Math 560 Homework 3 Fall 2021

Due: Thursday, September 16

1. **Regular Expressions**

   Give regular expressions that describe each of the following languages. You may assume that the alphabet in each case is $\Sigma = \{0, 1\}$.

   (a) $\{w \,|\, \text{the length of } w \text{ is odd}\}$

   (b) $\{w \,|\, w \text{ has an odd number of 0s}\}$

   (c) $\{w \,|\, w \text{ contains at least two 0s or exactly two 1s}\}$

   (d) $\{w \,|\, w \text{ does not contain the substring 11}\}$

   (e) $\{w \,|\, w \text{ does not end in a double letter, i.e., does not end in 00 or 11}\}$

   (f) $\{w \,|\, \text{every odd position of } w \text{ is a 1}\}$

2. **Lexers/Tokenizers**

   A *lexer* (also known as a *tokenizer*) is a program that takes a sequence of characters and splits it up into a sequence of words, or "tokens." Compilers typically do this as a prepass before parsing programs. For example, "`if (count == 42) ++n;`" might divide into `if` , `(` , `count` , `==` , `42` , `)` , `++` , `n` , `;` .

   Regular expressions are a convenient way to describe tokens (e.g., C integer constants) because they are unambiguous and compact. Further, they are easy for a computer to understand: *lexer generators* such as `lex` or `flex` can turn regular expressions into program code for dividing characters into tokens.

   In general, there may be many ways to divide the input up into tokens. For example, we might see the input `ifoundit = 1` as starting with a single token `ifoundit` (a variable name), or as starting with the keyword `if` followed immediately by the variable `oundit`. Most commonly, lexers are implemented to be *greedy*: given a choice, they prefer to produce the longest possible tokens. (Hence, `ifoundit` is preferred over `if` as the first token.)

   Lexers commonly skip over whitespace and comments. **A "traditional" comment in C starts with the characters `/*` and runs until the next occurrence of `*/`.** Nested comments are forbidden.

   Your task is to construct a regular expression for traditional C comments, one suitable for use in a lexer generator.

   Before you start, there is some notation you may find useful:

   - To indicate the union of every character in the alphabet, we can write Σ. For example, if the alphabet is $\{a, b, c, d, e\}$, then

     $$\Sigma = (a|b|c|d|e),$$

     when written in a regular expression.

- We use the symbol $\neg$ to indicate not. For example, if the alphabet is $\{a, b, c, d, e\}$, then

$$\neg\{a\} = (b|c|d|e), \quad \text{and} \quad \neg\{b, d, e\} = (a|c).$$

(a) When they see this problem for the first time, people often immediately suggest

$$/ * \ (\Sigma \,|\, \backslash n\,)^* \ * /$$

Explain why this regular expression would not make a lexer skip comments correctly. (Big hint: greedy).

(b) Once the problem with the previous expression is noted, most folks decide that comments should contain only characters that are not stars, plus stars that are not immediately followed by a slash. This leads to the following regular expression:

$$/ * \ (\neg\{*\} \,|\, * \neg\{/\}\,)^* \ * /$$

Find a legal 5-character C comment that this regular expression fails to match, and a 7-character ill-formed (non-valid-comment) string that the regular expression erroneously matches.

(c) Draw an NFA that accepts all and only valid traditional C comments.

(d) Provide a correct regular expression that describes all and only valid traditional C comments, by converting your NFA into a regular expression. Show all of your work.

NOTE: be sure it's completely clear where you are using the character $*$ and when you are using the regular expression operator *.

3. **Powers of 2**

Describe an algorithm for a Turing Machine that decides the language consisting of all strings of 0s whose length is a power of 2:

$$\{0^{2^n} \,|\, n \geq 0\}.$$

You may assume that the input alphabet in this case is $\Sigma = \{0\}$.

4. **Ones and Zeros**

Describe an algorithm for a Turing Machine that can decide the language consisting of all strings of n 1s followed by n 0s:

$$\{1^n 0^n \,|\, n \in \mathbb{N}\}.$$

You may assume that the input alphabet is $\Sigma = \{0, 1\}$.