Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

(Week 11) Lecture 21 & 22

Objectives: Learning objectives of these lectures are

Students will able to understand:

- What is Bottom-Up Parser?
- What are types of Bottom-Up Parser?
- CLR(1) Parser Steps
    1. Number the Production
    2. Augment the given grammar (Augmented Grammar)
    3. Draw the Canonical Collection
    4. Create the Parsing Table
    5. Stack Implementation
    6. Draw Parse Tree

Text Book & Resources:

Compilers Principles Techniques and Tools (2nd Edition) by Alfread V. Aho, Ravi Sethi.

Videos Links:

https://youtu.be/OFSM7gV66zE (Part 1)
https://youtu.be/GapyvMhJdDU (Part 2)
https://youtu.be/F64pAwqDqQ4 (Part 3)
https://youtu.be/iduYikgpcNE (Part 4)
https://youtu.be/Gf5vvA0k21k (Part 5)
https://youtu.be/Ctl6A4p9SEo (Part 6)
https://youtu.be/cyYZuL2GELk (Part 7)
https://youtu.be/1Nr58CngFwY (Part 8)

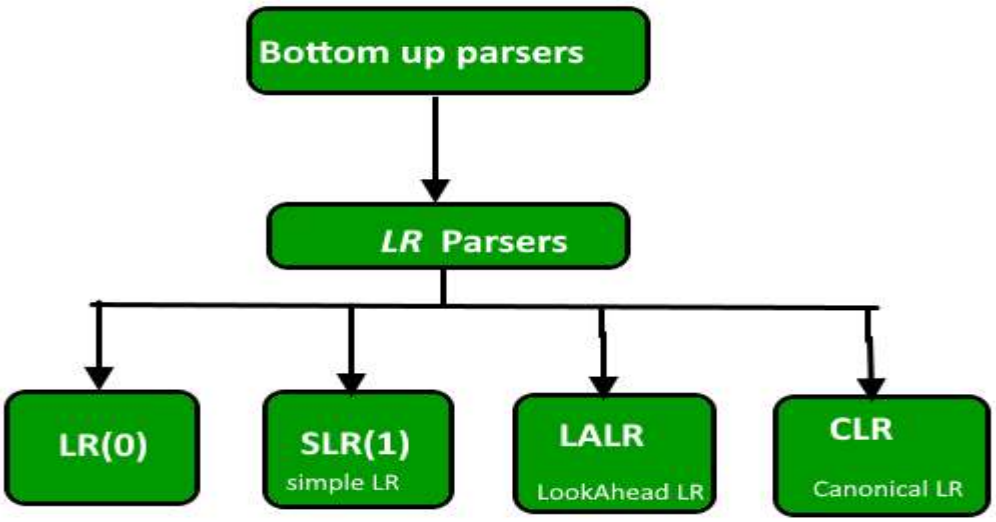
Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Bottom Up or Shift Reduce Parsers

In the last week, we had discussed the Bottom Up Parsers and also discussed the following types of parsers. From these parsers we had discussed SLR(1) Parser and in this week we will discuss the Canonical LR or CLR(1) Parser.

Classification (Types) of bottom up parsers



CLR (1) Parsing

CLR refers to canonical look-ahead. CLR parsing use the canonical collection of LR (1) items to build the CLR(1) parsing table. CLR (1) parsing table produces the more number of states as compare to the SLR (1) parsing.
In the CLR (1), we place the reduce node only in the look-ahead symbols.

Canonical LR Parsing

Algorithm:

SetOfItems CLOSURE(I) {
    repeat
        for (each item $[A \to \alpha \cdot B\beta, a]$ in I)
            for (each production $B \to \gamma$ in G')
                for (each terminal b in FIRST(βa))
                    add $[B \to \cdot\gamma, b]$ to set I;
    until no more items are added to I;
    return I;
}

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

```
SetOfItems GOTO(I, X) {
        initialize J to be the empty set;
        for ( each item [A → α·Xβ, a] in I )
                add item [A → αX·β, a] to set J;
        return CLOSURE(J);
}
void items(G') {
        initialize C to CLOSURE({[S' → ·S, $]});
        repeat
                for ( each set of items I in C )
                        for ( each grammar symbol X )
                                if ( GOTO(I, X) is not empty and not in C )
                                        add GOTO(I, X) to C;
        until no new sets of items are added to C;
}
```

Construction of Canonical LR Parsing Table

Algorithm:

1. Construct $C' = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of LR(1) items for G'.

2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.

