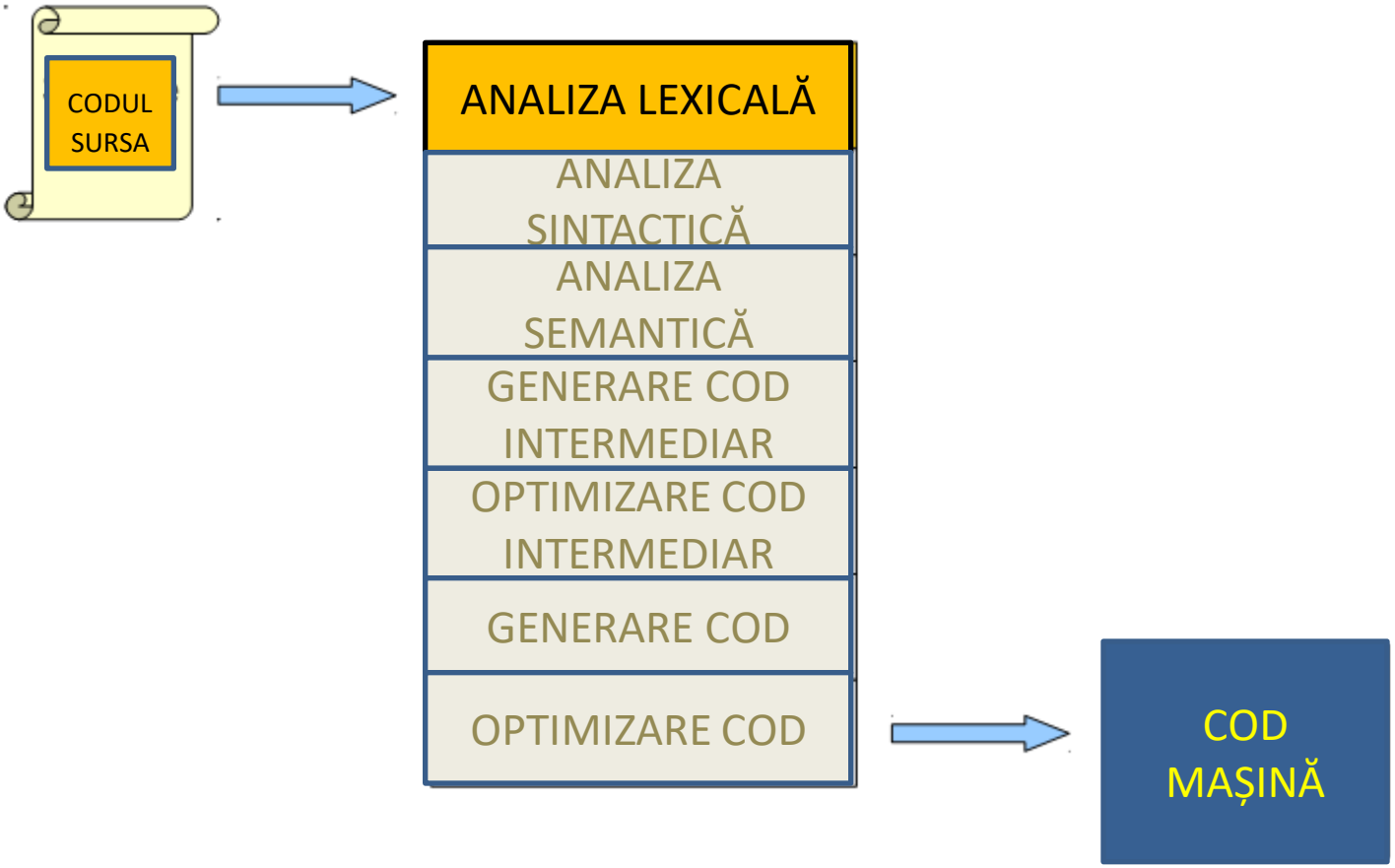
push u;

} // end_if

}// end_while

ANALIZA LEXICALĂ
- Exemple -

UNDE SUNTEM


CODUL SURSA
ANALIZA LEXICALĂ
ANALIZA SINTACTICĂ
ANALIZA SEMANTICĂ
GENERARE COD INTERMEDIAR
OPTIMIZARE COD INTERMEDIAR
GENERARE COD
OPTIMIZARE COD
COD MAȘINĂ


```
while (ip < z)
    ++ip;
```

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

T_While

SCANAREA FIȘIERULUI SURSĂ

PORȚIUNEA DIN FIȘIERUL SURSĂ DIN CARE PROVINE TOKEN-UL SE NUMEȘTE LEXEMĂ

T_While

ACESTA ESTE UN TOKEN. PUTEȚI SĂ VI-L REPREZENTAȚI CA PE UN TIP ENUMERARE CE REPREZINTĂ O ENTITATE LOGICĂ PE CARE O CITIM DIN FIȘIERUL SURSĂ

SCANAREA FIȘIERULUI SURSĂ

w	h	i	l	e		(	1	3	7		<		i	)	\n	\t	+	+	i	;

T_While

SCANAREA FIȘIERULUI SURSĂ

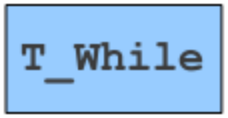
T_While

SCANAREA FIȘIERULUI SURSĂ

T_While

UNEORI IGNORĂM O LEXEMĂ, DECI NU O STOCĂM. AICI IGNORĂM WHITWSPWCE, DEOARECE NU ARE NICIO SEMNIFICAȚIE PENTRU ANALIZA SINTACTICĂ

SCANAREA FIȘIERULUI SURSĂ



SCANAREA FIȘIERULUI SURSĂ

T_While

SCANAREA FIȘIERULUI SURSĂ

T_While

SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

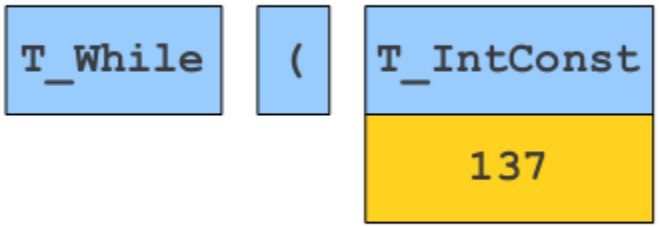
SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

