Compiler Construction

Lecture 7 - LR(1), LALR and SLR(1)
Parsing
© 2003 Robert M. Siegfried
All rights reserved

Outline

I. SLR(1) State Machine
II. LR(1) State Machine
III. LALR State Machine
Shift-Reduce Conflicts

Let’s consider our original expression grammar:

\[
E ::= E + T | T \\
T ::= T * F | F \\
F ::= id | ( E )
\]

If we try to build an LR(0) parser, we will have a problem.

Example: \( x + y * z \)

We have a conflict - do we shift or reduce?

Looking at the state machine by itself does not tell us.

Answer: We must use a lookahead.

The simplest version of an LR(1) parser is called SLR(1) (The “S” is for “Simplified”)

We will use the follows as a means of resolving the shift-reduce conflict.
The Initial State of the SLR(1) State Machine

0

\[ S ::= . \text{E}$ \]
\[ E ::= . \text{E + T} \]
\[ E ::= . \text{T} \]
\[ T ::= . \text{T * F} \]
\[ T ::= . \text{F} \]
\[ F ::= . \text{id} \]
\[ F ::= . (\text{E}) \]

The SLR(1) State Machine
- Initial State’s Transitions
The SLR(1) State Machine
The Next Set of Transitions

0
S ::= .
E ::= E + T
E ::= T
T ::= T + F
T ::= F
F ::= . id
F ::= . ( E )

1
S ::= E . S
E ::= E + T

2
E ::= T .
T ::= T * F
T ::= F
F ::= . id
F ::= . ( E )

3
T ::= F.

4
F ::= . id.
F ::= . ( E )

5
F ::= . id.
F ::= . ( E )
E ::= E + T

6
F ::= K.
E ::= E + T
E ::= T
T ::= T * F
T ::= F
F ::= . id
F ::= . ( E )

7
E ::= E + T
T ::= T * F
T ::= F
F ::= . id
F ::= . ( E )

8
E ::= E + T
T ::= T * F

9
T ::= T * F
F ::= . id
E ::= E + T

10
T ::= T * F.

11
F ::= ( E )
E ::= E + T

12
S ::= E $ .
E

?
The SLR(1) State Machine (continued)

To make the ACTION/GOTO table, we need the FOLLOW sets, finding the FIRST sets where necessary.

FIRST(E) = FIRST(T) = FIRST(F) = \{ id, ( ) \}
FIRST(T) = FIRST(F) = \{ id, ( ) \}
FIRST(F) = \{ id, ( ) \}

FOLLOW(E) = \{ $, +, ) \}
FOLLOW(T) = \{ $, +, *, ) \}
FOLLOW(F) = \{ $, +, *, ) \}
The SLR(1) Parse Table

- We will use the FOLLOW sets to determine which action we will take depending on which lookahead token.
- We will include the GOTOs for all terminals wherever the action is *Shift*.
- Nonterminals will have only GOTOs.

<table>
<thead>
<tr>
<th>State</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>id</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

The SLR(1) Parse Table

```
 state | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
-------|---|---|---|---|---|---|---|---|---|---|----|----|----|
 ACTION |+ | id |      | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
 GOTO   |      |      |      |      |      |      |      |      |      |      |      |      |      |
-------|---|---|---|---|---|---|---|---|---|---|----|----|----|
```
LR(1) Parsing

• SLR(1) grammars are somewhat limited in terms of the languages that they can process.
• LR(1) use a lookahead and place the lookahead within the term:
  1. \( S ::= . E $, \{ \varepsilon \} \)  
     *Nothing can follow S*
  2. \( E ::= . E + T, \{ $ + \} \)  
     *$ comes from prod. 1*
  3. \( E ::= T, \{ $ + \} \)  
     *+ comes from prod. 2*

Building The LR(1) State Machine

```
0
S ::= . E$, \{ \varepsilon \}
E ::= . E + T, \{ $+ \}
E ::= . T, \{ $+ \}
T ::= . T * F, \{ $+* \}
T ::= . F, \{ $+* \}
F ::= . id, \{ $+* \}
F ::= . ( E ), \{ $+* \}
```

*comes from production 4*
*comes from production 4*
Building The LR(1) State Machine

The follow sets in the items do not change from state to state

The follow sets in initial items are based on what follows E in the item above

Building The LR(1) State Machine

1. $S ::= E S$, {$ε$
2. $E ::= E + T$, {$+$}
3. $E ::= E * T$, {$*$}
4. $E ::= T$, {$+$}
5. $F ::= id$, {$+$}
6. $F ::= ( E )$, {$+$}

