Parsing Algorithms 2: LR Parsing

CS 447 -- Stephen Watt
University of Western Ontario
Readings

- Purple Dragon Chapter 3. Lexical analysis
- Purple Dragon Chapter 4. Parsing
LR(k) Parsing

Left-to-right scan, Right-most derivation, with $k$ tokens of look-ahead.

+ Most general non-backtracking shift-reduce parsers
+ Larger class of grammars than LL parsing
+ Detect syntax errors as soon as possible with left-to-right scan
- Tables not suitable to build by hand
Three Methods

• SLR – simple LR.
  *Easiest to implement; least powerful.*

• Canonical LR.
  *Hard to implement; most powerful.*

• LALR – look ahead LR.
  *Relatively easy to implement; quite powerful.*
Overall Idea

- Input string of tokens
- Stack of parser states
- Action and Goto tables
- Parsing engine
The LR Parsing Engine

- Stack contains
  - \( X[i] \) grammar symbols
  - \( s[i] \) “states”

- Action table gives, for each \((s[i], a[j])\) pair, one of
  - *shift* \( s[j] \), for some state \( j \)
  - *reduce* \( A \rightarrow \beta \), for some production of the grammar
  - *accept* parsing is finished
  - *error* parser has discovered an error

- Goto table gives, for each state + grammar symbol, a new state.
Parser Configurations

• A pair
  – Stack contents
  – Rest of input
• For our figure
  \[(s_0 \ X_1 \ s_1 \ X_2 \ s_2 \ \ldots \ \ X_m \ s_m , \ \ a_k \ a_{k+1} \ \ldots \ \ a_n \ \$)\]
• This corresponds to a mid-derivation form
  \[X_1 \ X_2 \ \ldots \ \ X_m \ \ a_k \ a_{k+1} \ \ldots \ \ a_n \ \$\]
  interleaved with parser states.
Parser Action

Config = (s0 X1 s1 X2 s2 ... Xm sm, ak ak+1 ... an $)

• If Action(s[m], a[k]) == shift s.
  s = Goto(s[m], a[k])
  Config = (s0 X1 s1 X2 s2 ... Xm sm ak s, ak+1 ... an $)

• If Action(s[m], a[k]) == reduce A → β
  r = |β|
  s = Goto(s[m-r], A)
  Config = (s0 X1 s1 X2 s2 ... Xm-r sm-r A s, ak+1 ... an $)

• If Action(s[m], a[k]) == accept
  accept

• If Action(s[m], a[k]) == error
  halt, or attempt error recovery
1. E → E “+” T
2. E → T
3. T → T “*” F
4. T → F
5. F → “(” E “)"
6. F → id

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Goto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Id</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>s5</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>r4</td>
</tr>
<tr>
<td>4</td>
<td>s5</td>
<td>s4</td>
</tr>
<tr>
<td>5</td>
<td>r6</td>
<td>r6</td>
</tr>
<tr>
<td>6</td>
<td>s5</td>
<td>s4</td>
</tr>
<tr>
<td>7</td>
<td>s5</td>
<td>s4</td>
</tr>
<tr>
<td>8</td>
<td>s6</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>r1</td>
<td>s7</td>
</tr>
<tr>
<td>10</td>
<td>r3</td>
<td>r3</td>
</tr>
<tr>
<td>11</td>
<td>r5</td>
<td>r5</td>
</tr>
</tbody>
</table>
Constructing Parsing Tables

• Build a Finite Automaton to recognize viable prefixes of rules.
• LR(0) items:

  E → • E “+” T

  E → E • “+” T

  E → E “+” • T

  E → E “+” T •

• Indicates how much of production has been seen
• Can be represented as (production #, dot position)
Closure of an Item Set

Given set of items $I$ for grammar $G$, $\text{closure}(I)$ is the set formed by:

- All elements of $I$ are in $\text{closure}(I)$
- If $A \rightarrow \alpha \cdot B \beta$ is in $\text{closure}(I)$ and $B \rightarrow \gamma$ is a production in $G$, then $B \rightarrow \cdot \gamma$ is in $\text{closure}(I)$

$\text{closure}(I)$ captures the idea of finding all the rules that might be applicable at a given point.
Closure Example

• Augment previous grammar with $E' \rightarrow E$.
• Closure of $\{E' \rightarrow \cdot E\}$ is

$$\{E' \rightarrow \cdot E, \quad E \rightarrow \cdot E \quad + \quad T, \quad E \rightarrow \cdot T, \quad T \rightarrow \cdot T \quad * \quad F, \quad T \rightarrow \cdot F, \quad F \rightarrow \cdot \quad (\quad E \quad ), \quad F \rightarrow \cdot \quad \text{id}\}$$
The Goto Operation

