Syntax Analysis
(Part 4)

CSE 415: Compiler Construction

Bottom-up Parsing

- Begin at the leaves, build the parse tree in small segments, combine the small trees to make bigger trees, until the root is reached
- This process is called *reduction* of the sentence to the start symbol of the grammar
- One of the ways of "reducing" a sentence is to follow the rightmost derivation of the sentence in *reverse*
  - *Shift-Reduce* parsing implements such a strategy
  - It uses the concept of a *handle* to detect when to perform reductions

Shift-Reduce (SR) Parsing

- **Handle**: A *handle* of a right sentential form γ, is a production $A \rightarrow \beta$ *and* a position in γ, where the string β may be found and replaced by A, to produce the previous right sentential form in a rightmost derivation of γ
- That is, if $S \Rightarrow^*_{rm} \alpha A w \Rightarrow_{rm} \alpha \beta w$, then $A \rightarrow \beta$ in the position following α is a handle of $\alpha\beta w$
- A handle will always eventually appear on the top of the stack, never submerged inside the stack
- In S-R parsing, we locate the handle and reduce it by the LHS of the production repeatedly, to reach the start symbol
- These reductions, in fact, trace out a rightmost derivation of the sentence in reverse. This process is called handle pruning
- *LR-Parsing* is a method of shift-reduce parsing

SR Parsing Example 1

1. $S \rightarrow aAcBe,\ A \rightarrow Ab \mid b,\ B \rightarrow d$

   For the string = *abbcde*, the rightmost derivation marked with handles is shown below

$$
\begin{aligned}
S &\Rightarrow \underline{aAcBe} && (aAcBe,\ S \rightarrow aAcBe)\\
&\Rightarrow aAc\underline{d}e && (d,\ B \rightarrow d)\\
&\Rightarrow a\underline{Ab}cde && (Ab,\ A \rightarrow Ab)\\
&\Rightarrow a\underline{b}bcde && (b,\ A \rightarrow b)
\end{aligned}
$$

The handle is unique if the grammar is unambiguous!

SR Parsing Example 2

$E \to E + E$, $E \to E * E$, $E \to (E)$, $E \to id$

For the string = $id + id * id$, two rightmost derivation marked with handles are shown below

$$
\begin{aligned}
E &\Rightarrow \underline{E + E} \quad (E + E,\ E \to E + E) \\
&\Rightarrow E + \underline{E * E} \quad (E * E,\ E \to E * E) \\
&\Rightarrow E + E * \underline{id} \quad (id,\ E \to id) \\
&\Rightarrow E + \underline{id} * id \quad (id,\ E \to id) \\
&\Rightarrow \underline{id} + id * id \quad (id,\ E \to id) \\
E &\Rightarrow \underline{E * E} \quad (E * E,\ E \to E * E) \\
&\Rightarrow E * \underline{id} \quad (id,\ E \to id) \\
&\Rightarrow E + E * \underline{id} \quad (E + E,\ E \to E + E) \\
&\Rightarrow E + \underline{id} * id \quad (id,\ E \to id) \\
&\Rightarrow \underline{id} + id * id \quad (id,\ E \to id)
\end{aligned}
$$

