CMSC 22610	**Implementation of**	**Handout 4**
Winter 2009	**Computer Languages - I**	**February 3, 2009**

Shift-Reduce Parsing Notes

1 $LR(k)$ Parsing Overview

- L : left-to-right parse
- R : rightmost derivation
- k : k-symbols lookahead

1.1 LR Parsing Example

Suppose we wish to parse $a \wedge b \vee c$ in the ***Infix with parens with*** $\{\vee_L\} < \{\wedge_L\} < \{\neg\}$ grammar using the LR parsing algorithm.

First, consider a rightmost derivation of the string in the grammar:

$$
\begin{array}{ll}
\hat{S} & \hat{S} \rightarrow P\ \$ \\
\Rightarrow\ P\ \$ & P \rightarrow O \\
\Rightarrow\ O\ \$ & O \rightarrow O \vee A \\
\Rightarrow\ O \vee A\ \$ & A \rightarrow Z \\
\Rightarrow\ O \vee Z\ \$ & Z \rightarrow var \\
\Rightarrow\ O \vee c\ \$ & O \rightarrow A \\
\Rightarrow\ A \vee c\ \$ & A \rightarrow A \wedge Z \\
\Rightarrow\ A \wedge Z \vee c\ \$ & Z \rightarrow var \\
\Rightarrow\ A \wedge b \vee c\ \$ & A \rightarrow Z \\
\Rightarrow\ Z \wedge b \vee c\ \$ & Z \rightarrow var \\
\Rightarrow\ a \wedge b \vee c\ \$ &
\end{array}
$$

Now, consider the execution of the LR parsing algorithm:

Stack	Input	Action
	$\dot{a} \wedge b \vee c\ \$$	shift *var*
var	$\dot{\wedge}\ b \vee c\ \$$	reduce $Z \rightarrow var$
Z	$\dot{\wedge}\ b \vee c\ \$$	reduce $A \rightarrow Z$
A	$\dot{\wedge}\ b \vee c\ \$$	shift $\wedge$
$A \wedge$	$\dot{b} \vee c\ \$$	shift *var*
$A \wedge var$	$\dot{\vee}\ c\ \$$	reduce $Z \rightarrow var$
$A \wedge Z$	$\dot{\vee}\ c\ \$$	reduce $A \rightarrow A \wedge Z$
A	$\dot{\vee}\ c\ \$$	reduce $O \rightarrow A$
O	$\dot{\vee}\ c\ \$$	shift $\vee$
$O \vee$	$\dot{c}\ \$$	shift *var*
$O \vee var$	$\dot{\$}$	reduce $Z \rightarrow var$
$O \vee Z$	$\dot{\$}$	reduce $A \rightarrow Z$
$O \vee A$	$\dot{\$}$	reduce $O \rightarrow O \vee A$
O	$\dot{\$}$	reduce $P \rightarrow O$
P	$\dot{\$}$	accept

The algorithm works by performing a sequence of reductions that correspond to the rightmost derivation in reverse. Furthermore, note that if we concatenate the stack with the input in one line of the execution, then we obtain a line in the rightmost derivation. In essence, the stack records what was parsed in the past.

1.2 LR Parsing Concepts

Although there are several variations on LR parsing (in addition to varying k), they all share some basic concepts and the same parsing algorithm.

1.2.1 States and Items

An LR parser maintains a stack of *states*. Each state is a set of *items*. In its fullest generality, an LR item is a production, a position within the production, and k-symbols of lookahead:

$$\langle A \rightarrow X_1 \cdots X_n, i, \underline{a}_1 \cdots \underline{a}_k \rangle$$

Every state may be mapped to a sequence of terminals and nonterminals; hence, in the example above, we used terminals and nonterminals on the stack.

1.2.2 Action Table

The *action table* is indexed by states and (sequences of) terminal symbols; every entry in the table is one of four possible kinds of actions:

- **shift** t
- **reduce** $A \rightarrow X_1 \cdots X_n$
- **accept**
- **error**

1.2.3 Goto Table

The *goto table* is indexed by states and nonterminal symbols; every entry in the table is a state.

