CSE 311: Foundations of Computing (Fall, 2015)

Homework 7 Out: Fri, 20-Nov. Due: Friday, 4-Dec, **11: 59 pm** on **Gradescope**

Additional directions: You should write down carefully argued solutions to the following problems. Your first goal is to be complete and correct. A secondary goal is to keep your answers simple and succinct. The idea is that your solution should be easy to understand for someone who has just seen the problem for the first time. (Re-read your answers with this standard in mind!) You may use any results proved in lecture (without proof). Anything else must be argued rigorously. Make sure you indicate the specific steps of your proofs by induction.

1. Regular expressions (15 points)

For each of the following, construct regular expressions that match the given set of strings.

a) Binary strings where every occurrence of a 1 is followed by a 0.

b) The set of binary strings that do not contain 111

c) Binary strings that contain equal number of occurrences of 10 and 01. [Note that 101 contains a single 01 and a single 10 so it is in the language, but 1010 is not in the language because it contains two 10s and one 01.

2. CFGs (15 points)

For each of the following problems, construct a context free grammar (CFG) that generates the described language.

a) The set of binary strings with at least 3 copies of 101.

b) $\{0^m 1^{m+n} 2^n \colon\ m, n \geq 0\}$

c) Binary strings where the number of 0s is congruent to 2 modulo 5.

3. DFA Design (15 points)

For each of the following create a DFA that recognizes the given language.

a) Binary strings where the number of 0s is congruent to 2 modulo 5.

b) Consider the alphabet

$$\Sigma_2 = \left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

A string over Σ_2 gives two rows of 0s and 1s. Consider each row as a binary number and let L be the language

$$L = \{w \in \Sigma_2^* : \text{the bottom row of } w \text{ is larger than its top row}\}.$$

For example, $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} \in L$ but $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} \notin L$.

c) Binary strings where if we treat them as a binary number, that number is congruent to 2 modulo 5. For example, 00111 is in the language (because $7 \equiv 2 \bmod 5$) but 011 is not.

4. NFA Design (15 points)

For each of the following create an NFA that recognizes the given language.

a) The set of binary strings that contain 11 **or** do not contain 00

b) The set of binary strings that contain 11 **and** do not contain 00

c) The set of binary strings such that if they contain 11 then they do not contain 00.

5. FSM design (20 points)

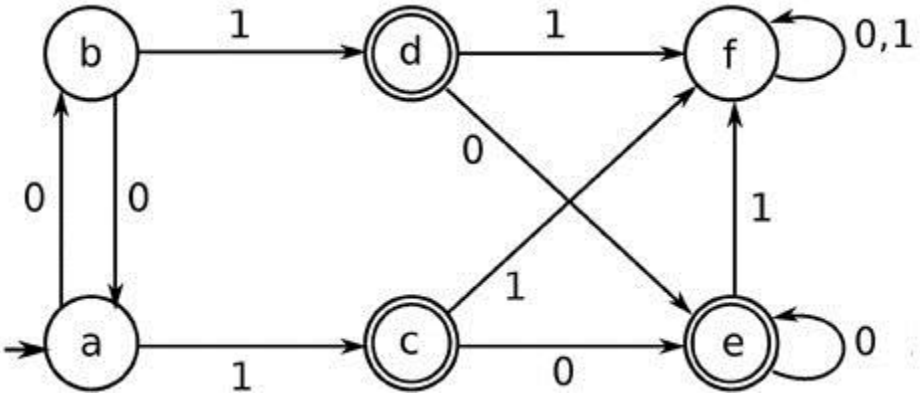
In the old days of video gaming, you had to go through a very specific set of actions in order to deal damage to an enemy boss. Your FSM should have inputs **S** (sword), **A** (arrow), **D** (dodge). The outputs are **OUCH** (boss damage), **OOF** (player damage), and **FAIL** (player death).

- If the actions are **D**, **S**, **D**, **A** in order, you should output **OUCH**.
- Any other sequence of actions should result in output **OOF**.
- After an **OUCH** or **OOF** output, the sequence of actions is reset.
- If three **OOF** outputs occur in a row without an **OUCH** in between, you should output **FAIL**. From the **FAIL** state, no actions matter. Nothing matters. You have failed.

You may notice there is no way to defeat the boss. Tough. Video games used to be harder.

6. State minimization (10 points)

Use the algorithm for minimization we discussed in lecture to minimize the following automaton. Be sure to write down every step of the algorithm and circle the groups at every step.



7. Extra credit: Boss mode

We can't leave the boss undefeated. It turns out that, since we didn't do the tutorial (in the old days, you read a manual!), we didn't know that there are two stances, and in each stance the actions **S**, **A**, **D** have different effects. The input **T** causes your character to change stances.

They published a strategy guide on reddit, but it's kind of confusing. Every non-empty sequence of inputs causes exactly one of three things to happen: Either your character dies, or the boss dies, or the sequence allows the boss to heal, resetting the battle. All we have are the following cryptic rules about sequences of inputs. Here, Φ and Ψ represent different sequences of inputs.

- $\Phi \vdash \Psi$ represents the two rules:
  - If Φ kills you, then Ψ causes the boss to heal
  - If Φ causes the boss to heal, then Ψ kills you.

- For any sequence of inputs Φ, we have $\mathbf{T}\Phi\mathbf{T} \vdash \Phi$

- If $\Phi \vdash \Psi$, then
  - $\mathbf{D}\Phi \vdash \mathbf{T}\Psi$
  - $\mathbf{A}\Phi \vdash \Psi^R$
  - $\mathbf{S}\Phi \vdash \Psi\Psi$

But after you invested all this time in reading, you're not interested in just killing the boss. You want to prove your dominance by beating the boss as fast as possible. Find the shortest possible sequence of moves that are guaranteed to kill the boss, and prove your answer correct.

8. Extra credit: Robot TA

After years of holding office hours, you've finally decided that maybe a robot could do just as well. Your robot's EmpathyEngine™ is capable of recognizing three possible student states: **whining**, **confusion**, and **asleep**. And your robot can take four actions: **console** ("Your answer is wrong, but I like your enthusiasm"), **destroy**, **offer-more-points**, and **lame-joke**.

As a first order of business, you design a relatively simple language to control the robot's reactions.

⟨Stmt⟩ = if ⟨EmotionalState⟩ then ⟨Stmt⟩ |
if ⟨EmotionalState⟩ then ⟨Stmt⟩ else ⟨Stmt⟩ |
⟨ActionSeq⟩

⟨ActionSeq⟩ = ⟨Action⟩ | ⟨ActionSeq⟩ ⟨Action⟩

⟨EmotionalState⟩ = **whining** | **confusion** | **asleep**

⟨Action⟩ = **console** | **destroy** | **offer-more-points** | **lame-joke**

After some initial experiments, you realize that sometimes the robot is performing the **destroy** action when it should be making a lame joke. (Fortunately, your robot skills are somewhat worse than your AI skills, and the RobotDeathChop™ is more like an awkward pat on the shoulder while the robot screams "death chop! death chop!") This is because your grammar is ambiguous. There are some snippets of code that can be parsed in different ways, and those different ways have different meanings.

(a) Show an example of a string in the language that has two different parse trees.

(b) Give a new grammar for the same language that is **unambiguous** in the sense that every string has a unique parse tree.