LR Parsing

Lecture 9

Dr. Sean Peisert – ECS 142 – Spring 2009
Status

- Project 1 Back Soon
- Project 2 Due Friday, Apr. 24, 11:55pm
Functions

• We have defined several functions to help us on the way:

• closure (s) - where s is a state

• goto (S, X) where S is a set of LR items

• First (X)

• Follows (X)

• ...and for all cases X is a grammar symbol (either terminal or non-terminal)
Algorithms

• We have defined several algorithms which take us from the CFG to the decision table and the process which walks us through the parsing.

• CFG $\rightarrow$ DFA

• Fill in the action/goto entries in decision table

• Using the DFA, a stack and action/goto tables, parse a grammar.
Example Grammar

\[
\begin{align*}
S & \rightarrow Xb \\
X & \rightarrow YZ \\
Y & \rightarrow aa \\
Z & \rightarrow cS \\
Z & \rightarrow \varepsilon
\end{align*}
\]
CFG $\rightarrow$ DFA

- Start by adding a new state to the grammar

$$S' \rightarrow \cdot S$$

Take the new state, add closure to get the new start state, which is state 1.
Example Grammar

rule 1: $S' \rightarrow S$
rule 2: $S \rightarrow Xb$
rule 3: $X \rightarrow YZ$
rule 4: $Y \rightarrow aa$
rule 5: $Z \rightarrow cS$
rule 6: $Z \rightarrow \varepsilon$
State 0: Closure of S

\[ S' \rightarrow \bullet S \]

\[ S \rightarrow \bullet Xb \]
State 0: Closure of X

\[ S' \rightarrow \cdot S \]
\[ S \rightarrow \cdot Xb \]
\[ X \rightarrow \cdot YZ \]
State 0: Closure of Y

\[ S' \rightarrow \bullet S \]
\[ S \rightarrow \bullet Xb \]
\[ X \rightarrow \bullet YZ \]
\[ Y \rightarrow \bullet aa \]
Loop through the states

• State 0
  • mark it
  • go through all the grammar symbols (S, X, Y, Z, a, b, c). If the state expects any of these symbols (i.e., the symbol precedes a “•”), add a transition.
Looping through the states: State 0

State 0:

$S' \rightarrow \bullet S$

$S \rightarrow \bullet Xb$

$X \rightarrow \bullet YZ$

$Y \rightarrow \bullet aa$

On “S” get:

state 1: $S' \rightarrow S\bullet$

On “X” get:

state 2: $S \rightarrow X\bullet b$

On “Y” get:

state 3: $X \rightarrow Y\bullet Z$

don’t forget closure!
Looping through the states: State 0

State 0:
S’ → •S
S → •Xb
X → •YZ
Y → •aa

On “S” get:
state 1: S’ → S•

On “X” get:
state 2: S → X•b

On “Y” get:
state 3: X → Y•Z

Z → •cS
Z → •ε
Looping through the states: State 0

State 0:
S' → •S
S → •Xb
X → •YZ
Y → •aa

On “a” get:
state 4: Y → a•a

On “b” or “c” get:
nothing (error)
Finishing the Decision Table

• Keep looping through all the other states, as you continue to produce them.

• What have you done? You’ve created the outline of a decision table with all of your states

• But the decision table is empty
Fill in actions/gotos

- Fill in shift actions (terminal transitions)
  - derive from the DFA. Look at the DFA. On state \( N \), on input \( x \), shift to state \( M \).
  - \( \text{action}(N, x) = sM \)

- Fill in goto actions (nonterminal transitions)
  - derive from the DFA. In state \( R \), on input \( Y \), shift to state \( S \).
  - \( \text{goto}(R, Y) = S \)
Fill in accept state

- In state $T$, given this situation:
  
  $S' \rightarrow S\cdot$

  - We get action $(T,\$) = \text{accept}$
## Decision Table So Far

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$</th>
<th>S</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What about reduce?

• An LR parser does a right-most derivation, which means the derivation is done in reverse.

