Chapter 4 Cont.

Recursive-Descent Parsing (continued)

- The LL Grammar Class
  - The Left Recursion Problem
    - If a grammar has left recursion, either direct or indirect, it cannot be the basis for a top-down parser
      - A grammar can be modified to remove direct left recursion as follows:

For each nonterminal, A,

1. Group the A-rules as $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

   where none of the β's begins with A

2. Replace the original A-rules with

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

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

Copyright © 2018 Pearson. All rights reserved.

Recursive-Descent Parsing (continued)

- The other characteristic of grammars that disallows top-down parsing is the lack of pairwise disjointness
  - The inability to determine the correct RHS on the basis of one token of lookahead
  - Def: FIRST(α) = {a | α =>* aβ }

    (If α =>* ε, ε is in FIRST(α))

Copyright © 2018 Pearson. All rights reserved.

Recursive-Descent Parsing (continued)

- Pairwise Disjointness Test:
  - For each nonterminal, A, in the grammar that has more than one RHS, for each pair of rules, $A \rightarrow \alpha_i$ and $A \rightarrow \alpha_j$, it must be true that

    $FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$
- Example:

  $A \rightarrow a \mid bB \mid cAb$

  $A \rightarrow a \mid aB$

Copyright © 2018 Pearson. All rights reserved.

Recursive-Descent Parsing (continued)

- Left factoring can resolve the problem

  Replace

<variable> → identifier | identifier [<expression>]

  with

<variable> → identifier <new>

<new> → ε | [<expression>]

  or

<variable> → identifier [[<expression>]]

(the outer brackets are metasymbols of EBNF)

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing

- The parsing problem is finding the correct RHS in a right-sentential form to reduce to get the previous right-sentential form in the derivation

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Intuition about handles:
  - Def: β is the *handle* of the right sentential form

    $\gamma = \alpha\beta w$ if and only if $S \Rightarrow^*_{rm} \alpha A w \Rightarrow_{rm} \alpha\beta w$

  - Def: β is a *phrase* of the right sentential form

    γ if and only if $S \Rightarrow^* \gamma = \alpha_1 A \alpha_2 \Rightarrow^+ \alpha_1\beta\alpha_2$

  - Def: β is a *simple phrase* of the right sentential form γ if and only if $S \Rightarrow^* \gamma = \alpha_1 A \alpha_2 \Rightarrow \alpha_1\beta\alpha_2$

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Intuition about handles (continued):
  - The handle of a right sentential form is its leftmost simple phrase
  - Given a parse tree, it is now easy to find the handle
  - Parsing can be thought of as handle pruning

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Shift-Reduce Algorithms
  - Reduce is the action of replacing the handle on the top of the parse stack with its corresponding LHS
  - Shift is the action of moving the next token to the top of the parse stack

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Advantages of LR parsers:
  - They will work for nearly all grammars that describe programming languages.
  - They work on a larger class of grammars than other bottom-up algorithms, but are as efficient as any other bottom-up parser.
  - They can detect syntax errors as soon as it is possible.
  - The LR class of grammars is a superset of the class parsable by LL parsers.

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- LR parsers must be constructed with a tool
- Knuth's insight: A bottom-up parser could use the entire history of the parse, up to the current point, to make parsing decisions
  - There are only a finite and relatively small number of different parse situations that could have occurred, so the history could be stored in a parser state, on the parse stack

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- An LR configuration stores the state of an LR parser