- Goto(\(I, X\)) for an item set \(I\) and grammar symbol \(X\), is the set of items obtained by “moving the dot past \(X\)” in the items.

\[ J := \{ \} \]

for all \(A \rightarrow \alpha \cdot X \beta\) in \(I\), add \(A \rightarrow \alpha X \beta\) to \(J\).

return closure(\(J\)).
Constructing the Automaton

Initialize T to \{ \text{closure(} \{ S' \rightarrow \cdot \cdot S \}) \} \\
Initialize E to \{ \} \\
repeat \\
  for each state I in T \\
    for each item A \rightarrow \alpha \cdot X \beta \text{ in I} \\
      J := \text{Goto}(I,X) \\
      T := T \cup \{ J \} \\
      E := E \cup \{ I \rightarrow[X] J \} \\
until E and T do not change \\

Note that for X = $ we do not compute Goto(I,\$). \\
Instead we make an accept action.
Constructing the Tables

• For each edge $I \rightarrow [X] J$
  – If $X$ is a terminal,
    put the action **shift** $J$ at position $(I,X)$ of the table.
  – If $X$ is a nonterminal,
    put **goto** $J$ at position $(I,X)$
• For each state $I$ containing an item $S' \rightarrow S \cdot \$$,
  – put an accept action at $(I, \$$)
• For a state containing $A \rightarrow \gamma \cdot$
  (production $n$ with a dot at the end),
  put **reduce** $n$ at $(I, K)$ for every token $K$. 
LR(1) Items

• Some languages cannot be handled with LR(0). Some look-ahead is needed.
• An LR(1) item is of the form

\[ A \rightarrow \alpha \bullet \beta , \ a, \]\n
for \( A \) a non-terminal, \( a \) a terminal, \( \alpha \) and \( \beta \) strings of terminals and non-terminals.
• The terminal \( a \) is the “look-ahead”.

It has no effect when \( \beta \) is non-empty.

When \( \beta \) is empty, i.e. for \([ A \rightarrow \alpha \bullet , \ a]\), the item says to reduce the production \( A \rightarrow \alpha \bullet \).
Closure with LR(1) Items

• Compute the closure of a set \( I \) of LR(1) in items with grammar \( G' \) as follows:

\[
\text{Closure}(I) == \{ \\
\quad \text{repeat} \\
\quad \quad \text{for each item } [A \rightarrow \alpha \bullet B \beta, a] \text{ in } I \text{ repeat} \\
\quad \quad \quad \text{for each production } B \rightarrow \gamma \text{ in } G' \text{ repeat} \\
\quad \quad \quad \quad \text{for each terminal } b \text{ in FIRST}(\beta a) \text{ repeat} \\
\quad \quad \quad \quad \quad I := I \cup \{ [B \rightarrow \bullet \gamma, b] \} \\
\quad \quad \text{until } I \text{ stops growing} \\
\quad \text{return } I \\
\}\n\]
Goto with LR(1) Items

• Compute the goto of a set $I$ of LR(1) in items with grammar $G'$ as follows:

$$
\text{Goto}(I, X) == \{ \\
\quad J := \{ \} \\
\quad \text{for each item } [ A \rightarrow \alpha \bullet X\beta, a] \text{ in } I \text{ repeat} \\
\quad \quad J := J \cup \{ [ A \rightarrow \alpha X\bullet \beta, a] \} \\
\quad \text{return } \text{Closure}(J) \\
\}$$
Many potential LR(1) items will not be used. Compute the needed ones as follows:

\[
\text{Items}(G') == \{ \\
C := \{ \text{Closure}( \{ [S' \rightarrow \bullet S, \$] \} ) \} \\
\text{repeat} \\
\quad \text{for each item set } I \text{ in } C \text{ repeat} \\
\quad \quad \text{for each grammar symbol } X \text{ repeat} \\
\quad \quad \quad J := \text{Goto}[I,X] \\
\quad \quad \quad \text{if } J \text{ nonempty and } J \text{ not in } C \text{ then } C := C \cup \{ J \} \\
\quad \text{until } C \text{ stops growing}
\]
Constructing an LR(1) Parser

• To build the automaton, use the new definitions of Closure and Goto in the previous algorithm.

• To build the tables, change

  For a state containing $A \rightarrow \gamma \cdot$
  (production $n$ with a dot at the end),
  put $reduce \ n$ at $(I, K)$ for every token $K$.

  to

  For a state containing $A \rightarrow [\gamma \cdot, a]$
  (production $n$ with a dot at the end),
  put $reduce \ n$ at $(I, a)$.