LR(0) Parsing Summary

• LR(0) item = a production with a dot in RHS
• LR(0) state = set of LR(0) items valid for a set of viable prefixes
• Compute LR(0) states and build DFA:
  - Start state: \( V(\varepsilon) = \{ [S'\rightarrow.S] \} \downarrow^* \)
  - Other states: \( V(\alpha X) = V(\alpha)\rightarrow_x \downarrow^* \)
• Build the LR(0) parsing table from the DFA
• Use the LR(0) parsing table to determine whether to reduce or to shift
LR(0) Limitations

• An LR(0) machine only works if each state with a reduce action has only one possible reduce action and no shift action

• With some grammars, construction gives states with shift/reduce or reduce/reduce conflicts

• Need to use look-ahead to choose

\[
\begin{align*}
\text{ok} & : [ L & \rightarrow L,S ] \\
\text{shift /reduce} & : [ L & \rightarrow L,S ] \\
& : [ S & \rightarrow S,L ] \\
\text{reduce / reduce} & : [ L & \rightarrow S,L ] \\
& : [ L & \rightarrow S ]
\end{align*}
\]
# LR(0) Parsing Table

<table>
<thead>
<tr>
<th></th>
<th>(</th>
<th>id</th>
<th>,</th>
<th>$\epsilon$</th>
<th>S</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>S→id</td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>s6</td>
<td>s8</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></td>
</tr>
<tr>
<td>7</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td>L→L,S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Non-LR(0) Grammar

• Grammar for addition of numbers:
  \[ S \rightarrow S + E \mid E \]
  \[ E \rightarrow \text{num} \]

• Left-associative version is LR(0)

• Right-associative version is not LR(0)
  \[ S \rightarrow E + S \mid E \]
  \[ E \rightarrow \text{num} \]
LR(0) Parsing Table

What to do in state 2: shift or reduce?

<table>
<thead>
<tr>
<th>State</th>
<th>LR(0) Parsing Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image" alt="State 1 Diagram" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image" alt="State 2 Diagram" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image" alt="State 3 Diagram" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="State 4 Diagram" /></td>
</tr>
<tr>
<td>5</td>
<td><img src="image" alt="State 5 Diagram" /></td>
</tr>
<tr>
<td>6</td>
<td><img src="image" alt="State 6 Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Symbol</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S' → S</td>
<td>E</td>
<td>2</td>
</tr>
<tr>
<td>S → .E+S</td>
<td>num</td>
<td>4</td>
</tr>
<tr>
<td>S → .E</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>E → .num</td>
<td>num</td>
<td>5</td>
</tr>
<tr>
<td>S' → S.</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Symbol</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s3/S → E</td>
<td>S → E</td>
</tr>
</tbody>
</table>

What to do in state 2: shift or reduce?
SLR(1) Parsing

- **SLR Parsing = easy extension of LR(0)**
  - For each reduction $A \rightarrow \beta$, look at the next symbol $c$
  - Apply reduction only if $c$ is in $\text{FOLLOW}(A)$, or $c = \varepsilon$ and $S \Rightarrow^{*} \gamma A$

- **SLR parsing table eliminates some conflicts**
  - Same as LR(0) table except reduction rows
  - Adds reductions $A \rightarrow \beta$ only to columns of symbols in $\text{FOLLOW}(A)$, or to column $\varepsilon$ if $S \Rightarrow^{*} \gamma A$

- **Example:**
  - $\text{FOLLOW}(S) = \{\}$
  - but $S \Rightarrow^{*} \gamma E$

<table>
<thead>
<tr>
<th></th>
<th>num</th>
<th>+</th>
<th>$\varepsilon$</th>
<th>E</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s4</td>
<td></td>
<td></td>
<td>g2</td>
<td>g6</td>
</tr>
<tr>
<td>2</td>
<td>s3</td>
<td></td>
<td>$S \rightarrow E$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CS 412/413   Spring 2008   Introduction to Compilers 7
## SLR Parsing Table

- Reductions do not fill entire rows
- Otherwise, same as LR(0)

<table>
<thead>
<tr>
<th></th>
<th>num</th>
<th>+</th>
<th>$\epsilon$</th>
<th>E</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>g2 g6</td>
</tr>
<tr>
<td>2</td>
<td>s3</td>
<td>$S\rightarrow E$</td>
<td></td>
<td>g2 g5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$S\rightarrow E$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>$S\rightarrow E+S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>s7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>accept</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SLR(k)

