PCS ANIL KUMAR CHUKKALA

11CS10032

Group – 15 Date : 26-08-2013

Table Driven Predictive Parser

A non-recursive predictive parser can be constructed that maintains a stack (explicitly) and a table to select the appropriate production rule.

Parsing Table

The rows of the predictive parser table are indexed by the non-terminals and the columns are indexed by the terminals including the end-of-input marker ($). The content of the table are production rules or error situations. The table cannot have multiple entries.

Parsing Stack

The parsing stack can hold both terminals and non-terminals. At the beginning, the stack contains the end-of-stack marker ($) and the start symbol on top of it.

Parsing Table Construction

- If $A \rightarrow \alpha$ is a production rule and $a \in \text{FIRST}(\alpha)$, then $P[A][a] = A \rightarrow \alpha$.
- If $A \rightarrow \varepsilon$ is a production rule and $a \in \text{FOLLOW}(A)$, then $P[A][a] = A \rightarrow \varepsilon$.

Actions

- If the top-of-stack is a terminal symbol (token) and matches with input token, both are consumed. A mismatch is an error.
- If the top-of-stack is a non-terminal A, the input token is a, $P[A][a]$ has the entry $A \rightarrow \alpha$, then A is to be replaced by α, with the head of α on the top of the stack.

Example

Consider the production rules of the non-terminal SL.

$$\texttt{SL} \rightarrow \texttt{S SL} \mid \varepsilon$$

The FIRST($\texttt{SL} \rightarrow \texttt{S SL}$) = {id if while scan print} and FOLLOW(SL) ={end else}. So, $P[\texttt{SL}][\texttt{IDNTIFIER}] = P[\texttt{SL}][\texttt{IF}] = P[\texttt{SL}][\texttt{WHILE}] = P[\texttt{SL}][\texttt{SCAN}] = P[\texttt{SL}][\texttt{PRINT}] = \texttt{SL} \rightarrow \texttt{S SL}$ and $P[\texttt{SL}][\texttt{END}] = P[\texttt{SL}][\texttt{ELSE}] = \texttt{SL} \rightarrow \varepsilon$.

Note

Multiple entries in a table indicates that the grammar is not $LL(1)$. But it is interesting to note that in some cases we can drop (with proper consideration) some of these entries and construct a parser.

Example

Consider the ambiguous grammar G_1 for expressions.

$$E \rightarrow E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid ic$$

After the removal of left-recursion we get the following ambiguous, no-left-recursive grammar:

Example

$$E \rightarrow (E)E' \mid icE'$$

$$E' \rightarrow +EE' \mid -EE' \mid *EE' \mid /EE' \mid \varepsilon$$

We calculate FIRST(E') = {+ - * / ε } and the FOLLOW(E') = FOLLOW(E) = {$) + - * /}.

Example

Naturally,
$P[E'][\pm] = \{E' \rightarrow +EE', E' \rightarrow \varepsilon\}$ and
$P[E'][*/] = \{E' \rightarrow *EE', E' \rightarrow \varepsilon\}$.
We may drop the ε-productions from these four places and get a nice parsing table[a].

[a] But it does not work for all grammars. Consider $S \rightarrow aSa \mid bSb \mid \varepsilon$.

Note

It seems that the removal of two ε-production disambiguates the grammar. The corresponding unambiguoes grammar G_2 is as follows:

$$E \rightarrow (E)E' \mid icE' \mid (E) \mid ic$$
$$E' \rightarrow +E \mid -E \mid *E \mid /E \mid \varepsilon$$

We have $L(G_1) = L(G_2)$ and FOLLOW(E') = `{$ )}`, so there is no multiple entries in the table[a].

Error Recovery

- The token on the top of stack does not match with the token in the input stream.
- The entry in the parsing table corresponding to nonterminal on the top of stack and the current input token is an error.