

Compilatoare

Analiza sintactică - LR



Ce am facut pana acum

- Structura generala a unui compilator
- Analiza top-down (LL)
- Urmeaza:
 - Analiza bottom-up (LR)
 - Analiza semantica
 - Generare de cod
 - Optimizari

Sumar pt. cursul trecut

- Analiza descendenta – urmareste derivarea stanga
- Doua tipuri:
 - Descendent recursiva – parsere ‘de mana’; design pe baza de diagrame de tranzitii
 - Poate fi predictiva sau cu backtracking
 - **Cum?**
 - LL bazat pe tabele (construite cu FIRST,FOLLOW) – pentru generatoare de parsere (de ex. JavaCC, ANTLR)
 - In general, predictive ANTLR are backtracking
 - **Unde are backtracking?**
- Analizarele predictive nu functioneaza pe gramatici recursive stanga, nefactorizate, cu ambiguitati

Analiza descendente recursiva

Nonterminal

$A \rightarrow a_1 \mid a_2 \mid \dots \mid a_n$

```
A() {  
    if lookahead ∈ FIRST(a1 FOLLOW(A))  
        code for a1 ...  
    else if lookahead ∈ FIRST(a2 FOLLOW(A))  
        code for a2 ...  
    .  
    .  
    else if lookahead ∈ FIRST(an FOLLOW(A))  
        code for an ...  
    else ERROR();  
}
```

Terminal (sym)

```
MATCH(sym)() {  
    if (lookahead == sym)  
        lookahead = NEXTSYM();  
    else ERROR();  
}
```

Terminologie: LL vs LR

- LL(k)
 - Scaneaza intrarea "Left-to-right"
 - "Left-most derivation" – deriveaza mereu cel mai din stanga neterminal
 - k simboluri de lookahead
 - Face o traversare in pre-ordine a arborelui de parsare
- LR(k)
 - Scaneaza intrarea "Left-to-right"
 - "Right-most derivation" – 'deriveaza' cel mai din dreapta neterminal
 - k simboluri de lookahead
 - Face o traversare in post-ordine a arborelui de parsare

Algoritmi LL

- LL(1) – cauta 1 simbol
- LL(k) – cauta k simboluri (k=finit)
 - Full LL(k) vs. Strong LL(k)
- LL(*) - cauta un sir de simboluri definit de un AFD (limbaj regulat)
- Mai multe informatii
 - LL(*) - The definitive ANTLR reference
 - Grune,Jacobs - Parsing Techniques (www.cs.vu.nl/~dick)

ANALIZA ASCENDENTA

Parserul ascendent

Un parser ascendent, sau “parser shift-reduce”, incepe de la ‘frunze’ si construiește spre varf arborele de derivare

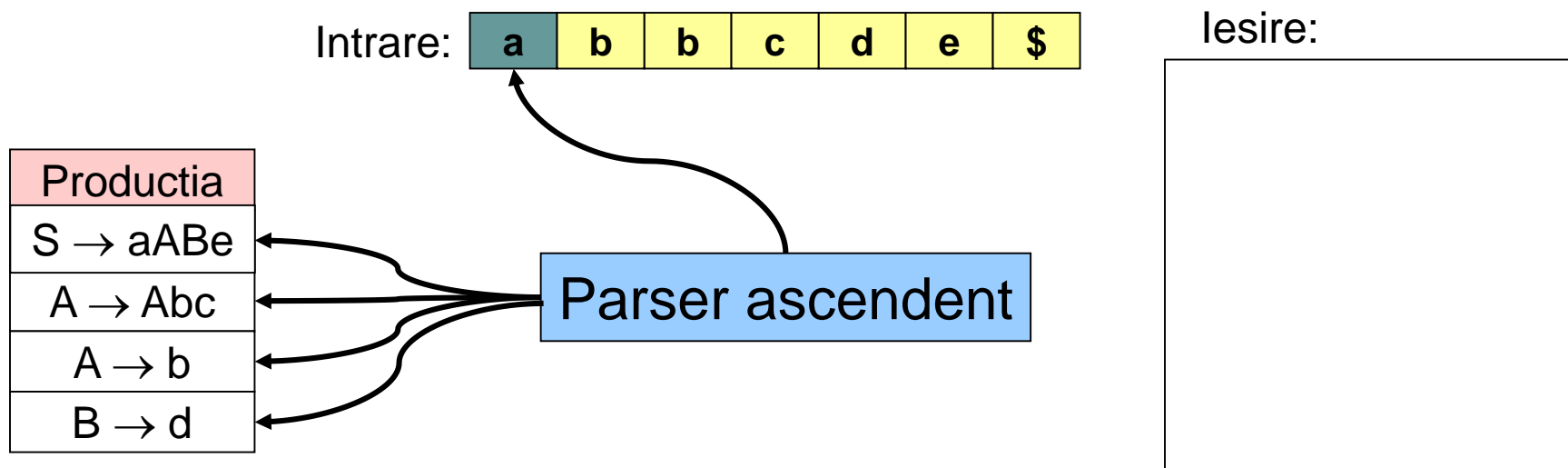
Pasii de reducere urmaresc o derivare dreapta in ordine inversa

Sa consideram GRAMATICA:

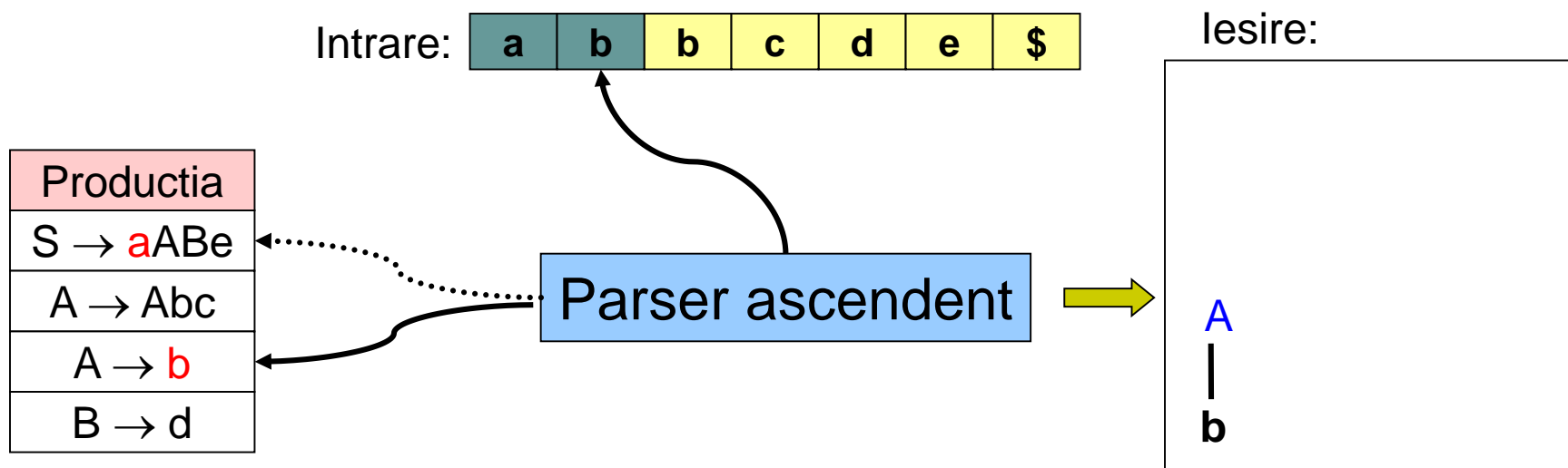
$$\begin{array}{l} S \rightarrow aABe \\ A \rightarrow Abc \mid b \\ B \rightarrow d \end{array}$$

Vrem sa parsam sirul de Intrare **abbcd**e.

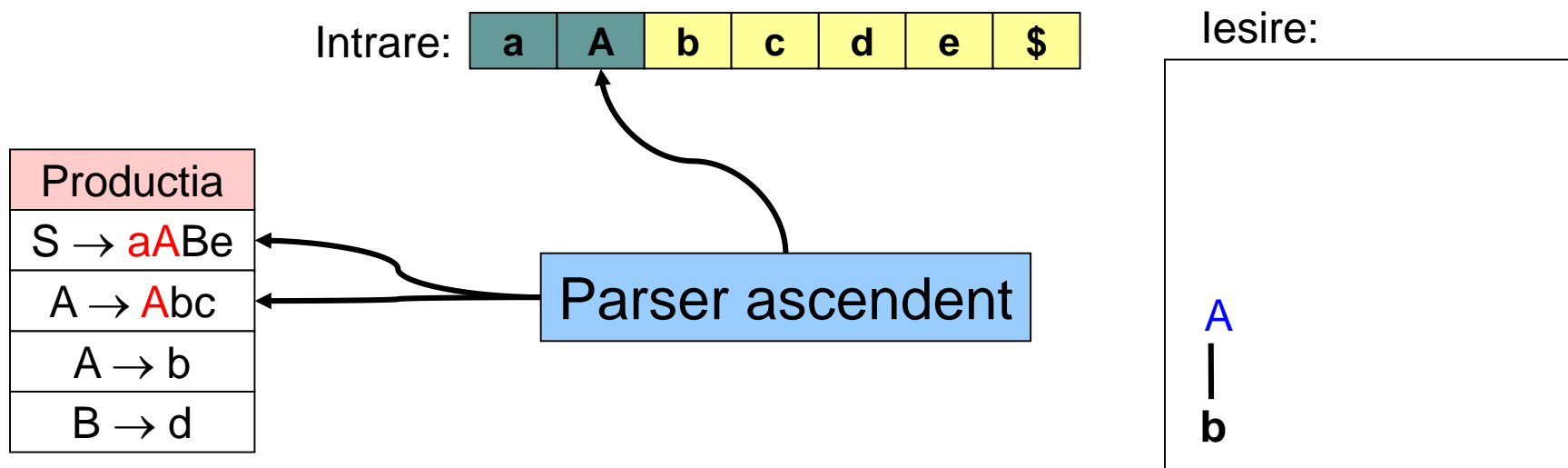
Exemplu de parser ascendent



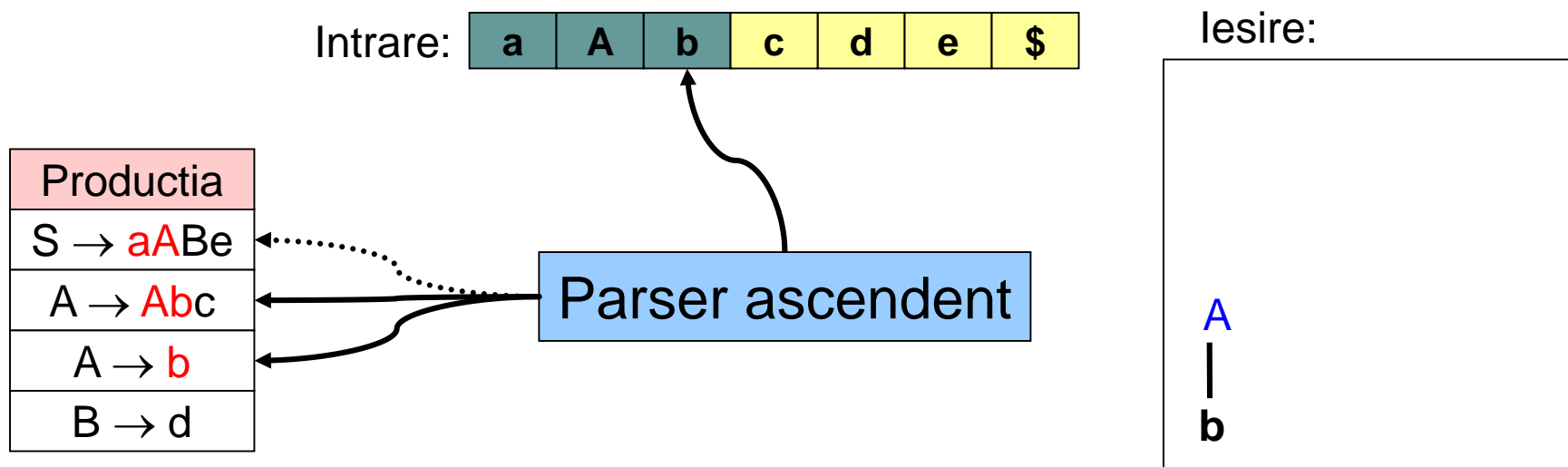
Exemplu de parser ascendent



Exemplu de parser ascendent

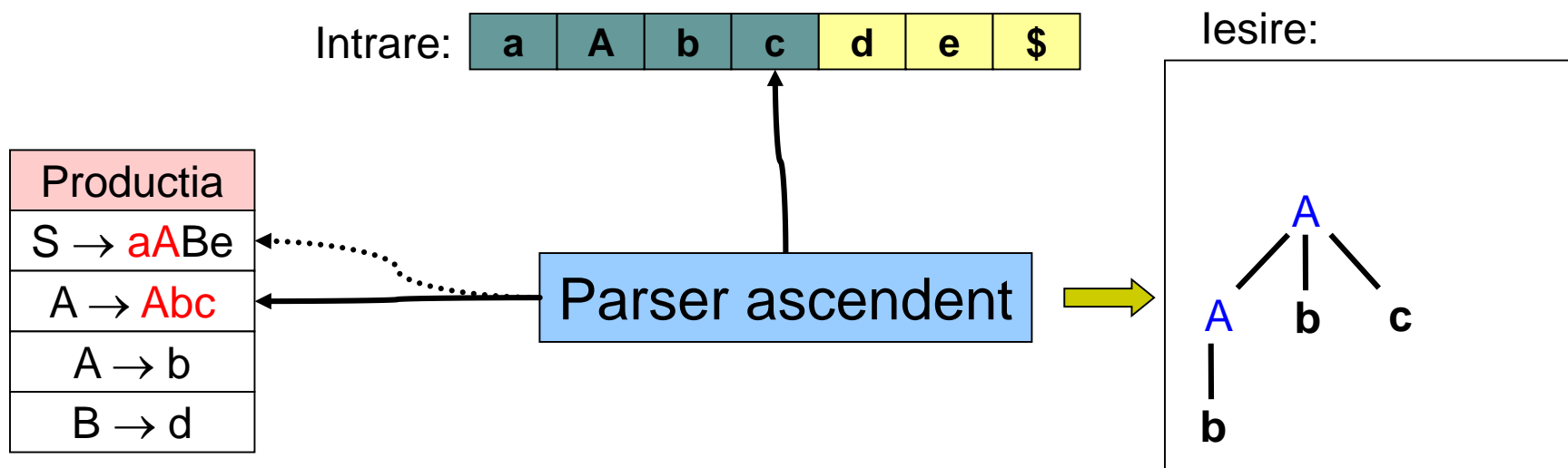


Exemplu de parser ascendent

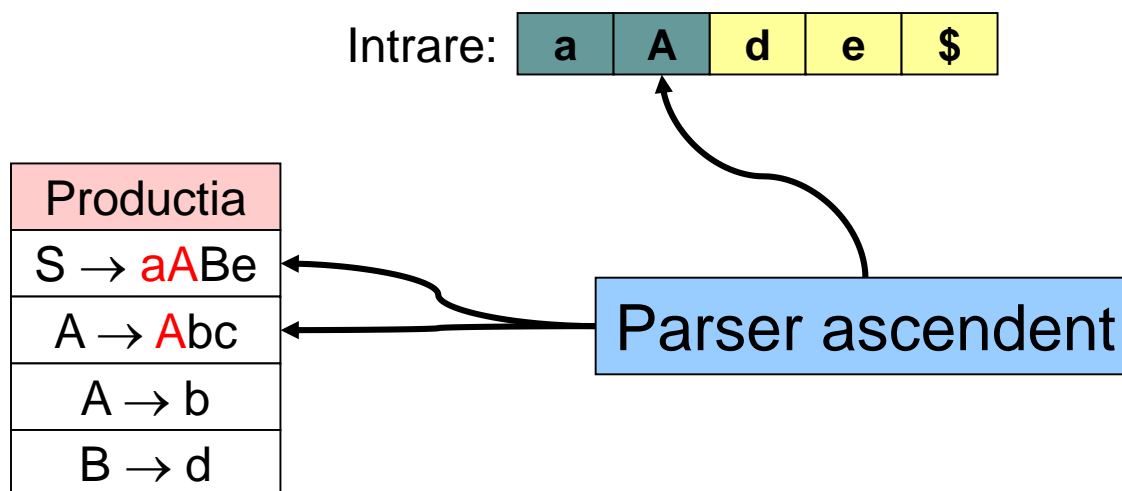


Nu reducem in acest exemplu. Un parser ar reduce, s-ar impotmoli si ar trebui sa faca backtracking!

