Syntax Analysis
Part IV

Chapter 4: Bottom-Up Parsing

Slides adapted from:
© Robert van Engelen, Florida State University
Bottom-Up Parsing

- **LR methods** (Left-to-right, Rightmost derivation)
  - SLR, Canonical LR, LALR

- Other cases:
  - Shift-reduce parsing
  - Operator-precedence parsing
Bottom-Up Parsing

• The process of “reducing” an input string $w$ to the start symbol of the grammar

• At each reduction step, a specific substring matching the right-hand side of a production is replaced by the nonterminal at the left-hand side of that production
Bottom-Up Parsing

• Grammar

\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T \ast F \mid F \]
\[ F \rightarrow (E) \mid \text{id} \]

• Rightmost derivation

\[ E \Rightarrow_{rm} T \Rightarrow_{rm} T \ast F \Rightarrow_{rm} T \ast \text{id} \]
\[ \Rightarrow_{rm} F \ast \text{id} \Rightarrow_{rm} \text{id} \ast \text{id} \]
Bottom-Up Parsing

\[
E \Rightarrow_{rm} T \Rightarrow_{rm} T \ast F \Rightarrow_{rm} T \ast id \Rightarrow_{rm} F \ast id \Rightarrow_{rm} id \ast id \Rightarrow_{rm} id
\]
Handle Pruning

• A rightmost derivation in reverse can be obtained by “handle-pruning”

• A handle is a substring of grammar symbols that matches a right-hand side of a production in a rightmost sentential form
Handle Pruning

• Locate the handle and replace by the left-hand side of the production

\[ S \]

\[ \alpha \quad \beta \quad x \]

Step in a rightmost derivation
Handle Pruning

Grammar:
\[ S \rightarrow a \ A \ B \ e \]
\[ A \rightarrow A \ b \ c \mid b \]
\[ B \rightarrow d \]

Reducing a sentence:
\[ a \ b \ b \ c \ d \ e \]
\[ \underline{a \ A \ b \ c \ d \ e} \]
\[ \underline{a \ A \ d \ e} \]
\[ \underline{a \ A \ B \ e} \]
\[ S \]

Shift-reduce corresponds to a rightmost derivation:
\[ S \Rightarrow_{rm} a \ A \ B \ e \]
\[ \Rightarrow_{rm} a \ A \ d \ e \]
\[ \Rightarrow_{rm} a \ A \ b \ c \ d \ e \]
\[ \Rightarrow_{rm} a \ b \ b \ c \ d \ e \]

Underlined substrings are handles
Handle Pruning

Grammar:

\[
S \rightarrow a \ A \ B \ e \\
A \rightarrow A \ b \ c \mid b \\
B \rightarrow d \\
\]

\[
a \ b \ b \ c \ d \ e \\
a \ A \ b \ c \ d \ e \\
a \ A \ d \ e \\
a \ A \ B \ e \\
S
\]

\[
\]

NOT a handle, because further reductions will fail

(result is not a sentential form)
Exercise

• Consider the grammar:

\[
E \rightarrow E' \mid E' + E \\
E' \rightarrow - E' \mid \text{id} \mid (E)
\]

• What is the correct series of reductions for the string \(-(\text{id} + \text{id}) + \text{id}\) ?
Exercise (cont’d)

-(id + id) + id
-(id + E’) + id
-(id + E) + id
-(E’ + E) + id
-(E) + id
-E’ + id
E’ + id
E’ + E’
E’ + E
E

\[
\begin{align*}
E & \rightarrow E’ \mid E’ + E \\
E’ & \rightarrow -E’ \mid \text{id} \mid (E)
\end{align*}
\]
Exercise (cont’d)

-\((\text{id} + \text{id}) + \text{id}\)
-\((-E' + \text{id}) + \text{id}\)
-\((-E' + E') + \text{id}\)
-\(-E' + E'\)
-\((E' + E') + E'\)
-\(-E' + E'\)
-\(E' + E'\)
-\(E' + E\)
-\(E' + E\)
-\(E\)

\[
\begin{align*}
E & \rightarrow E' \mid E' + E \\
E' & \rightarrow -E' \mid \text{id} \mid (E)
\end{align*}
\]
Exercise (cont’d)

- (id + id) + id
- (E' + id) + id
- (E' + E') + id
- (E' + E) + id
- (E) + id
- E' + id
E' + id
E' + E'
E' + E
E

E → E'  |  E' + E
E' → - E'  |  id  |  (E)
Shift-Reduce Parsing

- **Shift-reduce** is a family of bottom-up parsers based on handle pruning
- Use a stack data structure and four operations
  - **Shift**: move symbol from input to stack
  - **Reduce**: replace handle in stack with lhs
  - **Accept**
  - **Reject**
Shift-Reduce Parsing

Grammar:

$E \rightarrow E \, + \, E$
$E \rightarrow E \, * \, E$
$E \rightarrow ( \, E \, )$
$E \rightarrow \mathtt{id}$

Find handles to reduce

How to resolve conflicts?

