Jim Lambers
CS143 Compilers
Summer Quarter 2000-01
Homework Assignment 1 Solution

This assignment is worth 100 points.

1. The ANSI C specification dictates that nested comments are not permitted. In other words, once a `/*` has been seen in the input to begin a comment, the comment extends to the first `*/` that occurs, even if another `/*` has been seen. Some languages, or compilers for languages, allow nested comments, so that each `/*` must be matched by a `*/`.

   (a) Write a C program that will compile regardless of whether the compiler allows nested comments. If the compiler does support nested comments, the program must print "Cheep cheep!" Otherwise, it must print "Moo!"

   This code does the job:

   ```
   main()
   {
       int a = 2;
       if ( /* /*/ a*/* */1 == 2 )
           printf( "Moo!\n" );
       else
           printf( "Cheep cheep!\n" )l
   }
   ```

   (b) Explain why the language of nested C comments cannot be described using a regular expression. Can it be described using a context-free grammar?

   The language of nested comments can't be described by a regular expression because regular expressions can only be used to denote a fixed number of repetitions or an arbitrary number of repetitions of a given construct. The language of nested commends is analgous to the language of strings of balanced parentheses in this respect: no regular expression can "remember" how many /*'s need to be matched by */.

A rigorous proof of this can be obtained by trying to construct a DFA that recognizes nested comments. It must have a finite number of states, say k, and distinct states are necessary for different nesting levels, since for each nesting level, a different path must be taken to an accepting state. When attempting to nest more than k levels, you obtain two paths from the inital state to the final state but only one of them describes a valid string, so the DFA accepts strings not in the language.

Nested comments can be described (loosely) by a context-free grammar such as:

$$
\begin{array}{rcl}
S & \rightarrow & bSe \quad | \quad CS \quad | \quad \epsilon \\
b & \rightarrow & /* \\
c & \rightarrow & */ \\
C & \rightarrow & \text{char other than } * \text{ or } / \quad | \quad * \text{ not followed by } / \quad | \\
 & & / \text{ not followed by } *
\end{array}
$$

2. Consider the following grammar:

$$R \rightarrow R'|'R \quad | \quad RR \quad | \quad R^* \quad | \quad (R) \quad | \quad a \quad | \quad b \quad | \quad c \quad | \quad '\epsilon' \quad (1)$$

Note that the first | is a terminal symbol denoting the character |, rather than a separator between productions. Also, the ϵ denotes the character ϵ, rather than an ϵ-production.

(a) Show that this grammar generates all regular expressions over the alphabet $\Sigma = \{a, b, c\}$.

The idea is to show that R derives in zero or more steps all regular expressions, and no other strings. It clearly derives the basic expressions, epsilon and the alphabet symbols. Using the other productions, we see that for any strings r and s that R derives, it also derives $r|s$, rs, $r*$, and (r). Thus the language of strings derived by R mimics the construction of the set of all regular expressions over the alphabet.

(b) Show that this grammar is ambiguous.

A regular expression such as $ab*c$ will have more than one parse tree:

$$\begin{aligned} R &\Rightarrow RR \\ &\Rightarrow RR*R \\ &\Rightarrow ab*c \end{aligned}$$

or

$$\begin{aligned} R &\Rightarrow RR \\ &\Rightarrow R*R \\ &\Rightarrow RR*R \\ &\Rightarrow ab*c \end{aligned}$$

(c) Construct an equivalent unambiguous grammar that conforms to the following rules for precedence and associativity:

- The unary operator $*$ has the highest precedence, followed by concatentation, with | having the lowest precedence.
- All operators are left associative.

This grammar does the job:

$$\begin{aligned} S &\rightarrow A \mid S'|'A \\ A &\rightarrow AB \mid B \\ B &\rightarrow B* \mid C \\ C &\rightarrow a \mid b \mid c \mid \epsilon \mid (S) \end{aligned}$$

(d) Using the new grammar, construct a parse tree for the sentence $(ab|b)^*c$.

The derivation for this tree is

$$S \Rightarrow A$$

$$
\begin{aligned}
&\Rightarrow AB \\
&\Rightarrow AC \\
&\Rightarrow Ac \\
&\Rightarrow Bc \\
&\Rightarrow B * c \\
&\Rightarrow C * c \\
&\Rightarrow (S) * c \\
&\Rightarrow (S|A) * c \\
&\Rightarrow (S|B) * c \\
&\Rightarrow (S|C) * c \\
&\Rightarrow (S|b) * c \\
&\Rightarrow (A|b) * c \\
&\Rightarrow (AB|b) * c \\
&\Rightarrow (AC|b) * c \\
&\Rightarrow (Ab|b) * c \\
&\Rightarrow (Bb|b) * c \\
&\Rightarrow (Cb|b) * c \\
&\Rightarrow (ab|b) * c
\end{aligned}
$$

3. Consider the following grammar. As usual, upper case letters denote non-terminals while lower-case letters denote terminals.

$$
\begin{aligned}
S &\rightarrow Em \mid sB \\
B &\rightarrow lCr \\
C &\rightarrow AC \mid \epsilon \\
A &\rightarrow DF \\
F &\rightarrow SF \mid \epsilon \\
D &\rightarrow cEnD \\
E &\rightarrow w \mid aqE
\end{aligned}
$$

(a) Compute the FIRST and FOLLOW sets for each nonterminal.

	FIRST	FOLLOW
S	$s,\ w,\ a$	$\$,\ s,\ w,\ a,\ c,\ r$
B	l	$\$,\ s,\ w,\ a,\ c,\ r$
C	$c,\ \epsilon$	r
A	c	$c,\ r$
F	$s,\ w,\ a,\ \epsilon$	$c,\ r$
D	c	$s,\ w,\ a,\ c,\ r$
E	$w,\ a$	$m,\ n$

(b) Prove that this grammar is LL(1).

This grammar can be shown to be LL(1) by building the parsing table, it will have no multiply defined entries.

(c) Create a LL(1) parsing table for this grammar.

	a	c	l	m	r	s	w	$\$$
S	Em					sB	Em	
B			lCr					
C		AC			ϵ			
A		DF						
F	SF	ϵ			ϵ	SF	SF	
D		$cEnD$						
E	aqE						w	

(d) (BONUS) Show that no LL(1) grammar is ambiguous.

The basic idea is that when building the parse tree, any input symbol provides at most one way of expanding a node and adding child nodes. Thus there is only one way to build a parse tree for any given input.