- Use the LR(0) machine states as rows of table
- Let Q be a state and u be a lookahead string
  - Action(Q,u) = shift Goto(Q,b)
    if Q contains an item of the form \([A \rightarrow \beta_1 \cdot b \beta_3]\), with \(u \in \text{FIRST}_k(b \beta_3 \cdot \text{FOLLOW}_k(A))\)
  - Action(Q,u) = accept
    if Q = \{ [S' \rightarrow S. ] \} and \(u = \varepsilon\)
  - Action(Q,u) = reduce i
    if Q contains the item \([A \rightarrow \beta. ]\), where \(A \rightarrow \beta\) is the ith production of G and \(u \in \text{FOLLOW}_k(A)\), or \(u = \varepsilon\) and \(S \Rightarrow^{*} \gamma A\)
  - Action(Q,u) = error otherwise
- G is SLR(k) iff the Action function given above is single-valued for all Q and u, i.e, there are no shift-reduce or reduce-reduce conflicts.
LR(1) Parsing

• Get as much power as possible out of 1 look-ahead symbol parsing table

• LR(1) grammar = recognizable by a shift/reduce parser with 1-symbol look-ahead

• LR(1) parsing uses concepts similar to LR(0)
  - Parser states = sets of items
  - LR(1) item = LR(0) item + look-ahead symbol following the production

\[
\begin{align*}
\text{LR(0) item:} & \quad [ S \rightarrow .S+E ] \\
\text{LR(1) item:} & \quad [ S \rightarrow .S+E + ]
\end{align*}
\]
LR(1) States

- LR(1) state = set of LR(1) items
- LR(1) item = \([ A \rightarrow \alpha \cdot \beta \ b] \), where \(b \in \Sigma \cup \{\varepsilon\}\)
- Meaning: \(\alpha\) already matched at top of the stack; next expect to see \(\beta b\)

- Shorthand notation
  \([ A \rightarrow \alpha \cdot B \ b_1, \ldots, b_n] \)

  means:
  \([ A \rightarrow \alpha \cdot \beta \ b_1] \)
  ...
  \([ A \rightarrow \alpha \cdot \beta \ b_n] \)

- Extend closure and goto operations
LR(1) Closure

• LR(1) closure operation on set of items S
  - For each item in S:
    \([A \rightarrow \alpha \cdot B\beta \ b]\)
    and for each production \(B \rightarrow \gamma\), add the following item to S:
    \([B \rightarrow .\gamma \ FIRST(\beta b)],\ or\]
    \([B \rightarrow .\gamma \ \epsilon] \ if \ FIRST(\beta b)=\{\}\]
  - Repeat until nothing changes

• Similar to LR(0) closure, but also keeps track of the look-ahead symbol
LR(1) Start State

- Initial state: start with \([S' \rightarrow .S \quad \varepsilon]\), then apply the closure operation.

- Example: sum grammar

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

\[
\begin{align*}
[S' \rightarrow .S \quad \varepsilon] \\
[S \rightarrow .E + S \quad \varepsilon] \\
[S \rightarrow .E \quad \varepsilon] \\
[E \rightarrow .\text{num} \quad +, \varepsilon]
\end{align*}
\]
LR(1) Goto Operation

- **LR(1) goto operation** = describes transitions between LR(1) states

- **Algorithm**: for a state $S$ and a symbol $Y$
  - $S' = \{ [A \rightarrow \alpha Y . \beta \ b] \mid [A \rightarrow \alpha . Y \beta \ b] \in S \}$
  - $\text{Goto}(S, Y) = \text{Closure}(S')$

![Diagram of LR(1) Goto Operation]

1. $S_1$
   - $[S \rightarrow E . + S \ \varepsilon]$
   - $[S \rightarrow E . \ \varepsilon]$

2. $S_2$
   - $\text{Goto}(S_1, '+')$
   - $\text{Closure}(\{ [S \rightarrow E + . S \ \varepsilon] \})$
LR(1) DFA Construction

- If $S' = \text{Goto}(S,X)$ then add an edge labeled $X$ from $S$ to $S'$

```
[S' \rightarrow S. \varepsilon]
[S \rightarrow .E+.S \varepsilon]
[S \rightarrow .E \varepsilon]
[E \rightarrow .num +,\varepsilon]
```

```
[S 
  \rightarrow .S \varepsilon]
[S \rightarrow .E+.S \varepsilon]
[S \rightarrow .E \varepsilon]
[E \rightarrow .num +,\varepsilon]
```

```
[S \rightarrow E+.S \varepsilon]
[S \rightarrow .E+.S \varepsilon]
[S \rightarrow .E \varepsilon]
[E \rightarrow .num +,\varepsilon]
```

```
[E \rightarrow num,+ ,\varepsilon]
```

```
[S' \rightarrow S. \varepsilon]
[S \rightarrow .E+.S \varepsilon]
[S \rightarrow .E \varepsilon]
[E \rightarrow .num +,\varepsilon]
```

```
[S \rightarrow .E+.S \varepsilon]
[S \rightarrow .E \varepsilon]
[E \rightarrow .num +,\varepsilon]
```

```
[E \rightarrow num,+ ,\varepsilon]
```