   (a) If $[A \to \alpha \cdot a\beta, b]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot, a]$ is in I_i, $A \neq S'$, then set ACTION$[i, a]$ to "reduce $A \to \alpha$."

   (c) If $[S' \to S \cdot, \$]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S, \$]$.

Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

CLR(1) Parser Steps

1. Number the Production
2. Augment the given grammar (Augmented Grammar)
3. Draw the Canonical Collection(DFD)
4. Create the Parsing Table
5. Stack Implementation
6. Draw Parse Tree

Example # 1

Consider the grammar

S →AA

A → aA | b

Step 1: Number the Production

1. S →AA
2. A → aA
3. A → b

Step 2: Augmented CFG

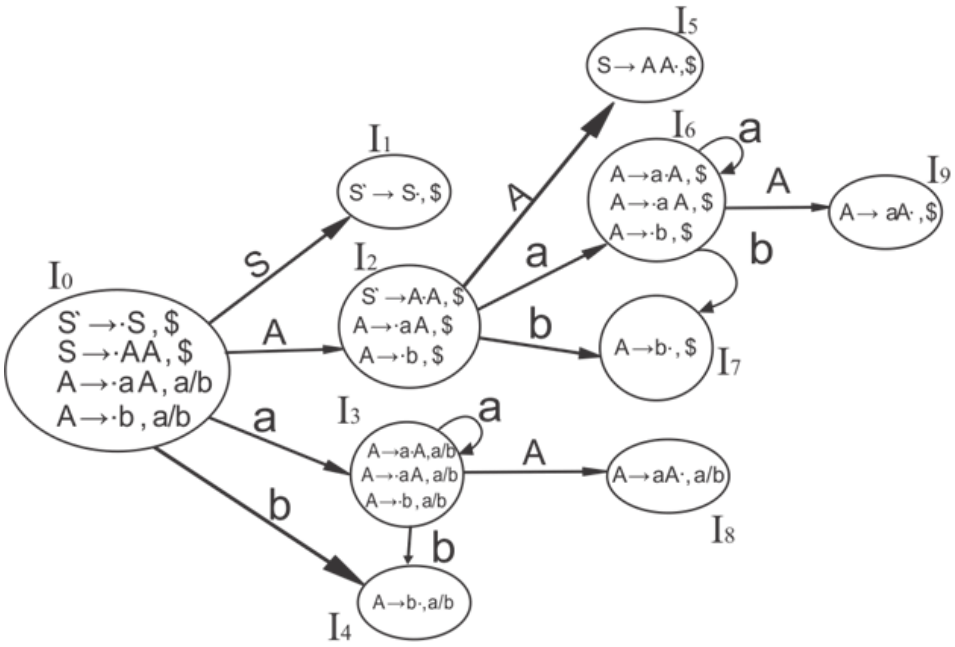
S'→S

S →AA

A → aA

A → b

Step 3: Canonical Collection



Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

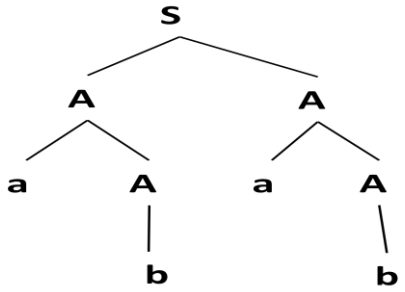
Step 4: Parsing Table (R1: S→AA , R2: A→aA , R3: A→b)

States	Action Part			Goto Part	
	a	**b**	**$**	**S**	**A**
I_0	S3	S4		1	2
I_1			Accept		
I_2	S6	S7			5
I_3	S3	S4			8
I_4	R3	R3			
I_5			R1		
I_6	S6	S7			9
I_7			R3		
I_8	R2	R2			
I_9			R2		