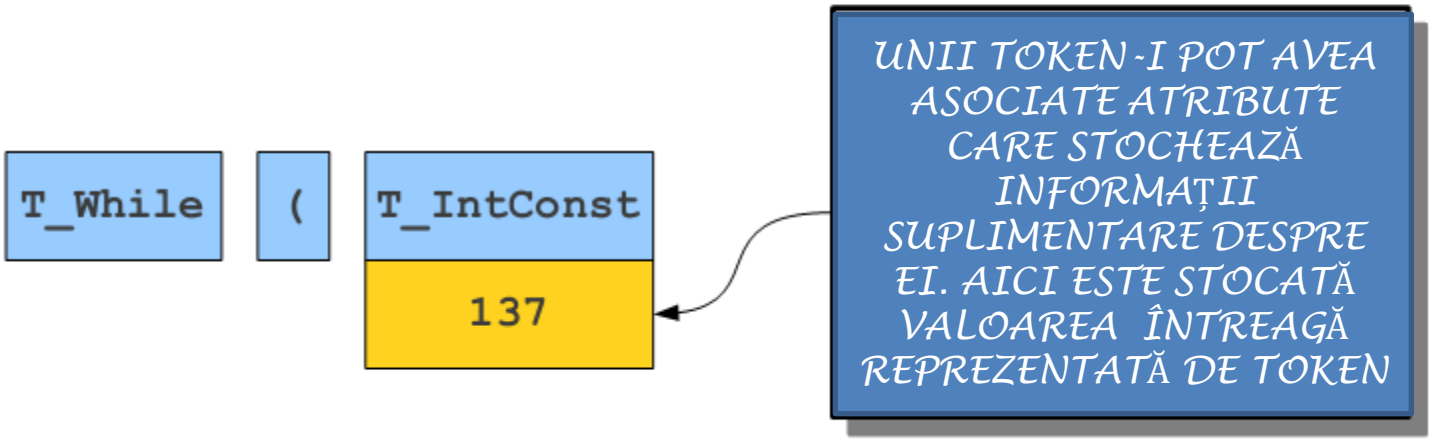
SCANAREA FIȘIERULUI SURSĂ

SCANAREA FIȘIERULUI SURSĂ

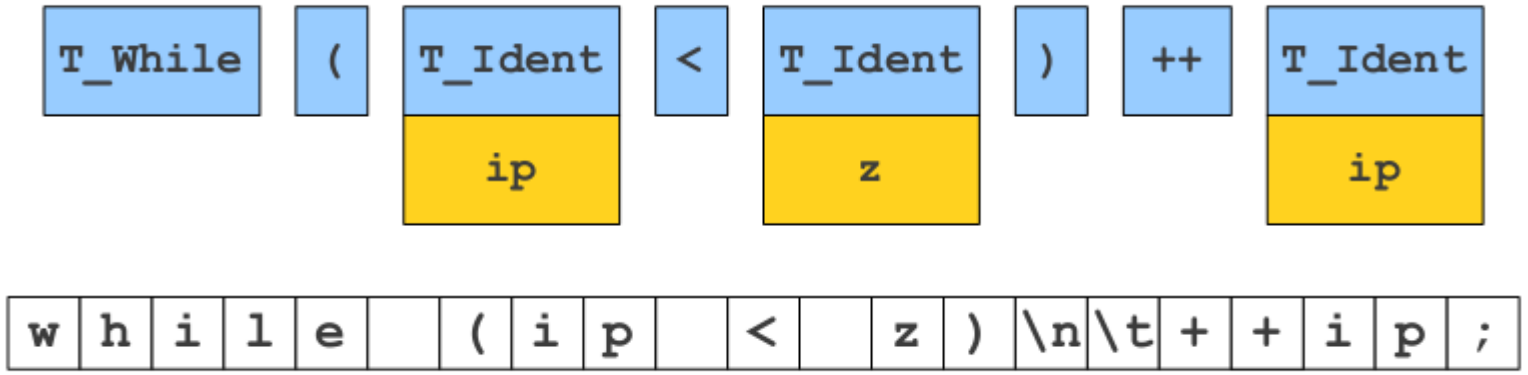
SCANAREA FIȘIERULUI SURSĂ



SCANAREA FIȘIERULUI SURSĂ



… rezultatul scanării va fi




```
while (ip < z)
    ++ip;
```

SCANAREA POATE FI DIFICILĂ

ÎN FORTRAN SPAȚIILE SUNT IRELEVANTE

```
DO 5 I = 1,25
DO 5 I = 1.25
```

```
DO 5 I = 1,25
DO5I   = 1.25
```

SCANAREA POATE FI DIFICILĂ

ÎN PL1 CUVINTELE CHEIE POT FI FOLOSITE CA IDENTIFICATORI

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

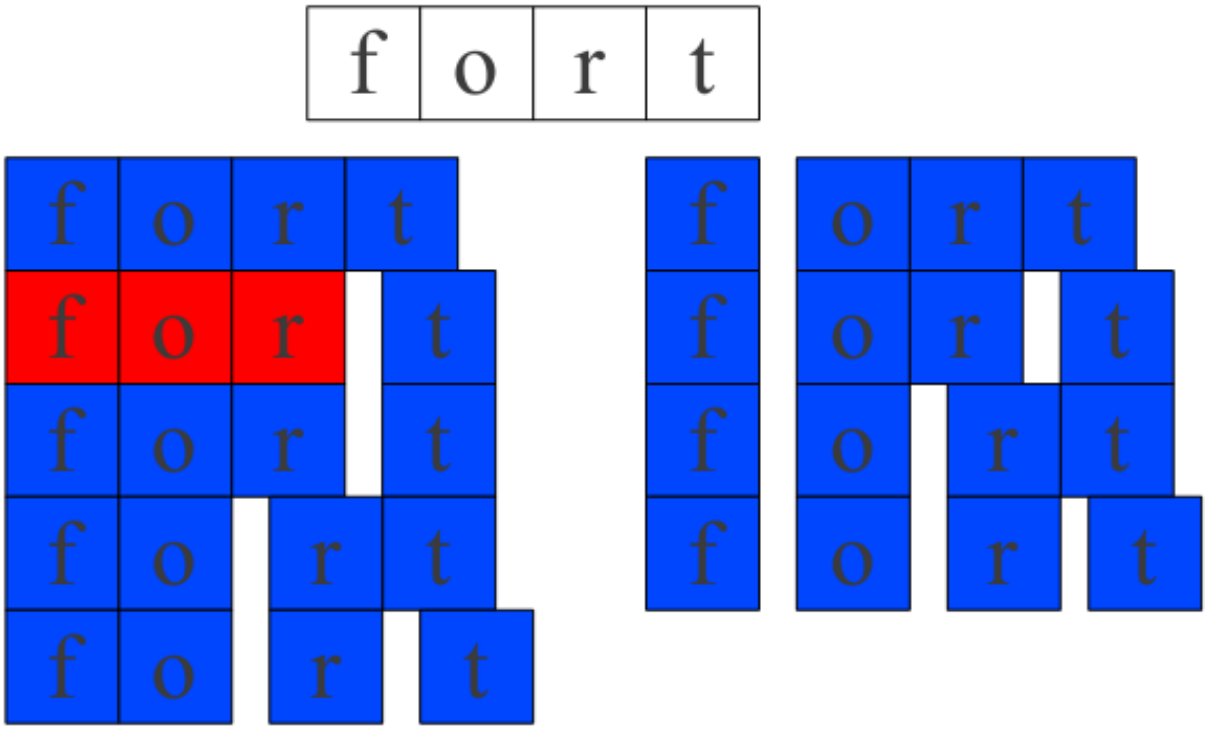
DIN NOU DESPRE EXPRESII REGULATE

Expresiile regulate sunt moduri de a descrie un limbaj (un limbaj regulat).

Adesea expresiile regulate ajută la descrierea unui limbaj într-un mod compact și apropiat de limbajul uman

Sunt utilizate ca bază pentru numeroase sisteme software, inclusiv de *flex* pe care îl vom folosi în acest curs.

REZOLVAREA AMBIGUITĂȚII PENTRU EXPRESII REGULATE



MAI SUS SUNT ILUSTRATE TOATE MODURILE DE ÎMPĂRȚIRE A ȘIRULUI $fort$ ÎN TOKEN-I. ÎN LINIA 2 for ESTE CUVÂNT CHEIE

REZOLVAREA AMBIGUITĂȚII PENTRU EXPRESII REGULATE

