

# Limbaje independente de context

## Limbaje formale și translațoare (Compilatoare)

December 4, 2020

Mihai-Lica Pura

## ► Analiza sintactică descendentă

- Analiza sintactică predictivă
- Mașina abstractă de analiză predictivă
- Analiza sintactică descendentă cu reveniri
- Analiza sintactică LL(1)
- Mașina abstractă de analiză LL(1)
- Analiza sintactică prin coborâre recursivă

## ► Analiza sintactică ascendentă

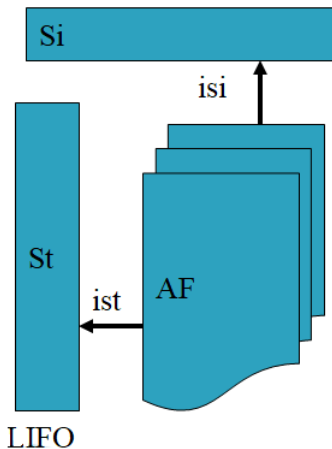
- Analiza sintactică deplasare - reducere (shift - reduce)
- Analiza sintactică ascendentă cu reveniri
- Analiza sintactică bazată pe relații de precedență
  - Analiza sintactică bazată pe precedența operator (Robert Floyd)
  - Analiza sintactică bazată pe precedența simplă (Wirth-Weber)
  - Analiza sintactică bazată pe precedența slabă (Ichbiach-Morse)
- Analiza sintactică LR(1)
  - Analiza sintactică SLR(1)
  - Analiza sintactică LALR(1)

## **Analiza sintactică descendentă**

# Analiza sintactică descendentă

- ▶ încearcă construirea arborelui sintactic pentru propoziția dată **pleacând de la simbolul de start** către propoziție
- ▶ folosește două operații:
  - ▶ **expandare**
  - ▶ **potrivire**
- ▶ poate fi modelată prin intermediul unui automat finit cu stivă care lucrează asupra unui șir de intrare

# Analiza sintactică predictivă



# Analiza sintactică predictivă

- ▶ Si – șirul de intrare (conține atomii lexicali obținuți în urma etapei de analiză lexicală)
- ▶ isi – indice pentru Si
- ▶ St – stiva (acces LIFO) pentru construirea arborelui sintactic
- ▶ ist – vârful stivei

# Analiza sintactică predictivă

Algoritmul analizei sintactice predictive pentru un limbaj definit printr-o gramatică  $G$  este:

1. Analiza sintactică predictivă, fiind un tip de analiză sintactică descendentă, **pleacă de la simbolul de start  $S$ , care este introdus în stivă.**
2. Dacă în vârful stivei se află un neterminal, atunci se face **expandarea** vârfului stivei, alegând convenabil o regulă de producție.

# Analiza sintactică predictivă

3. Dacă în vârful stivei se află un terminal, acesta este comparat cu simbolul curent din șirul de intrare.
  - ▶ Dacă cele două simboluri sunt identice, atunci este scos simbolul din vârful stivei și se avansează un pas în șirul de intrare.
  - ▶ Altfel, se semnalează eroare.
4. La sfârșit:
  - ▶ Dacă **stiva este goală** și dacă **s-a ajuns la sfârșitul șirului de intrare**, atunci acesta este **corect**.
  - ▶ Altfel, șirul de intrare nu este corect.



# Analiza sintactică predictivă

- ▶ algoritm **nedeterminist**
  - ▶ dacă pentru neterminalul aflat în vârful stivei există mai multe reguli de producție
    - ▶ algoritmul NU precizează **cum se alege regula de producție** care va fi folosită pentru expandare
- ▶ poate fi implementat prin
  - ▶ mașina abstractă de analiză predictivă
  - ▶ analizor sintactic descendent cu reveniri
- ▶ însă este ineficient, presupunând încercarea tuturor variantelor posibile

# Analiza sintactica predictiva

- ▶ Fie  $G$  gramatica care definește limbajul pentru care se constăruiește analizorul sintactic predictiv. În construirea arborelui sintactic, analiza sintactică predictivă folosește derivarea stânga:
- ▶  $S \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$ , unde:
  - ▶  $w_i = x_i A \alpha_i$ ,  $w_{i+1} = x_i \beta_i \alpha_i$
  - ▶ cu proprietatea că  $A \rightarrow \beta_i \in P$ ,
  - ▶ iar  $x_i \in \Sigma^*$  și  $\alpha_i \in (V_N \cup \Sigma)$

# Analiza sintactică predictivă

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, \text{id}, (, )\}$
- ▶  $V_N = \{E, T, E', T', F\}$
- ▶  $E$
- ▶  $P = \{$ 
  - $E \rightarrow T E'$
  - $E' \rightarrow + T E'$
  - $E' \rightarrow \epsilon$
  - $T \rightarrow F T'$
  - $T' \rightarrow * F T'$
  - $T' \rightarrow \epsilon$
  - $F \rightarrow \text{id}$
  - $F \rightarrow (E) \}$

Funcționarea automatului de analiză sintactică predictivă pentru analiza propoziției **id+(id\*id)** date este:

# Analiza sintactică predictivă

Stack (top on left)	Input	Remarks
E	id+(id*id)	replace
<b>TE'</b>	id+(id*id)	replace
<b>FT'E'</b>	id+(id*id)	replace
<b>idT'E'</b>	<b>id</b> +(id*id)	erase
T'E'	+(id*id)	replace
E'	+(id*id)	replace
<b>+TE'</b>	<b>+</b> (id*id)	erase
TE'	(id*id)	replace
<b>FT'E'</b>	(id*id)	replace
<b>(E)T'E'</b>	(id*id)	erase
E)T'E'	id*id)	replace
<b>TE')T'E'</b>	id*id)	replace
<b>FT'E')T'E'</b>	id*id)	replace
<b>idT'E')T'E'</b>	<b>id</b> *id)	erase

# Analiza sintactică predictivă

Stack (top on left)	Input	Remarks
$T'E')T'E'$	$*id)$	replace
$*FT'E')T'E'$	$*id)$	erase
$FT'E')T'E'$	$id)$	replace
$idT'E')T'E'$	$id)$	erase
$T'E')T'E'$	$)$	replace
$E')T'E'$	$)$	replace
$)T'E'$	$)$	erase
$T'E'$	$\epsilon$	replace
$E'$	$\epsilon$	replace
$\epsilon$	$\epsilon$	accept

# Automatul finit cu stivă

- ▶ **Modelul matematic** al analizorului sintactic descendent este automatul finit cu stivă:
- ▶  $AP = \langle Q, \Sigma, \Gamma, f, q_0, z_0, F \rangle$ , unde:
  - ▶  $Q$  - mulțimea stărilor
  - ▶  $\Sigma$  - alfabetul automatului
  - ▶  $\Gamma$  - alfabetul stivei
  - ▶  $z_0$  - simbolul inițial al stivei
  - ▶  $q_0$  - starea inițială
  - ▶  $F$  - mulțimea stărilor finale
  - ▶  $f$  - funcția de tranziție
$$f : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*),$$

$P$  fiind o mulțime de perechi

# Automatul finit cu stivă

## ► Configurația automatului

- $(q, x, \gamma)$ ,  $q \in Q$ ,  $x \in \Sigma^*$ ,  $\gamma \in \Gamma^*$
- configurația inițială:  $(q_0, x, S)$

## ► Relația de mișcare

- $(q_1, ax, z\gamma) \vdash (q_2, x, \alpha\gamma) \Leftrightarrow (q_2, \alpha) \in f(q_1, a, z)$

## ► Un șir $w \in \Sigma^*$ este **acceptat** de către automat:

- criteriul stării vide:  
 $(q_0, w, z_0) \vdash^+ (q, \varepsilon, \varepsilon)$
- criteriul stării finale:  
 $(q_0, w, z_0) \vdash^+ (q, \varepsilon, \gamma)$ ,  $q \in F$

# Automatul finit cu stivă

► f

- $f(q, a, a) = \{(q, \epsilon)\}, \forall a \in \Sigma$  - (**potrivire**)
- $f(q, \epsilon, A) = \{(q, \alpha)\}, \forall A \rightarrow \alpha \in P$  - (**expandare**)
- $f(q, \epsilon, \epsilon) = \{(q, \epsilon)\}$  - (**accept**)
- $f(q, \epsilon, \alpha)$  - (**error**)



# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, (, ), i\}$
- ▶  $V_N = \{E, T, F\}$
- ▶  $E$
- ▶  $P = \{$   
     $E \rightarrow E + T \mid T$   
     $T \rightarrow T * F \mid F$   
     $F \rightarrow i \mid (E)$   
     $\}$

# Exemplu

Automatul de analiza sintactică predictivă corespunzător gramaticii este:  
 $AP = \langle Q, \Sigma, \Gamma, f, q_0, z_0, F \rangle$

- ▶  $Q = \{q\}$
- ▶  $\Sigma = \{+, *, (, ), i\}$
- ▶  $\tau = \{E, T, F, +, *, (, ), i\}$
- ▶  $z_0 = E$
- ▶  $q_0 = q$
- ▶  $F = \emptyset$

# Exemplu

## ► f

- $f(q, a, a) = \{(q, \epsilon)\}, \forall a \in \Sigma$
- $f(q, \epsilon, A) = \{(q, \alpha)\}, \forall A \rightarrow \alpha \in P$ 
  - $f(q, \epsilon, E) = \{(q, E + T), (q, T)\}$
  - $f(q, \epsilon, T) = \{(q, T * F), (q, F)\}$
  - $f(q, \epsilon, F) = \{(q, i), (q, (E))\}$

# Exemplu

Sucesiunea de relații de mișcare ale automatului pentru analiza propoziției  $i * i + i$  este:

$(q, i*i+i, E)$

$\vdash (q, i*i+i, E+T)$

$\vdash (q, i*i+i, T+T)$

$\vdash (q, i*i+i, T*F+T)$

$\vdash (q, i*i+i, F*F+T)$

$\vdash (q, i*i+i, i*F+T)$

$\vdash (q, *i+i, *F+T)$

$\vdash (q, i+i, F+T)$

$\vdash (q, i+i, i+T)$

$\vdash (q, +i, +T)$

$\vdash (q, i, T)$

$\vdash (q, i, F)$

$\vdash (q, i, i)$

$\vdash (q, \epsilon, \epsilon)$

# Condiții

Fie  $G$  o gramatică. Pentru a se putea construi un analizor sintactic predictiv pentru limbajul definit de gramatica  $G$ , aceasta trebuie să îndeplinească condițiile:

1. în cadrul regulile de producție, **nu se folosește recursivitate stânga** (directă sau indirectă):
  - ▶ nu există niciun neterminal  $A$  astfel încât  $A \Rightarrow^+ A\alpha$
  - ▶ îndeplinirea acestei condiții asigură faptul că stiva nu se va extinde la infinit, având în vedere că, pentru înlocuirea neterminalelor, se folosește derivarea stânga

2. pentru orice neterminal, **părțile drepte ale regulilor sale de producție încep diferit:**
- ▶ nu există două reguli de producție  $A \rightarrow \alpha\beta$  și  $A \rightarrow \alpha\gamma$ , cu  $\alpha \neq \epsilon$
  - ▶ această condiție asigură identificarea ușoară a regulii de producție care trebuie folosită pentru a expanda neterminalul din vârful stivei, pe baza simbolului curent din șirul de intrare

# Eliminarea recursivității stânga

- ▶ Dacă există reguli de producție care folosesc recursivitatea stânga, de exemplu:

$$A \rightarrow A\alpha$$

$$A \rightarrow \beta$$

- ▶ Atunci:

- ▶ se adaugă în mulțimea  $V_N$  a gramaticii un nou neterminal  $A'$
- ▶ iar cele două reguli de producție din mulțimea  $P$  se înlocuiesc cu:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A'$$

$$A' \rightarrow \epsilon$$

# Eliminarea recursivității stânga

Exemplu:

- ▶  $E \rightarrow E + T$   
 $E \rightarrow T$
- ▶  $E \rightarrow T E'$   
 $E' \rightarrow + T E'$   
 $E' \rightarrow \epsilon$



# Eliminarea regulilor cu același început

- ▶ Dacă nu este îndeplinită a doua condiție, iar gramatica conține, de exemplu, regulile de producție:

$$A \rightarrow \alpha\beta$$

$$A \rightarrow \alpha\gamma$$

- ▶ atunci:

- ▶ se adaugă în mulțimea  $V_N$  a gramaticii un nou neterminat  $A'$

- ▶ iar cele două reguli de producție sunt înlocuite cu:

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta$$

$$A' \rightarrow \gamma$$

# Eliminarea regulilor cu același început

Exemplu:

- ▶  $C \rightarrow \text{if } t \text{ then } C \text{ endif}$   
 $C \rightarrow \text{if } t \text{ then } C \text{ else } C \text{ endif}$
- ▶  $C \rightarrow \text{if } t \text{ then } C \ C'$   
 $C' \rightarrow \text{endif}$   
 $C' \rightarrow \text{else } C \text{ endif}$

# Eliminarea recursivității stânga

Un exemplu mai complex:

$$A \rightarrow Ac$$

$$A \rightarrow Aa$$

$$A \rightarrow b$$

► Soluția este:

$$A \rightarrow bA'$$

$$A' \rightarrow aA'$$

$$A' \rightarrow cA'$$

$$A' \rightarrow \epsilon$$

► după cum se observă, a mai apărut o tranziție pentru  $A'$ . Acum există câte o tranziție pentru  $A'$  pentru fiecare regulă a lui  $A$  care folosește recursivitate stânga

# Eliminarea regulilor cu același început

Un exemplu mai complex:

$$A \rightarrow ab$$
$$A \rightarrow ac$$
$$A \rightarrow Aa$$

- ▶ Mai întâi se rezolvă problema primelor două reguli de producție, obținând:

$$A \rightarrow Aa$$
$$A \rightarrow aA'$$
$$A' \rightarrow b$$
$$A' \rightarrow c$$

- ▶ Apoi, pentru primele două reguli de producție se aplică transformarea pentru eliminarea recursivității stânga:

$$A \rightarrow aA'A''$$
$$A'' \rightarrow aA''$$
$$A'' \rightarrow \epsilon$$
$$A' \rightarrow b$$
$$A' \rightarrow c$$

# Exercițiu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

►  $\Sigma = \{+, -, *, /, id\}$

►  $V_N = \{E\}$

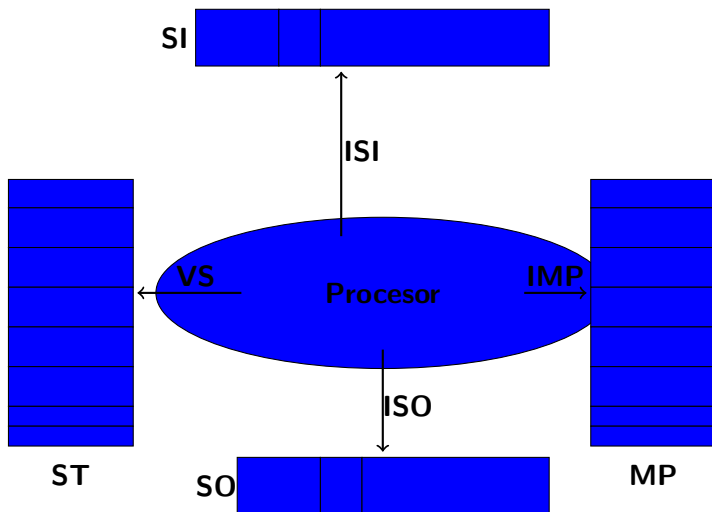
►  $E$

►  $P = \{$   
 $E \rightarrow +EE$   
 $E \rightarrow -EE$   
 $E \rightarrow *EE$   
 $E \rightarrow /EE$   
 $E \rightarrow id \}$

Să se construiască automatul finit cu stivă pentru analiza sintactică predictivă pentru limbajul definit de gramatică și să se analizeze sintactic propoziția  $/ * id id - id id$

# Mașina de analiză predictivă

## Structura mașinii de analiză predictivă



# Mașina de analiză predictivă

- ▶ **Procesorul** este unitatea centrală virtuală care interpretează programele și le execută.
- ▶ **SI** este șirul de intrare cu indicele ISI care marchează poziția atomului lexical curent de analizat
- ▶ **SO** este șirul de ieșire cu indicele ISO care marchează ultimul element introdus în șir
- ▶ **ST** este o memorie de tip stivă, cu VS vârful stivei
- ▶ **MP** este memoria program, accesibilă doar la citire, cu IMP indicele instrucțiunii curente

# Mașina de analiză predictivă

- ▶ Procesorul citește programele din MP și le executa la fel ca și procesorul oricărui calculator real
- ▶ El poate citi un atom din SI, poate scrie un atom lexical în SO și poate scrie sau citi din vârful stivei ST



