HW3

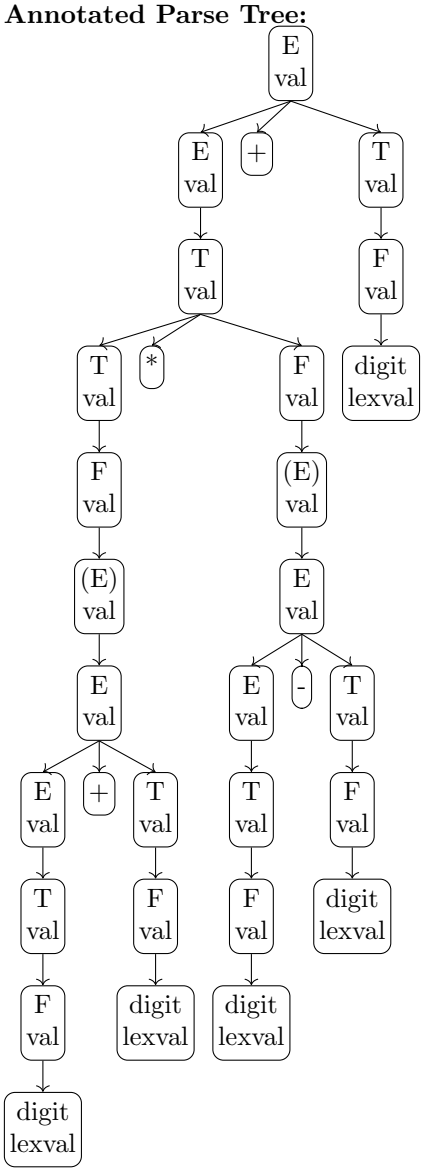
Mojtaba Mollaei
40131383

May 26, 2025

1

Expression: $(4 + 2) * (9 - 3) + 5$

Annotated Parse Tree:



