Bottom-up Parsing

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University
• Fourth homework due Sunday, March 10, 11:59pm. Will post sample solution on Monday, March 11.

• First project new due dates: CODE Thursday March 7; REPORT Saturday March 9; benchmark programs have been posted.
  ~uli/cs415/projects/proj1/benchmarks

Late submission policy: 20% penalty for every started 24 hours period late; weekend counts as a single 24 hour period. Examples: Submit code on Friday at 9:30pm, 20% penalty; submit on Sunday before 11:59pm, 40% penalty.

• Midterm: Wednesday, March 13, in class; closed book, closed notes, 80 minutes
Bottom-up Parsing
(Syntax Analysis)

EAC Chapters 3.4
**LR(1), operator precedence**

1 input symbol lookahead
construct rightmost derivation (backwards)
input: read left-to-right

**Rule:** \( B ::= γ \)

\[ S \Rightarrow^*_{rm} αB \Rightarrow_{rm} αγ \Rightarrow^*_{rm} x \ y \]

\[ S \]

\[ α \]

\[ γ \]

? Means that we don’t know yet this part of the parse tree

\[ x \ y \]
LR(1), operator precedence

1 input symbol lookahead
construct rightmost derivation (backwards)
input: read left-to-right

rule \( B ::= \gamma \)

\[
S \Rightarrow_{rm}^* \alpha \ B \ y \Rightarrow_{rm} \ alpha \ \gamma \ y \Rightarrow_{rm}^* \ x \ y
\]

upper fringe

? Means that we don’t know yet this part of the parse tree
Is the following grammar LL(1), L(2), or LR(1)?

\[ S ::= a \ b \mid a \ b \ c \]

Is the following grammar LR(1) or even LR(0)?

\[ S ::= a \ S \ b \mid c \]

Basic idea:

*shift* symbols from input onto the stack until top of the stack is a RHS of a rule; if so, “apply” rule backwards (*reduce*) by replacing top of the stack by the LHS non-terminal.

**Challenge:** When to shift, and when to reduce
Consider the simple grammar

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Goal</td>
<td>$\to$</td>
<td>a A B e</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>$\to$</td>
<td>A b c</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>$\to$</td>
<td>d</td>
</tr>
</tbody>
</table>

And the input string **abbcde**

<table>
<thead>
<tr>
<th>Sentential Form</th>
<th>Next Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prod’n</td>
</tr>
<tr>
<td>abbcde</td>
<td>3</td>
</tr>
<tr>
<td>a A bcde</td>
<td>2</td>
</tr>
<tr>
<td>a A de</td>
<td>4</td>
</tr>
<tr>
<td>a A B e</td>
<td>1</td>
</tr>
<tr>
<td>Goal</td>
<td>—</td>
</tr>
</tbody>
</table>

The trick is scanning the input and finding the next reduction. The mechanism for doing this must be efficient.
The parser must find a substring $\beta$ of the tree’s frontier that matches some production $A \rightarrow \beta$ that occurs as one step in the rightmost derivation.

Informally, we call this substring $\beta$ a handle.

Formally, a handle of a right-sentential form $\gamma$ is a pair $<A \rightarrow \beta, k>$ where $A \rightarrow \beta \in P$ and $k$ is the position in $\gamma$ of $\beta$’s rightmost symbol.

If $<A \rightarrow \beta, k>$ is a handle, then replacing $\beta$ at $k$ with $A$ produces the right sentential form from which $\gamma$ is derived in the rightmost derivation.

Because $\gamma$ is a right-sentential form, the substring to the right of a handle contains only terminal symbols.

$\Rightarrow$ the parser doesn’t need to scan past the handle (only lookahead).

$\Rightarrow$ The right end of the handle will be on top of the stack, not within the stack. Need lookahead to determine whether we reached the handle.
Critical Insight (Theorem)

If $G$ is unambiguous, then every right-sentential form has a unique handle.

If we can find those handles, we can build a derivation!

Sketch of Proof:

1. $G$ is unambiguous $\Rightarrow$ rightmost derivation is unique
2. $\Rightarrow$ a unique production $A \rightarrow \beta$ applied to derive $\gamma_i$ from $\gamma_{i-1}$
3. $\Rightarrow$ a unique position $k$ at which $A \rightarrow \beta$ is applied
4. $\Rightarrow$ a unique handle $\langle A \rightarrow \beta, k \rangle$

This all follows from the definitions
Revisit previous example

Consider the simple grammar

| 1 | Goal  | → | a A B e |
| 2 | A     | → | A b c   |
| 3 | l     | → | b       |
| 4 | B     | → | d       |

And the input string **abbcde**

<table>
<thead>
<tr>
<th>Sentential Form</th>
<th>Next Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbcde</td>
<td>Prod’n</td>
</tr>
<tr>
<td>a A bcde</td>
<td>3</td>
</tr>
<tr>
<td>a A de</td>
<td>2</td>
</tr>
<tr>
<td>a A B e Goal</td>
<td>4</td>
</tr>
<tr>
<td>Goal</td>
<td>1</td>
</tr>
</tbody>
</table>

The trick is scanning the input and finding the next reduction
The mechanism for doing this must be efficient
LR(0) items and LR(0) canonical collection

S0: \{[\text{Goal} \rightarrow \cdot a A B e]\}
S1: \{[\text{Goal} \rightarrow a \cdot A B e], [A \rightarrow \cdot A b c], [A \rightarrow \cdot b]\}
S2: \{[\text{Goal} \rightarrow a A \cdot B e], [A \rightarrow A \cdot b c], [B \rightarrow \cdot d] \}
S3: \{[A \rightarrow b \cdot] \}
S4: \{[\text{Goal} \rightarrow a A B \cdot e] \}
S5: \{[A \rightarrow A b \cdot c] \}
S6: \{[B \rightarrow d \cdot] \}
S7: \{[A \rightarrow A b c \cdot] \}
S8: \{[\text{Goal} \rightarrow a A B e \cdot] \}
More Syntax Analysis (bottom-up)

Review session for midterm