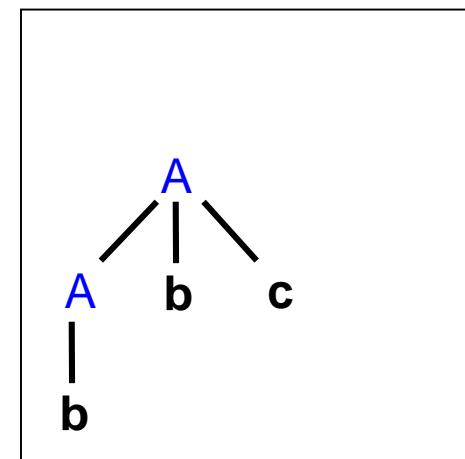
Exemplu de parser ascendent



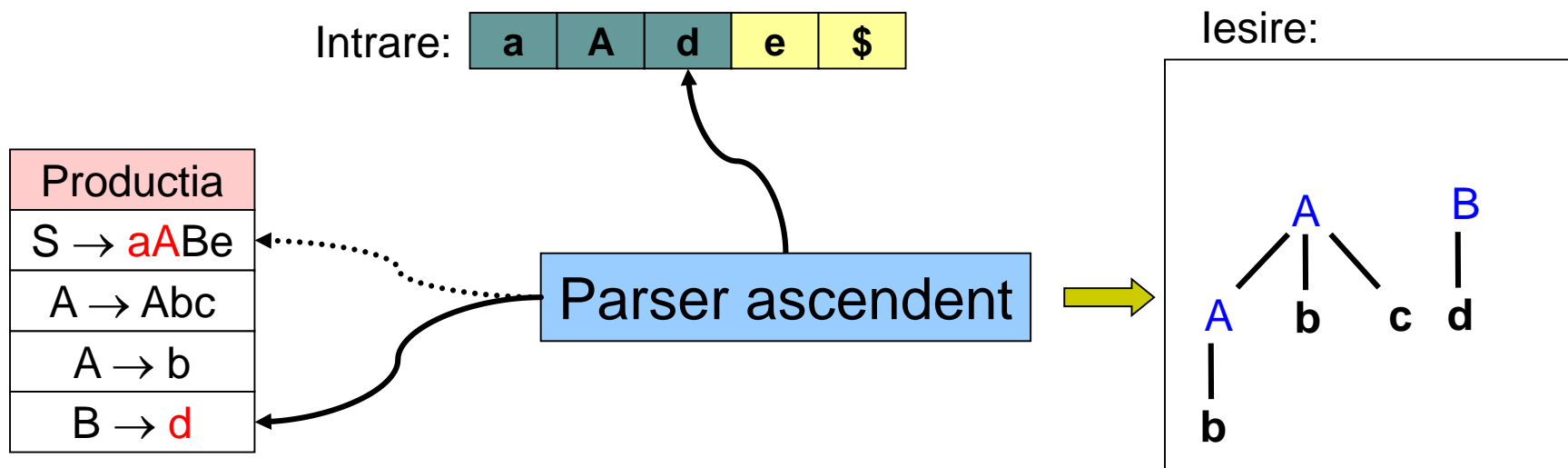
Exemplu de parser ascendent



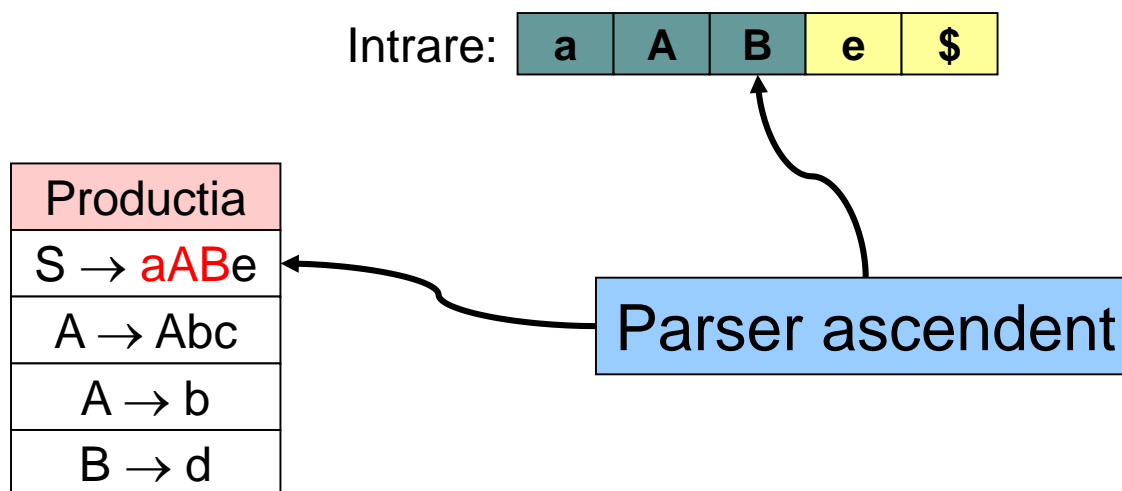
lesire:



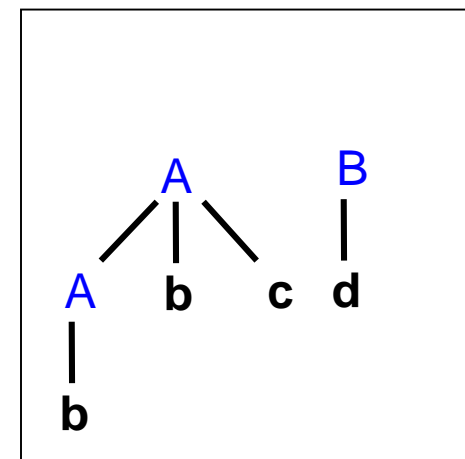
Exemplu de parser ascendent



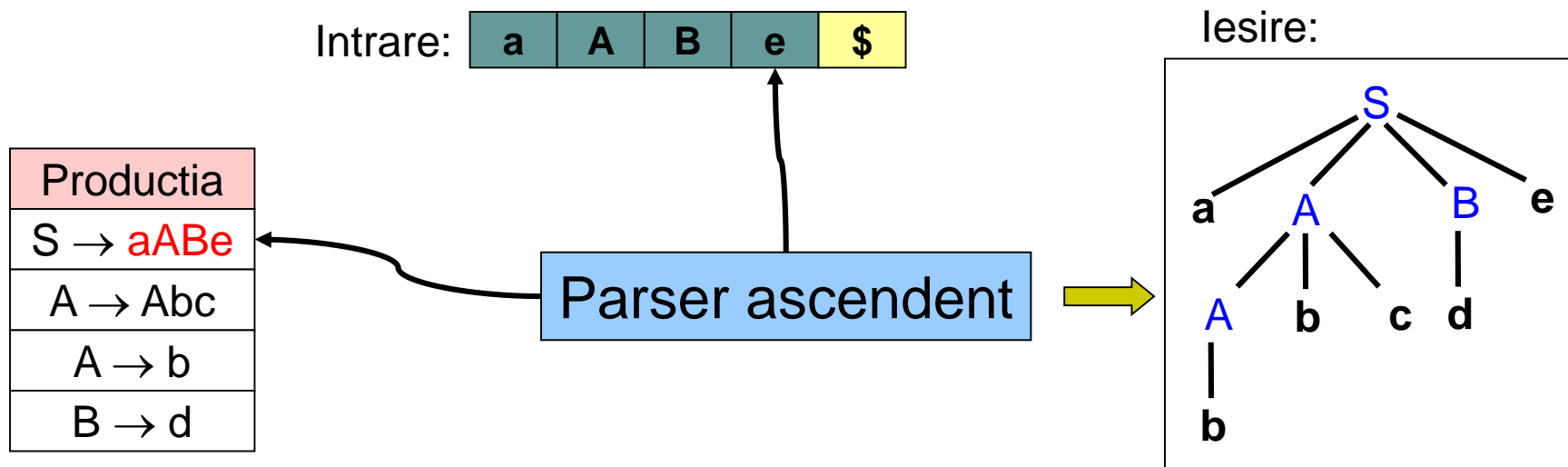
Exemplu de parser ascendent



Iesire:



Exemplu de parser ascendent



Exemplu de parser ascendent

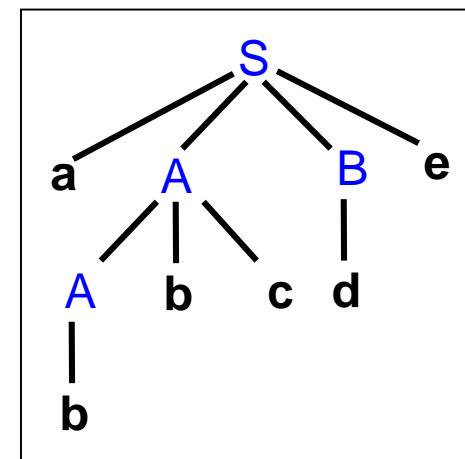
Productia
$S \rightarrow aABe$
$A \rightarrow Abc$
$A \rightarrow b$
$B \rightarrow d$

Intrare:



Parser ascendent

Iesire:



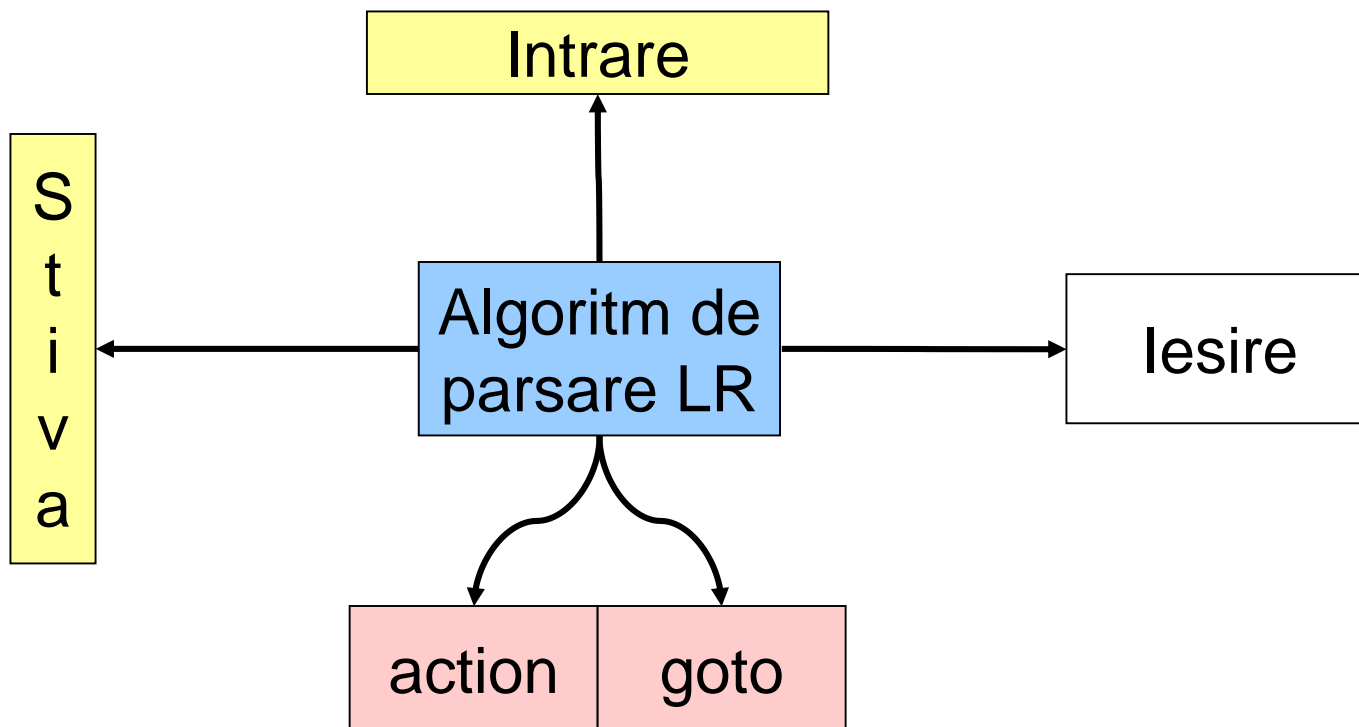
Acest parser este cunoscut ca **Parser LR** fiindca scaneaza Intrarea “**Left to right**”, si construieste “**Rightmost derivation**” in ordine inversa.

Exemplu de parser ascendent

Scanarea Productiilor pentru a detecta potrivirea cu subsiruri din Intrare, si backtrackingul face metoda din exemplul precedent foarte ineficienta.

Se poate mai bine?

Exemplu de parser LR



Exemplu de parser LR

Urmatoarea GRAMATICA:

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

Poate fi parsata cu urmatoarele
tabele 'action' si 'goto'

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Exemplu de parser LR

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

Intrare:

id	+	id	*	id	\$
----	---	----	---	----	----

Iesire:

Stiva:

0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

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

Exemplu de parser LR

Intrare: id * id + id \$

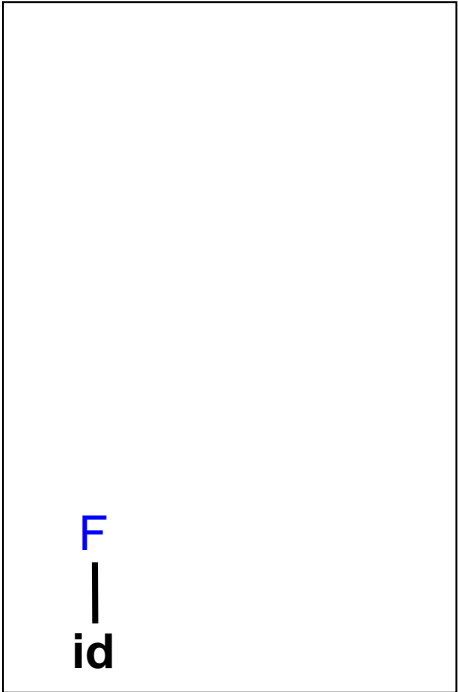
Iesire:

Stiva:

5
id
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



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

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

Iesire:

Stiva:

0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

F
|
id

Exemplu de parser LR

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

Intrare: id * id + id \$

Iesire:

Stiva:

3
F
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

T
|
F
|
id

Exemplu de parser LR

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

Intrare: id * id + id \$

Stiva: 0

LR Parsing Program

lesire:

T
|
F
|
id

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

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

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

Iesire:

Stiva:

2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

T
|
F
|
id

- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

Iesire:

Stiva:

7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

T
|
F
|
id

- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare: id * id + id \$

Iesire:

Stiva:

5
id
7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

T	F
F	id
id	

Exemplu de parser LR

- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Intrare: id * id + id \$

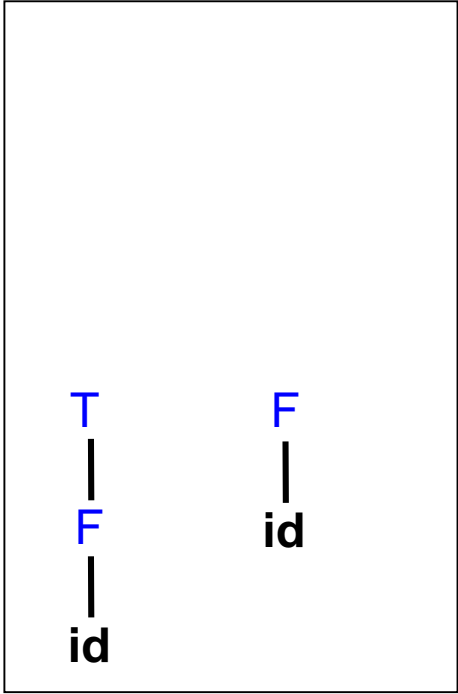
Iesire:

Stiva:

7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare: id * id + id \$

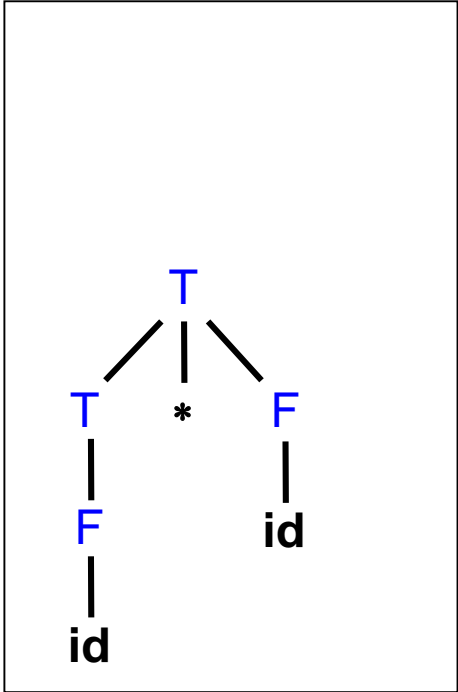
Stiva:

10
F
7
*
2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



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

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

Stiva:

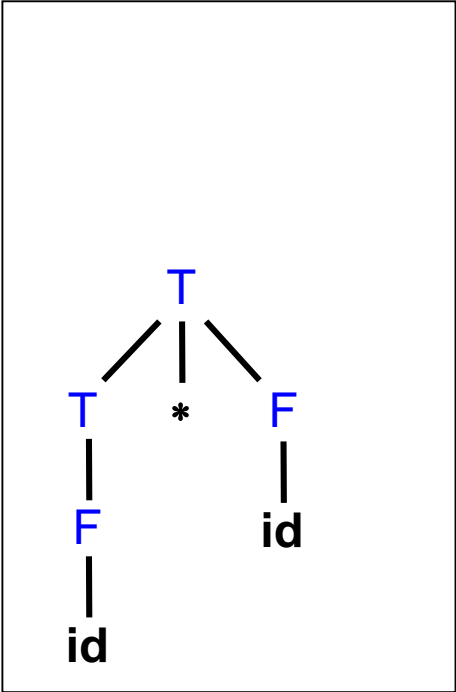
0

LR Parsing Program



State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



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

Exemplu de parser LR

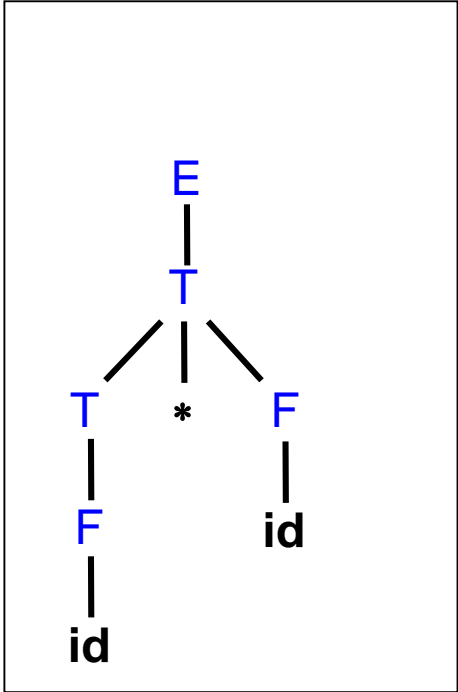
Intrare: id * id + id \$

Stiva: 2
T
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



Exemplu de parser LR

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

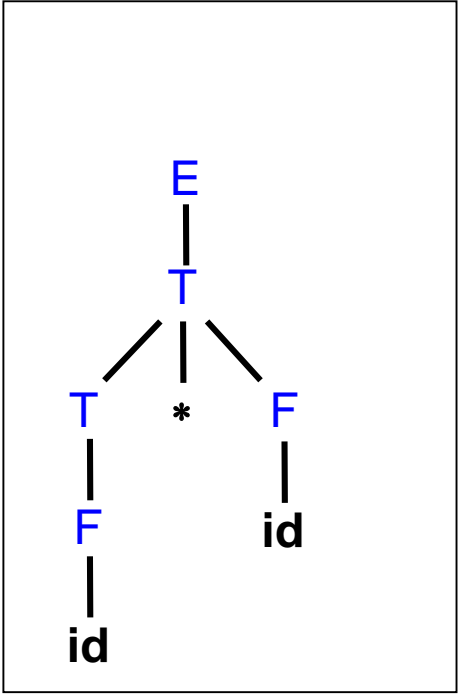
Intrare: id * id + id \$

Stiva: 0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

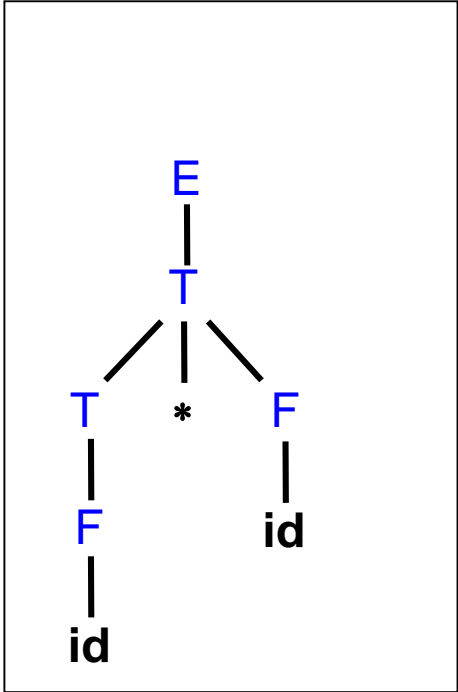
Stiva:

1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

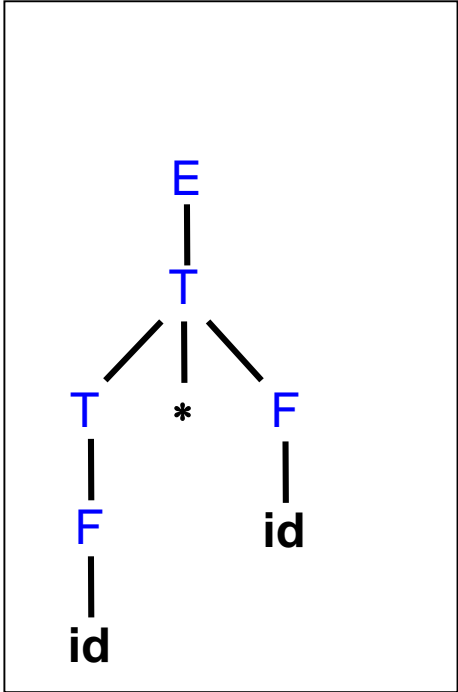
Stiva:

6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

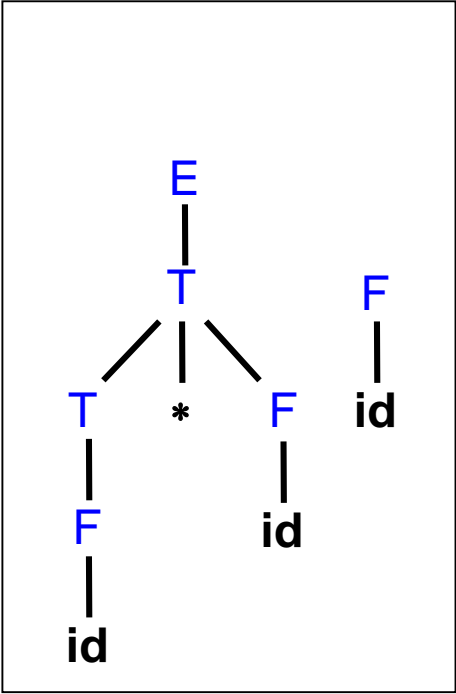
Iesire:

Stiva:

5
id
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

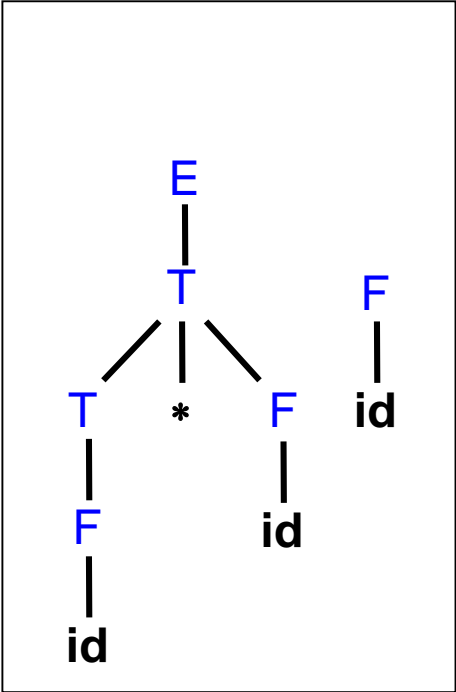
Intrare: id * id + id \$

Iesire:

Stiva: 6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare: id * id + id \$

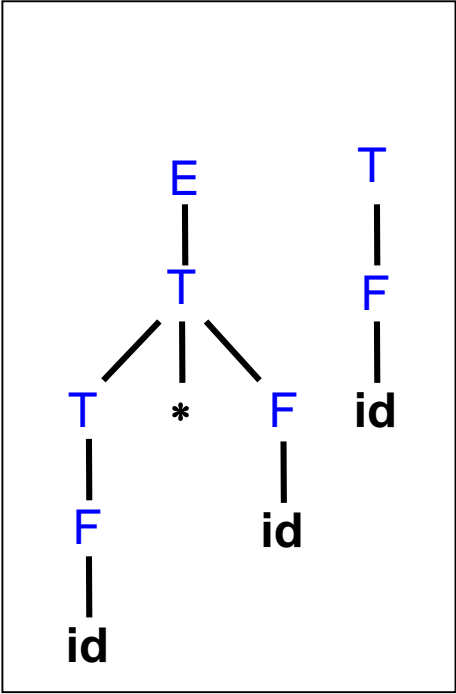
Iesire:

Stiva:

3
F
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



Exemplu de parser LR

- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

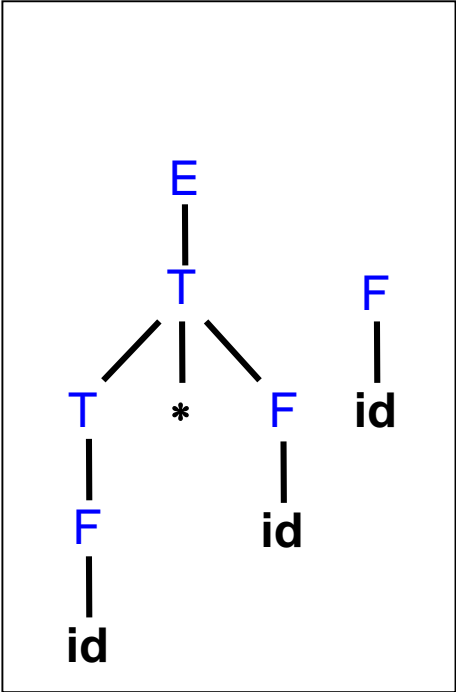
Intrare: id * id + id \$

Stiva: 6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare: id * id + id \$

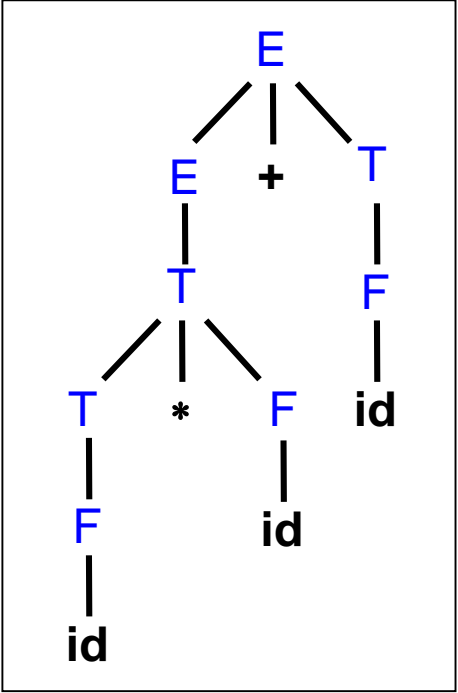
Stiva:

9
T
6
+
1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



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

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

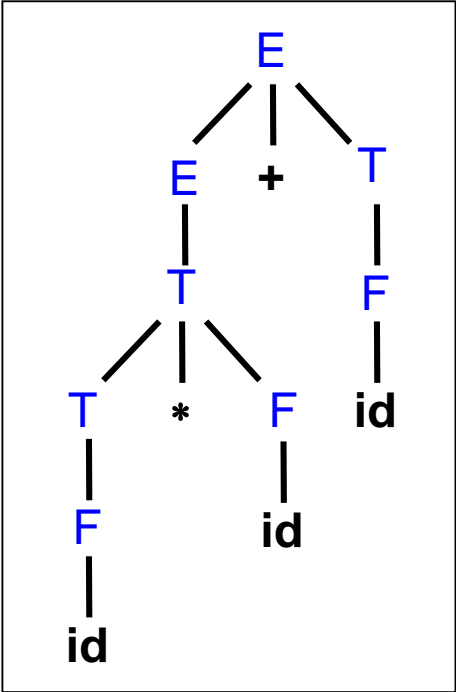
Stiva:

0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



- (1) $E \rightarrow E + T$
- (2) $E' \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

Exemplu de parser LR

Intrare:

id	*	id	+	id	\$
----	---	----	---	----	----

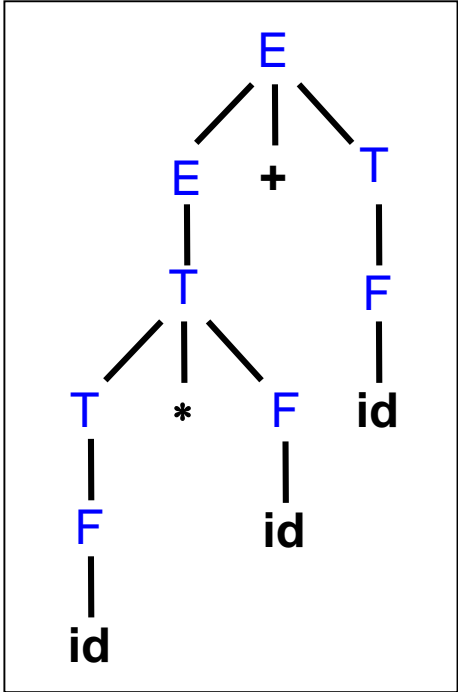
Stiva:

1
E
0

LR Parsing Program

State	action						goto		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Iesire:



Construirea tabelelor de parsare

Toate parserele LR folosesc acelasi algoritm pe care l-am aratat in slide-urile precedente.

Diferentierea intre ele se face prin tabelele action si goto

Simple LR (SLR): merge pe cele mai putine GRAMATICI, dar e cel mai usor de implementat (AhoSethiUllman pp. 221-230).

Canonical LR: merge pe cele mai multe GRAMATICI, dar e cel mai greu de implementat. Imparte starile cand e necesar pentru a preveni reduceri care ar bloca parserul. (AhoSethiUllman pp. 230-236).

Lookahead LR (LALR): merge pe majoritatea constructiilor sintactice folosite in limbajele de programare, dar produce tabele mult mai mici decat Canonical LR.

(AhoSethiUllman pp. 236-247).