- Presupunem că toți token-ii sunt specificați de expresii regulate
- Pentru fiecare expresie regulată se implementează AFN corespondent cu λ-tranziții (λ este șirul vid), cu alg Thompson, de exemplu
- Scanarea se face de la stânga la dreapta
- Regula de departajare a token-ilor: deplasare spre dreapta cât de mult se poate
- Întotdeauna se reține cel mai lung prefix al textului rămas de scanat.

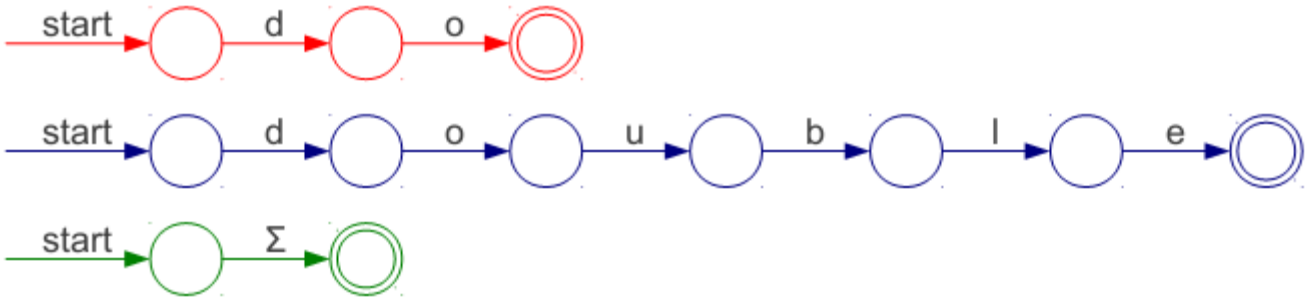
IMPLEMENTAREA DEPLASĂRII MAXIMALE SPRE DREAPTA CU AJUTORUL AFN

Ideea:

- Convertim expresiile în AFN cu λ-tranziții.
- Rulăm toate AFN-urile în paralel, păstrând ultima potrivire.
- Când toate automatele nu se mai pot mișca, raportează ultima potrivire și reîncepem căutarea din acel punct.

EXEMPLU PENTRU IMPLEMENTAREA DEPLASĂRII MAXIMALE SPRE DREAPTA CU AJUTORUL AFN PENTRU FORTRAN

```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

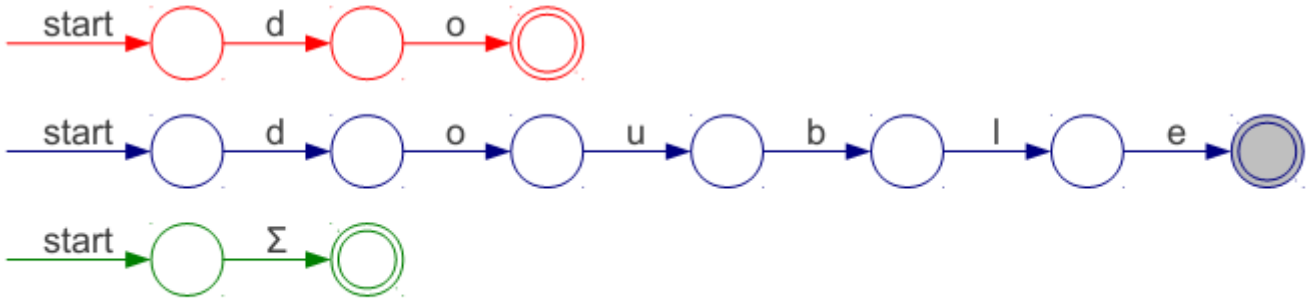


D	O	U	B	D	O	U	B	L	E

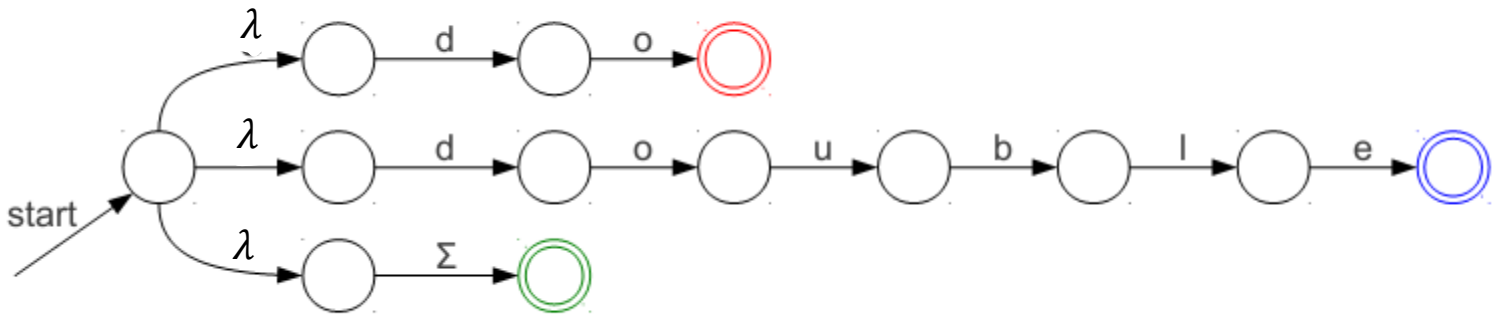
ÎN FORTRAN TOKEN-II NU SUNT NEPĂRAT DELIMITAȚI DE SPAȚII
Σ REPREZINTĂ ALFABETUL CARACTERELOR

EXEMPLU PENTRU IMPLEMENTAREA DEPLASĂRII MAXIMALE SPRE DREAPTA CU AJUTORUL AFN PENTRU FORTRAN

