ECS120 Introduction to the Theory of Computation
Fall Quarter 2007

Homework 2 Help
Due Friday October 12, 2007

Problem 2.1

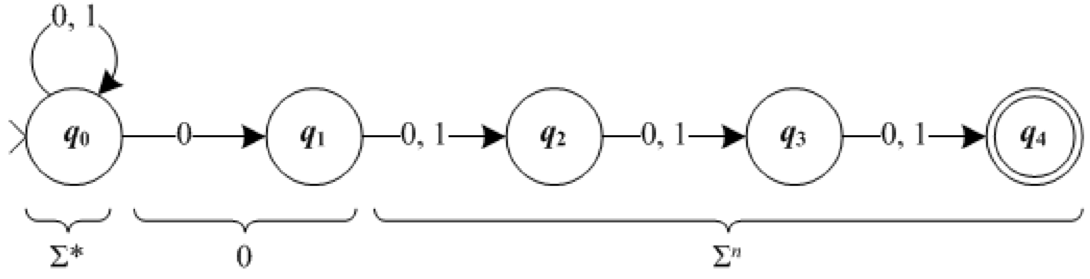
Show that if $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA accepting the language L then $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$ is a DFA accepting the complement of L, $\overline{L}$.

This problem asks you to "show" that this is true, so you do not need a rigorous proof. However, you will still need to use formal elements/definitions to make your argument.

Problem 2.2

(a) Show that there is an $(n+2)$-state NFA for $L_n = (\Sigma^*)0\Sigma^n$ where $\Sigma = \{0, 1\}$.

Draw out a couple NFAs and then generalize. For example, here is an example NFA for $n = 3$ with $n + 2 = 5$ states:



(b) Prove that any DFA for L_n requires at least 2^n states.

You'll be using the same technique to prove that a DFA is minimal. For example, you might want to start with something like:

> Assume (for the sake of contradiction) that there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ for L_n that has fewer than 2^n states.

Eventually, you'll want to use the pigeonhole principle:

By the pigeonhole principle, some two of these strings x_i and x_j (where $x_i \neq x_j$) must collide such that $\widehat{\delta}(q_0, x_i) = \widehat{\delta}(q_0, x_j)$.

At some point, you'll need to show that for every possible pair of strings x_i and x_j, there is a contradiction such that one string (with padding) $\in L_n$ but the other $\notin L_n$. (*Just how many pairs of strings are there anyway?*)

The challenge here is to choose the padding correctly so that this will always work. What do you know about x_i and x_j? Well, for starters, we know they are different strings. Go from there, and good luck!

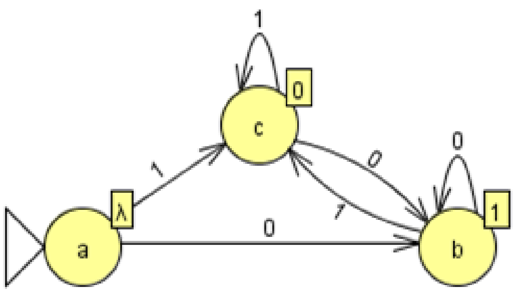
Problem 2.3

A *Moore Machine* is like a DFA except that we associate an output string with each state. The machine outputs that string when it leaves a state. Thus a Moore machine M computes a function $f_M : \Sigma^* \rightarrow \Gamma^*$.
(a) Formally define a Moore machine M and the function f_M which it computes.

See the definitions in the book for examples. It must be a complete formal definition! This means also formally defining any functions involved.

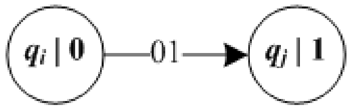
(b) Give a Moore machine which adds two binary numbers. The input to the Moore machine is a string $x = a_1 b_1 \cdots a_n b_n \#$, for some $n \geq 1$. The symbol # is just a formal symbol in the alphabet; each a_i and b_i are binary digits. The output is a string $c_1 \cdots c_{n+1}$ which encodes the sum of $a = a_n \cdots a_1$ and $b = b_n \cdots b_1$. By convention, we feed in the numbers least-significant-bit to most-significant-bit, which is the same order which the machine produces output. Represent your machine in an easy-to-read manner; describe any conventions you employ.

Here is an example Moore machine (created in JFLAP). The label on each state gives the symbol output by that state. This example flips the input string such that 0101 becomes 1010. (The JFLAP file is also available on the TA website.)

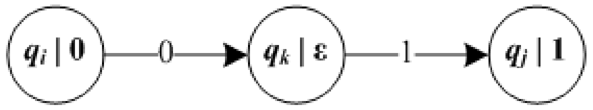


You must create a binary adder using a Moore machine. If you need a refresher on binary addition, try `http://en.wikipedia.org/wiki/Binary_addition#Addition`. The tricky part will be determining how to carry the 1 when adding $1 + 1$.

If you want, you can define a "shorthand" to make things easier. For example, you are always dealing with two input symbols at a time. Therefore, it would be nice to define shorthand like:



which expands to



If you do decide to use this, be sure to define your convention in your homework!

Problem 2.4

(a) Transform the following NFA into a DFA accepting the same language. (Diagram on homework.)

Yes, JFLAP does have a handy "Convert to DFA" option. No, you should not use it. Now, let me tell you why you actually don't want to!

First, it may not be correct. This has been discussed a little bit on the newsgroups. JFLAP does not find and fill in missing transitions in your state machine. The end result may not actually be a complete DFA. Remember, every state in a DFA must have an outgoing transition for every symbol in the alphabet. JFLAP is unable to check or account for this!

Second, homework is worth only 25% of your final grade. The other 75% of your grade comes from exams and quizzes. Therefore, getting 10 points on the homework isn't worth not being able to get 10 points on your exam. You have to (1) know how to do this by hand and (2) be able to finish it in reasonable time. If you just use JFLAP to find the answers, you'll be hurting on the exam!

As said by Filkov on the newsgroups:

JFLAP only answers will NOT be graded!

A printout of the state machine created by JFLAP will not be graded. You need to have actual work to demonstrate you performed the operation yourself!

(b) Find a regular expression for that language.

Don't forget to do this part as well! It is a pretty complicated regular expression, so you may want to play around with JFLAP. Also, Filkov posted on the newsgroups:

> "If there are exceptions (ie some problems are allowed to be solved using JFLAP), I will mention them explicitly in class. **For HW2 the only exception is the reg. exp. part of the last regular problem.**"

Bonus Problem

Prove that if L, a language over Σ, is DFA-acceptable then the language $\text{PAL}(L) = \{x \in \Sigma^* : xx^R \in L\}$ is also DFA-acceptable.

This is the first bonus problem. It is meant to be challenging, and gives everyone a chance for extra points. However, since it is a *bonus* problem, I will provide no help or hints. Also, Professor Filkov personally grades these problems (not me).