MA/CSSE 474 Homework #5 54 points **Updated for Summer, 2018**

A note that I posted on Piazza in Winter 2015-16:

HW5 has only three problems, and the first one is short and fairly easy once you read the algorithm in the textbook.

There is a reason why this assignment has only two other problems. Both are challenging and both will take a while to digest what I am asking you to do, then to figure out how to do them and how to write them up.

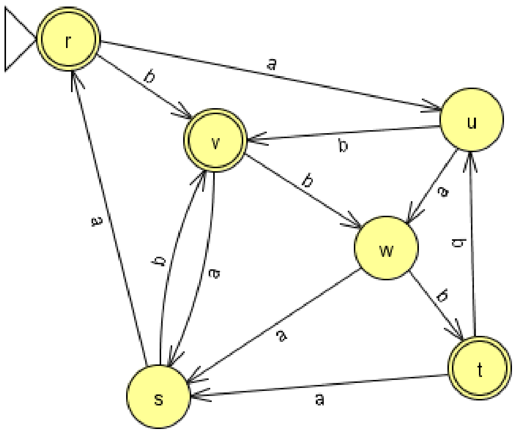
Even if you have not finished HW4, I suggest that you read problems 2 and 3 from HW5 **today**. You don't have to begin solving them today, but you should at least understand the problems and know what a solution to each will involve. **Ask questions soon about anything in the problem descriptions that you do not understand.** If the problems are firmly embedded in your mind, your brain may keep working on them even when you are not consciously doing so.

I also suggest that you block out a significant amount of time this weekend to work on these problems. Not necessarily one big chunk of time; several smaller chunks with some "incubation time" in between may be best.

The Problems

1. (t-6) The DFSM pictured below is minimal, and r is its start state. Construct the canonical form representation of this DFSM as described in Section 5.8 of the Rich textbook. Each vertex in the graph will be given a new name, one of {q0, q1, …, q5}. Complete the following table to indicate which new name will be given to which state. Assume that *a* comes before *b*.

Old state name	New state name
r	
s	
t	
u	
v	
w	



Example: http://www.rose-hulman.edu/class/cs/csse474/201830/Resources/DFSM-Canonical-Form/DFSM-canonical-form.pdf

Note: For many problems, most of the credit will be for having the right ideas. Your score for *the next* problem will very much depend on precise mathematical formulation and careful proofs. My solution occupies more than a page (using 11-point font), and I do not see how a careful formulation and proof can be a whole lot shorter.

2. (t-9-9-15) Let $M_1=(K_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2=(K_2, \Sigma, \delta_2, s_2, A_2)$ be DFSMs that accept the regular languages $L_1 = L(M_1)$ and $L_2= L(M_2)$.
   a. Show that $L = L_1 \cap L_2$ is regular by carefully constructing a DFSM $M=(K, \Sigma, \delta, s, A)$ such that $L=L(M)$. The idea is that each state of M is an ordered pair of states from the other machines. Give the details of the construction (i.e. define each of the five parts of M) using precise mathematical notation.

      HINTS:
      i. The construction from Sessions 7-8 of the minimal DFSM based on the equivalence classes of a regular language can give you a model for how to express this. But there is probably no place for Myhill-Nerode style equivalence classes in this particular problem or its solution. Your construction of the 5 parts of M should be based on the 5 parts of M_1 and M_2.
      ii. The key to the construction is figuring out what the transition function δ for the intersection machine should be, based on δ_1 and δ_2 from the original machines.
      iii. Cartesian product of states.
   b. Let M_1 be the 3-state DFSM that accepts { $w \in \{a,b\}^*$: length of w is a multiple of 3}, and let M_2 be the 3-state DFSM whose diagram is shown on page 58 of the textbook. M_2 accepts { $w \in \{a,b\}^*$: every *a* in w is followed by a *b*}. Using your construction in part a, show a transition table or transition diagram for the machine (based on your construction in part a) that accepts the intersection of the languages accepted by M_1 and M_2.
   c. Carefully prove that $L=L(M)$. You will need to use mathematical induction (induction on the length of a string) somewhere in your proof. There will be some similarities between the general approach of this proof and the $L=L(M)$ proof for the minimal DFSM construction, but not many similarities in the details.
      The similarity is that you will need to prove a lemma by induction that is more general than what is needed to solve this problem, then use a special case of that lemma to get the desired result.