1.2.4 $LR(k \leq 1)$ Parsing Algorithm

$stack \leftarrow []$ (* *empty stack* *)
$push(s_0, stack)$
while true do
  $s \leftarrow peek(stack)$
  if $Action[s, curTok()] = \{\text{shift } t\}$ **then**
    $push(t, stack)$ (* t represents the current token *)
    $advanceTok()$
  else if $Action[s, curTok()] = \{\text{reduce } A \rightarrow X_1 \cdots X_n\}$ **then**
    $\underbrace{pop(stack) \text{ ; } \cdots \text{ ; } pop(stack)}_{n}$
    $t \leftarrow Goto[peek(stack), A]$
    $push(t, stack)$ (* t represents the A *)
  else if $Action[s, curTok()] = \{\text{accept}\}$ **then**
    break
  else $error()$
end
$accept()$

2 $LR(0)$ Parsing

$LR(0)$ is the weakest of the LR parsing methods (using 0 symbols of lookahead), but serves as the basis for the other methods.

2.1 Definitions

Since an $LR(0)$ parser uses no symbols of lookahead, its items are of the form:

$$\langle A \ \rightarrow \ X_1 \ \cdots \ X_n, i \rangle$$

which we will often write using the notation:

$$[A \ \rightarrow \ X_1 \ \cdots \ X_i \ . \ X_{i+1} \ \cdots \ X_n]$$

For a set of items I, we define $\mathit{Closure}$ and Closed as follows:

$$\mathit{Closure}(I) \ = \ I \cup \{[B \ \rightarrow \ . \ \beta] \mid [A \ \rightarrow \ \alpha \ . \ B \ \gamma] \in \mathit{Closure}(I) \text{ and } B \ \rightarrow \ \beta \in \mathcal{P}\}$$

$$\mathit{Closed}(I) \ = \ \begin{cases} \textbf{true} & I = \mathit{Closure}(I) \\ \textbf{false} & \text{otherwise} \end{cases}$$

States in an $LR(0)$ parser are closed sets of items.

Given a set of items I, the *kernel* of I is the smallest set $J \subseteq I$ such that $\mathit{Closure}(J) = \mathit{Closure}(I)$. We may efficiently represent closed sets (hence, states) by their kernels. All kernel items (i.e., items in the kernel of a set of items) must be of the form $[\hat{S} \ \rightarrow \ . \ S \ \$]$ or $[A \ \rightarrow \ \alpha \ . \ \gamma]$ where $\alpha \neq \epsilon$.

For a state I, we define Goto as follows:

$$\mathit{Goto}(I, X) \ = \ \mathit{Closure}(\{[A \ \rightarrow \ \alpha \ X \ . \ \beta] \mid [A \ \rightarrow \ \alpha \ . \ X \ \beta] \in I\})$$

2.2 Computing Closure

The closure of a set of items I can be computed by the following algorithm:

$\mathit{Closure}_I \leftarrow \{I\}$
do
  foreach $[A \ \rightarrow \ \alpha \ . \ X \ \gamma] \in \mathit{Closure}_I$ **do**
    foreach $B \ \rightarrow \ \beta \in \mathcal{P}$ **do**
      if $B = X$ **then**
        $\mathit{Closure}_I \leftarrow \mathit{Closure}_I \cup \{[B \ \rightarrow \ . \ \beta]\}$
      end
  end
until $\mathit{Closure}_I$ *does not change*

2.3 Computing Canonical States

The canonical $LR(0)$ states (item sets) are computed by the following algorithm:

$I_0 \leftarrow \mathit{Closure}(\{[\hat{S} \rightarrow . S \$]\})$
$C \leftarrow \{I_0\}$
do
  foreach $I \in C$ **do**
    foreach $X \in ((\mathcal{T} \setminus \{\$\}) \cup \mathcal{N})$ **do**
      $J \leftarrow \mathit{Goto}(I, X)$
      if $J \neq \emptyset$ **and** $J \notin C$
        then $C \leftarrow C \cup \{J\}$
    end
  end
until C *does not change*