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Next Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id+id*id$</td>
<td>shift</td>
</tr>
<tr>
<td>$id$</td>
<td>$</td>
<td>reduce $E \rightarrow \mathtt{id}$</td>
</tr>
<tr>
<td>$E$</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+$</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+id$</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+E$</td>
<td>$</td>
<td>reduce $E \rightarrow \mathtt{id}$</td>
</tr>
<tr>
<td>$E+E*$</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+E*id$</td>
<td>$</td>
<td>reduce $E \rightarrow \mathtt{id}$</td>
</tr>
<tr>
<td>$E+E*E$</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>$E+E$</td>
<td>$</td>
<td>reduce $E \rightarrow E , * , E$</td>
</tr>
<tr>
<td>$E$</td>
<td>$</td>
<td>reduce $E \rightarrow E , + , E$</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>accept</td>
</tr>
</tbody>
</table>
Exercise

• Consider the grammar:

\[ E \rightarrow E' \mid E' + E \]
\[ E' \rightarrow - E' \mid \text{id} \mid (E) \]

• What is the correct shift-reduce parse for the string \text{id + -id}?
Exercise (cont’d)

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id+-id$</td>
</tr>
<tr>
<td>$id</td>
<td>+-id$</td>
</tr>
<tr>
<td>$E'+</td>
<td>-id$</td>
</tr>
<tr>
<td>$E'+-$</td>
<td>id$</td>
</tr>
<tr>
<td>$E'+-id$</td>
<td>$</td>
</tr>
<tr>
<td>$E'+-E'$</td>
<td>$</td>
</tr>
<tr>
<td>$E'+-E'$</td>
<td>$</td>
</tr>
<tr>
<td>$E'+E'$</td>
<td>$</td>
</tr>
<tr>
<td>$E'+E$</td>
<td>$</td>
</tr>
<tr>
<td>$E$</td>
<td>$</td>
</tr>
</tbody>
</table>

\[
E \to E' \mid E' + E \\
E' \to -E' \mid \text{id} \mid (E)
\]
Exercise (cont’d)

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id+-id$</td>
</tr>
<tr>
<td>$id</td>
<td>+-id$</td>
</tr>
<tr>
<td>$id+$</td>
<td>-id$</td>
</tr>
<tr>
<td>$id+-$</td>
<td>id$</td>
</tr>
<tr>
<td>$id+-id$</td>
<td>$</td>
</tr>
<tr>
<td>$id+-$</td>
<td>$</td>
</tr>
<tr>
<td>$id+-E’</td>
<td>$</td>
</tr>
<tr>
<td>$id+E’</td>
<td>$</td>
</tr>
<tr>
<td>$id+E</td>
<td>$</td>
</tr>
<tr>
<td>$E’+E</td>
<td>$</td>
</tr>
<tr>
<td>$E</td>
<td>$</td>
</tr>
</tbody>
</table>

$$
E \rightarrow E' \mid E' + E \\
E' \rightarrow - E' \mid \text{id} \mid (E)
$$
Exercise (cont’d)

\[
\begin{array}{|c|c|}
\hline
\text{Stack} & \text{Input} \\
\hline
$ & \text{id+-id}$ \\
$id & ++id$ \\
$E'$ & ++id$ \\
$E'$+ & -id$ \\
$E'$+- & id$ \\
$E'$+-id & $ \\
$E'$+-E' & $ \\
$E'$+E' & $ \\
$E'$+E & $ \\
E & $ \\
\hline
\end{array}
\]

\[
E \to E' \mid E'+E \\
E' \to -E' \mid \text{id} \mid (E)
\]
Exercise

• Consider the grammar:
  \[ E \rightarrow E' \mid E' + E \]
  \[ E' \rightarrow - E' \mid \text{id} \mid (E) \]

