

CS385-Assignment 5

Due Thursday, October 11

The basics of Context-Free Languages.

1. Exercise 2.1.

Make sure you can do all the parts of Exercise 2.3 (you don't need to hand it in).

2. Exercises 2.4(b,f). Observe that (b) is a regular language. You can approach these problems concerning grammars for regular languages, since there is a simple mechanical way to derive a CFG from an NFA or a regular expression, but here it's just as simple to guess. The language in (f) is *not* regular.

3. Exercise 2.6 (b). As a hint, observe that there are three different ways that a string can fail to be in $\{a^n b^n : n \geq 0\}$. Either it consists of a 's followed by b 's with more a 's than b 's, or with more b 's than a 's, or it doesn't follow the pattern of a 's followed by b 's—that is, it contains ba somewhere.

4. Show that if L is a CFL, then so is L^R . (Explain how to turn a grammar for L into a grammar for L^R .)

5. Problem 2.27. This is starred, but you probably already know something about this problem. It is the classic “dangling else” ambiguity present in many programming languages. The problem is that in a construct with **if** . . . **if** . . . **else**, which **if** does the **else** go with? Show how to get two different derivation trees for a string in $L(G)$ that exhibits this ambiguity. To disambiguate (love that word!) try to design a grammar that forces the conventional interpretation: The **else** goes with the *closest if*.

6. Here's a tough one: Devise a context-free grammar for the set L of strings over $\{a, b\}$ in which the number of a 's is equal to the number of b 's. As a hint, let's do this like the balanced parenthesis problem: If the string $w \in L$ starts with a , keep track of the excess of a 's over b 's, and look at where this excess becomes 0 again. We thus get $w = aubv$, where u and v both have the same number of a 's and b 's. You can reason similarly for strings in L that start with b . Use these observations to produce a grammar. Is the grammar ambiguous? If not, explain why not; If so, show this by giving a string with two different derivation trees.

7. And now here's an easier one. *Postfix expressions* are formed using letters as operands and two binary operators $+$ and \times , and always writing the operator

immediately after its two operands. No parentheses are used. For instance

$$ab + cde \times + \times$$

denotes the same thing as

$$(a + b) \times (c + (d \times e)).$$

Write a CFG for the set of postfix expressions, and use it to produce a derivation tree for the example above. Is this grammar ambiguous? Explain.