CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Phases of a compiler

Figure 1.7, page 5 of text

Syntactic structure
LR(k)

- LR(k) parser
- L → left-to-right scanning of input
- R → rightmost derivation in reverse
- k → number of lookahead symbols
- k is typically 0 or 1
- LR → LR(1)
"The LR-parsing method is the most general nonbacktracking shift-reduce parsing method known."

"[The LR-parsing method] can be implemented as efficiently as other [...] shift-reduce methods."

"An LR parser can detect a syntactic error as soon as it is possible to do so on a left-to-right scan of the input."

"The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with predictive or LL methods."
LR(0) automaton and SLR

- SLR => Simple LR
- LR(0) automaton is constructed from G'
- "Suppose that the string \( \gamma \) of grammar symbols takes the LR(0) automaton from the start state 0 to some state \( j \). Then, shift on next input symbol \( a \) if state \( j \) has a transition on \( a \). Otherwise, we choose to reduce; the items in state \( j \) will tell us which production to use." [p 247]
Figure 4.35 [p. 248]
In the SLR method, the stack holds states from the LR(0) automaton; the canonical LR and LALR methods are similar.
"By construction, each state has a corresponding grammar symbol. Recall that states correspond to sets of items, and that there is a transition from state \( i \) to state \( j \) if \( \text{GOTO}(I_i, X) = I_j \). All transitions to state \( j \) must be for the same grammar symbol \( X \). Thus, each state, except the start state 0, has a unique grammar symbol associated with it." [p. 248]
LR parsing table

**ACTION function**
- Inputs: state \( i \) and an input symbol \( a \) (terminal or \( \$ \))
- \( \text{ACTION}[i,a] \) is:
  - Shift \( j \) - shift \( a \) onto stack, using state \( j \) to represent \( a \)
  - Reduce \( A \rightarrow \beta \)
  - Accept
  - Error

**GOTO function** - extend from sets of items to states.
- \( \text{GOTO}[I_i,A] = I_j \Rightarrow \text{GOTO}[i,A] = j \)
LR parser configuration

- An LR parser configuration is a pair:
  \[(s_0 s_1 \ldots s_m, a_i a_{i+1} \ldots a_n \$)\]
  - \(s_0 s_1 \ldots s_m\) is the stack (bottom to top)
  - \(a_i a_{i+1} \ldots a_n \$\) is the (remaining) input

- Represents the right-sentential form
  \[X_1 X_2 \ldots X_m a_i a_{i+1} \ldots a_n\]
Algorithm 4.44 [p. 250-251]  
The LR-parsing algorithm

INPUT: An input string $w$ and an LR-parsing table with functions ACTION and GOTO for a grammar $G$

OUTPUT: If $w$ is in $L(G)$, the reduction steps of a bottom-up parse for $w$; otherwise, an error indication

METHOD: Initially, the parser has $s_0$ on its stack, where $s_0$ is the initial state. The parser then executes the program in Fig. 4.36.
let $a$ be the first symbol of $w$

while (true) {
    let $s$ be the state on top of the stack
    if ($\text{ACTION}[s,a] = \text{shift } t$) {
        push $t$ onto the stack
        let $a$ be the next input symbol
    } else if ($\text{ACTION}[s,a] = \text{reduce } A \rightarrow \beta$) {
        pop $|\beta|$ symbols off the stack
        let state $t$ now be on top of the stack
        push $\text{GOTO}[t,A]$ onto the stack
        output the production $A \rightarrow \beta$
    } else if ($\text{ACTION}[s,a] = \text{accept}$) break
    else call error-recovery routine
}
Algorithm 4.46 [p. 253]
Constructing an SLR-parsing table

INPUT: An augmented grammar $G'$
OUTPUT: The SLR-parsing table functions $\text{ACTION}$ and $\text{GOTO}$ for $G'$

METHOD:
1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for $G'$
2. State $i$ is constructed from $I_i$. The parsing items for state $i$ are determined as follows:
   A. If $[A \rightarrow \alpha \cdot a \beta]$ is in $I_i$ and $\text{GOTO}(I_i,a) = I_j$, then set $\text{ACTION}[i,a]$ to "shift j". Here $a$ must be a terminal.
   B. If $[A \rightarrow \alpha \cdot]$ is in $I_i$, then set $\text{ACTION}[i,a]$ to "reduce $A \rightarrow \alpha$" for all $a$ in $\text{FOLLOW}(A)$; here $A$ may not be $S'$.
   C. If $[S' \rightarrow S \cdot]$ is in $I_i$, then set $\text{ACTION}[i,\$]$ to "accept."

If conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.
3. The goto transitions for state $I$ are constructed for all nonterminals $A$ using the rule: If $\text{GOTO}(I_i,A) = I_j$, then $\text{GOTO}[i,A] = j$.
4. All entries not defined by rules (2) and (3) are made "error".
5. The initial state of the parser is the one constructed from the set of items containing $[S' \rightarrow S \cdot]$
### Parsing table for expression grammar

<table>
<thead>
<tr>
<th>STATE</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id</td>
<td>s5</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>s6</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>s7</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>r2</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>r4</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>s6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>s5</td>
</tr>
<tr>
<td>+</td>
<td>s6</td>
</tr>
<tr>
<td>*</td>
<td>s7</td>
</tr>
<tr>
<td>(</td>
<td>r2</td>
</tr>
<tr>
<td>)</td>
<td>r4</td>
</tr>
<tr>
<td>$</td>
<td>s6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GOTO</th>
<th>E</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Textbook Typo:

- On page 254, line-4:
- "Fig. 4.31" should be "Fig. 4.37".
Workshop Wednesdays

- Teams should come to class with questions.
- Teams sit together and discuss project.
- I will circulate to teams to discuss project progress and answer questions.
Make sure you work through remainder of chapter 4, especially 4.9.

Also consult sections you read for HW1, such as 2.7.

Symbol table structure and functionality will need to be adapted to meet changing needs as project continues. Write your code with growth/change in mind.