• Identify the handle for the shift-reduce parse state
  $E' + -$id$ \quad + -($id + id)$
  $E' + -$id$
  $-$id
  $ -$id
  $E' + -E'$
Handle and Stack

- The handle will always eventually appear at the top of the stack (not at a deeper position)
- Case 1: $B$ inside $A$

\[
\begin{align*}
S & \Rightarrow \alpha A z \\
S & \Rightarrow \alpha \beta B y z \\
S & \Rightarrow \alpha \beta \gamma y z
\end{align*}
\]
Handle and Stack

• The handle will always eventually appear at the top of the stack

• Case 2: $B$ and $A$ on different branches

\[
S \Rightarrow \alpha B x A z \\
\Rightarrow \alpha B x y z \\
\Rightarrow \alpha \gamma x y z
\]
Conflicts

- **Shift-reduce** and **reduce-reduce conflicts** are caused by
  - Ambiguity of the grammar
  - The limitations of the parsing method (even when the grammar is unambiguous)
### Shift-Reduce Conflicts

**Ambiguous grammar:**

\[ S \rightarrow \text{if } E \text{ then } S \]
\[ \text{if } E \text{ then } S \text{ else } S \]
\[ \text{other} \]

Resolve in favor of shift, so **else** matches closest **if**

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Next Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ldots$</td>
<td>$\ldots$ if $E$ then $S$</td>
<td>shift or reduce?</td>
</tr>
<tr>
<td>$\ldots$ else $\ldots$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reduce-Reduce Conflicts

Grammar:

\[ C \rightarrow A B \]
\[ A \rightarrow a \]
\[ B \rightarrow a \]

Resolve in favor of reduce \( A \rightarrow a \), otherwise we are stuck!

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a$a$</td>
<td>shift</td>
</tr>
</tbody>
</table>
| $a    | a$    | reduce \( A \rightarrow a \) or \( B \rightarrow a \)?
Rightmost Derivations

• Handle recognition is related to the recognition of rightmost sentential forms
• We need to develop a formal model for the above language
Rightmost Derivations

Grammar:
\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T \ast F \mid F \]
\[ F \rightarrow (E) \mid \text{id} \]

Derivation:
\[ E \Rightarrow_{\text{rm}} E + T \]
Rightmost Derivations

Grammar:

\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T \ast F \mid F \]
\[ F \rightarrow (E) \mid \text{id} \]

Derivation:

\[ E \Rightarrow_{rm} E + T \]
\[ \Rightarrow_{rm} E + T \ast F \]
Rightmost Derivations

Grammar:
\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T * F \mid F \]
\[ F \rightarrow (E) \mid \text{id} \]

Derivation:
\[ E \Rightarrow_{rm} E + T \]
\[ \Rightarrow_{rm} E + T * F \]
\[ \Rightarrow_{rm} E + T * \text{id} \]
Rightmost Derivations

Grammar:

\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T * F \mid F \]
\[ F \rightarrow (E) \mid \text{id} \]

Derivation:

\[ E \Rightarrow_{rm} E + T \]
\[ \Rightarrow_{rm} E + T * F \]
\[ \Rightarrow^{*}_{rm} E + \text{id} * \text{id} \]
Rightmost Derivations

Grammar:

\[
E \rightarrow E + T \mid T \\
T \rightarrow T * F \mid F \\
F \rightarrow (E) \mid \text{id}
\]

Derivation:

\[
E \Rightarrow_{rm} E + T \\
\Rightarrow^* E + \text{id} \ast \text{id} \\
\Rightarrow_{rm} E + \text{id} + \text{id} \ast \text{id}
\]
Rightmost Derivations

- In order to “cut” a derivation tree into a rightmost sentential form, iterate the following steps
  - Vertically descend one step
  - Move right horizontally zero or more steps within the current rule
  - Stop when the right corner of the tree is reached
Viable Prefixes

- A **viable prefix** is the prefix of a rightmost sentential form, up to the part of terminal symbols already derived.
- We can show that a viable prefix is the tree cut obtained through the vertical-horizontal procedure, up to the point in which the yield of the tree is reached (proof omitted).
Viable Prefixes

viable prefix

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

already derived terminals
Viable Prefixes

• A viable prefix can always be decomposed into the concatenation of prefixes of right-hand sides of productions
Viable Prefixes

Viable prefix:

\[ E + T^* \]

Concatenation of prefixes of right-hand sides in the cut
Viable Prefixes

• A viable prefix can be decomposed into prefixes of right-hand sides in more than one way
Viable Prefixes

Viable prefix:

A B C D
Viable Prefixes

• For any CFG $G$, the set of all possible viable prefixes of $G$ is a regular language (proof omitted)

• We provide the construction of a NFA for the viable prefixes of $G$
LR(0) Items

• An **LR(0) item** is any production of $G$ with a dot at some position in the right-hand side.

