Phases of a compiler

Syntactic structure

Figure 1.7, page 5 of text
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." [p. 248]
"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."
**LR parsing table**

**ACTION function**
- Inputs: state \( i \) and an input symbol \( a \) (terminal or $)
- 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.
- GOTO\([I_i,A]\) = \( I_j \) => GOTO\([i,A]\) = \( j \)
LR parser configuration

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

- Represents the right-sentential form
  \[X_1X_2 \ldots X_m a_ia_{i+1} \ldots a_n\]
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 (ACTION[s,a] = shift t) {
        push t onto the stack
        let a be the next input symbol
    } else if (ACTION[s,a] = reduce A -> β) {
        pop |β| symbols off the stack
        let state t now be on top of the stack
        push GOTO[t,A] onto the stack
        output the production A -> β
    } else if (ACTION[s,a] = 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 ACTION and GOTO for G'  

METHOD:  
1. Construct $C = \{I_0, I_1, ..., I_n\}$, 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 $[S' \rightarrow S \cdot]$
\textbf{FIRST}(X)

- if \( X \in T \) then \( \text{FIRST}(X) = \{ X \} \)

- if \( X \in N \) and \( X \rightarrow Y_1 \ Y_2 \ldots Y_k \in P \) for \( k \geq 1 \), then
  - add \( a \in T \) to \( \text{FIRST}(X) \) if \( \exists i \) s.t. \( a \in \text{FIRST}(Y_i) \) and \( \varepsilon \in \text{FIRST}(Y_j) \ \forall j < i \) (i.e. \( Y_1 \ Y_2 \ldots Y_k \Rightarrow^* \varepsilon \))

- if \( \varepsilon \in \text{FIRST}(Y_j) \ \forall j < k \) add \( \varepsilon \) to \( \text{FIRST}(X) \)
FOLLOW(X)

- Place $ in FOLLOW(S), where $ is the start symbol ($ is an end marker)
- if $A \rightarrow \alpha B\beta \in \mathcal{P}$, then $\text{FIRST}(\beta) - \{\varepsilon\}$ is in FOLLOW(B)
- if $A \rightarrow \alpha B \in \mathcal{P}$ or $A \rightarrow \alpha B\beta \in \mathcal{P}$ where $\varepsilon \in \text{FIRST}(\beta)$, then everything in FOLLOW(A) is in FOLLOW(B)
## FIRST(X) and FOLLOW(X)

<table>
<thead>
<tr>
<th>X</th>
<th>FIRST(X)</th>
<th>FOLLOW(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S'$</td>
<td>id, (</td>
<td>$</td>
</tr>
<tr>
<td>$E$</td>
<td>id, (</td>
<td>$+, ), $</td>
</tr>
<tr>
<td>$T$</td>
<td>id, (</td>
<td>$*, +, ), $</td>
</tr>
<tr>
<td>$F$</td>
<td>id, (</td>
<td>$*, +, ), $</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>$*, +, ), $</td>
</tr>
<tr>
<td>(</td>
<td>(</td>
<td>id, (</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>$*, +, ), $</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>id, (</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>id, (</td>
</tr>
</tbody>
</table>
2A. If \([A \rightarrow \alpha \cdot a \beta] \) is in \(I_i\) and \(GOTO(I_i, a) = I_j\), then set \(ACTION[i, a]\) to "shift \(j\)". Here \(a\) must be a terminal.

2B. If \([A \rightarrow \alpha\cdot]\) is in \(I_i\), then set \(ACTION[i, a]\) to "reduce \(A \rightarrow \alpha\)" for all \(a\) in \(FOLLOW(A)\); here \(A\) may not be \(S'\).

2C. If \([S' \rightarrow S\cdot]\) is in \(I_i\), then set \(ACTION[i, \$]\) to "accept."

3. The goto transitions for state \(I\) are constructed for all nonterminals \(A\) using the rule: If \(GOTO(I_i, A) = I_j\), then \(GOTO[i, A] = j\).

Production numbers:

1. \(E \rightarrow E + T\)
2. \(E \rightarrow T\)
3. \(T \rightarrow T \cdot F\)
4. \(T \rightarrow F\)
5. \(F \rightarrow ( E )\)
6. \(F \rightarrow id\)
**Figure 4.37 [p. 252]**

**Parsing table for expression grammar**

<table>
<thead>
<tr>
<th>STATE</th>
<th>ACTION</th>
<th>GOTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>s6</td>
<td>s4</td>
</tr>
<tr>
<td>1</td>
<td>s6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>r2</td>
<td>s7</td>
</tr>
<tr>
<td>3</td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s6</td>
<td>s4</td>
</tr>
<tr>
<td>5</td>
<td>r6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>r6</td>
<td></td>
</tr>
</tbody>
</table>

**Production numbers:**
1: E → T + T
2: E → T * T
3: T → T + T
4: T → T * T
5: T → id
6: F → ( E )
7: F → id

When we reduce by E → T we enter 'r2' in the table, regardless of which state we're in.

At a later point, we reduce by F → id in state 5. The correct entry for those cells is 'r6'.

Lecture goof today:
Hi everyone,
I realized, right at the end of class, that I goofed in describing the number used for reductions in the table. When we first encountered the reduction I said it was the state number, which in that example was 2. '2' was correct, but it's not the state number - it's the number of the reducing production in our grammar.
### Figure 4.37 [p. 252]
#### Parsing table for expression grammar

<table>
<thead>
<tr>
<th>STATE</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>id</td>
</tr>
<tr>
<td>1</td>
<td>s6</td>
</tr>
<tr>
<td>2</td>
<td>r2</td>
</tr>
<tr>
<td>3</td>
<td>r4</td>
</tr>
<tr>
<td>4</td>
<td>s6</td>
</tr>
<tr>
<td>5</td>
<td>r6</td>
</tr>
<tr>
<td>6</td>
<td>s6</td>
</tr>
<tr>
<td>7</td>
<td>s5</td>
</tr>
<tr>
<td>8</td>
<td>s6</td>
</tr>
<tr>
<td>9</td>
<td>r1</td>
</tr>
<tr>
<td>10</td>
<td>r3</td>
</tr>
<tr>
<td>11</td>
<td>r5</td>
</tr>
</tbody>
</table>

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

- On page 254, line-4:

  "Fig. 4.31" should be "Fig. 4.37".
Project 2 notes

- 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.