Introduction

- Shift/Reduce Conflict
- Error Recovery in LR Parsing
Shift/Reduce Conflict

- We say that we cannot introduce a shift/reduce conflict during the shrink process for the creation of the states of a LALR parser.
- Assume that we can introduce a shift/reduce conflict. In this case, a state of LALR parser must have:
  \[ A \rightarrow \alpha, a \] and \[ B \rightarrow \beta, a\gamma, b \]
- This means that a state of the canonical LR(1) parser must have:
  \[ A \rightarrow \alpha, a \] and \[ B \rightarrow \beta, a\gamma, c \]

But, this state has also a shift/reduce conflict. i.e. The original canonical LR(1) parser has a conflict.
(Reason for this, the shift operation does not depend on lookaheads)
Reduce/Reduce Conflict

- But, we may introduce a reduce/reduce conflict during the shrink process for the creation of the states of a LALR parser.

\[
\begin{align*}
I_1 : & \ A \rightarrow \alpha \cdot, a \\
& \ B \rightarrow \beta \cdot, b \\
\downarrow \\
I_{12} : & \ A \rightarrow \alpha \cdot, \{a, b\} \quad \text{⇒ reduce/reduce conflict} \\
& \ B \rightarrow \beta \cdot, \{b, c\}
\end{align*}
\]
Canonical LALR(1) Collection - Example2

S' → S   I_0: S' → • S, $
1) S → L=R   S → • L=R,$
2) S → R    S → • R,$
3) L → *R   L → • *R, {$, =}$
4) L → id   L → • id, {$, =}$
5) R → L    R → • L,$

I_6: S → L= • R,$
R → • L,$
L → • *R,$
L → • id,$

I_7:3: S → R •,$
R → • R,$
L → • id,$

I_8:10: R → • L,$
L → • id,$

I_9: S → L=R •,$

Same Cores
I_4 and I_11
I_5 and I_12
I_7 and I_13
I_8 and I_10
LALR(1) Parsing Tables - (for Example2)

<table>
<thead>
<tr>
<th>id</th>
<th>*</th>
<th>=</th>
<th>$</th>
<th>S</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s5</td>
<td>s4</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>s6</td>
<td>r5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td>s4</td>
<td></td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s12</td>
<td>s11</td>
<td></td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>r5</td>
<td>r5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

no shift/reduce or no reduce/reduce conflict

so, it is a LALR(1) grammar
Using Ambiguous Grammars

• All grammars used in the construction of LR-parsing tables must be un-ambiguous.
• Can we create LR-parsing tables for ambiguous grammars?
  ▫ Yes, but they will have conflicts.
  ▫ We can resolve these conflicts in favor of one of them to disambiguate the grammar.
  ▫ At the end, we will have again an unambiguous grammar.
• Why we want to use an ambiguous grammar?
  ▫ Some of the ambiguous grammars are much natural, and a corresponding unambiguous grammar can be very complex.
  ▫ Usage of an ambiguous grammar may eliminate unnecessary reductions.
• Ex.

\[
E \rightarrow E+E | E*E | (E) | \text{id} \quad \Rightarrow \quad E \rightarrow E+T \mid T \\
T \rightarrow T*F \mid F \\
F \rightarrow (E) \mid \text{id}
\]
Sets of LR(0) Items for Ambiguous Grammar

I₀: $E' \rightarrow \cdot E$
   $E \rightarrow \cdot E+E$
   $E \rightarrow \cdot E*E$
   $E \rightarrow \cdot (E)$
   $E \rightarrow \cdot id$

I₁: $E' \rightarrow E \cdot$
   $E \rightarrow E \cdot +E$
   $E \rightarrow E \cdot *E$

I₂: $E \rightarrow (\cdot E)$
   $E \rightarrow \cdot E+E$
   $E \rightarrow \cdot E*E$
   $E \rightarrow \cdot (E)$
   $E \rightarrow \cdot id$

I₃: $E \rightarrow id \cdot$

I₄: $E \rightarrow E + \cdot E$
   $E \rightarrow \cdot E+E$
   $E \rightarrow \cdot E*E$
   $E \rightarrow \cdot (E)$
   $E \rightarrow \cdot id$

