Syntax Analysis: Top-Down Parsing

Lecture 4

Objectives

By the end of this lecture you should be able to:

1. Identify top-down parsing.
2. Compute the functions *First* and *Follow* for a given grammar.
3. Identify $LL(1)$ grammars.
4. Construct the parsing table for a grammar.
5. Construct a predictive top-down parser using pushdown automata.

Outline

1. Top-Down Parsing
2. *First* and *Follow*
3. Predictive Top-Down Parsing
4. Predictive Parsing with Pushdown Automata

Outline

1. Top-Down Parsing
2. *First* and *Follow*
3. Predictive Top-Down Parsing
4. Predictive Parsing with Pushdown Automata

What is Top-Down Parsing?

Definition

Top-down parsing consists in a preorder construction of a parse tree for a given input string and CFG.

Equivalently:

Definition

Top-down parsing consists in finding a left-most derivation of a given string in a given CFG.

What is Top-Down Parsing?

Definition

Top-down parsing consists in a preorder construction of a parse tree for a given input string and CFG.

Equivalently:

Definition

Top-down parsing consists in finding a left-most derivation of a given string in a given CFG.

Example: G_3

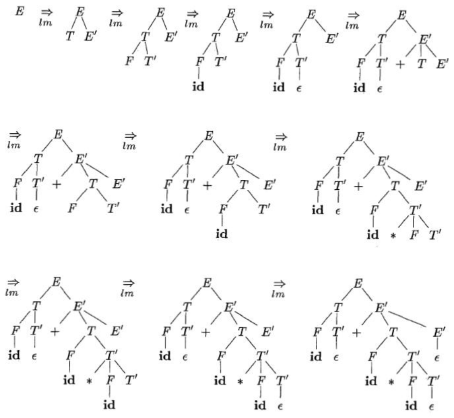
Example

$$
\begin{array}{lcl}
E & \longrightarrow & T\ E' \\
E' & \longrightarrow & +\ T\ E' \mid \varepsilon \\
T & \longrightarrow & F\ T' \\
T' & \longrightarrow & *\ F\ T' \mid \varepsilon \\
F & \longrightarrow & (\ E\ ) \mid \mathbf{id} \mid \mathbf{number}
\end{array}
$$

Input: $\mathbf{id} + \mathbf{id} * \mathbf{id}$.

Example: Top-Down Parsing

Example



© Aho et al. (2007)

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree`(S)
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children`(A, β)
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S\rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree`(S)
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children`(A, β)
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree`(S)
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children`(A, β)
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S\rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree`(S)
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children`(A, β)
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S\rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S \rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S\rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(`S`)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
    - $\delta = \varepsilon$ or
    - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(`A, β`)`
11. Goto 2.

General Structure of a Top-Down Parser

Given $G = \langle V, \Sigma, R, S\rangle$ and $w = w_1 \cdots w_n$.

1. $SF = S$. `tree t = new tree(S)`
2. For $SF = u\,\delta$, where $u \in \Sigma^*$ and
   - $\delta = \varepsilon$ or
   - $\delta = A\alpha$, with $A \in V$ and $\alpha \in (\Sigma \cup V)^*$
3. if $|w| < |u|$, then fail.
4. if $w_1 \cdots w_{|u|} \neq u$ then fail.
5. $SF = \delta$.
6. $w = w_{|u|+1} \cdots w_n$.
7. If $\delta = w = \varepsilon$, then return. `return(t)`
8. If $\delta = \varepsilon$, then fail.
9. Choose $(A \longrightarrow \beta) \in R$.
10. $SF = \beta\,\alpha$. `t.children(A, β)`
11. Goto 2.

Remarks

- Top-down parsing may be implemented using recursive transition networks (giving rise to recursive-descent parsing).
- As is, the algorithm is nondeterministic.
  - Backtracking may be needed.
- A left-recursive grammar may cause an infinite loop.

Remarks

- Top-down parsing may be implemented using recursive transition networks (giving rise to recursive-descent parsing).
- As is, the algorithm is nondeterministic.
  - Backtracking may be needed.
- A left-recursive grammar may cause an infinite loop.

