Compiler Design

IT 423

Lecture - 9

Dr. Manish Khare
DAIICT, Gandhinagar
Bottom-up Parsing

➢ Bottom-up parsing starts from the leaf nodes of a tree and works in upward direction till it reaches the root node.

➢ Here, we start from a sentence and then apply production rules in reverse manner in order to reach the start symbol.

➢ This corresponds to starting at the leaves of the parse tree, and working back to the root.

➢ Bottom-up parsing is also known as shift-reduce parsing
Bottom-up Parsing

- Bottom-up parsing is also known as **shift-reduce parsing** because its two main actions are shift and reduce.
  
  - At each shift action, the current symbol in the input string is pushed to a stack.
  - At each reduction step, the symbols at the top of the stack (this symbol sequence is the right side of a production) will replaced by the non-terminal at the left side of that production.
  
  - There are also two more actions: accept and error.
A shift-reduce parser tries to reduce the given input string into the starting symbol.

a string \rightarrow the starting symbol reduced to

At each reduction step, a substring of the input matching to the right side of a production rule is replaced by the non-terminal at the left side of that production rule.

If the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

Rightmost Derivation: \[ S \Rightarrow \omega \]

Shift-Reduce Parser finds: \[ \omega \leftarrow \ldots \leftarrow S \]
Bottom-up Parsing

- “Shift-Reduce” Parsing
- Reduce a string to the start symbol of the grammar.
- At every step a particular sub-string is matched (in left-to-right fashion) to the right side of some production and then it is substituted by the non-terminal in the left hand side of the production.

Consider:

\[
\begin{align*}
S &\rightarrow aABe \\
A &\rightarrow Abc \ | \ b \\
B &\rightarrow d
\end{align*}
\]

Rightmost Derivation:

\[
S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde
\]
Thus this process of bottom-up parsing is like tracing out the rightmost derivations in reverse.

The bottom-up parsing as the process of “reducing” a token string to the start symbol of the grammar.

At each *reduction*, the token string matching the RHS of a production is replaced by the LHS non-terminal of that production.

The key decisions during bottom-up parsing are about when to reduce and about what production to apply.
Handle

- Handle of a string: Substring that matches the RHS of some production AND whose reduction to the non-terminal on the LHS is a step along the reverse of some rightmost derivation.

- A handle of a right sentential form $\gamma (\equiv \alpha \beta \omega)$ is a production rule $A \rightarrow \beta$ and a position of $\gamma$
  
  where the string $\beta$ may be found and replaced by $A$ to produce the previous right-sentential form in a rightmost derivation of $\gamma$.

$$S \Rightarrow \alpha A \omega \Rightarrow \alpha \beta \omega$$

i.e. $A \rightarrow \beta$ is a handle of $\alpha \beta \gamma$ at the location immediately after the end of $\alpha$,

- If the grammar is unambiguous, then every right-sentential form of the grammar has exactly one handle.

- $\omega$ is a string of terminals
Consider the Grammar

\[ E \to E + E \mid E \ast E \mid id \]

<table>
<thead>
<tr>
<th>Right sentential form</th>
<th>Handle</th>
<th>Production Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1 + id2 * id3</td>
<td>id1</td>
<td>E → id</td>
</tr>
<tr>
<td>E + id2 * id3</td>
<td>id2</td>
<td>E → id</td>
</tr>
<tr>
<td>E + E * id3</td>
<td>id3</td>
<td>E → id</td>
</tr>
<tr>
<td>E + E * E</td>
<td>E * E</td>
<td>E → E * E</td>
</tr>
<tr>
<td>E + E</td>
<td>E + E</td>
<td>E → E + E</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handle

Consider:

\[ S \rightarrow aABe \]
\[ A \rightarrow Abc \mid b \]
\[ B \rightarrow d \]

\[ S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde \]

It follows that:
\[ S \rightarrow aABe \] is a handle of \( aABe \) in location 1.
\[ B \rightarrow d \] is a handle of \( aAde \) in location 3.
\[ A \rightarrow Abc \] is a handle of \( aAbcde \) in location 2.
\[ A \rightarrow b \] is a handle of \( abbcde \) in location 2.
Handle Pruning

- A rightmost derivation in reverse can be obtained by “handle-pruning.”
- Apply this to the previous example.

\[
\begin{align*}
S & \rightarrow aABe \\
A & \rightarrow Abc \mid b \\
B & \rightarrow d
\end{align*}
\]

abbcde
Find the handle = b at loc. 2
aAbcde
b at loc. 3 is not a handle:
aAAcde
... blocked.
Handle Pruning

- The process of discovering a handle & reducing it to the appropriate left-hand side is called *handle pruning*. Handle pruning forms the basis for a bottom-up parsing method.

- To construct a rightmost derivation
  \[ S = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n = \omega \]
  input string

  Apply the following simple algorithm
  - Start from \( \gamma_n \), find a handle \( A_n \rightarrow \beta_n \) in \( \gamma_n \), and replace \( \beta_n \) by \( A_n \) to get \( \gamma_{n-1} \).
  - Then find a handle \( A_{n-1} \rightarrow \beta_{n-1} \) in \( \gamma_{n-1} \), and replace \( \beta_{n-1} \) by \( A_{n-1} \) to get \( \gamma_{n-2} \).
  - Repeat this, until we reach \( S \).
➢ Here it is apparent that the string appearing to the right of a handle contains only terminal symbols

➢ Since here the grammar is ambiguous the choices for the handles can be different depending upon right derivations used.

