

# **How to Teach LR Parsing**

by

*Manuel E. Bermudez*

*George Logothetis*

University of Florida  
Gainesville, FL 32611

## LR Parsing in current textbooks:

- Not intuitive
- Variety of terminologies
- Cluttered with unnecessary notation
- Don't deal with efficient implementation (non-trivial)

We propose a three-part treatise in which:

- Explain how to build the LR(0) automaton, and do it right !
- Explain how to compute SLR(1) lookahead.
- Explain how to compute LALR(1) lookahead.

Our treatise (as we shall see) is **simple, concise and easy to teach**.

We assume knowledge of:

- Context-free grammars.
- Deterministic and non-deterministic FSA's.
- Parsing in general (e.g. recursive descent)

Notation:

$A, B, C, \dots$	nonterminal symbols
$t, a, b, c, \dots$	terminal symbols
$\dots, x, y, z$	terminal strings
$\dots, X, Y, Z$	grammar symbols
$\alpha, \beta, \gamma, \dots, \omega$	strings of grammar symbols
$\varepsilon$	the empty string
$A \rightarrow \omega$	a production in a CFG
$\Rightarrow$	right-most derivation
$\text{First}(\alpha)$	$\{t \mid \alpha \Rightarrow^* tx, \text{ for some } x\}.$
$p, q, r, s$	states in a FSA

## Generation of LR(0) Automata.

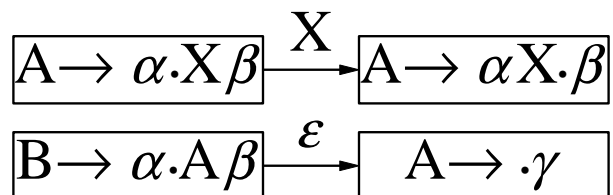
- One procedure per nonterminal
- Unknown conditions as required.

Example:  $S \rightarrow A \perp, A \rightarrow aAb, A \rightarrow ab$

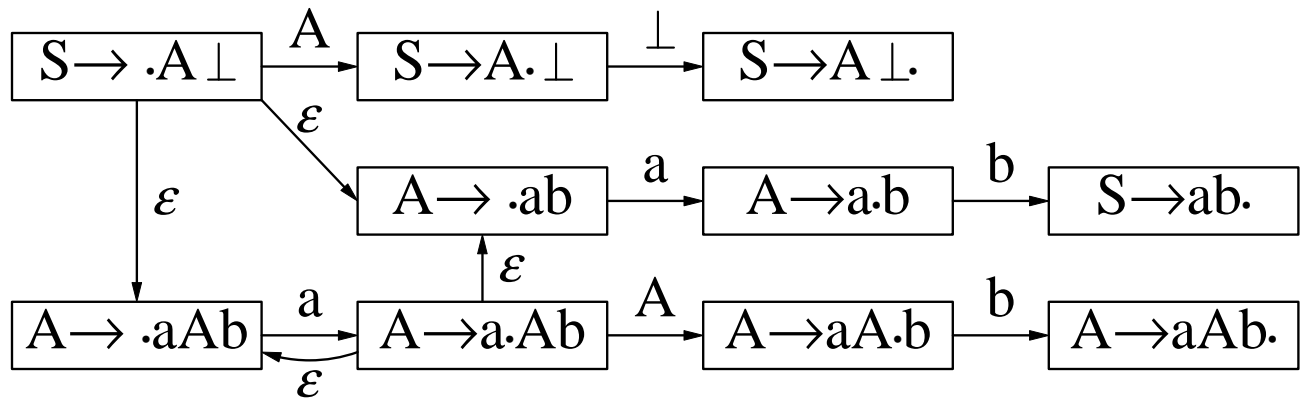
```
procedure S;  {  $S \rightarrow \cdot A \perp$  }  
    call A;  {  $S \rightarrow A \cdot \perp$  }  
    Read( $\perp$ ); {  $S \rightarrow A \perp \cdot$  }  
end;
```

```
procedure A;  {  $A \rightarrow \cdot ab, A \rightarrow \cdot aAb$  }  
    Read(a);  {  $A \rightarrow a \cdot b, A \rightarrow a \cdot Ab$  }  
    if ??? then Read(b) {  $A \rightarrow ab \cdot$  }  
    else call A; {  $A \rightarrow aA \cdot b$  }  
    Read(b) {  $A \rightarrow aAb \cdot$  }  
end;
```