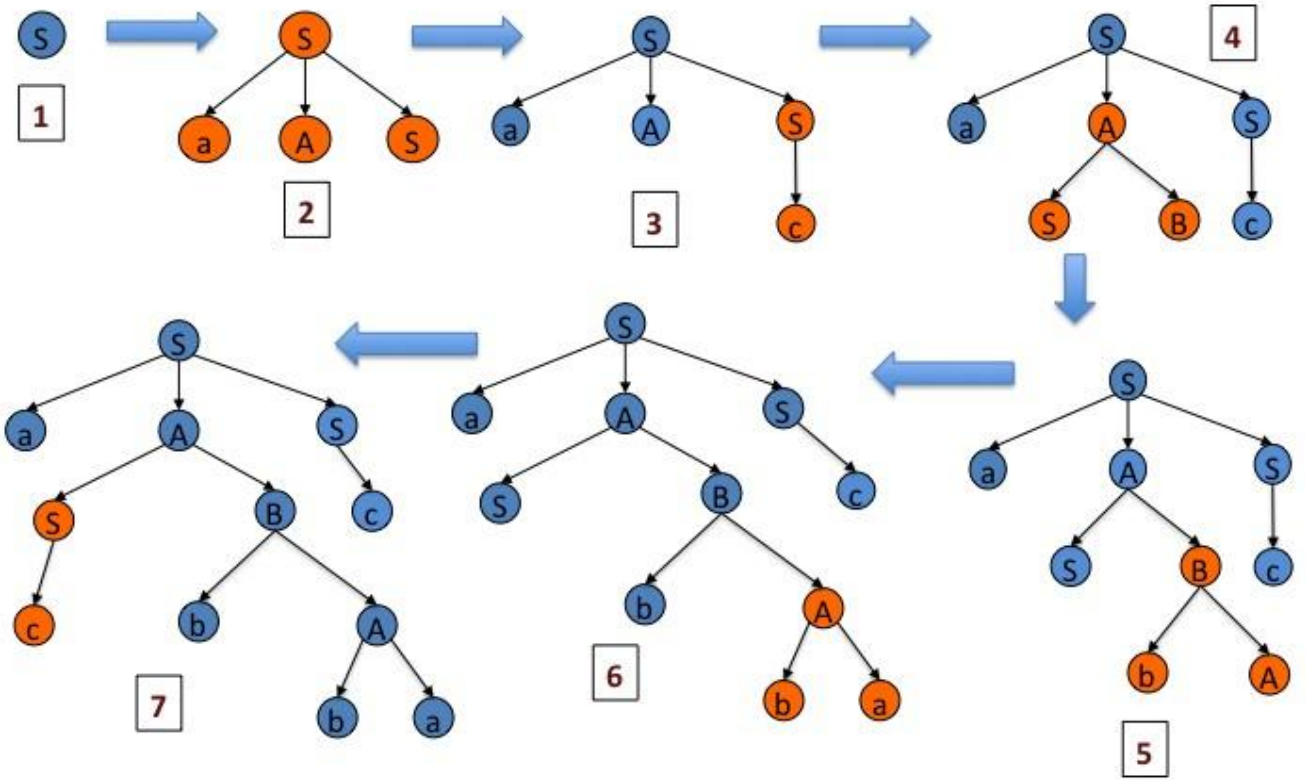
Right-most Derivation and Bottom-up Parsing