➢ This process can be made algorithmic using a stack implementation
Types of Bottom up Parsing

- Bottom-Up
- Shift-Reduce
- LR Parsing
  - SLR Parsing
  - LR Parser
  - LALR Parser
Shift –reduce Parsing

➢ Shift-reduce parsing is a form of bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the rest of the tokens to be parsed.

➢ We use $ to mark the bottom of the stack and also the end of the input.

➢ During a left-to-right scan of the input tokens, the parser shifts zero or more input tokens into the stack, until it is ready to reduce a string $\beta$ of grammar symbols on top of the stack.
E → E+T | T
T → T*F | F
F → (E) | id

Right-Most Derivation of \( \text{id+id*id} \)

\[
E \Rightarrow E+T \Rightarrow E+T*F \Rightarrow E+T*\text{id} \Rightarrow E+F*\text{id}
\]

\[
\Rightarrow E+\text{id*id} \Rightarrow T+\text{id*id} \Rightarrow F+\text{id*id} \Rightarrow \text{id+id*id}
\]

Right-Most Sentential Form
id+id*id
F+id*id
T+id*id
E+id*id
E+F*id
E+T*id
E+T*F
E+T
E

Reducing Production
F → id
T → F
E → T
F → id
T → F
F → id
T → T*F
E → E+T

Handles are red and underlined in the right-sentential forms
<table>
<thead>
<tr>
<th>STACK</th>
<th>INPUT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id₁ * id₂ $</td>
<td>shift</td>
</tr>
<tr>
<td>$ id₁</td>
<td>* id₂ $</td>
<td>reduce by $F \rightarrow id$</td>
</tr>
<tr>
<td>$ F</td>
<td>* id₂ $</td>
<td>reduce by $T \rightarrow F$</td>
</tr>
<tr>
<td>$ T</td>
<td>* id₂ $</td>
<td>shift</td>
</tr>
<tr>
<td>$ T *</td>
<td>id₂ $</td>
<td>shift</td>
</tr>
<tr>
<td>$ T * id₂</td>
<td>$</td>
<td>reduce by $F \rightarrow id$</td>
</tr>
<tr>
<td>$ T * F</td>
<td>$</td>
<td>reduce by $T \rightarrow T * F$</td>
</tr>
<tr>
<td>$ T</td>
<td>$</td>
<td>reduce by $E \rightarrow T$</td>
</tr>
<tr>
<td>$ E</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
A Stack Implementation of A Shift-Reduce Parser

- There are four possible actions of a shift-parser action:

  1. **Shift**: The next input symbol is shifted onto the top of the stack.
  2. **Reduce**: Replace the handle on the top of the stack by the non-terminal.
  3. **Accept**: Successful completion of parsing.
  4. **Error**: Parser discovers a syntax error, and calls an error recovery routine.

- Initial stack just contains only the end-marker $.

- The end of the input string is marked by the end-marker $.
A Stack Implementation of A Shift-Reduce Parser

- Two problems:
  - locate a handle and
  - decide which production to use (if there are more than two candidate productions).

- General Construction: using a stack:
  - “shift” input symbols into the stack until a handle is found on top of it.
  - “reduce” the handle to the corresponding non-terminal.

- other operations:
  - “accept” when the input is consumed and only the start symbol is on the stack, also: “error”
**A Stack Implementation of A Shift-Reduce Parser**

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id+id*id$</td>
<td>shift</td>
</tr>
<tr>
<td>$id$</td>
<td>+id*id$</td>
<td>reduce by $F \rightarrow$ id</td>
</tr>
<tr>
<td>$F$</td>
<td>+id*id$</td>
<td>reduce by $T \rightarrow$ F</td>
</tr>
<tr>
<td>$T$</td>
<td>+id*id$</td>
<td>reduce by $E \rightarrow$ T</td>
</tr>
<tr>
<td>$E$</td>
<td>+id*id$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+$</td>
<td>id*id$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+id$</td>
<td>*id$</td>
<td>reduce by $F \rightarrow$ id</td>
</tr>
<tr>
<td>$E+F$</td>
<td>*id$</td>
<td>reduce by $T \rightarrow$ F</td>
</tr>
<tr>
<td>$E+T$</td>
<td>*id$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+T*$</td>
<td>id$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+T*id$</td>
<td>$</td>
<td>reduce by $F \rightarrow$ id</td>
</tr>
<tr>
<td>$E+T*F$</td>
<td>$</td>
<td>reduce by $T \rightarrow$ T*F</td>
</tr>
<tr>
<td>$E+T$</td>
<td>$</td>
<td>reduce by $E \rightarrow$ E+T</td>
</tr>
<tr>
<td>$E$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Exercise

Exercise 1

Exercise 2

Exercise 3
There are two main categories of shift-reduce parsers

1. Operator-Precedence Parser
   - simple, but only a small class of grammars.

2. LR-Parsers
   - covers wide range of grammars.
     - SLR – simple LR parser
     - Canonical LR – most general LR parser
     - LALR – intermediate LR parser (lookahead LR parser)
   - SLR, Canonical LR and LALR work same, only their parsing tables are different.