Analiza sintactica Shift-Reduce

- Actiunile analizorului : o secventa de operatii **shift** si **reduce**
- Starea analizorului : O stiva de terminali si neterminali, si o stiva de stari
- Pasul curent in analiza e dat atat de stiva cat si de banda de intrare

Actiuni Shift-Reduce

- Parsarea e o secventa de actiuni shift si reduce
- Shift: muta token-ul de look-ahead pe stiva

stiva	intrare	actiune
(1+2+(3+4))+5	shift 1
(1	+2+(3+4))+5	

- Reduce: Inlocuieste simbolurile β din varful stivei cu simbolul neterminal X din partea stanga a productiei $X \rightarrow \beta$ (adica: pop β , push X)

stiva	intrare	actiune
(<u>S+E</u>	+(3+4))+5	reduce $S \rightarrow S + E$
(S	+(3+4))+5	

Analiza Shift-Reduce

$$\begin{array}{l} S \rightarrow S + E \mid E \\ E \rightarrow \text{num} \mid (S) \end{array}$$

derivarea

(1+2+(3+4))+5
(1+2+(3+4))+5
(1+2+(3+4))+5
(E+2+(3+4))+5
(S+2+(3+4))+5
(S+2+(3+4))+5
(S+2+(3+4))+5
(S+E+(3+4))+5
(S+(3+4))+5
(S+(3+4))+5
(S+(3+4))+5
(S+(3+4))+5

...

stiva

(
(1
(E
(S
(S+
(S+2
(S+E
(S
(S+
(S+(
(S+(3

sir de intrare

(1+2+(3+4))+5
1+2+(3+4))+5
+2+(3+4))+5
+2+(3+4))+5
+2+(3+4))+5
2+(3+4))+5
+(3+4))+5
+(3+4))+5
+(3+4))+5
(3+4))+5
3+4))+5
+4))+5

actiune

shift
shift
reduce $E \rightarrow \text{num}$
reduce $S \rightarrow E$
shift
shift
reduce $E \rightarrow \text{num}$
reduce $S \rightarrow S+E$
shift
shift
shift
reduce $E \rightarrow \text{num}$

Probleme (selectarea actiunii)

- De unde stim ce actiune sa aplicam: shift sau reduce? Si cu ce regula sa reducem?
 - Uneori putem reduce dar nu e bine sa o facem – am ajunge cu analiza intr-un punct mort (sau nu am respecta precedenta operatorilor)
 - Uneori putem reduce stiva in mai multe feluri, nu intr-un singur fel

Selectarea actiunii

- Starea curenta:
 - stiva β
 - simbolul look-ahead b
 - exista productia $X \rightarrow \gamma$, si stiva e de forma $\beta = \alpha\gamma$
- Ar trebui analizorul sa:
 - Shifteze b pe stiva, transformand-o in βb ?
 - Reduca aplicand productia $X \rightarrow \gamma$, presupunand ca stiva e de forma $\beta = \alpha\gamma$ - si sa o transforme astfel in αX ?
- Decizie in functie de b și de prefixul α
 - α e diferit pentru productii diferite, deoarece partea dreapta a productiilor(γ -urile) poate avea lungimi diferite

Algoritmul de parsare LR

- Mecanismul de baza
 - Folosim un set de stari ale parser-ului
 - Folosim o stiva de stari (eventual, alternam simboluri cu stari)
 - De ex., 1 (6 S 10 + 5 (verde = stari)
 - Folosim tabela de parsare pentru:
 - A determina ce actiune se aplica (shift/reduce)
 - A determina starea urmatoare
- Actiunile analizatorului pot fi determinate cu exactitate din tabellele de parsare

Tabela de parsare LR

	Terminali	Non-terminali
Stare	Actiunea si starea urmatoare	Starea urmatoare
	Tabela 'action'	Tabela 'Goto'

- Algoritm: ne uitam la starea curenta S si terminalul C de la intrare
 - Daca $Action[S, C] = s(S')$ atunci 'shift' si trece in S' :
 - push(C), push(S')
 - Daca $Action[S, C] = X \rightarrow \alpha$ atunci 'reduce':
 - pop($2 * |\alpha|$), $S' = top()$, push(X), push($Goto[S', X]$)

Gramatici LR(k)

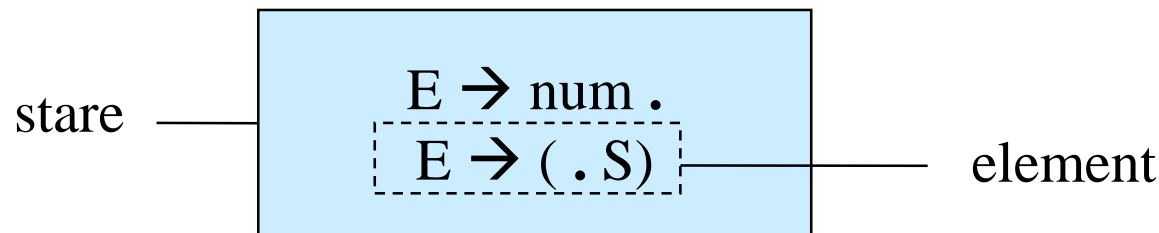
- LR(k) = Left-to-right scanning, right-most derivation, k lookahead tokens
- Cazurile principale
 - LR(0), LR(1)
 - Variante: SLR and LALR(1)
- Analizările pt. gramatici LR(0):
 - Aleg shift/reduce fara sa se uite la lookahead
 - Incepem cu ele, caci ne vor ajuta sa intelegem parsarea shift-reduce

Construirea tableleor de parsare LR(0)

- Pentru a construi tabela de parsare:
 - Se definesc starile analizorului
 - Se construiește un AFD care descrie tranzițiile dintre stări
 - Se folosește AFD pt. a construi tabela de parsare
- Fiecare stare LR(0) e un set de elemente LR(0)
 - Un element LR(0): $X \rightarrow \alpha \cdot \beta$ unde $X \rightarrow \alpha\beta$ e o producție din gramatică
 - Elementele LR(0) urmăresc progresul tuturor producțiilor care ar putea urma
 - Elementul $X \rightarrow \alpha \cdot \beta$ abstractizează faptul că parser-ul are deja sirul α în vârful stivei

Exemplu de stare LR(0)

- Un element LR(0) e o productie din gramatica cu un separator "." unde va in partea dreapta a productiei



- Subsirul de dinainte de "." e deja pe stiva (inceputul unui posibil sir γ ce urmeaza a fi 'reduc')
Subsirul de dupa "." ne indica ce am putea intalni in continuare

Intrebare

Pentru productia,

$E \rightarrow \text{num} \mid (S)$

Doua elemente LR(0) sunt:

$E \rightarrow \text{num} \cdot$

$E \rightarrow (\cdot S)$

Mai sunt si altele? Daca da, care? Daca nu, de ce?

Gramatica LR(0)

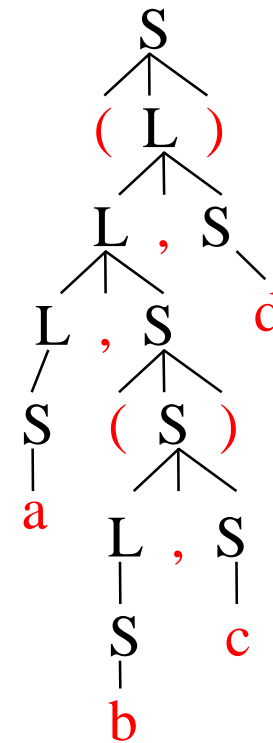
- Gramatica listelor imbricate

- $S \rightarrow (L) \mid \text{id}$
- $L \rightarrow S \mid L, S$

- Exemple:

- (a,b,c)
- $((a,b), (c,d), (e,f))$
- $(a, (b,c,d), ((f,g)))$

Arbore de derivare pentru
 $(a, (b,c), d)$



Starea de start si Inchiderile

- Starea de start
 - Extindem gramatica, cu productia $S' \rightarrow S$
 - Starea de start a AFD : $\text{Inchidere}(S' \rightarrow \cdot S)$
- Inchiderea unei multimi de elemente LR(0):
 - Incepem cu $\text{Inchidere}(S) = S$
 - Pentru fiecare element din S:
 - $X \rightarrow \alpha \cdot Y \beta$
 - Adauga toate elementele LR(0) de forma $Y \rightarrow \cdot \gamma$, pentru toate productiile $Y \rightarrow \gamma$ din gramatica

Exemplu de inchidere

$$\begin{array}{l} S \rightarrow (L) \mid id \\ L \rightarrow S \mid L,S \end{array}$$

Elementul LR(0) “de start”

pentru AFD

$$S' \rightarrow .S$$

inchidere

$$S' \rightarrow .S$$

$$S \rightarrow .(L)$$

$$S \rightarrow .id$$

- Setul tuturor productiilor care ar putea fi ‘reduse’ in continuare
- Elementele adaugate au simbolul “.” la inceput:
nu avem inca tokeni pe stiva, pentru aceste productii.

Operatia 'Goto'

- Operatia Goto descrie tranzitiile intre starile automatului (care sunt multimi de elemente LR(0))
 - A nu se confunda cu tabela 'Goto'
- Algoritm: pentru starea S si simbolul Y
 - Daca elementul $[X \rightarrow \alpha \cdot Y \beta]$ e in I, atunci
 - $\text{Goto}(I, Y) \subseteq \text{Inchidere}([X \rightarrow \alpha Y \cdot \beta])$

$S' \rightarrow \cdot S$
$S \rightarrow \cdot (L)$
$S \rightarrow \cdot id$

Goto(S, '(')

Inchidere ({ $S \rightarrow (\cdot L)$ })

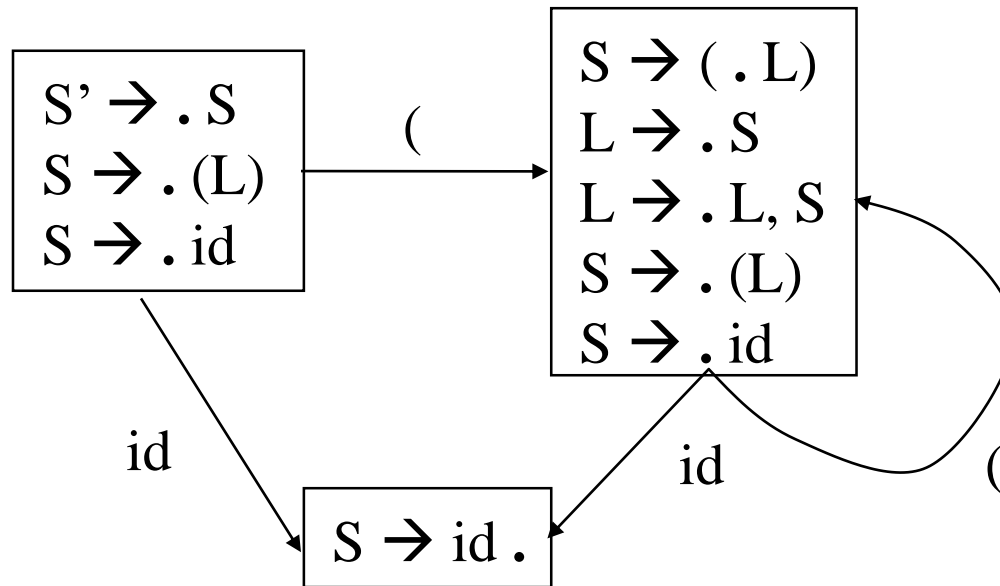
Intrebari

$E' \rightarrow E$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

Daca $I = \{ [E' \rightarrow \cdot E] \}$, atunci $Inchidere(I) = ??$

Daca $I = \{ [E' \rightarrow E \cdot], [E \rightarrow E \cdot + T] \}$, atunci $Goto(I, +) = ??$

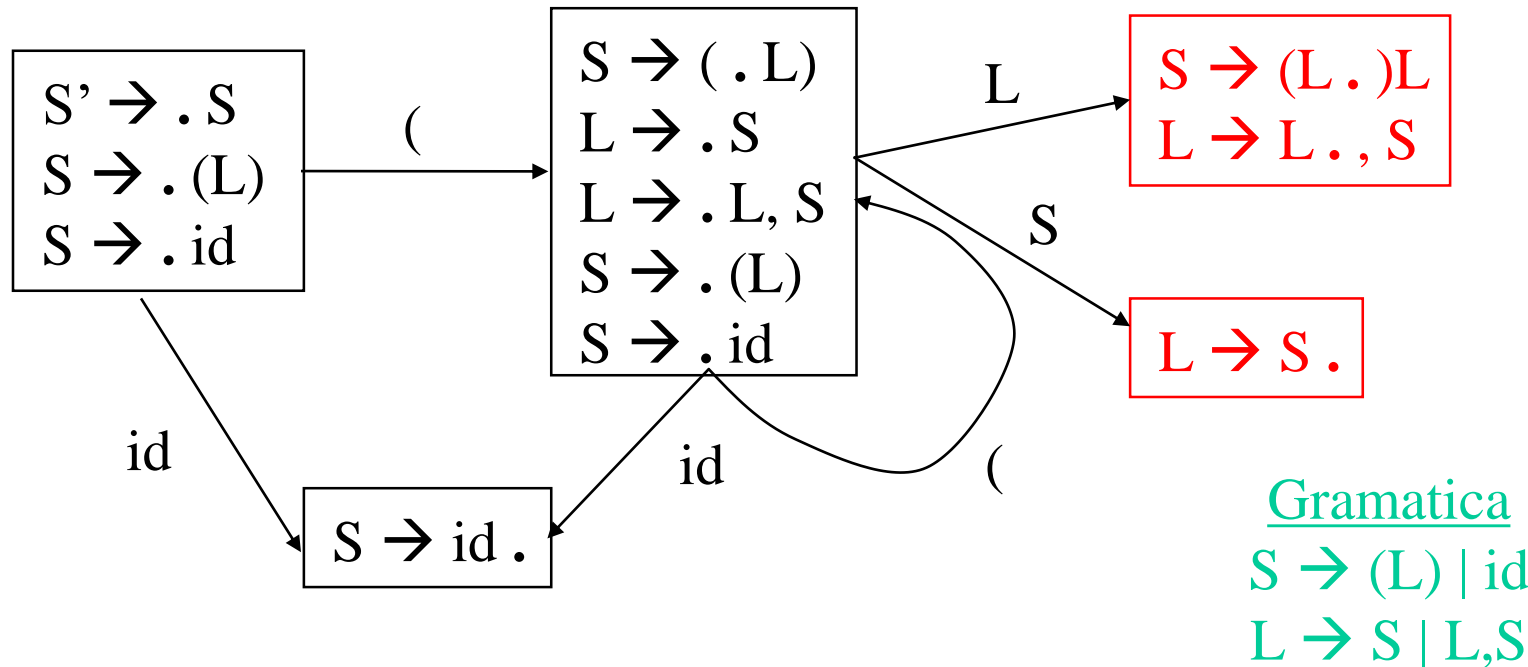
Goto: Simboli terminali (tokeni)