S → aAS | c
A → ba | SB
B → bA | S

Rightmost derivation of the string *acbbac*

S => aAS => aAc => aSBc => aSbAc => aSbbac => acbbac

1 2 3 4 5 6 7



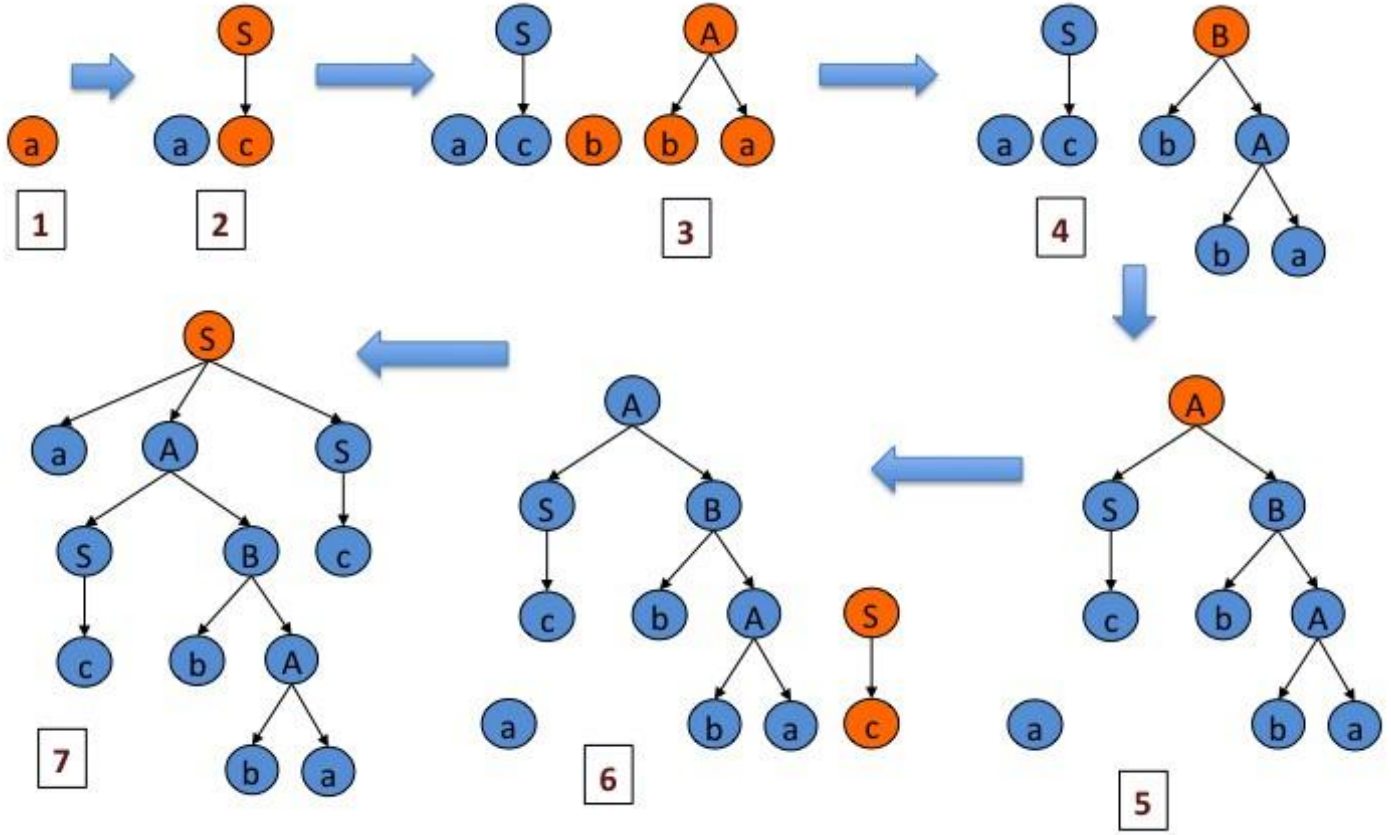
Right-most Derivation and Bottom-up Parsing

S → aAS | c
A → ba | SB
B → bA | S

Rightmost derivation of the string *acbbac* in reverse

S <= aAS <= aAc <= aSBc <= aSbAc <= aSbbac <= acbbac

7 6 5 4 3 2 1



SR Parsing Algorithm

- How do we locate a handle in a right sentential form?
  - An LR parser uses a DFA to detect the condition that a handle is now on the stack
- Which production to use, in case there is more than one with the same RHS?
  - An LR parser uses a parsing table similar to an LL parsing table, to choose the production
- A stack is used to implement an S-R parser, The parser has four actions
  1. **shift**: the next input symbol is shifted to the top of stack
  2. **reduce**: the right end of the handle is the top of stack; locates the left end of the handle inside the stack and replaces the handle by the LHS of an appropriate production
  3. **accept**: announces successful completion of parsing
  4. **error**: syntax error, error recovery routine is called

SR Parsing Algorithm Trace – Example 1

$ marks the bottom of stack and the right end of the input

Stack	Input	Action
$	*acbbac*$	shift
$ *a*	*cbbac*$	shift
$ *ac*	*bbac*$	reduce by $S \rightarrow c$
$ *aS*	*bbac*$	shift
$ *aSb*	*bac*$	shift
$ *aSbb*	*ac*$	shift
$ *aSbba*	*c*$	reduce by $A \rightarrow ba$
$ *aSbA*	*c*$	reduce by $B \rightarrow bA$
$ *aSB*	*c*$	reduce by $A \rightarrow SB$
$ *aA*	*c*$	shift
$ *aAc*	$	reduce by $S \rightarrow c$
$ *aAS*	$	reduce by $S \rightarrow aAS$
$ *S*	$	accept