

# **Top-down parsing and introduction to bottom-up parsing**

# Grammar transformations

## 1. Elimination of left recursion

Immediate self recursion can be eliminated by replacing the set of all productions for a nonterminal A:

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

into the set of productions:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

Then, arrange the nonterminal in some order  $A_1, A_2, \dots, A_n$  and apply the loop **for i=1 to n**

**for j=1 to i-1**

- Let  $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$  be all  $A_j$  productions

- Replace each  $A_i \rightarrow A_j \gamma$  with  $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$

Eliminate the immediate left recursion among the  $A_i$  productions

After iteration  $i-1$  of outer loop, any  $A_k \rightarrow A_p \alpha$ , for  $k < i$  must have  $p > k$ .

After iteration  $j$  of inner loop, any  $A_i \rightarrow A_q \alpha$  must be such that  $q > j$ . So, at the end of the inner loop, any  $A_i \rightarrow A_q \alpha$  must be such that  $q \geq i$

## 2. Left factoring a grammar

repeatedly apply the following until no more changes:

- For each nonterminal  $A$  find longest prefix  $\alpha \neq \varepsilon$  common to two or more productions.
- Replace  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$  with  $A \rightarrow \alpha A' \mid \gamma$  and  $A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

## Nonrecursive predictive parsing

Recursive descent parsers can be simulated using a stack to avoid the cost of recursion.

Let the input be  $w\$$  where  $\$$  is an EOF marker.

The algorithm proceeds as follows:

```
Stack = $ S
repeat
  if Stack.top is a terminal or $ then
    if Stack.top == lookahead then
      Stack.pop(); match(lookahead);
    else error();
  else // Stack.top is a nonterminal
    if  $M[\textit{Stack.top}, \textit{lookahead}] == \textit{Stack.top} \rightarrow Y_1 Y_2 \dots Y_k$  then
      Stack.pop()
      Stack.push( $Y_k Y_{k-1} \dots Y_1$ )
      print the production  $\textit{Stack.top} \rightarrow Y_1 Y_2 \dots Y_k$ 
    else error();
until Stack.top == $
```

# FIRST and FOLLOW

## FIRST:

```
for each terminal  $X$   
     $FIRST(X) = \{X\}$ ;  
for each production  $X \rightarrow \varepsilon$   
     $FIRST(X) = \{\varepsilon\}$   
repeat  
    for each production  $X \rightarrow Y_1 Y_2 \dots Y_k$   
        for  $i = 1$  to  $k$   
             $FIRST(X) \cup = FIRST(Y_i) - \{\varepsilon\}$   
            if  $\varepsilon$  is not in  $FIRST(Y_i)$  then break;  
            elseif  $i == k$  then  $FIRST(X) \cup = \{\varepsilon\}$   
until no more terminals or  $\varepsilon$  can be added to any FIRST set.
```

```
for any string  $X_1 X_2 \dots X_n$   
    for  $i = 1$  to  $n$   
         $FIRST(X_1 X_2 \dots X_n) = FIRST(X_i) - \{\varepsilon\}$ ,  
        if  $\varepsilon$  is not in  $FIRST(X_i)$  then break;  
        elseif  $i == k$  then  $FIRST(X_1 X_2 \dots X_n) \cup = \{\varepsilon\}$ 
```

## **FOLLOW:**

**FOLLOW(S)={\\$};**

**for** each nonterminal B

**for** each production occurrence of  
    B in a production  $A \rightarrow \alpha B \beta$

**FOLLOW(B) U=FIRST( $\beta$ )- $\{\epsilon\}$**

**repeat**

**for** each nonterminal B

**for** each production occurrence of  
        B in a production  $A \rightarrow \alpha B$  or production  
         $A \rightarrow \alpha B \beta$  where  $\epsilon$  is in **FIRST( $\beta$ )**

**FOLLOW(B) U=FOLLOW(A)**

**until** nothing can be added to a FOLLOW set.

## Construction of predictive parsing tables

1. For each production  $A \rightarrow \alpha$  do 2 and 3
2. For each terminal  $a$  in  $\text{FIRST}(\alpha)$  add  $A \rightarrow \alpha$  to  $M[A,a]$
3. If  $\epsilon$  is in  $\text{FIRST}(\alpha)$  add  $A \rightarrow \alpha$  to  $M[A,b]$  for each  $b$  in  $\text{FOLLOW}(A)$ .

## LL(1) grammar

A grammar whose parsing table has no multiply-defined entries is said to be LL(1).

The first L stands from scanning the input from left to right and the second L for producing a leftmost derivation, and the 1 for using one input symbol of lookahead at each step to make parsing decisions.

LL(1) grammars are particularly attractive for practical use. For each LL(1) grammar we can find a parser which is small, fast, and produces a left parse, which is advantageous for translation purposes.

However, an LL(1) grammar for a given language can be unnatural and difficult to construct.

Moreover, not every deterministic context free language has an LL grammar, let alone an LL(1) grammar.

Note: For  $K \geq 0$  the LL(k) languages are a proper subset of the LL(k+1) languages.

Also, LL languages are a proper subset of LR languages.