2.4 Computing Action Tables

To construct the $LR(0)$ action table:

- initialize $\mathit{Action}[I, \underline{a}] = \{\}$ for $I \in C$ and $\underline{a} \in \mathcal{T}$.
- add actions according to the following rules (for $I \in C$):
  - if $\mathit{Goto}(I, \underline{a}) = J$ and $J \neq \emptyset$,
    then add shift J to $\mathit{Action}[I, \underline{a}]$.
  - if $[\hat{S} \rightarrow S . \$] \in I$,
    then add accept to $\mathit{Action}[I, \$]$.
  - if $[A \rightarrow X_1 \cdots X_n .] \in I$ and $A \neq \hat{S}$,
    then add reduce $A \rightarrow X_1 \cdots X_n$ to $\mathit{Action}[I, \underline{a}]$ for each $\underline{a} \in \mathcal{T}$.

If any $\mathit{Action}[I, \underline{a}]$ entry has more than one action, then the grammar is not $LR(0)$.

The zero tokens of lookahead is captured by the fact that the reductions do not depend upon the current input token.

2.5 Computing Goto Tables

To construct the $LR(0)$ goto table:

- if $\mathit{Goto}(I, A) = J$ and $J \neq \emptyset$,
  then set $\mathit{Goto}[I, A]$ equal to goto J.

2.6 Example

Consider computing the canonical $LR(0)$ states for ***Postfix***.

$$
\begin{aligned}
I_0 &= \{[\hat{S} \rightarrow . \, P \; \$], \\
&\quad [P \rightarrow . \, var], [P \rightarrow . \, P \neg], [P \rightarrow . \, P \, P \wedge], [P \rightarrow . \, P \, P \vee]\} \\
Goto(I_0, var) &= \\
Goto(I_0, \neg) &= \\
Goto(I_0, \wedge) &= \\
Goto(I_0, \vee) &= \\
Goto(I_0, \hat{S}) &= \\
Goto(I_0, P) &= \\
\\
I_1 &= \{[P \rightarrow var \, .]\} \\
Goto(I_1, \_) &= \emptyset \\
\\
I_2 &= \{[\hat{S} \rightarrow P \, . \, \$], [P \rightarrow P \, . \, \neg], [P \rightarrow P \, . \, P \wedge], [P \rightarrow P \, . \, P \vee], \\
&\quad [P \rightarrow . \, var], [P \rightarrow . \, P \neg], [P \rightarrow . \, P \, P \wedge], [P \rightarrow . \, P \, P \vee]\} \\
Goto(I_2, var) &= \\
Goto(I_2, \neg) &= \\
Goto(I_2, \wedge) &= \\
Goto(I_2, \vee) &= \\
Goto(I_2, \hat{S}) &= \\
Goto(I_2, P) &= \\
\\
I_3 &= \{[P \rightarrow P \neg .]\} \\
Goto(I_3, \_) &= \emptyset \\
\\
I_4 &= \{[P \rightarrow P \, P \, . \wedge], [P \rightarrow P \, P \, . \vee], \\
&\quad [P \rightarrow P \, . \neg], [P \rightarrow P \, . \, P \wedge], [P \rightarrow P \, . \, P \vee], \\
&\quad [P \rightarrow . \, var], [P \rightarrow . \, P \neg], [P \rightarrow . \, P \, P \wedge], [P \rightarrow . \, P \, P \vee]\} \\
Goto(I_4, var) &= \\
Goto(I_4, \neg) &= \\
Goto(I_4, \wedge) &= \\
Goto(I_4, \vee) &= \\
Goto(I_4, \hat{S}) &= \\
Goto(I_4, P) &= \\
\\
I_5 &= \{[P \rightarrow P \, P \wedge .]\} \\
Goto(I_5, \_) &= \emptyset \\
\\
I_6 &= \{[P \rightarrow P \, P \vee .]\} \\
Goto(I_6, \_) &= \emptyset
\end{aligned}
$$

Now consider computing the $LR(0)$ action and goto tables for ***Postfix***.

