Quiz Section 7: Induction, Regular Expressions – Solutions

Task 1 – Walk the Dawgs

Suppose that a dog walker takes care of $n \geqslant 12$ dogs. The dog walker is not a strong person, and will walk dogs in groups of 3 or 7 at a time (every dog gets walked exactly once). Prove that the dog walker can always split the n dogs into groups of 3 dogs or 7 dogs.

Let $P(n)$ be "a group with n dogs can be split into groups of 3 dogs or 7 dogs." We will prove $P(n)$ for all natural numbers $n \geqslant 12$ by strong induction.

Base Cases $n = 12, 13, 14,$ or 15: $12 = 3 + 3 + 3 + 3$, $13 = 3 + 7 + 3$, $14 = 7 + 7$, So $P(12)$, $P(13)$, and $P(14)$ hold.

Inductive Hypothesis: Assume that $P(12), \ldots, P(k)$ hold for some arbitrary $k \geqslant 14$.

Inductive Step: $\boxed{\text{Goal: Show } k+1 \text{ dogs can be split into groups of 3 dogs or 7 dogs.}}$

We first form one group of 3 dogs out of the $k + 1$ dogs. Then we can divide the remaining $k - 2$ dogs into groups of 3 or 7 by the assumption $P(k - 2)$. (Note that $k \geqslant 14$ and so $k - 2 \geqslant 12$; thus, $P(k - 2)$ is among our assumptions $P(12), \ldots, P(k)$.)

Conclusion: $P(n)$ holds for all integers $n \geqslant 12$ by by principle of strong induction.

Task 2 – Seeing double

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 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 every string X, len(double(X)) $= 2\,$len(X).

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

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

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

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

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

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

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

Task 3 – Leafy Trees

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 $\texttt{Tree}(L, R)$ is a **Tree**.

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

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

Also, recall the definition of size on trees:

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

Prove that $\text{leaves}(T) \geqslant \text{size}(T)/2 + 1/2$ for all Trees T.

For a tree T, let P be $\text{leaves}(T) \geqslant \text{size}(T)/2 + 1/2$. We prove P for all trees T by structural induction on T.

Base Case (T = $\bullet$): By definition of $\text{leaves}(\bullet)$, $\text{leaves}(\bullet) = 1$ and $\text{size}(\bullet) = 1$. So, $\text{leaves}(\bullet) = 1 \geqslant 1/2 + 1/2 = \text{size}(\bullet)/2 + 1/2$, so $P(\bullet)$ holds.

Inductive Hypothesis: Suppose $P(L)$ and $P(R)$ hold for some arbitrary trees L, R.

Inductive Step: $\boxed{\text{Goal: Show that } P(\texttt{Tree}(L, R)) \text{ holds.}}$

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

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

Conclusion: Thus, $P(T)$ holds for all trees T by structural induction.

Task 4 – Reversing a Binary Tree

Consider the following definition of a **Tree** that has integer values at its nodes in which each node has at most two children.

Basis Step `Nil` is a **Tree**.

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

The sum function returns the sum of all elements in a **Tree**.

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

The following recursively defined function produces the mirror image of a **Tree**.

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

Show that, for all **Trees** T that

$$\mathsf{sum}(T) = \mathsf{sum}(\mathsf{reverse}(T))$$

For a **Tree** T, let $P(T)$ be "$\mathsf{sum}(T) = \mathsf{sum}(\mathsf{reverse}(T))$". We show $P(T)$ for all **Trees** T by structural induction.

Base Case: By definition we have $\mathsf{reverse}(\texttt{Nil}) = \texttt{Nil}$. Applying sum to both sides we get $\mathsf{sum}(\texttt{Nil}) = \mathsf{sum}(\mathsf{reverse}(\texttt{Nil}))$, which is exactly $P(\texttt{Nil})$, so the base case holds.

Inductive Hypothesis: Suppose $P(L)$ and $P(R)$ hold for some arbitrary **Trees** L and R.