Final Result: $E.val = 36 + 5 = \boxed{41}$

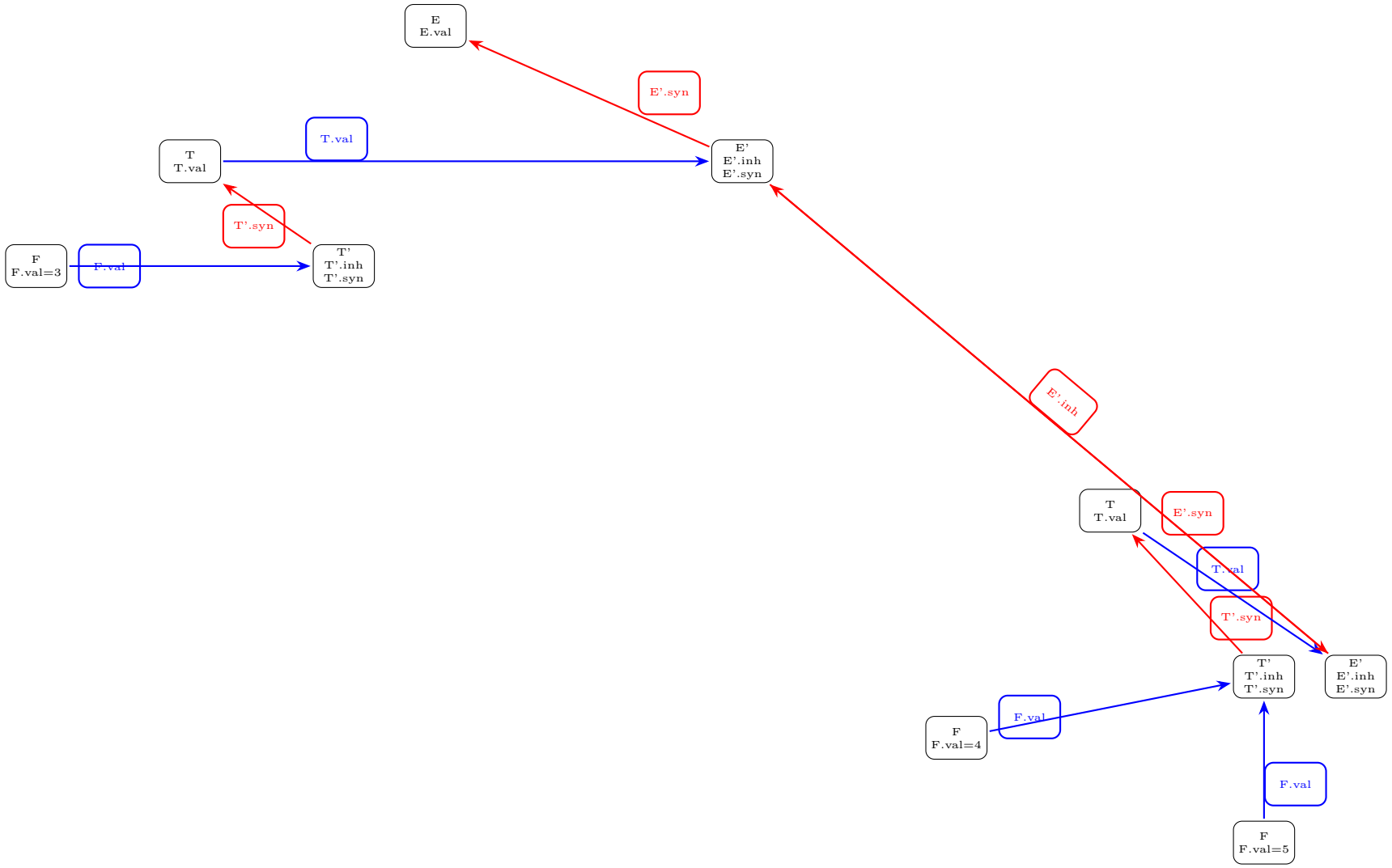
2

Production Rules and Semantic Actions:

1.	$E \to T\ E'$	$E'.inh = T.val, \quad E.val = E'.syn$
2.	$E' \to +\ T\ E'_1$	$E'_1.inh = E'.inh + T.val, \quad E'.syn = E'_1.syn$
3.	$E' \to \epsilon$	$E'.syn = E'.inh$
4.	$T \to F\ T'$	$T'.inh = F.val, \quad T.val = T'.syn$
5.	$T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh * F.val, \quad T'.syn = T'_1.syn$
6.	$T' \to \epsilon$	$T'.syn = T'.inh$
7.	$F \to digit$	$F.val = digit.lexval$

Input String: $3 + 4 \times 5$

Dependency Graph:


E
E.val
E'.syn
T
T.val
T.val
E'
E'.inh
E'.syn
T'.syn
F
F.val=3
F.val
T'
T'.inh
T'.syn
E'.inh
T
T.val
E'.syn
T.val
T'.syn
T'
T'.inh
T'.syn
E'
E'.inh
E'.syn
F
F.val=4
F.val
F.val
F
F.val=5

Final Evaluated Values:

- `F1.val = 3`, `T'.inh = 3` $\rightarrow$ `T.val = 3`
- `F2.val = 4`, `F3.val = 5`, `T'.inh = 4`, `T'.syn = 20` $\rightarrow$ `T.val = 20`
- `E'.inh = 3`, `E'.syn = 3 + 20 = 23` $\rightarrow$ `E.val = 23`

3

Grammar 1

$$S \rightarrow A\ B$$
$$A.\text{val} = 1$$
$$B.\text{inh} = A.\text{val}$$
$$B.\text{val} = B.\text{inh} + 2$$

Classification: L-attributed but not S-attributed
Justification:

- **S-attributed** grammars allow only synthesized attributes. This grammar uses the inherited attribute `B.inh`, so it is not S-attributed.
- **L-attributed** grammars allow inherited attributes as long as each inherited attribute of a symbol on the right-hand side depends only on:
  - The attributes of symbols to its left in the production
  - The inherited attributes of the head

  Here, `B.inh` depends on `A.val`, which is to the left of `B`, so the grammar is L-attributed.

