

A Procedure for Parsing (contd.)

```
Grammar:  S → (S)S
          S → a
          S → ε
```

```
case TOKEN_A: /* Production 2 */
    consume(TOKEN_A);
    return;
case TOKEN_CLOSE_PAREN:
case TOKEN_EOF: /* Production 3 */
    return;
default:
    /* Parse Error */
```

Predictive Parsing: Restrictions

1. May not be able to choose a *unique* production

In general, we may need a backtracking parser:

Recursive Descent Parsing

```
S → a B d
B → b
B → bc
```

Today's Lecture

- Predictive Parsing
- Parsing Tables
- FIRST and FOLLOW
- LL parsing and LL(1) grammars

A Procedure for Parsing

```
Grammar:  S → (S)S
          S → a
          S → ε
```

```
Algorithm parse_S() {
    switch (input_token) {
    case TOKEN_OPEN_PAREN: /* Production 1 */
        consume(TOKEN_OPEN_PAREN);
        parse_S();
        consume(TOKEN_CLOSE_PAREN);
        parse_S();
    return;
```

Predictive Parsing

Grammar: $A \rightarrow a$ $S \rightarrow A S B$
 $B \rightarrow b$ $S \rightarrow \epsilon$

```
Algorithm parse_A() {  
  consume(TOKEN_A); /* Production 1 */  
  return;  
}
```

```
Algorithm parse_B() {  
  consume(TOKEN_B); /* Production 2 */  
  return;  
}
```

Predictive Parsing (contd.)

Grammar: $A \rightarrow a$ $S \rightarrow A S B$
 $B \rightarrow b$ $S \rightarrow \epsilon$

```
Algorithm parse_S() {  
  switch (input_token) {  
    case TOKEN_A: /* Production 3 */  
      parse_A();  
      parse_S();  
      parse_B();  
      return;  
    case TOKEN_B:  
    case TOKEN_EOF: /* Production 4 */  
      return;  
  }  
}
```

Recursive Descent Parsing

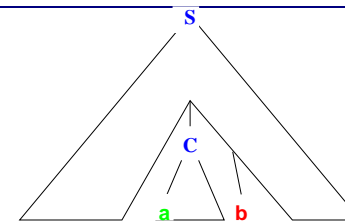
Grammar: $S \rightarrow a B d$
 $B \rightarrow b$
 $B \rightarrow bc$

```
Algorithm parse_B() {  
  switch (input_token) {  
    case TOKEN_B: /* Production 2 */  
      consume(TOKEN_B);  
      return;  
    case TOKEN_C: /* Production 3 */  
      consume(TOKEN_B);  
      consume(TOKEN_C);  
      return;  
  }  
}
```

FIRST and FOLLOW

Grammar: $S \rightarrow (S)S \mid a \mid \epsilon$

- $FIRST(X)$ = First character of any string that can be derived from X
 $FIRST(S) = \{ (, a, \epsilon \}$.
- $FOLLOW(A)$ = First character that, in any derivation of a string in the language, appears immediately after A .
 $FOLLOW(S) = \{ \}, EOF\}$



$a \in FIRST(C)$
 $b \in FOLLOW(C)$

FIRST and FOLLOW

Grammar: $A \rightarrow a \quad S \rightarrow A S B$
 $B \rightarrow b \quad S \rightarrow \epsilon$

$FIRST(X)$: First terminal in some α such that $X \xrightarrow{*} \alpha$.
 $FOLLOW(A)$: First terminal in some β such that $S \xrightarrow{*} \alpha A \beta$.

$FIRST(S) = \{ a, \epsilon \}$ $FOLLOW(S) = \{ b, EOF \}$
 $FIRST(A) = \{ a \}$ $FOLLOW(A) = \{ a, b \}$
 $FIRST(B) = \{ b \}$ $FOLLOW(B) = \{ b, EOF \}$

Definition of FIRST

Grammar: $A \rightarrow a \quad S \rightarrow A S B$
 $B \rightarrow b \quad S \rightarrow \epsilon$

$FIRST(\alpha)$ is the smallest set such that

$\alpha =$	Property of $FIRST(\alpha)$
a , a terminal	$a \in FIRST(\alpha)$
A , a nonterminal	$A \rightarrow \epsilon \in G \implies \epsilon \in FIRST(\alpha)$ $A \rightarrow \beta \in G, \beta \neq \epsilon \implies FIRST(\beta) \subseteq FIRST(\alpha)$
X_1, X_2, \dots, X_k , a string of terminals and nonterminals	$FIRST(X_1) - \{\epsilon\} \subseteq FIRST(\alpha)$ $FIRST(X_i) \subseteq FIRST(\alpha)$ if $\forall j < i \quad \epsilon \in FIRST(X_j)$ $\epsilon \in FIRST(\alpha)$ if $\forall j < k \quad \epsilon \in FIRST(X_j)$

Parsing Table

Grammar: $A \rightarrow a \quad S \rightarrow A S B$
 $B \rightarrow b \quad S \rightarrow \epsilon$

```
Algorithm parse_S() {
  switch (input_token) {
    case TOKEN_A: /* Production 3 */
      parse_A();
      parse_S();
      parse_B();
      return;
    case TOKEN_B:
    case TOKEN_EOF: /* Production 4 */
      return;
  }
}
```

