LR Parsing for If-Else Grammar

Building Action and Goto Tables

Compiler Design

December 1, 2025

Step 1: Define the Grammar

Original Grammar:

$$
\begin{aligned}
S &\rightarrow \text{if } (E)\, S \\
S &\rightarrow \text{if } (E)\, S \text{ else } S \\
S &\rightarrow \text{id} = \text{num} \\
E &\rightarrow \text{id} == \text{num}
\end{aligned}
$$

Augmented Grammar:

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow \text{if } (E)\, S \quad \text{(Production 1)} \\
S &\rightarrow \text{if } (E)\, S \text{ else } S \quad \text{(Production 2)} \\
S &\rightarrow \text{id} = \text{num} \quad \text{(Production 3)} \\
E &\rightarrow \text{id} == \text{num} \quad \text{(Production 4)}
\end{aligned}
$$

Step 2: LR(0) Items

What is an LR(0) Item?

- A production with a dot ($\bullet$) indicating parsing position
- Example: $S \to \text{if } \bullet (E)\, S$
- The dot shows how much we've seen and what we expect next

Item Types:

- **Initial item:** $A \to \bullet\alpha$ (nothing parsed yet)
- **Intermediate item:** $A \to \alpha \bullet \beta$ (partially parsed)
- **Complete item:** $A \to \alpha\bullet$ (ready to reduce)

Step 3: Construct LR(0) States - State 0

State 0 (Start State):

$$\begin{aligned} S' &\rightarrow \bullet S \\ S &\rightarrow \bullet \text{if } (E)\, S \\ S &\rightarrow \bullet \text{if } (E)\, S \text{ else } S \\ S &\rightarrow \bullet \text{id} = \text{num} \end{aligned}$$

Transitions:

- On `if`: go to State 2
- On `id`: go to State 3
- On S: go to State 1

Step 3 (continued): States 1, 2, 3

State 1 (Accept State):

$$S' \to S\bullet$$

State 2:

$$S \to \text{if } \bullet (E)\, S$$
$$S \to \text{if } \bullet (E)\, S \text{ else } S$$

On (: go to State 4

State 3:

$$S \to \text{id}\bullet = \text{num}$$

On =: go to State 5

Step 3 (continued): States 4-7

State 4:

$$\begin{aligned} S &\rightarrow \text{if } (\bullet E)\, S \\ S &\rightarrow \text{if } (\bullet E)\, S \text{ else } S \\ E &\rightarrow \bullet \text{id} == \text{num} \end{aligned}$$

On `id`: State 6, On E: State 7

State 5: $S \rightarrow \text{id} = \bullet \text{num}$ On `num`: State 8

State 6: $E \rightarrow \text{id}\bullet == \text{num}$ On **==**: State 9

State 7:

$$\begin{aligned} S &\rightarrow \text{if } (E\bullet)\, S \\ S &\rightarrow \text{if } (E\bullet)\, S \text{ else } S \end{aligned}$$

On): State 10

Step 3 (continued): States 8-11

State 8:

$$S \to \text{id} = \text{num}\bullet$$

Reduce by production 3

State 9: $E \to \text{id} == \bullet\text{num}$ On `num`: State 11

State 10:

$$\begin{aligned} S &\to \text{if } (E) \bullet S \\ S &\to \bullet\text{if } (E)\, S \\ S &\to \bullet\text{if } (E)\, S \text{ else } S \\ S &\to \bullet\text{id} = \text{num} \end{aligned}$$

(Closure adds all S productions since dot precedes S)
On `if`: State 2, On `id`: State 3, On S: State 12

Step 3 (continued): State 11

State 11:

$$E \rightarrow \text{id} == \text{num}\bullet$$

Reduce by production 4

Step 3 (continued): State 12 - Shift/Reduce Conflict

State 12:

$$S \to \text{if } (E)\, S\bullet$$
$$S \to \text{if } (E)\, S \bullet \text{ else } S$$

CONFLICT on lookahead `else`:

- **REDUCE** by $S \to \text{if } (E)\, S$ (production 1)
- **SHIFT** the `else` token to State 13

This is the famous **“Dangling Else”** problem!

On `else`: go to State 13

Step 3 (continued): States 13-14

State 13:

$$\begin{aligned} S &\rightarrow \text{if } (E)\, S \text{ else} \bullet S \\ S &\rightarrow \bullet\text{if } (E)\, S \\ S &\rightarrow \bullet\text{if } (E)\, S \text{ else } S \\ S &\rightarrow \bullet\text{id} = \text{num} \end{aligned}$$

On `if`: State 2, On `id`: State 3, On S: State 14

State 14:

$$S \rightarrow \text{if } (E)\, S \text{ else } S\bullet$$

Reduce by production 2

Step 4: Build Action Table (Part 1)

State	if	(	)	else	id
0	s2				s3
1					
2		s4			
3					
4					s6
5					
6					
7			s10		
8			r3	r3	
9					
10	s2				s3
11			r4	r4	
12			r1	s13	
13	s2				s3
14			r2	r2	

Step 4: Build Action Table (Part 2)

State	`==`	`=`	`num`	`$`
0				
1				acc
2				
3		s5		
4				
5			s8	
6	s9			
7				
8				r3
9			s11	
10				
11				
12				r1
13				
14				r2

State 12, `else`: Conflict resolved by choosing **SHIFT**

Step 5: Build Goto Table

State	S	E
0	1	
1		
2		
3		
4		7
5		
6		
7		
8		
9		
10	12	
11		
12		
13	14	
14		

Step 6: Analyze the Conflict

The Dangling Else Problem:

- In State 12, when we see `else`, we have two choices:
  1. **REDUCE:** Complete the inner if-statement
  2. **SHIFT:** Continue parsing to match else with current if

Example:

```
if (x == 1)
    if (y == 2)
        id = 3
    else
        id = 4
```

Question: Which `if` does the `else` belong to?

Step 7: Resolve the Conflict

Resolution Strategy:

Choose SHIFT over REDUCE

When lookahead is `else` in State 12, we **SHIFT** instead of reducing.

Rationale:

- This matches `else` with the *nearest* unmatched `if`
- Standard behavior in C, Java, Python, etc.
- More intuitive for programmers

Result:

```
if (x == 1)            // outer if
    if (y == 2)        // inner if
        id = 3
    else               // matches inner if
        id = 4
```

Step 8: Example Parse

Input: `if (id == num) id = num else id = num`

Stack	Input	Action
0	if (id == num) id = num else ...	s2
0 if 2	(id == num) id = num else ...	s4
0 if 2 (4	id == num) id = num else ...	s6
0 if 2 (4 id 6	== num) id = num else ...	s9
...	...	...
0 if 2 (4 E 7	) id = num else ...	s10
0 if 2 (4 E 7) 10	id = num else ...	s3
...	...	...
0 if 2 (4 E 7) 10 S 12	else id = num **$**	s13
...	...	...
0 S 1	**$**	acc

Summary

Key Takeaways:

1. We built an LR parser for if-else statements
2. Created 15 states (0-14) with LR(0) items
3. Identified shift/reduce conflict in State 12
4. Resolved by preferring SHIFT over REDUCE for `else`
5. This implements the "else matches nearest if" rule

Shift/Reduce Conflict Resolution:

- **Location:** State 12, lookahead `else`
- **Decision:** SHIFT (go to State 13)
- **Effect:** Associates `else` with closest `if`