Grammar 2

$$E \rightarrow E_1 + T$$
$$E.\text{val} = E_1.\text{val} + T.\text{val}$$

$$E \rightarrow T$$
$$E.\text{val} = T.\text{val}$$

$$T \rightarrow \text{digit}$$
$$T.\text{val} = \text{digit.lexval}$$

Classification: S-attributed
Justification:

- All attributes in the grammar are synthesized:
  - `E.val`, `T.val` are computed based on the synthesized attributes of their children.
- There are no inherited attributes, so this grammar is S-attributed.
- Since all S-attributed grammars are also L-attributed, this grammar is also L-attributed.

Grammar 3

$$L \to L_1,\ id$$
$$id.\text{pos} = L_1.\text{pos} + 1$$
$$L.\text{pos} = id.\text{pos}$$

$$L \to id$$
$$id.\text{pos} = 1$$
$$L.\text{pos} = id.\text{pos}$$

Classification: L-attributed but not S-attributed
Justification:

- The attribute `id.pos` is inherited from `L-1.pos`, so inherited attributes are used.
- Since `id.pos` depends on `L1.pos`, which is to the left of `id`, this fits the L-attributed grammar constraints.
- The presence of inherited attributes means it is not S-attributed.

4

1. LR(k)

- **Power:** Most powerful among the listed methods. Can recognize the largest class of context-free grammars for any fixed $k \geq 1$.
- **Simplicity:** Most complex to implement. Parsing tables grow exponentially with k, making it impractical for large k. Rarely used in practice beyond $k = 1$.

2. LR(1)

- **Power:** Can handle all deterministic context-free grammars that require only one symbol of lookahead.
- **Simplicity:** More practical than LR(k) but still generates large parsing tables (many states). Used in some parser generators (e.g., GNU Bison supports canonical LR(1)).

3. LALR(1) (Look-Ahead LR)

- **Power:** Less powerful than full LR(1) but more powerful than SLR(1). Can handle many practical programming language grammars.
- **Simplicity:** Merges LR(1) states with identical cores to reduce table size, making it similar in size to SLR(1) tables. Commonly used in practice (e.g., Yacc, Bison).

4. SLR(1) (Simple LR)

- **Power:** Least powerful among the four. Uses FOLLOW sets for reduce decisions, which may lead to conflicts in some grammars that LR(1) or LALR(1) could handle.
- **Simplicity:** Easiest to implement. Smallest parsing tables. Good for educational purposes and simple languages.

5

(a) Grammar

First Sets:

$$\begin{aligned}\text{First}(A) &= \{a, \varepsilon\}\\ \text{First}(B) &= \{b\}\\ \text{First}(D) &= \{a, \varepsilon\}\\ \text{First}(S) &= \{a, b\}\\ \text{First}(S') &= \{a, b\}\end{aligned}$$

Follow Sets:

$$\begin{aligned}\text{Follow}(S) &= \{\$\}\\ \text{Follow}(A) &= \{b\}\\ \text{Follow}(B) &= \{d\}\\ \text{Follow}(D) &= \{\$\}\end{aligned}$$

Canonical LR(1) Items: Initial State I_0

$$\begin{array}{ll} S' \to \cdot S, & \$\\ S \to \cdot ABdD, & \$\\ S \to \cdot bD, & \$\\ A \to \cdot aA, & b\\ A \to \cdot \varepsilon, & b\\ B \to \cdot b, & d \end{array}$$

Conflict in LALR(1) Table:
After merging LR(1) items with the same LR(0) core, we obtain:

- `A → ε`, lookahead: {b, d}
- `B → b`, lookahead: {d}

On lookahead `b`, the parser cannot decide whether to:

- Reduce using `A → ε`, or
- Shift `b` to match `B → b`

This results in a **shift/reduce conflict** in the LALR(1) parsing table.
Conclusion: The grammar is **not LALR(1)**.