Gramatica
 $S \rightarrow (L) \mid id$
 $L \rightarrow S \mid L, S$

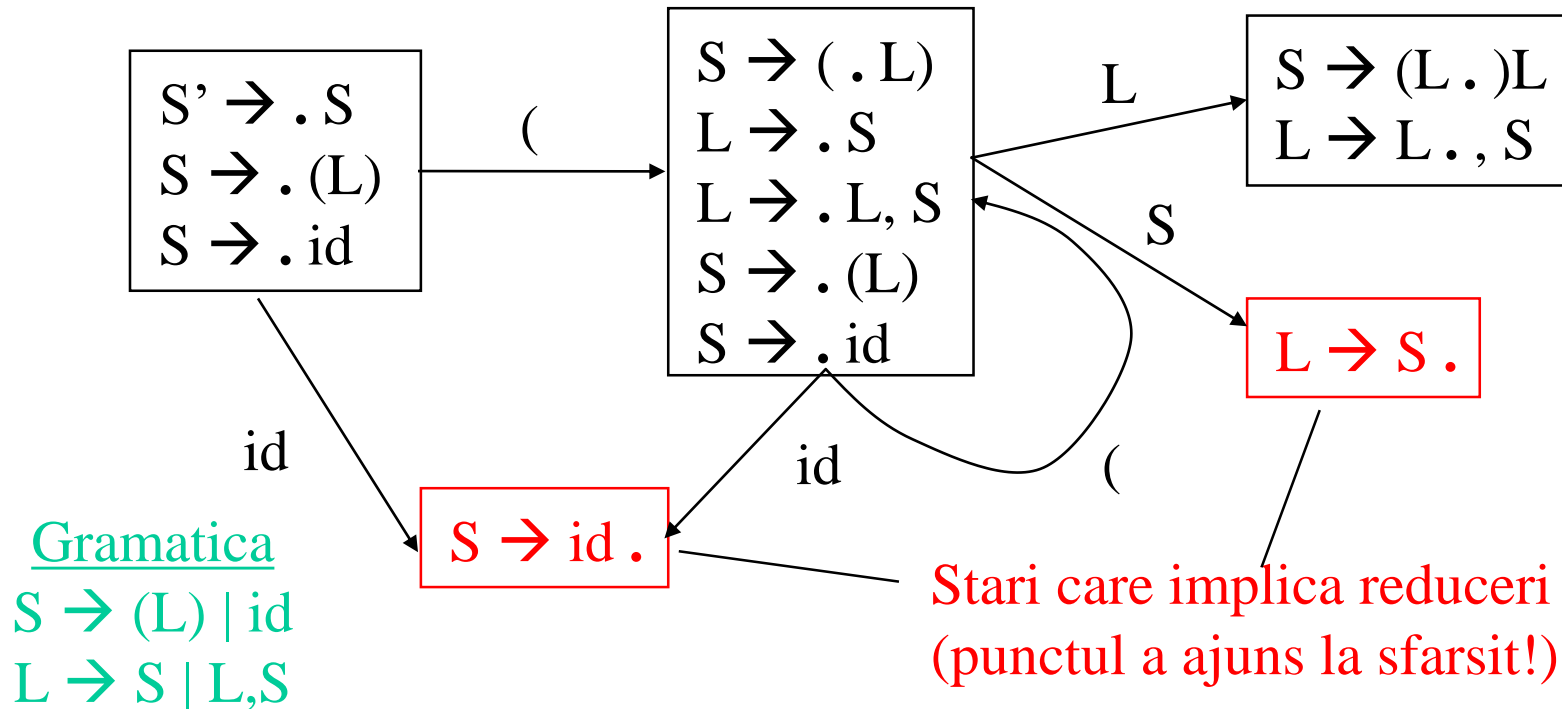
In starea noua, includem toate elementele care au simbolul potrivit
Imediat dupa punct – mutam punctul la aceste elemente si aplicam
‘Inchiderea’ pt. a obtine toate elementele starii.

Goto: Simboli neterminali

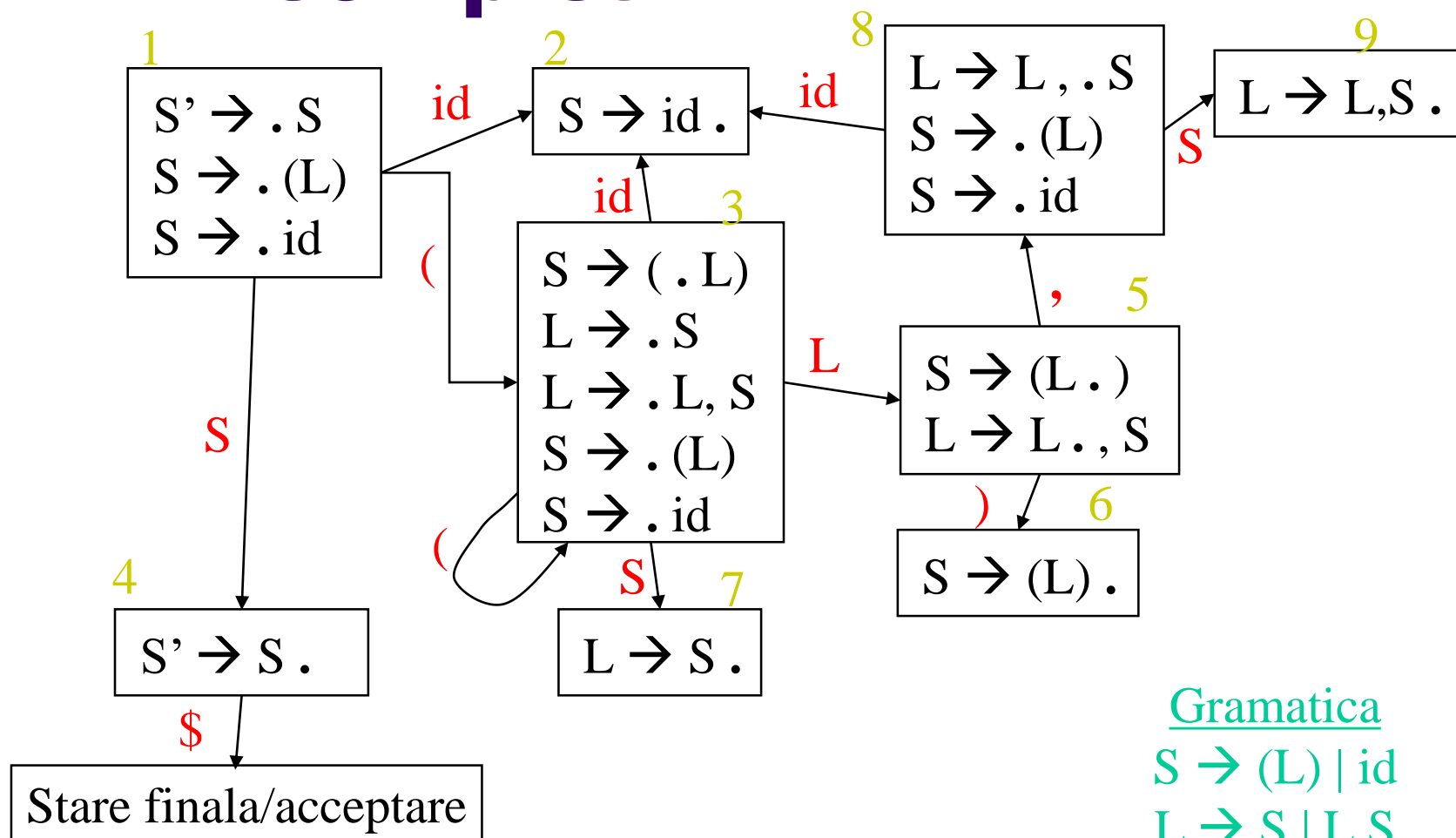


Acelasi algoritm pentru tranzitii pe neterminali

Aplicarea actiunilor de reducere



AFD Complet



Exemplu de parsare ((a),b)

Derivare	stiva	intrare	actiune
((a),b) ←	1	((a),b)	shift, goto 3
((a),b) ←	1(3	(a),b)	shift, goto 3
((a),b) ←	1(3(3	a),b)	shift, goto 2
((a),b) ←	1(3(3a2),b)	reduce $S \rightarrow id$
((S),b) ←	1(3(3(S7),b)	reduce $L \rightarrow S$
((L),b) ←	1(3(3(L5),b)	shift, goto 6
((L),b) ←	1(3(3(L5)6	,b)	reduce $S \rightarrow (L)$
(S,b) ←	1(3S7	,b)	reduce $L \rightarrow S$
(L,b) ←	1(3L5	,b)	shift, goto 8
(L,b) ←	1(3L5,8	b)	shift, goto 9
(L,b) ←	1(3L5,8b2)	reduce $S \rightarrow id$
(L,S) ←	1(3L8,S9)	reduce $L \rightarrow L,S$
(L) ←	1(3L5)	shift, goto 6
(L) ←	1(3L5)6		reduce $S \rightarrow (L)$
S ←	1S4	\$	acceptare

$S \rightarrow (L) \mid id$
 $L \rightarrow S \mid L,S$

Reduceri

- Cand reducem cu productia $X \rightarrow \beta$ si stiva $\alpha\beta$
 - Pop β de pe stiva – ramane prefixul α
 - Fa un singur pas in AFD, plecand de la starea care e acum in varful stivei α
 - Push X pe stiva, impreuna cu noua stare a AFD
- Exemplu

derivare	stiva	intrare	actiune
$((a),b) \leftarrow$	1 (3 (3	a),b)	shift, goto 2
$((a),b) \leftarrow$	1 (3 (3 a 2),b)	reduce $S \rightarrow id$
$((S),b) \leftarrow$	1 (3 (3 S 7),b)	reduce $L \rightarrow S$

Construirea tabelii de parsare LR(0)

- Starile din tabela = starile din AFD
- Pentru tranzitia $S \rightarrow S'$ pe terminalul C:
 - $\text{Action}[S, C] += \text{Shift}(S')$
- Pentru tranzitia $S \rightarrow S'$ pe neterminalul N:
 - $\text{Goto}[S, N] += S'$
- Daca S e stare de reducere $X \rightarrow \beta$. atunci:
 - $\text{Action}[S, *] += \text{Reduce}(X \rightarrow \beta)$

Tabela de parsare LR din exemplul nostru

Stare	Tokeni/terminali					Neterminali	
	()	id	,	\$	S	L
	1	s3	s2			g4	
	2	S→id	S→id	S→id	S→id	S→id	
	3	s3	s2			g7	g5
	4				accept		
	5	s6	s8				
	6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)	
	7	L→S	L→S	L→S	L→S	L→S	
	8	s3	s2			g9	
	9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S	

Verde = shift

rosu = reduce

Sumarul LR(0)

- Reteta de parsare LR(0):
 - Pornim cu o gramatica LR(0)
 - Calculam starile LR(0) si construim AFD:
 - Folosim operatia de inchidere pentru a construi stari
 - Folosim operatia goto pentru a calcula tranzitii
 - Construim tabela de parsare LR(0) din AFD
- Toate acestea pot fi facute automat!

Limitările LR(0)

- O masina LR(0) functioneaza doar daca starile cu actiuni 'reduce' au o singura actiune (de reducere) – in acele stari, se reduce mereu, ignorand lookahead-ul
- Cu o gramatica mai complexa, construirea tabelelor ne-ar da stari cu conflicte shift/reduce sau reduce/reduce
- Trebuie folosit lookahead pt. a putea alege corect

OK

$L \rightarrow L, S.$

shift/reduce

$L \rightarrow L, S.$
$S \rightarrow S., L$

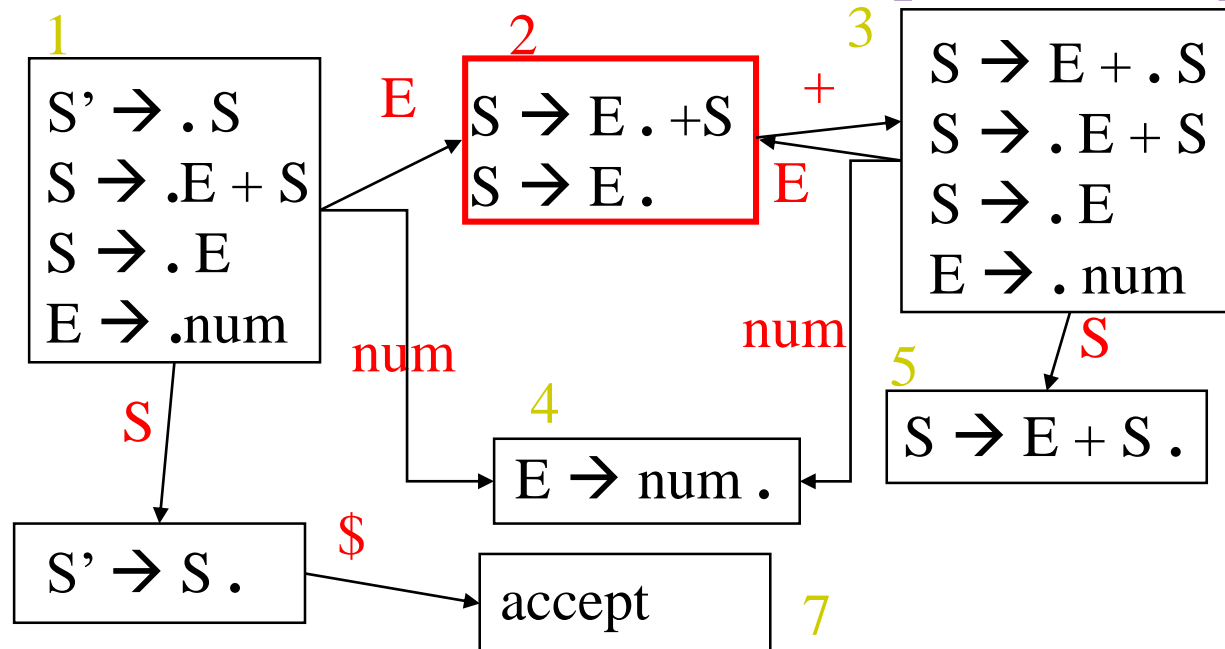
reduce/reduce

$L \rightarrow S, L.$
$L \rightarrow S.$

O gramatica non-LR(0)

- Gramatica pentru adunarea numerelor
 - $S \rightarrow S + E \mid E$
 - $E \rightarrow \text{num}$
- Scrisa asa e LR(0)
- Scrisa cu recursivitate dreapta nu e LR(0)
 - $S \rightarrow E + S \mid E$
 - $E \rightarrow \text{num}$

Cum am face AFD pt. LR(0)



Gramatica
 $S \rightarrow E + S \mid E$
 $E \rightarrow \text{num}$

Shift sau
reduce
in starea 2?

	num	+	\$	E	S
1	s4			g2	g6
2	S→E	s3/S→E	S→E		

Parsarea SLR

- SLR = "Simple LR" = O extensie simpla a LR(0)
 - Pentru fiecare reducere $X \rightarrow \beta$, ne uitam la urmatorul simbol C
 - Aplicam reducerea doar daca C e in FOLLOW(X)
- Tabelele SLR elimina o parte din conflicte
 - Sunt la fel ca LR(0) cu exceptia randurilor de 'reducere'
 - Adauga reduceri $X \rightarrow \beta$ doar in coloanele cu simbolii corespunzand lui FOLLOW(X)

Exemplu: FOLLOW(S) = {\$}

	num	+	\$	E	S
1	s4			g2	g6
2		s3	S→E		

Tabela de parsare SLR

- Reducerile nu mai ocupa un rand intreg ca in cazul LR(0)

- In rest, identic cu LR(0)

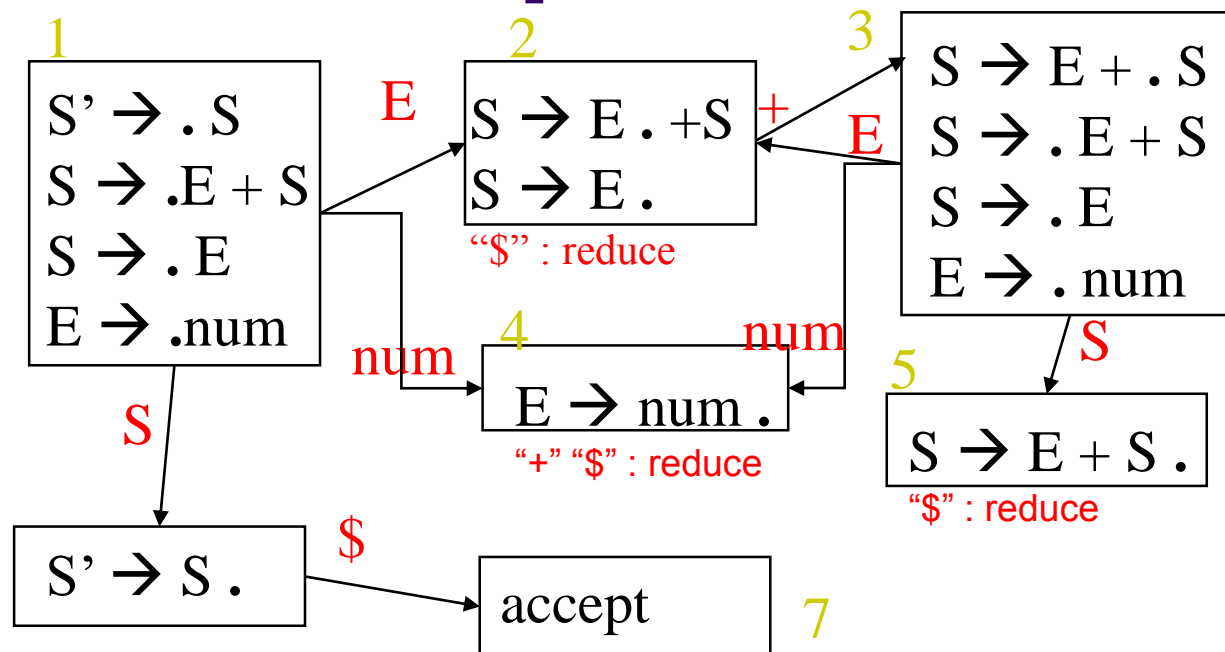
Gramatica

$S \rightarrow E + S \mid E$

$E \rightarrow \text{num}$

	num	+	\$	E	S
1	s4			g2	g6
2		s3	$S \rightarrow E$		
3	s4			g2	g5
4		$E \rightarrow \text{num}$	$E \rightarrow \text{num}$		
5			$S \rightarrow E + S$		
6			s7		
7			accept		

Automatul pt. SLR



Gramatica

$S \rightarrow E + S \mid E$

$E \rightarrow \text{num}$

	num	+	\$	E	S
1	s4			g2	g6
2		s3	S→E		

Un exemple non-SLR

Fie gramatica:

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow *R$

$L \rightarrow \text{ident}$

$R \rightarrow L$

L este l-value, R este r-value, si * e dereferentiere de pointer.