```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```

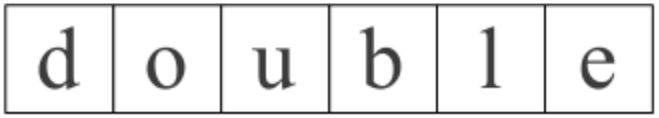


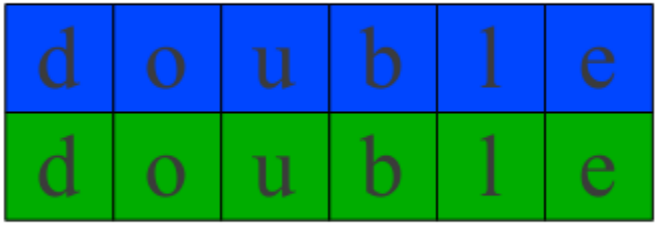
EXEMPLU PENTRU IMPLEMENTAREA DEPLASĂRII MAXIMALE SPRE DREAPTA CU AJUTORUL AFN PENTRU FORTRAN - O SIMPLIFICARE



ALTE TIPURI DE CONFLICTE

```
T_Do          do
T_Double      double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```





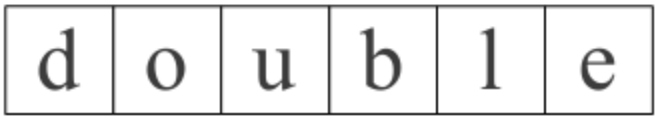
AICI *double* POATE FI ÎNȚELES CA ȘI CUVÂNT CHEIE SAU CA IDENTIFICATOR

ALTE TIPURI DE CONFLICTE – CUM SE REZOLVĂ

Când două expresii regulate se pot aplica pentru un același token, o alegem pe cea care are ”prioritatea” cea mai mare.
Sistem simplu de priorități: alege regula care este definită prima

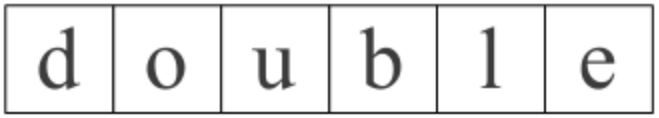
ALTE TIPURI DE CONFLICTE – CUM SE REZOLVĂ

```
T_Do          do
T_Double      double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```



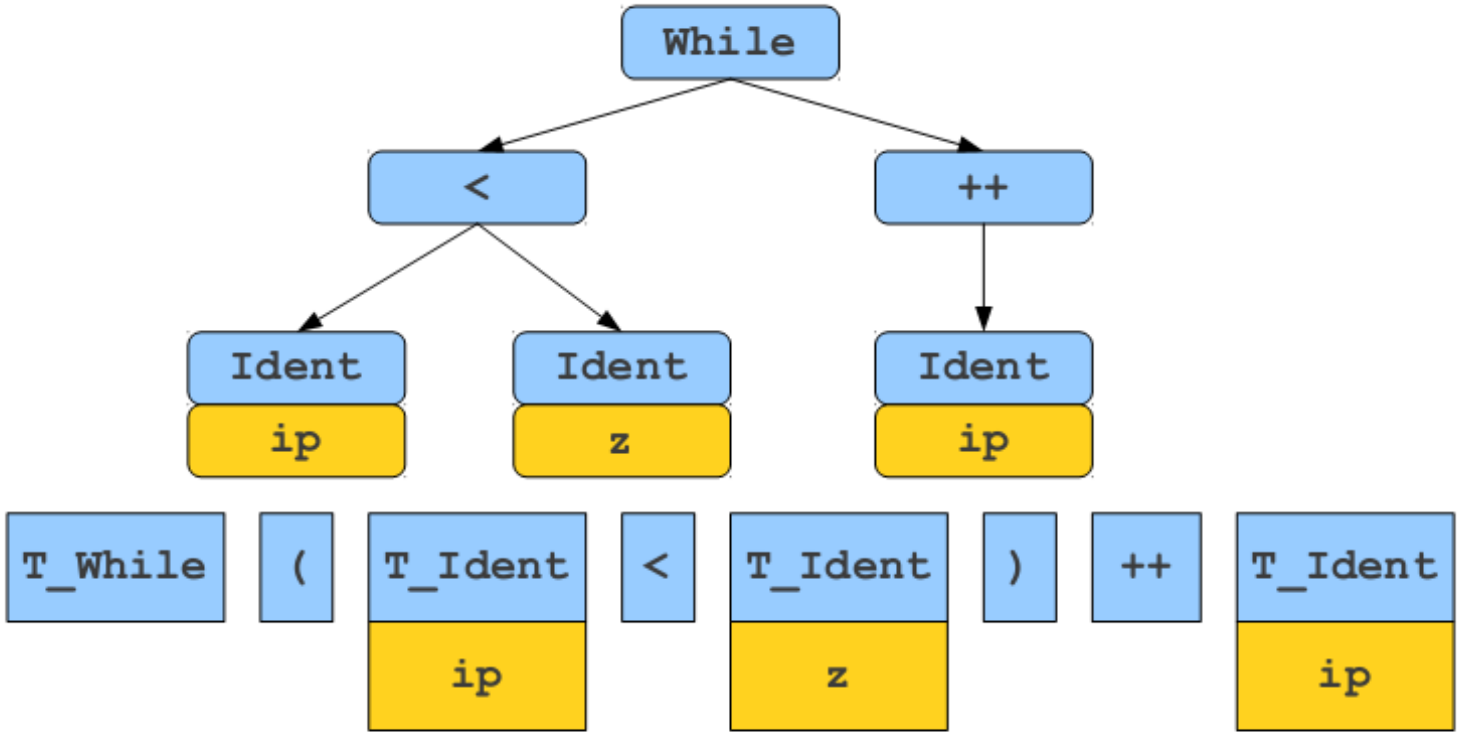
ALTE TIPURI DE CONFLICTE – CUM SE REZOLVĂ

```
T_Do            do
T_Double        double
T_Identifier [A-Za-z_][A-Za-z0-9_]*
```



De ce nu este conflict între acestea două?

