ECE 573

Problem Set 2: Context free grammars, recursive descent parsers and LL(1) parsers.

For the following problems, consider this context-free grammar:

$$
\begin{array}{lrcl}
1. & S & \rightarrow & Xa \\
2. & X & \rightarrow & YXb \\
3. & X & \rightarrow & \lambda \\
4. & Y & \rightarrow & cd
\end{array}
$$

1. Can this language be expressed as a regular expression? Why or why not?

   Answer: No, it cannot. This language is $(cd)^n b^n a$. We need to find an equal number of cd substrings as bs. A finite automaton cannot capture this language.

2. What are the terminals and non-terminals of this language?

   Answer:

   Terminals: $\{a, b, c, d\}$

   Non-terminals: $\{S, X, Y\}$

3. Show the derivation of the string "*cdcdbba*" starting from S (specify which production you used in each step), and give the parse tree according to that derivation.

   Answer:

Current derivation	**Rule to apply**
S	1
Xa	2
YXba	4
cdXba	2
cdYXbba	4
cdcdXbba	3
cdcdbba	Done!

4. Construct the *first* sets for each of the strings on the right-hand-side of a production rule.

   Answer:

$$
\begin{aligned}
First(Xa) &= \{a, c\} \\
First(YXb) &= \{c\} \\
First(\lambda) &= \{\lambda\} \\
First(cd) &= \{c\}
\end{aligned}
$$

5. Construct the *follow* sets for each of the non-terminals in the language.

   Answer:

$$\begin{aligned} Follow(S) &= \{\} \\ Follow(X) &= \{a, b\} \\ Follow(Y) &= \{b, c\} \end{aligned}$$

6. Give the LL(1) parse table for the grammar. Use the first/follow sets you constructed in the previous two steps.

   Answer:

	a	**b**	**c**	**d**
S	1		1	
X	3	3	2	
Y			4	

7. Write a recursive descent parser (in pseudocode) which will accept strings in this language. Assume, as we did in class, that there is a scanner which provides the functions `match`, which matches a token, and `peek`, which returns the next token in the string.

   Answer:

   Left as an exercise—this is fairly straightforward given the above parse table.

8. Show the steps that an LL(1) parser will take to parse the string *cdcdbba*

   Answer:

Parse stack	**Remaining input**	**Action**
S	cdcdbba	Predict 1
Xa	cdcdbba	Predict 2
YXba	cdcdbba	Predict 4
cdXba	cdcdbba	match(cd)
Xba	cdbba	Predict 2
YXbba	cdbba	Predict 4
cdXbba	cdbba	match(cd)
Xbba	bba	Predict 3
bba	bba	match(bba)
		Accept!