	Action					Goto	
	var	$\neg$	$\wedge$	$\vee$	$\$$	$\hat{S}$	*P*
I_0	s I_1						g I_2
I_1	r $P \rightarrow var$	r $P \rightarrow var$	r $P \rightarrow var$	r $P \rightarrow var$	r $P \rightarrow var$		
I_2	s I_1	s I_3			a		g I_4
I_3	r $P \rightarrow P \neg$	r $P \rightarrow P \neg$	r $P \rightarrow P \neg$	r $P \rightarrow P \neg$	r $P \rightarrow P \neg$		
I_4	s I_1	s I_3	s I_5	s I_6			g I_4
I_5	r $P \rightarrow P\ P \wedge$	r $P \rightarrow P\ P \wedge$	r $P \rightarrow P\ P \wedge$	r $P \rightarrow P\ P \wedge$	r $P \rightarrow P\ P \wedge$		
I_6	r $P \rightarrow P\ P \vee$	r $P \rightarrow P\ P \vee$	r $P \rightarrow P\ P \vee$	r $P \rightarrow P\ P \vee$	r $P \rightarrow P\ P \vee$		

Now consider parsing $a\ b\ \wedge\ c\ \vee$:

Stack	**Input**	**Action**
I_0	$\dot{a}\ b \wedge c \vee \$$	shift I_1
$I_0\ I_1$	$\dot{b} \wedge c \vee \$$	reduce $P \rightarrow var$
$I_0\ I_2$	$\dot{b} \wedge c \vee \$$	shift I_1
$I_0\ I_2\ I_1$	$\dot{\wedge}\ c \vee \$$	reduce $P \rightarrow var$
$I_0\ I_2\ I_4$	$\dot{\wedge}\ c \vee \$$	shift I_5
$I_0\ I_2\ I_4\ I_5$	$\dot{c} \vee \$$	reduce $P \rightarrow P\ P \wedge$
$I_0\ I_2$	$\dot{c} \vee \$$	shift I_1
$I_0\ I_2\ I_1$	$\dot{\vee}\ \$$	reduce $P \rightarrow var$
$I_0\ I_2\ I_4$	$\dot{\vee}\ \$$	shift I_6
$I_0\ I_2\ I_4\ I_6$	$\dot{\$}$	reduce $P \rightarrow P\ P \vee$
$I_0\ I_2$	$\dot{\$}$	accept

3 Simple LR (SLR) Parsing

The major weakness of $LR(0)$ action tables is that any reduction reduces on every input token. This property can lead to conflicts, but we can avoid some of these conflicts by only reducing when the next input token is in the $Follow$ set.

SLR parsing uses the same items, $Closure$ and $Goto$ functions, canonical states, and goto table as $LR(0)$ parsing. The only difference is in the computation of the action table and, furthermore, only in the rule to add a reduce item to the table.

3.1 Computing Action Tables

To construct the SLR action table:

- initialize $Action[I, \underline{a}] = \{\}$ for $I \in C$ and $\underline{a} \in \mathcal{T}$.
- add actions according to the following rules (for $I \in C$):
  - if $Goto(I, \underline{a}) = J$ and $J \neq \emptyset$,
    then add shift J to $Action[I, \underline{a}]$.
  - if $[\hat{S} \rightarrow S \,.\, \texttt{\$}] \in I$,
    then add accept to $Action[I, \texttt{\$}]$.
  - if $[A \rightarrow X_1 \cdots X_n \,.] \in I$ and $A \neq \hat{S}$,
    then add reduce $A \rightarrow X_1 \cdots X_n$ to $Action[I, \underline{a}]$ for each $\underline{a} \in Follow(A)$.

