CS412/CS413

Introduction to Compilers
Tim Teitelbaum

Lecture 8: LR parsing
February 6, 2008
Bottom-up Parsing

• A more powerful parsing technology

• LR grammars -- more expressive than LL
  – Scan the input from Left to right and determine a Right-most derivation of program (backwards)
  – Allows left-recursive grammars, virtually all programming languages
  – Easier to express programming language syntax

• Shift-reduce parsers
  – Parsers for LR grammars
  – Automatic parser generators (e.g., yacc, CUP)
Bottom-up Parsing

• Right-most derivation -- backward
  - Start with the tokens; end with the start symbol

\[
\begin{align*}
(1+2+(3+4))+5 & \leftarrow \\
(E+2+(3+4))+5 & \leftarrow \\
(S+2+(3+4))+5 & \leftarrow \\
(S+E+(3+4))+5 & \leftarrow \\
(S+(3+4))+5 & \leftarrow \\
(S+(E+4))+5 & \leftarrow \\
(S+(S+4))+5 & \leftarrow \\
(S+(S+E))+5 & \leftarrow \\
(S+(S))+5 & \leftarrow \\
(S+E)+5 & \leftarrow \\
(S)+5 & \leftarrow \\
E+5 & \leftarrow \\
S+5 & \leftarrow \\
S+E & \leftarrow \\
S & \leftarrow
\end{align*}
\]

\[
S \rightarrow S + E | E \\
E \rightarrow \text{num | ( S )}
\]
Progress of Bottom-up Parsing

(1+2+(3+4))+5 ⇐ (1+2+(3+4))+5
(E+2+(3+4))+5 ⇐ (1+2+(3+4))+5
(S+2+(3+4))+5 ⇐ (1+2+(3+4))+5
(S+E+(3+4))+5 ⇐ (1+2+(3+4))+5
(S+(3+4))+5 ⇐ (1+2+(3+4))+5
(S+(E+4))+5 ⇐ (1+2+(3+4))+5
(S+(S+4))+5 ⇐ (1+2+(3+4))+5
(S+(S+E))+5 ⇐ (1+2+(3+4))+5
(S+(S))+5 ⇐ (1+2+(3+4))+5
(S+E)+5 ⇐ (1+2+(3+4))+5
(S)+5 ⇐ (1+2+(3+4))+5
E+5 ⇐ (1+2+(3+4))+5
S+E ⇐ (1+2+(3+4))+5
S ⇐ (1+2+(3+4))+5
Bottom-up Parsing

• $(1+2+(3+4))+5 \iff (E+2+(3+4))+5 \iff (S+2+(3+4))+5 \iff (S+E+(3+4))+5 \ldots$

• If $S \Rightarrow^* \alpha Aw \Rightarrow \alpha \beta w$ then $\beta$ is called a handle of $\alpha \beta w$

• Advantage of bottom-up parsing: the selection of production is postponed until after the handle has been scanned
Top-down Parsing

\[(1+2+(3+4))+5\]

\[S \Rightarrow S+E \Rightarrow E+E \Rightarrow (S)+E \Rightarrow (S+E)+E\]
\[\Rightarrow (S+E+E)+E \Rightarrow (E+E+E)+E\]
\[\Rightarrow (1+E+E)+E \Rightarrow (1+2+E)+E \ldots\]

- In left-most derivation, entire tree above a token (2) has been expanded when it is encountered.
Top-down vs. Bottom-up

**Bottom-up:** Don’t need to figure out as much of the parse tree for a given amount of input.
Shift-reduce Parsing

- Parsing actions: a sequence of shift and reduce operations
- Parser state: a stack of terminals and non-terminals (grows to the right)
- Current derivation is always stack+input

<table>
<thead>
<tr>
<th>Derivation step</th>
<th>stack</th>
<th>unconsumed input</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1+2+(3+4))+5 ⇐</td>
<td>(1+2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1  1+2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1   +2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td>(E+2+(3+4))+5 ⇐</td>
<td>(E   +2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td>(S+2+(3+4))+5 ⇐</td>
<td>(S   +2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(S+   2+(3+4))+5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(S+E  +(3+4))+5</td>
<td></td>
</tr>
<tr>
<td>(S+E+(3+4))+5 ⇐</td>
<td>(S+E  +(3+4))+5</td>
<td></td>
</tr>
</tbody>
</table>
Shift-reduce Parsing

- Parsing is a sequence of shifts and reduces
- **Shift**: move look-ahead token to stack
  
<table>
<thead>
<tr>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>1+2+(3+4))+5</td>
<td>shift 1</td>
</tr>
<tr>
<td>1+2</td>
<td>+2+(3+4))+5</td>
<td></td>
</tr>
</tbody>
</table>