$(S_0X_1S_1X_2S_2…X_mS_m, a_ia_{i+1}…a_n\$)$

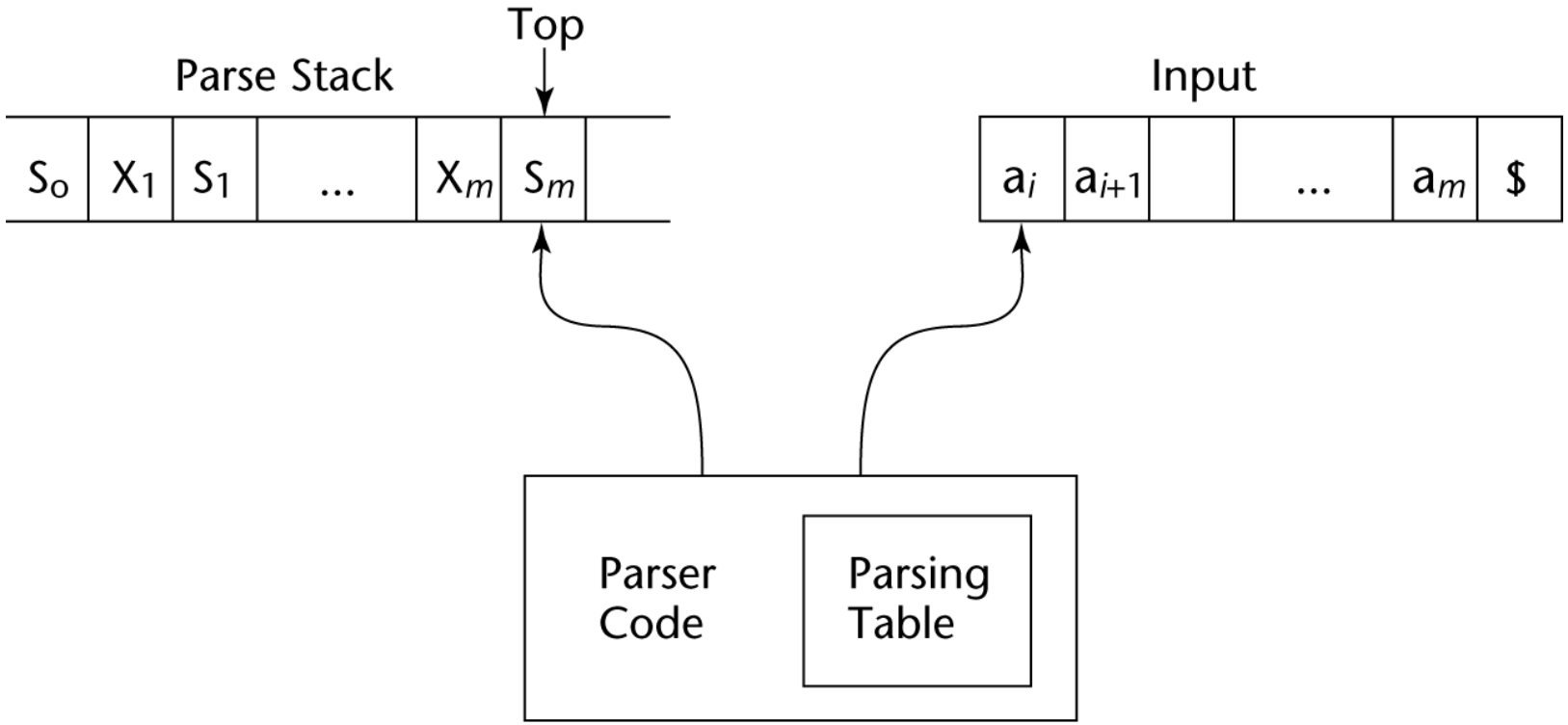
Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- LR parsers are table driven, where the table has two components, an ACTION table and a GOTO table
  - The ACTION table specifies the action of the parser, given the parser state and the next token
    - Rows are state names; columns are terminals
  - The GOTO table specifies which state to put on top of the parse stack after a reduction action is done
    - Rows are state names; columns are nonterminals

Copyright © 2018 Pearson. All rights reserved.

Structure of An LR Parser



Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Initial configuration: (S_0, $a_1 \ldots a_n\$$)
- Parser actions:
  - For a Shift, the next symbol of input is pushed onto the stack, along with the state symbol that is part of the Shift specification in the Action table
  - For a Reduce, remove the handle from the stack, along with its state symbols. Push the LHS of the rule. Push the state symbol from the GOTO table, using the state symbol just below the new LHS in the stack and the LHS of the new rule as the row and column into the GOTO table

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- Parser actions (continued):
  - For an Accept, the parse is complete and no errors were found.
  - For an Error, the parser calls an error-handling routine.

Copyright © 2018 Pearson. All rights reserved.

LR Parsing Table

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5		S4				1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

Copyright © 2018 Pearson. All rights reserved.

Bottom-up Parsing (continued)

- A parser table can be generated from a given grammar with a tool, e.g., **`yacc`** or **`bison`**

Copyright © 2018 Pearson. All rights reserved.

Summary

- Syntax analysis is a common part of language implementation
- A lexical analyzer is a pattern matcher that isolates small-scale parts of a program
  - Detects syntax errors
  - Produces a parse tree
- A recursive-descent parser is an LL parser
  - EBNF
- Parsing problem for bottom-up parsers: find the substring of current sentential form
- The LR family of shift-reduce parsers is the most common bottom-up parsing approach

Copyright © 2018 Pearson. All rights reserved.