1. Structural Induction

(a) Consider the following recursive definition of strings.

\textbf{Basis Step:} "" is a string

\textbf{Recursive Step:} If \(X\) is a string and \(c\) is a character then append\((c, X)\) is a string.

Recall the following recursive definition of the function \(\text{len}\):

\[
\begin{align*}
\text{len}\("\) &= 0 \\
\text{len}(\text{append}(c, X)) &= 1 + \text{len}(X)
\end{align*}
\]

Now, consider the following recursive definition:

\[
\begin{align*}
\text{double}\("\) &= "" \\
\text{double}(\text{append}(c, X)) &= \text{append}(c, \text{append}(c, \text{double}(X))).
\end{align*}
\]

Prove that for any string \(X\), \(\text{len}(\text{double}(X)) = 2\text{len}(X)\).

\textbf{Solution:}

For a string \(X\), let \(P(X)\) be "\(\text{len}(\text{double}(X)) = 2\text{len}(X)\). We prove \(P(X)\) for all strings \(X\) by structural induction.

\textbf{Base Case.} We show \(P(\"")\) holds. By definition \(\text{len}(\text{double}(\""))) = \text{len}(\"") = 0\). On the other hand, \(2\text{len}(\"") = 0\) as desired.

\textbf{Induction Hypothesis.} Suppose \(P(X)\) holds for some arbitrary string \(X\).

\textbf{Induction Step.} We show that \(P(\text{append}(c, X))\) holds for any character \(c\).

\[
\begin{align*}
\text{len}(\text{double}(\text{append}(c, X))) &= \text{len}(\text{append}(c, \text{append}(c, \text{double}(X)))) \\
&= 1 + \text{len}(\text{append}(c, \text{double}(X))) \\
&= 1 + 1 + \text{len}(\text{double}(X)) \\
&= 2 + 2\text{len}(X) \\
&= 2(1 + \text{len}(X)) \\
&= 2(\text{len}(\text{append}(c, X))) \\
&= 2(1 + \text{len}(X)) \\
&= 2(\text{len}(\text{append}(c, X)))
\end{align*}
\]

This proves \(P(\text{append}(c, X))\).

Thus, \(P(X)\) holds for all strings \(X\) by structural induction.

(b) Consider the following definition of a (binary) Tree:

\textbf{Basis Step:} \(\bullet\) is a Tree.

\textbf{Recursive Step:} If \(L\) is a Tree and \(R\) is a Tree then \(\text{Tree}(\bullet, L, R)\) is a Tree.

The function \(\text{leaves}\) returns the number of leaves of a Tree. It is defined as follows:

\[
\begin{align*}
\text{leaves}(\bullet) &= 1 \\
\text{leaves}(\text{Tree}(\bullet, L, R)) &= \text{leaves}(L) + \text{leaves}(R)
\end{align*}
\]
Also, recall the definition of size on trees:
\[
\begin{align*}
\text{size}(\bullet) &= 1 \\
\text{size}(\text{Tree}(\bullet, L, R)) &= 1 + \text{size}(L) + \text{size}(R)
\end{align*}
\]

Prove that \(\text{leaves}(T) \geq \text{size}(T)/2\) for all Trees \(T\).

**Solution:**
In this problem, we define a strengthened predicate. For a tree \(T\), let \(P\) be \(\text{leaves}(T) \geq \text{size}(T)/2 + 1/2\). We prove \(P\) for all trees \(T\) by structural induction.

**Base Case.** We show that \(P(\cdot)\) holds. By definition of \(\text{leaves}(\cdot)\), \(\text{leaves}(\bullet) = 1\) and \(\text{size}(\bullet) = 1\). So, \(\text{leaves}(\bullet) = 1 \geq 1/2 + 1/2 = \text{size}(\bullet)/2 + 1/2\).

**Induction Hypothesis:** Suppose \(P(L)\) and \(P(R)\) hold for some arbitrary trees \(L\) and \(R\).

**Induction Step:** We prove that \(P(\text{Tree}(\bullet, L, R))\) holds.

\[
\text{leaves}(\text{Tree}(\bullet, L, R)) = \text{leaves}(L) + \text{leaves}(R) \quad \text{[By Definition of leaves]}
\geq \frac{1}{2} \times \left(\frac{\text{size}(L)}{2} + 1/2\right) + \frac{1}{2} \times \left(\frac{\text{size}(R)}{2} + 1/2\right) \quad \text{[By IH]}
= \frac{1}{2} \times \left(\text{size}(L) + \text{size}(R) + 1\right) + 1/2
= \frac{1}{2} \times \text{size}(\text{Tree}(\bullet, L, R)) + 1/2 \quad \text{[By Definition of size]}
\]

This proves \(P(\text{Tree}(\bullet, L, R))\).

Thus, the \(P(T)\) holds for all trees \(T\).

2. Regular Expressions
   (a) Write a regular expression that matches base 10 non-negative numbers (e.g., there should be no leading zeros).

   **Solution:**
   \[
   0 \cup ((1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^*)
   \]

   (b) Write a regular expression that matches all non-negative base-3 numbers that are divisible by 3.

   **Solution:**
   \[
   0 \cup ((1 \cup 2)(0 \cup 1 \cup 2)^*0)
   \]

   (c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

   **Solution:**
   \[
   (01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)111(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)
   \]
   (If you don’t want the substring 000, the only way you can produce 0s is if there are only one or two 0s in a row, and they are immediately followed by a 1 or the end of the string.)