4.5: Bottom-Up Parsing

**Bottom-Up Parsing**

- Starts at leaves (bottom) of parse tree and works upward to start symbol
- A.k.a. *Shift-Reduce* Parsing
- *Shift*: put RHS of rule on stack
- *Reduce*: pop RHS and push LHS, consuming input
- Outputs a rightmost derivation, in reverse, of the input string

**Bottom-Up Parsing: Example**

**Grammar:**

\[
S \rightarrow aAbBe \\
A \rightarrow Abc | b \\
B \rightarrow d
\]

**Input:** `abbcde`

**Output:**

```
abbcde
aAbcde
aAde
aAbBe
S
```

**Bottom-Up Parsing: Handles**

- *Handle*: Substring matching RHS of production.
- Replacement of RHS by LHS must be a step in a reverse rightmost derivation.

**Yes:**

```
abbcde
aAbcde
aAde
aAbBe
S
```

**No:**

```
abbcde
aAbcde
aAAdcde
STUCK!

S \rightarrow aAbBe \\
A \rightarrow Abc | b \\
B \rightarrow d
```
Bottom-Up Parsing: Handles

- Formally: if \( S \Rightarrow_r \alpha \alpha \Rightarrow \alpha \beta w \), then \( A \Rightarrow \beta \) in the position following \( \alpha \) is a handle of \( \alpha \beta w \).
- Often just refer to \( \beta \) as the handle.
- Reducing \( \beta \) to \( A \) is called “pruning”.
- For an unambiguous grammar, every right-sentential form of the grammar has exactly one handle.

Bottom-Up Parsing: Handle Pruning

Example: \( E \rightarrow E + E \mid E \ast E \mid (E) \mid \text{id} \)

<table>
<thead>
<tr>
<th>Right-Sentential Form</th>
<th>Handle</th>
<th>Reducing Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (_1 ) + id (_2 ) \ast id (_3 )</td>
<td>id (_1 )</td>
<td>( E \rightarrow \text{id} )</td>
</tr>
<tr>
<td>E + id (_2 ) \ast id (_3 )</td>
<td>id (_2 )</td>
<td>( E \rightarrow \text{id} )</td>
</tr>
<tr>
<td>E + E \ast id (_3 )</td>
<td>id (_3 )</td>
<td>( E \rightarrow \text{id} )</td>
</tr>
<tr>
<td>E + E \ast E \ast E</td>
<td>E \ast E</td>
<td>( E \rightarrow \text{E} \ast \text{E} )</td>
</tr>
<tr>
<td>E \ast E</td>
<td>E \ast E</td>
<td>( E \rightarrow \text{E} \ast \text{E} )</td>
</tr>
</tbody>
</table>

Stack Implementation of Shift-Reduce Parsing

- Two problems to solve
  1) Locate substring to be reduced
  2) Choose which production to use if more than one has that substring as a RHS
- As in top-down parsing, starts with stack having $ on the bottom (but not start symbol), and $ at end of input
- Parser shifts zero or more input symbols onto stack until a handle \( \beta \) is on top of the stack.
- Then reduces \( \beta \) using appropriate production
- Success means start symbol on stack and input = $
Stack Implementation of Shift-Reduce Parsing

- Possible actions:
  1. *Shift*: push next input symbol onto stack
  2. *Reduce*: right end of handle is on top of stack. Locate left end of handle in stack and pick nonterminal to replace handle.
  3. *Accept*: Report success

Example: $\text{id}_1 + \text{id}_2 \ast \text{id}_3$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$\text{id}_1 + \text{id}_2 \ast \text{id}_3$</td>
<td>shift</td>
</tr>
<tr>
<td>$S \text{id}_1$</td>
<td>$+ \text{id}_2 \ast \text{id}_3$</td>
<td>reduce by $E \rightarrow \text{id}$</td>
</tr>
<tr>
<td>$S E$</td>
<td>$+ \text{id}_2 \ast \text{id}_3$</td>
<td>shift</td>
</tr>
<tr>
<td>$S E \text{id}_1$</td>
<td>$\ast \text{id}_2 \ast \text{id}_3$</td>
<td>shift</td>
</tr>
<tr>
<td>$S E + \text{id}_2$</td>
<td>$\ast \text{id}_2 \ast \text{id}_3$</td>
<td>reduce by $E \rightarrow \text{id}$</td>
</tr>
<tr>
<td>$S E + E \ast $</td>
<td>$\ast \text{id}_2 \ast \text{id}_3$</td>
<td>shift</td>
</tr>
<tr>
<td>$S E + E \ast \text{id}_1$</td>
<td>$S$</td>
<td>reduce by $E \rightarrow \text{id}$</td>
</tr>
<tr>
<td>$S E + E \ast E$</td>
<td>$S$</td>
<td>reduce by $E \rightarrow \text{id} \ast E$</td>
</tr>
<tr>
<td>$S E + E$</td>
<td>$S$</td>
<td>reduce by $E \rightarrow \text{id} \ast E$</td>
</tr>
<tr>
<td>$S E$</td>
<td>$S$</td>
<td>accept</td>
</tr>
</tbody>
</table>

Conflicts During Shift-Reduce Parsing

- There are grammars for which the parser won't know what to do at a given point – i.e., they are not LR($k$) for any $k$.
- *Shift/reduce conflict*: Don’t know whether to shift or reduce
- *Reduce/reduce conflict*: Don’t know which production to reduce by
- Conflict can be caused by ambiguity, or by multiple RHS's having same token

Conflicts Caused by Ambiguity

- No ambiguous grammar can be LR($k$)
- Recall ambiguous if-then-else grammar:
  
  \[
  S \rightarrow \text{i} E \text{t} S \mid \text{i} E \text{t} S e S \mid a \\
  E \rightarrow \text{b}
  \]
- Consider parse actions on \text{i b t i b t a e a}
Shift/Reduce Conflicts Caused by Ambiguity

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$i b t i b t a e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$i</td>
<td>$b t i b t a e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$b</td>
<td>$t i b t a e a$</td>
<td>reduce by $E \rightarrow b$</td>
</tr>
<tr>
<td>$E</td>
<td>$t i b t a e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$E t</td>
<td>$b t a e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$E t b</td>
<td>t a e a$</td>
<td>reduce by $E \rightarrow b$</td>
</tr>
<tr>
<td>$E t E</td>
<td>t a e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$E t E t</td>
<td>e a$</td>
<td>shift</td>
</tr>
<tr>
<td>$E t E t a</td>
<td>e a$</td>
<td>reduce by $S \rightarrow a$</td>
</tr>
<tr>
<td>$E t E t S</td>
<td>e a$</td>
<td>???</td>
</tr>
</tbody>
</table>

Should we shift, or should we reduce by $S \rightarrow i E t S$ ?

Reduce/Reduce Conflicts

- Some languages (e.g., Matlab) use parens both for functions and arrays:
  
  ```
  fun_call -> id(expr_list)
  array_ref -> id(expr_list)
  ```

- E.g., `foo(a, b)` could be a function call or a reference into a 2D array
- So we don’t know whether to reduce by the first or second production.

Conflict Resolution

- Shift/reduce: as in top-down parsing, simply decide in favor of one option (shift) or another (reduce).
- Reduce/reduce: add keyword (proc, function) that goes before the identifier, and augment lexical analyzer to recognize a procedure identifier (procid) after that keyword. Lexical analyzer then uses symbol table to recognize procid.