LR(k) Grammars

- **LR(k)** = Left-to-right scanning, Right-most derivation, $k$ look-ahead characters

- Main cases: **LR(0)**, **LR(1)**, **SLR(k)**, and **LALR(1)**

- Parsers for **LR(0)** Grammars:
  - Know whether to shift or reduce without consulting the lookahead symbol
  - Give intuition and techniques relevant for creating parsers for all grammar classes to be considered
Building LR(0) Parsing Tables

• To build the parsing table:
  – Define states of the parser
  – Build a DFA to describe the transitions between states
  – Use the DFA to build the parsing table
Viable Prefix

• $\gamma$ is a viable prefix for $G$ iff there is some derivation
  $$S \Rightarrow^* \alpha A \gamma \Rightarrow \alpha \beta \gamma$$
  where $\gamma$ is a prefix of $\alpha \beta$

• $\{\gamma \mid \gamma$ is a viable prefix of $G\}$ is a regular language, i.e., it can be recognized by a DFA known as the Canonical LR(0) Machine
Viable Prefix (Informally)

- $\gamma$ is a **viable prefix** for $G$ if it is a prefix of a sentential form derived from $S$ that does not extend past the end of the handle of the sentential form.

\[
\begin{align*}
S & \Rightarrow* \alpha Az \Rightarrow \alpha \beta z
\end{align*}
\]
LR(0) Items

• An LR(0) item for G is a triple $\langle A, \beta_1, \beta_2 \rangle$ such that $A \rightarrow \beta_1 \beta_2$ is a production of G. The item $\langle A, \beta_1, \beta_2 \rangle$ is denoted by $[A \rightarrow \beta_1 \beta_2]$
Validity of LR(0) Items

• The item \([A \rightarrow \beta_1 . \beta_2]\) is valid for viable prefix \(\alpha \beta_1\) iff \(S \Rightarrow^* \alpha Az \Rightarrow \alpha \beta_1 \beta_2 z\)

• Note:
  - \(\beta_1\) may be \(\varepsilon\)
  - \(\beta_2\) may be \(\varepsilon\)

• For any viable prefix \(\alpha\), let \(V(\alpha)\) denote the set of LR(0) items that are valid for \(\alpha\).
Sets of Valid Items

• Observations
  – There are only finitely many distinct LR(0) items for a given G.
  – Thus, there are only finitely many sets of LR(0) items for G.

• Sets of valid items for viable prefixes of G will serve as the states of a DFA, i.e., the canonical LR(0) machine.
Relation ↓

- The relation ↓ on LR(0) items is defined by $I \downarrow I'$ iff $\exists A, B, \beta_1, \beta_2, \beta_3$ such that
  
  
  \[
  I = [A \rightarrow \beta_1 B \beta_3]
  \]

  \[
  I' = [B \rightarrow \beta_2]
  \]

- **Lemma.** Let $I, I'$ be as above. If $I \in V(\alpha \beta_1)$ and $I \downarrow I'$, then $I' \in V(\alpha \beta_1)$.

  - $I \in V(\alpha \beta_1)$ implies $S \Rightarrow^* \alpha Az \Rightarrow \alpha \beta_1 B \beta_3 z$
  
  - Assuming $G$ has no useless productions, $\exists y$ such that $\beta_3 \Rightarrow^* y$
  
  - Thus, $S \Rightarrow^* \alpha Az \Rightarrow \alpha \beta_1 B \beta_3 z \Rightarrow^* \alpha \beta_1 Byz \Rightarrow \alpha \beta_1 \beta_2 yz$
  
  - Thus, $I'$ (i.e., $[B \rightarrow \beta_2]$) $\in V(\alpha \beta_1)$
Relation $\rightarrow_x$

- For any $X \in (V \cup \Sigma)$, the relation $\rightarrow_x$ is defined by $I \rightarrow_x I'$ iff $\exists A, \beta_1, \beta_3$ such that
  
  $I = [A \rightarrow \beta_1X\beta_3]$
  
  $I' = [A \rightarrow \beta_1X\beta_3]$

- **Lemma.** Let $I$, $I'$ be as above. If $I \in V(\alpha\beta_1)$ then $I' \in V(\alpha\beta_1X)$.

