

## Homework 7, CS301, McConnell, Spring '09

Due Friday 4/24 at 4:00 at the front desk of the C.S. department. I'll post solutions after that.

1. Either use the pumping lemma to show that the following language is not context-free, or show that it is by giving a context-free grammar for it.

$$\{a^i b^j a^i b^j \mid i, j \geq 0\}$$

2. Either use the pumping lemma to show that the following language is not context-free, or give a grammar for it:

$$\{a^i b a^i b \mid i \geq 1\}.$$

3. Use the pumping lemma to show that the following language is not context-free:

$$\{a^i \mid i \text{ is a power of two that is greater than or equal to } 1\}.$$

4. Show that the intersection of two context-free languages is not necessarily context-free by giving context-free grammars for the following languages, and then showing that their intersection is not context-free:  $\{a^n b^m c^k \mid n = m\}$  and  $\{a^n b^m c^k \mid m = k\}$ .

5. Use the construction of Theorem 3.5.1 to find context-free grammars for the following languages:

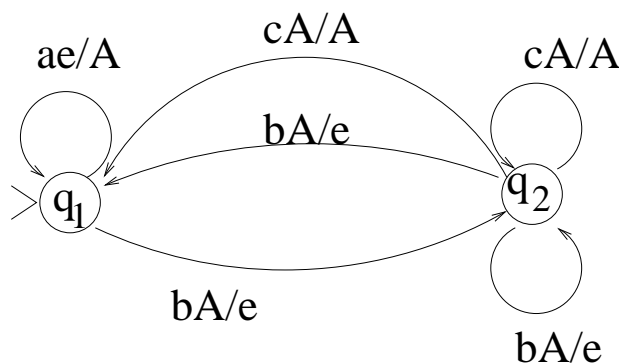
(a)  $\{a^n b^n a^m b^m \mid n \geq 1, m \geq 0\}$ . *Hint: this is the concatenation of the language  $\{a^n b^n \mid n \geq 1\}$  and the language  $\{a^m b^m \mid m \geq 0\}$ .*

(b)  $\{a^n b^m c^k \mid n, m, k \geq 1 \text{ and } (n = m \text{ or } m = k)\}$  *Hint: this is the union of two languages, one where  $n = m$  and the other where  $m = k$ . The first of these is the concatenation of  $\{a^n b^n \mid n \geq 1\}$  and the language  $cc^*$ .*

(c)  $\{a^n b^n \mid n \geq 1\}^*$ . (Notice the Kleene star.)

6. In my posting on constructing a context-free grammar from a PDA, study the relationship between the example PDA near the end and the rules I generate for it. Make guesses about the general strategy. Refer to earlier in the handout for confirmation of your guesses, or for an explanation.

Consider the following PDA:



- (a) Each nonterminal that we need to generate for a PDA is of the form  $[q, e, r]$  or  $[q, B, r]$ , where  $B$  is any stack symbol, and  $q$  and  $r$  are any states and needn't be distinct. (In the case of the above PDA, the only stack symbol is  $A$ .)  
Use a cartesian product to generate a list of the possible possible nonterminals that can appear in the grammar for it. Use the notation the handout uses for the nonterminals. If you list some nonterminals that aren't actually used, you won't be counted off, as long as you list all the ones that are used.
- (b) Some of the rules say a successful computation corresponds to a walk that starts in  $q_1$  with an empty stack and ends in any state with an empty stack. Write these rules, using  $S$  as the start symbol.
- (c) One of the rules generates the language of all strings that can be consumed during a successful walk that begins by taking the loop that reads  $a$  at  $q_1$ , starting with an empty stack, and ends at  $q_1$  with an empty stack. Write this rule, using our notation for the nonterminals.
- (d) One of the rules generates the language of all strings that can be consumed during a successful walk that begins by taking the loop that reads  $a$  at  $q_1$ , starting with an empty stack, and ends at  $q_2$  with an empty stack. Write this rule, using our notation for the nonterminals.
- (e) One of the rules reflects the fact that you can get from  $q_1$  to  $q_1$  starting with an empty stack and ending with an empty stack by taking no action. Write this rule.
- (f) One of the rules generates the language of all strings that can be consumed during a successful walk that begins at  $q_2$  with just an  $A$  on the stack, ends at  $q_2$  with an empty stack, and begins with the transition from  $q_2$  to  $q_1$  that reads  $b$ .
- (g) Some of the rules generate the language of all strings that can be consumed during a successful walk that begins at  $q_1$  with a single  $A$  on the stack, and ends at  $q_2$  with an empty stack. Write these rules, using our notation.

