MA/CSSE 474 Homework #2 (51 points total) Submit your solutions to the HW2 drop box on Moodle. 3.1 means Exercise 1 from Chapter 3 of the Elaine Rich textbook.

Please reread the instructions that precede the problem list in the HW1 assignment sheet. They apply here also.

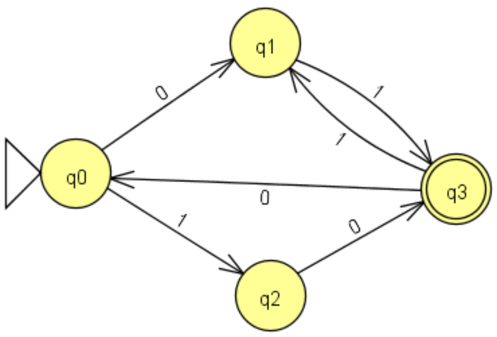
A separate document contains the statements of the textbook problems. See HW1 for a note on why it is there.

Key: (No symbol) Not required to be turned in. Just be sure that you can do it.
(t -12) To be turned in and graded, worth 12 points.

1. 3.1 *Description of a circuit problem as a language to be decided.*
2. (t-3) 3.2 *Square root as a language recognition problem.* (you should only deal with square roots that are integers)
3. 3.4 *OCR as a language recognition problem.*
4. (t-3) 3.5 *Why a PDA doesn't work for $A^nB^nC^n$.* This requires only a high-level understanding of what a PDA is (as given in section 3.3.2). A formal proof is not required.
5. (t-6) 3.6 *2-stack PDA for $A^nB^nC^n$* . This also requires only a high-level understanding of what a PDA is (section 3.3.2)
6. 4.1 *List of integers contains a factor of n?*
7. (t-6) 4.2 *Does Java program p output anything?*
For part b, if the answer is yes, briefly explain how the procedure would work. If no, a simple "no" answer is sufficient.
8. (t-3) 4.3 *chop(L).* See Example 4.10 for the definition of chop.
9. 4.4 *Closure of sets under operations.*
10. (t-9) 4.4(c) *Closure under oddsL.* This is a tough problem. **Start early.** You'll have to understand the definitions in parts (a) and (b) before you can do this part. **Some clarification on this closure problem:** Saying that INF is closed under *oddsL* is saying that for every language L in INF, the language *oddsL*(L) is also in INF. This is not the same as saying that for every L in INF, *oddsL*(L) is a subset of L. Here are two analogies (in the realm of numbers instead of languages) to what you are supposed to prove or disprove. Let INTS be the collection of all sets of integers and let POSINTS be the collection of all sets of positive integers. Define the function *negs* on sets of integers by *negs*(S) = { -n : n ∊ S }. INTS is closed under *negs*, because applying *negs* to any set of integers yields a set of integers. But POSINTS is not closed under *negs*, because if S is a set of positive integers, *negs*(S) is not a set of positive integers.

    Hint: If we are talking about strings over a fixed, finite alphabet, a Language (i.e. a set of strings) L is infinite if and only if there is no upper bound on the lengths of strings in L. And L is finite if and only if there is an upper bound on the lengths of strings in L. Think about these statements. Do you understand what they are saying? Do you believe them? Can you see how they apply to this problem?

11. 5.2 c,e,h *DFSMs to accept various languages*.
12. (t-3) 5.2(a) (No need for formal descriptions; just draw a transition diagram or a transition table).
13. (t-3) 5.2(b) (No need for formal descriptions; just draw a transition diagram or a transition table).
14. (t-9) Consider all possible 2-state Deterministic Finite State Machines M. How many different languages over alphabet Σ={a} can be L(M)for one of those machines? How do you get this number?
15. (t-6) *How many strings of length k?* The finite automaton on the next page accepts no strings of length zero, no strings of length one, and only two strings of length two (`01` and `10`). There is a fairly simple recurrence relation for the number N(k) of strings whose length is k that this automaton accepts. Discover and write down this recurrence relation. **Note**: the recurrence's solution does not have an easy-to-use closed form, so you will have to compute the first several values by hand. You do not have to compute N(k) for any k greater than 14. To help you determine whether you have the correct recurrence, I will tell you that N(8)=10 and N(9)=12. Show the "recursive" part of the recurrence relation.
Use your recurrence relation to answer this question: What are the values of N(13) and N(14)?
You are not required to explain how you get the recurrence relation.



Previous Hints, Questions, and Answers from Piazza:

PDA problems form HW2

For HW2, you do not have to use the "state diagram" notation for PDA's; high-level English descriptions are fine. Mainly you need CSSE220/230-level knowledge of what a stack is in order to do problems 3.5 and 3.6.

No transition in diagram. What does it mean?

Q: If we draw a transition diagram doe a DFSM and there is not transition out of state **q** on alphabet symbol **a**, what does this mean?
A: This is an abbreviation for "this transition would go to the dead state, which we are not showing here because it would clutter up the diagram."

Hint for HW2 problem 10 (4.4c)

If we are talking (and we are in this course!) about strings over a fixed, finite alphabet, a Language (i.e. a set of strings) L is infinite if and only if there is no upper bound on the lengths of strings in L,
And it is finite if and only if there is an upper bound on the lengths of strings in L.

Think about these statements. Do you understand what they are saying? Do you believe them? Can you see how they apply to this problem?

Question 14 on HW2

Q: Is there a place that talks about how many languages are accepted by a DFSM? I didn't see anything in the reading, and I'm not seeing the connection between the number of languages and the number of states.
A: No, there is nothing like that. This is intended as an "experimental" question. Just start drawing state diagrams for 2-state machines and keep going until you have all of the languages.
The number of languages is small enough so that this is not a daunting task. There are four possibilities for which of the two states are accepting states; For each of those, there are various possibilities for how they can be connected. One could use combinatorics to calculate how many different 2-state machines there are, But that turns out to be a considerably larger number than the number of different languages they accept, which is what this question asks for.
If there is a dead state, it has to be one of the two states. If we could have two states plus a dead state, it would cause there to be many more possible languages.
Followup question: So for each FSM, is there only one accepting language? Like do we use {a}* as the accepting language, or can we count all the subsets of that, like {a, aa}? I'm just making sure I understand what makes up an accepting language.

Answer: It is "accepted language", not "accepting language". It is the language L(M) accepted by the DGFSM M. It is unique. It contains all strings w such that if M is started in the start state and reads w, it ends up in an accepting state. The "all strings" part guarantees that there is only one language L(M).

Examples: If every state of M is an accepting state, then L(M) is sigma*
If none of the states of M is an accepting state, then L(M) is the empty set.
If some states of M are accepting states and some are not, then what L(M) is depends on the specific transitions in M.

Another followup: Last clarification, you can only have one dead state correct?
A: There is never a *need* to have more than one dead state, but no reason why you can't make a machine with more than one. If a 2state machine has more than one, one of them will be unreachable from the start state