LR Parser Construction
Hal Perkins
Autumn 2011
Agenda

- LR(0) state construction
- FIRST, FOLLOW, and nullable
- Variations: SLR, LR(1), LALR
LR State Machine

- Idea: Build a DFA that recognizes handles
  - Language generated by a CFG is generally not regular, but
  - Language of handles for a CFG is regular
    - So a DFA can be used to recognize handles
  - Parser reduces when DFA accepts
Prefixes, Handles, &c (review)

- If $S$ is the start symbol of a grammar $G$,  
  - If $S \Rightarrow^* \alpha$ then $\alpha$ is a *sentential form* of $G$  
  - $\gamma$ is a *viable prefix* of $G$ if there is some derivation  
    $S \Rightarrow^*_r \alpha A \omega \Rightarrow^*_r \alpha \beta \omega$  
    and $\gamma$ is a prefix of $\alpha \beta$.  
  - The occurrence of $\beta$ in $\alpha \beta \omega$ is a *handle* of $\alpha \beta \omega$.

- An *item* is a marked production (a . at some position in the right hand side)  
  - $[A ::= .XY]$ $[A ::= X.Y]$ $[A ::= XY]$
Building the LR(0) States

- Example grammar

\[
S' ::= S \$
\]
\[
S ::= ( \ L \ )
\]
\[
S ::= \times
\]
\[
L ::= S
\]
\[
L ::= L , S
\]

- We add a production $S'$ with the original start symbol followed by end of file ($$)

- Question: What language does this grammar generate?
Start of LR Parse

Initially

- Stack is empty
- Input is the right hand side of $S'$, i.e., $S$
- Initial configuration is $[S'::=S\].
- But, since position is just before $S$, we are also just before anything that can be derived from $S$.
Initial state

A state is just a set of items
- Start: an initial set of items
- Completion (or closure): additional productions whose left hand side appears to the right of the dot in some item already in the state

\[
S' ::= . S $\\nS ::= . ( L )\\nS ::= . x
\]
Shift Actions (1)

To shift past the $x$, add a new state with the appropriate item(s)
- In this case, a single item; the closure adds nothing
- This state will lead to a reduction since no further shift is possible
Shift Actions (2)

- If we shift past the (, we are at the beginning of \( L \)
- the closure adds all productions that start with \( L \), which requires adding all productions starting with \( S \)
Goto Actions

0. $S' ::= S$
1. $S ::= ( \hat{L} )$
2. $S ::= x$
3. $L ::= S$
4. $L ::= L, S$

$S' ::= S \cdot$

$S ::= \cdot x$

Once we reduce $S$, we’ll pop the rhs from the stack exposing the first state. Add a *goto* transition on $S$ for this.
Basic Operations

- **Closure** \((S)\)
  - Adds all items implied by items already in \(S\)

- **Goto** \((I, X)\)
  - \(I\) is a set of items
  - \(X\) is a grammar symbol (terminal or non-terminal)
  - **Goto** moves the dot past the symbol \(X\) in all appropriate items in set \(I\)
Closure Algorithm

Closure (S) =
repeat
for any item [A ::= \alpha . X \beta] in S
for all productions X ::= \gamma
add [X ::= . \gamma] to S
until S does not change
return S
Goto Algorithm

\[ Goto (I, X) = \]

set new to the empty set

for each item \([A ::= \alpha \cdot X \beta]\) in \(I\)

add \([A ::= \alpha X \cdot \beta]\) to new

return \(\text{Closure}(\text{new})\)

- This may create a new state, or may return an existing one.
LR(0) Construction

- First, augment the grammar with an extra start production $S' ::= S$.
- Let $T$ be the set of states.
- Let $E$ be the set of edges.
- Initialize $T$ to $\text{Closure}( [S' ::= . S$ $])$.
- Initialize $E$ to empty.
LR(0) Construction Algorithm

repeat
  for each state $I$ in $T$
    for each item $[A ::= \alpha \cdot X \beta]$ in $I$
      Let new be $\text{Goto} (I, X)$
      Add new to $T$ if not present
      Add $I \xrightarrow{X} \text{new}$ to $E$ if not present
  until $E$ and $T$ do not change in this iteration

Footnote: For symbol $\$, we don’t compute $\text{goto} (I, \)$; instead, we make this an accept action.
Building the Parse Tables (1)

- For each edge $I \xrightarrow{x} J$
  - if $X$ is a terminal, put $s_j$ in column $X$, row $I$ of the action table (shift to state $j$)
  - If $X$ is a non-terminal, put $g_j$ in column $X$, row $I$ of the goto table
Where Do We Stand?

