CPS 102: Discrete Mathematics	**Instructor:** Bruce Maggs
Assignment 1	**Date:** Monday, September 7th, 2009

1 Designing DFAs (20 points)

Let $\Sigma = \{0, 1\}$. For each of the following languages, give the state diagram for a DFA that recognizes it.
(a) $L_1 = \{w{:}w$ does not contain the substring $110\}$
(b) $L_2 = \{w{:}w$ contains an even number of 0's or exactly two 1's$\}$
(c) L_3 is the language that consists of all strings such that between every two 0s, there is an even number of 1s.
(d) L_4 is the language that consists of all strings w such that w ends in a 1 and w contains an even number of 0s.

2 A Tool for Proving Irregularity (25 points)

When proving that a language isn't regular, a tool that is often used is the following lemma. Below is its formal statement:

If $\mathcal{L}$ is a regular language, there exists a number $p > 0$ such that if $w \in \mathcal{L}$ and $|w| \geq p$, then there are strings x, y, z such that the following hold:

- $w = xyz$
- $|xy| \leq p$
- $|y| > 0$
- $xy^iz \in \mathcal{L}$ *for all* $i = 0, 1, 2, \ldots$

In words, this lemma says that for every regular language $\mathcal{L}$, there is some number p such that every string that's at least p symbols long can be decomposed into three parts meeting certain criteria. A particularly imporant result is that there is some subword y of w that we can repeat to create new strings which still belong to $\mathcal{L}$. This is a powerful result that allows us to prove nonregularity of languages in a concise manner.

(20 points) Prove the above lemma. Think back to the formal argument we gave in class that the language 0^n1^n is not regular and see which parts of the argument correspond to the different pieces of the lemma. Keep in mind the following facts about regular languages:

- Every regular language $\mathcal{L}$ has a DFA $\mathcal{M}$ which accepts it. This machine has some number of states, say m states.

(5 points) Use this lemma to show that the language $a^n b^{2n}$ (the set of strings which consist of some number of as followed by twice as many bs) is not regular.

3 Regular Expressions (20 points)

Regular expressions are a useful tool to computer scientists because they allow us to concisely represent a regular language[1]. In particular, it is often easier to write strings which represent a potentially infinite language than to write down a DFA for that language.

Remember that a language is just a set of strings. Formally, a regular expression over an alphabet Σ can be defined inductively:

- $\emptyset$ is a regular expression and denotes the empty set.
- ϵ is a regular expression and denotes the set containing the empty string, $\{\epsilon\}$.
- For each $a \in \Sigma$, $\mathbf{a}$ is a regular expression and denotes the set $\{a\}$.
- If r, s are two regular expressions that represent the languages R, S respectively, then $r+s$, rs, and r^* are also regular expressions. $r + s$ denotes the set $R \cup S$. rs denotes the set of strings $\{ab \mid a \in R, b \in S\}$. Think of this as the set of concatenations of a string from R and a string from S. Finally, r^* represents the set R^* which is the concatenation of any number (including none) of strings from R. The first few strings in r^* include the empty string, all of the strings of r, all strings which are the concatenation of two strings in r, and so on. Please use the following website to find out more information about the * operator: `http://planetmath.org/encyclopedia/KleeneStar.html`
- We will let Σ be shorthand for any single character in the alphabet.

As examples, $(\mathbf{0} + \mathbf{1})^*$ represents all binary strings (including the empty string), $(\Sigma\Sigma)^*$ represents all strings of even length over an alphabet Σ, and Σ^* represents all strings over the alphabet Σ.

Consider each of the following languages over the alphabet $\Sigma = \{0, 1\}$. If the language is regular, give a regular expression for it. If the language isn't regular, give a proof that it is not.

(a) The set of strings which contain at least four consecutive zeros.

(b) The set of strings w such that if w has even length, w starts with a 1 and if w has odd length, it starts with a 0.

(c) The set of strings w which, when viewed as binary strings written in the usual way (most significant bit first, least significant bit last), represent numbers that are divisible by 3.

(d) The set of strings which contain at least twice as many 1's as 0's.

[1] To prove this equivalence in a reasonable manner requires an extension of DFA's called *nondeterministic finite automata* (NFA) and a little more machinery. If we measure the complexity of an NFA by the number of states it has, and the complexity of a regular expression by the number of symbols it takes to write it down, a regular expression for a language could be exponentially larger than an NFA for that language. Take 15-453 with a subset of the Blum family to see a full proof of this fact.