# Mașina de analiză predictivă

MAP recunoaște și execută **patru instrucțiuni**:

1. Instrucțiunea de **test (V)**
2. Instrucțiunea de **apel cu revenire (A)**
3. Instrucțiunea **adevărat (T)**
4. Instrucțiunea **fals (F)**

# Instrucțiunea de test

$$V(a), E_1, E_2$$

- ▶ Instrucțiunea de test face **verificarea identității dintre atomul lexical curent din SI și atomul lexical indicat între paranteze**, adică a.
- ▶ În caz de identitate:
  - ▶ a este scris în șirul SO
  - ▶ se avansează în SI
  - ▶ și se continuă execuția cu instrucțiunea  $E_1$
- ▶ În caz contrar:
  - ▶ se continuă execuția cu instrucțiunea  $E_2$

# Instrucțiunea de test

$$V(a), E_1, E_2$$

- ▶ Dacă câmpurile  $E_1$  și  $E_2$  sunt vide, atunci execuția se continuă cu următoarea instrucțiune din MP
- ▶  $E_1$  și  $E_2$  pot fi etichete ale unor instrucțiuni din MP, sau pot fi una dintre instrucțiunile T și F

# Instrucțiunea de apel cu revenire

$$A(E), E_1, E_2$$

- ▶ Permite execuția secvenței de program de la eticheta  $E$
- ▶ Înainte de execuția primei instrucțiuni de la eticheta  $E$ , se salvează starea mașinii de analiză, adică tripletul (ISI, ISO, IMP), în ST
- ▶ Dacă returul din secvență se face cu instrucțiunea adevarat (T), atunci se va executa în continuare instrucțiunea  $E_1$
- ▶ Dacă returul din secvență se face cu instrucțiunea fals (F), atunci se va executa în continuare instrucțiunea  $E_2$

# Instrucțiunea de apel cu revenire

$$A(E), E_1, E_2$$

- ▶ Dacă câmpurile  $E_1$  și  $E_2$  sunt vide, atunci execuția se continuă cu următoarea instrucțiune din MP
- ▶  $E_1$  și  $E_2$  pot fi etichete ale unor instrucțiuni din MP, sau pot fi una dintre instrucțiunile T și F

# Algoritmul mașinii de analiză predictivă

ISI,ISO,VS,IMP $\leftarrow$ 0

repetă

    dacă MP(IMP)=A

        atunci \*A

    altfel \*V

# Instrucțiunea V

k=1

daca  $MP(IMP+1)=SI(ISO)$

atunci

$ISO \leftarrow ISO+1$

$SO(ISO) \leftarrow MP(IMP+1)$

$ISO \leftarrow ISO+1$

$K1 \leftarrow IMP+2$

altfel

$K1 \leftarrow IMP+3$

# Intruțiunea V

```
repetă până când k=0
    dacă MP(K1)=blank
        atunci IMP = IMP + 4; K=0
    altfel
        dacă MP(K1)=T
            atunci *T
        altfel
            dacă MP(K1)=F
                atunci *F
            altfel IMP = MP(K1); K=0
```



# Instrucțiunea A

$ST(VS) = ISI, ISO, IMP$

$VS = VS + 3$

$IMP \leftarrow MP(IMP + 1)$

# Instrucțiunea F

```
daca VS=0
    atunci *eexec
altfel
    IMP=ST(VS)
    ISO=ST(VS)
    ISI=ST(VS)
    K1=IMP+3
    VS=VS-3
```

# Intru iunea T

```
daca VS != 0 atunci
    IMP = ST(VS)
    VS = VS-3
    SO(ISO) = MP(IMP+1)
    ISO = ISO + 1
    K1 = IMP + 2
altfel daca VS=0 atunci
    daca SI(ISO-1) =  
        atunci *succes
    altfel *ecec
```

# Algoritmul mașinii de analiză predictivă

Pentru a putea utiliza mașina de analiză predictivă:

- ▶ gramatica generatoare a limbajului țintă **trebuie să îndeplinească numai prima condiție** dintre cele cerute pentru analiza sintactică predictivă

## Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, X, P \rangle$ , unde:

▶  $\Sigma = \{+, *, -, a, (, )\}$

▶  $V_N = \{X, Y, Z, W\}$

▶  $X$

▶  $P = \{$   
     $X \rightarrow Y$   
     $Y \rightarrow Z + W$   
     $Y \rightarrow Z * W$   
     $Y \rightarrow Z$   
     $Z \rightarrow -W$   
     $Z \rightarrow W$   
     $W \rightarrow a$   
     $W \rightarrow (Y) \}$

- ▶ Scrieti programul mașinii de analiză predictivă pentru limbajul definit de gramatica  $G$ .
- ▶ Analizați propoziția  $a+(-a)$ .

# Exemplu

	+	*	-	a	(	)
X			Y	Y	Y	
Y			Z+W Z*W Z	Z+W Z*W Z	Z+W Z*W Z	
Z			-W	W	W	
W				a	(Y)	

# Exemplu

```
00 X: A(Y), ,F
04   : V( $\epsilon$ ),T,F
08 Y: A(Z), ,F
12   : V(+),Y1,
16   : V(*),Y1,T
20 Y1: A(W),T,F
24 Z: V(-), ,
28   : A(W),T,F
32 W: V(a),T,
36   : V((), ,F
40   : A(Y), ,F
44   : V()),T,F
```

# Exemplu

SI	ISI	ISO	IMP	SO	ST
$\underline{a} + (-a)\$$	0	0	0	-	
$\underline{a} + (-a)\$$	0	0	0	-	(0,0,0)
$\underline{a} + (-a)\$$	0	0	8	-	(0,0,0)(0,0,8)
$\underline{a} + (-a)\$$	0	0	24	-	(0,0,0)(0,0,8)
$\underline{a} + (-a)\$$	0	0	28	-	(0,0,0)(0,0,8)(0,0,28)
$\underline{a} + (-a)\$$	0	0	32	-	(0,0,0)(0,0,8)(0,0,28)
$\underline{a} + (-a)\$$	1	2	28	aw	(0,0,0)(0,0,8)
$\underline{a} + (-a)\$$	1	3	8	awz	(0,0,0)
$\underline{a} + (-a)\$$	1	3	12	awz	(0,0,0)



# Exemplu

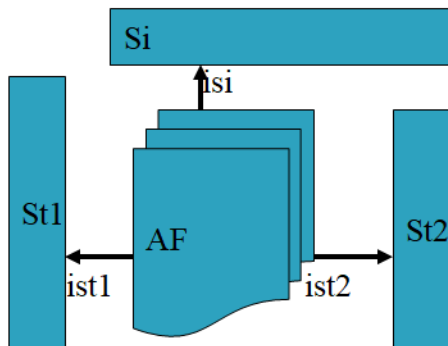
SI	ISI	ISO	IMP	SO	ST
$a + (-a)$$	2	4	20	awz+	(0,0,0)
$a + (-a)$$	2	4	32	awz+	(0,0,0)(2,2,20)
$a + (-a)$$	2	4	36	awz+	(0,0,0)(2,2,20)
$a + (-a)$$	3	5	40	awz+(	(0,0,0)(2,2,20)
$a + (-a)$$	3	5	8	awz+(	(0,0,0)(2,2,20)(3,3,40)
$a + (-a)$$	3	5	24	awz+(	(0,0,0)(2,2,20)(3,3,40)(3,3,8)
$a + (-a)$$	4	6	28	awz+(-	(0,0,0)(2,2,20)(3,3,40)(3,3,8)
$a + (-a)$$	4	6	32	awz+(-	(0,0,0)(2,2,20)(3,3,40)(3,3,8)(4,4,28)
$a + (-a)$$	5	8	28	awz+(-aw	(0,0,0)(2,2,20)(3,3,40)(3,3,8)

# Exemplu

SI	ISI	ISO	IMP	SO	ST
$a + (-a)\$$	5	9	8	$awz + (-awz$	$(0, 0, 0)(2, 2, 20)(3, 3, 40)$
$a + (-a)\$$	5	9	12	$awz + (-awz$	$(0, 0, 0)(2, 2, 20)(3, 3, 40)$
$a + (-a)\$$	5	9	16	$awz + (-awz$	$(0, 0, 0)(2, 2, 20)(3, 3, 40)$
$a + (-a)\$$	5	10	40	$awz + (-awzy$	$(0, 0, 0)(2, 2, 20)$
$a + (-a)\$$	6	11	20	$awz + (-awzy)$	$(0, 0, 0)$
$a + (-a)\$$	6	12	0	$awz + (-awzy)w$	—
$a + (-a)\$$	6	13	4	$awz + (-awzy)wv$	—

# Analiza sintactică descendentă cu reveniri

- ▶ Analiza sintactică descendentă cu reveniri este o implementare deterministă a analizei sintactice predictive



# Analiza sintactica descendenta cu reveniri

- ▶ Si - șirul de intrare (conține atomii lexicali obținuți în urma etapei de analiză lexicală)
- ▶ isi - indice pentru Si
- ▶ St1 - stiva (acces LIFO) pentru construirea arborelui sintactic
- ▶ ist1 - vârful stivei St1
- ▶ St2 - stiva de retur
- ▶ Ist2 - vârful stivei St2

# Analiza sintactica descendentă cu reveniri

Configurația automatului finit cu memorie:

$(s, i, \alpha, \beta)$

- ▶ **s** - starea automatului finit, care poate fi:
  - ▶ q (stare normală)
  - ▶ b (stare de revenire)
  - ▶ e (stare de eroare)
  - ▶ a (stare de acceptare)
- ▶ **j** - indicele șirurului de intrare
- ▶  $\alpha$  - șirul din stiva St2, formată din atomi lexicali deja recunoscuți în șirul de intrare și din indicii regulilor de producție utilizate în relațiile de derivare
- ▶  $\beta$  - șirul din stiva St1

# Analiza sintactica descendenta cu reveniri

## 1. configurația inițială

$(q, 0, \epsilon, S)$

## 2. expandarea

$(q, j, \alpha, A\beta) \rightarrow (q, j, \alpha A1, \alpha_1\beta)$

unde  $A1: A \rightarrow \alpha_1$  este prima regulă de producție pentru neterminalul  $A$  din vârful stivei  $St1$  ( $A$  va fi expandat conform primei alternative)

## 3. avans

$(q, j, \alpha, a\beta) \rightarrow (q, j+1, \alpha a, \beta),$

cand în vârful stivei  $St1$  se află un terminal identic cu cel din poziția curentă ( $j$ ) a șirului de intrare

# Analiza sintactica descendentă cu reveniri

## 4. necoincidența

$$(q, j, \alpha, i\beta) \rightarrow (b, j, \alpha, i\beta)$$

- ▶ când în vârful stivei St1 se află un terminal care nu este identic cu cel din poziția curentă (j) a șirului de intrare

## 5. revenire

$$5.1 \quad (b, j, \alpha i, \beta) \rightarrow (b, j-1, \alpha, i\beta)$$

- ▶ când în vârful stivei St2 se află un terminal
- ▶ pasul 5.1 se repetă atâta timp cât se află un terminal în vârful stivei St2

$$5.2 \quad (b, j, \alpha A_i, \alpha_i \beta) \rightarrow (q, j, \alpha A_{i+1}, \alpha_{i+1} \beta)$$

- ▶ se sare la pasul 3

$$5.3 \quad (b, j, \alpha A_n, \alpha_n \beta) \rightarrow (b, j, \alpha, A\beta)$$

- ▶ se sare la pasul 5.1

# Analiza sintactica descendenta cu reveniri

## 6 eșec

$$(b, j, \alpha, S\beta) \rightarrow (e, j, \alpha, S\beta)$$

## 7 succes

$$(q, n+1, \alpha, \epsilon) \rightarrow (a, n+1, \alpha, \epsilon)$$



# Analiza sintactica descendentă cu reveniri

Pentru a putea utiliza analiza sintactică cu reveniri:

- ▶ gramatica generatoare a limbajului țintă **trebuie să îndeplinească numai prima condiție** dintre cele cerute pentru analiza sintactică predictivă

## Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, S, P \rangle$ , unde:

- ▶  $\Sigma = \{a, b, c\}$
- ▶  $V_N = \{S, A, B\}$
- ▶  $S$
- ▶  $P = \{$ 
  - $(S1) S \rightarrow Ab$
  - $(S2) S \rightarrow Ac$
  - $(A1) A \rightarrow a$
  - $(A2) A \rightarrow bAB$
  - $(B1) B \rightarrow a$ $\}$

Folosind automatul finit cu memorie cu reveniri, analizați propozițiile:

- ▶ bbaaab
- ▶ bbaabab

# Exemplu

$(q, 1, \epsilon, S)$

- $\vdash (2) \quad (q, 1, S1, Ab) \quad // \text{expandare}$
- $\vdash (2) \quad (q, 1, S1A1, ab) \quad // st2(ist2) \neq si(isi)$
- $\vdash (4) \quad (b, 1, S1A1, ab) \quad // \text{necoincidenta}$
- $\vdash (5.2) \quad (q, 1, S1A2, bABb) \quad // \text{revernire}$
- $\vdash (3) \quad (q, 2, S1A2b, ABb) \quad // \text{avans})$
- $\vdash (2) \quad (q, 2, S1A2bA1, aBb) \quad // \text{expandare } A1 \rightarrow a)$
- $\vdash (4) \quad (b, 2, S1A2bA1, aBb) \quad // \text{necoincidenta})$
- $\vdash (5.2) \quad (q, 2, S1A2bA2, bABBBb) \quad // \text{revenire})$

# Exemplu

$\vdash (3)$	$(q, 3, S1A2bA2b, ABBb)$	//avans
$\vdash (2)$	$(q, 3, S1A2bA2bA1, aBBb)$	//expandare
$\vdash (3)$	$(q, 4, S1A2bA2bA1a, BBb)$	//avans
$\vdash (2)$	$(q, 4, S1A2bA2bA1aB1, aBb)$	//expandare
$\vdash (3)$	$(q, 5, S1A2bA2bA1aB1a, Bb)$	//avans
$\vdash (2)$	$(q, 5, S1A2bA2bA1aB1aB1, ab)$	//expandare
$\vdash (3)$	$(q, 6, S1A2bA2bA1aB1aB1a, b)$	//avans
$\vdash (3)$	$(q, 7, S1A2bA2bA1aB1aB1ab, \$)$	//avans
$\vdash (7)$	$(q, 7, S1A2bA2bA1aB1aB1ab, \epsilon)$	//succes
$\vdash (7)$	$(a, 7, S1A2bA2bA1aB1aB1ab, \epsilon)$	

# Analiza sintactică LL(1)

- ▶ Analiza LL(k)
  - ▶ tip de analiză sintactică descendentă
  - ▶ analizează intrarea de la stânga la dreapta (Left to right)
  - ▶ construiește o derivare stânga pentru aceasta (Leftmost derivation)
  - ▶ de aici denumirea de LL
  - ▶ decizia de a folosi a anumită regulă de producție pentru a expanda vârful stivei se ia după inspectarea în avans a unui număr de k atomi lexicali din șirul de intrare
  - ▶ de aici denumirea de LL(k)
  - ▶ în consență, analiza nu folosește reveniri

# Analiza sintactică LL(1)

- ▶ Dacă pentru a anumită gramatică generatoare  $G$  se poate defini un analizor  $LL(k)$ , atunci gramatica se numește **gramtică  $LL(k)$**
- ▶ Limbajul definit printr-o astfel de gramatică se numește limbaj  $LL(k)$
- ▶ Cele mai răspândite sunt **gramaticile  $LL(1)$** 
  - ▶ analizorul trebuie să inspecteze doar următorul atom lexical din șirul de intrare pentru a putea lua deciziile de analiză

# Analiza sintactică LL(1)

Fie gramatica  $G = \langle \Sigma, V_N, S, P \rangle$ , unde:

- ▶  $\Sigma = \{a, b, c\}$
- ▶  $V_N = \{S, A\}$
- ▶  $S$
- ▶  $P = \{$ 
  - (1.1)  $S \rightarrow \underline{a}SAc$
  - (1.2)  $S \rightarrow \underline{\epsilon}$
  - (2.1)  $A \rightarrow \underline{b}A$
  - (2.2)  $A \rightarrow \underline{\epsilon}$ $\}$

	a	b	c
S	1.1	1.2	1.2
A		2.1	2.2

# Analiza sintactică LL(1)

- Succesiunea de relații de mișcare ale automatului finit cu memorie, conform algoritmului de analiză LL(1) pentru propoziția: **aabcc**

$(q, 0, S)$  - a următorul atom, deci alegem 1.1

$\vdash (q, 0, aSAc)$

$\vdash (q, 1, SAc)$  - a următorul atom, deci alegem 1.1

$\vdash (q, 1, aSAcAc)$

$\vdash (q, 2, SAcAc)$  - b următorul atom, deci alegem 1.2

$\vdash (q, 2, AcAc)$  - b următorul atom, deci alegem 2.1



# Analiza sintactică LL(1)

$\vdash (q, 2, bAcAc)$

$\vdash (q, 3, AcAc)$

- c următorul atom, deci alegem 2.2

$\vdash (q, 3, cAc)$

$\vdash (q, 4, Ac)$

- c următorul atom, deci alegem 2.2

$\vdash (q, 4, c)$

$\vdash (q, 5, \epsilon)$

# Gramatici LL(1)

- ▶ Pentru a putea construi un analizor LL(1) pentru o gramatică  $G = \langle \Sigma, V_N, S, P \rangle$ , aceasta trebuie să fie o **gramatică LL(1)**
- ▶ Condiții necesare și suficiente pentru ca o gramatică independentă de context să fie o gramatică LL(1):

$\forall A \in V_N, A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n \in P$

- ▶ 1.  $FIRST^+(\alpha_i) \cap FIRST^+(\alpha_j) = \emptyset, \forall i \neq j$
- ▶ 2.  $FOLLOW^+(A) \cap FIRST^+(\alpha_j) = \emptyset, \forall i, \alpha_i \Rightarrow^* \epsilon \wedge \forall j$

Pentru fiecare  $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n \in P, A \in V_N, \forall i \alpha_i \neq \epsilon$

- ▶  $FIRST^+(\alpha_i) = \{a \in \Sigma | \alpha_i \Rightarrow^* a\gamma, \gamma \in V^*\}$
- ▶ Dacă  $\alpha_i \Rightarrow^* \epsilon$ , atunci trebuie să se țină cont de ceea ce urmează după A
- ▶ (În cazul metodei LL1 se analizează în avans **un singur** atom lexical.)