Exemple:

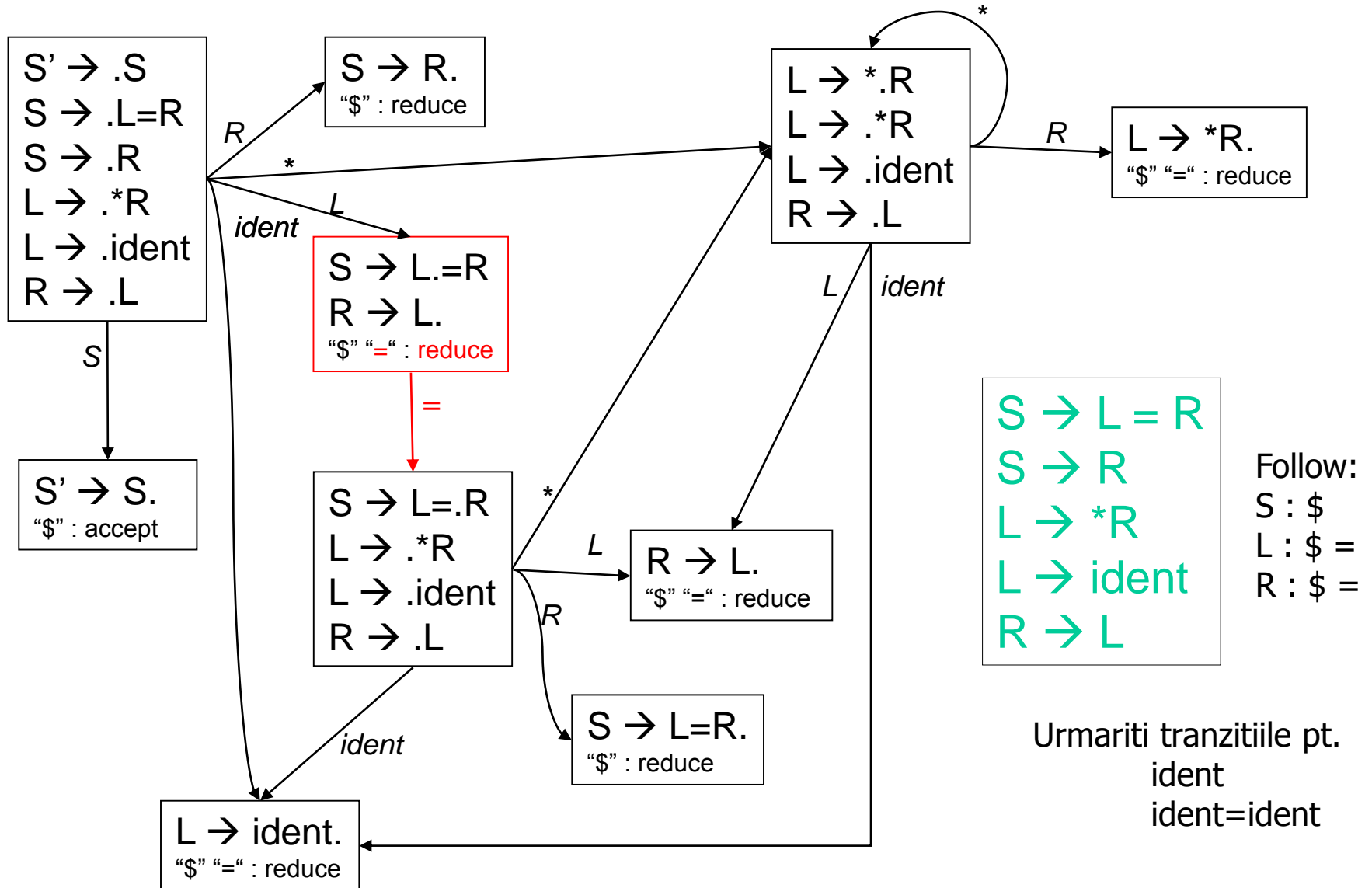
ident

*ident

ident = *ident

**ident = ident

DFA pentru SLR

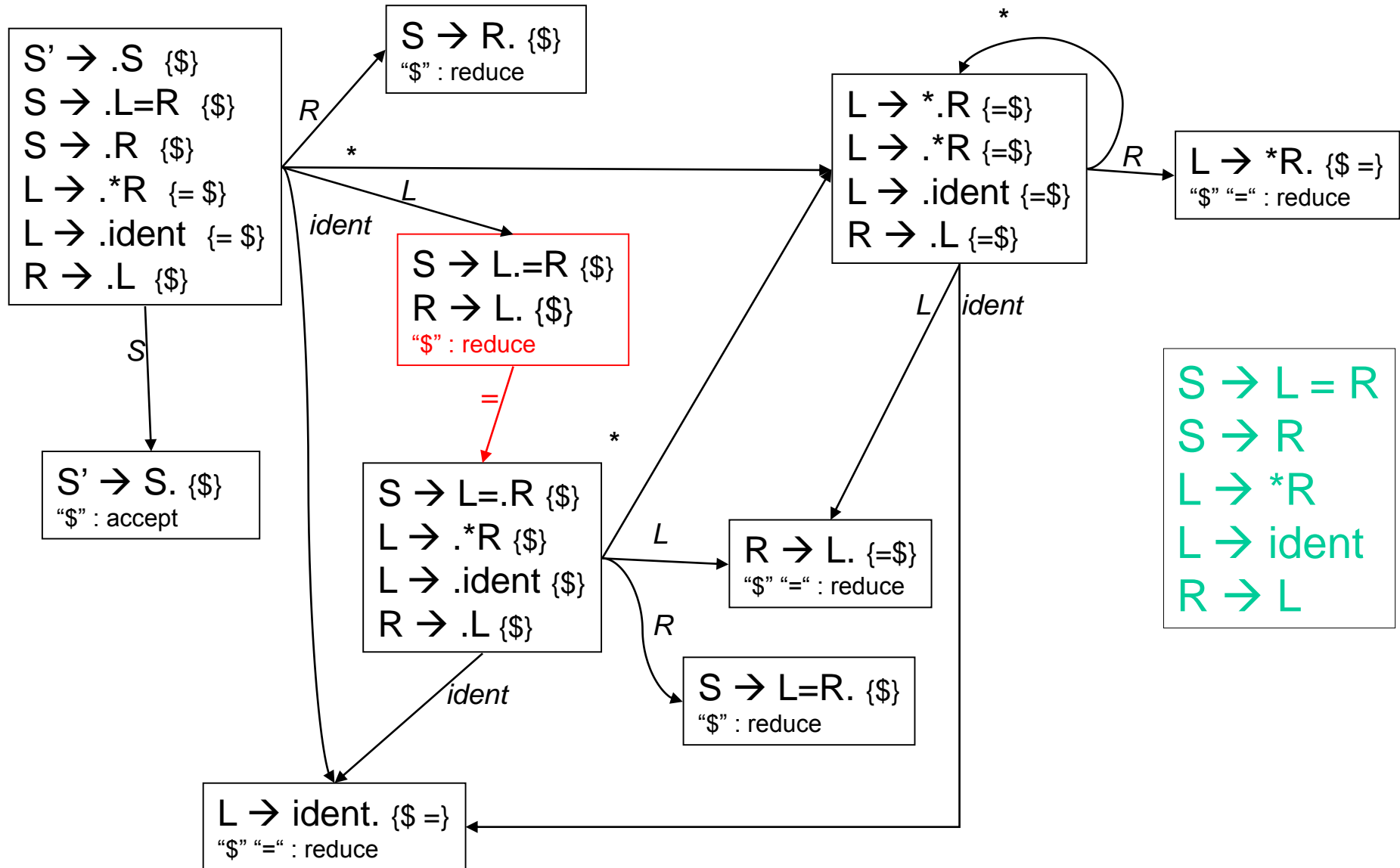


LALR(1)

- SLR: Decizia de reduce pentru " $X \rightarrow \beta$ " se face doar daca urmatorul simbol e in FOLLOW(X)
- LALR(1): Fiecare productie are un set de simbolii care ii pot urma. " $X \rightarrow \beta$, S" se reduce doar daca urmatorul simbol e in S.
- Seturile se pot calcula prin aplicarea recursiva a regulilor:

$S' \rightarrow .S, \{\$ \}$	$B \rightarrow a.A\beta$ $c \in \text{FIRST}(\beta)$ $A \rightarrow .X\delta, \{c\}$	$B \rightarrow a.A\beta, \{c\}$ $c \in \text{FOLLOW}(\beta)$ $\beta \rightarrow^* \epsilon$ $A \rightarrow .X\delta, \{c\}$	$X \rightarrow a.X\beta, c$ $X \rightarrow aX.\beta, c$
$S' \rightarrow .S, \{\$ \}$ $S \rightarrow .L=R$ $S \rightarrow .R$ $L \rightarrow .*R$ $L \rightarrow .\text{ident}$ $R \rightarrow .L$	$S' \rightarrow .S, \{\$ \}$ $S \rightarrow .L=R$ $S \rightarrow .R$ $L \rightarrow .*R, \{=\}$ $L \rightarrow .\text{ident}, \{=\}$ $R \rightarrow .L$	$S' \rightarrow .S, \{\$ \}$ $S \rightarrow .L=R, \{\$ \}$ $S \rightarrow .R, \{\$ \}$ $L \rightarrow .*R, \{=\ \$ \}$ $L \rightarrow .\text{ident}, \{=\ \$ \}$ $R \rightarrow .L, \{\$ \}$	$L \rightarrow .\text{ident}, \{=\ \$ \}$ $L \rightarrow \text{ident.}, \{=\ \$ \}$

DFA pentru LALR



Limitari LALR(1)

- Exista gramatici ce pot fi recunoscute cu un singur simbol "lookahead", dar nu de algoritmul LALR(1)

DEF \rightarrow PARAM RETURN ,

PARAM \rightarrow T

PARAM \rightarrow NAMES : T

RETURN \rightarrow T

RETURN \rightarrow N : T

NAMES \rightarrow N

NAMES \rightarrow N, NAMES

T \rightarrow id

N \rightarrow id

Exemple:

"id id," \rightarrow "T T," Int String,

"id,id:id id," \rightarrow "N,N:T T," a,b:Int String,

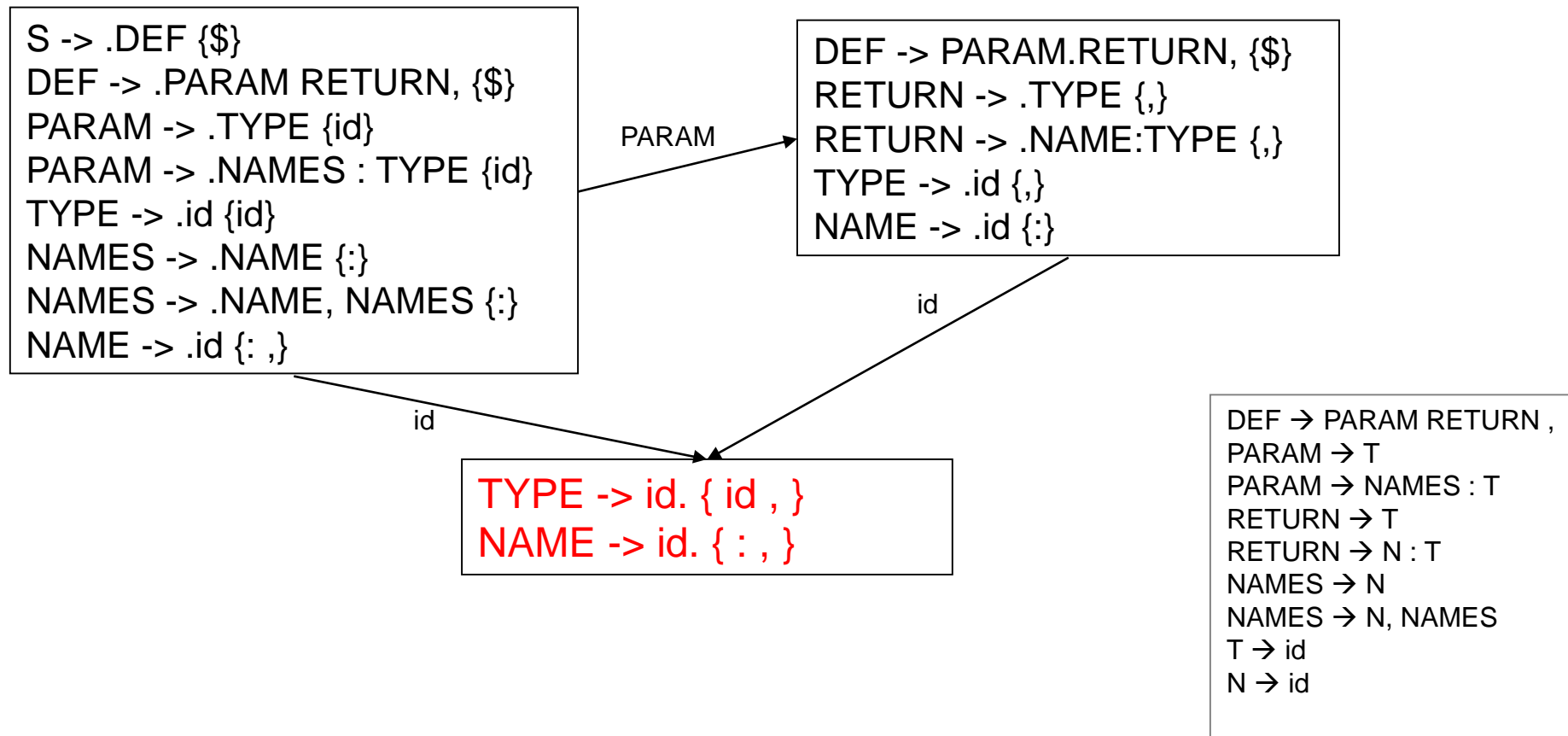
"id id:id," \rightarrow "T N:T," a result:String,

LALR(1) nu poate separa T de N

atunci cand sunt urmate de caracterul ","