Step 5: Stack Implementation (String: abab)

Stack	Input	Action
$0	abab$	Shift a→S3
$0a3	bab$	Shift b→S4
$0a3b4	ab$	Reduction (A→b) (R3)
$0a3A8	ab$	Reduction (A→aA) (R2)
$0A2	ab$	Shift a→S6
$0A2a6	b$	Shift b→S7
$0A2a6b7	$	Reduction (A→b) (R3)
$0A2a6A9	$	Reduction (A→aA) (R2)
$0A2A5	$	Reduction (S→AA) (R1)
$0S1	$	Accept

Step 6: Draw Syntax Tree



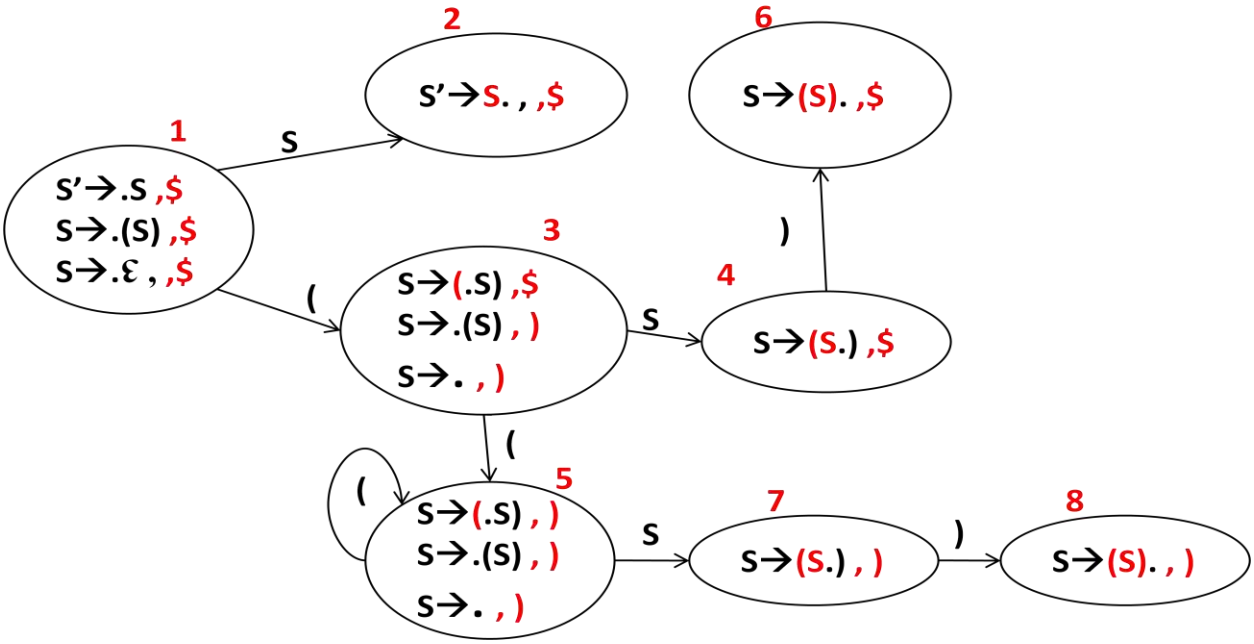
Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Example # 2
Given Grammar
S→(S) | ∈
Step # 1: Number the Production
1. S→(S)
2. S→∈

Step # 2: Augmented Grammar
S'→S
S→(S)
S→∈

Step 3: Canonical Collection



Note: S→.∈ = S→. so there is to need to check next symbol

Step 4: Parsing Table (R1: S→(S) , R2: S→∈)

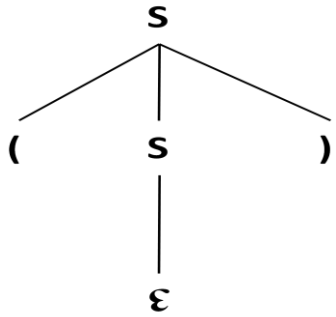
States	Action Part			Goto Part
	(	)	$	S
1	S3		R2	2
2			Accept	
3	S5	R2		4
4		S6		
5	S5	R2		7
6			R1	
7		S8		
8		R1		

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 5: Stack Implementation (String: ( ))

Stack	Input	Action
$1	( )$	Shift (→S3
$1(3	)$	Reduction (S→∈ (R2)
$1(3S4	)$	Shift)→S6
$1(3S4)6	$	Reduction (S→(S) (R1)
$1S2	$	Accept

Step 6: Draw Syntax Tree



Example # 3

In this example we will perform all above steps to check whether the given grammar is CLR(1) conflict or not.

