Lesson 9

- LR parsing with ambiguous grammars
- Error detection in LR parsing
- Some exercises on parsing
LL, SLR, LR, LALR Summary

- LL parse tables
  - Nonterminals × terminals → productions
  - Computed using FIRST/FOLLOW
- LR parsing tables computed using closure/goto
  - LR states × terminals → shift/reduce actions
  - LR states × nonterminals → goto state transitions
- A grammar is
  - LL(1) if its LL(1) parse table has no conflicts
  - SLR if its SLR parse table has no conflicts
  - LALR if its LALR parse table has no conflicts
  - LR(1) if its LR(1) parse table has no conflicts
LL, SLR, LR, LALR Grammars
Dealing with Ambiguous Grammars

1. $S' \rightarrow E$
2. $E \rightarrow E + E$
3. $E \rightarrow \text{id}$

<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th>+</th>
<th>$\rightarrow$</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s3</td>
<td>acc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s3/r2</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shift/reduce conflict:
- $\text{action}[4,+] = \text{shift 4}$
- $\text{action}[4,+] = \text{reduce } E \rightarrow E + E$

Stack: $\text{id+id+id}$
Input: $\text{id+(id+id)}$

When shifting on $+$:
yields right associativity

When reducing on $+$:
yields left associativity

(id+id)+id
Using Associativity and Precedence to Resolve Conflicts

- Left-associative operators: reduce
- Right-associative operators: shift
- Operator of higher precedence on stack: reduce
- Operator of lower precedence on stack: shift

\[
\begin{align*}
S' & \rightarrow E \\
E & \rightarrow E + E \\
E & \rightarrow E * E \\
E & \rightarrow id
\end{align*}
\]

Stack and input:

\[
\begin{array}{c|c}
\text{Stack} & \text{Input} \\
\hline
\$ 0 & \text{id\text{*id+id}} \\
\text{...} & \text{...} \\
\$ 0 \ E \ 1 \ * \ 3 \ E \ 5 & \text{id}$
\end{array}
\]

Reduce \( E \rightarrow E \ast E \)
Error Detection in LR Parsing

• Canonical LR parser uses full LR(1) parse tables and will never make a single reduction before recognizing the error when a syntax error occurs on the input

• SLR and LALR may still reduce when a syntax error occurs on the input, but will never shift the erroneous input symbol
Error Recovery in LR Parsing

• Panic mode
  – Pop until state with a goto on a nonterminal $A$ is found, (where $A$ represents a major programming construct), push $A$
  – Discard input symbols until one is found in the FOLLOW set of $A$

• Phrase-level recovery
  – Implement error routines for every error entry in table

• Error productions
  – Pop until state has error production, then shift on stack
  – Discard input until symbol is encountered that allows parsing to continue
Exercises onParsing

1. Contex-free Languages strictly include Regular Language
   – Prove it by showing that \( L = \{ a^n b^n \mid n > 0 \} \) is context-free but not regular

2. Consider the grammar:
   \[
   \begin{align*}
   A & \rightarrow aB \mid BC \\
   B & \rightarrow bB \mid \varepsilon \\
   C & \rightarrow c
   \end{align*}
   \]
   – Construct the LL(1) parsing table
   – Show configurations of stack and input recognizing strings \( bbc, \ abb \)
Exercises on Parsing

• Consider the grammar augmented with a new start symbol $S'$ and production $S' \rightarrow S$:
  ① $S' \rightarrow S$
  ② $S \rightarrow A B$
  ③ $A \rightarrow A a$
  ④ $A \rightarrow \epsilon$
  ⑤ $B \rightarrow b C$
  ⑥ $C \rightarrow c$

a) Construct the LR(0) sets of items.
b) Construct the SLR parsing table from the LR(0) items.
c) Is the grammar LR(0)? Is it SLR?