  - $I = [A \rightarrow \beta_1X\beta_3] \in V(\alpha\beta_1)$ implies
    
    $S \Rightarrow^* \alpha Az \Rightarrow \alpha\beta_1X\beta_3z$

    which by definition means $I' (= [A \rightarrow \beta_1X\beta_3]) \in V(\alpha\beta_1X)$
Technical Details

• Start symbol never appears on RHS
  – It is convenient if the start symbol never appears on the RHS of any production.
  – Given $G = \langle V, \Sigma, S, \rightarrow \rangle$, let $S' \notin V$ and
    
    $G' = \langle V, \Sigma, S', \rightarrow \cup \{S' \rightarrow S\} \rangle$
  
  – Assume that the grammars we work with have the form of $G'$.

• If $S$ is a set and $R$ is a relation, then
  
  $SR = \{y | x \in S \text{ and } \langle x, y \rangle \in R\}$

  $SR$ is called $S$ mapped by $R$
V(ε), the base case

• Let \( S' \) be the start symbol of \( G \). Then
  
  \[
  V(\varepsilon) = \{ [S' \rightarrow \varepsilon.S] \} \downarrow^*
  \]

  (i.e., the “initial item” of \( G \) \([S' \rightarrow \varepsilon.S]\) mapped by the reflexive transitive closure of the \( \downarrow \) relation.)

• If \( Q \) is a set of items, we call \( Q \downarrow^* \) the closure(\( Q \)).
V(αX), the inductive case

• For any α and X, \( V(\alpha X) = V(\alpha) \rightarrow_{x}^{*} \)

• For any set Q of items, we call \( Q \rightarrow_{x}^{*} \) the X-successor of Q, or Goto(Q,X).
Canonical LR(0) Machine

- **States**: Sets of valid items
- **Transition function**: Goto, as defined above.
- **Algorithm**: To compute all sets of valid items

\[
\text{STATES} := \mathcal{V}(\varepsilon)
\]

\[
\text{while } \exists \, Q \in \text{STATES}, \, X \in (\mathcal{V} \cup \Sigma) \text{ such that } \text{Goto}(Q,X) \notin \text{STATES} \\
\text{do } \text{STATES} := \text{STATES} \cup \{ \text{Goto}(Q,X) \}
\]

- Clearly, this terminates, as STATES is bounded above by the Powerset(LR(0) items)
LR(0) Grammar

- Nested lists:
  \[ S \rightarrow (L) | \text{id} \]
  \[ L \rightarrow S | L, S \]

- Sample strings
  - \((a, b, c)\)
  - \(((a, b), (c, d), (e, f))\)
  - \((a, (b, c, d), ((f, g)))\)

Parse tree for \((a, (b, c), d)\)
Start State

• Start state

  - \( V(\varepsilon) = \{ [ S' \rightarrow .S ] \}^* \)
    
    \[ = \{ [ S' \rightarrow .S ] [ S \rightarrow .(L) ], [ S \rightarrow .id ] \} \]

• Closure of a parser state \( Q \):

  - Start with \( \text{Closure}(Q) := Q \)
  - Then for each item in \( Q \):
    
    \( A \rightarrow \alpha.B\beta \)

    add the items for all the productions \( B \rightarrow \gamma \) to the closure of \( Q \):

    \( B \rightarrow .\gamma \)
Goto: Terminal Symbols

In new state, include all items that have appropriate input symbol just after dot, advance dot in those items, and take closure.
Goto: Nonterminal Symbols

(same algorithm for transitions on nonterminals)
Reduce States

\[
\begin{align*}
S' &\rightarrow S \\
S &\rightarrow (L) \\
S &\rightarrow \text{id} \\
L &\rightarrow S \\
L &\rightarrow S, S \\
S' &\rightarrow S \\
S &\rightarrow \text{id} \\
S &\rightarrow (L) \\
S &\rightarrow \text{id} \\
L &\rightarrow S \\
L &\rightarrow S, S \\
S &\rightarrow \text{id} \\
S &\rightarrow (L) \\
L &\rightarrow S \\
L &\rightarrow S, S \\
\end{align*}
\]
Full LR(0) Machine

Grammar:

\[
S \rightarrow (L) | \text{id} \\
L \rightarrow S | L,S 
\]
## Parsing Example: ((a),b)