SCANAREA ÎN PYTHON



w	h	i	l	e		(	i	p		<		z	)	\n	\t	+	+	i	p	;

```
while (ip < z)
    ++ip;
```

SCANAREA ÎN PYTHON - BLOCURILE

- Delimitarea prin spații:

```
if w == z:
    a = b
    c = d
else:
    e = f
g = h
```

- Ce înseamnă aceasta pentru scanner?

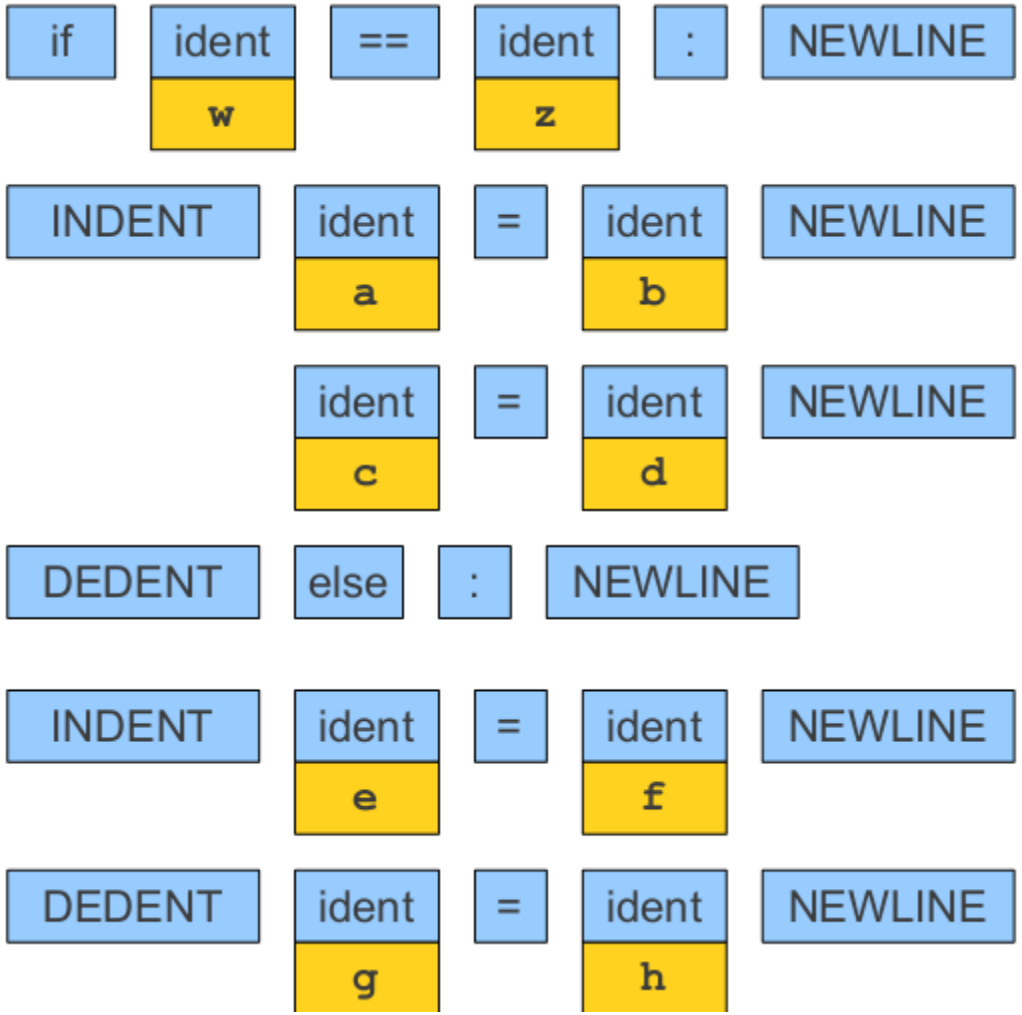
SCANAREA ÎN PYTHON

- NEWLINE marchează sfârșitul liniei.
- INDENT indică creșterea nivelului de indentare.
- DEDENT indică descreșterea nivelului de indentare.

INDENT și DEDENT indică modificarea indentării, nu și numărul total de indentări.