If any $Action[I, \underline{a}]$ entry has more than one action, then the grammar is not SLR.

3.2 Example

Consider computing the canonical $LR(0)/SLR$ states for ***Infix with parens with*** $\{\vee_L\} < \{\wedge_L\} < \{\neg\}$.

$$
\begin{array}{rcl}
I_0 & = & \{[\underline{\hat{S} \rightarrow . P \ \$}], \\
 & & \underline{[P \rightarrow . O]}, \\
 & & [O \rightarrow . O \vee A], [O \rightarrow . A], \\
 & & [A \rightarrow . A \wedge Z], [A \rightarrow . Z], \\
 & & [Z \rightarrow . var], [Z \rightarrow . \neg Z], [Z \rightarrow . \ (\ P\ )\ ]\} \\
Goto(I_0, var) & = & I_1 \\
Goto(I_0, \neg) & = & I_2 \\
Goto(I_0, \wedge) & = & \emptyset \\
Goto(I_0, \vee) & = & \emptyset \\
Goto(I_0, \ (\ ) & = & I_3 \\
Goto(I_0, \ )\ ) & = & \emptyset \\
Goto(I_0, \$) & = & \emptyset \\
Goto(I_0, \hat{S}) & = & \emptyset \\
Goto(I_0, P) & = & I_4 \\
Goto(I_0, O) & = & I_5 \\
Goto(I_0, A) & = & I_6 \\
Goto(I_0, Z) & = & I_7
\end{array}
$$

$$
\begin{array}{rcl}
I_1 & = & \{\underline{[Z \rightarrow var .]}\} \\
Goto(I_1, \_) & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
I_2 & = & \{\underline{[Z \rightarrow \neg . Z]}, [Z \rightarrow . var], [Z \rightarrow . \neg Z], [Z \rightarrow . \ (\ P\ )\ ]\} \\
Goto(I_2, var) & = & I_1 \\
Goto(I_2, \neg) & = & I_2 \\
Goto(I_2, \wedge) & = & \emptyset \\
Goto(I_2, \vee) & = & \emptyset \\
Goto(I_2, \ (\ ) & = & I_3 \\
Goto(I_2, \ )\ ) & = & \emptyset \\
Goto(I_2, \$) & = & \emptyset \\
Goto(I_2, \hat{S}) & = & \emptyset \\
Goto(I_2, P) & = & \emptyset \\
Goto(I_2, O) & = & \emptyset \\
Goto(I_2, A) & = & \emptyset \\
Goto(I_2, Z) & = & I_8
\end{array}
$$