Given CFG:

E→E+(E)
E→id

Given String:

id+(id)

Step 1: Number the Production

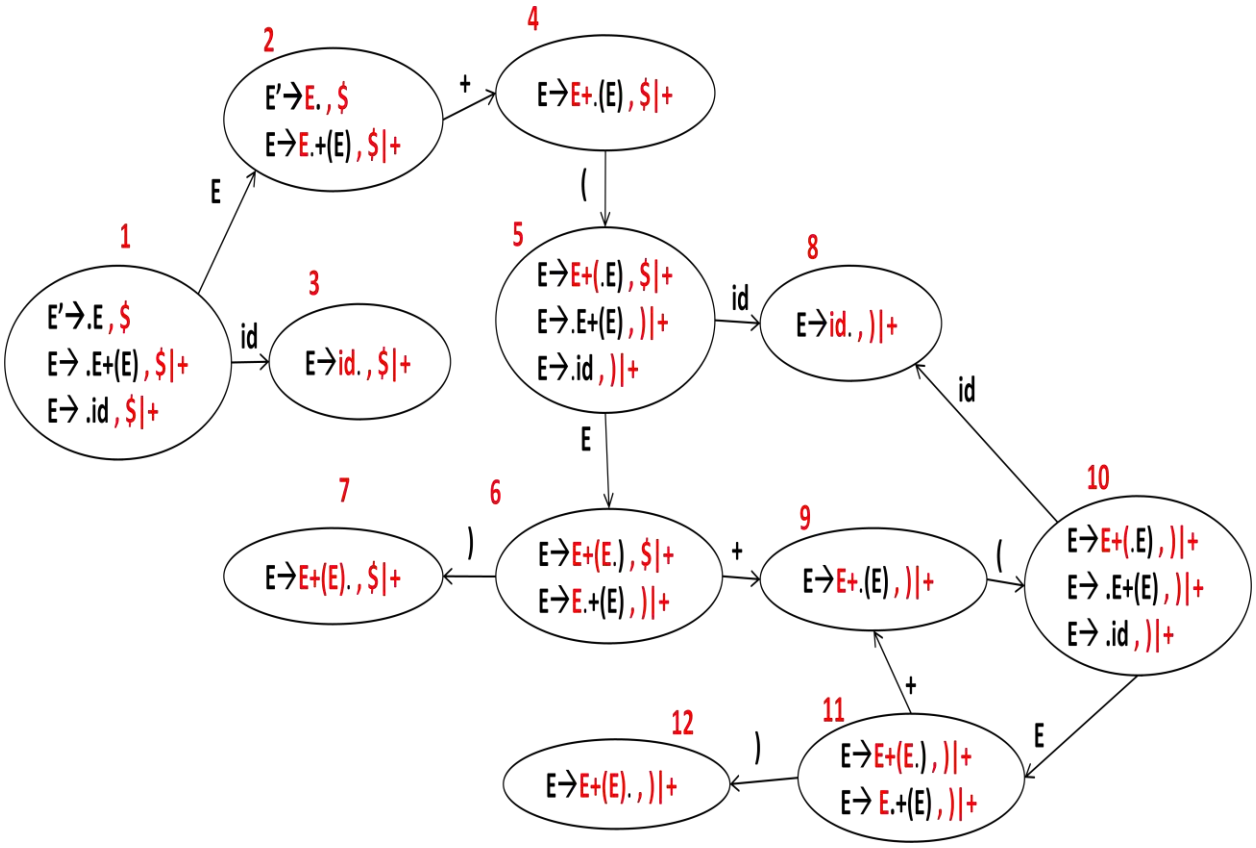
1. E→E+(E)
2. E→id

Step 2: Augmented CFG

E'→E
E→E+(E)
E→id

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 3: Canonical Collection



Step 4: Parsing Table (R1: E→E+(E) , R2: E→id)

