CSE 311: Foundations of Computing I

Section : Structural Induction and Regular Expressions Solutions

1. Structural Induction

(a) Consider the following recursive definition of strings.

Basis Step: "" is a string

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

Recall the following recursive definition of the function len:

$$\begin{aligned} \text{len}(\texttt{""}) &= 0 \\ \text{len}(\text{append}(c, X)) &= 1 + \text{len}(X) \end{aligned}$$

Now, consider the following recursive definition:

$$\begin{aligned} \text{double}(\texttt{""}) &= \texttt{""} \\ \text{double}(\text{append}(c, X)) &= \text{append}(c, \text{append}(c, \text{double}(X))). \end{aligned}$$

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

Solution:

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

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

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

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

$$\begin{aligned} \text{len}(\text{double}(\text{append}(c, X))) &= \text{len}(\text{append}(c, \text{append}(c, \text{double}(X)))) && \text{[By Definition of double]} \\ &= 1 + \text{len}(\text{append}(c, \text{double}(X))) && \text{[By Definition of len]} \\ &= 1 + 1 + \text{len}(\text{double}(X)) && \text{[By Definition of len]} \\ &= 2 + 2\text{len}(X) && \text{[By IH]} \\ &= 2(1 + \text{len}(X)) && \text{[Algebra]} \\ &= 2(\text{len}(\text{append}(c, X))) && \text{[By Definition of len]} \end{aligned}$$

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

Thus, $\text{P}(X)$ holds for all strings X by structural induction.

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

Basis Step: $\bullet$ is a **Tree**.

Recursive Step: If L is a **Tree** and R is a **Tree** then $\texttt{Tree}(\bullet, L, R)$ is a **Tree**.

The function leaves returns the number of leaves of a **Tree**. It is defined as follows:

$$\begin{aligned} \text{leaves}(\bullet) &= 1 \\ \text{leaves}(\texttt{Tree}(\bullet, L, R)) &= \text{leaves}(L) + \text{leaves}(R) \end{aligned}$$

Also, recall the definition of size on trees:

$$\begin{aligned}
&\text{size}(\bullet) &&= 1 \\
&\text{size}(\texttt{Tree}(\bullet, L, R)) &&= 1 + \text{size}(L) + \text{size}(R)
\end{aligned}$$

Prove that leaves$(T) \geq$ size$(T)/2$ for all Trees T.

Solution:

In this problem, we define a strengthened predicate. For a tree T, let P be leaves$(T) \geq$ 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 leaves$(.)$, leaves$(\bullet) = 1$ and size$(\bullet) = 1$. So, leaves$(\bullet) = 1 \geq 1/2 + 1/2 =$ 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$(\texttt{Tree}(\bullet, L, R))$ holds.

$$\begin{aligned}
\text{leaves}(\texttt{Tree}(\bullet, L, R)) &= \text{leaves}(L) + \text{leaves}(R) && \text{[By Definition of leaves]} \\
&\geq (\text{size}(L)/2 + 1/2) + (\text{size}(R)/2 + 1/2) && \text{[By IH]} \\
&= (\text{size}(L) + \text{size}(R) + 1)/2 + 1/2 \\
&= \text{size}(\texttt{Tree}(\bullet, L, R))/2 + 1/2 && \text{[By Definition of size]}
\end{aligned}$$

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

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 zeroes).

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.)