CS164 Programming Languages and Compilers **Spring 2024**

Discussion Worksheet 3: Week of 2/13

In this discussion, we will go through an in-depth example of building a predictive parser for an LL(1) grammar, i.e. LL(1) parsing. Recall that in LL(k):

- first L means "left-to-right" scan of the input
- second L means "leftmost derivation"
- and (k) means "predict based on k tokens of lookahead"

An LL(1) language is one for which we can build a predictive parse for with only 1 token of lookahead, i.e. given a current nonterminal and the next token in the input string, we know which production rule to use.

1 Left-Recursion and Left-Factoring

A left-recursive grammar is one such that, for some non-terminal S there is a derivation $S \rightarrow^* S\alpha$ for a grammar, where α is any sequence of terminals and nonterminals.

These languages are not LL(1) since it would be valid to derive $S \rightarrow S\alpha \rightarrow S\alpha\alpha \rightarrow \cdots$, and thus infinitely recurse on matching S before actually attempting to match a character from an input stream.

We can re-write a left-recursive grammar of the form $S \rightarrow S\ \alpha \mid \beta$ to a right-recursive one of the form:

$$S \rightarrow \beta\ S'$$
$$S' \rightarrow \alpha\ S' \mid \varepsilon$$

Exercise 1 The following grammar is such a left-recursive grammar. Rewrite its rules and give an equivalent grammar which is no longer left-recursive. Do not left-factor the grammar for now. (Assume that the start nonterminal is S, and that the lowercase names `aj`, `v`, `n`, and `c` refer to terminals.)

Note that the grammar is ambiguous due to the parse rule "$S \rightarrow S\ \texttt{c}\ S$". When you rewrite the grammar, this rule should be replaced with an equivalent rule which forces c to be a right-associative operator.

$$NP \rightarrow NP\ \texttt{aj} \mid \texttt{n}$$
$$VP \rightarrow \texttt{v}\ NP$$
$$S \rightarrow NP\ VP \mid S\ \texttt{c}\ S$$

The simplified grammar is still unusable by an LL(1) parser generator. We have multiple productions that have the same possible starting symbols, and therefore cannot determine which production to derive by only looking at the current symbol in the input stream.

Exercise 2 Left-factor the grammar to fix this problem.

2 First and Follow Sets

A *First set* of A gives the set of all terminals that can be the first terminal in some expansion of A. The formal definition is $\mathit{First}(X) = \{b | X \rightarrow^* b\alpha\} \cup \{\varepsilon | X \rightarrow^* \varepsilon\}$. A procedure to compute them is:

1. $\mathit{First}(b) = \{b\}$
2. For $\alpha = A_1, \ldots, A_n$, do the following
   - Add $\mathit{First}(A_i) - \varepsilon$ to $\mathit{First}(\alpha)$. Stop if $\varepsilon \notin \mathit{First}(A_i)$; else repeat for A_{i+1}
   - If we have reached all the way to A_n without stopping ($\varepsilon \notin \mathit{First}(A_i)$ for all i), then add ε to $\mathit{First}(\alpha)$
3. If there exist production rules $X \rightarrow \alpha_1$, $X \rightarrow \alpha_2, \ldots, X \rightarrow \alpha_n$, then $\mathit{First}(X) = \mathit{First}(\alpha_1) \cup \mathit{First}(\alpha_2) \cup \cdots \cup \mathit{First}(\alpha_n)$

The *First set* helps us determine which production rule to choose given the next token in the string.

A *Follow set* of A gives the set of all terminals that can be the first terminal *after* A is fully expanded. The formal definition is $\mathit{Follow}(X) = \{b | S \rightarrow^* \beta X b \omega\}$. A procedure to compute them is:

1. $\mathit{First}(b) = \{b\}$
2. For all productions $Y \rightarrow \ldots X A_1 \ldots A_n$, do the following
   - Add $\mathit{First}(A_i) - \varepsilon$ to $\mathit{Follow}(X)$. Stop if $\varepsilon \notin \mathit{First}(A_i)$; else repeat for A_{i+1}.
   - If we have reached all the way to A_n without stopping ($\varepsilon \notin \mathit{First}(A_i)$ for all i), then add $\mathit{Follow}(Y)$ to $\mathit{Follow}(X)$.

This procedure may take a few rounds to converge if Y and X occur in production rules for each other. In the case that the next token is not in the First set of A, the *Follow set* helps us determine whether there will be a production rule that can consume the token. If the next token is neither in the first nor the follow sets, this designates a parser error.

Exercise 3 Create the First and Follow sets for each nonterminal, for the grammar from Exercise 2. You may need several rounds to converge to the follow sets.

3 LL(1) Parsing

We can contstruct the parsing table T for CFG G by, for each $A \to \alpha$ in G:

- For each terminal $b \in \mathit{First}(\alpha)$, add $T[A, b] = \alpha$.
- If $\varepsilon \in \mathit{First}(\alpha)$, for each $b \in \mathit{Follow}(A)$, add $T[A, b] = \alpha$.

Exercise 4 Now that you have the First and Follow sets for each nonterminal, fill in the LL(1) parsing table.

	n	v	aj	c	$
NP'					
NP					
VP					
S'					
R					
S					

Exercise 5 Use the parse table to parse the following string. Draw the resulting parse tree. **You will need a lot of space.**

`n aj aj v n c n v n aj`