Week 6

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at 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. In addition to discussing with your cohortmates, you may discuss the problems with any other current CS3102 students you want, and use any resources you want except for any materials from previous offerings of this course or complete solutions that might be available on the web, which are not permitted. **At the end of your assessed cohort meeting, your Cohort Coach will assign one of these problems as a writeup. You may not collaborate on this writeup with your cohort-mates, but you may use notes taken before or during your assessed cohort meeting.**

Problem 1: Finite State Automata

Define a finite state automaton which computes each of the following functions/languages. Give both a drawing and a description (i.e. a 5-tuple) or your automaton.

1. $\{x \in \{0,1\}^* \mid x$ as interpreted as a binary representation of a natural number is divisible by $8\}$ (note that the empty string is a binary representation of 0, which is divisible by 8).
2. $\texttt{AND} : \{0,1\}^* \to \{0,1\}$. In other words, the language $\{x \in \{0,1\}^* \mid \texttt{AND}(x) = 1\}$. (Like most definitions of mathematical conjunction and Python, $\texttt{AND}(\varepsilon) = 1$).)
3. $\texttt{NAND} : \{0,1\}^* \to \{0,1\}$. In other words, the language $\{x \in \{0,1\}^* \mid \texttt{NAND}(x) = 1\}$.
4. $\{x \in \{0,1\}^* \mid x$ starts with a 1 and has even length or else starts with a 0 and has odd length$\}$.
5. $\{x \in \{0,1\}^* \mid x$ as interpreted as a binary representation of a natural number minus 3 is $0\}$

Problem 2: Extending Straightline

Suggest a modification we can make to the NAND-Circ Straightline programming language computing model that would cause it to be equivalent to the finite state automata computing model. Provide informal justification of why these two models would be equivalent.

Problem 3: Infinite Languages

Show that the language computed by a finite state automaton is infinite if and only if there is some string accepted by the automaton with more characters than the automaton has states. That is, show that for finite state automaton $M = (Q, \Sigma, \delta, q_0, F)$ the size of the language computed by M is infinite if and only if there is some $s \in \Sigma^*$ such that $|s| > |Q|$.

Problem 4: Regular Expressions

For each of the following languages, give a regular expression that describes the language or explain why no such regular expression exists.

1. $\{x \in \{0,1\}^* \mid x$ as interpreted as a binary representation of a natural number is divisible by $8\}$ (note that the empty string is a binary representation of 0, which is divisible by 8).
2. $\{x \in \{0,1\}^* \mid x$ does not contain the substring $011\}$
3. $\{x \in \{0,1\}^* \mid$ every odd position of x is a $1\}$ (we will say the first bit is indexed by 0)

Problem 5: Kleene

One of our components for regular expressions is the *Kleene star*. Describe precisely the languages that can be defined using regular expressions without a Kleene star. Provide an informal justification of your answer.

Problem 6: Sugary Regexes

In lecture, our definition of regular expressions could only use the empty string (i.e. "" or ε), string literals (e.g. a or b), concatenation, alternation, or Kleene star. In many practical uses of regular expressions, additional components and operations are supported.

In the lecture, we argued that most of these extensions do not change the power of regular expressions (the set of languages they can describe), but there are some exceptions. For each of the following operations allowed in regular expressions in practice, show either that it is merely syntactic sugar (that is, that you can convert any regular expression that uses those components into one using only those from class) or that it does expand the power of regular expressions (that is, with this extension, it is possible to define some language that cannot be defined using a standard regular expression). (All of the extensions listed here are supported in some form by Python's `re` library.)

(a) **Question Mark**: If R is a regular expression, then the regular expression R`?` will match either the empty string or a string which matches R. For example, in Python, `ab?` matches only the strings `a` and `ab`.

(b) **Plus**: If R is a regular expression, then the regular expression R+ will match any string composed of 1 or more substrings which match R. For example, in Python, `(a|b)+` matches the strings `a`, `b`, `ab`, `ba`, etc., but not ε.

(c) **Character Class**: A character class gives a set of characters, and matches any one character from that set. For example, in Python, the character class `[abc]` will match only strings `a`, `b`, and `c`.

(d) **Bindings**: The subexpression `\P`R binds a variable to string matching R, and then `\1` matches that string. For example, in Python, `\P(a*)Z\1` matches `aaaZaaa` and `aaaaZaaaa`, but not `aaaaZaaa`. (Note: this simplifies the actual syntax in Python. The actual syntax for the example would be `(?P<v1>(a*))Z(?P=v1)`.)

(e) **Count**: If R is a regular expression, and n is a natural number, then $R\{n\}$ will match any string composed of exactly n substrings which match R. For example, in Python, `(0|1){9}` will match any binary string of length 9.

(f) **Count Range**: If R is a regular expression, and m, n are natural numbers with $m > n$, then $R\{m, n\}$ will match any string entirely composed of between n and m (inclusive) substrings which match R. For example, in Python, `b{2,4}` will match `bb`, `bbb`, and `bbbb`.

Problem 7: Endless Paperwork

Much to my frustration and embarrassment, I learned the hard way that Google allows such a thing as an unsubmittable form. I had mistakenly believed that Google would somehow warn me if a form was not submittable. After my embarrassment of asking 300 people to do the impossible, I sought to determine why Google would do this to me.

In my investigation, I found that Google Forms are (roughly, but deeply) equivalent to finite state automata!

We have created one such form here: *A Google Form Automaton*(https://forms.gle/8o1JEKbV9kaa7cp4A). States in this automaton are represented as sections in the form. Transitions between sections/states depend on the response selected for that section. "DONE" indicates that the last input has been given. Once the last input has been given, you will be informed about whether or not the machine accepted or rejected your input. To be convinced that Google forms can indeed implement arbitrary DFAs, first find an input sequence that is accepted by the form, and then determine the language computed by the form.

(a) What would it involve for Google to provide a tool that warned form creators if they have created an unsubmittable form? (A good answer would first formalize this as a property of the DFA corresponding to the Google form, and then explain what it would involve to test the DFA for the unsubmittable form property. You should think about the distinction between a form that is unsubmittable (no way to ever reach the submission) and a form where there are inputs of unbounded length that never reach a submission).

(b) Why doesn't Google forms implement such a tool? (This is intentionally open-ended. Your answer may be technical, logistical, economic, etc.)