PS6: Automata and Regular Expressions

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. 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. Sharing and subsequently submitting any text, code, images, figures, etc. constitutes plagiarism, so make sure all submitted materials are created exclusively by members of your cohort.

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: 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 3: 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`.