Pentru  $A \rightarrow \epsilon \in P, A \in V_N$

- ▶  $FOLLOW^+(A) = \{a \in \Sigma \mid (S \Rightarrow^* \alpha A a \beta, \alpha, \beta \in V^*) \vee (S \Rightarrow^* x A Y, a \in FIRST^+(Y), x \in \Sigma^*) \vee (S \Rightarrow^* \alpha A, a \in FOLLOW^+(S))\}$
- ▶ (În cazul metodei LL1 se analizează în avans **un singur** atom lexical.)

# Analiza sintactică LL(1)

Pentru a putea construi un analizor sintactic LL(1):

- ▶ gramatica generatoare a limbajului țintă **trebuie să îndeplinească ambele condiții** cerute pentru analiza sintactică predictivă

# Analiza sintactică LL(1)

Etapele analizei sintactice LL(1):

- ▶ Construirea analizorului sintactic LL(1):
  - ▶ Modificarea gramaticii astfel încât să respecte cele două condiții cerute pentru analiza sintactică predictivă
  - ▶ Calcularea mulțimilor de simbolii directori (mulțimile  $FIRST^+$  sau  $FOLLOW^+$ )
  - ▶ Verificarea îndeplinirii condițiilor pentru ca G să fie o gramatică LL(1)
  - ▶ Construirea tabelului de analiză sintactică
  - ▶ Implementarea analizorului sintactic LL(1)
- ▶ Analiza propriu-zisă a unei propoziții

# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

▶  $\Sigma = \{+, *, (, ), a\}$

▶  $V_N = \{E, T, F\}$

▶  $E$

▶  $P = \{$   
     $E \rightarrow E + T \mid T$   
     $T \rightarrow T * F \mid F$   
     $F \rightarrow a \mid (E)$   
     $\}$

▶ Gramatica folosește **recursivitatea stânga**

▶ pentru a se putea construi un analizor sintactic LL(1) (descendent), ea trebuie să fie modificată

# Exemplu

Gramatica modificată este  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, (, ), a\}$
- ▶  $V_N = \{E, E_1, T, T_1, F\}$
- ▶  $E$
- ▶  $P = \{$ 
  1.  $E \rightarrow TE_1$
  2.  $E_1 \rightarrow +TE_1$
  3.  $E_1 \rightarrow \varepsilon$
  4.  $T \rightarrow FT_1$
  5.  $T_1 \rightarrow *FT_1$
  6.  $T_1 \rightarrow \varepsilon$
  7.  $F \rightarrow (E)$
  8.  $F \rightarrow a$ $\}$



# Exemplu

- ▶ Se calculează mulțimile de simbolii directori:
- ▶  $D_1 = \text{FIRST}^+(TE_1) = \text{FIRST}^+(F) = \{ (, a \}$
- ▶  $D_2 = \text{FIRST}^+(+TE_1) = \{ + \}$
- ▶  $D_3 = \text{FOLLOW}^+(E_1) = \text{FOLLOW}^+(E) = \{ ) \}$
- ▶  $D_4 = \text{FIRST}^+(FT_1) = \text{FIRST}^+(F) = \{ (, a \}$
- ▶  $D_5 = \text{FIRST}^+(*FT_1) = \{ * \}$
- ▶  $D_6 = \text{FOLLOW}^+(T_1) = \text{FOLLOW}^+(T) = \text{FIRST}^+(E_1) \cup \text{FOLLOW}^+(E) = \{ + \} \cup \{ ) \} = \{ +, ) \}$
- ▶  $D_7 = \text{FIRST}^+((E)) = \{ ( \}$
- ▶  $D_8 = \text{FIRST}^+(a) = \{ a \}$

# Exemplu

- ▶ Se verifică îndeplinirea condițiilor pentru ca  $G$  să fie o gramatică LL(1):
  - ▶  $D_2 \cap D_3 = \emptyset$
  - ▶  $D_5 \cap D_6 = \emptyset$
  - ▶  $D_7 \cap D_8 = \emptyset$
- ▶ **Rezultă faptul că gramatica este o gramatică LL(1)**

## Tabela de analiză sintactică

- ▶ pe baza mulțimilor FIRST și FOLLOW calculate, se generează tabela de analiză sintactică

$\Sigma \cup \{\$ \}$

$V_N$			
$\cup$			
$\Sigma$			
$\{\$ \}$			

- ▶ P - pop - scoate un simbol din stivă și înaintează în SI
- ▶ A - accept - propoziția este corectă
- ▶ E - error - propoziția este incorectă
- ▶ R n - replace - se înlocuiește VS cu partea dreaptă a regulii de producție  $n$ , după care  $n$  se scrie în SO
- ▶ simbolul \$ reprezintă terminatorul de șir pentru SI

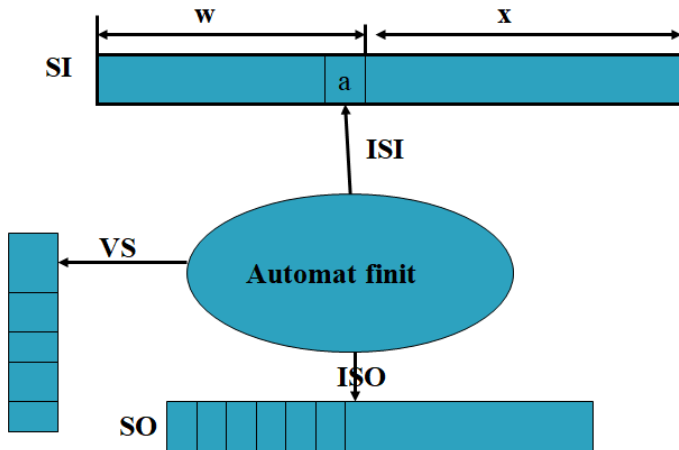
# Tabela de analiză sintactică

## Algoritmul de construcție a tabelii de analiză sintactică:

- ▶ Pentru fiecare regulă de producție  $n : A \rightarrow \alpha \in P$ 
  - ▶ pentru fiecare terminal din  $FIRST^+(\alpha)$ , se completează cu R n celula din tabelă corespunzătoare coloanei terminalului respectiv și liniei neterminalului A
  - ▶ **sau**
  - ▶ pentru fiecare terminal din  $FOLLOW^+(A)$ , se completează cu R n celula din tabelă corespunzătoare terminalului respectiv și liniei neterminalului A

# Analiza sintactică LL(1)

- ▶ pe baza tabelului de analiză sintactică se poate implementa un analizor sintactic determinist cu structura



## Exemplu

- pentru exemplul anterior, tabela de analiză sintactică este (celulele goale reprezintă E):

	$a$	$($	$)$	$+$	$*$	$\$$
$E$	R 1	R 1				
$E_1$			R 3	R 2		R 3
$T$	R 4	R 4				
$T_1$			R 6	R 6	R 5	R 6
$F$	R 8	R 7				
$a$	P					
$($		P				
$)$			P			
$+$				P		
$*$					P	
$\$$						A

# Algoritmul de analiză sintactică LL(1)

```
sf  $\leftarrow$  false  
repetă până când sf = true  
    dacă  $\text{St}(\text{VS}) \in \Sigma \wedge \text{St}(\text{VS}) = \text{SI}(\text{ISI})$   
    atunci  
        pop  
         $\text{ISI} = \text{ISI} + 1$   
    sau  $\text{St}(\text{VS}) \in \Sigma$  și  $\text{St}(\text{VS}) \neq \text{SI}(\text{ISI})$   
    atunci  
        error
```

# Algoritmul de analiză sintactică LL(1)

sau  $\text{St}(\text{VS}) = A \in V_N$

atunci

    replace n

$\text{SO}(\text{ISO}) = n$

$\text{ISO} = \text{ISO} + 1$

sau  $\text{St}(\text{VS}) = \text{SI}(\text{ISI}) = \$$

atunci

    accept

$\text{sf} \leftarrow \text{true}$

altfel

    error

$\text{sf} \leftarrow \text{true}$



# Analiza sintactică LL(1)

- ▶ Modelul matematic al acestui tip de analiză rămâne automatul finit cu stivă
  - ▶ un element suplimentar: șirul de ieșire
- ▶ Funcția de tranziție ( $a, b, i, x \in \Sigma$ ):
  - ▶  $f(q, a, a) = \{(q, \epsilon)\}$  pop
  - ▶  $f(q, \$, \$) = \{(a, \epsilon)\}$  accept
  - ▶  $f(q, b, A) = \{(q, \alpha)\}$  replace  $A \rightarrow b\alpha$  conform tablei de analiză sintactică
  - ▶  $f(q, i, x) = \{(e, \epsilon)\}$  error

# Analiza sintactică LL(1)

## ► Configurația automatului:

- $(q, x, \gamma, y)$
- $q$  - starea automatului
- $x$  - șirul de intrare rămas de analizat
- $\gamma$  - stiva
- $y$  - șirul de ieșire, format din indicii regulilor de producție utilizate în derivări

## ► Configurații posibile: $(a, b, i \in \Sigma, x \in \Sigma^*)$

- $(q, ax, a\gamma, y) \rightarrow (q, x, \gamma, y)$  pop
- $(q, ax, A\gamma, y) \rightarrow (q, x, \alpha\gamma, yi)$  replace  $i$ , conform tabelii de analiză
- $(q, \$, \$, y) \rightarrow (a, \epsilon, \epsilon, y)$  accept
- $(q, bx, i\gamma, y) \rightarrow (e, \epsilon, \epsilon, y)$  error

# Exemplu

Sucesiunea relațiilor de mișcare pentru analiza propoziției  $a * a + a$  de către automatul construit în cadrul exemplului:

$$(q, a * a + a$, E$, \epsilon)$$

$$\vdash^r (q, a * a + a$, TE_1$, 1)$$

$$\vdash^r (q, a * a + a$, FT_1 E_1$, 14)$$

$$\vdash^r (q, a * a + a$, aT_1 E_1$, 148)$$

$$\vdash^p (q, *a + a$, FT_1 E_1$, 148)$$

$$\vdash^r (q, *a + a$, *FT_1 E_1$, 1485)$$

$$\vdash^p (q, a + a$, FT_1 E_1$, 1485)$$

$$\vdash^r (q, a + a$, aT_1 E_1$, 14858)$$

$$\vdash^p (q, +a$, aT_1 E_1$, 14858)$$

# Exemplu

Sucesiunea relațiilor de mișcare pentru analiza propoziției  $a * a + a$  de către automatul construit în cadrul exemplului:

$$\vdash^r (q, +a$,  $aE_1$, 148586)$$$

$$\vdash^r (q, +a$,  $+TE_1$, 1485862)$$$

$$\vdash^p (q, a$,  $TE_1$, 1485862)$$$

$$\vdash^r (q, a$,  $FT_1E_1$, 14858624)$$$

$$\vdash^r (q, a$,  $aT_1E_1$, 148586248)$$$

$$\vdash^p (q, $,  $T_1E_1$, 148586248)$$$

$$\vdash^r (q, $,  $E_1$, 1485862486)$$$

$$\vdash^r (q, $, $, 14858624863)$$

$$\vdash^a (a, $, $, 14858624863)$$

# Exercițiu

Fie gramatica  $G = \langle \Sigma, V_N, \langle \textit{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{\text{if, then, (, ), else, repeat, until, :=, <, <=, i,,}\}$
- ▶  $V_N = \{\langle \textit{instructiune} \rangle, \langle \textit{expr-logica} \rangle, \langle \textit{factor} \rangle, \langle \textit{lista} \rangle\}$
- ▶  $\langle \textit{instructiune} \rangle$

# Exercițiu

- $P = \{$
1.  $\langle \text{instructiune} \rangle \rightarrow \text{if } \langle \text{expr-logica} \rangle \text{ then } (\langle \text{instructiune} \rangle) [ \text{else } \langle \text{instructiune} \rangle ]$
  2.  $\langle \text{instructiune} \rangle \rightarrow \text{repeat } \langle \text{instructiune} \rangle \text{ until } \langle \text{expr-logica} \rangle$
  3.  $\langle \text{instructiune} \rangle \rightarrow \langle \text{factor} \rangle := \langle \text{factor} \rangle$
  4.  $\langle \text{expr-logica} \rangle \rightarrow \langle \text{factor} \rangle < \langle \text{factor} \rangle \mid \langle \text{factor} \rangle < = \langle \text{factor} \rangle$
  5.  $\langle \text{factor} \rangle \rightarrow i \mid i (\langle \text{lista} \rangle)$
  6.  $\langle \text{lista} \rangle \rightarrow i \mid \langle \text{lista} \rangle, i$
- $\}$

# Exercițiu

Gramatica modificată este  $G = \langle \Sigma, V_N, \langle \textit{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{\text{if, then, (, ), else, repeat, until, :=, <, <=, i, ,,}\}$
- ▶  $V_N = \{\langle \textit{instructiune} \rangle, \langle \textit{expr-logica} \rangle, \langle \textit{factor} \rangle, \langle \textit{lista} \rangle, I1, E1, F1, L1\}$
- ▶  $\langle \textit{instructiune} \rangle$

# Exercițiu

- $P = \{$
1.  $\langle \text{instructiune} \rangle \rightarrow \text{if } \langle \text{expr-logica} \rangle \text{ then } (\langle \text{instructiune} \rangle) \text{ l1}$
  2.  $\text{l1} \rightarrow \text{else } \langle \text{instructiune} \rangle$
  3.  $\text{l1} \rightarrow \epsilon$
  4.  $\langle \text{instructiune} \rangle \rightarrow \text{repeat } \langle \text{instructiune} \rangle \text{ until } \langle \text{expr-logica} \rangle$
  5.  $\langle \text{instructiune} \rangle \rightarrow \langle \text{factor} \rangle := \langle \text{factor} \rangle$
  6.  $\langle \text{expr-logica} \rangle \rightarrow \langle \text{factor} \rangle \text{ E1}$
  7.  $\text{E1} \rightarrow \langle \text{factor} \rangle$



# Exercițiu

- ▶ 8.  $E1 \rightarrow <= <factor>$
- 9.  $<factor> \rightarrow i F1$
- 10.  $F1 \rightarrow (<lista>)$
- 11.  $F1 \rightarrow \epsilon$
- 12.  $<lista> \rightarrow i L1$
- 13.  $L1 \rightarrow , i <lista>$
- 14.  $L1 \rightarrow \epsilon$
- }

# Exercițiu

- ▶ **FIRST<sup>+</sup>** { if }
- ▶ **FIRST<sup>+</sup>** { else }
- ▶ **FOLLOW<sup>+</sup>** { ) until \$ }
- ▶ **FIRST<sup>+</sup>** { repeat }
- ▶ **FIRST<sup>+</sup>** { i }
- ▶ **FIRST<sup>+</sup>** { i }
- ▶ **FIRST<sup>+</sup>** { < }
- ▶ **FIRST<sup>+</sup>** { <= }
- ▶ **FIRST<sup>+</sup>** { i }
- ▶ **FIRST<sup>+</sup>** { ( }

# Exercițiu

- ▶  $\text{FOLLOW}^+(F1) = \text{FOLLOW}^+(\langle \text{factor} \rangle)$   
 $= \{ := \} \cup \text{FIRST}^+(E1) = \{ := < <= \} \cup$   
 $\text{FOLLOW}^+(E1) \cup \text{FOLLOW}^+(\langle \text{instrucțiune} \rangle)$   
 $= \{ := < <= \text{ then} \rangle \text{ until } \$ \}$
- ▶  $\text{FIRST}^+ \quad \{ i \}$
- ▶  $\text{FIRST}^+ \quad \{ , \}$
- ▶  $\text{FOLLOW}^+ \quad \{ ) \}$

# Mașina abstractă de analiză LL(1)

- ▶ Structura mașinii de analiză LL(1) corespunde cu structura mașinii abstracte de analiză predictivă

# Mașina abstractă de analiză LL(1)

- ▶ Setul de instrucțiuni al procesorului acestei mașini este format din:
- ▶ **check(a) L**
  - ▶ verifică dacă atomul lexical curent din șirul de intrare este identic cu a
  - ▶ dacă da, atunci execuția se continuă cu instrucțiunea de la eticheta L
  - ▶ dacă nu, se continuă execuția cu instrucțiunea următoare
- ▶ **call A**
  - ▶ apel cu revenire la eticheta A

# Mașina abstractă de analiză LL(1)

- ▶ **return**
  - ▶ revenire dintr-un **call**
- ▶ **pop**
  - ▶ avansare în șirul de intrare
- ▶ **accept**
  - ▶ încheiere execuție program - propoziția este corectă
- ▶ **error**
  - ▶ încheiere execuție program - propoziția este incorectă

# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

►  $\Sigma = \{+, *, (, ), a, ,, \}$

►  $V_N = \{E, T, F, L\}$

►  $E$

►  $P = \{$

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow a$$

$$F \rightarrow a(L)$$

$$F \rightarrow (E)$$

$$L \rightarrow E$$

$$L \rightarrow E, L$$

}

# Exemplu

- ▶ Să se constuiască analizorul sintactic LL(1) pentru limbajul definit de G.
- ▶ Să se scrie programul pentru mașina abstractă de analiză LL(1) pentru analizorul construit.