$$
\begin{array}{rcl}
I_3 & = & \{\underline{[Z \rightarrow \texttt{(} \,.\, P \,\texttt{)}\,]}, \\
 & & [P \rightarrow \,.\, O], \\
 & & [O \rightarrow \,.\, O \vee A], [O \rightarrow \,.\, A], \\
 & & [A \rightarrow \,.\, A \wedge Z], [A \rightarrow \,.\, Z], \\
 & & [Z \rightarrow \,.\, var], [Z \rightarrow \,.\, \neg Z], [Z \rightarrow \,.\, \texttt{(}\, P \,\texttt{)}\,]\} \\
Goto(I_3, var) & = & I_1 \\
Goto(I_3, \neg) & = & I_2 \\
Goto(I_3, \wedge) & = & \emptyset \\
Goto(I_3, \vee) & = & \emptyset \\
Goto(I_3, \texttt{(}) & = & I_3 \\
Goto(I_3, \texttt{)}) & = & \emptyset \\
Goto(I_3, \texttt{\$}) & = & \emptyset \\
Goto(I_3, \hat{S}) & = & \emptyset \\
Goto(I_3, P) & = & I_9 \\
Goto(I_3, O) & = & I_5 \\
Goto(I_3, A) & = & I_6 \\
Goto(I_3, Z) & = & I_7 \\
 & & \\
I_4 & = & \{\underline{[\hat{S} \rightarrow P \,.\, \texttt{\$}]}\} \\
Goto(I_4, \_) & = & \emptyset \\
 & & \\
I_5 & = & \{\underline{[P \rightarrow O \,.]}, \underline{[O \rightarrow O \,.\, \vee A]}\} \\
Goto(I_5, \vee) & = & I_{10} \\
Goto(I_5, \_) & = & \emptyset \\
 & & \\
I_6 & = & \{\underline{[O \rightarrow A \,.]}, \underline{[A \rightarrow A \,.\, \wedge Z]}\} \\
Goto(I_6, \wedge) & = & I_{11} \\
Goto(I_6, \_) & = & \emptyset \\
 & & \\
I_7 & = & \{\underline{[A \rightarrow Z \,.]}\} \\
Goto(I_7, \_) & = & \emptyset \\
 & & \\
I_8 & = & \{\underline{[Z \rightarrow \neg Z \,.]}\} \\
Goto(I_8, \_) & = & \emptyset \\
 & & \\
I_9 & = & \{\underline{[Z \rightarrow \texttt{(}\, P \,.\, \texttt{)}\,]}\} \\
Goto(I_9, \texttt{)}) & = & I_{12} \\
Goto(I_9, \_) & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
I_{10} & = & \{\underline{[O \rightarrow O \vee . A]}, \\
 & & \underline{[A \rightarrow . A \wedge Z]}, [A \rightarrow . Z], \\
 & & [Z \rightarrow . var], [Z \rightarrow . \neg Z], [Z \rightarrow . \texttt{(} P \texttt{)}]\} \\
Goto(I_{10}, var) & = & I_1 \\
Goto(I_{10}, \neg) & = & I_2 \\
Goto(I_{10}, \wedge) & = & \emptyset \\
Goto(I_{10}, \vee) & = & \emptyset \\
Goto(I_{10}, \texttt{(}) & = & I_3 \\
Goto(I_{10}, \texttt{)}) & = & \emptyset \\
Goto(I_{10}, \$) & = & \emptyset \\
Goto(I_{10}, \hat{S}) & = & \emptyset \\
Goto(I_{10}, P) & = & \emptyset \\
Goto(I_{10}, O) & = & \emptyset \\
Goto(I_{10}, A) & = & I_{13} \\
Goto(I_{10}, Z) & = & I_7
\end{array}
$$

$$
\begin{array}{rcl}
I_{11} & = & \{\underline{[A \rightarrow A \wedge . Z]}, \\
 & & [Z \rightarrow . var], [Z \rightarrow . \neg Z], [Z \rightarrow . \texttt{(} P \texttt{)}]\} \\
Goto(I_{11}, var) & = & I_1 \\
Goto(I_{11}, \neg) & = & I_2 \\
Goto(I_{11}, \wedge) & = & \emptyset \\
Goto(I_{11}, \vee) & = & \emptyset \\
Goto(I_{11}, \texttt{(}) & = & I_3 \\
Goto(I_{11}, \texttt{)}) & = & \emptyset \\
Goto(I_{11}, \$) & = & \emptyset \\
Goto(I_{11}, \hat{S}) & = & \emptyset \\
Goto(I_{11}, P) & = & \emptyset \\
Goto(I_{11}, O) & = & \emptyset \\
Goto(I_{11}, A) & = & \emptyset \\
Goto(I_{11}, Z) & = & I_{14}
\end{array}
$$

$$
\begin{array}{rcl}
I_{12} & = & \{\underline{[Z \rightarrow \texttt{(} P \texttt{)} .]}\} \\
Goto(I_{12}, \_) & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
I_{13} & = & \{\underline{[O \rightarrow O \vee A .]}, \underline{[A \rightarrow A . \wedge Z]}\} \\
Goto(I_{13}, \wedge) & = & I_{11} \\
Goto(I_{13}, \_) & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
I_{14} & = & \{\underline{[A \rightarrow A \wedge Z .]}\} \\
Goto(I_{14}, \_) & = & \emptyset
\end{array}
$$

Now consider computing the $LR(0)$ action and goto tables for ***Infix with parens with*** $\{\vee_L\} < \{\wedge_L\} < \{\neg\}$.

