LR Parsers

Definition: First_k(x) = first k symbols of x.
(x is a string of terminals)

A grammar is LR(k) iff

• $S \Rightarrow^*_{rm} \alpha A w \Rightarrow \alpha \beta w$
• $S \Rightarrow^*_{rm} \gamma B x \Rightarrow \alpha \beta y$
• $First_k(w) = First_k(y)$

Imply that $\alpha A w = \gamma B x$.

• same context (alpha)
• same lookahead ($First_k(w), First_k(y)$)
LR(0) -- No Lookahead

Not a practical parser generator

Parser Construction

- Based on the idea of a "configuration" or "item"
  \[ A \rightarrow X_1 \ldots X_i . X_{i+1} \ldots X_j \]

- And on a set of items
  \[ stmt \rightarrow ID . := expr \]
  \[ stmt \rightarrow ID . : stmt \]
  \[ stmt \rightarrow ID . \]

  ID has been matched, but nothing following
  all three are possibilities
Building Configuration Sets (Item Sets)

- Assume S is start symbol
- Add new start symbol
  
  \[ S' \rightarrow S \; \$ \; (\$ \text{ is EOF}) \]

- Initial set, S_0, STARTS as:
  
  \[
  \{ \; S' \rightarrow . \; S \; \$ \; \}
  \]

- Closure is next:
  
  - . \; A \; \rightarrow \; \text{All } A \rightarrow Y_1 \; \ldots \; Y_n \text{ need to be added}
  
  - A \rightarrow . \; Y_1 \; \ldots \; Y_n \text{ is added}
LR(0) Example

\[
S \rightarrow E$

\[
E \rightarrow E + T \mid T$

\[
T \rightarrow ID \mid (E)$

□Initial set, \( S_0 \)

\{
S \rightarrow . E$
\}

Algorithm Closure_LR0 (set \( S \))

repeat

for all items \( B \rightarrow \alpha . A \beta \) in \( S \), \( A \) in Variables

add all items of the form \( A \rightarrow . \gamma \) to \( S \)

until no new items can be added
Set $S_0$

\[
S \rightarrow E \$
E \rightarrow E + T \mid T
T \rightarrow ID \mid (E)
\]

Initial set, $S_0$
\{
$S \rightarrow . E \$
\}

Closure_{LR0} ($S_0$):
\{
$S \rightarrow . E \$
$E \rightarrow . E + T$
$E \rightarrow . T$
$T \rightarrow . ID$
$T \rightarrow . (E)$
\}
GoTo Algorithm

- Compute "successor" states from a state
- For an item: $A \rightarrow \alpha . X \beta$ a new set is started
- Based on the $X$ part ($X$ a terminal or variable)

Algorithm `go_to_LR0 (Set S, symbol X)`

New set is $S'$

1) $S' \leftarrow \{\}$

2) for each configuration $C$ in $S$ where $C$ is of the form $A \rightarrow \alpha . X \beta$
   
   Add $A \rightarrow \alpha X . \beta$ to $S'$

3) compute closure_LR0 ($S'$)

4) return $S'$
Go to of $S_0$

\[
S_0 = \{ S \rightarrow .E$  
    \ 
    E \rightarrow .E + T  \ 
    E \rightarrow .T  \ 
    T \rightarrow .ID  \ 
    T \rightarrow .(E) \ 
\}
\]

\[
S_1 = \{ S \rightarrow E.$  
    \ 
    E \rightarrow E + T \ 
\}
\]

\[
S_2 = \{ E \rightarrow T.$  
\}
\]

\[
S_3 = \{ T \rightarrow ID.$  
\}
\]

\[
S_4 = \{ T \rightarrow (E) \} \ -- \ but \ must \ do \ closure
\]
LR0 Sets (Page 2)

\[ S_4 = \{ \begin{align*}
T & \rightarrow ( \ . \ E ) \\
E & \rightarrow . \ E + T \\
E & \rightarrow . \ T \\
T & \rightarrow . \ ID \\
T & \rightarrow . \ ( \ E )
\end{align*} \} \]