# Exemplu

**Se modifică gramatica pentru a respecta condițiile pentru implementarea analizei sintactice predictive:**

Gramatica modificată este  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, (, ), a, ,, \}$
- ▶  $V_N = \{E, T, F, L\}$
- ▶  $E$
- ▶  $P = \{$ 
  1.  $E \rightarrow TE_1$
  2.  $E_1 \rightarrow +TE_1$
  3.  $E_1 \rightarrow \epsilon$
  4.  $T \rightarrow FT_1$

# Exemplu

$$5. T_1 \rightarrow *FT_1$$

$$6. T_1 \rightarrow \epsilon$$

$$7. F \rightarrow aF_1$$

$$8. F \rightarrow (E)$$

$$9. F_1 \rightarrow (L)$$

$$10. F_1 \rightarrow \epsilon$$

$$11. L \rightarrow EL_1$$

$$12. L_1 \rightarrow , L$$

$$13. L_1 \rightarrow \epsilon$$

}

# Exemplu

**Calculăm mulțimile de simbolii directori:**

$$D_1 = FIRST^+(TE_1) = \{a; (\}$$

$$D_2 = FIRST^+(+TE_1) = \{+\}$$

$$D_3 = FOLLOW^+(E_1) = FOLLOW^+(E) = \{); , \} \cup FOLLOW^+\{L_1\} = \{); , ; \$\}$$

$$D_4 = FIRST^+(FT_1) = FIRST^+(F) = \{a; (\}$$

$$D_5 = FIRST^+(*FT_1) = \{*\}$$

$$D_6 = FOLLOW^+(T_1) = FOLLOW^+(T) =$$

$$FIRST^+(E_1) \cup FOLLOW^+(E_1) = \{+\} \cup \{)\} \cup FIRST^+(L_1) = \{+; ); , ; \$\}$$

# Exemplu

$$D_7 = FIRST^+(*aF_1) = \{a\}$$

$$D_8 = FIRST^+((E)) = \{(\}$$

$$D_9 = FIRST^+((L)) = \{(\}$$

$$D_{10} = FOLLOW^+(F_1) = FOLLOW^+(F) =$$

$$FIRST^+(T_1) \cup FOLLOW^+(T_1) = \{*\} \cup \{+; ,; \$\} = \{*; ,; ,; \$\}$$

$$D_{11} = FIRST^+(EL_1) = FIRST^+(F) = \{a; (\}$$

$$D_{12} = FIRST^+(, L) = \{, \}$$

$$D_{13} = FOLLOW^+(L_1) = FOLLOW^+(L) = \{)\}$$

## Exemplu

Construim tabelul de analiză LL(1):

	+	*	<i>a</i>	(	)	,	\$
<i>E</i>			R 1	R 1			
<i>E</i> <sub>1</sub>	R 2				R 3	R 3	R 3
<i>T</i>			R 4	R 4			
<i>T</i> <sub>1</sub>	R 6	R 5			R 6	R 6	R 6
<i>F</i>			R 7	R 8			
<i>F</i> <sub>1</sub>	R 10	R 10		R 9	R 10	R 10	R 10
<i>L</i>			R 11	R 11			
<i>L</i> <sub>1</sub>					R 13	R 12	
+	P						
*		P					
<i>a</i>			P				
(				P			
)					P		
,						P	
\$							A

# Exemplu

**PP:** call E  
    check(\$) EE  
    error  
**EE:** accept

# Exemplu

**E:** check(a) EA  
    check() EA  
    error

EA: call T  
    call E1  
    return

# Exemplu

```
E1: check(+) E1A
      check()) E1B
      check(,) E1B
      check($) E1B
      error
E1A: pop (+)
      call T
      call E1
E1B: return
```



# Exemplu

**T:** check(a) TA  
      check(()) TA  
      error

**TA:** call F  
      call T1  
      return

# Exemplu

**T1:** check(\*) T1A  
check(+) T1B  
check()) T1B  
check(,) T1B  
check(\$) T1B  
error

T1A: pop (\*)  
call F  
call T1

T1B: return

# Exemplu

**F:** check(a) FA

check() FB

error

**FA:** pop (a)

call F1

return

**FB:** pop ((

call E

check()) FC

error

**FC:** pop ())

return

# Exemplu

```
F1: check (() F1A
      check (+) F1B
      check (*) F1B
      check ()) F1B
      check (,) F1B
      check ($) E1B
      error
F1A: pop (()
      call L
      check ()) F1C
      error
F1C: pop ())
F1B: return
```

# Exemplu

**L:** check(a) LA  
    check() LA  
    error

**LA:** call E  
      call L1  
      return

# Exemplu

```
L1: check(,) L1A  
    check()) L1B  
    error  
L1A: pop (<)  
    call L  
L1B: return
```

# Exercițiu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

▶  $\Sigma = \{+, -, *, /, (, ), a, ,, \}$

▶  $V_N = \{E, T, F, L\}$

▶  $E$

▶  $P = \{$   
 $E \rightarrow E + T \mid E - T \mid T$   
 $T \rightarrow T * F \mid T / F \mid F$   
 $F \rightarrow a \mid a(L) \mid (E)$   
 $L \rightarrow E \mid E, L$   
 $\}$

Să se scrie programul mașinii abstracte de analiză predictivă LL(1) pentru limbajul definit de gramatica G.

# Analiza sintactică prin coborâre recursivă

- ▶ **implementare a analizei sintactice predictive**
- ▶ fiecare neterminal devine un nume de funcție
- ▶ funcția principală
  - ▶ apelează funcția corespunzătoare simbolului de start
  - ▶ apoi verifică dacă s-a ajuns la sfârșitul șirului de intrare
- ▶ funcția corespunzătoare unui anumit neterminal
  - ▶ este responsabilă să verifice dacă, de la poziția curentă din șirul de intrare, se poate găsi o derivare care pornește de la acest neterminal
  - ▶ dacă nu este posibil, funcția trebuie să sară peste ceea ce a găsit și să semnaleze o eroare
- ▶ există o funcție **next** care face un avans cu o poziție în șirul de intrare



# Exemplu

Exemplificarea implementării analizei sintactice LL(1) prin coborâre recursivă pentru limbajul definit de gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ :

- ▶  $\Sigma = \{+, *, (, ), a\}$
- ▶  $V_N = \{E, T, F\}$
- ▶  $E$
- ▶  $P = \{$   
     $E \rightarrow E + T \mid T$   
     $T \rightarrow T * F \mid F$   
     $F \rightarrow a \mid (E)$   
}

# Exemplu

Gramatica modificată este  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, (, ), a\}$
- ▶  $V_N = \{E, E_1, T, T_1, F\}$
- ▶  $E$
- ▶  $P = \{$ 
  1.  $E \rightarrow TE_1$
  2.  $E_1 \rightarrow +TE_1$
  3.  $E_1 \rightarrow \varepsilon$
  4.  $T \rightarrow FT_1$
  5.  $T_1 \rightarrow *FT_1$
  6.  $T_1 \rightarrow \varepsilon$
  7.  $F \rightarrow (E)$
  8.  $F \rightarrow a$ $\}$

# Exemplu

- ▶ Se calculează mulțimile de simbolii directori:
- ▶  $D_1 = \text{FIRST}^+(TE_1) = \text{FIRST}^+(F) = \{ (, a \}$
- ▶  $D_2 = \text{FIRST}^+(+TE_1) = \{ + \}$
- ▶  $D_3 = \text{FOLLOW}^+(E_1) = \text{FOLLOW}^+(E) = \{ ) \}$
- ▶  $D_4 = \text{FIRST}^+(FT_1) = \text{FIRST}^+(F) = \{ (, a \}$
- ▶  $D_5 = \text{FIRST}^+(*FT_1) = \{ * \}$
- ▶  $D_6 = \text{FOLLOW}^+(T_1) = \text{FOLLOW}^+(T) = \text{FIRST}^+(E_1) \cup \text{FOLLOW}^+(E) = \{ + \} \cup \{ ) \} = \{ +, ) \}$
- ▶  $D_7 = \text{FIRST}^+((E)) = \{ ( \}$
- ▶  $D_8 = \text{FIRST}^+(a) = \{ a \}$

# Exemplu

- ▶ Se verifică îndeplinirea condițiilor pentru ca  $G$  să fie o gramatică  $LL(1)$ :
- ▶  $D_2 \cap D_3 = \emptyset$
- ▶  $D_5 \cap D_6 = \emptyset$
- ▶  $D_7 \cap D_8 = \emptyset$
- ▶ **Rezultă faptul că gramatica este o gramatică  $LL(1)$**

# Exemplu

	$a$	$($	$)$	$+$	$*$	$\$$
$E$	R 1	R 1				
$E_1$			R 3	R 2		R 3
$T$	R 4	R 4				
$T_1$			R 6	R 6	R 5	R 6
$F$	R 8	R 7				
$a$	P					
$($		P				
$)$			P			
$+$				P		
$*$					P	
$\$$						A

# Analiza sintactică prin coborâre recursivă

```
PROCEDURE MAIN IS  
BEGIN  
    E;  
    IF input is empty  
    THEN  
        accept;  
    ELSE  
        error;  
    ENDIF;  
END MAIN;
```

# Analiza sintactică prin coborâre recursivă

```
PROCEDURE E IS
BEGIN
    IF head of input='a' OR
       head of input='('
    THEN
        T;
        E1;
    ELSE
        error;
    ENDIF;
END E;
```

# Analiza sintactică prin coborâre recursivă

```
PROCEDURE E1 IS
BEGIN
    IF      head of input='+'
    THEN
        next;
        T;
        E1;
    ELSEIF  head of input=')' OR
            head of input=  $\epsilon$ 
    THEN
        ;
    ELSE
        error;
    EDIF;
END E1;
```



# Analiza sintactică prin coborâre recursivă

```
PROCEDURE F IS
BEGIN
    IF      head of input='a'
    THEN next
    ELSEIF head of input='('
    THEN next;
        E;
        IF      head of input=')'
        THEN next;
        ELSE error;
        ENDIF;
    ELSE error;
    ENDIF;
END F;
```

# Exercițiu

1. Să se scrie funcțiile corespunzătoare neterminalelor  $T$  și  $T1$  pentru gramatica anterioară.
2. Să se analizeze propoziția "(a)", reprezentând stările stivei de apeluri.

## Întrebări recapitulative

- ▶ Fie  $AP = (Q, \Sigma, \Gamma, f, q_0, z_0, F)$  un automat finit cu stivă care implementează analiza sintactică descendentă pentru gramatica  $G = (\Sigma, V_N, S, P)$ . Scrieți care este configurația inițială a automatului pentru analiza propoziției  $p$  date și explicați.
- ▶ În cazul în care s-ar folosi derivarea dreapta, enumerați care ar fi restricțiile pe care ar trebui să le îndeplinească o gramatică  $G$ , pentru a se putea construi un analizor sintactic descendent care să verifice dacă o propoziție dată aparține mulțimii  $L(G)$ .
- ▶ Explicați de ce este nedeterminist algoritmul general de analiză sintactică descendentă.

# Întrebări recapitulative

- ▶ Explicați în ce scop analizorul LL(1) citește un atom lexical în avans.
- ▶ Explicați de ce nu se poate aplica analiza sintactică descendentă în cazul unei propoziții care aparține limbajului definit de o gramatică ale cărei reguli de producție folosesc recursivitatea stânga.
- ▶ Care sunt intrarea și ieșirea analizei sintactice?

# Întrebări recapitulative

- ▶ Explicați care ar fi avantajul utilizării analizei  $LL(k)$ ,  $k > 1$ , față de analiza  $LL(1)$ .
- ▶ Explicați de ce, în cazul gramaticilor  $LL(1)$ , intersecția mulțimilor de simbolii directori calculate pentru regulile de producție ale aceluiași neterminal trebuie să fie mulțimea vidă.
- ▶ Explicați ce rol are recursivitatea în cadrul analizei sintactice prin coborâre recursivă.

# Întrebări recapitulative

- ▶ Explicați cum rezolvă analiza sintactică descendentă cu reveniri nedeterminismul analizei sintactice predictive.
- ▶ Explicați cum rezolvă analiza LL(1) nedeterminismul analizei sintactice predictive.

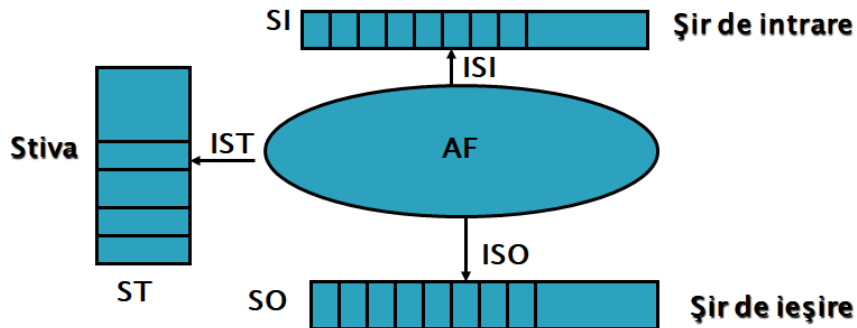
## **Analiza sintactică ascendentă**

# Analiza sintactică ascendentă

- ▶ încearcă construirea arborelui sintactic pentru propoziția dată **pleacând de la propoziție** către simbolul de start
- ▶ folosește două operații:
  - ▶ căutarea frazei simple (priza) - **shift**
  - ▶ reducerea frazei simple - **reduce**
- ▶ poate fi modelată prin intermediul unui automat finit cu stivă care lucrează asupra unui șir de intrare



# Analiza sintactică ascendentă



# Analiza sintactică ascendentă

Algoritmul analizei sintactice ascendente pentru un limbaj definit printr-o gramatică  $G$  este:

- ▶ 1. Stiva este inițial goală.
- ▶ 2. Se deplasează un atom lexical din șirul de intrare în vârful stivei (**shift**).
- ▶ 3. Se repetă pasul al 2-lea până când în vârful stivei se regăsește partea dreaptă a unei reguli de producție.

# Analiza sintactică ascendentă

- ▶ 4. Se înlocuiește vârful stivei care reprezintă partea dreaptă a unei reguli de producție, cu partea stângă a regulii de producție respective (**reduce**).
- ▶ 5. Se repetă pașii 2, 3, 4.
- ▶ 6. La sfârșit:
  - ▶ Dacă **s-a ajuns la sfârșitul șirului de intrare** și dacă **stiva conține numai simbolul de start**, atunci acesta este **corect**.
  - ▶ Altfel, șirul nu este corect.

