CSCI 742 - Compiler Construction

Lecture 19
SLR and LR(1) Parsing
Instructor: Hossein Hojjat

March 8, 2017
Consider the grammar \( S \rightarrow (S) \mid \text{num} \)
Creating Parse Tables

For each state:

- Transition to another state using a terminal symbol is a **shift** to that state.
- Transition to another state using a non-terminal is a **goto** to that state.
- If there is a single item $A \rightarrow \alpha \cdot$ in the state **reduce** with that production for all terminals.
Building Parse Table Example

<table>
<thead>
<tr>
<th>(   )</th>
<th>num</th>
<th>$</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s2</td>
<td>s3</td>
<td>g1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>s3</td>
<td>g4</td>
</tr>
<tr>
<td>3</td>
<td>r(S→num)</td>
<td>r(S→num)</td>
<td>r(S→num)</td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>r(S→(S))</td>
<td>r(S→(S))</td>
<td>r(S→(S))</td>
</tr>
</tbody>
</table>

```
S' → S $  
S' → S •  
S → •(S)  
S → •num  
S → •num  
S → (S$)  
S → (S)  
S → (S•)  
S → (S).  
```

Diagram:

0
S' → S $  
0
S → •(S)  
0
S → •num  
S
1
S' → S •  
S
3
S → •num
num
3
S → •num  
num
3
S → •num
num
4
S → (S$)  
4
S → (S)  
4
S → (S•)  
5
S → (S).  
5
S
LR(0) Limitations

- LR(0) only works if states with reduce actions have a single reduce action
  
  $E \rightarrow T \cdot$

- In those states it always reduce without looking at lookahead

- LR(0) is vulnerable to unnecessary conflicts

- Shift/Reduce Conflicts (may reduce too soon in some cases)
  
  $E \rightarrow E \cdot +T$
  $S \rightarrow E \cdot$

- Reduce/Reduce Conflicts
  
  $E \rightarrow \text{num} \cdot$
  $T \rightarrow \text{num} \cdot$
LR(0) Parsing Table With Conflicts

<table>
<thead>
<tr>
<th></th>
<th>(   )</th>
<th>+</th>
<th>num</th>
<th>$</th>
<th>S</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
<td>s4</td>
<td>g2</td>
<td>g1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>g5</td>
</tr>
<tr>
<td>4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s8</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s7</td>
</tr>
<tr>
<td>7</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
S' & \rightarrow \cdot S$

S & \rightarrow \cdot E

S & \rightarrow \cdot E + \text{num}

E & \rightarrow \cdot (E)

S & \rightarrow \cdot \text{num}

E & \rightarrow \text{num}\
\end{align*}

\[
\begin{align*}
r1 & \quad S \rightarrow E

r2 & \quad E \rightarrow E + \text{num}

r3 & \quad E \rightarrow (E)

r4 & \quad E \rightarrow \text{num}
\end{align*}

Simple LR parsing (SLR) is a simple extension of LR(0) parsing.
For each reduction \( A \rightarrow \gamma \), look at the lookahead symbol \( c \).
Apply reduction only if \( c \) is in \( \text{FOLLOW}(A) \).

SLR Parsing Table
- Eliminates some conflicts
- Same as LR(0) table except reduction rows
- Reductions do not fill entire rows
- Add reductions \( A \rightarrow \gamma \) only in the columns of symbols in \( \text{FOLLOW}(A) \).
### LR(0) Parsing Table

<table>
<thead>
<tr>
<th></th>
<th>(     )</th>
<th>+</th>
<th>num</th>
<th>$</th>
<th>S</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
<td>s4</td>
<td>g2</td>
<td>g1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>r1</td>
<td>r1</td>
<td>r1/s6</td>
<td>r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>accept</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>s4</td>
<td>g5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s8</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Grammar Rules

- $S \rightarrow E$
- $E \rightarrow E + \text{num}$
- $E \rightarrow (E)$
- $E \rightarrow \text{num}$

### Diagram

```
0   E
   S' → •S$
   S → •E
   S → •E + num
   E → •(E)
   S → •num

1   S' → S •$
   S → E •+num
   S → E •$

2   S' → S •$
   S → E •$

3   (E

4   E
   S → E •$
   S → •num

5   E
   S → E •+num
   S → (E)

6   E
   S → E •$
   S → E •$

7   S → E + •num
   S → E + num

8   S → (E)

num

+ num

+ num
```

```
SLR Parsing Table

<table>
<thead>
<tr>
<th></th>
<th>(   )</th>
<th>+</th>
<th>num</th>
<th>$</th>
<th>S</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
<td>s4</td>
<td>g2</td>
<td>g1</td>
<td>S</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>s6</td>
<td>r1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>accept</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>s4</td>
<td>g5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s8</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
r1  S → E
r2  E → E + num
r3  E → (E)
r4  E → num
```
**LR(1) Parsing**

- **Idea:** Get as much as possible out of 1 lookahead symbol parsing table
- LR(1) grammar = recognizable by a shift/reduce parser with 1 lookahead
- LR(1) parsing uses similar concepts as LR(0)
- Parser states = set of LR(1) items
- LR(1) item = LR(0) item + lookahead symbols possibly following production
- LR(0) item: $S \rightarrow \bullet S + E$
- LR(1) item: $S \rightarrow \bullet S + E$, $+$
- Lookahead only has impact on reduce operations: apply when lookahead = next input
LR(1) States

- LR(1) state = set of LR(1) items
- LR(1) item = \((X \rightarrow \alpha \cdot \beta, y)\)
- Meaning: \(\alpha\) already matched at top of the stack, next expect to see \(\beta y\)
- Shorthand notation: \((X \rightarrow \alpha \cdot \beta, \{x_1, \cdots, x_n\})\) means:
  - \((X \rightarrow \alpha \cdot \beta, x_1)\)
  - \(\cdots\)
  - \((X \rightarrow \alpha \cdot \beta, x_n)\)

- Need to extend closure and goto operations
LR(1) Closure

Similar to LR(0) closure, but also keeps track of lookahead symbol.

If $L$ is a set of items, $\text{CLOSURE}(L)$ is the set of items such that:

- every item in $L$ is in $\text{CLOSURE}(L)$
- if item $(X \rightarrow \alpha \cdot Y \beta, z)$ is in $\text{CLOSURE}(L)$ and $Y \rightarrow \gamma$ is a production then $(Y \rightarrow \cdot \gamma, \text{FIRST}(\beta z))$ is also in $\text{CLOSURE}(L)$
LR(1) Start State

Initial state: start with \((S' \rightarrow \cdot S, \$)\), then apply closure operation

Goto is analogous to goto in LR(0) parsing

**Goto(L, X)**

\[ I = \emptyset \]

for any item \([A \rightarrow \alpha \cdot X \beta, x]\) in \(L\)

\[ I = I \cup \{[A \rightarrow \alpha X \cdot \beta, x]\} \]

return \text{CLOSURE}(I)
Exercise

Construct the LR(1) automaton for the following grammar:

\[
\begin{align*}
S' & \rightarrow S$ \\
S & \rightarrow E + S \mid E \\
E & \rightarrow \text{num}
\end{align*}
\]