# CSE 311: Foundations of Computing I

## Section 3: Structural Induction and Regular Expressions

## 0. Structural Induction

(a) Recall the recursive definition of a list:

$$\textbf{List} = [] \mid \text{Integer} :: \textbf{List}$$

And the definition of “len" on lists:

$$\begin{aligned} \text{len}([]) &= 0 \\ \text{len}(x :: L) &= 1 + \text{len}(L) \end{aligned}$$

Consider the following recursive definition:

$$\begin{aligned} \text{stutter}([]) &= [] \\ \text{stutter}(x :: L) &= x :: x :: \text{stutter}(L) \end{aligned}$$

Prove that $\text{len}(\text{stutter}(L)) = 2\text{len}(L)$ for all Lists $L$.

(b) Consider the recursive definition of a tree:

$$\textbf{Tree} = \texttt{Nil} \mid \texttt{Tree}(\text{Integer}, \textbf{Tree}, \textbf{Tree})$$

And the definition of “size" on trees:

$$\begin{aligned} \text{size}(\texttt{Nil}) &= 0 \\ \text{size}(\texttt{Tree}(x, L, R)) &= 1 + \text{size}(L) + \text{size}(R) \end{aligned}$$

And the definition of “height" on trees:

$$\begin{aligned} \text{height}(\texttt{Nil}) &= 0 \\ \text{height}(\texttt{Tree}(x, L, R)) &= 1 + \max(\text{height}(L), \text{height}(R)) \end{aligned}$$

Prove that $\text{size}(T) \leq 2^{\text{height}(T)+1} - 1$ for all Trees $T$.

(c) In this problem, we will use the same definitions for Tree defined above. Now, consider the definition of “mirror" on trees:

$$\begin{aligned} \text{mirror}(\texttt{Nil}) &= \texttt{Nil} \\ \text{mirror}(\texttt{Tree}(x, L, R)) &= \texttt{Tree}(x, \text{mirror}(R), \text{mirror}(L)) \end{aligned}$$

Prove that $\text{size}(T) = \text{size}(\text{mirror}(T))$ for all Trees $T$ by structural induction.

## 1. Meta-mathematical

Consider the following, simplified, recursive definition of an arithmetic expression:

$$\textbf{Expr} = \text{Natural} \mid \texttt{VarName}(\text{String}) \mid \texttt{Sum}(\textbf{Expr}, \textbf{Expr}) \mid \texttt{Prod}(\textbf{Expr}, \textbf{Expr})$$

And the definition of "eval" on expressions:

$$\begin{aligned}
\text{eval}(x) &= x \\
\text{eval}(\texttt{VarName}(s)) &= \text{eval}(\text{lookup}(s)) \\
\text{eval}(\texttt{Sum}(L, R)) &= \text{eval}(L) + \text{eval}(R) \\
\text{eval}(\texttt{Prod}(L, R)) &= \text{eval}(L) \times \text{eval}(R)
\end{aligned}$$

Note that "lookup" is a function that returns an Expr corresponding to the given string (which represents a variable name). You may assume "lookup" will always return an Expr – that is, we assume all variables are defined. For simplicity, we omit the definition of this function.

Now, consider the definition of "replace" on expressions:

$$\begin{aligned}
\text{replace}(t, r, x) &= x \\
\text{replace}(t, r, \texttt{VarName}(s)) &= \texttt{if } s = t \texttt{ then } r \texttt{ else VarName} s \\
\text{replace}(t, r, \texttt{Sum}(L, R)) &= \texttt{Sum}(\text{replace}(t, r, L), \text{replace}(t, r, R)) \\
\text{replace}(t, r, \texttt{Prod}(L, R)) &= \texttt{Prod}(\text{replace}(t, r, L), \text{replace}(t, r, R))
\end{aligned}$$

Let $a$ be an arbitrary string. Suppose $\text{eval}(\text{lookup}(a)) \geq 0$. Let $F = \texttt{Sum}(\texttt{VarName}(a), 1)$.

(a) Prove that $\text{eval}(\texttt{VarName}(a)) \leq \text{eval}(F)$.

(b) Prove that for any arbitrary Expr $E$ that $\text{eval}(E) \leq \text{eval}(\text{replace}(a, F, E))$.

## 2. Regular Expressions

(a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

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

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