This finishes up the go_to for \( S_0 \)!

\[ S_5 = \{ \begin{align*}
S & \rightarrow E \$ \ . \}
\end{align*} \} \] (From \( S_1 \))

\[ S_6 = \{ \begin{align*}
E & \rightarrow E + . \ T \\
E & \rightarrow E + . \ T \\
T & \rightarrow . \ ID \\
T & \rightarrow . \ ( \ E )
\end{align*} \} \] (From \( S_1 \)) (needs closure)

\[ S_6 = \{ \begin{align*}
E & \rightarrow E + . \ T \\
T & \rightarrow . \ ID \\
T & \rightarrow . \ ( \ E )
\end{align*} \} \]
LR0 Sets (Page 3)

\[ S_7 = \{ \ T \rightarrow ( \ E \ . ) \ \text{(From S}_4) \ \\
      E \rightarrow E \ . + T \ \} \]

\[ S_8 = \{ \ E \rightarrow E + T \ . \ \} \ \text{(From S}_6) \]

\[ S_9 = \{ \ T \rightarrow ( \ E \ ) \ . \ \} \ \text{(From S}_7) \]

Draw State diagram ....
Algorithm to Build CFSM

CFSM = Characteristic finite state machine

Algorithm Build_CFSM_LR0 (Grammar G)
   1) Let $S_0 = \text{closure}_{LR0}({S'} \rightarrow . S \, \$})$
   2) $S = \{ \, S_0 \, \}$
   3) While $S$ is not empty do
       remove set $s$ from $S$.
       for all $X$ in $s$ where $.X$ is part of a config
           if $\text{go}_\text{to}_{LR0} (s, X)$ is new,
               add $\text{go}_\text{to}_{LR0} (s, X)$ to $S$ with a new state number
               create a transition under $X$ from $s$ to $\text{go}_\text{to}_{LR0} (s, X)$
LR Parser tables

Build Action from information in CFSM

- Transitions are Shift
- \{ S' -> S . $ \} => Accept
- \{ A -> alpha . \} => reduce A -> alpha

Build Go_to table from CFSM

Basically the table form of the CFSM.
Go To Table for the example grammar

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>T</th>
<th>ID</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Action Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R3 R3 R3 R3 R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R4 R4 R4 R4 R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A A A A A A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>R2 R2 R2 R2 R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>R5 R5 R5 R5 R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example parse
  id + (id + id)

Errors in grammars

Shift-Reduce conflict

{ X -> ... ID .
  Y -> ... ID . XYZ }

Reduce-Reduce conflict

{ X -> ... ID .
  Y -> ... ID . }
SLR(0) parser tables

a) Compute LR(0) Sets

b) State i is constructed from S_i in the LR(0) Sets as:

   a) if  A -> alpha . a beta  in S_i
       and  goto(S_i,a) = S_j  then
       action [i, a] <-- Shift : j

   b) if  A -> alpha  .  in S_i  then
       action[i,a] <- Reduce A -> alpha, a in follow[A] (A != S')

   c) if  S' -> S . $  in S_i  then
       action[i,$] <-- Accept

   d) if  A -> alpha . B beta  in S_i
       and goto(S_i, B) = S_j  then
       goto[i,B] <-- j

   e) All other entries in action are error
LR(1) Parsing

Similar ideas except we add a "lookahead" to the items

\[ A \rightarrow X_1 \ldots X_i \cdot X_{i+1} \ldots X_j, \ a \ (a \ is \ terminal \ or \ lambda) \]

a is the lookahead at the end of the production!

May have many similar items with different lookaheads
May be written:

\[ A \rightarrow X_1 \ldots X_i \cdot X_{i+1} \ldots X_j, \ \{a_1, \ldots, a_m\} \]

Initial set looks like: (Before closure)

\[ \{ \ S' \rightarrow \cdot S \ \$, \ \{ \ \lambda \} \ \} \]
LALR(1)

- More powerful than SLR(1)
- Less powerful than LR(1)
- Complicated way to merge LR(1) configuration sets
- Ignore details ..... 
- On to other things !!!