I₅: $E \rightarrow E \cdot \cdot E$
   $E \rightarrow \cdot E+E$
   $E \rightarrow \cdot E*E$
   $E \rightarrow \cdot (E)$
   $E \rightarrow \cdot id$

I₆: $E \rightarrow (E \cdot)$
   $E \rightarrow E \cdot +E$
   $E \rightarrow E \cdot *E$

I₇: $E \rightarrow E+E \cdot$
   $E \rightarrow \cdot E+E$
   $E \rightarrow \cdot E*E$

I₈: $E \rightarrow E*E \cdot$
   $E \rightarrow E \cdot +E$
   $E \rightarrow E \cdot *E$

I₉: $E \rightarrow (E) \cdot$
SLR-Parsing Tables for Ambiguous Grammar

FOLLOW(E) = \{ $, +, *, ) \} \\

State I_7 has shift/reduce conflicts for symbols + and *.

\[ I_0 \xrightarrow{E} I_1 \xrightarrow{+} I_4 \xrightarrow{E} I_7 \]

when current token is +
  shift \(\Rightarrow\) + is right-associative
  reduce \(\Rightarrow\) + is left-associative

when current token is *
  shift \(\Rightarrow\) * has higher precedence than +
  reduce \(\Rightarrow\) + has higher precedence than *
SLR-Parsing Tables for Ambiguous Grammar

FOLLOW(E) = \{ $, +, *, ) \} 

State $I_8$ has shift/reduce conflicts for symbols + and *.

$\begin{align*}
I_0 & \xrightarrow{E} I_1 \xrightarrow{*} I_5 \xrightarrow{E} I_8 \\
\end{align*}$

when current token is *

- shift $\Rightarrow$ * is right-associative
- reduce $\Rightarrow$ * is left-associative

when current token is +

- shift $\Rightarrow$ + has higher precedence than *
- reduce $\Rightarrow$ * has higher precedence than +
### SLR-Parsing Tables for Ambiguous Grammar

<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>s4</td>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>r4</td>
<td>r4</td>
<td></td>
<td>r4</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>s3</td>
<td></td>
<td></td>
<td>s2</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>s4</td>
<td>s5</td>
<td></td>
<td></td>
<td>s9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>r1</td>
<td>s5</td>
<td></td>
<td></td>
<td>r1</td>
<td>r1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td>r2</td>
<td>r2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td>r3</td>
<td>r3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **id:** Identifier
- **+:** Addition
- ***:** Multiplication
- **(:** Opening parenthesis
- **)**: Closing parenthesis
- **$**: End of input
- **E**: Start symbol
Error Recovery in LR Parsing

• An LR parser will detect an error when it consults the parsing action table and finds an error entry. All empty entries in the action table are error entries.
• Errors are never detected by consulting the goto table.
• An LR parser will announce error as soon as there is no valid continuation for the scanned portion of the input.
• A canonical LR parser (LR(1) parser) will never make even a single reduction before announcing an error.
• The SLR and LALR parsers may make several reductions before announcing an error.
• But, all LR parsers (LR(1), LALR and SLR parsers) will never shift an erroneous input symbol onto the stack.
Panic Mode Error Recovery in LR Parsing

• Scan down the stack until a state $s$ with a goto on a particular nonterminal $A$ is found. (Get rid of everything from the stack before this state $s$).
• Discard zero or more input symbols until a symbol $a$ is found that can legitimately follow $A$.
  ▫ The symbol $a$ is simply in $\text{FOLLOW}(A)$, but this may not work for all situations.
• The parser stacks the nonterminal $A$ and the state $\text{goto}[s,A]$, and it resumes the normal parsing.
• This nonterminal $A$ is normally is a basic programming block (there can be more than one choice for $A$).
  ▫ $\text{stmt, expr, block, ...}$
Phrase-Level Error Recovery in LR Parsing

- Each empty entry in the action table is marked with a specific error routine.
- An error routine reflects the error that the user most likely will make in that case.
- An error routine inserts the symbols into the stack or the input (or it deletes the symbols from the stack and the input, or it can do both insertion and deletion).
  - missing operand
  - unbalanced right parenthesis