CSE 311: Foundations of Computing (Spring, 2015)

Homework7 Out:Fri,22-May. Due:Friday,29-May,beforeclass(1pm) 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. CFG design (12 points)

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

(a) The set of all binary strings of even length whose middle two characters are equal, e.g. 0100111111 or 0011001100.

(b) $\{0^m 1^n 0^{2m+n}\}$

(c) All binary strings that contain at least three 0's and at most two 1's.

(d) The alphabet contains the four characters), (,], [. Give a CFG for the language of all correctly parenthesized expressions where "(" matches ")" and "[" matches "]". For example,([]([[]))[(()())[([[]))]]) but not ([)]

2. DFA design (20 points)

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

(a) The set of all binary strings that end with a 0 and have odd length, or start with a 1 and have even length.

(b) The set of all binary strings that don't contain 100.

(c) The set of all binary strings that contain at least two 0's.

(d) The set of all binary strings that contain at most two 1's. Use different state labels from the ones you used in the previous part.

(e) Combine the machines from the previous two parts to produce a machine that recognizes the set of all binary strings that contain at least two 0's OR at most two 1's.

3. NFA design (15 points)

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

(a) The set of all binary strings that start with an even numbers of 0's **or** end with an odd number of $1's$.

(b) The set of all binary strings that start with an even number of 0's **and** end with an odd number of 1's.

4. FSM design (20 points)

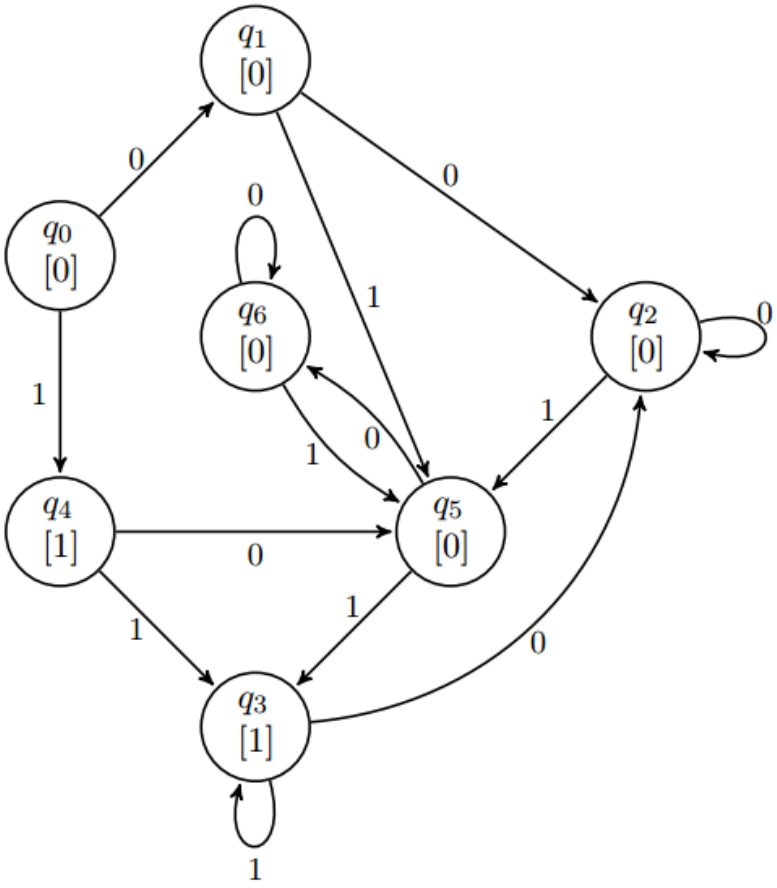
In the old days of video gaming, you had to go through a very specific set of actions in order to deal damage to an enemy boss. Your FSM should have inputs **S** (sword), **A** (arrow), **D** (dodge), **P** (pause). The outputs are **OUCH** (boss damage), **OOF** (player damage), and **FAIL** (player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

- If the actions are **D**, **A**, **D**, **S** in order, you should output **OUCH**.
- The preceding sequence can have pauses interspersed, but when the game is paused, no inputs are registered. (So after one pause, no other inputs matter until pause is pressed again.)
- Any other sequence of (non-pause) 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.

5. 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.



6. 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.

6. 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.

$$
\begin{aligned}
\langle \text{Stmt} \rangle &= \text{if } \langle \text{EmotionalState} \rangle \text{ then } \langle \text{Stmt} \rangle \mid \\
&\quad \text{if } \langle \text{EmotionalState} \rangle \text{ then } \langle \text{Stmt} \rangle \text{ else } \langle \text{Stmt} \rangle \mid \\
&\quad \langle \text{ActionSeq} \rangle \\
\langle \text{ActionSeq} \rangle &= \langle \text{Action} \rangle \mid \langle \text{ActionSeq} \rangle \, \langle \text{Action} \rangle \\
\langle \text{EmotionalState} \rangle &= \textbf{whining} \mid \textbf{confusion} \mid \textbf{asleep} \\
\langle \text{Action} \rangle &= \textbf{console} \mid \textbf{destroy} \mid \textbf{offer-more-points} \mid \textbf{lame-joke}
\end{aligned}
$$

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.