CSE 311: Foundations of Computing I

Section 3: Structural Induction and Regular Expressions Solutions

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.

Solution:

We go by structural induction. Let L be a list.

Case $L = []$. Note that $\text{len}(\text{stutter}([])) = \text{len}([]) = 0 = 2\text{len}([])$.

Case $L = x :: L'$. Suppose that $\text{len}(\text{stutter}(L')) = 2\text{len}(L')$ for some list L'. Note that:

$$\begin{aligned} \text{len}(\text{stutter}(x :: L')) &= \text{len}(x :: x :: \text{stutter}(L')) && \text{[Definition of stutter]} \\ &= 1 + \text{len}(x :: \text{stutter}(L')) && \text{[Definition of len]} \\ &= 1 + 1 + \text{len}(\text{stutter}(L')) && \text{[Definition of len]} \\ &= 2 + 2\text{len}(L') && \text{[By IH]} \\ &= 2(1 + \text{len}(L')) && \text{[Distributivity]} \\ &= 2(\text{len}(x :: L')) && \text{[Definition of len]} \end{aligned}$$

Thus, the claim is true for all Lists by structural induction.

(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.

Solution:

We go by structural induction. Let T be a tree.

Case $T = \texttt{Nil}$**.** Note that $\text{size}(\texttt{Nil}) = 0 \leq 1 = 2^{0+1} - 1 = 2^{\text{height}(\texttt{Nil})+1} - 1$.

Case $T = \texttt{Tree}(x, L, R)$**.** Suppose that $\text{size}(L) \leq 2^{\text{height}(L)+1} - 1$ and $\text{size}(R) \leq 2^{\text{height}(R)+1} - 1$ for some trees L and R.

Note that:

$$\begin{aligned}
\text{size}(\texttt{Tree}(x, L, R)) &= 1 + \text{size}(L) + \text{size}(R) && \text{[Definition of size]} \\
&\leq 1 + 2^{\text{height}(L)+1} - 1 + 2^{\text{height}(R)+1} - 1 && \text{[By IH]} \\
&\leq 1 + 2^{\max(\text{height}(L),\text{height}(R))+1} - 1 \\
&\quad + 2^{\max(\text{height}(L),\text{height}(R))+1} - 1 && \text{[By max]} \\
&\leq 2\left(2^{\max(\text{height}(L),\text{height}(R)+1}\right) - 1 && \text{[Distributivity]} \\
&\leq 2\left(2^{\text{height}(\texttt{Tree}(x,L,R))}\right) - 1 && \text{[Definition of height]} \\
&\leq 2^{\text{height}(\texttt{Tree}(x,L,R))+1} - 1 && \text{[Properties of exponents]}
\end{aligned}$$

Thus, the claim is true for all Trees by structural induction.

(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.

Solution:

Case $T = \texttt{Nil}$**.** Note that $\text{size}(\texttt{Nil}) = 0 = \text{size}(\text{mirror}(\texttt{Nil}))$.

Case $T = \texttt{Tree}(x, L, R)$**.** Suppose that $\text{size}(L) = \text{size}(\text{mirror}(L))$ and $\text{size}(R) = \text{size}(\text{mirror}(R))$ for some trees L and R. Note that:

$$\begin{aligned}
\text{size}(\texttt{Tree}(x, L, R)) &= 1 + \text{size}(L) + \text{size}(R) && \text{[Definition of size]} \\
&= 1 + \text{size}(\text{mirror}(L)) + \text{size}(\text{mirror}(R)) && \text{[By IH]} \\
&= 1 + \text{size}(\text{mirror}(R)) + \text{size}(\text{mirror}(L)) && \text{[Commutivity]} \\
&= \text{size}(\texttt{Tree}(x, \text{mirror}(R), \text{mirror}(L))) && \text{[Definition of size]} \\
&= \text{size}(\text{mirror}(\texttt{Tree}(x, L, R))) && \text{[Definition of mirror]}
\end{aligned}$$

Thus, the claim is true for all Trees 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 $\text{eval}(\texttt{VarName}(a)) \leq \text{eval}(F)$.