What you need to show: Computations in your new DFSM simultaneously mirror computations in the two original DFSMs, and accept iff both computations from the original machine would accept.

Hint for the lemma: Ultimately we only care about computations that begin at the start state. But it is difficult to show what we need by induction if we only consider those. SO you need to make and prove a statement about the relationship between paths through the new DFSM (starting at any state) and paths in the original DFSMs.

Writing this proof: Probably by the time you get to this point, how the new DFSM works will be obvious. Then you have this problem about half done. The issue now is how to carefully write it up, using one of the notations for a computation (the textbook's $\vdash$ notation or the $\hat{\delta}$ notation described below) and the notation of your constructions from part a,

A possibly helpful notation for part c. Extended delta function. An alternative to the textbook's $\vdash$ notation. For any DFSM with transition function δ, define $\hat{\delta}$ by

$$\hat{\delta}(q, \varepsilon) = q, \text{ and } \quad \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$$

for all states $q \in K$, strings $x \in \Sigma^*$ and symbols $a \in \Sigma$. It should be clear (and could be proven by induction, but you don't have to) that if M's start state is s, and if w is any string in Σ^*, then $\hat{\delta}(q, w)$ is the state that M will reach when M is started in state s with input w.

Of course part b can also be done using the textbook's $\vdash$ "yields" notation instead of $\hat{\delta}$.

3. (t-15) Use the Myhill-Nerode theorem (instead of the pumping theorem, which we will study later) to show that the language

$\Sigma = \{a\}$, $L = \{a^p : p \text{ is a positive prime integer }\}$

is not regular. **In particular, show that for any pair of different positive prime integers p and q with p<q, the strings a^p and a^q are in different equivalence classes of $\approx_L$.**

No fancy number theory is needed for this problem. You only need to know and use the definitions of "prime" and "composite".

Reminder: 1 is not a prime integer. The first few positive prime integers are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37.

Your proof should not mention the word "state"; it should directly use the definition of $\approx_L$.

Hints: Let k = q - p. Here are a couple of possible approaches. (You may come up with something different):

Approach 1. Assume that there is such a pair of primes p and q that are in the same equivalence class; show that this leads to a contradiction. **A start:** If q and p are in the same equivalence class, what can you say about p + k?

Approach 2: Of course if there is a distinguishing string for a^p and a^q, it will be a^t for some t. Your job is to find a non-negative integer t such that concatenating a^t onto a^p and a^q produces one string of prime length and one whose length is composite. Obviously this value of t will depend on p and q. So you must show (as a formula or algorithm) how to find the t for each p and q, and convince me that it is correct. This is not trivial. **Hint for the hint:** For any n and m, we can find n consecutive composite numbers that are at least as large as n.
Finding the distinguishing t for a few particular p-q pairs may help you discover the general pattern, but you will not receive much credit for only doing some specific cases.

Some past questions and answers from Piazza:

Q2: "Each state of M is an ordered pair of states from the other machines"

How can a state be an ordered pair of states? Would the start state s then be (s1, s2)?

Answer: A state from a gven machine can't be an ordered pair of its own states. But a state from a new machine can certainly be an ordered pair of states from two other machines. This is somewhat akin to the NDFSM->DFSM construction, where each state in the DFSM is a set of states from the NDFSM.
Yes, (s_1,s_2) is the start state for the intersection machine.

Q2 part b

After doing part a, is there a way to determine what lemma you need?
We've done one in class and for the practice test, but I haven't been able to figure out how you can come up with any lemma. I know what we have to prove, L=L(M), but I don't know how to find a lemma that would help me prove it.

Answer: The theorem is about what happens after starting in the start state and doing a computation on the entire input string. "Processing the entire string" is something that we can't really look at by induction, But "the result of processing a string, starting in a state of the intersection machine" and how that relates to what happens when processing the same string starting in related states of the original machines, that is something we can show by induction on the length of the string. So the lemma should be something like that.