Remarks

- Top-down parsing may be implemented using recursive transition networks (giving rise to recursive-descent parsing).
- As is, the algorithm is nondeterministic.
  - Backtracking may be needed.
- A left-recursive grammar may cause an infinite loop.

Remarks

- Top-down parsing may be implemented using recursive transition networks (giving rise to recursive-descent parsing).
- As is, the algorithm is nondeterministic.
  - Backtracking may be needed.
- A left-recursive grammar may cause an infinite loop.

Outline

First

Let $G = \langle V, \Sigma, R, S \rangle$ be a CFG.

Definition

For every sentential form α of G,

$$First(\alpha) = \{a \in \Sigma \mid \alpha \overset{*}{\Rightarrow} a\beta\} \cup \Upsilon$$

where

$$\Upsilon = \begin{cases} \{\varepsilon\} & \text{if } \alpha \overset{*}{\Rightarrow} \varepsilon \\ \varnothing & \text{otherwise} \end{cases}$$

with β a sentential form of G.

Follow

Let $G = \langle V, \Sigma, R, S \rangle$ be a CFG.

Definition

For every $A \in V$,

$$Follow(A) = \{a \in \Sigma \mid S \stackrel{*}{\Rightarrow} \alpha\, A\, a\, \beta\} \cup \Xi$$

where

$$\Xi = \begin{cases} \{\$\} & \text{if } S \stackrel{*}{\Rightarrow} \alpha\, A \\ \varnothing & \text{otherwise} \end{cases}$$

with α and β sentential forms of G and \$ is not.

Simple Example

Example

Consider

$$S \longrightarrow \mathtt{a}S\mathtt{b} \mid T$$

$$T \longrightarrow \mathtt{a}T \mid \varepsilon$$

- $\mathit{First}(\mathtt{a}) = \{\mathtt{a}\}$ $\qquad$ $\mathit{First}(\mathtt{b}) = \{\mathtt{b}\}$.
- $\mathit{First}(S) = \mathit{First}(T) = \{\mathtt{a}, \varepsilon\}$.
- $\mathit{Follow}(S) = \mathit{Follow}(T) = \{\mathtt{b}, \$\}$

Simple Example

Example

Consider

$$S \longrightarrow \mathtt{a}S\mathtt{b} \mid T$$

$$T \longrightarrow \mathtt{a}T \mid \varepsilon$$

- $\mathit{First}(\mathtt{a}) = \{\mathtt{a}\}$ $\quad$ $\mathit{First}(\mathtt{b}) = \{\mathtt{b}\}$.
- $\mathit{First}(S) = \mathit{First}(T) = \{\mathtt{a}, \varepsilon\}$.
- $\mathit{Follow}(S) = \mathit{Follow}(T) = \{\mathtt{b}, \$\}$

Simple Example

Example

Consider

$$S \longrightarrow \mathtt{a}S\mathtt{b} \mid T$$

$$T \longrightarrow \mathtt{a}T \mid \varepsilon$$

- $\mathit{First}(\mathtt{a}) = \{\mathtt{a}\}$ $\quad$ $\mathit{First}(\mathtt{b}) = \{\mathtt{b}\}$.
- $\mathit{First}(S) = \mathit{First}(T) = \{\mathtt{a}, \varepsilon\}$.
- $\mathit{Follow}(S) = \mathit{Follow}(T) = \{\mathtt{b}, \$\}$

Simple Example

Example

Consider

$$S \longrightarrow \mathtt{a}S\mathtt{b} \mid T$$

$$T \longrightarrow \mathtt{a}T \mid \varepsilon$$

- $\mathit{First}(\mathtt{a}) = \{\mathtt{a}\}$ $\quad\quad$ $\mathit{First}(\mathtt{b}) = \{\mathtt{b}\}$.
- $\mathit{First}(S) = \mathit{First}(T) = \{\mathtt{a}, \varepsilon\}$.
- $\mathit{Follow}(S) = \mathit{Follow}(T) = \{\mathtt{b}, \$\}$