# Analiza sintactică ascendentă

- ▶ algoritm **nedeterminist**

- ▶ dacă vârful stivei coincide cu părțile drepte ale mai multor reguli de producție
  - ▶ algoritmul NU precizează **cum se alege regula de producție** care va fi folosită pentru reducere
- ▶ dacă vârful stivei coincide cu partea dreaptă a unei reguli de producție și dacă mai există atomi lexicali în șirul de intrare
  - ▶ algoritmul NU precizează **cum se va alege operația** care va fi realizată: shift sau reduce

# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{\text{id}, (, ), +\}$
- ▶  $V_N = \{E\}$
- ▶  $E$
- ▶  $P = \{$ 
  1.  $E \rightarrow (E + E)$
  2.  $E \rightarrow \text{id}$ $\}$

Propoziția de analizat:

$(\text{id} + (\text{id} + \text{id})) \in \Sigma^*$

Stack	Input	Comments
$\epsilon$	(id+(id+id))	shift, shift
(id	+(id+id))	reduce 2
(E	+(id+id))	shift, shift, shift
(E+(id	+id))	reduce 2
(E+(E	+id))	shift, shift
(E+(E+id	))	reduce 2
(E+(E+E	))	shift
(E+(E+E)	)	reduce 1
(E+E	)	shift
(E+E)	$\epsilon$	reduce 1
E	$\epsilon$	accept

# Automatul finit cu stivă

- ▶ **Modelul matematic** al analizorului sintactic ascendent este automatul finit cu stivă:
- ▶  $AP = \langle Q, \Sigma, \Gamma, f, q_0, z_0, F \rangle$ , unde:
  - ▶  $Q$  - mulțimea stărilor
  - ▶  $\Sigma$  - alfabetul automatului
  - ▶  $\Gamma$  - alfabetul stivei
  - ▶  $z_0$  - simbolul inițial al stivei
  - ▶  $q_0$  - starea inițială
  - ▶  $F$  - mulțimea stărilor finale
  - ▶  $f$  - funcția de tranziție
$$f : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*),$$

$P$  fiind o mulțime de perechi

# Automatul finit cu stivă

## ► Configurația automatului

- $(q, x, \gamma)$ ,  $q \in Q$ ,  $x \in \Sigma^*$ ,  $\gamma \in \Gamma^*$
- configurația inițială:  $(q_0, x, \epsilon)$

## ► Relația de mișcare

- $(q_1, ax, \alpha\beta) \vdash (q_2, x, \alpha\gamma) \Leftrightarrow (q_2, \gamma) \in f(q_1, a, \beta)$

## ► Un șir $w \in \Sigma^*$ este **acceptat** de către automat:

- criteriul stării vide:  
 $(q_0, w, z_0) \vdash^+ (q, \epsilon, S)$
- criteriul stării finale:  
 $(q_0, w, z_0) \vdash^+ (q, \epsilon, \gamma)$ ,  $q \in F$



# Automatul finit cu stivă

► f

- $f(q, a, \alpha) = \{(q, a\alpha)\}$  - (**shift**)
- $f(q, \epsilon, \alpha\beta) = \{(q, A\beta)\} \Leftrightarrow A \rightarrow \alpha \in P$  - (**reduce**)
- $f(q, \epsilon, S) = \{(q, \epsilon)\}$  - (**accept**)
- $f(q, \epsilon, \alpha)$  - (**error**)

# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, S, P \rangle$ , unde:

- ▶  $\Sigma = \{a, b, c\}$
- ▶  $V_N = \{S, A\}$
- ▶  $S$
- ▶  $P = \{$ 
  1.  $S \rightarrow aAc$
  2.  $A \rightarrow bA$
  3.  $A \rightarrow b$ $\}$

Propoziția de analizat:

$abbbc \in \Sigma^*$

# Exemplu

$(q, \text{abbbbc}\$, \$)$

$\vdash^s (q, \text{bbbc}\$, \text{a}\$)$

$\vdash^s (q, \text{bbc}\$, \text{ab}\$)$

$\vdash^s (q, \text{bc}\$, \text{abb}\$)$

$\vdash^s (q, \text{c}\$, \text{abbb}\$)$

$\vdash^{r(3)} (q, \text{c}\$, \text{abbA}\$)$

$\vdash^{r(2)} (q, \text{c}\$, \text{abA}\$)$

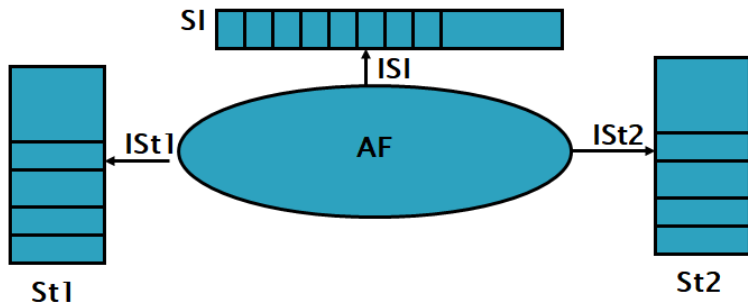
$\vdash^{r(2)} (q, \text{c}\$, \text{aA}\$)$

$\vdash^s (q, \$, \text{aAc}\$)$

$\vdash^{r(1)} (q, \$, \text{S}\$)$

$\vdash^a (q, \$, \$)$

## Analiza sintactică ascendentă cu revenire



# Analiza sintactică ascendentă cu revenire

1. SI - șirul de intrare (conține atomii lexicali obținuți în urma etapei de analiză lexicală)
2. ISI - indice pentru SI
3. St1 - stiva (acces LIFO) pentru construirea arborelui sintactic
4. ISt1 - vârful stivei St1
5. St2 - stiva de retur
6. ISt2 - vârful stivei St2

# Analiza sintactică ascendentă cu revenire

- ▶ (1) **Configurația inițială:**

$(q, 0, \$, \epsilon)$

- ▶ (2) **reduce:**

$(q, i, \beta\alpha, \gamma) \vdash (q, i, \beta A, \gamma j) \Leftrightarrow \exists j : A \rightarrow \alpha \in P$

- ▶ regulile de producție sunt ordonate crescator, după lungimea părților drepte
- ▶ se buclează (2) atât timp cât se mai poate face o reducere

# Analiza sintactică ascendentă cu revenire

## ► (3) **shift**

$(q, i, \alpha, \gamma) \vdash (q, i+1, \alpha a, \gamma/)$

- / indică un avans în șirul de intrare
- dacă  $i < n+1$ , atunci se sare la (2)
- dacă  $i = n+1$ , atunci se sare la (4)

## ► (4) **accept**

$(q, n+1, \$\$, \gamma) \vdash (a, n+1, \$\$, \gamma)$

- $\gamma$  va conține șirul derivărilor stânga
- dacă nu se poate aplica (4), atunci se sare la (5)

# Analiza sintactică ascendentă cu revenire

## (5) revenire

$$(q, n+1, \alpha, \gamma) \vdash (b, n+1, \alpha, \gamma)$$

5.1.  $(b, i, \alpha a, \gamma/) \vdash (b, i-1, \alpha, \gamma)$

se ciclează cât timp există / în vârful St2

5.2.  $(b, i, \alpha A, \gamma j) \vdash (q, i, \alpha'' B, \gamma k) \Leftrightarrow \alpha = \alpha'' \alpha'$

și  $\exists k : B \rightarrow \alpha' \beta$  și  $j : A \rightarrow \beta \in P$

se sare apoi la (2)

5.3.  $(b, i, \alpha A, \gamma j) \vdash (q, i+1, \alpha \beta a, \gamma/) \Leftrightarrow i < n$

se sare apoi la (2)

5.4.  $(b, n+1, \alpha A, \gamma j) \vdash (q, n+1, \alpha \beta, \gamma)$

se sare apoi la (2)



# Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, S, P \rangle$ , unde:

- ▶  $\Sigma = \{a, b, c\}$
- ▶  $V_N = \{S, A, B, C\}$
- ▶  $S$
- ▶  $P = \{$ 
  1.  $S \rightarrow Bab$
  2.  $S \rightarrow Cac$
  3.  $A \rightarrow BA$
  4.  $A \rightarrow a$
  5.  $B \rightarrow a$
  6.  $C \rightarrow a$ $\}$

Propoziția de analizat: aab

# Exemplu

$(q, 0, \$, \epsilon)$

$\vdash^s (q, 1, \$a, /)$

$\vdash^r (q, 1, \$A, /4)$

$\vdash^s (q, 2, \$Aa, /4/)$

$\vdash^r (q, 2, \$AA, /4/4)$

$\vdash^s (q, 3, \$AAb, /4/4/)$

$\vdash^b (b, 3, \$AAb, /4/4/)$

$\vdash_{s.1}^b (b, 2, \$AA, /4/4)$

$\vdash_{s.2}^b (q, 2, \$AB, /4/5)$

$\vdash^s (q, 3, \$ABb, /4/5/)$

$\vdash^s (b, 3, \$ABb, /4/5/)$

$\vdash_{s.1}^b (b, 2, \$AB, /4/5)$

# Exemplu

$\vdash_{s.2}^b (q, 2, \$AC, /4/6)$   
 $\vdash^s (q, 3, \$ACb, /4/6/)$   
 $\vdash_s^b (b, 3, \$ACb, /4/6/)$   
 $\vdash_{s.1}^b (b, 2, \$AC, /4/6)$   
 $\vdash_{s.3}^b (q, 3, \$Aab, /4//)$   
 $\vdash_s^b (b, 3, \$Aab, /4//)$   
 $\vdash_{s.1}^b (b, 1, \$A, /4)$   
 $\vdash_{s.2}^b (q, 1, \$B, /5)$   
 $\vdash^s (q, 2, \$Ba, /5/)$   
 $\vdash^r (q, 2, \$BA, /5/4)$   
 $\vdash^r (q, 2, \$A, /5/43)$   
 $\vdash^r (q, 3, \$Ab, /5/43/)$   
 $\vdash^r (b, 3, \$Ab, /5/43/)$   
 $\vdash^r (b, 2, \$A, /5/43)$   
 $\vdash^s (q, 3, \$BAb, /5/4/)$

# Exemplu

$\vdash^s (b, 3, \$BAb, /5/4/)$

$\vdash^{s.1} (b, 2, \$BA, /5/4)$

$\vdash^{s.2} (q, 2, \$BB, /5/5)$

$\vdash^{5.2} (q, 2, \$BB, /5/5)$

$\vdash^s (q, 3, \$BBb, /5/5/)$

$\vdash^5 (b, 3, \$BBb, /5/5/)$

$\vdash^{5.1} (b, 2, \$BB, /5/5)$

$\vdash^{5.2} (q, 2, \$BC, /5/6)$

$\vdash^s (q, 3, \$BCb, /5/6/)$

$\vdash^5 (b, 3, \$BCb, /5/6/)$

$\vdash^{5.1} (b, 2, \$BC, /5/6)$

$\vdash^{5.3} (q, 3, \$Bab, /5//)$

# Exemplu

$\vdash^r (q, 3, \$S, /5//1)$

$\vdash^s (q, 4, \$S$, /5//1)$

$\vdash (a, 4, \$S$, /5//1)$

# Exemplu

Șirul reducerilor stânga: 5 1

$aab \rightarrow^{(5)} Bab \rightarrow^{(1)} S$

Șirul derivărilor stânga: 1 5

$S \Rightarrow^{(5)} Bab \Rightarrow^{(1)} aab$

# Analiza sintactică bazată pe relații de precedență

- ▶ mulțimea limbajelor de precedență  $\subset$  mulțimea limbajelor independente de context
- ▶ relații de precedență  $<$   $=$   $>$ 
  - ▶ operator
  - ▶ simplă
  - ▶ slabă
- ▶ pentru a determina un capăt se analizează relațiile de precedență dintre ST(IST) și SI(ISI)

# Analiza sintactică bazată pe relații de precedență

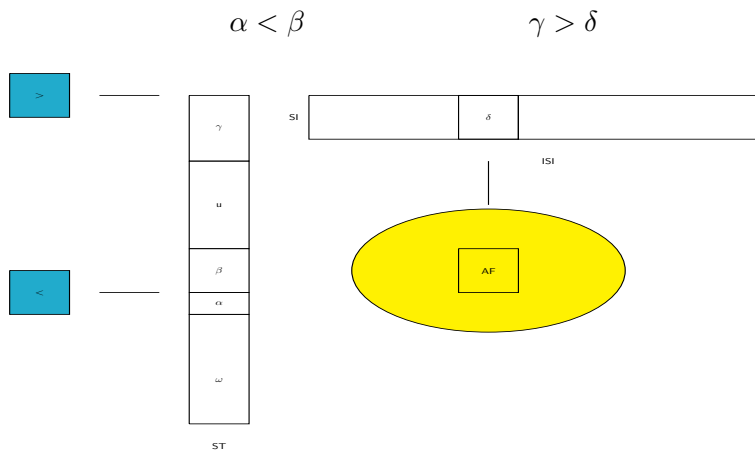
$$S \Rightarrow^* \omega \alpha \underline{A} \delta \nu \Rightarrow \omega \alpha \underline{\beta v \gamma} \delta \nu, \exists A \rightarrow \beta v \gamma \in P$$

$$\alpha \in (V_N \cup \Sigma) \quad \beta \in (V_N \cup \Sigma) \quad v \in (V_N \cup \Sigma)^*$$

$$\gamma \in (V_N \cup \Sigma) \quad \delta \in \Sigma \quad \nu \in \Sigma^*$$



# Analiza sintactică bazată pe relații de precedență



# Analiza sintactică bazată pe precedența operator (Robert Floyd)

## Robert Floyd

Carnegie Mellon University, Stanford University

### Lucrari in:

theory of parsing, semantics of programming languages, automatic program verification, automatic program synthesis, and analysis of algorithms



# Analiza sintactică bazată pe precedența operator

O gramatică

- ▶ independentă de context
- ▶ fără reguli vide

este o **gramatică în forma operator** dacă

- ▶ nu are reguli de producție vide
- ▶ nu are reguli de producție de forma:

$$\mathbf{A} \rightarrow \alpha BC\beta \text{ și } A \rightarrow B$$

$$\alpha, \beta \in (V_N \cup \Sigma)^*, B, C, \in V_N$$

(Adică în partea dreaptă a oricărei reguli de producție nu există două neterminale unul după celălalt, și nici un singur neterminal.)

# Analiza sintactică bazată pe precedența operator

Într-o astfel de gramatică operanzii sunt neterminalele, iar operatorii sunt terminalele.

Orice gramatică independentă de context poate fi adusă la forma operator fără a afecta limbajul definit de gramatică.

