LR parsing

- LR(k): deterministic bottom-up parsing using rightmost derivations with $k$-symbol lookahead.

eg: non-determinism in bottom-up parse

\[
S \rightarrow aAb \mid BaAa \\
A \rightarrow ab \mid b \\
B \rightarrow Bb \mid b
\]

input: $aabb \Rightarrow aAb \quad aaAb \quad aaBb$ ?
Towards determinism in BUP: How to locate what to reduce, and how to decide by which rule to do the reduction.

Recall that bottom-up shift-reduce parsers work on right sentential forms; they obtain these forms in reverse order.

- **handle**: use of a rule $A \rightarrow \beta$ at certain position in a right sentential form.

\[
S \xrightarrow{R^*} \alpha Aw \xrightarrow{R} \alpha\beta w
\]

($A \rightarrow \beta$ is a handle at position after $\alpha$; $w \in \Sigma^*$).
• There may be more than one handle if the grammar is ambiguous.

\[
S \rightarrow A \\
A \rightarrow T \mid A + T \\
T \rightarrow b \mid (A)
\]

input b+(b+b)  (T->b at positions 1,4,6)
T+(b+b)      (A->T at 0; T->b at 4,6)
A+(T+b)      (not S->A at 1; A->T at 4; T->b at 6)
A+(A+b)      (S->A at 4; T->b at 6; not S->A at 1)
A+(A+T)      (A->A+T at 4; A->T at 6; not S->A 1,4)
A+(A)        (S->A at 4; not S->A at 1; T->(A) at 3)
A+T          (A->T at 3; not S->A at 1; A->A+T at 1)
A
S

The ones that can lead to $S$ are handles.
• Bottom-up parsing can be seen as ’handle pruning’:

Start with input

locate a handle

use handle to reduce the sentential form

do until S is reached

• Problems: how to locate the handle; reduce by which rule?

Using a stack to keep sentential forms solves the first problem: the right end of the handle is always on top of the stack. The left end must be found.
Knowledge of context in which a rule can lead to a viable alternative solves the second problem: don’t use the rule if it’s context is not satisfied.

- **viable prefix (LR(0) context):** the set of prefixes of a right-sentential form that can appear on stack.

\[
S \xrightarrow{R}^* uAv \xrightarrow{A \Rightarrow w} uwv \quad v \in \Sigma^*
\]

\(uw\) is a viable prefix

rightmost derivations that terminate with the application of the rule.
Why LR(0) context: no look into $v$

- Viable prefixes may contain patterns if there's recursion in the grammar. Finding them on-the-fly is costly.

\[
S \rightarrow aA \mid bB \\
A \rightarrow abA \mid bB \\
B \rightarrow bBc \mid bc
\]

\[
S \Rightarrow aA \Rightarrow^* a(ab)^i A \Rightarrow a(ab)^i bB \Rightarrow^* a(ab)^i bb^j Bc^j \Rightarrow a(ab)^i bb^j+1 c^j+1
\]

\[
S \Rightarrow bB \Rightarrow^* bb^j Bc^j \Rightarrow bb^j+1 c^j+1
\]
LR0C(S → aA) = \{aA\}

LR0C(A → abA) = \{a(ab)^iA \mid i > 0\}

LR0C(B → bBc) = \{a(ab)^ib^{j+1}Bc, b^{j+1}Bc \mid i \geq 0, j > 0\}

why not \(c^j\) at the end? Viable prefix includes up to and including RHS of the rule.

- Solutions to the viable prefix problem:

  1. Shift-reduce parsing with stack search to locate the left-end of the handle.
2. Shift-reduce parsing with no stack search; design a recognizer to keep progress over the RHSs so that we can tell whether the handle is a viable prefix for a particular rule $\Rightarrow$ LR parsing

In the first alternative, the stack contains right sentential forms.

In the second alternative, the stack contains right sentential forms and states which give a summary of viable prefixes. No search in stack.
- $S \xrightarrow{R}^* uAv \xrightarrow{R} uwv \quad v \in \Sigma^*$

$A \rightarrow w$ is a handle at the end of $u$

$uw$ is the viable prefix (LR0C).
Only viable prefixes appear in the stack

- S/R parsing:
repeat

    if a handle is on top, reduce by the handle (pop)

    otherwise shift (put in stack)

until accept or input exhausts

\[ G_{AE} : \]

\[
S' \rightarrow S$
\]

\[
S \rightarrow A
\]

\[
A \rightarrow T \mid A + T
\]

\[
T \rightarrow b \mid (A)
\]

<table>
<thead>
<tr>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>b+(b+b)$ $</td>
<td>sh</td>
</tr>
<tr>
<td>$b$</td>
<td>+(b+b)$ $</td>
<td>red by T-&gt;b</td>
</tr>
</tbody>
</table>
\$T
\$A
\$A+ (b+b) \$
\$A+ (b+b) \$
\$A+ (b+b) \$
\$A+ (T
\$A+ (A
\$A+ (A+ b) \$
\$A+ (A+b) \$
\$A+ (A+T \$
\$A+ (A
\$A+ (A) \$
\$A+T
\$A
\$S
\$S$
\$S$
\$S’

A+A would not be a handle
- Efficiency of left-recursion vs. right-recursion in S/R parsing

\[
\begin{align*}
Xs & \rightarrow Xs \, , \, X \mid X \quad \text{vs.} \quad Xs & \rightarrow X \, , \, Xs \mid X \\
X & \rightarrow id \\
\end{align*}
\]

input: id1, id2, id3

- Reduce-reduce conflict shows the significance of the viable prefix problem. How to choose the right one?