4 DFAs can verify addition (15 points)

DFA's "can't count," but they can verify addition. Consider an alphabet with rather interesting symbols:

$$\Sigma = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

which represent columns which could appear when writing the addition and result of two binary numbers. An input of length n represents the addition of two n-bit numbers where the input is given from least to most significant bit. The first number's binary representation is in the top row, the second number's representation is in the middle row, and the result is in the bottom row. Let $\mathcal{L}$ be the language of strings over this alphabet which represent correct addition statements as described above. As an example, correctly representing $4 + 2 = 6$ in this scheme would give the following string (first input symbol on the left)

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The following input would represent $4 + 5 = 8$, and thus this string should not be in $\mathcal{L}$:

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Write a DFA which accepts $\mathcal{L}$. Keep your DFA small; machines with more than 5 states will be penalized.

5 DFAs cannot verify multiplication! (10 points)

We've proven that the language of valid addition statements in our vector alphabet is regular. It is somewhat surprising that the language of valid multiplication statements is *not* regular. Prove that the language of valid multiplication statements is not regular, where the semantics of the strings is the same as above.

Here is an example of the valid multiplication statement $3 \times 4 = 12$:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

The multiplication statement $3 \times 2 = 9$ is invalid:

$$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

6 Regular languages and set operations (15 points)

It's an amazing fact that for the class of regular languages (also called regular sets) a plethora of so-called *closure properties* hold. Namely, if R and S are regular languages, then the following are also regular languages:

- $\overline{R}$, the *complement* of R, consisting of those strings not in R (sometimes denoted R^c).
- R^*, the *Kleene closure* of R as described in question 2.
- $R \cdot S$, the concatenation language of R and S.
- $R \cup S$, the union language consisting of strings s that are in R or in S
- $R \cap S$, the intersection language consisting of strings s that are in both R and S
- $R^{\mathcal{R}}$, the reversal language consisting of the reverse of all strings in R.
- $R \setminus S$, the difference language consisting of strings s that are in R but not in S.
- $R \Delta S$, the symmetric difference (xor) of two languages is the language $(R \setminus S) \cup (S \setminus R)$.

Without any additional tools beyond DFAs we can prove several of these facts, as seen in lecture. The goal here is to give a *rigorous* argument for why these constructions work- we expect you to be more detailed than we were in class!

Let me define a few additional concepts before you get going. In class, we loosely described the notion of acceptance as the machine ends in a final state when it has no more input. We can formally define this concept as follows. Suppose we are given a DFA $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$. Define the function $\delta^* : Q \times \Sigma^* \to Q$ inductively as

$$\delta^*(q, \epsilon) = q$$

$$\delta^*(q, \sigma w) = \delta^*(\delta(q, \sigma), w), \qquad \sigma \in \Sigma$$

The δ^* function takes in a state and a word and tells us where the machine ends a computation if we were to start in state q and read in the word w. Now we can formally describe the acceptance as $w \in \Sigma^*$ is accepted if and only if $\delta^*(q_0, w) \in F$.

(a) 3 points Prove that the complement operation is a regular operation

(b) 7 points Prove that the union operation is a regular operation

(c) 5 points Consider the following construction for the reversal operation: Given a regular language $\mathcal{L}$ there is some DFA $\mathcal{M}$ whose acceptance language is $\mathcal{L}$. We can reverse all of the arrows in $\mathcal{M}$ and make the start state a final state. Furthermore, collect all final states of the original machine into a single state, and make it the start state in the augmented machine. This DFA accepts the reversal language.

Does the above construction work? Either prove that it is correct or explain why it is incorrect.

7 Being artistic (15 points)

Draw finite state machines for the following languages. Assume that the languages are over the alphabet $\Sigma = \{0, 1\}$. Please keep your machines small. Contrary to the title of the question, artistic renditions of finite state machines may make the TA's happy, but may not yield a higher score.

(a) 7 points $\mathcal{L} = \{w \mid$ w is a binary number divisible by 3, given least significant digit first$\}$. Examples in $\mathcal{L}$: 0, 11, 1001, 1111, 011. Examples not in $\mathcal{L}$: 01, 111, 0001.

(b) 4 points $\mathcal{L} = \{w \mid w$ contains an even number of 1s or exactly one 0$\}$. Examples in $\mathcal{L}$: 11, 10, 101, ε. Examples not in $\mathcal{L}$: 111, 00, 1.

(c) 4 points $\Sigma^*\mathbf{0}\Sigma^*\mathbf{1}\Sigma^*\mathbf{0}\Sigma^*$ Examples in $\mathcal{L}$: 010, 000110, 11011001. Examples not in $\mathcal{L}$: 111, 011, 00011.

8 Star-Craziness (EXTRA CREDIT) (10 points)

Let L be an arbitrary subset of 0^*.

(a) 2 points Is L necessarily regular (over the alphabet $\Sigma = \{0\}$)?

(b) 8 points Is L^* necessarily regular (over the alphabet $\Sigma = \{0\}$)?

This question is hard. Don't attempt it until you have tried all the other problems. As always, try small examples. My advice to you is to write down some *simple* examples of languages L and look at the Kleene closure of that language to hone your intuition.

Each part requires *proof*, which means that a yes/no/handwaving answer will get you no points.