# Analiza sintactică bazată pe precedența operator

## Definirea **relațiilor de precedență operator**

- ▶ se face pentru simbolurile terminale
- ▶ are scopul de a elimina ambiguitățile algoritmului general de analiză sintactică descendentă și anume:
  - ▶ de a identifica partea dreaptă a regulii de producție care va fi folosită pentru reducere (atunci când există mai multe variante posibile)
  - ▶ de a identifica operația care fi executată (shift sau reduce) atunci când ambele sunt posibile

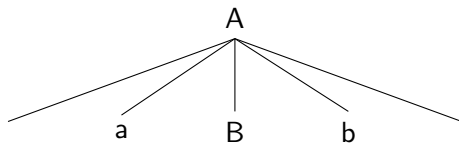
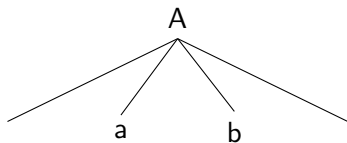
# Analiza sintactică bazată pe precedența operator

- ▶  $<_o$  Determină capătul din stânga al părții drepte a regulii de producție
- ▶  $>_o$  Determină capătul din dreapta al părții drepte a regulii de producție
- ▶  $=_o$  Determină interiorul părții drepte al regulii de producție

# Definirea relațiilor de precedență operator

$$a =_o b \Leftrightarrow \exists A \rightarrow \alpha ab\beta \in P \textbf{ sau } A \rightarrow \alpha aBb\beta \in P$$

unde  $a, b \in \Sigma$ ,  $A, B \in V_N$  și  $\alpha, \beta \in V^*$

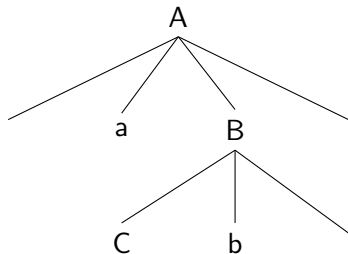
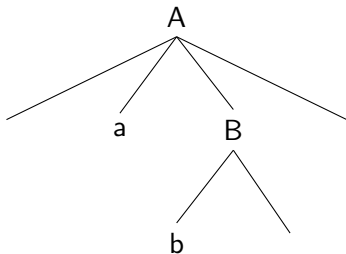


## Definirea relațiilor de precedență operator

$$a <_o b \Leftrightarrow \exists A \rightarrow \alpha a B \beta \in P \text{ și } B \Rightarrow^+ b\gamma \text{ sau } B \Rightarrow^+ Cb\gamma$$

unde  $a, b \in \Sigma$ ,  $A, B, C \in V_N$  și  $\alpha, \beta \in V^*$   
și se scrie  $a <_o FIRST \sim^+(B)$

( $FIRST \sim^+ =$  terminalul care apare pe prima poziție sau terminalul care apare pe a doua poziție, dacă pe prima este un neterminal)





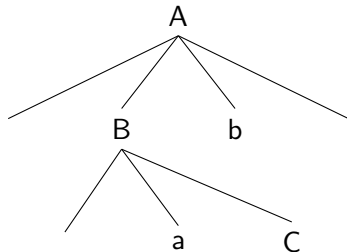
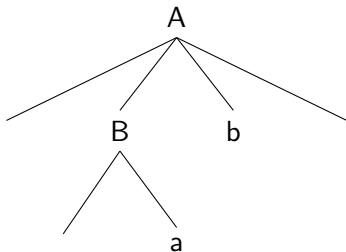
## Definirea relațiilor de precedență operator

$$a >_o b \Leftrightarrow \exists A \rightarrow \alpha B b \beta \in P \text{ și } B \Rightarrow^+ \gamma a \text{ sau } B \Rightarrow^+ \gamma a C$$

unde  $a, b \in \Sigma$ ,  $A, B, C \in V_N$  și  $\alpha, \beta \in V^*$

și se scrie  $LAST \sim^+ (B) >_o b$

( $LAST \sim^+ =$  terminalul care apare pe ultima poziție sau terminalul care apare pe penultima poziție, dacă pe ultima este un neterminal)



# Definirea relațiilor de precedență operator

$$\$ <_o a \Leftrightarrow S \Rightarrow^* a\alpha \text{ sau } S \Rightarrow^* Aa\alpha$$

\$ este mai mic decât orice terminal care apare pe prima poziție sau pe a doua, dacă pe prima este un neterminal, în formele propoziționale

$$a >_o \$ \Leftrightarrow S \Rightarrow^* \alpha a \text{ sau } S \Rightarrow^* \alpha aA$$

este mai mare decât \$ orice terminal care apare pe ultima poziție sau pe penultima, dacă pe ultima este un neterminal, în formele propoziționale

# Definirea relațiilor de precedență operator

- ▶ la fel ca și la calculul mulțimilor  $FIRST^+$  și  $FOLLOW^+$  de la analiza LL(1), și în cazul mulțimilor  $FIRST \sim^+$  și  $LAST \sim^+$  de la precedența operator se merge (în adâncime) până la nivelul maxim posibil
- ▶ e.g. dacă, în cazul relației de precedență operator mai mic, se află un neterminal pe prima poziție, atunci se ia terminalul de după, dar se și continuă aplicând pe  $FIRST \sim^+$  și pentru acel neterminal
- ▶ analog și în cazul relației de precedență operator mai mare



# Analiza sintactică bazată pe precedența operator

Etapele analizei sintactice bazate pe precedența operator:

- ▶ Construirea analizorului sintactic bazat pe precedența operator:
  - ▶ Modificarea gramaticii astfel încât să respecte condițiile cerute pentru gramaticile operator
  - ▶ Calcularea relațiilor de precedență operator (pe baza mulțimilor  $FIRST \sim^+$  sau  $LAST \sim^+$ )
  - ▶ Construirea matricei de precedență operator
  - ▶ Implementarea analizorului bazat pe relațiile de precedență operator
- ▶ Analiza propriu-zisă a unei propoziții

# Matricea de precedență operator

	$\Sigma \cup \{\$, \$\}$		
$\Sigma$			
$\cup$			
$\{\$, \$\}$			

- ▶ relația de precedență operator existentă între terminalul de pe linia  $i$  și terminalul de pe coloana  $j$
- ▶ în celula corespunzătoare perechii  $(\$, \$)$  relația este "accept" - propoziția este acceptată
- ▶ în toate celulele care rămân necomplete relația este "error" - propoziția nu este acceptată

# Algoritmul de analiză pe baza matricei de precedență operator

$ISI \leftarrow 0$

repetă

dacă  $ST(IST) = \$$  și  $SI(ISI) = \$$  atunci propoziția este acceptată  
altfel

$a \leftarrow ST(IST)$

$b \leftarrow SI(ISI)$

dacă  $a <_o b$  sau  $a =_o b$  atunci

shift  $b$  în stivă

$ISI \leftarrow ISI + 1$

altfel dacă  $a >_o b$  atunci

repetă

scoate un terminal din stivă până când

$ST(IST) <_o$  decât ultimul terminal scos din stivă

altfel eroare

## Exemplu

Fie gramatica  $G = \langle \Sigma, V_N, E, P \rangle$ , unde:

- ▶  $\Sigma = \{+, *, (, ), a\}$
- ▶  $V_N = \{E, T, F, S, D\}$
- ▶  $E$
- ▶  $P = \{$ 
  - $E \rightarrow E + T$
  - $E \rightarrow T$
  - $T \rightarrow T * F$
  - $T \rightarrow F$
  - $F \rightarrow SED$
  - $F \rightarrow a$
  - $S \rightarrow ($
  - $D \rightarrow )$ $\}$

Să se analizeze sintactic propoziția  $\$(a + (a + a) * a)\$$ , folosind modelul automatului finit cu stivă.



# Exemplu

Gramatica dată nu este o gramatică operator deoarece:

- ▶ are terminale inutile ( $E \rightarrow T$ ,  $T \rightarrow F$ )
- ▶ în partea dreaptă a unor reguli de producție există neterminale unul lângă celălalt ( $F \rightarrow SED$ )
- ▶ prin urmare, pentru a construi un analizor sintactic ascendent bazat pe relațiile de precedență operator, **gramatica trebuie transformată într-o gramatică operator**

# Exemplu

Gramatica modificată este  $G = \langle \Sigma, V_N, F, P \rangle$ , unde:

►  $\Sigma = \{+, *, (, ), a\}$

►  $V_N = \{F\}$

►  $F$

►  $P = \{$   
    1.  $F \rightarrow F + F$   
    2.  $F \rightarrow F * F$   
    3.  $F \rightarrow (F)$   
    4.  $F \rightarrow a$   
}

# Exemplu

Se calculează relațiile de precedență operator:

$$(1) + <_o FIRST \sim^+ (F)$$

$$(1) LAST \sim^+ (F) >_o +$$

$$(2) * <_o FIRST \sim^+ (F)$$

$$(2) LAST \sim^+ (F) >_o *$$

$$(3) (=)$$

$$(3) (<_o FIRST \sim^+ (F)$$

$$(3) LAST \sim^+ (F) >_o )$$

$$\$ <_o FIRST \sim^+ (F)$$

$$LAST \sim^+ (F) >_o \$$$

## Exemplu

Calculând mulțimile  $FIRST \sim^+$  și  $LAST \sim^+$  se obține:

$$LAST \sim^+ (F) = \{+, *, ), a\}$$

$$FIRST \sim^+ (F) = \{+, *, (, a\}$$

# Exemplu

Atunci, relațiile de precedență operator ar fi:

$$(1) + <_o FIRST \sim^+ (F) \quad \text{și } FIRST \sim^+ (F) = \{+, *, (, a\}$$

$$(1) LAST \sim^+ (F) >_o + \quad \text{și } LAST \sim^+ (F) = \{+, *, ), a\}$$

$$\blacktriangleright + <_o + \quad + >_o +$$

$$\blacktriangleright + <_o * \quad * >_o +$$

$$\blacktriangleright + <_o ( \quad ) >_o +$$

$$\blacktriangleright + <_o a \quad a >_o +$$

# Exemplu

$$(2) * <_o FIRST \sim^+(F)$$

$$\text{și } FIRST \sim^+(F) = \{+, *, (, a\}$$

$$(2) LAST \sim^+(F) >_o *$$

$$\text{și } LAST \sim^+(F) = \{+, *, ), a\}$$

$$\blacktriangleright * <_o + \quad + >_o *$$

$$\blacktriangleright * <_o * \quad * >_o *$$

$$\blacktriangleright * <_o ( \quad ) >_o *$$

$$\blacktriangleright * <_o a \quad a >_o *$$

# Exemplu

►  $(=_o)$

(3)  $(<_o FIRST \sim^+ (F))$

și  $FIRST \sim^+ (F) = \{+, *, (, a\}$

(3)  $LAST \sim^+ (F) >_o)$

și  $LAST \sim^+ (F) = \{+, *, ), a\}$

►  $(<_o + \quad + >_o)$

►  $(<_o * \quad * >_o)$

►  $(<_o ( \quad ) >_o)$

►  $(<_o a \quad a >_o)$

# Exemplu

$$\$ <_o FIRST \sim^+ (F)$$

$$\text{și } FIRST \sim^+ (F) = \{+, *, (, a\}$$

$$LAST \sim^+ (F) >_o \$$$

$$\text{și } LAST \sim^+ (F) = \{+, *, ), a\}$$

$$\blacktriangleright \$ <_o + \quad + >_o \$$$

$$\blacktriangleright \$ <_o * \quad * >_o \$$$

$$\blacktriangleright \$ <_o ( \quad ) >_o \$$$

$$\blacktriangleright \$ <_o a \quad a >_o \$$$



## Exemplu

Se construiește matricea de precedență:

	+	*	(	)	a	\$
+	<,>	<,>	<	>	<	>
*	<,>	<,>	<	>	<	>
(	<	<	<	=	<	
)	>	>		>		>
a	>	>		>		>
\$	<	<	<		<	accept

# Ambiguitatea limbajelor

- ▶ dacă un limbaj  $L(G)$  este un **limbaj ambiguu**, atunci există propoziții ale acestui limbaj pentru care **se pot construi doi sau mai mulți arbori sintactici distincți**
- ▶ dacă un limbaj  $L(G)$  este ambiguu, atunci nu este obligatoriu ca pentru toate propozițiile sale să se poată construi mai mulți arbori sintactici distincți
- ▶ în cazul unui limbaj ambiguu, calculul relațiilor de precedență operator în baza gramaticii care îl definește va duce la situația:
  - ▶ **între aceleași două terminale vor exista două relații de precedență operator diferite**

# Ambiguitatea limbajelor

- ▶ pentru propoziția "**a+a+a**" se pot construi doi arbori sintactici distincți:

$+ >_o +$  - asociativitate stânga

$+ <_o +$  - asociativitate dreapta

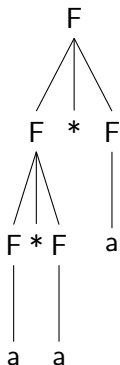


# Ambiguitatea limbajelor

- ▶ la fel, pentru propoziția "**a\*a\*a**" se pot construi doi arbori sintactici distincți:

$* >_o *$  - asociativitate stânga

$* <_o *$  - asociativitate dreapta



# Ambiguitatea limbajelor

- ▶ de asemenea, pentru propoziția "**a+a\*a**" se pot construi doi arbori sintactici distincți:

$+ >_o *$  - adunarea are  $+ <_o *$  - înmulțirea are  
prioritate prioritate



# Ambiguitatea limbajelor

- ▶ în aceste cazuri, trebuie stabilită care este precedența și asociativitatea operatorilor
- ▶ pentru acest exemplu, se va stabili că:
  - ▶ **adunarea este asociativă la stânga**
    - ▶ se va păstra relația  $+ >_o +$
  - ▶ **înmulțirea este asociativă la stânga**
    - ▶ se va păstra relația  $* >_o *$
  - ▶ **înmulțirea are precedența mai mare decât adunarea**
    - ▶ se vor păstra relațiile  $+ <_o *$  și  $* >_o +$

## Exemplu

Prin urmare, matricea de precedență va fi:

	+	*	(	)	a	\$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	<	<	<	=	<	
)	>	>		>		>
a	>	>		>		>
\$	<	<	<		<	accept

## Exemplu

$$(q, \$, (a+(a+a)^*a)\$, \epsilon)$$

$$\vdash^d (q, \$, (a+(a+a)^*a)\$, \epsilon)$$

$$\vdash^d (q, \$, (a, +(a+a)^*a)\$, \epsilon)$$

$$\vdash^r (q, \$, (F, +(a+a)^*a)\$, 4)$$

$$\vdash^d (q, \$, (F+, (a+a)^*a)\$, 4)$$

$$\vdash^d (q, \$, (F+, (a, +a)^*a)\$, 4)$$

$$\vdash^d (q, \$, (F+(a, +a)^*a)\$, 4)$$

$$\vdash^r (q, \$, (F+(F, +a)^*a)\$, 44)$$



## Exemplu

$$\vdash^d (q, \$(F+(F+,a)*a)\$,44)$$

$$\vdash^d (q, \$(F+(F+a,)*a)\$,44)$$

$$\vdash^r (q, \$(F+(F+F,)*a)\$,444)$$

$$\vdash^r (q, \$(F+(F,)*a)\$,4441)$$

$$\vdash^d (q, \$(F+(F),*a)\$,4441)$$

$$\vdash^r (q, \$(F+F,*a)\$,44413)$$

$$\vdash^d (q, \$(F+F*,a)\$,44413)$$

$$\vdash^d (q, \$(F+F*a,)\$,44413)$$

$$\vdash^r (q, \$(F+F*F,)\$,444134)$$

## Exemplu

$\vdash^r (q, \$(F+F,)\$,4441342)$

$\vdash^r (q, \$(F+F,)\$,4441342)$

$\vdash^r (q, \$(F,)\$,44413421)$

$\vdash^d (q, \$(F,)\$,44413421)$

$\vdash^r (q, \$F,\$,444134213)$

$\vdash^r (a, \$F,\$,444134213)$

## Exercițiu

Fie gramatica  $G = \langle \Sigma, V_N, \langle \text{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{\text{for, to, step, let, call, i, (, ), ,, =, *, **}\}$
- ▶  $V_N = \{\langle \text{instructiune} \rangle, \langle \text{atribuire} \rangle, \langle \text{lista} \rangle, \langle \text{expresie} \rangle\}$
- ▶  $\langle \text{instructiune} \rangle$
- ▶  $P = \{$ 
  - $\langle \text{instructiune} \rangle \rightarrow \text{for } \langle \text{atribuire} \rangle \text{ to } \langle \text{expresie} \rangle [\text{step } \langle \text{expresie} \rangle]$
  - $\langle \text{instructiune} \rangle \rightarrow \text{let } \langle \text{atribuire} \rangle$
  - $\langle \text{instructiune} \rangle \rightarrow \text{call } i \text{ } (\langle \text{lista} \rangle)$
  - $\langle \text{lista} \rangle \rightarrow i \mid i, \langle \text{lista} \rangle$
  - $\langle \text{atribuire} \rangle \rightarrow i = \langle \text{expresie} \rangle$
  - $\langle \text{expresie} \rangle \rightarrow i * \langle \text{expresie} \rangle \mid i ** \langle \text{expresie} \rangle \mid i$ $\}$

Să se construiască un analizor sintactic ascendent bazat pe relațiile de precedență operator.