Solution:

We wish to show that $\text{eval}(\texttt{VarName}(a)) \leq \text{eval}(F)$. Note that:

$$\begin{aligned}
\text{eval}(\texttt{VarName}(a)) &\leq \text{eval}(\texttt{VarName}(a)) + 1 && \text{[Properties of inequalities]} \\
&= \text{eval}(\texttt{VarName}(a)) + \text{eval}(1) && \text{[Definition of eval]} \\
&= \text{eval}(\texttt{Sum}(\texttt{VarName}(a), 1)) && \text{[Definition of eval]} \\
&= \text{eval}(F) && \text{[Variable substitution]}
\end{aligned}$$

So our claim is proven.

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

Solution:

Let E be an arbitrary Expr. We go by structural induction on E.

Case $E = x$. Note that $\text{eval}(x) \leq x \leq \text{eval}(x) \leq \text{eval}(\text{replace}(a, F, x))$.

Case $E = \texttt{VarName}(s)$. We go by cases.

Consider $s = a$. In this case, note that

$$\begin{aligned}
\text{eval}(\texttt{VarName}(a)) &\leq \text{eval}(F) && \text{[By part a]} \\
&= \text{eval}(\texttt{if } a = a \texttt{ then } F \texttt{ else VarName}(s)) && \text{[Semantics of if statements]} \\
&= \text{eval}(\text{replace}(a, F, \texttt{VarName}(s))) && \text{[Definition of replace]}
\end{aligned}$$

Consider $s \neq a$. In this case, note that

$$\begin{aligned}
\text{eval}(\texttt{VarName}(a)) &\leq \text{eval}(\texttt{if } s = a \texttt{ then } F \texttt{ else VarName}(s)) && \text{[Semantics of if statements]} \\
&= \text{eval}(\text{replace}(a, F, \texttt{VarName}(s))) && \text{[Definition of replace]}
\end{aligned}$$

Case $E = \texttt{Sum}(L, R)$. Suppose $\text{eval}(L) \leq \text{eval}(\text{replace}(a, F, L))$ and suppose $\text{eval}(R) \leq \text{eval}(\text{replace}(a, F, R))$ for some expressions L and R. Note that:

$$\begin{aligned}
\text{eval}(\texttt{Sum}(L, R)) &= \text{eval}(L) + \text{eval}(R) && \text{[Definition of eval]} \\
&\leq \text{eval}(\text{replace}(a, F, L)) + \text{eval}(\text{replace}(a, F, R)) && \text{[By IH and inequality addition]} \\
&= \texttt{Sum}(\text{replace}(a, F, L), \text{replace}(a, F, R)) && \text{[Definition of eval]}
\end{aligned}$$

Case $E = \texttt{Prod}(L, R)$**.** Suppose $\text{eval}(L) \leq \text{eval}(\text{replace}(a, F, L))$ and suppose $\text{eval}(R) \leq \text{eval}(\text{replace}(a, F, R))$ for some expressions L and R. Note that:

$$\begin{aligned}
\text{eval}(\texttt{Prod}(L, R)) &= \text{eval}(L) \times \text{eval}(R) && \text{[Definition of eval]} \\
&\leq \text{eval}(\text{replace}(a, F, L)) \times \text{eval}(\text{replace}(a, F, R)) && \text{[By IH and (positive) inequality mult.]} \\
&\leq \texttt{Prod}(\text{replace}(a, F, L), \text{replace}(a, F, R)) && \text{[Definition of eval]}
\end{aligned}$$

Thus, the claim is true for all expressions by structual induction.

2. Regular Expressions

(a) Write a regular expression that matches base 10 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 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:

$$(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)111(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \varepsilon)$$