(b) Grammar

First Sets:

- FIRST$(S) = \{a, b\}$
- FIRST$(S) = \{a, b\}$
- FIRST$(A) = \{a, b\}$

Follow Sets:

- FOLLOW$(S') = \{\$\}$

- FOLLOW(S) = {\$,)}
- FOLLOW(A) = {\$,)}

state table

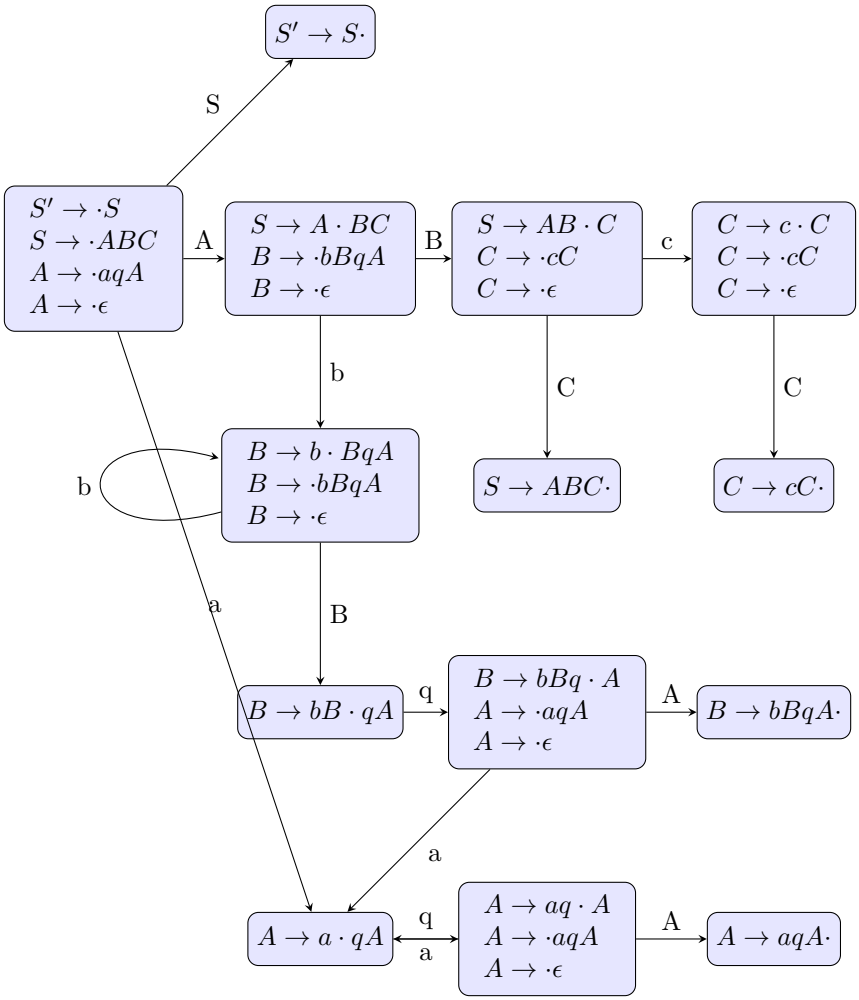
State	Kernel	GOTO	Closure
0	$S' \to S, \$$		$S' \to S, \$$ $S \to (A, \$), \$$ $A \to AS, \$$ $A \to .b, \$$
1	$S' \to S., \$$	getc(G, S)	$S' \to S., \$$
2	$S \to (A, S), \$,$	getc(G, C)	$S \to (A, \$), \$$ $A \to .aS, \$$ $A \to .b, \$$
3	$S \to A., \$,$	getc(G, A)	$S \to A., \$,$
4	$A \to a.S, \$,$	getc(G, a)	$A \to .aS, \$,$ $S \to (A, \$), \$,$ $A \to .aS, \$,$ $A \to .b, \$,$
5	$A \to b., \$,$	getc(G, b)	$A \to b., \$,$
6	$S \to (A., S), \$,$	getc$(2, A)$	$S \to (A., \$), \$,$
4	$A \to a.S, \$,$	gotc$(2, a)$	
5	$A \to b.\$,$	getc$(2, b)$	
7	$S \to aS., \$,$	getc(G, S)	$A \to aS., \$,$
4	$\{A \to a.A, b\}$	getc(G, a)	
2	$S \to (A, S), \$,$	getc(G, C)	
3	$S \to A., \$,$	getc(G, A)	
4		getc(G, a)	
5		getc(G, b)	
8	$S = (A, .S), \$$	getc$(6, .)$	$A = .aS, 0, S = (A, .S), \$$ $S = (A, .S), 0, S = .A, 0, A = .aS, 0, A = .b, 0$
9	$S = (A, .S), \$$	getc$(8, .S)$	$S = (A, .S), \$$
2	$S = (A, .S), \$$	getc$(8, .C)$	
3	$S = A, .\$$	getc$(8, .A)$	
4	$S = a.S, \$$	getc$(8, .a)$	
5	$S = b., \$$	getc$(8, .b)$	
10	$S = (A, .S), \$$	getc$(9, .)$	$S = (A, .S), \$$