- We have built the LR(0) state machine and parser tables
  - No lookahead yet
  - Different variations of LR parsers add lookahead information, but basic idea of states, closures, and edges remains the same
A Grammar that is not LR(0)

- Build the state machine and parse tables for a simple expression grammar

\[
S ::= E$
\]
\[
E ::= T + E
\]
\[
E ::= T
\]
\[
T ::= x
\]
LR(0) Parser for:

0. $S ::= E$
1. $E ::= T + E$
2. $E ::= T$
3. $T ::= x$

- State 3 has two possible actions on +:
  - Shift 4 or reduce 2
- Grammar is not LR(0)
SLR Parsers

Idea: Use information about what can follow a non-terminal to decide if we should perform a reduction

Easiest form is SLR – Simple LR

We need to be able to compute $\text{FOLLOW}(A)$ – the set of symbols that can follow $A$ in any possible derivation

- i.e., $t$ is in $\text{FOLLOW}(A)$ if any derivation contains $At$
- To compute this, we need to compute $\text{FIRST}(\gamma)$ for strings $\gamma$ that can follow $A$
Calculating FIRST(\(\gamma\))

- Sounds easy... If \(\gamma = XYZ\), then FIRST(\(\gamma\)) is FIRST(\(X\)), right?

- But what if we have the rule \(X ::= \epsilon\)?
- In that case, FIRST(\(\gamma\)) includes anything that can follow an \(X\) — i.e. FOLLOW(\(X\)).
FIRST, FOLLOW, and nullable

- \( \text{nullable}(X) \) is true if \( X \) can derive the empty string
- Given a string \( \gamma \) of terminals and non-terminals, \( \text{FIRST}(\gamma) \) is the set of terminals that can begin strings derived from \( \gamma \).
- \( \text{FOLLOW}(X) \) is the set of terminals that can immediately follow \( X \) in some derivation
- All three of these are computed together
Computing FIRST, FOLLOW, and nullable (1)

- Initialization
  - set FIRST and FOLLOW to be empty sets
  - set nullable(X) false for all non-terminals X
  - set FIRST[a] to a for all terminal symbols a
Computing FIRST, FOLLOW, and nullable (2)

repeat
  for each production \( X := Y_1 Y_2 \ldots Y_k \)
  if \( Y_1 \ldots Y_k \) are all nullable (or if \( k = 0 \))
    set nullable[\( X \)] = true
  for each \( i \) from 1 to \( k \) and each \( j \) from \( i+1 \) to \( k \)
    if \( Y_1 \ldots Y_{i-1} \) are all nullable (or if \( i = 1 \))
      add FIRST[\( Y_i \)] to FIRST[\( X \)]
    if \( Y_{i+1} \ldots Y_k \) are all nullable (or if \( i = k \))
      add FOLLOW[\( X \)] to FOLLOW[\( Y_i \)]
    if \( Y_{i+1} \ldots Y_{j-1} \) are all nullable (or if \( i+1 = j \))
      add FIRST[\( Y_j \)] to FOLLOW[\( Y_i \)]

Until FIRST, FOLLOW, and nullable do not change
Example

Grammar

\[ Z ::= d \]
\[ Z ::= X Y Z \]
\[ Y ::= \varepsilon \]
\[ Y ::= c \]
\[ X ::= Y \]
\[ X ::= a \]

<table>
<thead>
<tr>
<th>nullable</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x )</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>( z )</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

nullable
FIRST
FOLLOW
LR(0) Reduce Actions

- In an LR(0) parser, if a state contains a reduction, it is unconditional regardless of the next input symbol.

- Algorithm:
  
  Initialize $R$ to empty
  for each state $I$ in $T$
    for each item $[A ::= \alpha.]$ in $I$
      add $(I, A ::= \alpha)$ to $R$
SLR Construction

- This is identical to LR(0) – states, etc., except for the calculation of reduce actions

- Algorithm:
  
  Initialize $R$ to empty
  for each state $I$ in $T$
    for each item $[A ::= \alpha .]$ in $I$
      for each terminal $a$ in FOLLOW($A$)
        add $(I, a, A ::= \alpha)$ to $R$
  i.e., reduce $\alpha$ to $A$ in state $I$ only on lookahead $a$
SLR Parser for

0. S ::= E \$
1. E ::= T + E
2. E ::= T
3. T ::= x

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>+</th>
<th>$</th>
<th>E</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s5</td>
<td></td>
<td></td>
<td>g2</td>
<td>g3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>acc</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>r2</td>
<td>s4</td>
<td>r2</td>
<td></td>
<td>g2</td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td></td>
<td></td>
<td>g6</td>
<td>g3</td>
</tr>
<tr>
<td>5</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10/12/2011 © 2002-11 Hal Perkins & UW CSE