# Analiză sintactică bazată pe precedența simplă (Wirth-Weber)

Nikolaus Wirth  
Inventatorul lui: Algol W, Euler,  
Modula, Oberon, Pascal



# Analiză sintactică bazată pe precedența simplă

## Relațiile de precedență simplă (Wirth-Weber)

$$<_w, =_w, >_w$$

Fie  $\alpha, \beta \in V_N \cup \Sigma$

$$(1) \quad \alpha =_w \beta \iff \exists A \rightarrow x\alpha\beta y \in P$$

$$(2) \quad \alpha <_w \beta \iff \exists A \rightarrow x\alpha B y \in P \wedge B \Rightarrow^+ \beta z$$

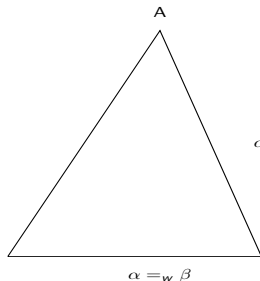
$$(3) \quad \alpha >_w \beta \iff \exists A \rightarrow xB\beta y \in P \wedge B \Rightarrow^+ z\alpha$$

sau

$$\iff \exists A \rightarrow xBCy \in P \wedge B \Rightarrow^+ z\alpha \wedge C \Rightarrow^+ \beta w$$

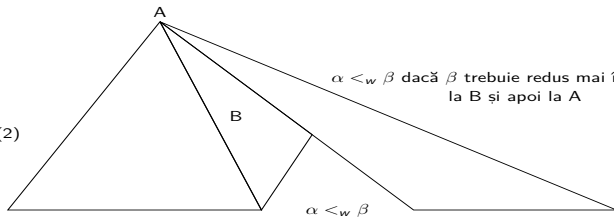
# Analiză sintactică bazată pe precedența simplă

(1)



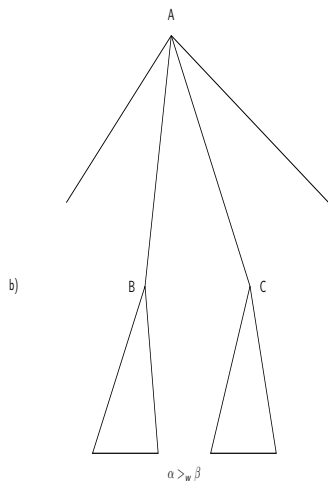
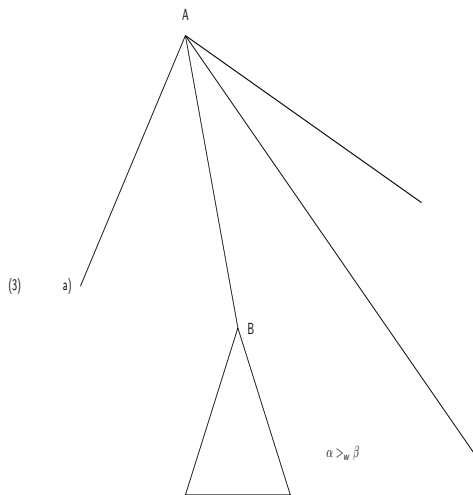
$\alpha =_w \beta$  dacă ele trebuie reduse în același timp la A

(2)



$\alpha <_w \beta$  dacă  $\beta$  trebuie redus mai întâi la B și apoi la A

# Analiză sintactică bazată pe precedența simplă



# Analiză sintactică bazată pe precedența simplă

## Observație:

Relațiile de prezență Wirth-Weber nu sunt simetrice, adică dacă  $\alpha >_w \beta$  nu înseamnă că și  $\beta <_w \alpha$ .

## Definiție:

- ▶ O gramatică Wirth-Weber este o gramatică independentă de context, fără reguli vide, reversibilă și în care orice pereche  $\alpha, \beta$  verifică cel mult una din relațiile  $<_w, =_w, >_w$ .
- ▶ O gramatică este reversibilă dacă nu există mai multe producții cu aceeași parte dreaptă, căci din  $A \rightarrow \alpha$  și  $B \rightarrow \alpha \Rightarrow A=B$ .
- ▶ Relațiile sunt disjuncte dacă între oricare două simboluri  $\alpha, \beta$  există cel mult o relație în același timp.



# Analiză sintactică bazată pe precedența simplă

## Calculul relațiilor Wirth-Weber:

(1)  $\alpha =_w \beta \rightarrow$  este o relație între toate simbolurile succesive (vecine) din partea dreaptă a regulilor de producție

(2)  $\alpha <_w \beta \iff A \rightarrow x\alpha By \in P \wedge \beta \in FIRST^+(B)$

(3)  $\alpha >_w \beta \iff A \rightarrow xB\beta y \in P \wedge \alpha \in LAST^+(B)$

$\vee A \rightarrow xBCy \in P, \alpha \in LAST^+(B), \beta \in FIRST^+(C)$

# Analiză sintactică bazată pe precedență simplă

## Observație:

În urma calculului relațiilor de precedență simplă pot apărea **conflicte**:

- (1)  $\alpha =_w \beta$  se transformă în  $<_w$  :  $A \rightarrow x\alpha\beta y$   
 $\alpha <_w \beta$   $A \rightarrow x\alpha D y$  și  $D \Rightarrow^+ \beta z$
- (2)  $\alpha =_w \beta$  se transformă în  $>_w$  :  $A \rightarrow x\alpha\beta y$   
 $\alpha >_w \beta$   $A \rightarrow xC\beta y$  și  $C \Rightarrow^+ z\alpha$

# Analiză sintactică bazată pe precedența simplă - Exemplu

Fie gramatica:

$$1. S \rightarrow aSAb$$

$$2. S \rightarrow d$$

$$3. A \rightarrow S$$

$$4. A \rightarrow c$$

Calculul relațiilor de precedență simplă:

$=_w$

$$a =_w S$$

$$S =_w A$$

$$A =_w b$$

# Analiză sintactică bazată pe precedență simplă - Exemplu

Fie gramatica:

1.  $S \rightarrow aSAb$

2.  $S \rightarrow d$

3.  $A \rightarrow S$

4.  $A \rightarrow c$

Calculul relațiilor de precedență simplă:

$<_w$

$$a <_w FIRST^+(S)$$

$$S <_w FIRST^+(A)$$

$$\$ <_w FIRST^+(S)$$

$$(\$ S \$)$$

$$FIRST^+(S) = \{a, d\}$$

$$FIRST^+(A) = \{a, c, d, S\}$$

# Analiză sintactică bazată pe precedența simplă - Exemplu

Fie gramatica:

1.  $S \rightarrow aSAb$

2.  $S \rightarrow d$

3.  $A \rightarrow S$

4.  $A \rightarrow c$

Calculul relațiilor de precedență simplă:

$>_w$

$$LAST^+(S) >_w FIRST^+(A)$$

$$LAST^+(A) >_w \{b\}$$

$$LAST^+(S) >_w \$$$

$$(\$ S \$)$$

$$\{b, d\} >_w \{a, c, d, S\}$$

$$\{S, c, b, d\} >_w \{b\}$$

$$\{b, d\} >_w \$$$

# Analiză sintactică bazată pe precedență simplă - Exemplu

	S	A	a	b	c	d	\$
S	$<_w$	$=_w$	$<_w$	$>_w$	$<_w$	$<_w$	
A				$=_w$			
a	$=_w$		$<_w$			$<_w$	
b	$>_w$		$>_w$	$>_w$	$>_w$	$>_w$	$>_w$
c				$>_w$			
d	$>_w$		$>_w$	$>_w$	$>_w$	$>_w$	$>_w$
\$			$<_w$			$<_w$	

# Analiză sintactică bazată pe precedența simplă

**Algoritmul de analiză sintactică bazat pe relațiile de precedență simplă este de tip deplasare - reducere**

- ▶  $(q, \alpha a, bx, \beta) \vdash^d (q, \alpha ab, x, \beta)$   
 $\iff f(a, b) = \text{depl}$
- ▶  $(q, \alpha c\gamma a, bx, \beta) \vdash^r (q, \alpha cA, bx, \beta i)$   
 $\iff f(a, b) = \text{red}, g(c\gamma a) = i, i: A \rightarrow \gamma a \in P$

# Analiză sintactică bazată pe precedența simplă

f - funcția de deplasare-reducere

$f: (V_N \cup \Sigma) \times \Sigma \rightarrow \{\underline{\text{depl}}, \underline{\text{red}}, \underline{\text{accept}}, \underline{\text{error}}\}$

- ▶  $f(\alpha, a) = \underline{\text{depl}} \iff \alpha <_w a \text{ sau } \alpha =_w a$
- ▶  $f(\alpha, a) = \underline{\text{red}} \iff \alpha >_w a$
- ▶  $f(\alpha, a) = \underline{\text{error}} \iff$  între  $\alpha$  și  $a$  nu există relație de precedență simplă
- ▶  $f(\$S, \$) = \underline{\text{accept}}$



# Analiză sintactică bazată pe precedența simplă

$g$  - funcția de reducere

$g : (V_N \cup \Sigma)^* \rightarrow \{1, 2, 3, \dots\}$  unde:

- ▶  $(V_N \cup \Sigma)^*$  - vârful stivei
- ▶  $\{1, 2, 3, \dots\}$  - indicele regulilor de producție

$$g(\beta x_1 x_2 \dots x_n) = i \iff \exists i : A \rightarrow x_1 x_2 \dots x_n \in P \wedge \beta <_w x_1$$

# Analiză sintactică bazată pe precedența simplă - Exemplu

Valorile funcția  $f$  rezultă imediat din matricea relațiilor de precedență simplă.

e.g.

$$f(S, a) = \text{depl}$$

$$f(S, b) = \text{red}$$

Valorile funcției  $g$  rezultă imediat din regulile de producție.

e.g.

$$g(\text{SaSAb}) = 1$$

$$g(\text{aaSAb}) = 1$$

$$g(\$ \text{aSAb}) = 1$$

$$g(\text{Sd}) = 2$$

$$g(\text{ad}) = 2$$

$$g(\$ \text{d}) = 2$$

$$g(\text{SS}) = 3$$

$$g(\text{Sc}) = 4$$

# Analiză sintactică bazată pe precedență simplă - Exemplu

	S	A	a	b	c	d	\$
S	<	=	<	>	<	<	
A				=			
a	=		<			<	
b	>		>	>	>	>	>
c				>			
d	>		>	>	>	>	>
\$			<			<	

# Analiză sintactică bazată pe precedența simplă - Exemplu

$(q_0, \$, \text{adadcbb}\$, \epsilon)$

$\vdash^d (q, \$a, \text{dadcb}\$, 1)$

$\vdash^d (q, \$ad, \text{adcb}\$, 1)$

$\vdash^r (q, \$aS, \text{adcb}\$, 2)$

$\vdash^d (q, \$aSa, \text{dcb}\$, 2)$

$\vdash^d (q, \$aSad, \text{cbb}\$, 2)$

$\vdash^r (q, \$aSaS, \text{cbb}\$, 22)$

$\vdash^d (q, \$aSaSc, \text{bb}\$, 22)$

$\vdash^r (q, \$aSaSA, \text{bb}\$, 224)$

$\vdash^d (q, \$aSaSAb, \text{b}\$, 224)$

$\vdash^r (q, \$aSS, \text{b}\$, 2241)$

$\vdash^r (q, \$aSA, \text{b}\$, 22413)$

$\vdash^d (q, \$aSAb, \$, 22413)$

$\vdash^r (q, \$S, \$, 224131)$

# Analiză sintactică bazată pe precedența simplă

Algoritmul de analiză este următorul:

```

{
    step  $\leftarrow$  false
    ISI; ISO; IST  $\leftarrow$  1
    ST(IST) = $
    {
        repeta
        {
            {
                daca  $f(ST(IST), ST(IST)) = \text{deplaseaza}$ 
                atunci
                    IST  $\leftarrow$  IST + 1
                    ST(IST)  $\leftarrow$  SI(ISI)
                    ISI  $\leftarrow$  ISI + 1
            }
            ori daca  $f(ST(IST), SI(ISI)) = \text{reducere}$ 
            atunci
                * cauta in jos in stiva doua simboluri  $x, y$  a.i  $x <_w y$ 
                * redu sirul  $y\alpha$  si inlocuieste varful stivei conform
                     $g(xy\alpha)$  ( $i : A \rightarrow y\alpha$ )
                    SO(ISO)  $\leftarrow$   $g(xy\alpha) = i$ 
                    ISO  $\leftarrow$  ISO + 1
            }
            ori daca  $f(ST(IST), SI(ISI)) = \text{accept}$ 
            atunci
                * OK
                step  $\leftarrow$  true
            ori daca  $f(ST(IST), SI(ISI)) = \text{error}$ 
            atunci
                * NOT OK
                step  $\leftarrow$  true
        }
    }
    până când step este true
}
```

# Analiză sintactică bazată pe precedența simplă - Exercițiu

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow a$
6.  $F \rightarrow a(L)$
7.  $F \rightarrow (E)$
8.  $L \rightarrow E$
9.  $L \rightarrow E, L$

# Analiză sintactică bazată pe precedența slabă (Ichbiach - Morse)

$$\alpha, \beta \in V_N \cup \Sigma$$

$$(1) \alpha <_I \beta \iff \exists (A \rightarrow x\alpha\beta y \in P) \vee (A \rightarrow x\alpha B y \in P),$$

unde:

$$B \Rightarrow^+ \beta \gamma$$

$$(2) \alpha >_I \beta \iff \exists (A \rightarrow xB\beta y \in P) \vee (A \rightarrow xBCy \in P),$$

unde:

$$B \Rightarrow^+ \gamma \alpha$$

$$C \Rightarrow^+ \beta z$$

# Analiză sintactică bazată pe precedența slabă (Ichbiach - Morse)

## Gramatici cu precedența slabă (Ichbiach - Morse)

O gramatică cu precedență slabă (Ichbiach - Morse) este o gramatică independentă de context, fără reguli vide și fără variabile inutile și care îndeplinește următoarele două condiții:

- (1)  $<_l$  și  $>_l$  sunt disjuncte
- (2) Pentru regulile de producție de forma  $A \rightarrow x\alpha y$ ,  $B \rightarrow y$  nu există o relație între  $\alpha$  și **B** (imbricare ilegală)



# Analiză sintactică bazată pe precedența slabă

Conflicte posibile:

$$\alpha <_I \beta \text{ si } \alpha >_I \beta \\ \iff \exists A \rightarrow x\alpha\beta y \vee A \rightarrow xB\beta y \text{ ( } B \Rightarrow^+ z\alpha \text{)}$$

Rezolvare:

Se introduce un neterminal  $C$ , o regula de producție  $C \rightarrow \alpha$  și se fac înlocuirile:

$$A \rightarrow xC\beta y$$

sau respectiv

$$A \rightarrow xCB y$$

rezultând

$$\alpha >_I \beta$$

# Analiză sintactică bazată pe precedența slabă

**Algoritmul de analiza este de tipul deplasare-reducere**

f

$f(\alpha, a) = \underline{\text{depl}}, \alpha <_I a$

$f(\alpha, a) = \underline{\text{red}}, \alpha >_I a$

$f(\$S, \$) = \underline{\text{accept}}$

$f(\alpha, a) = \underline{\text{error}} \iff (\alpha, a) \notin R$ , unde R este multimea relatiilor de precedență slabă

# Analiză sintactică bazată pe precedența slabă

$g$

$$g(\alpha\beta)=i$$

$\iff$

$(\exists) i: B \rightarrow \beta \in P$  (se alege regula de producție cu partea dreapta cea mai lungă, care are sufixul  $\beta$  )

$\wedge$

$(\nexists) j: A \rightarrow x\alpha\beta \in P, j > i$

Observatie:

Regulile de productie sunt ordonate descrescator in functie de lungimea partii drepte.

# Analiză sintactică bazată pe precedența slabă - Exemplu

Fie gramatica:

1.  $E \rightarrow E+T$
2.  $E \rightarrow T$
3.  $T \rightarrow T*F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow a$

# Analiză sintactică bazată pe precedența slabă - Exemplu

$<_I$

$E <_I +$

$+ <_I T$

$+ <_I \text{FIRST}^+(T) = \{ T, F, (, a \}$

$T <_I *$

$* <_I F$

$* <_I \text{FIRST}^+(F) = \{ (, a \}$

$( <_I E$

$( <_I \text{FIRST}^+(E) = \{ E, T, F, (, a \}$

$E <_I )$

$\$ <_I \text{FIRST}^+(E)$

# Analiză sintactică bazată pe precedența slabă - Exemplu

$>_I$

$\text{LAST}^+(E) >_I +, \text{LAST}^+(E) = \{ T, F, ), a \}$

$\text{LAST}^+(T) >_I *, \text{LAST}^+(T) = \{ F, ), a \}$

$\text{LAST}^+(E) >_I )$

$\text{LAST}^+(E) >_I \$$

# Analiză sintactică bazată pe precedența slabă - Exemplu

Analiza conflictelor:

$$E \rightarrow E \underline{+} T$$

$$\underline{E} \rightarrow \underline{T}$$

$$T \rightarrow T \underline{*} F$$

$$\underline{T} \rightarrow \underline{F}$$

- ▶ verificare pentru ( +, E ) - nu exista nicio relatie
- ▶ verificare pentru ( \*, T )  $\Rightarrow$  nu exista nicio relatie

Rezulta deci că nu exista imbricari ilegale.

# Analiză sintactică bazată pe precedența slabă - Exemplu

	E	T	F	a	(	)	+	*	\$
E						< <sub>I</sub>	< <sub>I</sub>		
T						> <sub>I</sub>	> <sub>I</sub>	< <sub>I</sub>	> <sub>I</sub>
F						> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>
a						> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>
(	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>				
)						> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>	> <sub>I</sub>
+		< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>				
*			< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>				
\$	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>	< <sub>I</sub>				



# Analiză sintactică bazată pe precedența slabă - Exemplu

**f()**

$f( E, ) ) = \text{deplasare}$

$f( E, + ) = \text{deplasare}$

$f( T, ) ) = \text{reducere}$

...