<table>
<thead>
<tr>
<th>derivation</th>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a),b) ⇐</td>
<td>1</td>
<td>((a),b)</td>
<td>shift, goto 3</td>
</tr>
<tr>
<td>((a),b) ⇐</td>
<td>13</td>
<td>(a),b</td>
<td>shift, goto 3</td>
</tr>
<tr>
<td>((a),b) ⇐</td>
<td>133</td>
<td>a),b)</td>
<td>shift, goto 2</td>
</tr>
<tr>
<td>((a),b) ⇐</td>
<td>1332</td>
<td>),b)</td>
<td>reduce $S \rightarrow \text{id}$</td>
</tr>
<tr>
<td>((S),b) ⇐</td>
<td>1337</td>
<td>),b)</td>
<td>reduce $L \rightarrow S$</td>
</tr>
<tr>
<td>((L),b) ⇐</td>
<td>1335</td>
<td>),b)</td>
<td>shift, goto 6</td>
</tr>
<tr>
<td>((L),b) ⇐</td>
<td>13356</td>
<td>,b)</td>
<td>reduce $S \rightarrow (L)$</td>
</tr>
<tr>
<td>(S,b) ⇐</td>
<td>137</td>
<td>,b)</td>
<td>reduce $L \rightarrow S$</td>
</tr>
<tr>
<td>(L,b) ⇐</td>
<td>135</td>
<td>,b)</td>
<td>shift, goto 8</td>
</tr>
<tr>
<td>(L,b) ⇐</td>
<td>1358</td>
<td>b)</td>
<td>shift, goto 9</td>
</tr>
<tr>
<td>(L,b) ⇐</td>
<td>13582</td>
<td>)</td>
<td>reduce $S \rightarrow \text{id}$</td>
</tr>
<tr>
<td>(L,S) ⇐</td>
<td>13589</td>
<td>)</td>
<td>reduce $L \rightarrow L, S$</td>
</tr>
<tr>
<td>(L) ⇐</td>
<td>135</td>
<td>)</td>
<td>shift, goto 6</td>
</tr>
<tr>
<td>(L) ⇐</td>
<td>1356</td>
<td>)</td>
<td>reduce $S \rightarrow (L)$</td>
</tr>
<tr>
<td>S</td>
<td>14</td>
<td></td>
<td>done</td>
</tr>
</tbody>
</table>

### Grammar:

$$S \rightarrow (L) \mid \text{id}$$  
$$L \rightarrow S \mid L, S$$
Reductions

• On reducing $B \rightarrow \beta$ with stack $\alpha \beta_2$:
  - pop $|\beta|$ states off stack
  - This reveals topmost state $Q$, which contains an item $[A \rightarrow \beta_1 \cdot B \beta_3]$
  - push state $\text{Goto}(Q,B)$ onto the stack
### LR(0) Parsing Table

<table>
<thead>
<tr>
<th></th>
<th>(</th>
<th>)</th>
<th>id</th>
<th>,</th>
<th>ε</th>
<th>S</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
<td>g4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>S→id</td>
<td>S→id</td>
<td>S→id</td>
<td>S→id</td>
<td>S→id</td>
<td></td>
<td>g7</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>s6</td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td>S→(L)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td>L→S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>s3</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>g9</td>
</tr>
<tr>
<td>8</td>
<td>L→L,S</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>

**Rules:**
- **S:**
  - S→id
  - S→id
  - S→id
  - S→id
  - S→id
- **L:**
  - L→S
  - L→S
  - L→S
  - L→S
  - L→S
LR(0) Summary

- LR(0) parsing recipe:
  Start with an LR(0) grammar
  Compute LR(0) states and build DFA:
  Build the LR(0) parsing table from the DFA
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 more complex grammars, construction gives states with shift/reduce or reduce/reduce conflicts.
• Need to use look-ahead to choose.
A Non-LR(0) Grammar

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

- Left-associative is LR(0)

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

Grammar
S → E + S | E
E → num | ( S )

What to do in state 2?
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 → β₁ . bβ₃], with \( u ∈ \text{FIRST}_k(bβ₃ \text{FOLLOW}_k(A)) \)
  - Action(Q,u) = accept
    - if Q = \{ [S’ → S ] \} and u=ε
  - Action(Q,u) = reduce i
    - if Q contains the item [A → βₖ], where A → β is the \( i \text{th} \) production of G and \( u ∈ \text{FOLLOW}_k(A) \)
  - 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.
Next Time

• Learn about other kinds of LR parsing:
  – SLR = improved LR(0)
  – LR(1) = 1 character lookahead
  – LALR(1) = Look-Ahead LR(1)

• Basic ideas are the same as for LR(0)
  – Parser states with LR items
  – DFA with transitions between parser states
  – Parser table with shift/reduce/goto actions