NONTERMINAL	INPUT SYMBOL		
	a	b	EOF
S	$S \rightarrow A S B$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$

Table-driven Parsing

Grammar: $A \rightarrow a \quad S \rightarrow A S B$
 $B \rightarrow b \quad S \rightarrow \epsilon$

Parsing Table:

NONTERMINAL	INPUT SYMBOL		
	a	b	EOF
S	$S \rightarrow A S B$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$
A	$A \rightarrow a$		
B		$B \rightarrow b$	

A Procedure to Construct Parsing Tables

```

Algorithm table_construct(G) {
  for each nonterminal A and terminal a
    M[A, a] = ⊥ /* initialize entries to error */

  for each  $A \rightarrow \alpha \in G$  {
    for each  $a \in FIRST(\alpha)$  such that  $a \neq \epsilon$ 
      add  $A \rightarrow \alpha$  to M[A, a];
    if  $\epsilon \in FIRST(\alpha)$ 
      for each  $b \in FOLLOW(A)$ 
        add  $A \rightarrow \alpha$  to M[A, b];
  }
}

```

Recursive Descent Parsing: Restrictions

Grammar cannot be left-recursive

Example: $E \rightarrow E + E \mid a$

```

Algorithm parse_E() {
  switch (input_token) {
    case TOKEN_a: /* Production 1 */
      parse_E();
      consume(TOKEN_PLUS);
      parse_E();
      return;
    case TOKEN_a: /* Production 2 */
      consume(TOKEN_a);
      return;
  }
}

```

Definition of FOLLOW

Grammar: $A \rightarrow a$ $S \rightarrow A S B$
 $B \rightarrow b$ $S \rightarrow \epsilon$

FOLLOW(*A*) is the smallest set such that

<i>A</i>	Property of <i>FOLLOW</i> (<i>A</i>)
= <i>S</i> , the start symbol	EOF ∈ <i>FOLLOW</i> (<i>S</i>) Book notation: \$ ∈ <i>FOLLOW</i> (<i>S</i>)
$B \rightarrow \alpha A \beta \in G$	$FIRST(\beta) - \{\epsilon\} \subseteq FOLLOW(A)$
$B \rightarrow \alpha A$, or $B \rightarrow \alpha A \beta, \epsilon \in FIRST(\beta)$	$FOLLOW(B) \subseteq FOLLOW(A)$

Constructing Parsing Table

Grammar: $A \rightarrow a$ $S \rightarrow A S B$
 $B \rightarrow b$ $S \rightarrow \epsilon$

$FIRST(S) = \{a, \epsilon\}$ $FOLLOW(S) = \{b, EOF\}$
 $FIRST(A) = \{a\}$ $FOLLOW(A) = \{a, b\}$
 $FIRST(B) = \{b\}$ $FOLLOW(B) = \{b, EOF\}$

NONTERMINAL	INPUT SYMBOL		
	a	b	EOF
<i>S</i>	$S \rightarrow A S B$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$
<i>A</i>	$A \rightarrow a$		
<i>B</i>		$B \rightarrow b$	

LL(1) Grammars

Grammars for which the recursive descent parsing table has no multiple entries.

$$E \rightarrow id E'$$

$$E' \rightarrow + E E'$$

$$E' \rightarrow \epsilon$$

NONTERMINAL	INPUT SYMBOL		
	id	+	EOF
E	$E \rightarrow id E'$		
E'		$E' \rightarrow + E E'$	$E' \rightarrow \epsilon$

Parsing with LL(1) Grammars

NONTERMINAL	INPUT SYMBOL		
	id	+	EOF
E	$E \rightarrow id E'$		
E'		$E' \rightarrow + E E'$	$E' \rightarrow \epsilon$

$\$E$	id + id\$	$E \Rightarrow idE'$
$\$E'id$	id + id\$	
$\$E'$	+ id\$	$\Rightarrow id+EE'$
$\$E'E+$	+ id\$	
$\$E'E$	id\$	$\Rightarrow id+idE'E'$
$\$E'E'id$	id\$	
$\$E'E'$	\$	$\Rightarrow id+idE'$
$\$E'$	\$	$\Rightarrow id+id$
\$	\$	

Removing Left Recursion

$$A \rightarrow A a$$

$$A \rightarrow b$$

$$\mathcal{L}(A) = \{b, ba, baa, baaa, baaaa, \dots\}$$

$$A \rightarrow bA'$$

$$A' \rightarrow aA'$$

$$A' \rightarrow \epsilon$$

Removing Left Recursion: An Example

$$E \rightarrow E + E$$

$$E \rightarrow id$$

↓

$$E \rightarrow id E'$$

$$E' \rightarrow + E E'$$

$$E' \rightarrow \epsilon$$

LL(1) Derivations

Left to Right Scan of input

Leftmost Derivation

(1) look ahead 1 token at each step

Alternative characterization of LL(1) Grammars:

Whenever $A \rightarrow \alpha \mid \beta \in G$

1. $FIRST(\alpha) \cap FIRST(\beta) = \{ \}$, and
2. if $\alpha \xRightarrow{*} \epsilon$ then $FIRST(\beta) \cap FOLLOW(A) = \{ \}$.

Corollary: No Ambiguous Grammar is LL(1).

Next Lecture

Shift-Reduce Parsing