Plan for Today

Review main idea syntax-directed evaluation and translation

Recall syntax-directed interpretation in recursive descent parsers

Syntax-directed evaluation and translation in shift-reduce parser
– Performs a right-most derivation in reverse
– Parsing unambiguous grammars
– Parsing ambiguous grammars using precedence and associativity rules
– LR parsing table algorithm
– LR parsing algorithm and syntax-directed evaluation and translation

Syntax-directed code generation for PA3

Parse Tree Example
Semantic Rules for Expression Example (JavaCUP)

Semantic Rules for Expression Example (book notation)
Recall recursive-descent parsing

```java
private void parse_elem() throws IOException, ParseException {
    switch (this.m_lookahead.tag) {
        // elem ->
        // RECT_START KW_X EQ NUM KW_Y EQ NUM KW_WIDTH EQ NUM KW_HEIGHT EQ NUM
        // KW_FILL EQ COLOR ELEM_END
        case RECT_START:
            this.match(Token.Tag.RECT_START);
            this.match(Token.Tag.KW_X);
            this.match(Token.Tag.EQ);
            Num x = (Num) this.match(Token.Tag.NUM);
            this.match(Token.Tag.KW_Y);
            this.match(Token.Tag.EQ);
            Num y = (Num) this.match(Token.Tag.NUM);
            this.match(Token.Tag.KW_WIDTH);
            this.match(Token.Tag.EQ);
            Num width = (Num) this.match(Token.Tag.NUM);
            this.match(Token.Tag.KW_HEIGHT);
            this.match(Token.Tag.EQ);
            Num height = (Num) this.match(Token.Tag.NUM);
            this.match(Token.Tag.KW_FILL);
            this.match(Token.Tag.EQ);
            Color color = (Color) this.match(Token.Tag.COLOR);
            this.match(Token.Tag.ELEM_END);
            this.m_drawer.draw_rect(x.value, y.value, width.value,
                                    height.value, color.value);
            this.m_reporter.report_rect(x.value, y.value, width.value,
                                         height.value, color.value);
            break;
    }
}
```

Recall Plan for Today

Review main idea syntax-directed evaluation and translation

Recall syntax-directed interpretation in recursive descent parsers

Syntax-directed evaluation and translation in shift-reduce parser
  – Performs a right-most derivation in reverse
  – Parsing unambiguous grammars
  – Parsing ambiguous grammars using precedence and associativity rules
  – LR parsing table algorithm
  – LR parsing algorithm and syntax-directed evaluation and translation

Syntax-directed code generation for PA3
**Shift-reduce parsing**

**Form of bottom-up parsing**
- Implicit parse tree is built bottom up
- Order is equivalent to a reverse right-most derivation
- A reduction is the reverse of a derivation step

**How it works**
- Start with a stack and a token stream as input
- Based on the symbols at the top of the stack and the next token in input perform one of the following actions:
  - Shift: consume the token from input and push onto stack
  - Reduce: replace right-hand side of a production rule on the top of the stack with the left-hand side nonterminal for that rule
  - Accept: indicate that parsing is complete
  - Error: indicate there is a parsing error

**Shift reduce parsing applied to unambiguous grammars, ex1**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>S' -&gt; E $</td>
</tr>
<tr>
<td>(2)</td>
<td>E -&gt; E + T</td>
</tr>
<tr>
<td>(3)</td>
<td>E -&gt; T</td>
</tr>
<tr>
<td>(4)</td>
<td>T -&gt; T * F</td>
</tr>
<tr>
<td>(5)</td>
<td>T -&gt; F</td>
</tr>
<tr>
<td>(6)</td>
<td>F -&gt; ( E )</td>
</tr>
<tr>
<td>(7)</td>
<td>F -&gt; NUM</td>
</tr>
</tbody>
</table>
Shift reduce parsing applied to unambiguous grammars, ex2

Start symbol is $S'$

- $S \rightarrow ( S )$
- $S' \rightarrow S \text{ EOF}$
- $S \rightarrow \text{ ID}$

Shift reduce parsing applied to ambiguous grammars, ex1

1. $S' \rightarrow E \ $  
2. $E \rightarrow E + E$  
3. $E \rightarrow E * E$  
4. $E \rightarrow ( E )$  
5. $E \rightarrow \text{ NUM}$
Shift reduce parsing applied to ambiguous grammars, ex2

1. S' -> E $
2. E -> E + E
3. E -> ( byte ) E
4. E -> NUM

Shift-reduce parsing in an LR parser

LR(k) parser
- Left-to-right parse
- Right-most derivation
- K-token look ahead

LR parsing algorithm
- Performs a shift-reduce parse
- Look at state at top of stack and input symbol to find action in table
  - shift(n): advance input, push state n on stack
  - reduce(k): pop rhs of grammar rule k, look up state on top of stack
    and lhs for goto n, push lhs(k) and n onto stack
  - accept: stop and success
  - error: stop and fail
### Example LR Parse Table, Single Paren Nest

[0] \( S \rightarrow ( S ) \)
[1] \( S' \rightarrow S \text{ EOF} \)
[2] \( S \rightarrow \text{ID} \)

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s3</td>
<td>s1</td>
</tr>
<tr>
<td>1</td>
<td>r2</td>
<td>r2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>s1</td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>r0</td>
<td>r0</td>
</tr>
</tbody>
</table>

- **Action**: ( for left parenthesis, ) for right parenthesis, $$/EOF$$ for end of input, **ID** for identifier, **S** for start symbol.
- **Goto**: States to transition to.