Computing *First* for Single Symbols

for all $a \in \Sigma$ **do**
 $\mathit{First}(a) = \{a\}$
for all $A \in V$ **do**
 $\mathit{First}(A) = \{\}$
$\mathit{change} = \text{TRUE}$
while (change) **do**
 $\mathit{change} = \text{FALSE}$
 for all $(A \longrightarrow B_1 \cdots B_k) \in R$ **do**
  if $\varepsilon \in \mathit{First}(B_1) \cap \cdots \cap \mathit{First}(B_k)$ **then** {//This also covers the case when $k = 0$}
   if $\varepsilon \notin \mathit{First}(A)$ **then**
    $\mathit{First}(A) = \mathit{First}(A) \cup \{\varepsilon\}$
    $\mathit{change} = \text{TRUE}$
  else
   for $i = 1$ to k **do**
    if $(i == 1)$ or $(\varepsilon \in \mathit{First}(B_1) \cap \cdots \cap \mathit{First}(B_{i-1}))$ **then**
     if $(\mathit{First}(B_i) - \{\varepsilon\}) \not\subseteq \mathit{First}(A)$ **then**
      $\mathit{First}(A) = \mathit{First}(A) \cup (\mathit{First}(B_i) - \{\varepsilon\})$
      $\mathit{change} = \text{TRUE}$

Computing *First*

How?

Computing *Follow*

for all $A \in V$ **do**
  $Follow(A) = \{\}$
$Follow(S) = \{\$\}$
$change$ = TRUE
while ($change$) **do**
  $change$ = FALSE
  for all $(A \longrightarrow \alpha B \beta) \in R$ **do**
    if $(First(\beta) - \{\varepsilon\}) \not\subseteq Follow(B)$ **then**
      $Follow(B) = Follow(B) \cup (First(\beta) - \{\varepsilon\})$
      $change$ = TRUE
    if $\varepsilon \in First(\beta)$ **then**
      if $Follow(A) \not\subseteq Follow(B)$ **then**
        $Follow(B) = Follow(B) \cup Follow(A)$
        $change$ = TRUE

Exercise

Example

$$
\begin{array}{lcl}
E & \longrightarrow & T\ E' \\
E' & \longrightarrow & +\ T\ E' \mid \varepsilon \\
T & \longrightarrow & F\ T' \\
T' & \longrightarrow & *\ F\ T' \mid \varepsilon \\
F & \longrightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

Compute *First* and *Follow* for all non-terminals.

Exercise: Solution

Example

- $First(E) = First(T) = First(F) = \{(, \mathbf{id}\}$
- $First(E') = \{+, \varepsilon\}$
- $First(T') = \{*, \varepsilon\}$
- $Follow(E) = Follow(E') = \{), \$\}$
- $Follow(T) = Follow(T') = \{+,), \$\}$
- $Follow(F) = \{+, *,), \$\}$

Outline

1. Top-Down Parsing
2. *First* and *Follow*
3. Predictive Top-Down Parsing
4. Predictive Parsing with Pushdown Automata

What is it?

A predictive top-down parser is a top-down parser which will fail only if the input is ungrammatical.

- That is, predictive parsers always choose the right production rule to apply, if one exists.
- Clearly, this is not always possible.
- However, it is always possible for certain classes of CFGs.

What is it?

A predictive top-down parser is a top-down parser which will fail only if the input is ungrammatical.

- That is, predictive parsers always choose the right production rule to apply, if one exists.
- Clearly, this is not always possible.
- However, it is always possible for certain classes of CFGs.

$LL(1)$ Grammars

Definition

A CFG $G = \langle V, \Sigma, R, S \rangle$ is an $LL(1)$ grammar if whenever $(A \longrightarrow \alpha \mid \beta) \in R$ we have

1. $First(\alpha) \cap First(\beta) = \varnothing$;
2. if $\varepsilon \in First(\alpha)$, then $First(\beta) \cap Follow(A) = \varnothing$; and
3. if $\varepsilon \in First(\beta)$, then $First(\alpha) \cap Follow(A) = \varnothing$

Note: $LL(1)$ stands for left-to-right input scanning in a left-most derivation with 1 input symbol of lookahead.