---

CS 412/413  Spring 2008  Introduction to Compilers  15
LR(1) Reductions

• Reductions correspond to LR(1) items of the form \([A \rightarrow \beta \cdot x]\)

\[
\begin{align*}
[S' \rightarrow .S & \quad \varepsilon] \\
[S \rightarrow .E+S & \quad \varepsilon] \\
[S \rightarrow .E & \quad \varepsilon] \\
[E \rightarrow .\text{num} & +,\varepsilon] \\
[E \rightarrow .\text{num} +,\varepsilon]
\end{align*}
\]
LR(1) Parsing Table Construction

• Same as construction of LR(0) parsing table, except for reductions

• If $[A \rightarrow \beta \cdot b] \in \text{state } Q$, then:
  $\text{Action}(Q, b) \text{ is } \text{Reduce}(A \rightarrow \beta)$
LR(1) Parsing Table Example

Fragment of the Parsing table:

1

1 [S’ → S ε]
   [S → E+S ε]
   [S → E ε]
   [E → .num +, ε]

2 [S → E+S ε]
   [S → E . ε]

3

3 [S → E+.S ε]
   [S → .E+S ε]
   [S → .E ε]
   [E → .num +, ε]

E

+ ε

E

+ ε

E

s3 S→E
LR(1) but not SLR(1)

• Let $G$ have productions
  
  $S \rightarrow aAb \mid Ac$
  
  $A \rightarrow a \mid \varepsilon$

• $V(a) = \{ \$

  $[ S \rightarrow a.\text{Ab} ]$
  
  $[ A \rightarrow a. ]$
  
  $[ A \rightarrow .a ]$
  
  $[ A \rightarrow . ]$

  $\}$ $\quad \text{FOLLOW}(A) = \{b, c\}$ $\quad \text{reduce-reduce conflict}$
LALR(1) Grammars

• Problem with LR(1): too many states
• LALR(1) Parsing (Look-Ahead LR)
  – Construct LR(1) DFA and then merge any two LR(1) states whose items are identical except look-ahead
  – Results in smaller parser tables
  – Theoretically less powerful than LR(1)

\[
[S \rightarrow \text{id}. +] + [S \rightarrow \text{id}. \varepsilon] = ?
\]

[S \rightarrow \text{E}. \varepsilon]
[S \rightarrow \text{E}. +]

• LALR(1) Grammar = a grammar whose LALR(1) parsing table has no conflicts
Classification of Grammars

- LR(0)
- LL(1)
- SLR(1)
- LALR(1)
- LR(1)
- LR(k) ⊆ LR(k+1)
- LL(k) ⊆ LL(k+1)
- LL(k) ⊆ LR(k)
- LR(0) ⊆ SLR(1)
- LALR(1) ⊆ LR(1)
Automate the Parsing Process

• Can automate:
  – The construction of LR parsing tables
  – The construction of shift-reduce parsers based on these parsing tables

• Automatic parser generators: yacc, bison, CUP

• LALR(1) parser generators
  – Not much difference compared to LR(1) in practice
  – Smaller parsing tables than LR(1)
  – Augment LALR(1) grammar specification with declarations of precedence, associativity

• output: LALR(1) parser program
What happens if we run this grammar through LALR construction?
Shift/Reduce Conflict

\[
E \rightarrow E + E \\
E \rightarrow \text{num}
\]

\[
\begin{align*}
[E & \rightarrow E + E. \quad +] \\
[E & \rightarrow E. + E \quad +, \varepsilon]
\end{align*}
\]

shift/reduce conflict

shift: 1+(2+3)  
reduce: (1+2)+3  
\[1+2+3\]
nonterminal \( E \); terminal \( PLUS, \ LPAREN \ldots \)  
precedence \texttt{left PLUS}; 

```
E ::= E  PLUS  E
|   LPAREN  E  RPAREN
|   NUMBER  ;
```

“when shifting a ‘+’ conflicts with reducing a production, choose reduce”
Precedence

• CUP can also handle operator precedence

\[
E \rightarrow E + E \mid T \\
T \rightarrow T \times T \mid \text{num} \mid (E)
\]

\[
E \rightarrow E + E \mid E \times E \\
\mid \text{num} \mid (E)
\]
Conflicts without Precedence

\[
E \rightarrow E + E \mid E \times E \\
\mid \text{num} \mid (E)
\]
Precedence in CUP

precedence left PLUS;
precedence left TIMES; // TIMES > PLUS
E ::= E PLUS E | E TIMES E | ...

RULE: in conflict, choose reduce if last terminal of production has higher precedence than symbol to be shifted; choose shift if vice-versa. In tie, use associativity (left or right) given by precedence rule

reduce  E → E×E

Shift  ×
Summary

• Look-ahead information makes SLR(1), LALR(1), LR(1) grammars expressive
• Automatic parser generators support LALR(1) grammars
• Precedence, associativity declarations simplify grammar writing