- **Reduce**: Replace symbols β from top of stack with non-terminal symbol A, corresponding to production $A \rightarrow \beta$ (pop β, push A)
  
<table>
<thead>
<tr>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S+E)</td>
<td>+(3+4))+5</td>
<td>reduce S→ S+E</td>
</tr>
<tr>
<td>(S)</td>
<td>+(3+4))+5</td>
<td></td>
</tr>
</tbody>
</table>
## Shift-reduce Parsing

### Grammar Rules

- **S** → **S** + **E** | **E**
- **E** → num | ( **S** )

### Derivation

<table>
<thead>
<tr>
<th>Input Stream</th>
<th>Stack</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1+2+(3+4))+5</td>
<td>(1+2+(3+4))+5</td>
<td>shift</td>
</tr>
<tr>
<td>(1+2+(3+4))+5</td>
<td>(1+2+(3+4))+5</td>
<td>reduce E → num</td>
</tr>
<tr>
<td>(E+2+(3+4))+5</td>
<td>(E +2+(3+4))+5</td>
<td>reduce S → E</td>
</tr>
<tr>
<td>(S+2+(3+4))+5</td>
<td>(S +2+(3+4))+5</td>
<td>shift</td>
</tr>
<tr>
<td>(S+2+(3+4))+5</td>
<td>(S+ 2+(3+4))+5</td>
<td>reduce E → num</td>
</tr>
<tr>
<td>(S+E+(3+4))+5</td>
<td>(S+E +(3+4))+5</td>
<td>reduce S → S+E</td>
</tr>
<tr>
<td>(S+(3+4))+5</td>
<td>(S +(3+4))+5</td>
<td>shift</td>
</tr>
<tr>
<td>(S+(3+4))+5</td>
<td>(S+ (3+4))+5</td>
<td>shift</td>
</tr>
<tr>
<td>(S+(3+4))+5</td>
<td>(S+( 3+4))+5</td>
<td>shift</td>
</tr>
<tr>
<td>(S+(3+4))+5</td>
<td>(S+(3 +4))+5</td>
<td>reduce E → num</td>
</tr>
</tbody>
</table>
Problem

- How do we know which action to take: whether to shift or reduce, and which production?

- Issues:
  - Sometimes can reduce but shouldn’t
  - Sometimes can reduce in different ways
Action Selection Problem

- Given stack $\sigma$ and look-ahead symbol $b$, should parser:
  - **shift** $b$ onto the stack (making it $\sigma b$)
  - **reduce** $A \rightarrow \beta$ assuming that stack has the form $\alpha \beta$ (making it $\alpha A$)
LR Parsing Engine

• Basic mechanism:
  – Use a set of parser states
  – Use a stack of states
  – Use a parsing table to:
    • Determine what action to apply (shift/reduce)
    • Determine the next state

• The parser actions can be precisely determined from the table
# The LR Parsing Table

<table>
<thead>
<tr>
<th>State</th>
<th>Action to take and next state to enter</th>
<th>Non-terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Action table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Goto table</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminals $\cup {\varepsilon}$</th>
<th>Next state</th>
</tr>
</thead>
</table>

- **Algorithm:** Look at entry for current state $Q$ and input terminal $c$
  
  If $\text{Table}[Q,c] = \text{shift}(Q')$ then **shift:**
  
  $\text{push}(Q')$

  If $\text{Table}[Q,c] = A \rightarrow \alpha$ then **reduce:**
  
  $\text{pop}(|\alpha|); \ Q'=\text{top}(); \ \text{push}(\text{Table}[Q',A])$
# LR(1) Parsing Table Example

<table>
<thead>
<tr>
<th></th>
<th>( )</th>
<th>id</th>
<th>,</th>
<th>ε</th>
<th>S</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>s3</td>
<td>s2</td>
<td></td>
<td>g4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s3</td>
<td>s2</td>
<td>s2</td>
<td></td>
<td>g7</td>
<td>g5</td>
</tr>
<tr>
<td>3</td>
<td>S→id</td>
<td>S→id</td>
<td>S→id</td>
<td>S→id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>s3</td>
<td>s2</td>
<td></td>
<td>g9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td></td>
</tr>
</tbody>
</table>
LR(k) Grammars

- **LR(k)** = Left-to-right scanning, Right-most derivation, *k* look-ahead characters
- Main cases: LR(0), LR(1), and some variations (SLR and LALR(1))
- Parsers for **LR(0)** Grammars:
  - Determine the actions without any lookahead symbol
  - Will help us understand shift-reduce parsing
- Read: **CUP User Manual**