	Action						
	var	$\neg$	$\wedge$	$\vee$	**(**	**)**	**$**
I_0	s I_1	s I_2			s I_3		
I_1	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*	r $Z \rightarrow$ *var*
I_2	s I_1	s I_2			s I_3		
I_3	s I_1	s I_2			s I_3		
I_4							a
I_5	r $P \rightarrow O$	r $P \rightarrow O$	r $P \rightarrow O$	s I_{10} r $P \rightarrow O$	r $P \rightarrow O$	r $P \rightarrow O$	r $P \rightarrow O$
I_6	r $O \rightarrow A$	r $O \rightarrow A$	s I_{11} r $O \rightarrow A$	r $O \rightarrow A$	r $O \rightarrow A$	r $O \rightarrow A$	r $O \rightarrow A$
I_7	r $A \rightarrow Z$	r $A \rightarrow Z$	r $A \rightarrow Z$	r $A \rightarrow Z$	r $A \rightarrow Z$	r $A \rightarrow Z$	r $A \rightarrow Z$
I_8	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$	r $A \rightarrow \neg Z$
I_9						s I_{12}	
I_{10}	s I_1	s I_2			s I_3		
I_{11}	s I_1	s I_2			s I_3		
I_{12}	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**	r $Z \rightarrow$ **(** P **)**
I_{13}	r $O \rightarrow O \vee A$	r $O \rightarrow O \vee A$	s I_{11} r $O \rightarrow O \vee A$	r $O \rightarrow O \vee A$	r $O \rightarrow O \vee A$	r $O \rightarrow O \vee A$	r $O \rightarrow O \vee A$
I_{14}	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$	r $A \rightarrow A \wedge Z$

	Goto				
	$\hat{S}$	P	O	A	Z
I_0		g I_4	g I_5	g I_6	g I_7
I_1					
I_2					g I_8
I_3		g I_9	g I_5	g I_6	g I_7
I_4					
I_5					
I_6					
I_7					
I_8					
I_9					
I_{10}			g I_{13}	g I_7	
I_{11}					g I_{14}
I_{12}					
I_{13}					
I_{14}					

4 $LR(1)$ Parsing

$LR(1)$ is the strongest of the practical LR parsing methods. (LR parsing with $k > 0$ quickly leads to impractically large parsing tables.)

4.1 Definitions

Since an $LR(1)$ parser uses one symbol of lookahead, its items are of the form:

$$\langle A \ \rightarrow \ X_1 \ \cdots \ X_n, i, \underline{a} \rangle$$

which we will often write using the notation:

$$\langle [A \ \rightarrow \ X_1 \ \cdots \ X_i \ . \ X_{i+1} \ \cdots \ X_n], \underline{a} \rangle$$

An $LR(1)$ item $\langle [A \ \rightarrow \ X_1 \ \cdots \ X_i \ . \ X_{i+1} \ \cdots \ X_n], \underline{a} \rangle$ represents a parser configuration where $X_1 \ \cdots \ X_i$ is at the top of the stack and the input has a prefix that is derivable from $X_{i+1} \ \cdots \ X_n \ \underline{a}$ (that is, $X_{i+1} \ \cdots \ X_n \ \underline{a} \Rightarrow^* u$ and the input is $u \ v$).

For a set of items I, we define $Closure$ and $Closed$ as follows:

$$\begin{aligned} Closure(I) \ &= \ I \cup \{ \langle [B \ \rightarrow \ . \ \beta], \underline{b} \rangle \mid \langle [A \ \rightarrow \ \alpha \ . \ B \ \gamma], \underline{a} \rangle \in Closure(I) \text{ and } B \ \rightarrow \ \beta \in \mathcal{P} \\ &\qquad \text{and } \underline{b} \in First(\gamma \ \underline{a}) \} \end{aligned}$$

$$Closed(I) \ = \ \begin{cases} \mathbf{true} & I = Closure(I) \\ \mathbf{false} & \text{otherwise} \end{cases}$$

States in an $LR(1)$ parser are closed sets of items.