Parsing table

State	(	)	a	b	$	S	A
0	s2		s4	s5	-1	1	3
1					accept		
2			s4	s5	-1	6	
3	r2	r2		-1	r2		
4	s2		s4	s5	-7	7	3
5	r4	r4		-1	r4		
6	s8						
7	r3	r3		-1	r3		
8	s2		s4	s5	-9	9	3
9		s10					
10	r1	r1		-1	r1		

6

1. $S \to ABC$
2. $A \to aqA$
3. $A \to \varepsilon$
4. $B \to bBqA$
5. $B \to \varepsilon$
6. $C \to cC$
7. $C \to \varepsilon$



State	a	b	c	q	$	S	A	B	C
0	s3	r3	r3	r3	r3	1	2		
1					accept				
2		s5	r5	r5	r5			4	
3				s6					
4			s8		r7				7
5		s5	r5	r5	r5			9	
6	s3	r3	r3	r3	r3		10		
7					r1				
8			s8		r7				11
9				s12					
10		r2	r2	r2	r2				
11					r6				
12	s3	r3	r3	r3	r3		13		
13		r4	r4	r4					

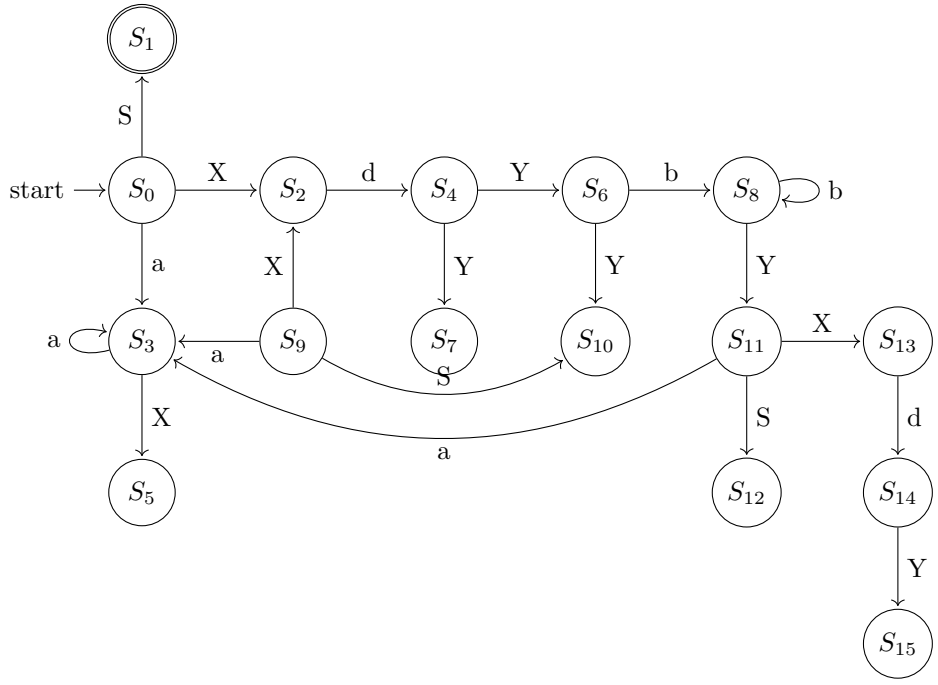
Step	Stack	Input	Action
1	0	aqbqcc$	s3
2	0a3	qbqcc$	s6
3	0a3q6	bqcc$	r3
4	0a3q6A10	bqcc$	r2
5	0A2	bqcc$	s5
6	0A2b5	qcc$	r5
7	0A2b5B9	qcc$	s12
8	0A2b5B9q12	cc$	r3
9	0A2b5B9q12A13	cc$	s8
10	0A2B4	cc$	s8
11	0A2B4c8	c$	s8
12	0A2B4c8c8	$	r7
13	0A2B4c8c8C11	$	r6
14	0A2B4c8C11	$	r6
15	0A2B4C7	$	r1
16	0S1	$	Accept