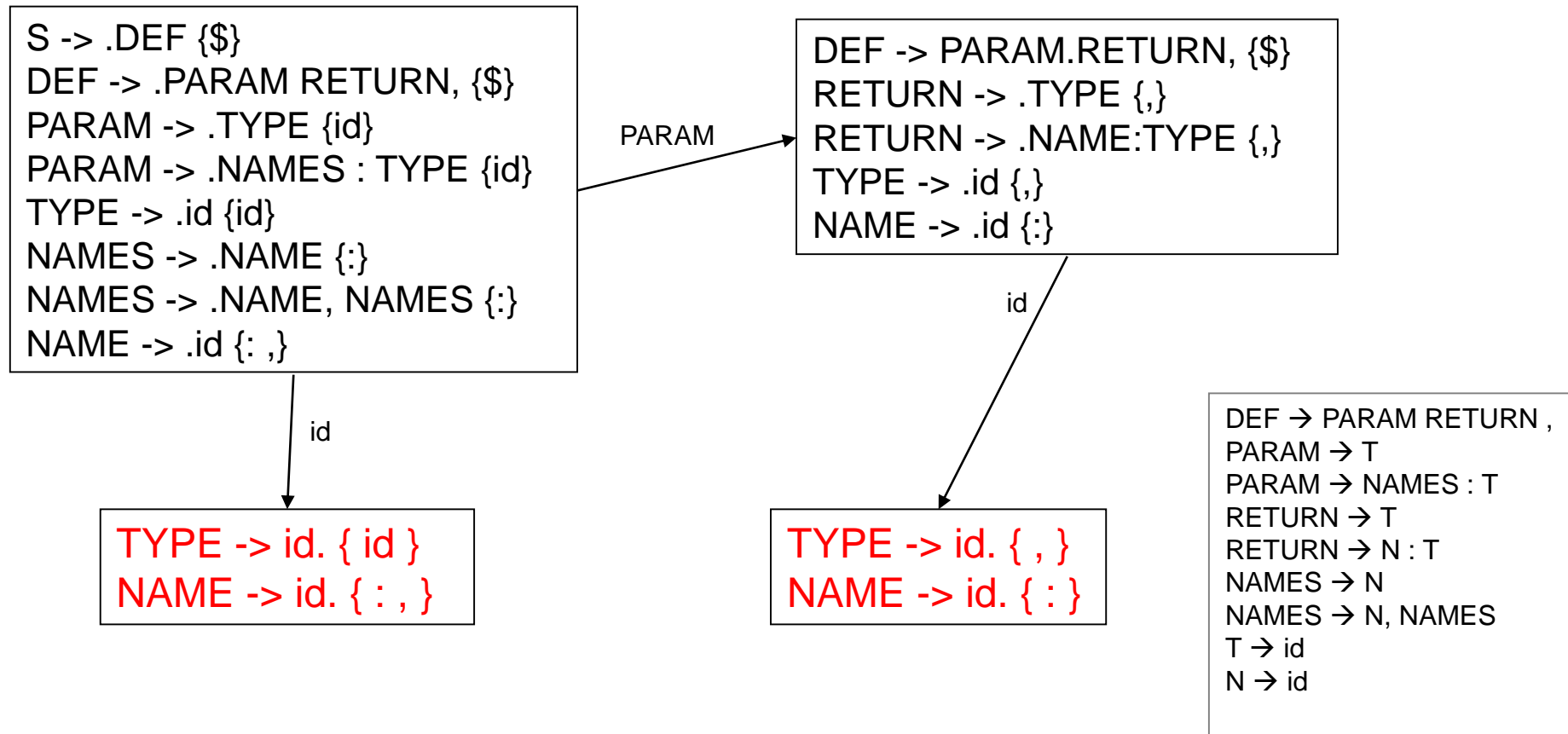
Limitari LALR(1)

O portiune din automatul LALR:



LR(1) vs. LALR(1)

Solutia – doua contexte diferite pornind de la aceleasi elemente LR(0):



Parsarea LR(1)

- Scoate cel mai mult posibil din 1 simbol de lookahead
- Gramatica LR(1) = analizabila de un parser shift/reduce cu lookahead 1
- Analiza LR(1) foloseste concepte similare cu LR(0)
 - Starile parserului = multimi de elemente LR(1)
 - Element LR(1) = element LR(0) + simbol lookahead care poate urma dupa productie
 - Element LR(0) : $S \rightarrow \cdot S + E$
 - Element LR(1) : $S \rightarrow \cdot S + E, +$
 - Lookahead are impact doar asupra operatiilor REDUCE, care se pot aplica doar atunci cand lookahead = input

Stari LR(1)

- Stare LR(1) = multime de elemente LR(1)
- Element LR(1) = $(X \rightarrow \alpha \cdot \beta, \gamma)$
 - Insemnand: α este deja pe varful stivei, ma astept sa urmeze in continuare $\beta\gamma$
- Notatie prescurtata
 - $(X \rightarrow \alpha \cdot \beta, \{x_1, \dots, x_n\})$
 - inseamna:
 - $(X \rightarrow \alpha \cdot \beta, x_1)$
 - \dots
 - $(X \rightarrow \alpha \cdot \beta, x_n)$
- Trebuie sa extindem operatiile 'inchidere' si 'goto'

$S \rightarrow S \cdot + E$	$+, \$$
$S \rightarrow S + \cdot E$	num

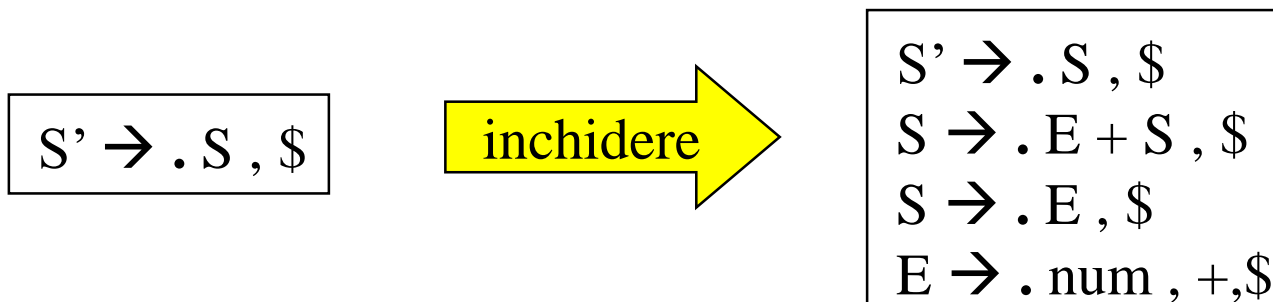
Inchidere LR(1)

- Operatia de inchidere LR(1) :
 - Incepem cu $\text{Inchidere}(S) = S$
 - Pentru fiecare element din S:
 - $X \rightarrow \alpha \cdot Y \beta, z$
 - si fiecare productie $Y \rightarrow \gamma$, adaugam urmatorul element la inchiderea lui S: $Y \rightarrow \cdot \gamma, \text{FIRST}(\beta z)$
 - Repetam pana nu mai sunt schimbari
- Similar cu inchiderea LR(0), dar tine cont si de simbolul de lookahead

Starea de start LR(1)

- Starea initiala : incepem cu $(S' \rightarrow . S , \$)$, apoi aplicam operatia de inchidere
- Exemplu: gramatica sumelor

$S' \rightarrow S \$$
 $S \rightarrow E + S \mid E$
 $E \rightarrow \text{num}$



Operatia 'Goto' in LR(1)

- Operation goto = descrie tranzitii intre stari LR(1)
- Algoritm: pentru o stare S si un simbol Y (ca si la LR(0))
 - Daca elementul $[X \rightarrow \alpha \cdot Y \beta, a]$ este in I, atunci
 - $\text{Goto}(I, Y) = \text{Inchidere}([X \rightarrow \alpha Y \cdot \beta, a])$

Gramatica:

$S' \rightarrow S\$$

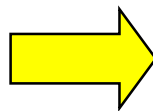
$S \rightarrow E + S \mid E$

$E \rightarrow \text{num}$

S1

$S \rightarrow E \cdot + S, \$$
 $S \rightarrow E \cdot, \$$

$\text{Goto}(S1, '+')$



S2

$\text{Inchidere}(\{S \rightarrow E + \cdot S, \$\})$

Intrebari

1. Calculati: **Inchidere** ($I = \{S \rightarrow E + \cdot S, \$\}$)

Gramatica

$S' \rightarrow S\$$

$S \rightarrow E + S \mid E$

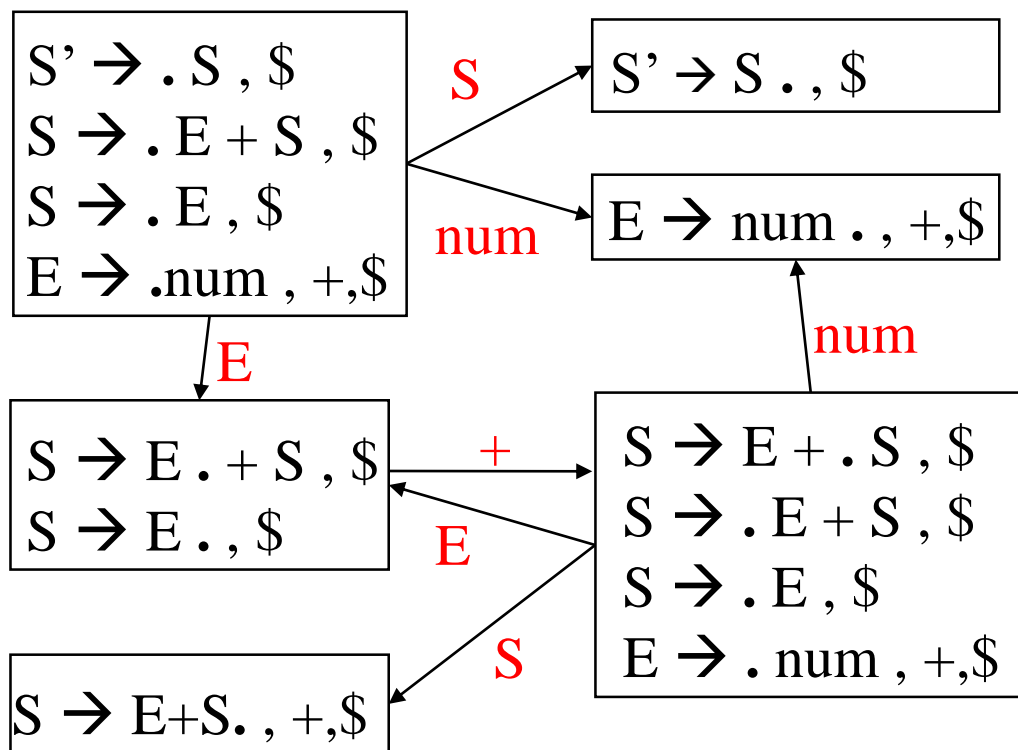
$E \rightarrow \text{num}$

2. Calculati: **Goto**(I, num)

3. Calculati: **Goto**(I, E)

Construirea AFD pt. LR(1)

- Dacă $S' = \text{goto}(S, x)$ atunci adauga arc etichetat x de la S la S'



Gramatica

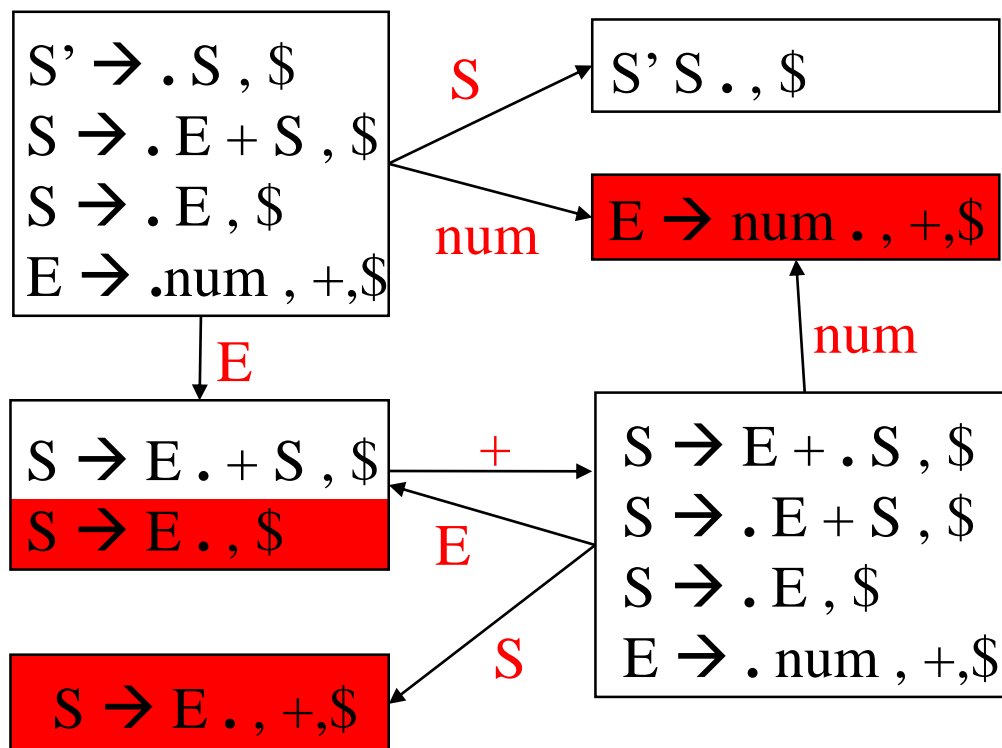
$S' \rightarrow SS$

$S \rightarrow E + S \mid E$

$E \rightarrow \text{num}$

Reducerile in LR(1)

- Reducerile corespund elementelor LR(1) de forma $(X \rightarrow \gamma . , y)$



Gramatica

$S' \rightarrow S\$$

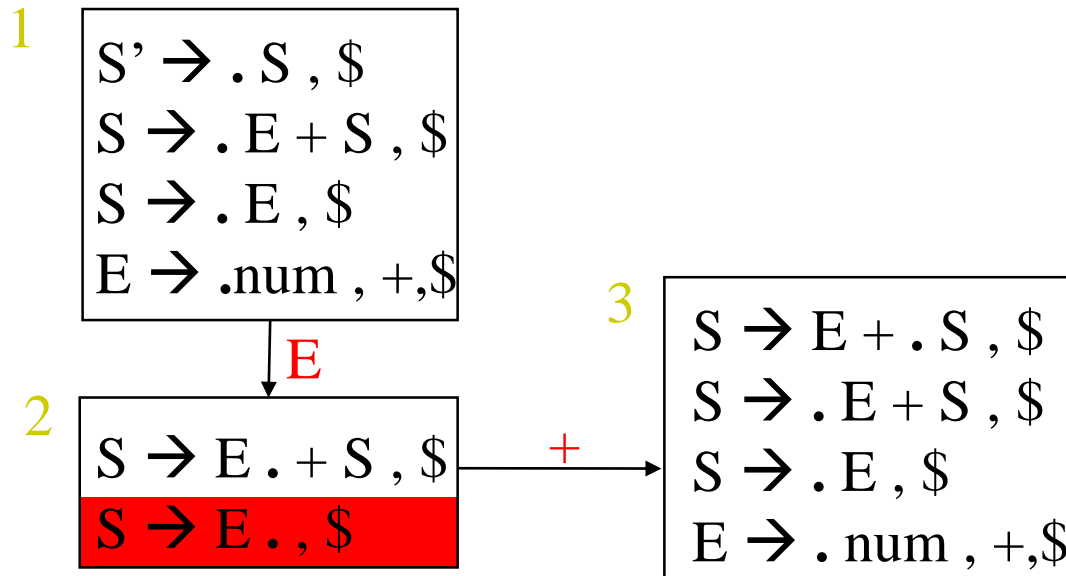
$S \rightarrow E + S \mid E$

$E \rightarrow num$

Construirea tabelelor LR(1)