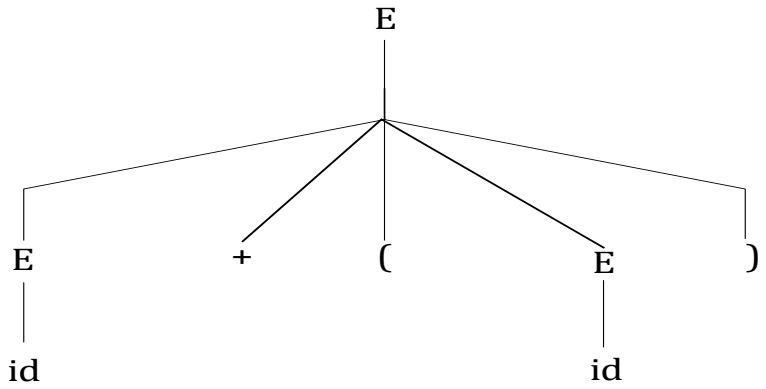
States	Action Part					Goto Part
	+	(	Id	)	$	E
1			S3			2
2	S4				Accept	
3	R2				R2	
4		S5				
5			S8			6
6	S9			S7		
7	R1				R1	
8	R2			R2		
9		S10				
10			S8			11
11	S9			S12		
12	R1			R1		

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 5:Stack Implementation (String: id+(id))

Stack	Input	Action
$1	id+(id)$	Shift id→S3
$1id3	+(id)$	Reduction (E→id) (R2)
$1E2	+(id)$	Shift +→S4
$1E2+4	(id)$	Shift (→S5
$1E2+4(5	id)$	Shift id→S8
$1E2+4(5id8	)$	Reduction E→id (R2)
$1E2+4(5E6	)$	Shift)→S7
$1E2+4(5E6)7	$	Reduction E→E+(E) (R1)
$1E2	$	Accept

Step 6: Draw Syntax Tree



Example # 4

Given CFG

E → E + T | T
T → T * F | F
F → id

In this example we will perform all above steps to check whether the given grammar is CLR(1) conflict or not.

Step 1: Number the Production

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → id

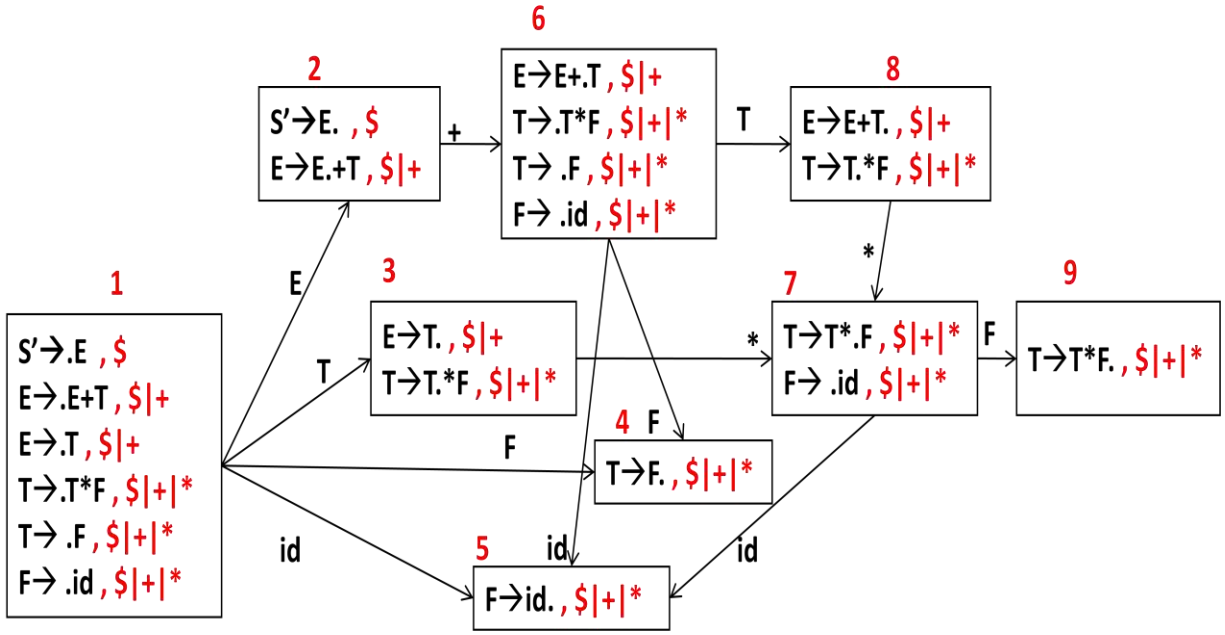
Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 2: Augmented CFG