• Given state $X \rightarrow abc$

• …the parser will recognize a reduce if the next input can "follow" $X$. 
Matching Patterns with Productions

• Back to the grammar, we add $ to S':

\[
\begin{align*}
S' & \rightarrow \$ \\
S & \rightarrow X \\
X & \rightarrow Y \\
Y & \rightarrow Z
\end{align*}
\]
Matching Patterns with Productions

- We start with $S' \rightarrow S$
- pattern: non terminal on end, add $\text{Follows}(S')$ to $\text{Follows}(S)$

$S'$  $\$
$S$  $\$
$X$
$Y$
$Z$
Matching Patterns with Productions

• We start with $S \rightarrow Xb$

• pattern: terminal following non terminal, add b to Follows($X$)

$S'$ $\$ 
$S$ $\$ 
$X$ $b$ 
$Y$ 
$Z$
Matching Patterns with Productions

- Looking at $X \rightarrow YZ$
- pattern: non-terminal (Z) at end, add $\text{Follows}(X)$ to $\text{Follows}(Z)$
- pattern: non-terminal (Y) followed by non-terminal (Z) going to empty, add $\text{Follows}(X)$ to $\text{Follows}(Y)$ (when Z is empty). Also, add $\text{First}(Z)$ to $\text{Follows}(Y)$, where $\text{First}(Z) = c$
Matching Patterns with Productions

\[ \begin{align*}
S' & \rightarrow \$ \\
S & \rightarrow \$ \\
X & \rightarrow b \\
Y & \rightarrow b, c \\
Z & \rightarrow b
\end{align*} \]
Okay, given this chart, we add actions. Look back to your states that you created from earlier. We only care about the ones which look like: $X \rightarrow \alpha^*$

- State 1 is $S' \rightarrow S^*$
- State 3 is $Z \rightarrow \epsilon$
- State 5 is $S \rightarrow Xb^*$
- State 6 is $Z \rightarrow YZ^*$
- State 8 is $Y \rightarrow aa^*$
- State 9 is $Z \rightarrow cS^*$
The number after “r” is the rule you reduce, not the state, like it is for shift and goto.

- State 1 is $S' \rightarrow S$
  - action (1, $\$) = r1 or accept

- State 3 is $Z \rightarrow \epsilon$
  - action (3, b) = r6
## Final Decision Table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$</th>
<th>S</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r1/acc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>r6</td>
<td>s7</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td>r2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LR Algorithm

• We've created a decision table and a DFA from a context-free grammar. Now we need to run the algorithm.

• The algorithm tells us, among other things, whether a given input is accepted or rejected.
Algorithm

- Push start on the top of the stack
- Repeat until accept or reject
- \( S = \text{top of stack (state)}, \ a = \text{current input token} \)
  - Consult action \( (S, a) \)
  - if action = Shift \( S' \)
    - push a
    - push \( S' \)
    - advance input
Algorithm

• if action = reduce $X \rightarrow \alpha$
  • pop $2 * \text{length}(\alpha)$
  • push $X$
  • consult goto $(S,x)$ to get $S'$
  • push $S'$
  • (input does not advance on reduce)
Example Grammar

S → D
D → Var L colon T
L → id
L → L comma id
T → real
decision table
  c = colon
  m = comma
  i = id
  r = real
Decision Table

- Contains shifts
  - i.e. \( s_3 = \text{read ("shift") new input onto stack, goto state 3} \)
- Contains reduces
  - i.e. \( r_2 = \text{"reduce by production #2"} \)
- Pop RHS from stack, push LHS
- Contains non-terminal directions
  - "Go where in the table when we reduce a rule?"
- Contains accept (start symbol) and reject (no entry in table). Stop for the former, give error for the latter.
Example Decision Table: run with input vimicr

<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>c</th>
<th>i</th>
<th>m</th>
<th>r</th>
<th>$</th>
<th>S</th>
<th>D</th>
<th>L</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td>$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r1/acc</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>s4</td>
<td></td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s8</td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stack</td>
<td>input</td>
<td>action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>vimicr</td>
<td>0, v= s2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2</td>
<td>imicr</td>
<td>2, i=s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2i4</td>
<td>micr</td>
<td>4, m=r3 (L-&gt;id)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L</td>
<td></td>
<td>goto 2, L=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3</td>
<td>micr</td>
<td>3, m=s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3m6</td>
<td>icr</td>
<td>6, i=s9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3m6i9</td>
<td>cr</td>
<td>9, c=r4 (L → L, i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L</td>
<td></td>
<td>goto 2, L=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3</td>
<td>cr</td>
<td>3, c=s5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3c5</td>
<td>r</td>
<td>5, r=s8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3c5r8</td>
<td>$</td>
<td>8, $=r5 (T → r)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3c5T</td>
<td></td>
<td>goto 5, T=7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0v2L3c5T7</td>
<td>$</td>
<td>7, $ = r2 (D → vLcT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td></td>
<td>goto0, D=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0D1</td>
<td>$</td>
<td>1, $ = accept/r1 (S →D)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>