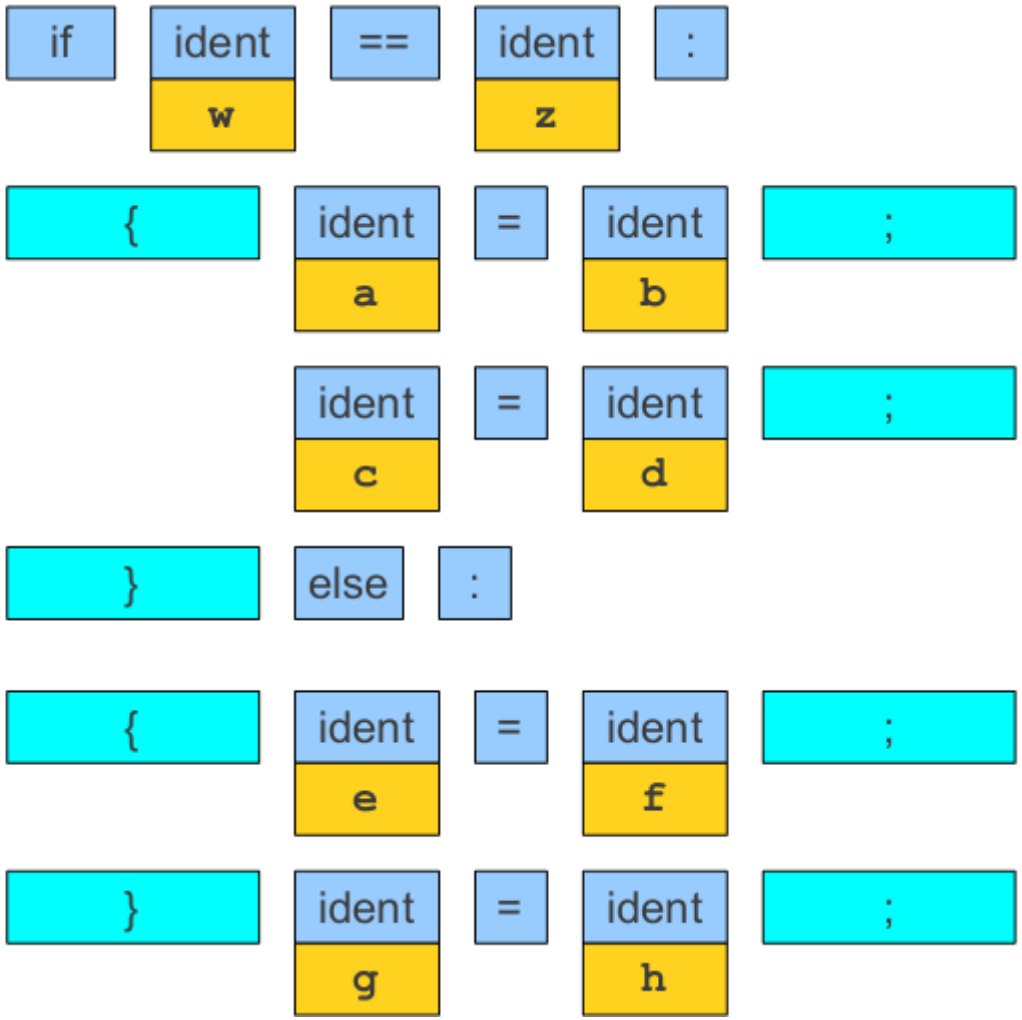
SCANAREA ÎN PYTHON

```
if w == z:
    a = b
    c = d
else:
    e = f
g = h
```



SCANAREA ÎN PYTHON – utilizarea ; și a acoladelor pentru delimitare

```
if w == z: {
     a = b;
     c = d;
} else {
     e = f;
}
g = h;
```



CUM REZOLVĂM CAZURILE DE INDENT/ DEDENT

● Scanner-ul menține o stivă a indentărilor liniilor păstrând toate contextele indentate până la momentul respectiv.
● Inițial, această stivă conține 0, deoarece contextul fișierului inițial nu este indentat.
● Când apare newline:
- Vezi câte spații albe sunt la începutul liniei noi,
- Dacă această valoare este mai mare decât cea din vârful stivei, pune această valoare pe stivă și emite un token INDENT.
● Altfel, cât timp valoarea este mai mică decât cea din vârful stivei:
– Pop stiva.
– Emite token-ul DEDENT.

Sursa: http://docs.python.org/reference/lexical_analysis.html

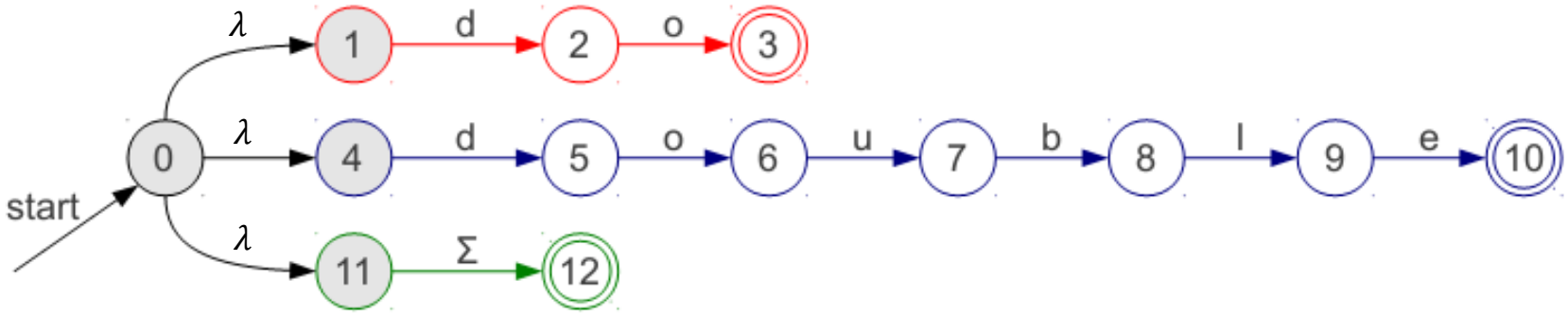
UN ULTIM DETALIU

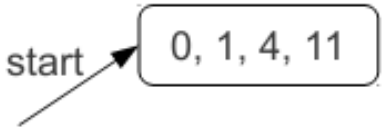
Știm ce să facem atunci când mai multe reguli (expresii regulate) se potrivesc unui șir.
Dar dacă șirul nu se potrivește cu nicio regulă?
Truc: Adăugați o regulă de “interceptare” care se potrivește cu orice caracter și raportați eroare.