• Thus, a production $A \rightarrow X Y Z$ has four items:

  $[A \rightarrow \bullet X Y Z] \quad [A \rightarrow X \bullet Y Z]$

  $[A \rightarrow X Y \bullet Z] \quad [A \rightarrow X Y Z \bullet]$

• Production $A \rightarrow \varepsilon$ has a single item $[A \rightarrow \bullet]$
LR(0) Items

• The bullet in an item \([A \rightarrow \alpha \cdot \beta]\) marks the portion \(\alpha\) of the right-hand side that has already been horizontally processed

• **Convention**: We add to the grammar a new starting symbol \(S'\), and a production of the form \(S' \rightarrow S\)
NFA for Viable Prefixes

• Input alphabet is the set of nonterminal symbols of $G$ union the set of terminal symbols of $G$
• State set is the set of LR(0) items
• Initial state: $[S' \rightarrow \bullet S]$
• Every state is also a final state
• NFA is partial
NFA for Viable Prefixes

• For each item \([A \rightarrow \alpha \cdot X \beta]\) add the following \textit{transitions}, represented by relation \(\vdash\)
  
  – For each nonterminal/terminal symbol \(X\)
    \(\vdash_X [A \rightarrow \alpha \cdot X \beta] \vdash [A \rightarrow \alpha \cdot X \beta]\)
  
  – For each production \(X \rightarrow \gamma\) in \(G\)
    \(\vdash_\varepsilon [A \rightarrow \alpha \cdot X \beta] \vdash_\varepsilon [X \rightarrow \cdot \gamma]\)
NFA for Viable Prefixes

• Transitions on nonterminal/terminal symbol $X$ are used to simulate **horizontal** part of the tree cut

• Transitions on symbol $\varepsilon$ are used to simulate **vertical** part of the tree cut
NFA : Example

Augmented grammar :
\[ S' \rightarrow E \]
\[ E \rightarrow T + E \mid T \]
\[ T \rightarrow \text{int} \ast T \mid \text{int} \mid (E) \]
NFA : Example

Augmented grammar :

\[ S' \rightarrow E \]

\[ E \rightarrow T + E \mid T \]

\[ T \rightarrow \text{int} \ast T \mid \text{int} \mid (E) \]
NFA : Example

Augmented grammar:

\[ S' \rightarrow E \]
\[ E \rightarrow T + E \quad \mid \quad T \]
\[ T \rightarrow \text{int} \ast T \quad \mid \quad \text{int} \quad \mid \quad (E) \]
NFA : Example

Augmented grammar:

\[
S' \rightarrow E \\
E \rightarrow T + E \mid T \\
T \rightarrow \text{int} \ast T \mid \text{int} \mid (E)
\]
NFA: Example

Augmented grammar:

\[ S' \rightarrow E \]
\[ E \rightarrow T + E \mid T \]
\[ T \rightarrow \text{int} \times T \mid \text{int} \mid (E) \]
Augmented grammar:

\[ S' \rightarrow E \]
\[ E \rightarrow T + E \mid T \]
\[ T \rightarrow \text{int} \ast T \mid \text{int} \mid (E) \]
NFA : Example

Augmented grammar:

\[
\begin{align*}
S' & \rightarrow E \\
E & \rightarrow T + E \\
T & \rightarrow \text{int} * T \\
& \quad \mid \text{int} \\
& \quad \mid (E)
\end{align*}
\]
DFA for Viable Prefixes

- The NFA for viable prefixes can be made **deterministic**
- Each state of the resulting DFA is a **set** of LR(0) items
DFA for Viable Prefixes

- Each path through the DFA represents several ways of cutting a viable prefix into prefixes of rule right-hand sides.
- If item \([X \rightarrow \gamma \bullet]\) belongs to a state of the DFA, we have found a handle.
LR Parser

- The LR parser uses the DFA for viable prefixes to detect handles
- The DFA is run on the stack of the LR parser
- To avoid reading the full stack at each step, we save DFA states for each symbol in the stack