CS453
LR(1), LALR, AMBIGUITY
**LR(1), LALR, Ambiguity**

The plan:

Shift reduce parsing
- LR(1) and LALR
- Ambiguous grammars
  - Precedence and associativity rules
  - Dangling else

Java-cup
- Precedence and associativity declarations
- Shift Reduce conflict in dangling else
**SLR still not good enough wrt reduce**

**Grammar for C style assignment:**

0: $S' \rightarrow S$  
1: $S \rightarrow V = E$  
2: $S \rightarrow E$  
3: $E \rightarrow V$  
4: $V \rightarrow x$  
5: $V \rightarrow ^* E$

$E$ is a simplified expression, in reality more complex ($E + T \ ...$)

Let’s build the SLR parse table:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>*</th>
<th>=</th>
<th>$</th>
<th>S</th>
<th>V</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td>g2</td>
<td>g3</td>
<td>g4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>s7,r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td>g9</td>
<td>g8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s5</td>
<td>s6</td>
<td></td>
<td>g9</td>
<td>g10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r5</td>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Is this grammar ambiguous?*

**NO**

*But we still have a shift reduce conflict.*

*We need to look ahead and decide when to reduce.*
LR(1) look ahead sets

LR(1) items take a look ahead into account: \((A \rightarrow \alpha . \beta . z)\)

item is pair: (dotted rule, look-ahead symbol)

indicating \(\alpha\) is on top of stack, and \(\beta . z\) on input, but \(z\) is not part of \(A\)

We will now reduce to \(A\), not on \(\text{follow}(A)\), but if \(\alpha\beta\) is on top of stack and \(z\) on input, i.e. we have more precise information.

\[
\text{Closure}(I) = \begin{array}{l}
\text{repeat} \\
\text{for all } (A \rightarrow \alpha . X\beta . z) \text{ in } I \\
\text{for any } X \rightarrow \gamma \\
\text{for any } w \text{ in } \text{First}(\beta z) \\
I+ = (X \rightarrow \gamma , w) \\
\text{until } I \text{ does no change} \\
\text{return } I
\end{array}
\]

\[
\text{Goto}(I,X) = \begin{array}{l}
J=\{\} \\
\text{for all } (A \rightarrow \alpha . X\beta . z) \text{ in } I \\
\text{for any } X \rightarrow \gamma \\
\text{for any } w \text{ in } \text{First}(\beta z) \\
J+ = (A \rightarrow \alpha X . \beta , z) \\
\text{return } \text{Closure}(J)
\end{array}
\]

\[
\text{Reduce:} \\
R = \{\} \\
\text{for all } (A \rightarrow \alpha .) \text{ in } I \\
R+ = (I, z, A \rightarrow \alpha) \\
(I, z, A \rightarrow \alpha) \text{ means:} \\
in \text{State } I, \\
on \text{symbol } z \\
\text{reduce: } A \rightarrow \alpha
\]
LR(1) state diagram: look-ahead sets

0: S’ → S$  1: S → V = E  2: S → E  3: E → V  4: V → x  5: V → *E

We will not shift the $, so the look ahead symbol for S’ does not matter
This is indicated by a ?

S’ → . S$  ?  when we have  S’ → . S$  ?
S → . V = E  $  the same dotted  S → . V = E  $  
S → . E  $  rule with  S → . E  $  
E → . V  $  different  E → . V  $  
V → . x  $  look aheads  V → . x  $, =
V → . *E  $  we combine  V → . *E  $, =
V → . x  =  them in
V → . *E  =  sets
LR(1) state diagram  (incomplete)

0: $S' \rightarrow SS$
1: $S \rightarrow V = E$
2: $S \rightarrow E$
3: $E \rightarrow V$
4: $V \rightarrow x$
5: $V \rightarrow *E$

In state 3:
we reduce $E \rightarrow V$ on $\$, not on $=$
( even though $=$ is in follow($E$) )
we shift on $=$

conflict resolved
LALR(1) Look Ahead LR(1) : state diagram (incomplete)

0: $S' \rightarrow S$
1: $S \rightarrow V = E$
2: $S \rightarrow E$
3: $E \rightarrow V$
4: $V \rightarrow x$
5: $V \rightarrow \ast E$

LALR combines states that are equal except for the look aheads.
LALR(1) Look ahead LR(1) : state diagram (incomplete)

0: S' → S$  1: S → V = E  2: S → E  3: E → V  4: V → x  5: V → *E

```
0: S' → S$
1: S → V = E
2: S → E
3: E → V
4: V → x
5: V → *E
```
Shift reduce: ambiguous expression grammar (incomplete)

0: \( S \rightarrow E \)  1: \( E \rightarrow E + E \)  2: \( E \rightarrow E * E \)  3: \( E \rightarrow (E) \)  4: \( E \rightarrow \text{id} \)

\[ S' \rightarrow E $ \]
\[ E \rightarrow . \ E + E \]
\[ E \rightarrow . \ E * E \]
\[ \ldots \]

State 4: two Shift/Reduce conflicts

input +: R1  Why?
input *: S  Why?

State 6: two Shift/Reduce conflicts

input +: R2  Why?
input *: R2  Why?

NOTICE:
+ and * are left associative
Shift reduce: ambiguous expression grammar (incomplete)

0: \( S' \rightarrow E\$ \)
1: \( E \rightarrow E^E \)
2: \( E \rightarrow \text{id} \)

// ^ for exponent

State 4: two Shift/Reduce conflicts

input ^ : S  Why?
Shift reduce: ambiguous dangling else grammar

1: $S \rightarrow \text{if } c \ S \ 	ext{else } S$
2: $S \rightarrow \text{if } c \ S$
3: $S \rightarrow \text{other}$

abstracted to: 1: $S \rightarrow i \ S \ e \ S$
2: $i \ S$
3: $o$

State 5: Shift/Reduce conflict

input e: $S$

Why?

Is else left or right associative?
Shift reduce: ambiguous cast grammar

1: \( E \rightarrow E + E \)  2: \( E \rightarrow (\text{byte}) E \)  3: \( E \rightarrow (E) \)  4: \( E \rightarrow \text{id} \)

\[ E \rightarrow E + E \]
\[ E \rightarrow (\text{byte}) E \]
\[ E \rightarrow (E) \]
\[ E \rightarrow \text{id} \]

---

\[ E \rightarrow E . + E \]
\[ E \rightarrow (\text{byte}) E . \]

Shift/Reduce conflict
input +=: R2    Why?

\((\text{type})\) is a right associative cast operator, but consists of three tokens. This is a place where the strict nesting of \((\) does not hold. The conflict can be resolved by declaring \((\) a right associative unary operator with higher precedence than the other operators.
Ambiguity in JavaCUP

There is no conflict free LR parser for ambiguous grammars
- So we have to fix the problem in other ways

Expressions: operators and productions are given precedence
- Tokens explicitly in precedence declarations
- Productions implicitly based on last operator in the stack

When in shift-reduce conflict
- Shift if look ahead token has higher precedence than production on stack
- Reduce if operator in the stack has higher precedence than look ahead

Within same precedence level
- Left associativity results in a reduce
- Right associativity results in a shift

Dangling else
- Shifting the else will bind else to last unbound if (Java semantics)
- Not shifting can cause a valid program to be rejected