  Shift-reduce conflict shows the significance of handle manipulation and structure of derivations. If the handle is not reduced, can we still get a derivation?

  eg.
\[ E \rightarrow E \ast E \mid E + E \mid \text{id} \]

<table>
<thead>
<tr>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>id1+id2*id3</td>
<td>sh</td>
</tr>
<tr>
<td>$id1</td>
<td>+id2*id3</td>
<td>r E-&gt;id1</td>
</tr>
<tr>
<td>$E</td>
<td>id2*id3</td>
<td>sh</td>
</tr>
<tr>
<td>$E+</td>
<td>id2*id3</td>
<td>sh</td>
</tr>
<tr>
<td>$E+id2</td>
<td>\ast id3</td>
<td>r E-&gt;id2</td>
</tr>
<tr>
<td>$E+E</td>
<td></td>
<td>conflict: shift or reduce?</td>
</tr>
</tbody>
</table>

- If shift-reduce conflict is not resolved by grammar re-writing, shift seems to work better for PL grammars.

\[ S \rightarrow \text{if E then S} \mid \text{if E then S else S} \]

input: if b then if S1 then S2 else S3
- Shift will favor innermost attachment of 'else'; reduce, outermost.

- Does the presence of conflicts mean non-LRness? not necessarily. If the grammar is ambiguous, it can’t be LR(k) for any k. But if the conflict is local, it may still be LR.

- Going from S/R parsing to LR parsing: keep track of ”progress” over the RHS of rules to select the right handle without a search for the
left-end of the handle.

- no look beyond the RHS $\Rightarrow$ Simple LR (SLR)

  lookahead $\Rightarrow$ LR(k) parsing

- We’ll see that SLR is more informative than simple LL.
• Simple LR: use of LR(0) items (no look past the RHS).

• construct a DFA for recognizing the progress over the RHSs

eg. $A \rightarrow \cdot B \cdot C \cdot a \cdot D \cdot$

$C \rightarrow \cdot B \cdot C \cdot$
SHIFT actions and GOTO transitions define the DFA for viable prefixes

SLR=SLR(1)       ACTION/GOTO tables look ahead one symbol
SLR= use of LR(0)−items with 1−symbol lookahead
• SLR table construction:

  Expand the grammar with $S' \to S\$$

  find LR(0) items

  find closure of items

  find set of items from closures

  construct the DFA from sets of items

  find FIRST and FOLLOW sets

  construct ACTION and GOTO tables

• LR(0) ITEM: progress of passing over the RHS of a rule
LR(0)-item\( (A \to uv) = \{ A \to \cdot uv, A \to u \cdot v, A \to uv \cdot \} \)

LR(0)-item\( (A \to \epsilon) = \{ A \to \cdot \} \)

e.g. for the grammar \( G_{AE} \):

\[
\begin{align*}
S & \to \cdot A \\
A & \to \cdot T \\
A & \to \cdot A + T \\
A & \to A + \cdot T \\
T & \to \cdot b \\
\end{align*}
\]

\[
\begin{align*}
S & \to A \cdot \\
A & \to T \cdot \\
A & \to A \cdot + T \\
A & \to A + T \cdot \\
T & \to b \cdot \\
\end{align*}
\]

\[
\begin{align*}
T & \to \cdot (A) \ldots
\end{align*}
\]

- CLOSURE OF an LR(0) ITEM: All the states of parsing that can be
reached from a state.

\[ \text{closure}(S \rightarrow \cdot A) = \{ S \rightarrow \cdot A, A \rightarrow \cdot T, A \rightarrow \cdot A + T, T \rightarrow \cdot b, T \rightarrow \cdot (A) \} \]

- \text{goto}(\text{item}, \text{symbol}) = \text{the set of states one can go from the item by consuming the symbol (note: this is not the GOTO table!)}

\[ \text{goto}(A \rightarrow A \cdot + T, +) = \{ A \rightarrow A + \cdot T, T \rightarrow \cdot b, T \rightarrow \cdot (A) \} \]

- \text{SETS OF ITEMS} = \text{possible states of parsing. Since there are finitely many RHS, there will be finitely many combinations.}
Find out where to go from $\text{goto}($item,$X$) for all $X \in V \cup \Sigma$. Add this to sets of items.