$LL(1)$ Grammars

Definition

A CFG $G = \langle V, \Sigma, R, S \rangle$ is an $LL(1)$ grammar if whenever $(A \longrightarrow \alpha \mid \beta) \in R$ we have

1. $First(\alpha) \cap First(\beta) = \varnothing$;
2. if $\varepsilon \in First(\alpha)$, then $First(\beta) \cap Follow(A) = \varnothing$; and
3. if $\varepsilon \in First(\beta)$, then $First(\alpha) \cap Follow(A) = \varnothing$

Note: $LL(1)$ stands for left-to-right input scanning in a left-most derivation with 1 input symbol of lookahead.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in First(\alpha)$, or
  2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
    1. $a \in First(\alpha)$, or
    2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in First(\alpha)$, or
  2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in First(\alpha)$, or
  2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in First(\alpha)$, or
  2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in \mathit{First}(\alpha)$, or
  2. $\varepsilon \in \mathit{First}(\alpha)$ and $a \in \mathit{Follow}(A)$.
- $M[A, a] = \varnothing$ is an error entry.

The Predictive Parsing Table

- The predictive parsing table, for a given CFG, is a $|V| \times (|\Sigma| + 1)$ table M.
- Rows are indexed by V; columns are indexed by $\Sigma \cup \{\$\}$.
- $M[A, a] \subseteq \{(A \longrightarrow \alpha) \in R\}$.
- In particular, $(A \longrightarrow \alpha) \in M[A, a]$ if
  1. $a \in First(\alpha)$, or
  2. $\varepsilon \in First(\alpha)$ and $a \in Follow(A)$.
- $M[A, a] = \varnothing$ is an error entry.

Exercise 1

Example

Construct the parsing table of the grammar

$$\begin{array}{lcl} E & \longrightarrow & T\ E' \\ E' & \longrightarrow & +\ T\ E' \mid \varepsilon \\ T & \longrightarrow & F\ T' \\ T' & \longrightarrow & *\ F\ T' \mid \varepsilon \\ F & \longrightarrow & (\ E\ ) \mid \mathbf{id} \end{array}$$

Exercise 1: Solution

Example

M	**id**	$+$	$*$	(	)	\$
E	$E \longrightarrow TE'$			$E \longrightarrow T\,E'$		
E'		$E' \longrightarrow +TE'$			$E' \longrightarrow \varepsilon$	$E' \longrightarrow \varepsilon$
T	$T \longrightarrow FT'$			$T \longrightarrow FT'$		
T'		$T' \longrightarrow \varepsilon$	$T' \longrightarrow *FT'$		$T' \longrightarrow \varepsilon$	$T' \longrightarrow \varepsilon$
F	$F \longrightarrow \mathbf{id}$			$F \longrightarrow (E)$		

Here we are using the *First* and *Follow* sets computed before .

Exercise 2

Example

Construct the parsing table of the grammar

$$
\begin{array}{lcl}
S & \longrightarrow & \mathbf{i}\; E\; \mathbf{t}\; S\; S' \mid \mathbf{a} \\
S' & \longrightarrow & \mathbf{e}\; S \mid \varepsilon \\
E & \longrightarrow & \mathbf{b}
\end{array}
$$

Exercise 2: Solution

Example

	a	**b**	**e**	**i**	**t**	$
S	$S \longrightarrow \mathbf{a}$			$S \longrightarrow \mathbf{i}E\mathbf{t}SS'$		
S'			$S' \longrightarrow \varepsilon$ $S' \longrightarrow \mathbf{e}S$			$S' \longrightarrow \varepsilon$
E		$E \longrightarrow \mathbf{b}$				

Important!

Observation

If $G = \langle V, \Sigma, R, S\rangle$ is an $LL(1)$ CFG then $|M[A, a]| \leq 1$, for every $A \in V$ and $a \in \Sigma \cup \{\$\}$.