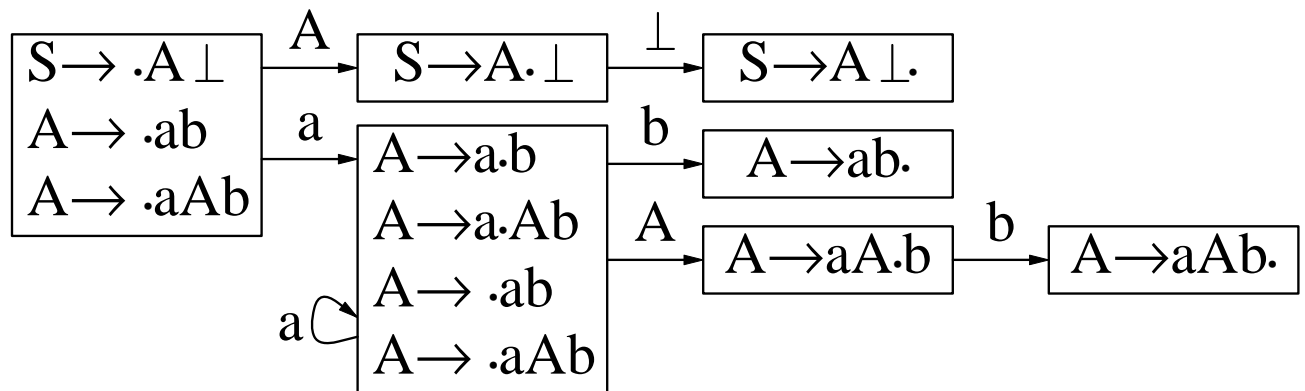
- Add items (marked productions).
- Transitions among items:



- The complete non-deterministic FSA:



- Transform it to a deterministic FSA:



- Driver algorithm reflects actions of code:

```

Algorithm LR_Driver;
begin
  Push Start_State on Stack;
  q := Start_State;
  while ACTION_OF(q) ≠ Accept do
    case ACTION_OF(q) of
      S/t: q := Go(q,t);
           { Move past Read statement }
           Push q on Stack
      R/A → ω: Pop | ω | states from Stack;
              { Return to point of call }
              NewTop := Top(Stack);
              q := Go(NewTop,A);
              { Move past call statement }
              Push q on Stack;
      Error:  Diagnose_Error; Stop;
      Accept: ;
    end
  end;
end;

```

- Parser is encoded into a table:

	$t_1$	$t_2$	$\dots$	$t_n$
$p_1$	S/q			
$\dots$				
$p_m$			R/A $\rightarrow \omega$	

- $\text{ACTION}(p,t)=S/q$  if  $\text{Go}(p,t)=q$
- $\text{ACTION}(p,t)=R/A \rightarrow \omega$  if  $t \in \text{Follow}(A)$
- $\text{Follow}(A) = \{t \mid S \Rightarrow^* \alpha A t x\}$ .

$\text{Follow}(A) = \text{IFollow}(A) \cup \text{DFollow}(A),$   
 $\text{IFollow}(A) = \cup \{ \text{Follow}(B) \mid B \rightarrow \alpha A \gamma, \gamma \Rightarrow^* \varepsilon \},$   
 $\text{DFollow}(A) = \cup \{ \text{First}(X) \mid B \rightarrow \alpha A \gamma X \delta, \gamma \Rightarrow^* \varepsilon \},$   
 $\text{First}(A) = \cup \{ \text{First}(X) \mid A \rightarrow \gamma X \delta, \gamma \Rightarrow^* \varepsilon \},$   
 $\text{First}(t) = \{t\}.$

### Definitions:

- $X \text{ ff } A$  if  $A \rightarrow \gamma X \delta$  and  $\gamma \Rightarrow^* \varepsilon$ .
- $X \text{ fF } A$  if  $B \rightarrow \alpha A \gamma X \delta$  and  $\gamma \Rightarrow^* \varepsilon$ .
- $B \text{ FF } A$  if  $B \rightarrow \alpha A \gamma$  and  $\gamma \Rightarrow^* \varepsilon$ .

Thus,  $t \in \text{Follow}(A)$  iff  $t (\text{ff}^* \circ \text{fF} \circ \text{FF}^*) A$ .

**Algorithm** Compute\_SLR\_Action\_Table:

**Input:** LR(0) automaton, ff, fF, FF;

**Output:** ACTION table;

**var** ff\_Visited: a bit vector indexed by symbols;

FF\_Visited: a bit vector indexed by nonterminals;

**procedure** Follow\_to\_Follow(A):

**begin**

**if** FF\_Visited[A] is set **then return;**

**set** FF\_Visited[A];

**for each** (q,  $A \rightarrow \omega$ ) **do**

    Add “Reduce/ $A \rightarrow \omega$ ” to ACTION[q,t];

**for each** B **such that** A fF B **do**

    Follow\_to\_Follow(B);

**end;**

**procedure** First\_to\_First(X):

**begin**

**if** ff\_Visited[X] is set **then return;**

**set** ff\_Visited[X];

**for each** A **such that** X fF A **do**

    Follow\_to\_Follow(A);

**for each** Y **such that** X ff Y **do**

    First\_to\_First(Y);

**end;**

**begin**

**for each** terminal t **do**

**begin**

**clear** ff\_Visited[X], **for each** symbol X;

**clear** FF\_Visited[A], **for each** nonterminal A;

    First\_to\_First(t);

**end;**

**end;**



## Conclusions.

- Our LR(0) construction is easier to understand: no Nucleus, Closure and Successors operations. Instead, we focus on the non-deterministic version of the LR(0) automaton.
- Treatise on Follow sets is intuitive, straightforward and **computational**.
- Similar principles apply to LALR(1); the Follow sets are slightly different.
- LALR(1) is “not harder” than SLR(1).
- LR parsing ought to be taught this way.