S'→E
E → E + T
E → T
T → T * F
T → F
F → id

Step 3: Canonical Collection (DFD)



Step 4: Parsing Table (R1: E→E+T , R2:E→T,R3: T→T*F, R4: T→F, R5: F→id)

States	Action Part				Goto Part		
	id	+	*	$	E	T	F
1	S5				2	3	4
2		S6		Accept			
3		R2	S7	R2			
4		R4	R4	R4			
5		R5	R5	R5			
6	S5					8	4
7	S5						9
8		R1	S7	R1			
9		R3	R3	R3			

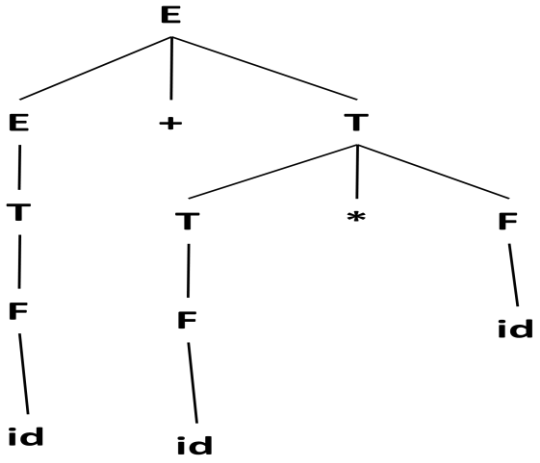
Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 5: Stack Implementation (String: id+id*id)

Stack	Input	Action
$1	id+id*id$	Shift id→S5
$1id5	+id*id$	Reduction (F→id) (R5)
$1F4	+id*id$	Reduction (T→F) (R4)
$1T3	+id*id$	Reduction (E→T) (R2)
$1E2	+id*id$	Shift +→S6
$1E2+6	id*id$	Shift id→S5
$1E2+6id5	*id$	Reduction (F→id) (R5)
$1E2+6F4	*id$	Reduction (T→F) (R4)
$1E2+6T8	*id$	Shift *→S7
$1E2+6T8*7	id$	Shift id→S5
$1E2+6T8*7id5	$	Reduction (F→id) (R5)
$1E2+6T8*7F9	$	Reduction (T→T*F) (R3)
$1E2+6T8	$	Reduction (E→E+T) (R1)
$1E2	$	Accept

Step 6: Draw Syntax Tree



Compiler Construction (CS-636)