7

- **S0**
  $[S' \to \cdot S, S]$
  $[S \to \cdot X dY, S]$
  $[X \to \cdot aX, d]$
  $[X \to \cdot \epsilon, d]$
- **S1**
  $[S' \to S\cdot, S]$
- **S2**
  $[S \to X \cdot dY, S]$
- **S3**
  $[X \to a \cdot X, d]$

$[X \to \cdot aX, d]$
$[X \to \cdot \epsilon, d]$

- **S5**
  $[X \to aX \cdot, d]$
- **S9**
  $[Y \to bY \cdot S, S]$
  $[S \to \cdot XdY, S]$
  $[X \to \cdot aX, d]$
  $[X \to \cdot \epsilon, d]$
- **S10**
  $[Y \to bYS \cdot, S]$
- **S12**
  $[Y \to bYS \cdot, a]$
- **S13**
  $[S \to X \cdot dY, a/d]$
- **S14**
  $[Y \to \cdot bYS, a/d]$
  $[Y \to \cdot \epsilon, a/d]$
- **S15**
  $[S \to XdY \cdot, a/d]$



state	ACTION				GOTO		
	a	b	d	$	S	X	Y
0	s3		r3		1	2	
1				accept			
2			s4				
3	s3		r3			5	
4		s6		r5			7
5			r2				
6	r5	s8		r5			9
7				r1			
8	r5	s8		r5			11
9	s3		r3		10	2	
10				r4			
11	s3		r3		12	13	
12	r4		r4				
13			s14				
14	r5	s8	r5				15
15	r1		r1				

Table 1: LR Parsing Steps for Input String "aadbbadd$"

Step	**Stack**	**Input**	**Action**
1	0	aadbbadd$	s3
2	0a3	adbbadd$	s3
3	0a3a3	dbbadd$	r3
4	0a3a3	dbbadd$	Goto X, state 5
5	0a3a3X5	dbbadd$	r2
6	0a3	dbbadd$	Goto X, state 5
7	0a3X5	dbbadd$	r2
8	0	dbbadd$	Goto X, state 2
9	0X2	dbbadd$	s4
10	0X2d4	bbadd$	s6
11	0X2d4b6	badd$	s8
12	0X2d4b6b8	add$	r5
13	0X2d4b6b8	add$	Goto Y, state 11
14	0X2d4b6b8Y11	add$	s3
15	0X2d4b6b8Y11a3	dd$	r3
16	0X2d4b6b8Y11a3	dd$	Goto X, state 13
17	0X2d4b6b8Y11X13	dd$	s14
18	0X2d4b6b8Y11X13d14	d$	r5
19	0X2d4b6b8Y11X13d14	d$	Goto Y, state 15
20	0X2d4b6b8Y11X13d14Y15	d$	r1
21	0X2d4b6b8Y11	d$	s12
22	0X2d4b6b8Y11S12	d$	r4
23	0X2d4b6	d$	s9
24	0X2d4b6Y9	d$	r4
25	0X2d4	d$	s7
26	0X2d4Y7	d$	r1
27	0	d$	Goto S, state 1
28	0S1	$	Accept

8

- **I0**
  $[S' \to \cdot S, \$]$
  $[S \to \cdot Aa, \$]$
  $[S \to \cdot bAc, \$]$
  $[S \to \cdot Bc, \$]$
  $[S \to \cdot bBa, \$]$
  $[A \to \cdot d, a]$
  $[B \to \cdot d, c]$

- **I1**
  $[S' \to S\cdot, \$]$

- **I2**
  $[S \to A \cdot a, \$]$

- **I3**
  $[S \to b \cdot Ac, \$]$
  $[S \to b \cdot Ba, \$]$
  $[A \to \cdot d, c]$
  $[B \to \cdot d, a]$