- The parsing table can be used to direct the choice of a production during parsing.
- If the grammar is an $LL(1)$ grammar, parsing will be predictive.
- Most programming constructs may be described by $LL(1)$ grammars.

Important!

Observation

If $G = \langle V, \Sigma, R, S \rangle$ is an $LL(1)$ CFG then $|M[A, a]| \leq 1$, for every $A \in V$ and $a \in \Sigma \cup \{\$\}$.

- The parsing table can be used to direct the choice of a production during parsing.
- If the grammar is an $LL(1)$ grammar, parsing will be predictive.
- Most programming constructs may be described by $LL(1)$ grammars.

Grammars Not $LL(1)$

- Some CFGs, though not $LL(1)$, may be transformed to equivalent $LL(1)$ grammars by left-factoring and left-recursion elimination.
- Ambiguous grammars, in general, may not; see Exercise 2.
- Note, however, that we can always opt for one of the multiple rules to enforce a disambiguation policy.
- In Exercise 2, if we always choose $S' \longrightarrow \mathbf{e}S$, we are effectively associating **e** with the closest **i**.

Outline

1. Top-Down Parsing
2. *First* and *Follow*
3. Predictive Top-Down Parsing
4. Predictive Parsing with Pushdown Automata

From CFGs to PDA

Lemma (Sipser 2.21)

If a language is context-free, then some PDA recognizes it.

Proof Strategy

- Let L be a CFL and G a CFG such that $L(G) = L$.
- We construct a PDA P such that $L(P) = L$.
- P determines whether an input string is in L by trying to derive it (leftmost) using G.
- It uses the stack as a scratch pad where it records the steps of the derivation.
- At each transition, it rewrites the topmost variable on the stack.
- A variable is brought to the top of the stack by matching top-of-the-stack terminals to input symbols.

From CFGs to PDA

Lemma (Sipser 2.21)

If a language is context-free, then some PDA recognizes it.

Proof Strategy

- Let L be a CFL and G a CFG such that $L(G) = L$.
- We construct a PDA P such that $L(P) = L$.
- P determines whether an input string is in L by trying to derive it (leftmost) using G.
- It uses the stack as a scratch pad where it records the steps of the derivation.
- At each transition, it rewrites the topmost variable on the stack.
- A variable is brought to the top of the stack by matching top-of-the-stack terminals to input symbols.

Informal Description of P

- P has three *main* states: q_s, q_{loop}, and q_a.
- In q_s, it pushes \$ and the start variable onto the stack, and enters q_{loop}.
- While in q_{loop}
  - If the top-of-the-stack symbol is a variable A, nondeterministically select a rule $A \longrightarrow u$ of G, and replace A by u.
  - If the top-of-the-stack symbol is a terminal, match it to the next input symbol.
  - If the top-of-the-stack symbol is \$, enter q_a.

Example

Example

Draw the state diagram of a PDA that equivalent to the following CFG.

$$\begin{array}{lcl} S & \longrightarrow & \mathtt{a}T\mathtt{b} \mid \mathtt{b} \\ T & \longrightarrow & T\mathtt{a} \mid \varepsilon \end{array}$$

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A, a]$ to choose a transition.
4. If there are no more input symbols, use $M[A, \$]$.
5. If $M[A, a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A, a]$ to choose a transition.
4. If there are no more input symbols, use $M[A, \$]$.
5. If $M[A, a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A, a]$ to choose a transition.
4. If there are no more input symbols, use $M[A, \$]$.
5. If $M[A, a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A, a]$ to choose a transition.
4. If there are no more input symbols, use $M[A, \$]$.
5. If $M[A, a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A,a]$ to choose a transition.
4. If there are no more input symbols, use $M[A,\$]$.
5. If $M[A,a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.

Parser

1. Given a CFG G, construct the parsing table M.
2. Construct the equivalent PDA P.
3. When variable A is on top of the stack and the head is pointing at a, use $M[A, a]$ to choose a transition.
4. If there are no more input symbols, use $M[A, \$]$.
5. If $M[A, a] = \varnothing$, report an error.
6. If we output the sequence of rules chosen, we may reconstruct the derivation and, hence, the parse tree.