- La fel ca la LR(0), cu exceptia reducerilor
- Pentru o tranzitie $S \rightarrow S'$ pe terminalul x :
 - $\text{Action}[S, x] = \text{Shift}(S')$
- Pt. o tranzitie $S \rightarrow S'$ pe neterminalul N :
 - $\text{Goto}[S, N] = S'$
- Daca I contine $\{(X \rightarrow \gamma \cdot, y)\}$ atunci:
 - $\text{Action}[I, y] = \text{Reduce}(X \rightarrow \gamma)$

LR(1) Exemplu de tabela



Gramatica

$S' \rightarrow S\$$

$S \rightarrow E + S \mid E$

$E \rightarrow num$

Fragment din tabela
de parsare

	+	\$	E	
1				g2
2		s3	S→E	

Analizoare LALR(1)

- Problema principala cu LR(1): prea multe stari
- Parsarea LALR(1) (aka LookAhead LR)
 - Construiește AFD LR(1) și apoi reunește orice două stări LR(1) a căror elemente sunt identice cu excepția lookahead-ului
 - Produce în practică tabele mult mai mici
 - Teoretic mai puțin puternic decât LR(1), practic diferențele se întâlnesc foarte rar
- Gramatica LALR(1) = o gramatică a cărei tabelă de parsare LALR(1) nu are conflicte
- Limbaj LALR(1) = un limbaj ce poate fi definit de o gramatică LALR(1)

Parsearele LALR

- LALR(1)
 - In general, cam acelasi numar de stari ca si SLR (mult mai putine ca LR(1))
 - Dar, cu “putere de analiza” comparabila cu LR(1) (mult mai bine decat SLR)

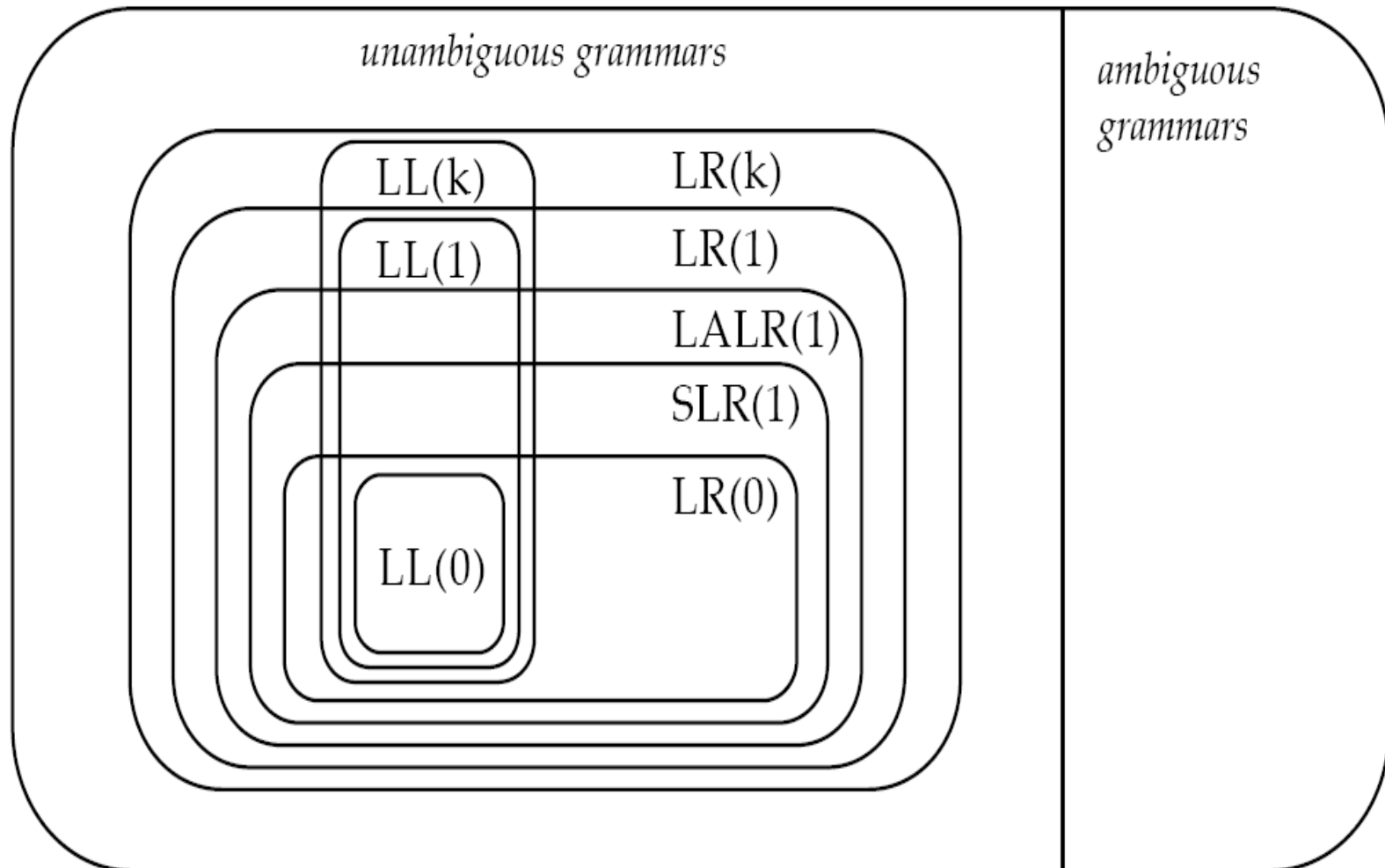
LR(0) vs. SLR vs. LR vs. LALR

- LR(0) – cand se poate reduce – reduce indiferent de lookahead
- SLR(1) – reducem $A \rightarrow u$ doar pt. terminalii $a \in \text{FOLLOW}(A)$
- LR(1) – imparte starile a.i. sa includa si informatie de 'prefix'
- LALR(1) – uneste starile care difera doar prin multimea lookahead/FOLLOW (face reuniune pe multimi)
- $\text{LR}(0) < \text{SLR}(1) < \text{LALR}(1) < \text{LR}(1)$
- yacc/bison e LALR

Limbaje LL vs. LR

- Stim ca unele gramatici nu sunt LL(k) – dar pot fi transformate. Exista limbaje LL(k) ce nu sunt LR(k)?
- $L = \{ x^{2n}y^{2n}e, x^{2n+1}y^{2n+1}o \}$
 - LL(k) nu se descurca, oricare ar fi k
 - LR(1) se descurca – asteapta sa citeasca 'e' sau 'o'
 - Puteti scrie gramatica? Care e dif. LL(k) vs LL(∞)?
- Totusi, LL(∞) e la fel de puternic ca LR(∞) !
- LR(1) \sim aut. cu stiva determinist; mai slab ca LL(∞)

Clasificarea analizoarelor



Automatizarea procesului de parsare

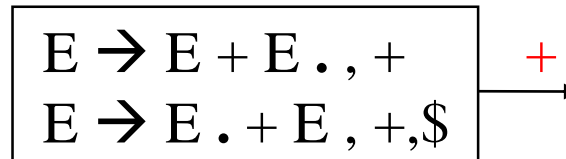
- Se poate automatiza:
 - Construirea tabelelor de parsare LR/LL
 - Construirea parserelor shift-reduce/top-down bazate pe aceste tabele
- Generatoarea de parsere LALR(1)
 - yacc, bison
 - In practica, nu e diferenta mare fata de LR(1)
 - Tabele de parsare mai mici decat LR(1)
 - Extind specificatia gramaticilor LALR(1) cu declaratii de **precedenta, asociativitate**
 - Output: program de parsare LALR(1)

Asociativitatea

$$S \rightarrow S + E \mid E$$
$$E \rightarrow \text{num}$$

$$E \rightarrow E + E$$
$$E \rightarrow \text{num}$$

Ce se intampla daca incercam sa construim o tabela LALR?

$$E \rightarrow E + E$$
$$E \rightarrow \text{num}$$

$$1 + 2 + 3$$

↑

shift/reduce
conflict

shift: $1 + (2 + 3)$
reduce: $(1 + 2) + 3$

Asociativitatea (2)

- Daca un operator este asociativ stanga
 - Asignam o valoare mai mare de precedenta daca e pe stiva de parsare decat daca e pe sirul de intrare
 - Deoarece precedenta stivei e mai mare, reducerea va fi prioritara (ceea ce e corect in cazul asociativitatii stanga)
- Daca operatorul e asociativ dreapta
 - Asignam o valoare mai mare daca e pe sirul de intrare
 - Deoarece sirul de intrare are precedenta, operatia de shift va avea precedenta (ceea ce e corect in cazul asociativitatii dreapta)

Precedenta

$$E \rightarrow E + E \mid T$$
$$T \rightarrow T \times T \mid \text{num} \mid (E)$$

$$E \rightarrow E + E \mid E \times E \mid \text{num} \mid (E)$$

Ce se intampla daca incercam sa construim o tabela LALR?

$$E \rightarrow E . + E , \dots$$
$$E \rightarrow E \times E . , +$$
$$E \rightarrow E + E . , \times$$
$$E \rightarrow E . \times E , \dots$$

Shift/reduce
conflict results

Precedenta: ataseaza indicatori de precedenta tokenilor
Conflictul shift/reduce e rezolvat astfel:

1. Daca precedenta token-ului de intrare mai mare decat ultimul terminal de pe stiva, favorizeaza 'shift'
2. Daca precedenta token-ului de intrare mai mica sau egala decat cea a ultimului terminal de pe stiva, favorizeaza 'reduce'

Automatizarea procesului de parsare - ANTLR

- Problema principala la LL – rescrierea gramaticii
- recursivitatea stanga
 - Foloseste notatie EBNF, mai usor de evitat recursivitatea stanga
- factorizarea stanga
 - Foloseste predicate sintactice -> backtracking izolat pt. a evita rescrierea gramaticii sau cresterea lookahead-ului
- Ambiguitati
 - rezolvate in YACC cu precedenta/asociativitate
 - Rezolvate in ANTLR tot cu predicate (semantice)

Sumar – analiza LL/LR

- Tabelele de parsare LL
 - Neterminali x terminali \rightarrow productii
 - Calculate folosind FIRST/FOLLOW
- Tabelele de parsare LR
 - Stari LR x terminali \rightarrow action {shift/reduce}
 - Stari LR x neterminali \rightarrow goto
 - Calculate folosind inchideri/operatii goto pe stari LR
- Spunem ca o gramatica e:
 - LL(1) daca tabela LL(1) nu are conflicte
 - La fel si la LR(0), SLR, LALR(1), LR(1)

Tratarea erorilor

- Detectie
 - Exista erori de sintaxa ?
 - Care este locul erorii?
- Recuperare din eroare
 - Care sunt toate erorile de sintaxa?
 - Continua analiza dupa prima eroare.
- Corectarea erorilor
 - Furnizeaza un arbore sintactic partial corect

Detectia erorii

- Primul caracter ce nu poate fi parsat nu este neaparat locul erorii
 - Exemplu: lipsa unui terminator – ; } – raportata ca o eroare in instructiunea urmatoare
- Cel mai bine - detectia imediata
 - cand simbolul gresit apare ca lookahead
- Mai usor de implementat
 - cand se face Match / Shift
- Strong-LL(k) vs. full LL(k)

Recuperare din eroare

- LR – se completeaza casutele lipsa cu actiuni ce trateaza eroarea (manual!)
 - De ex. modul “panica”: scaneaza pana la urmatorul token ce ar putea termina o productie (i.e **; } end**)
- LR – error token
 - Se pune in varful stivei un token special, *error*
 - Se adauga reguli de tratat eroarea – de ex. Statement → *error* ;
 - Se scot stari de pe stiva stari pana se poate face o reducere
- LL – FOLLOW set
 - Se citeste intrarea pana apare primul token din FOLLOW()
 - Pentru productia curenta, sau si pentru cele de deasupra din stiva

Corectarea erorilor

- Se insereaza / sterg numarul minim de tokeni pentru a obtine un sir din limbajul descris e gramatica.
- Unii algoritmi doar insereaza tokeni (pentru a putea parsa un text incomplet).
 - vezi Jacobs / Grune – Parsing Techniques

LL vs. LALR

- LL mai usor de scris 'de mana'; LALR, doar cu generatoare de parsere
 - Un generator te poate ajuta sa gasesti conflicte in gramatica
- LL mai simplu, ceva mai usor de depanat
- LALR poate parsa mai multe gramatici
 - dar orice limbaj de programare 'normal' poate fi parsat cam la fel de usor cu LL si LALR
- Conditionare – gramatica trebuie transformata pentru LL (vezi recursivitate stanga...)
 - Daca te descurci cu expresiile, restul e mai usor
 - Dar gramaticile LL tind sa fie mai mari...

LL vs. LALR (cont)

- Actiunile semantice mai simple in LL
 - Cod executat oricand in timpul parsarii unei productii, nu doar la momentul reduce
 - Se pot folosi nume de variabile pentru attribute
 - Mai multe amanunte – la analiza semantica
- Recuperarea din eroare e mai simpla in LL
- Eficienta – viteza de rulare si memoria consumata sunt comparabile, in general
 - Pt parsere generate automat
- In cele din urma, e o problema de disponibilitate a uneltelor – fiecare foloseste un 'parser generator' potrivit pentru limbajul pe care il utilizeaza.

Arbori sintactici

- “Abstract syntax trees”
- Arbori de parsare simplificati, fara nodurile suplimentare care nu adauga informatie
- Reprezinta ‘gramatica initiala’ si nu gramatica scrisa de noi.
- Este ‘good engineering’ sa construim arborii sintactici din “actiunile semantice” ale parserelor
 - Pastreaza in fisierele cu gramatica doar descrierea limbajului
 - E gresit sa faci prea multe lucruri odata, din acelasi cod

Backup slides

Algoritmi generici de parsare


GLR (Tomita)

- Suporta gramatici non LR(1), ambigue
- Automat nedeterminist
- Duplica stiva in cazul unui conflict shift/reduce

Exemplu:

Gramatica: $S \rightarrow v=E$; $E \rightarrow E+E$; $E \rightarrow id$

La intrare: $v=id_1+id_2+id_3\$$

$v=E+E \cdot +id_3\$$  $v=E+E+ \cdot id_3\$$ (shift)
 $v=E \cdot +id_3\$$ (reduce)

GLR (Tomita)

- Reducerea numarului de stari
 - Combina prefixele comune
 - Combina stivele ce se termina in aceeasi stare
 - Rezultatul – stiva sub forma de graf orientat

$$\begin{array}{l} v = E + E + \quad . \text{id}_3 \$ \\ v = E + \quad . \text{id}_3 \$ \end{array} \quad \longrightarrow \quad v = \begin{array}{l} \swarrow E + E + \\ \searrow E + \end{array} \quad . \text{id}_3 \$$$

- Complexitatea?
- Arborele sintactic e disponibil doar la sfarsitul parsarii

Algoritmi generici top-down

- Unger (1968)
- Acopera limbajele independente de context.
- Divide et impera

Exemplu: $S \rightarrow ABC \mid DE$; input: pqrs

Subprobleme:

$A \rightarrow p, B \rightarrow q, C \rightarrow rs$

$D \rightarrow p, E \rightarrow qrs$

$A \rightarrow p, B \rightarrow qr, C \rightarrow s$

$D \rightarrow pq, E \rightarrow rs$

$A \rightarrow pq, B \rightarrow r, C \rightarrow s$

$D \rightarrow pqr, E \rightarrow s$

- Trebuie detectate derivarile care pot forma bucle.
- Complexitate?

Algoritmi generici bottom-up

- CYK (Cocke-Younger-Kasami, 1967)
- Acopera limbajele independente de context.
- Reducerea gramaticilor la forma normala Chomsky

Toate regulile de forma $A \rightarrow a$ sau $A \rightarrow BC$

$A \rightarrow \varepsilon | \gamma$; $B \rightarrow \alpha A \beta$ se transforma in

$B \rightarrow \alpha A' \beta$; $B \rightarrow \alpha \beta$; $A' \rightarrow \gamma$

- Programare dinamica
- Complexitate?