Each set is a possible state of SLR parsing (GOTO/ACTION states).

- eg.: $I_0 = \{S' \rightarrow \cdot S$, $S \rightarrow \cdot A$, $A \rightarrow \cdot T, \quad A \rightarrow \cdot A + T, T \rightarrow \cdot b, T \rightarrow \cdot (A)\}$

$I_1 = \text{goto}(I_0, S) = \{S' \rightarrow S \cdot \}$

$I_2 = \text{goto}(I_0, A) = \{S \rightarrow A \cdot, A \rightarrow A \cdot + T\}$

$I_3 = \text{goto}(I_0, T) = \{A \rightarrow T \cdot\}$

$I_4 = \text{goto}(I_0, b) = \{T \rightarrow b \cdot\}$
\[I_5 = goto(I_0, \cdot) = \{T \rightarrow (\cdot A), \ A \rightarrow \cdot T, \ A \rightarrow \cdot A + T, \ T \rightarrow \cdot b, \ T \rightarrow \cdot (A)\}\]
\[I_6 = goto(I_1, \$) = \{S' \rightarrow S\$.\}\]
\[I_7 = goto(I_2, +) = \{A \rightarrow A + \cdot T, \ T \rightarrow \cdot b, \ T \rightarrow \cdot (A)\}\]
\[I_8 = goto(I_5, A) = \{T \rightarrow (A\cdot), \ A \rightarrow A\cdot + T\}\]
\[\text{goto}(I_5, T) = I_3 \quad \text{goto}(I_5, b) = I_4 \quad \text{goto}(I_5, \cdot) = I_5\]
\[I_9 = goto(I_7, T) = \{A \rightarrow A + T\cdot\}\]
\[\text{goto}(I_7, b) = I_4 \quad \text{goto}(I_7, \cdot) = I_5 \quad \text{goto}(I_8, +) = I_7\]
\[I_{10} = goto(I_8, \$) = \{T \rightarrow (A)\cdot\}\]
The GOTO function as a FA

(GOTO table is a subset which only contains variable transitions)
• FOLLOW sets for $G_{AE}$

$\text{FOLLOW}(S) = \{\$\}$  \hspace{1cm}  \text{FOLLOW}(A) = \{), +, \$\}$

$\text{FOLLOW}(T) = \{\$, +, )\}$
• setting up the ACTION table

for each $I_i$, if $A \rightarrow \alpha \cdot a \cdot \beta$ is in $I_i$, $ACTION[i,a]=\text{shift } j$ (where $\text{goto}(I_i,a)=I_j$)

for each $A \rightarrow \alpha \cdot$, $ACTION[I_i,a]=\text{reduce by } A \rightarrow \alpha$ for all $a \in FOLLOW(A)$

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>b ( )</td>
<td>$+$</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>sh4</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sh7</td>
</tr>
<tr>
<td>3</td>
<td>rA-&gt;T</td>
</tr>
<tr>
<td>4</td>
<td>rT-&gt;b</td>
</tr>
</tbody>
</table>
5  sh4  sh5  
6  accept  
7  sh4  sh5  
8  sh10  sh7  
9  rA->A+T  rA->A+T  rA->A+T  
10 rT->(A)  rT->(A)  rT->(A)
parsing \( (b+b) \) with SLR table

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( (b+b) ) $</td>
<td>sh5</td>
</tr>
<tr>
<td>0 (5</td>
<td>( b+b ) $</td>
<td>sh4</td>
</tr>
<tr>
<td>0 (5b4</td>
<td>+b) $</td>
<td>rT-&gt;b</td>
</tr>
<tr>
<td>0 (5T3</td>
<td></td>
<td>rA-&gt;T</td>
</tr>
<tr>
<td>0 (5A8</td>
<td></td>
<td>sh7</td>
</tr>
<tr>
<td>0 (5A8+7</td>
<td>b) $</td>
<td>sh4</td>
</tr>
<tr>
<td>0 (5A8+7b4</td>
<td>$</td>
<td>rT-&gt;b</td>
</tr>
<tr>
<td>0 (5A8+7T9</td>
<td>$</td>
<td>rA-&gt;A+T</td>
</tr>
<tr>
<td>0 (5A8</td>
<td></td>
<td>sh10</td>
</tr>
<tr>
<td>0 (5A8) 10</td>
<td>$</td>
<td>rT-&gt;(A)</td>
</tr>
<tr>
<td>0T3</td>
<td></td>
<td>rA-&gt;T</td>
</tr>
<tr>
<td>0A2</td>
<td></td>
<td>rS-&gt;A</td>
</tr>
<tr>
<td>0S1</td>
<td></td>
<td>accept</td>
</tr>
</tbody>
</table>

- A grammar with no conflict in the SLR tables is SLR(1).
- Reductions and new top determine the next viable prefix.

- In general, direct DFA construction is cumbersome. Design a NFA with empty transitions and convert to DFA.

- General LR(k) parsing is an extension of SLR where LR(k)-items are used.