For a state I, we define $Goto$ as follows:

$$Goto(I, X) \ = \ Closure(\{ \langle [A \ \rightarrow \ \alpha \ X \ . \ \beta], \underline{a} \rangle \mid \langle [A \ \rightarrow \ \alpha \ . \ X \ \beta], \underline{a} \rangle \in I \})$$

We can compute $Closure$ and the canonical $LR(1)$ states using the same techniques as above.

4.2 Computing Closure

The closure of a set of items I can be computed by the following algorithm:

```
Closure_I ← {I}
do
    foreach ⟨[A → α . X γ], a⟩ ∈ Closure_I do
        foreach B → β ∈ P do
            if B = X then
                foreach b ∈ First(γ a) do
                    Closure_I ← Closure_I ∪ {⟨[B → . β], b⟩}
                end
        end
    end
until Closure_I does not change
```

4.3 Computing Canonical States

The canonical $LR(1)$ states (item sets) are computed by the following algorithm:

$I_0 \leftarrow \mathit{Closure}(\{\langle[\hat{S} \rightarrow \ . \ S \ \$], \$\rangle\})$
$C \leftarrow \{I_0\}$
do
  foreach $I \in C$ **do**
    foreach $X \in ((\mathcal{T} \setminus \{\$\}) \cup \mathcal{N})$ **do**
      $J \leftarrow \mathit{Goto}(I, X)$
      if $J \neq \emptyset$ **and** $J \notin C$
        then $C \leftarrow C \cup \{J\}$
    end
  end
until C *does not change*

4.4 Computing Action Tables

To construct the $LR(1)$ action table:

- initialize $\mathit{Action}[I, \underline{a}] = \{\}$ for $I \in C$ and $\underline{a} \in \mathcal{T}$.
- add actions according to the following rules (for $I \in C$):
  - if $\langle[A \rightarrow \ \alpha \ . \ \underline{a} \ \gamma], \_\rangle \in I$ and $\mathit{Goto}(I, \underline{a}) = J$,
    then add shift J to $\mathit{Action}[I, \underline{a}]$.
  - if $\langle[\hat{S} \rightarrow \ S \ . \ \$], \$\rangle \in I$,
    then add accept to $\mathit{Action}[I, \$]$.
  - if $\langle[A \rightarrow \ X_1 \ \cdots \ X_n \ .], \underline{a}\rangle \in I$ and $A \neq \hat{S}$,
    then add reduce $A \rightarrow \ X_1 \ \cdots \ X_n$ to $\mathit{Action}[I, \underline{a}]$.

If any $\mathit{Action}[I, \underline{a}]$ entry has more than one action, then the grammar is not $LR(1)$.

4.5 Computing Goto Tables

To construct the $LR(1)$ goto table:

- if $\mathit{Goto}(I, A) = J$ and $J \neq \emptyset$,
  then set $\mathit{Goto}[I, A]$ equal to goto J.

5 $LALR(1)$ Parsing

The number of canonical $LR(1)$ states for a grammar can be very large. We can reduce the number of states by merging any states whose items are identical when ignoring the lookahead token. We compute action and goto tables as before. The resulting parse table is an $LALR(1)$ (*Look-Ahead* $LR(1)$) parse table.

Because we are merging states, the $LALR(1)$ parse table may have conflicts where the $LR(1)$ parse table did not. Hence, $LALR(1)$ is weaker than $LR(1)$ (but stronger than SLR). However, such conflicts are rare in practical programming langauges, and the benefit of the significantly smaller parse tables has made $LALR(1)$ the dominant parsing method.