**g()**

$g( E+T ) = 1$

$g( T * F ) = 3$

$g( ( E ) ) = 5$

$g( T ) = 2$

$g( F ) = 4$

$g( a ) = 6$

## Analiză sintactică bazată pe precedența slabă - Exemplu

$(q_0, \$, a^*(a+a)\$, \epsilon)$   
 $\vdash^d (q, \$a, *(a+a)\$, \epsilon)$   
 $\vdash^r (q, \$F, *(a+a)\$, 6)$   
 $\vdash^r (q, \$T, *(a+a)\$, 64)$   
 $\vdash^d (q, \$T^*, (a+a)\$, 64)$   
 $\vdash^d (q, \$T^*(, a+a)\$, 64)$   
 $\vdash^d (q, \$T^*(a, +a)\$, 64)$   
 $\vdash^r (q, \$T^*(F, +a)\$, 646)$   
 $\vdash^r (q, \$T^*(T, +a)\$, 6464)$   
 $\vdash^r (q, \$T^*(E, +a)\$, 64642)$   
 $\vdash^d (q, \$T^*(E+, a)\$, 64642)$

# Analiză sintactică bazată pe precedența slabă - Exemplu

$\vdash^d (q, \$T^*(E+a,)\$,64642)$   
 $\vdash^r (q, \$T^*(E+F,)\$,646426)$   
 $\vdash^r (q, \$T^*(E+T,)\$,6464264)$   
 $\vdash^r (q, \$T^*(E,)\$,64642641)$   
 $\vdash^d (q, \$T^*(E,)\$,64642641)$   
 $\vdash^r (q, \$T^*F,\$,646426415)$   
 $\vdash^r (q, \$T,\$,6464264153)$   
 $\vdash^r (q, \$E,\$,64642641532)$   
 $\vdash^a$

Sirul reducerilor stanga 64642641532

Sirul derivarilor dreapta 23514624646

# Analiză sintactică bazată pe precedența slabă - Exercițiu

1.  $E \rightarrow E+T$
2.  $E \rightarrow T$
3.  $T \rightarrow T*F$
4.  $T \rightarrow F$
5.  $F \rightarrow a$
6.  $F \rightarrow a(L)$
7.  $F \rightarrow (E)$
8.  $L \rightarrow E$
9.  $L \rightarrow E,L$

## Întrebări recapitulative

- ▶ Fie  $AP1 = (Q, \Sigma, \Gamma, f, q_0, z_0, F)$  un automat finit cu stivă care implementează analiza sintactică descendentă pentru gramatica  $G = (\Sigma, V_N, S, P)$  și fie  $AP2 = (Q', \Sigma', \Gamma', f', q'_0, z'_0, F')$  un automat finit cu stivă care implementează analiza sintactică ascendentă pentru aceeași gramatică. Arătați care sunt diferențele dintre elementele care definesc AP1 și cele care îl definesc pe AP2 și explicați.
- ▶ Explicați de ce este nedeterminist algoritmul general de analiză sintactică ascendentă.
- ▶ Explicați rolul relației de precedență operator mai mare în analiza sintactică corespunzătoare.

# Întrebări recapitulative

- ▶ Enumerați și explicați diferențele dintre automatul finit cu stivă definit pentru analiza sintactică descendentă și automatul finit cu stivă definit pentru analiza sintactică ascendentă.
- ▶ Explicați care este diferența dintre configurația inițială a automatului finit cu stivă pentru analiza sintactică LL(1) și pentru analiza sintactică bazată pe precedența operator.
- ▶ Explicați cum rezolvă analiza sintactică bazată pe relațiile de precedență operator nedeterminismul algoritmului generic de analiză sintactică ascendentă.

# Alte gramatici

$G = \langle \Sigma, V_N, \langle IFlogic \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{n, IF, (, ), GOTO, i, =, +, .EQ.\}$
- ▶  $V_N = \{ \langle IFlogic \rangle, \langle IFneetichetat \rangle, \langle expresie - logica \rangle, \langle instructiune \rangle, \langle expresie - aritmetica \rangle, \langle operator - logic \rangle \}$
- ▶  $\langle IFlogic \rangle$
- ▶  $P = \{$ 
  - $\langle IFlogic \rangle \rightarrow n \langle IFneetichetat \rangle \mid \langle IFneetichetat \rangle$
  - $\langle IFneetichetat \rangle \rightarrow IF(\langle expresie - logica \rangle) \langle instructiune \rangle$
  - $\langle instructiune \rangle \rightarrow GOTO n$
  - $\langle instructiune \rangle \rightarrow i = n \langle expresie - aritmetica \rangle$
  - $\langle expresie - logica \rangle \rightarrow \langle expresie - aritmetica \rangle \langle operator - logic \rangle \langle expresie - aritmetica \rangle$
  - $\langle expresie - aritmetica \rangle \rightarrow \langle expresie - aritmetica \rangle + i \mid i$
  - $\langle operator - logic \rangle \rightarrow .EQ.$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle \text{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{ \text{if}, :, i, :=, \text{next}, \text{sentence}, +, \text{is}, <, >, ,, \}$
- ▶  $V_N = \{ \langle \text{instructiune} \rangle, \langle \text{cond} \rangle, \langle \text{instructiune1} \rangle, \langle \text{operand} \rangle, \langle \text{oprel} \rangle, \langle \text{operator} \rangle, \langle \text{oprel} \rangle, \langle \text{zona} \rangle \}$
- ▶  $\langle \text{instructiune} \rangle$
- ▶  $P = \{$ 
  - $\langle \text{instructiune} \rangle \rightarrow \langle \text{iif} \rangle \mid \langle \text{imove} \rangle$
  - $\langle \text{iif} \rangle \rightarrow \text{if} \langle \text{cond} \rangle; \langle \text{instructiune1} \rangle [\text{else} \langle \text{instructiune1} \rangle]$
  - $\langle \text{instructiune1} \rangle \rightarrow i := \langle \text{operand} \rangle \mid \text{next sentence}$
  - $\langle \text{cond} \rangle \rightarrow \langle \text{operand} \rangle \langle \text{operator} \rangle \langle \text{operand} \rangle$
  - $\langle \text{operand} \rangle \rightarrow i \mid \langle \text{operand} \rangle + i$
  - $\langle \text{operator} \rangle \rightarrow \text{is} \langle \text{oprel} \rangle \mid \langle \text{oprel} \rangle$
  - $\langle \text{oprel} \rangle \rightarrow < \mid >$
  - $\langle \text{imove} \rangle \rightarrow \text{move} \langle \text{zona} \rangle \mid \text{to} \langle \text{zona} \rangle$
  - $\langle \text{zona} \rangle \rightarrow i \mid \langle \text{zona} \rangle, i$ $\}$



# Alte gramatici

$G = \langle \Sigma, V_N, \langle \textit{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{\textit{if}, \textit{then}, \textit{else}, i, :=, .\textit{or.}, .\textit{and.}, (, ), .\textit{not.}\}$
- ▶  $V_N = \{\langle \textit{instructiune} \rangle, \langle \textit{expresie} \rangle, \langle \textit{atribuire} \rangle, \langle \textit{termen} \rangle, \langle \textit{factor} \rangle\}$
- ▶  $\langle \textit{instructiune} \rangle$
- ▶  $P = \{$ 
  - $\langle \textit{instructiune} \rangle \rightarrow \textit{if} \langle \textit{expresie} \rangle \textit{ then} \langle \textit{atribuire} \rangle [\textit{else} \langle \textit{atribuire} \rangle]$
  - $\langle \textit{atribuire} \rangle \rightarrow i := \langle \textit{expresie} \rangle$
  - $\langle \textit{expresie} \rangle \rightarrow \langle \textit{expresie} \rangle .\textit{or.} \langle \textit{termen} \rangle \mid \langle \textit{termen} \rangle$
  - $\langle \textit{termen} \rangle \rightarrow \langle \textit{termen} \rangle .\textit{and.} \langle \textit{factor} \rangle \mid \langle \textit{factor} \rangle$
  - $\langle \textit{factor} \rangle \rightarrow i \mid (\langle \textit{expresie} \rangle) \mid .\textit{not.} \langle \textit{factor} \rangle$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle \text{instructiune} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{ \text{if, then, else, =, +, i, (, ), , } \}$
- ▶  $V_N = \{ \langle \text{instructiune} \rangle, \langle \text{expresie} \rangle, \langle \text{atribuire} \rangle, \langle \text{variabila} \rangle, \langle \text{lista} \rangle \}$
- ▶  $\langle \text{instructiune} \rangle$
- ▶  $P = \{$ 
  - $\langle \text{instructiune} \rangle \rightarrow \text{if } \langle \text{expresie} \rangle \text{ then } \langle \text{atribuire} \rangle [\text{else } \langle \text{atribuire} \rangle] \mid \langle \text{atribuire} \rangle$
  - $\langle \text{atribuire} \rangle \rightarrow \langle \text{variabila} \rangle = \langle \text{expresie} \rangle$
  - $\langle \text{expresie} \rangle \rightarrow \langle \text{variabila} \rangle \mid \langle \text{expresie} \rangle + \langle \text{variabila} \rangle$
  - $\langle \text{variabila} \rangle \rightarrow i \mid i(\langle \text{lista} \rangle)$
  - $\langle \text{lista} \rangle \rightarrow i \mid \langle \text{lista} \rangle, i$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle instr.DO \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{ DO, n, i, =, ,, \}$
- ▶  $V_N = \{ \langle instr.DO \rangle, \langle et \rangle, \langle expresie \rangle, \langle var \rangle \}$
- ▶  $\langle instr.DO \rangle$
- ▶  $P = \{$ 
  - $\langle instr.DO \rangle \rightarrow \langle et \rangle DO \langle et \rangle \langle expresie \rangle$
  - $\langle instr.DO \rangle \rightarrow DO \langle et \rangle \langle expresie \rangle$
  - $\langle et \rangle \rightarrow n$
  - $\langle expresie \rangle \rightarrow i = \langle lista \rangle$
  - $\langle lista \rangle \rightarrow \langle var \rangle, \langle var \rangle, \langle var \rangle$
  - $\langle var \rangle \rightarrow i$
  - $\langle var \rangle \rightarrow n$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle \text{decl} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{ \text{LET}, i, =, +, -, *, /, \text{FOR}, \text{TO}, \text{STEP}, \text{CALL}, ,, (, ) \}$
- ▶  $V_N = \{ \langle \text{decl} \rangle, \langle \text{declLET} \rangle, \langle \text{declFOR} \rangle, \langle \text{declCALL} \rangle, \langle \text{expr} \rangle, \langle \text{term} \rangle, \langle \text{fact} \rangle, \langle \text{lista} \rangle \}$
- ▶  $\langle \text{decl} \rangle$
- ▶  $P = \{$ 
  - $\langle \text{decl} \rangle \rightarrow \langle \text{declLET} \rangle \mid \langle \text{declFOR} \rangle \mid \langle \text{declCALL} \rangle$
  - $\langle \text{declLET} \rangle \rightarrow \text{LET } i = \langle \text{expr} \rangle$
  - $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \mid \langle \text{expr} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle$
  - $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{fact} \rangle \mid \langle \text{term} \rangle / \langle \text{fact} \rangle \mid \langle \text{fact} \rangle$
  - $\langle \text{fact} \rangle \rightarrow i \mid (\langle \text{expr} \rangle)$
  - $\langle \text{declFOR} \rangle \rightarrow \text{FOR } i = \langle \text{expr} \rangle \text{ TO } \langle \text{expr} \rangle \text{ STEP } \langle \text{expr} \rangle$
  - $\langle \text{declCALL} \rangle \rightarrow \text{CALL } i(\langle \text{lista} \rangle)$
  - $\langle \text{lista} \rangle \rightarrow i \mid \langle \text{lista} \rangle, i$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle instr \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{CALL, i, ,, +, -, *, /, (, )\}$
- ▶  $V_N = \{ \langle instr \rangle, \langle parametrii \rangle, \langle expr \rangle, \langle term \rangle, \langle factor \rangle \}$
- ▶  $\langle instr \rangle$
- ▶  $P = \{$ 
  - $\langle instr \rangle \rightarrow CALL\ i(\langle parametrii \rangle)$
  - $\langle instr \rangle \rightarrow n\ CALL\ i(\langle parametrii \rangle)$
  - $\langle parametrii \rangle \rightarrow \langle expr \rangle \mid \langle parametrii \rangle, \langle expr \rangle$
  - $\langle expr \rangle \rightarrow \langle term \rangle \mid \langle expr \rangle + \langle term \rangle \mid \langle expr \rangle - \langle term \rangle$
  - $\langle term \rangle \rightarrow \langle factor \rangle \mid \langle term \rangle * \langle factor \rangle \mid \langle term \rangle / \langle factor \rangle$
  - $\langle factor \rangle \rightarrow i \mid i(\langle parametrii \rangle)$ $\}$

# Alte gramatici

$G = \langle \Sigma, V_N, \langle \textit{prog} \rangle, P \rangle$ , unde:

- ▶  $\Sigma = \{ \text{PROGRAM, VAR, BEGIN, END, id, ,, ::, INTEGER, ,, :=, +, -, *, DIV, int, (, ), READ, WRITE, FOR, DO, TO} \}$
- ▶  $V_N = \{ \langle \textit{prog} \rangle, \langle \textit{prog} - \textit{name} \rangle, \langle \textit{dec} - \textit{list} \rangle, \langle \textit{stmt} - \textit{list} \rangle, \langle \textit{dec} \rangle, \langle \textit{id} - \textit{list} \rangle, \langle \textit{type} \rangle, \langle \textit{stmt} \rangle, \langle \textit{assign} \rangle, \langle \textit{read} \rangle, \langle \textit{write} \rangle, \langle \textit{for} \rangle, \langle \textit{exp} \rangle, \langle \textit{term} \rangle, \langle \textit{factor} \rangle \}$
- ▶  $\langle \textit{prog} \rangle$
- ▶  $P = \{$

# Alte gramatici

1.  $\langle prog \rangle \rightarrow PROGRAM \langle prog - name \rangle VAR \langle dec - list \rangle BEGIN \langle stmt - list \rangle END.$
2.  $\langle prog - name \rangle \rightarrow id$
3.  $\langle dec - list \rangle \rightarrow \langle dec \rangle | \langle dec - list \rangle ; \langle dec \rangle$
4.  $\langle dec \rangle \rightarrow \langle id - list \rangle : \langle type \rangle$
5.  $\langle type \rangle \rightarrow INTEGER$
6.  $\langle id - list \rangle \rightarrow id | \langle id - list \rangle , id$

# Alte gramatici

- 7.  $\langle \text{stmt} - \text{list} \rangle \rightarrow \langle \text{stmt} \rangle \mid \langle \text{stmt} - \text{list} \rangle ; \langle \text{stmt} \rangle$
- 8.  $\langle \text{stmt} \rangle \rightarrow \langle \text{assign} \rangle \mid \langle \text{read} \rangle \mid \langle \text{write} \rangle \mid \langle \text{for} \rangle$
- 9.  $\langle \text{assign} \rangle \rightarrow \text{id} := \langle \text{exp} \rangle$
- 10.  $\langle \text{exp} \rangle \rightarrow \langle \text{term} \rangle \mid \langle \text{exp} \rangle + \langle \text{term} \rangle \mid \langle \text{exp} \rangle - \langle \text{term} \rangle$
- 11.  $\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \mid \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{term} \rangle \text{DIV} \langle \text{factor} \rangle$
- 12.  $\langle \text{factor} \rangle \rightarrow \text{id} \mid \text{int} \mid (\langle \text{exp} \rangle)$



# Alte gramatici

- 13.  $\langle read \rangle \rightarrow READ(\langle id - list \rangle)$
  - 14.  $\langle write \rangle \rightarrow WRITE(\langle id - list \rangle)$
  - 15.  $\langle for \rangle \rightarrow FOR \langle index - exp \rangle DO \langle body \rangle$
  - 16.  $\langle index - exp \rangle \rightarrow id := \langle exp \rangle TO \langle exp \rangle$
  - 17.  $\langle body \rangle \rightarrow \langle stmt \rangle | BEGIN \langle stmt - list \rangle END$
- }

# Bibliografie

- ▶ R.B. Yehezkael, Course Notes on Formal Languages and Compilers, Jerusalem College of Technology  
<http://homedir.jct.ac.il/~rafi/formcomp.pdf>
- ▶ Paul N. Hilfinger, Course Notes, University of California, Berkeley  
<http://inst.eecs.berkeley.edu/~cs164/sp10/notes/notes.pdf>
- ▶ Gavrila Ionut, Limbaje Formale și Translatoare  
[http://facultate.regielive.ro/cursuri/calculatoare/limbaje\\_formale\\_si\\_translatoare-59028.html](http://facultate.regielive.ro/cursuri/calculatoare/limbaje_formale_si_translatoare-59028.html)
- ▶ CIS 324: Language Design and Implementation, Operator Precedence Parsing  
<http://homepages.gold.ac.uk/nikolaev/3246-2.doc>