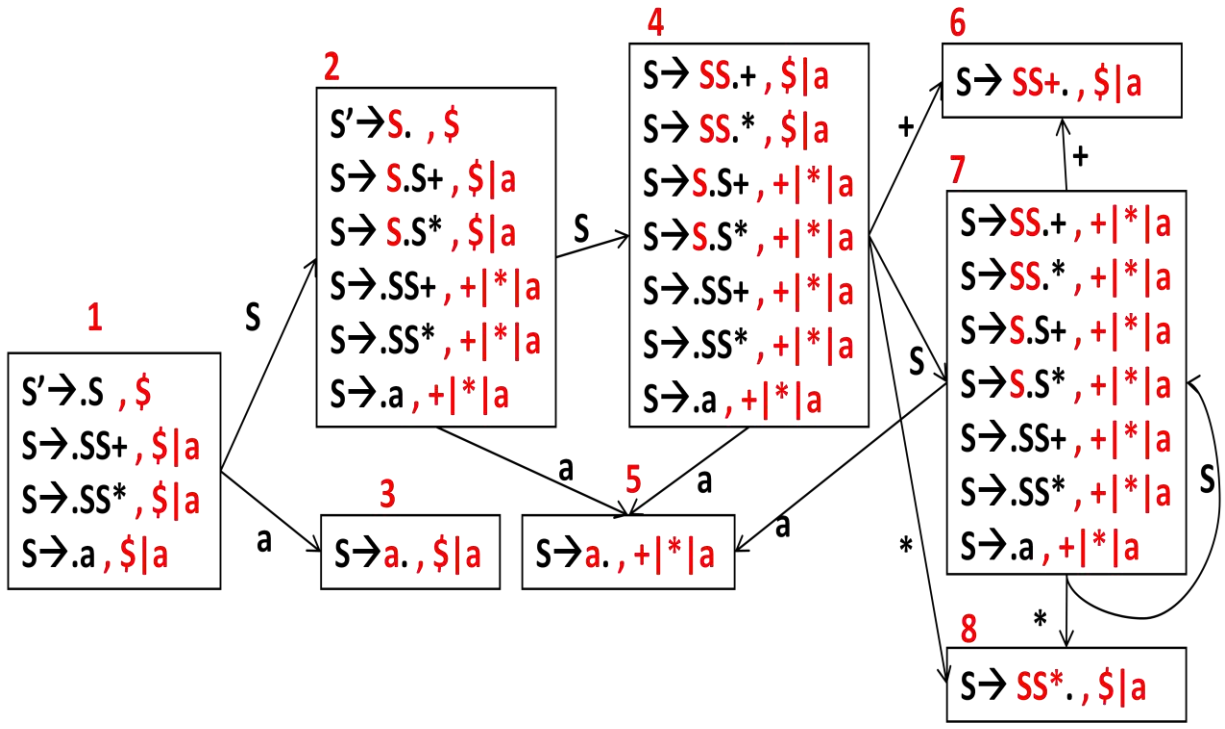
Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Example # 5
Given Grammar
S→ SS+ | SS*| a
Step # 1: Number the Production

1. S→SS+
2. S→ SS*
3. S→a

Step # 2: Augmented Grammar
S'→S
S→SS+
S→ SS*
S→a

Step 3: Canonical Collection of LR(1) Items



Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 4: Parsing Table (R1: S→SS+ , R2: S→SS* , R3: S→a)

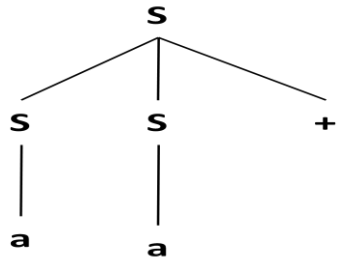
States	Action Part				GoTo Part
	+	*	a	$	S
1			S3		2
2			S5	**Accept**	4
3			R3	R3	
4	S6	S8	S5		7
5	R3	R3	R3		
6			R1	R1	
7	S6	S8	S5		7
8			R2	R2	

Step5: Stack Implementation (String: aa+)

Stack	Input	Action
$1	aa+$	Shift a→S3
$1a3	a+$	Reduction R3(S→a)
$1S2	a+$	Shift a→S5
$1S2a5	+$	Reduction R3(S→a)
$1S2S4	+$	Shift +→S6
$1S2S4+6	$	Reduction R1(S→SS+)
S1S2	$	Accept

