Week 7: Loony Automata Tune

Collaboration Policy: You should work on the problems yourself, before discussing with others, and with your cohorts are your cohort meeting. By the Assessed Cohort Meeting, you and all of your cohortmates, should be prepared to present and discuss solutions to all of the assigned problems (including the programming problems). In addition to discussing with your cohortmates, you may discuss the problems with anyone you want, and use any resources you want except for any materials from previous offerings of this course, which are not permitted.

Problem 1 *What if a cohort-mate is struggling?*

Probably all of you have experienced this at some point by now. You're in your assessed cohort-meeting, your cohort-mate is tasked with explaining a solution to some problem, and is really struggling. What would you suggest the other cohort members do in this situation? If you were the one struggling, what would you hope your cohort members would do? Keep in mind that the primary goal of your cohort is for you to help each other learn.

Problem 2 *Problem Re-attempt* (Optional)

If you got a 2.5 or below in any assessed cohort meeting, you're invited to prepare for a reattempt of that problem. Please inform your Cohort Coach (TA) which problem you would like to re-attempt so we can schedule when that can happen.

Problem 3 *Non-Deterministic Finite State Automata*

Give a non-deterministic finite state automaton which decides each of the languages described below, using no more than the number of states indicated. Give both a drawing and a description.

(a) $\{x \in \{0,1\}^* \mid x$ contains the substring 0101$\}$, using no more than five states.

(b) The language described by $0^*1^*0^*$, using no more than three states.

(c) The language described by $1^*(001^*)^*$, using no more than three states.

(d) The language described by $((0 \mid 00)^*(1|\varepsilon))^*$, using no more than one state.

Problem 4 *Kleene Star Construction*

In class, we showed how we could take a NFA for a language L, and construct an NFA for the language L^*. In this construction, in order to ensure the NFA returned 1 for the empty string, we created a brand new final state, and added an epsilon transition from the original start state to the new state. Why could we not simply make the start state a final state?

Problem 5 *Infinite State Automata*

We've been discussing finite state automata as a model of computing. Since we are talking about abstract models here, why should we restrict ourselves to a finite number of states?

Consider a new model of computing that we'll call *Infinite State Automata*. It behaves exactly the same way as finite state automata, except now the set of states Q is no longer required to be finite.

(a) Show that Infinite State Automata are *at least as powerful as* finite state automata. (This should be easy, but checks that you understand the definition of at least as powerful, and the machine definitions.)

(b) Show that Infinite State Automata are *more* powerful than finite state automata by showing how you could use it to implement regular expression binding, which you hopefully showed in Week 6 couldn't be done with finite state automata. For example, you should describe an Infinite State Automata that corresponds to the language `\b(a*)Z\1` which cannot be described by any FSA.

(c) ($\star$) Describe a language that *cannot* be decided by an Infinite State Automata (or argue why no such language exists).

Problem 6 *NFA Size*

Regular expressions are represented by strings of text. Consider that we have a regular expression that is n characters long.

Show that the language represented by this regular expression can be computed by a NFA with $O(n)$ states (that is, the function from the length of the a regular expression to the number of states needed for an NFA that decides the language represented by that regular expression is in $O(n)$ where n is the length of the input).

Problem 7 *Circuits vs. Automata*

For this problem we are going to look at the relationship between Boolean circuits and finite state automata. We showed in lecture that every function computable by a NAND circuit is also computable by some finite state automaton. We also showed that there are some functions we can compute using finite state automata that cannot be computed by any NAND circuit. Answer these additional questions regarding the relationship between finite state automata and circuits.

(a) Show that for any finite state automaton, the transition function δ can be computed by a NAND circuit.

(b) What features we could add to our NAND-CIRC straightline programming language to make it equivalent in computing power to finite state automata? Your extended NAND-CIRC language should be able to compute every function that can be computed by a finite state automaton, but be careful to not add features that would enable it to compute things that cannot be computed by a FSA. An excellent answer would include a proof showing that both of these properties are satisfied to establish equivalence.

Problem 8 *Operations on Languages*

In class, we showed that the Regular languages are closed under various operations. We showed, for example, that when regular languages L_1, L_2 are given as operands to Complement ($\overline{L_1}$), union ($L_1 \cup L_2$), and intersection ($L_1 \cap L_2$), the resulting language will also always be regular.

Show that the regular languages are closed under each of the operations below.

(a) NAND: a string x is in $NAND(L_1, L_2)$ if $\texttt{NAND}(x \in L_1, x \in L_2)$.

(b) NOR: a string x is in $NOR(L_1, L_2)$ if $\texttt{NOR}(x \in L_1, x \in L_2)$.

(c) XOR: a string x is in $XOR(L_1, L_2)$ if $\texttt{XOR}(x \in L_1, x \in L_2)$.

(d) Difference: a string x is in $SUBTRACT(L_1, L_2)$ if $x \in L_1$ and $x \notin L_2$.

Problem 9 *Fixing Brunelle's Mistakes*

In the *Equivalence of NFAs and DFAs* video, Professor Brunelle's method of converting an NFA to a DFA is not entirely correct. In particular, there is an error where he defines the DFA's transition function δ_D using the NFA's transition function δ as

$$\delta_D(s, \sigma) = \bigcup_{q \in s} \delta(q, \sigma).$$

The right-hand-side of this definition does not account for any ε transitions that might be present in the NFA.

(a) Give at least one NFA where converting using Brunelle's definition will result a DFA that computes a different language.

(b) Describe how we could correct this error (this may be informal).

(c) ($\star$) There's a least one other subtle problem in this construction, identify and correct that.

(One other subtlety with the definition is that sets of states in the DFA need to be mapped to labels for states in the NFA. The provided definition doesn't include the conversion between a set of DFA states and label for one NFA state. You can assume such conversions are done implicitly, that sets of DFA states are interpreted as you would expect as labels of NFA states.)