- **I4**
  $[S \to B \cdot c, \$]$

- **I5**
  $[A \to d\cdot, a]$
  $[B \to d\cdot, c]$

- **I6**
  $[S \to Aa\cdot, \$]$

- **I7**
  $[S \to bA \cdot c, \$]$

- **I8**
  $[S \to bB \cdot a, \$]$

- **I9**
  $[A \to d\cdot, c]$
  $[B \to d\cdot, a]$

- **I10**
  $[S \to Bc\cdot, \$]$

- **I11**
  $[S \to bAc\cdot, \$]$

- **I12**
  $[S \to bBa\cdot, \$]$

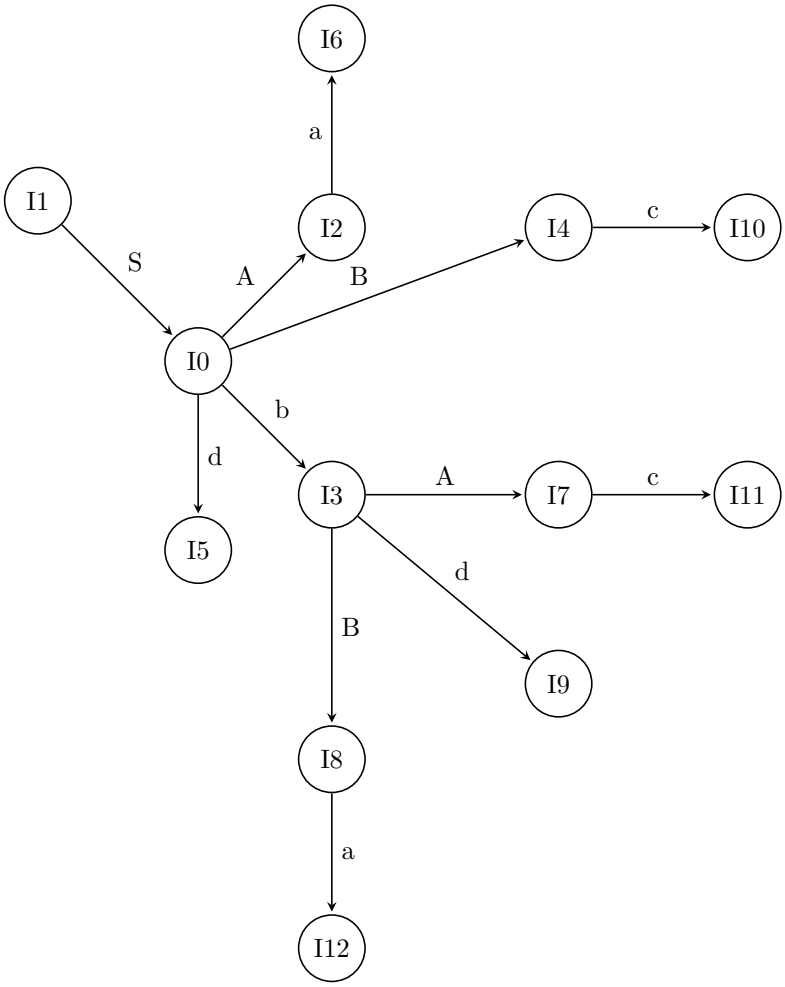


Table 2: LR(1) Parsing Table

State	ACTION					GOTO		
	a	b	c	d	$	S	A	B
0		s3		s5		1	2	4
1					accept			
2	s6							
3				s9			7	8
4			s10					
5	r6		r7					
6					r2			
7			s11					
8	s12							
9	r7		r6					
10					r4			
11					r3			
12					r5			

There is no conflict, and as a result, there is no problem for being an LR(1) parser.

Now, to examine conflicts in LALR(1), we need to follow the core shared items. Here, states 5 and 9 share the same core. If we merge them:

Table 3: LALR(1) Parsing Table

State	ACTION					GOTO		
	a	b	c	d	$	S	A	B
0		s3		s5		1	2	4
1					accept			
2	s6							
3				s9			7	8
4			s10					
5,9	r6/r7		r6/r7					
6					r2			
7			s11					
8	s12							
10					r4			
11					r3			
12					r5			

So as we have conflicts here, this is not grammar is not LALR(1).