Lecture 19 - Nov. 22

Syntactic Analysis

Left Factoring
TDP: Terminating & Min. Backtracking
LL(1) vs. LR(1) Parser
Bottom-Up Parsing, RMDs

Announcements

- **Project Milestone 2** due tonight
- **Assignment 3** due soon
- **Project Report Template**

Backtrack-Free Grammar: Exercise

A ***backtrack-free grammar*** has each of its productions $A \to \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$ satisfying:

$$\forall i, j : 1 \le i, j \le n \wedge i \neq j \bullet \textsc{Start}(\gamma_i) \cap \textsc{Start}(\gamma_j) = \varnothing$$

no ambiguity in choosing a production rule using a lookahead symbol

$$\textsc{Start}(A \to \beta) = \begin{cases} \textsc{First}(\beta) & \text{if } \epsilon \notin \textsc{First}(\beta) \\ \textsc{First}(\beta) \cup \textsc{Follow}(A) & \text{otherwise} \end{cases}$$

$\textsc{First}(\beta)$ is the extended version where β may be $\beta_1 \beta_2 \ldots \beta_n$

Is the following CFG ***backtrack free***?

word: name .

11	*Factor*	→	name
12		|	name [*ArgList*]
13		|	name (*ArgList*)
15	*ArgList*	→	*Expr MoreArgs*
16	*MoreArgs*	→	, *Expr MoreArgs*
17		|	ϵ

No common prefix

START (Factor → name) = ? {name}

FIRST(name) = {name}

START (Factor → name [ArgList]) = {name}

Left-Factoring: Removing Common Prefixes

Identify a common prefix α:

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_j$$

[each of $\gamma_1, \gamma_2, \ldots, \gamma_j$ does not begin with α]

Rewrite that production rule as:

$$\begin{aligned} A &\rightarrow \alpha B \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_j \\ B &\rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \end{aligned}$$

11	*Factor*	→	name (ϵ)
12		|	name [*ArgList*]
13		|	name (*ArgList*)
15	*ArgList*	→	*Expr MoreArgs*
16	*MoreArgs*	→	, *Expr MoreArgs*
17		|	ϵ

Factor → name Arguments

Arguments → ϵ | [Arglist] | (Arglist]

satisfy the backtrack-free property.

Implementing a Recursive-Descent Parser

	Production		FIRST+
2	Expr′ →	+ Term Expr′	{+}
3	|	- Term Expr′	{-}
4	|	ϵ	{ϵ, eof,)}

START

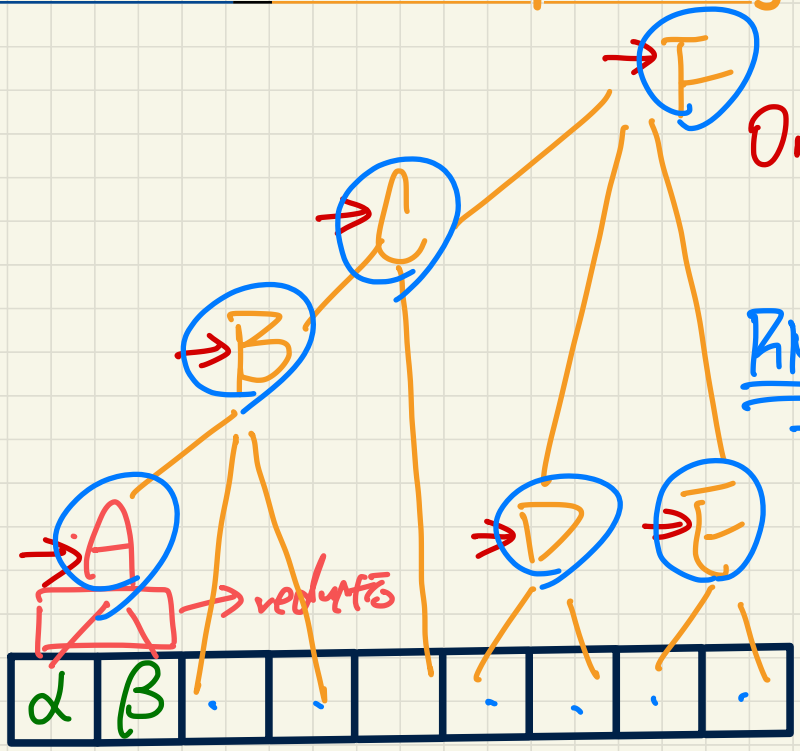
```
ExprPrim()
  if word = + ∨ word = - then /* Rules 2, 3 */
    word := NextWord()
    if Term()
      then return ExprPrim()
      else return false
  elseif word = ) ∨ word = eof then /* Rule 4 */
    return true
  else
    report a syntax error
    return false
  end

Term()
  ...
```

Discovering Derivations: Bottom-Up Parsing

production rule

$A \rightarrow \alpha\beta$

scanner



Order of reductions:

A B C D E F.

RMD:

F E D C B A