SUMAR: REZOLVAREA CONFLICTELOR LA POTRIVIREA TOKEN-ILOR CU EXPRESIILE REGULATE

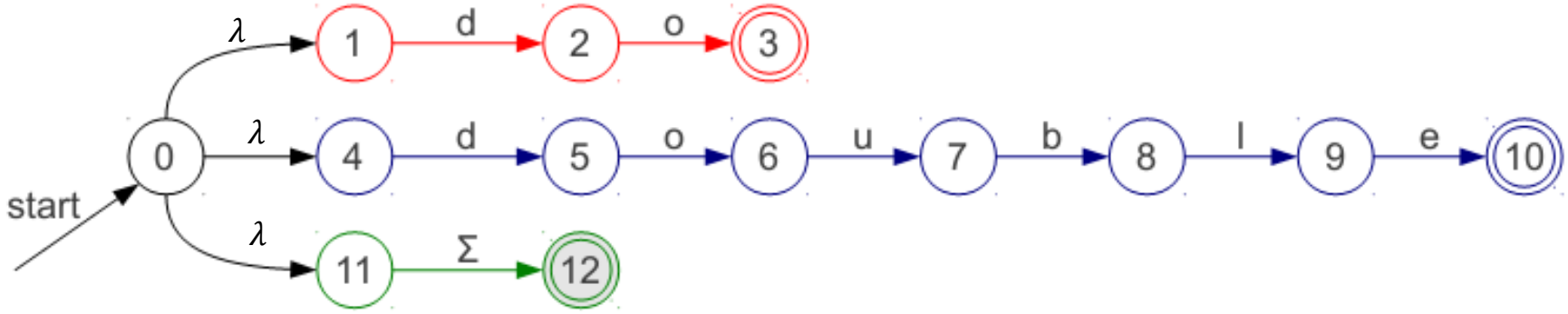
- Construiți un automat pentru fiecare expresie regulată.
- Unificați automatele într-un automat nou prin adăugarea unei noi stări de start. Se introduc λ-tranziții din noua stare inițială în fiecare dintre stările (foste) inițiale ale automatelor
- Scanează inputul, păstrând ultima potrivire.
- Departajează potrivirile pentru mai mult de o expresie alegând-o pe cea care are precedența cea mai mare.
- Introduceți o regulă de interceptare pentru manevrarea erorilor lexicale.