Inductive Step: Let x be an arbitrary integer. $\boxed{\text{Goal: Show } P(\texttt{Tree}(x, L, R)) \text{ holds.}}$

We have,

$$\begin{aligned} \mathsf{sum}(\mathsf{reverse}(\texttt{Tree}(x, L, R))) &= \mathsf{sum}(\texttt{Tree}(x, \mathsf{reverse}(R), \mathsf{reverse}(L))) && \text{[Definition of reverse]} \\ &= x + \mathsf{sum}(\mathsf{reverse}(R)) + \mathsf{sum}(\mathsf{reverse}(L)) && \text{[Definition of sum]} \\ &= x + \mathsf{sum}(R) + \mathsf{sum}(L) && \text{[Inductive Hypothesis]} \\ &= x + \mathsf{sum}(L) + \mathsf{sum}(R) && \text{[Commutativity]} \\ &= \mathsf{sum}(\texttt{Tree}(x, L, R)) && \text{[Definition of sum]} \end{aligned}$$

This shows $P(\texttt{Tree}(x, L, R))$.

Conclusion: Therefore, $P(T)$ holds for all **Trees** T by structural induction.

Task 5 – Recursively Defined Sets of Strings

For each of the following, write a recursive definition of the sets satisfying the following properties. Briefly justify that your solution is correct.

a) Binary strings of even length.

Basis: $\varepsilon \in S$.
Recursive Step: If $x \in S$, then $x00, x01, x10, x11 \in S$.

"Brief " Justification: We will show that $x \in S$ iff x has even length (i.e.,|x|= 2n for some $n \in \mathbb{N}$). (Note: "brief" is in quotes here. Try to write shorter explanations in your homework assignment when possible!)

Suppose $x \in S$. If x is the empty string, then it has length 0, which is even. Otherwise, x is built up from the empty string by repeated application of the recursive step, so it is of the form $x_1 x_2 ... x_n$, where each $x_i \in \{00, 01, 10, 11\}$. In that case, we can see that |x|=$|x_1|$+$|x_2|$+ · · · +$|x_n|$= 2n, which is even. Now, suppose that x has even length. If it's length is zero, then it is the empty string, which is in S. Otherwise, it has length 2n for some $n > 0$, and we can write x in the form $x_1 x_2 ... x_n$, where each $x_i \in \{00, 01, 10, 11\}$ has length 2. Hence, we can see that x can be built up from the empty string by applying the recursive step with x_1, then x_2, and so on up to x_n, which shows that $x \in S$.

b) Binary strings not containing 10.

If the string does not contain 10, then the first 1 in the string can only be followed by more 1s. Hence, it must be of the form $0^m 1^n$ for some $m, n \in \mathbb{N}$.

Basis: $\varepsilon \in S$.

Recursive Step: If $x \in S$, then $0x \in S$ and $x1 \in S$.

Brief Justification: The empty string satisfies the property, and the recursive step cannot place a 0 after a 1 since it only adds 0s on the left. Hence, every string in S satisfies the property.

In the other direction, from our discussion above, any string of this form can be written as $y = 0^m 1^n$ for some $m, n \in \mathbb{N}$. We can build up the string y from the empty string by applying the rule $x \to 0x$ m times and then applying the rule $x \to x1$ n times. This shows that the string y is in S.

c) Binary strings not containing 10 as a substring and having at least as many 1s as 0s.

These must be of the form $0^m 1^n$ for some $m, n \in \mathbb{N}$ with $m \leqslant n$. We can ensure that by pairing up the 0s with 1s as they are added:

Basis: $\varepsilon \in S$.

Recursive Step: If $x \in S$, then $0x1 \in S$ and $x1 \in S$.

Brief Justification: As in the previous part, we cannot add a 0 after a 1 because we only add 0s at the front. And since every 0 comes with a 1, we always have at least as many 1s as 0s.

In the other direction, from our discussion above, any string of this form can be written as xy, where $x = 0^m1^m$ and $y = 1^{nm}$, since $n \geqslant m$. We can build up the string x from the empty string by applying the rule $x \to 0x1$ m times and then produce the string xy by applying the rule $x \to x1$ nm times, which shows that the string is in S.

d) Binary strings containing at most two 0s and at most two 1s.

This is the set of all binary strings of length at most 4 *except* for these:

$$000, 1000, 0100, 0010, 0001, 0000, 111, 0111, 1011, 1101, 1110, 1111$$

Since this is a **finite set**, we can define it recursively using only basis elements and no recursive step.

Task 6 – Regular Expressions

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

$$0 \cup ((1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^*)$$

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

$$0 \cup ((1 \cup 2)(0 \cup 1 \cup 2)^*0)$$

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

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