7. Here's the gist of section 3.8, with a few modifications to make it easier. Recall that the *configuration* of a PDA consists of its current state, the current string of letters on its stack, and the string of remaining letters to read. A *move* is an action that changes the configuration. A *deterministic* PDA is a one-at-a-time PDA that starts out with a special symbol,  $Z$ , on the bottom of the stack, has at most one move that it can take for any configuration, and accepts on an empty stack. Which move it can take depends on the next input letter and/or the symbol at the top of the stack.

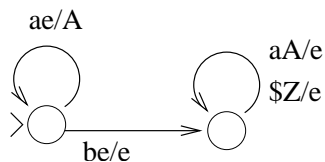
If it finds itself in a configuration that doesn't allow it any moves, it halts. It accepts if it halts on an empty stack with nothing left on the input.

Deterministic PDA's (DPDA's) are the PDA's of interest in the theory of compiler design.

A problem arises when some string  $x$  in the language it is supposed to accept is a prefix of another string,  $xy$ , that is also in the language. When it has read  $x$ , how does it know whether it's supposed to halt? If there are no more characters, trying to read a character to find this out is not an allowed move.

To avoid this, we append a special character, \$, to the end of each string placed on the input. This makes sure that no string that can be placed on the input can be a prefix of any other. It accepts  $w$  if and only if it halts with an empty stack after reading  $w\$$ .

Here's an example of a DPDA for the language  $\{a^n b a^n | n \geq 0\}$ :



Whenever a string is not in the language, it finds itself with no allowed move even though its stack is nonempty or there are characters left to read on the input.

- (a) Modify this PDA to get a DPDA for the language  $\{a^n b^m a^n | n \geq 0 \text{ and } m \geq 1\}$ .
  - (b) Draw one for the language  $\{w c^k d^j w^R | w \in \{a, b\}^* \text{ and } k \geq 0 \text{ and } j \geq 1\}$  (Recall that  $w^R$  is the reverse of  $w$ .)
  - (c) Draw one for  $\{a^m b^n | m, n \geq 0 \text{ and } m \neq n\}$ .
  - (d) Draw one for  $\{a^m c b^{2m} | m \geq 0\}$ .
8. A *deterministic context-free language* is one that's accepted by a deterministic PDA. Given what we know about NFA's and DFA's, you may suspect that the deterministic context-free languages are just the context-free languages. Let's see if this is so.

A fact that we will not prove is that the complement of every deterministic context-free language is a deterministic context-free language. The trick is based on the usual strategy: show how to convert a deterministic PDA for the language into one for the complement. It's more complicated than the way we turned a DFA for a regular language into one for the complement, however. One reason for this is that there is nothing in the definition of a DPDA that says that it can't get into an infinite loop or otherwise fail to halt, just as there is nothing in the definition of Java that says a Java program has to halt. (A Java program is also deterministic.) A string that a DPDA doesn't halt on is not in the language it accepts. Turning a DPDA into one for the complement has to deal with how to make the new DPDA accept these strings.

Now look at Theorem 3.8.2 and its proof.

The proof claims that  $\{a^n b^m c^k | n \neq m \text{ or } m \neq k\}$  is "obviously context-free." Show this by giving a grammar for this language. *Hint: refer to the tricks of problem 5.*