CSE 311: Foundations of Computing I

Homework 6 (due Wednesday, November 12)

Directions: *Write up carefully argued solutions to the following problems. The first task is to be complete and correct. The more subtle task is to keep it simple and succinct. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use any results proven in lecture without proof. Anything else must be argued rigorously. Unless otherwise specified, all answers are expected to be given in closed form.*

0. All Your Base (Case) are Belong to Us (20 points)

Define f_n and g_n as follows for $n \in \mathbb{N}$:

$$f_0 = 1 \qquad g_0 = 1$$
$$f_1 = 5 \qquad g_1 = 5$$
$$f_2 = 10 \qquad g_2 = 10$$
$$g_3 = 0$$
$$g_4 = -40$$
$$f_n = 2f_{n-1} - 4f_{n-2} \text{ for } n \geq 3 \qquad g_n = 2g_{n-1} - 3g_{n-2} - 2g_{n-3} + 4g_{n-4} \text{ for } n \geq 5$$

Prove that $f_n = g_n$ for all $n \in \mathbb{N}$.

1. Runtime. . . Better Go Catch It! (20 points)

Let $c > 0$ be an integer. The following recursive definition describes the running time of a recursive algorithm.

$$T(0) = 0$$
$$T(n) \leq c \qquad \text{for all } n \leq 20$$
$$T(n) = T\left(\left\lfloor \frac{3n}{4} \right\rfloor\right) + T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + cn \qquad \text{for all } n > 20$$

Prove by strong induction that $T(n) \leq 20cn$ for all $n \geq 0$.
Hint: The only fact about $\lfloor\ \rfloor$ that you will need is that when $x \geq 0$, $\lfloor x \rfloor$ is an integer, and $0 \leq \lfloor x \rfloor \leq x$.

2. Awwww! (20 points)

Recall that reversal (on strings) is defined by:

$$\varepsilon^{\mathcal{R}} = \varepsilon$$
$$(wa)^{\mathcal{R}} = aw^{\mathcal{R}} \qquad \text{for } w \in \Sigma^* \text{ and } a \in \Sigma$$

Prove that $(xy)^{\mathcal{R}} = y^{\mathcal{R}}x^{\mathcal{R}}$ for all $x, y \in \Sigma^*$.

3. Strings are Strings (20 points)

For this question, please submit your answers using the following website:

https://grinch.cs.washington.edu/regex/

For each of the following, construct regular expressions that match the given set of strings:

(a) [5 Points] The set of all binary strings that end with 0 and have even length, or start with 1 and have odd length.

(b) [5 Points] The set of all binary strings that have a 1 in every odd-numbered position counting from the start of the string (where the start of the string is position 0).

(c) [5 Points] The set of all binary strings that contain at least two 1's and at most two 0's.

(d) [5 Points] The set of all binary strings that don't contain 001.

4. Proving BST Insertion Works! (20 points)

Consider the following recursive definition of a tree:

- `Nil` is a tree
- If L and R are trees, then $\texttt{Tree}(x, L, R)$ is a tree.

Consider the recursive function `insert` (which takes a tree and a number as arguments):

$$\texttt{insert}(\texttt{Nil}, y) = \texttt{Tree}(y, \texttt{Nil}, \texttt{Nil})$$

$$\texttt{insert}(\texttt{Tree}(x, L, R), y) = \begin{cases} \texttt{Tree}(x, L, R) & \text{if } y = x \\ \texttt{Tree}(x, \texttt{insert}(L, y), R) & \text{if } y < x \\ \texttt{Tree}(x, L, \texttt{insert}(R, y)) & \text{if } y > x \end{cases}$$

Consider the function `sorted` (which tests if a tree is in BST search order):

$$\begin{aligned}
&\texttt{sorted}(\texttt{Nil}) && = \text{True} \\
&\texttt{sorted}(\texttt{Tree}(x, \texttt{Nil}, \texttt{Nil})) && = \text{True} \\
&\texttt{sorted}(\texttt{Tree}(x, \texttt{Nil}, \texttt{Tree}(y, L_1, R_1))) && = x < y \text{ and } \texttt{sorted}(\texttt{Tree}(y, L_1, R_1)) \\
&\texttt{sorted}(\texttt{Tree}(x, \texttt{Tree}(y, L_0, R_0), R)) && = y < x \text{ and } \texttt{sorted}(\texttt{Tree}(y, L_0, R_0)) \text{ and } \texttt{sorted}(\texttt{Tree}(x, \texttt{Nil}, R))
\end{aligned}$$

Prove that $\texttt{sorted}(T) \rightarrow \texttt{sorted}(\texttt{insert}(T, y))$ for all trees T and $y \in \Sigma^*$.

5. EXTRA CREDIT: Lock and Key (-NoValue- points)

In Adam's office, there is a peculiar safe with some documents that he would really like to recover. The combination is a subset of $[A - Z]^* \setminus \{\varepsilon\}$.

Each combination can either *open*, *jam*, or *still* the lock. If a combination *jam*s the lock, then it will no longer ever work. If a combination *still*s the lock, then it is as if no combination were entered at all.

We use greek letters like α and β to denote arbitrary combinations. We define the operation $\alpha \hookrightarrow \beta$ as follows:

(a) For any combination α, QαQ $\hookrightarrow \alpha$

(b) If $\alpha \hookrightarrow \beta$, then $\mathsf{L}\alpha \hookrightarrow \mathsf{Q}\beta$

(c) If $\alpha \hookrightarrow \beta$, then $\mathsf{V}\alpha \hookrightarrow \beta^{\mathcal{R}}$

(d) If $\alpha \hookrightarrow \beta$, then $\mathsf{R}\alpha \hookrightarrow \beta\beta$

(e) If $\alpha \hookrightarrow \beta$, then ($\alpha$ *jam*s the lock $\rightarrow \beta$ *still*s it) and (α *still*s the lock $\rightarrow \beta$ *jam*s it)

Find the shortest possible combination that is guaranteed to open the lock, and prove your answer correct.