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 6: Draw Syntax Tree

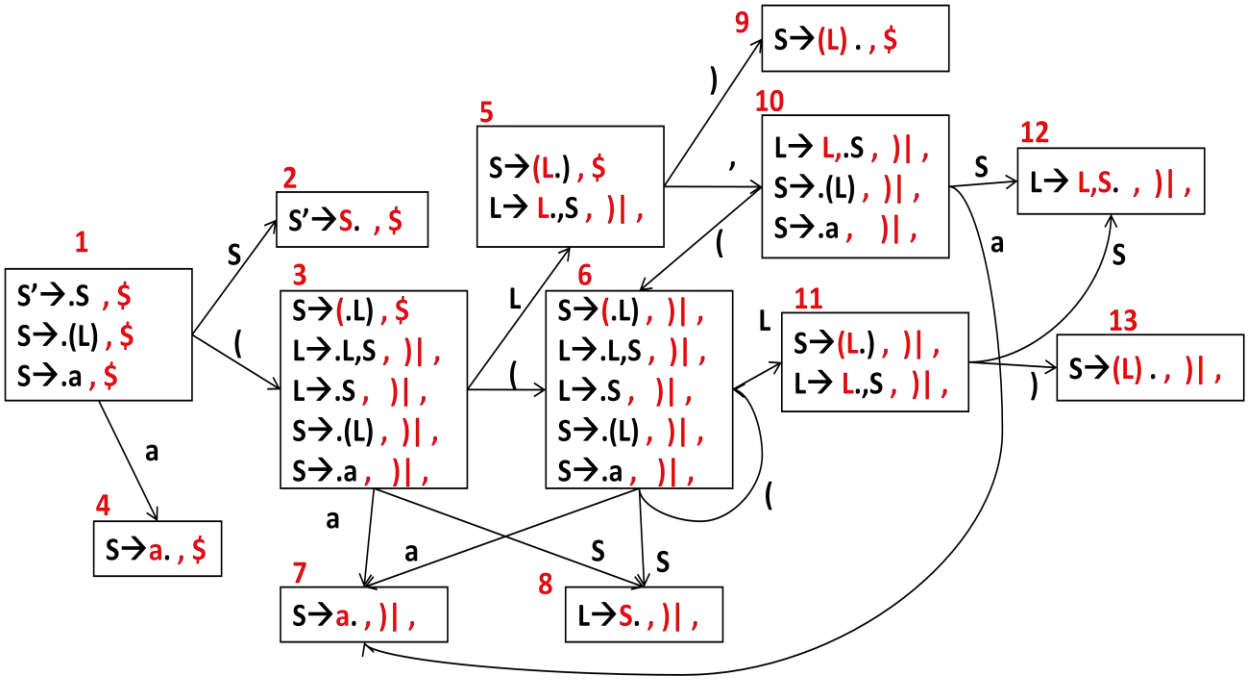


Example # 6
Given Grammar
S→ (L) | a
L→ L,S | S
Step # 1: Number the Production

1. S→ (L)
2. S→ a
3. L→ L,S
4. L→ S

Step # 2: Augmented Grammar
S'→S
S→ (L)
S→ a
L→ L,S
L→ S

Step 3: Canonical Collection



Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 4: Parsing Table (R1: S→(L) , R2: S→a , R3: L→L,S , R4: L→S)

States	Terminal Part					Goto Part	
	(	)	a	,	$	S	L
1	S3		S4			2	
2					Accept		
3	S6		S7			8	5
4					R2		
5		S9		S10			
6	S6		S7			8	11
7		R2		R2			
8		R4		R4			
9					R1		
10	S6		S7			12	
11		S13				12	
12		R3		R3			
13		R1		R1			

Compiler Construction (CS-636)

Dr. Naseer Ahmed Sajid email id: naseer@biit.edu.pk Whatsapp# 0346-5100010

Step 5 : Stack Implementation (String: (a,a))

Stack	Input	Action
$1	(a , a)$	Shift (→S3
$1(3	a , a)$	Shift a→S7
$1(3a7	,a)$	Reduction R2(S→a)
$1(3S8	,a)$	Reduction R4(L→S)
$1(3L5	,a)$	Shift , →S10
$1(3L5,10	a)$	Shift a →S7
$1(3L5,10a7	)$	Reduction R2(S→a)
$1(3L5,10S12	)$	Reduction R3(L→L,S)
$1(3L5	)$	Shift) →S9
$1(3L5)9	$	Reduction R1(S→(L)
$1S2	$	Accept

Step 6: Draw Syntax Tree