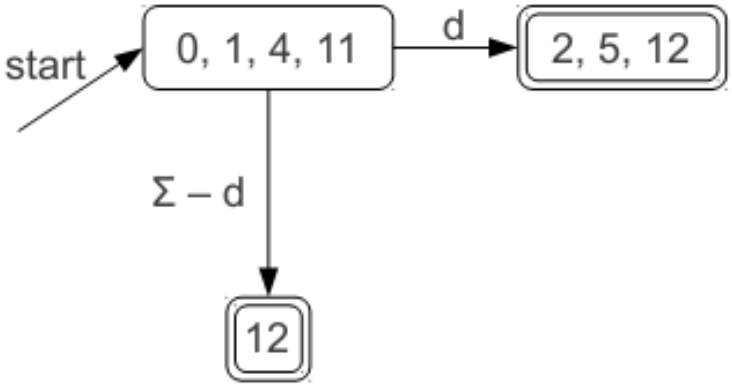
DE LA AFN_λ LA AFD



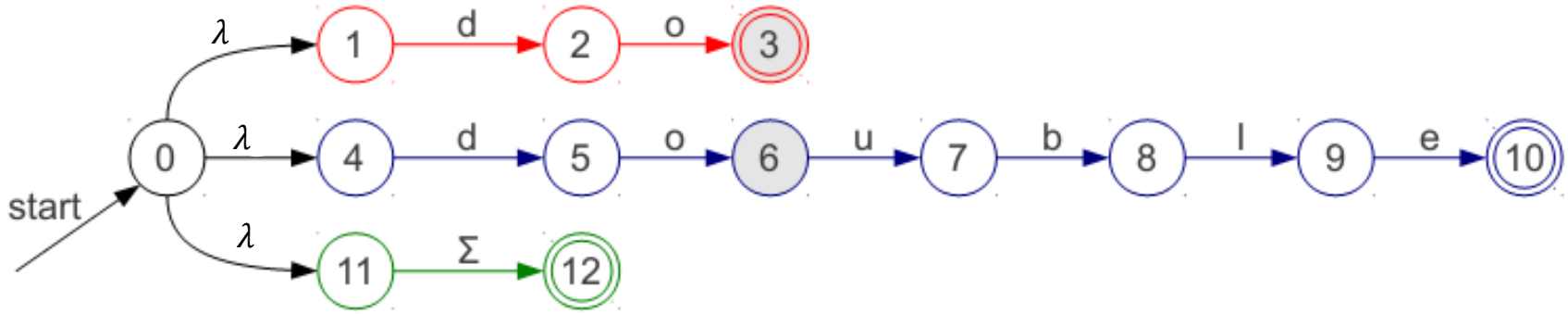


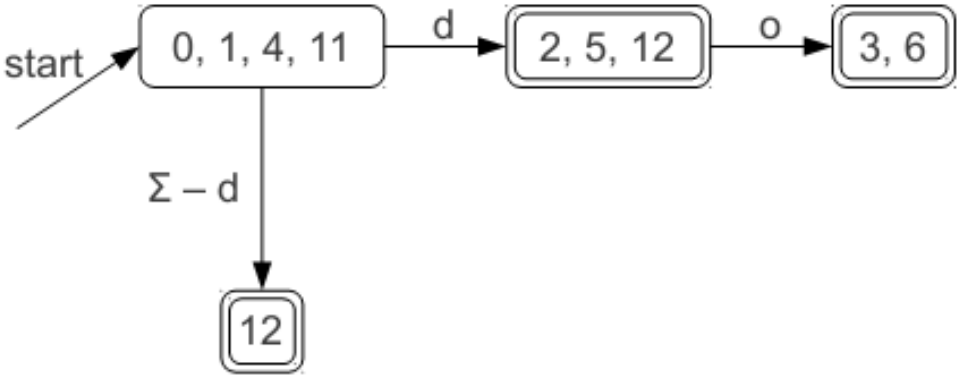
DE LA AFN_λ LA AFD



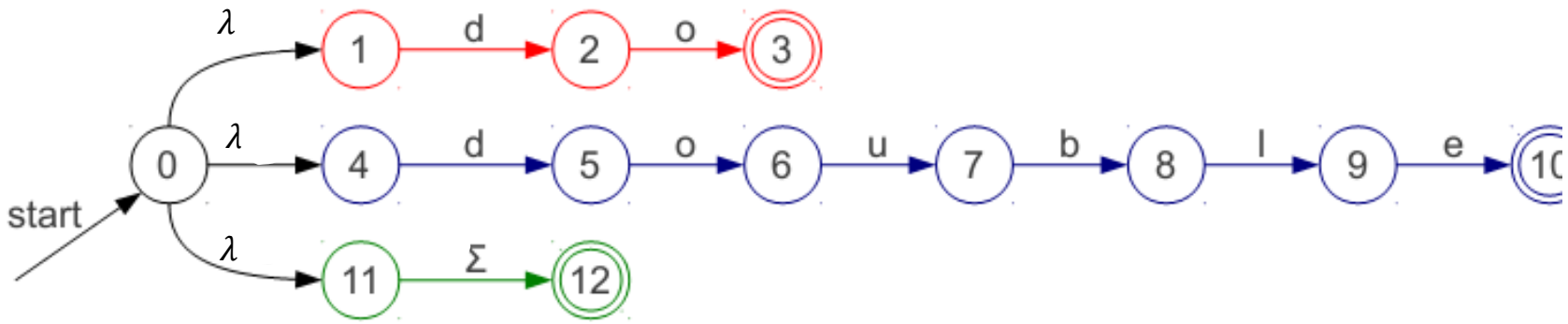


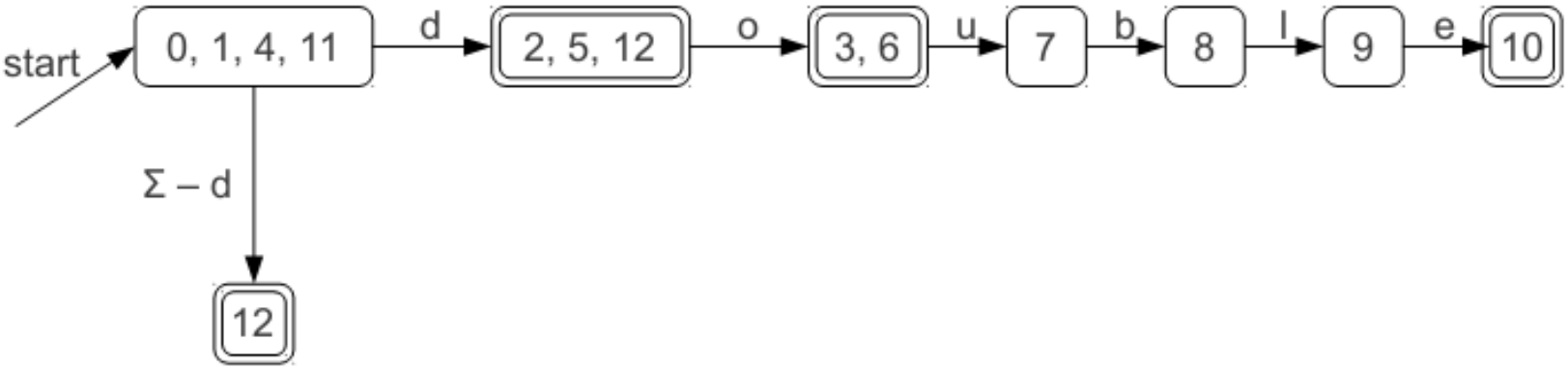
DE LA AFN_λ LA AFD



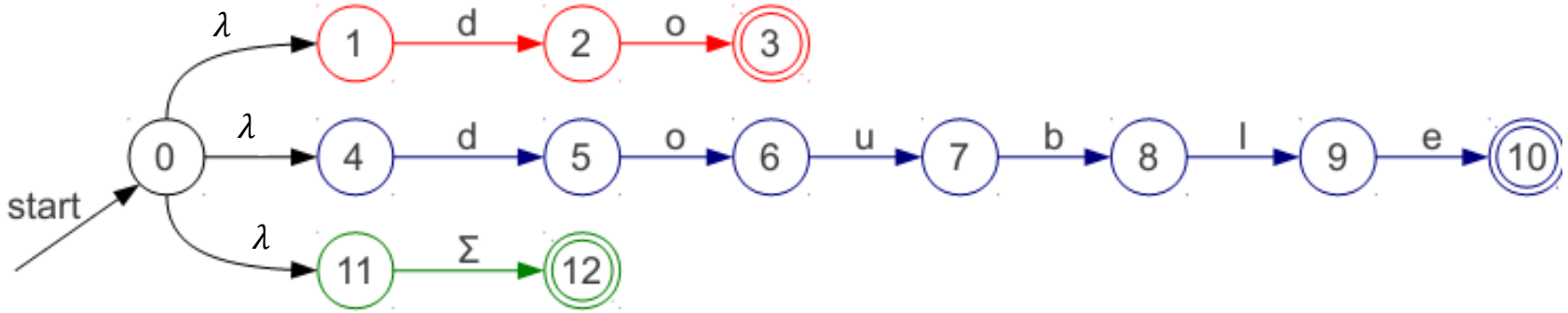


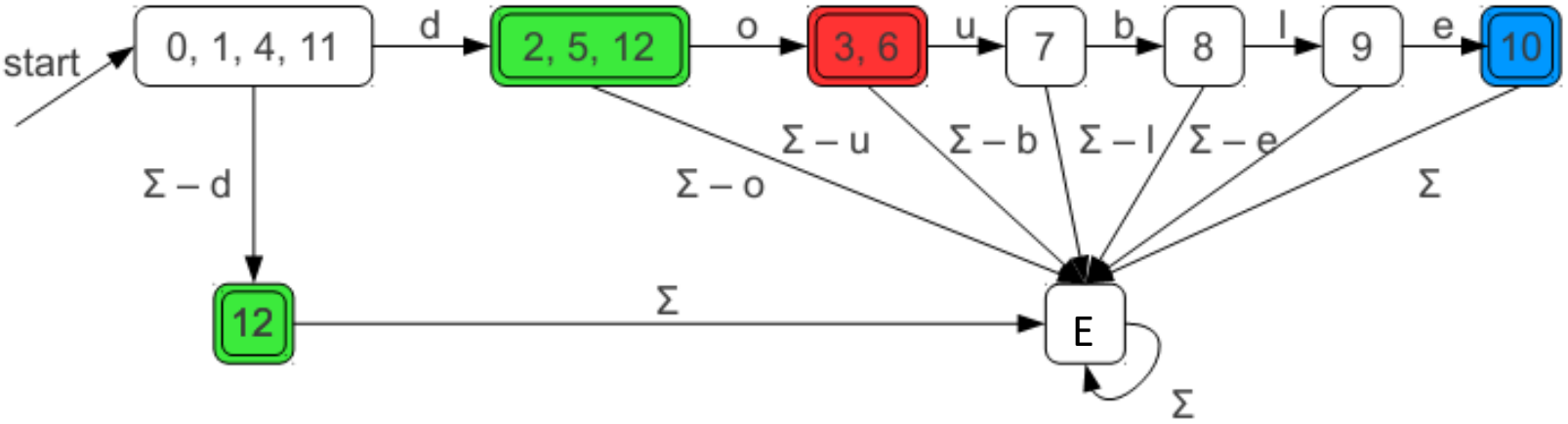
DE LA AFN_λ LA AFD





DE LA AFN_λ LA AFD





E este stare de eroare (”interceptare”)