Top Down vs Bottom Up parsers

Grammar problems for Top Down Parsers
☐ Top down parsers have problems with
  ☐ Common prefixes
    ☐ Stmt -> if Expr then StmtList end if
    ☐ Stmt -> if Expr then StmtList else StmtList end if
  ☐ Left recursion
    ☐ StmtList -> StmtList ; Stmt
    ☐ StmtList -> Stmt
  ☐ Issue: Predict function produces non-disjoint predict sets
☐ Solution for some issues is a grammar rewrite
  ☐ Read section 5.5 about LL(1) grammars
  ☐ LL(1) works for most programming language constructs
    ☐ except the dangling else problem of Java, C, C++ etc
      if (expr)
      if (expr)
      stmt
    else
      stmt
  ☐ Bottom Up parsers handle these easier. (e.g. grammar is ambiguous)
☐ Skip the remaining part of chapter 5
LR Parsers -- a practical bottom up parser

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

A grammar is LR(k) iff
- S =>*/rm alpha A w => alpha beta w
- S =>*/rm gamma B x => alpha beta y
- First_k(w) = First_k(y)
- Imply that alpha A w Equals gamma B x

Same context (alpha)
Same lookahead First_k(w), First_k(y)
=> Has to be the same rightmost derivation
LR(0) -- No Lookahead

Not a practical parser generator

Parser Construction
□ Based on the idea of a "configuration" or "item"
   A -> X_1 ... X_i . X_i+1 ... X_j
□ And on a set of items
   stmt -> ID . := expr
   stmt -> ID . : stmt
   stmt -> 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 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 -> E $
E -> E + T | T
T -> ID | ( E )

- Initial set, S_0
  \{ S -> . E $ \}

Algorithm Closure_LR0 (set S)
  repeat
    for all items B -> alpha . A beta in S, A in Variables
      add all items of the form A -> . 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_LLR0 ($S_0$):
$\{ S \rightarrow . \, E \, \$$
$E \rightarrow . \, E + T$
$E \rightarrow . \, T$
$T \rightarrow . \, ID$
$T \rightarrow . \, (E)$
$\}$
GoTo Algorithm

- Compute "successor" states from a state
- For an item: $A \rightarrow \alpha . X \beta$ a new set is started
- Based on the $\cdot X$ 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 . \text{ID} \\
T \rightarrow . ( \, E \, ) \, \} \\
$}

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

$S_2 = \{ E \rightarrow T \, . \, \} \\
$}

$S_3 = \{ T \rightarrow \text{ID} \, . \, \} \\
$}

$S_4 = \{ T \rightarrow ( \, . \, E \, ) \, \} -- \text{but must do closure}$
LR0 Sets (Page 2)

\[ S_4 = \{ \begin{array}{l}
T \rightarrow ( . E ) \\
E \rightarrow . E + T \\
E \rightarrow . T \\
T \rightarrow . ID \\
T \rightarrow . ( E ) \\
\end{array} \} \]

This finishes up the go_to for S_0!

\[ S_5 = \{ S \rightarrow E \, \$ \, \} \quad \text{(From S_1)} \]

\[ S_6 = \{ \begin{array}{l}
E \rightarrow E + . T \\
T \rightarrow . ID \\
T \rightarrow . ( E ) \\
\end{array} \} \quad \text{(From S_1) (needs closure)} \]

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

\[
S_7 = \{ \text{T -> ( E . ) (From S_4)}
\]
\[
\quad \text{E -> E . + T} \}
\]

\[
S_8 = \{ \text{E -> E + T .} \} \quad \text{(From S_6)}
\]

\[
S_9 = \{ \text{T -> ( E ).} \} \quad \text{(From S_7)}
\]

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 \( \text{go\_to\_LR0} (s, X) \) is new,
         add \( \text{go\_to\_LR0} (s, X) \) to \( S \) with a new state number
         create a transition under \( X \) from \( s \) to \( \text{go\_to\_LR0} (s, X) \)
LR Parser tables

Build Action from information in CFSM

- Transitions are Shift
- $\{ S' \rightarrow S . \$ \} \Rightarrow$ Accept
- $\{ A \rightarrow alpha . \} \Rightarrow$ reduce $A \rightarrow alpha$

Build Go_to table from CFSM

Basically the table form of the CFSM.
Go_To Table for the example grammar

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>E</th>
<th>T</th>
<th>ID</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td>$</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>S</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>S</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other things ...

Example parse
\[ \text{id} + (\text{id} + \text{id}) \]

Errors in grammars

Shift-Reduce conflict

\[
\begin{align*}
X & \rightarrow \ldots \text{ID} . \\
Y & \rightarrow \ldots \text{ID} . \text{XYZ}
\end{align*}
\]

Reduce-Reduce conflict

\[
\begin{align*}
X & \rightarrow \ldots \text{ID} . \\
Y & \rightarrow \ldots \text{ID} .
\end{align*}
\]
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 $A \to \alpha . a \beta$ in $S_i$
   and $\text{goto}(S_i, a) = S_j$ then
   \[ \text{action}[i, a] \leftarrow \text{Shift} : j \]

b) if $A \to \alpha \text{ in } S_i$ then
   \[ \text{action}[i, a] \leftarrow \text{Reduce } A \to \alpha, a \text{ in } \text{follow}[A] (A \neq S') \]

c) if $S' \to S . \$ \text{ in } S_i$ then
   \[ \text{action}[i, \$] \leftarrow \text{Accept} \]

d) if $A \to \alpha . B \beta$ in $S_i$
   and $\text{goto}(S_i, B) = S_j$ then
   \[ \text{goto}[i, B] \leftarrow j \]

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

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

A -> X_1 ... X_i . X_i+1 ... X_j, a (a is a terminal)

a is the lookahead at the end of the production!

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

A -> X_1 ... X_i . X_i+1 ... X_j, \{a_1, a_2, ... a_m\}

Initial set looks like: (Before closure)

\{ S' => . 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 !!!