1. $S ::= E S$, {$ε$
2. $E ::= E + T$, {$+$}
3. $E ::= E * T$, {$*$}
4. $E ::= T$, {$+$}
5. $F ::= id$, {$+$}
6. $F ::= ( E )$, {$+$}

The follow sets in initial items are based on what follows E in the item above
The LR(1) State Machine (continued)
### The LR(1) Parse Table

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>s4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>r3</td>
</tr>
<tr>
<td>3</td>
<td>r5</td>
</tr>
<tr>
<td>4</td>
<td>r6</td>
</tr>
<tr>
<td>5</td>
<td>s7</td>
</tr>
<tr>
<td>6</td>
<td>s16</td>
</tr>
<tr>
<td>7</td>
<td>s4</td>
</tr>
<tr>
<td>8</td>
<td>r2</td>
</tr>
<tr>
<td>9</td>
<td>s4</td>
</tr>
<tr>
<td>10</td>
<td>r4</td>
</tr>
<tr>
<td>11</td>
<td>r7</td>
</tr>
<tr>
<td>12</td>
<td>r2</td>
</tr>
<tr>
<td>13</td>
<td>r3</td>
</tr>
<tr>
<td>14</td>
<td>r5</td>
</tr>
<tr>
<td>15</td>
<td>s8</td>
</tr>
<tr>
<td>16</td>
<td>s6</td>
</tr>
<tr>
<td>17</td>
<td>s18</td>
</tr>
<tr>
<td>18</td>
<td>s16</td>
</tr>
<tr>
<td>19</td>
<td>r7</td>
</tr>
<tr>
<td>20</td>
<td>r2</td>
</tr>
<tr>
<td>21</td>
<td>r4</td>
</tr>
</tbody>
</table>
LALR Parsing

- We seek to simplify LR(1) without losing all its power. We do this by combining states that match in all regards other than the lookahead.
- Because this process can be cumbersome when there are a large number of states, it is often done by building something that is very similar to the SLR(1) machine.

<table>
<thead>
<tr>
<th>LALR States</th>
<th>LR(1) States combined to form this LALR State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2 &amp; 13</td>
</tr>
<tr>
<td>3</td>
<td>3 &amp; 14</td>
</tr>
<tr>
<td>4</td>
<td>4 &amp; 16</td>
</tr>
<tr>
<td>5</td>
<td>5 &amp; 17</td>
</tr>
<tr>
<td>6</td>
<td>6 &amp; 15</td>
</tr>
<tr>
<td>7</td>
<td>7 &amp; 16</td>
</tr>
<tr>
<td>8</td>
<td>8 &amp; 12</td>
</tr>
<tr>
<td>9</td>
<td>9 &amp; 20</td>
</tr>
<tr>
<td>10</td>
<td>10 &amp; 21</td>
</tr>
<tr>
<td>11</td>
<td>11 &amp; 19</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
</tr>
</tbody>
</table>
The LALR State Machine

0
S ::= . E, {ε}
E ::= E + T, { + }
E ::= E, { + }
T ::= T * F, { * }
T ::= T, { * }
F ::= id, { * }
F ::= ( E ), { * }

1
S ::= E, {ε}
E ::= E + T, { + }
E ::= E, { + }
T ::= T * F, { * }
T ::= T, { * }
F ::= id, { * }
F ::= ( E ), { * }

2
E ::= T, { ) + }
T ::= T * F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

3
T ::= F, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

4
F ::= id, { ) * }
F ::= ( E ), { ) * }

5
F ::= ( E . ), { ) * }
E ::= E + T, { ) + }
E ::= E, { ) + }
T ::= T * F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

6
F ::= ( E . ), { ) * }
E ::= E + T, { ) + }
E ::= E, { ) + }
T ::= T * F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

7
E ::= E + T, { ) + }
T ::= T + F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

8
E ::= E + T, { ) + }
T ::= T + F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

9
E ::= E + T, { ) + }
T ::= T + F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

10
E ::= E + T, { ) + }
T ::= T + F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

11
E ::= E + T, { ) + }
T ::= T + F, { ) * }
T ::= T, { ) * }
F ::= id, { ) * }
F ::= ( E ), { ) * }

12
S ::= E, {ε}
E ::= E + T, { + }
E ::= E, { + }
T ::= T * F, { * }
T ::= T, { * }
F ::= id, { * }
F ::= ( E ), { * }

The LALR State Machine

The LALR State Machine

The LALR State Machine

The LALR State Machine