1. Strong Induction

Consider the function \( a(n) \) defined for \( n \geq 1 \) recursively as follows.

\[
\begin{align*}
  a(1) &= 1 \\
  a(2) &= 3 \\
  a(n) &= 2a(n-1) - a(n-2) \quad \text{for } n \geq 3
\end{align*}
\]

Use strong induction to prove that \( a(n) = 2n - 1 \) for all \( n \geq 1 \).

Solution:

Let \( P(n) \) be \( "a(n) = 2n - 1" \). We will show that \( P(n) \) is true for all \( n \geq 1 \) by strong induction.

**Base Cases** \( (n = 1, n = 2) \):

\( n = 1 \)

\[
a(1) = 1 = 2 \cdot 1 - 1
\]

\( n = 2 \)

\[
a(2) = 3 = 2 \cdot 2 - 1
\]

So, \( P(1) \) and \( P(2) \) hold.

**Inductive Hypothesis:**

Suppose that \( P(j) \) is true for all integers \( 1 \leq j \leq k \) for some arbitrary \( k \geq 2 \).

**Inductive Step:**
We will show \( P(k + 1) \) holds.

\[
a(k + 1) = 2a(k) - a(k - 1) \quad \text{[Definition of } a]\n\]

\[
= 2(2k - 1) - (2k - 1) - 1 \quad \text{[Inductive Hypothesis]}\n\]

\[
= 2k + 1 \quad \text{[Algebra]}\n\]

\[
= 2(k + 1) - 1 \quad \text{[Algebra]}
\]

So, \( P(k + 1) \) holds.

**Conclusion:**

Therefore, \( P(n) \) holds for all integers \( n \geq 1 \) by principle of strong induction.

2. 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 \( \text{len} \):

\[
\begin{align*}
  \text{len}("") &= 0 \\
  \text{len}(\text{append}(c, X)) &= 1 + \text{len}(X)
\end{align*}
\]
Now, consider the following recursive definition:

\[
\text{double(""}) = ""
\text{double(append(c, X}) = append(c, append(c, double(X))).}
\]

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

**Solution:**

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

**Base Case (\(X = "\):** By definition, \(\text{len(double(""}) = \text{len}(""} = 0 = 2 \times 0 = 2\text{len(""})\), so \(P("\) holds.

**Inductive Hypothesis:** Suppose \(P(X)\) holds for some arbitrary string \(X\).

**Inductive Step:** Goal: Show that \(P(\text{append}(c, X))\) holds for any character \(c\).

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

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

**Conclusion:** \(P(X)\) 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 \(\text{Tree}(\bullet, L, R)\) 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 \(\text{size}\) 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 \(\text{leaves}(T) \geq \text{size}(T)/2 + 1/2\) for all **Trees** \(T\).

**Solution:**

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 on \(T\).

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

**Inductive Hypothesis:** Suppose \(P(L)\) and \(P(R)\) hold for some arbitrary trees \(L, R\).
(c) Prove the previous claim using strong induction. Define \( P(n) \) as “all trees \( T \) of size \( n \) satisfy \( \text{leaves}(T) \geq \text{size}(T)/2 + 1/2 \). You may use the following facts:

- For any tree \( T \) we have \( \text{size}(T) \geq 1 \).
- For any tree \( T \), \( \text{size}(T) = 1 \) if and only if \( T = \bullet \).

If we wanted to prove these claims, we could do so by structural induction.

Note, in the inductive step you should start by letting \( T \) be an arbitrary tree of size \( k + 1 \).

**Solution:**

Let \( P(n) \) be “all trees \( T \) of size \( n \) satisfy \( \text{leaves}(T) \geq \text{size}(T)/2 + 1/2 \).” We show \( P(n) \) for all integers \( n \geq 1 \) by strong induction on \( n \).

**Base Case:** Let \( T \) be an arbitrary tree of size 1. The only tree with size 1 is \( \bullet \), so \( T = \bullet \). By definition, \( \text{leaves}(T) = \text{leaves}(\bullet) = 1 \) and thus \( \text{size}(T) = 1 = 1/2 + 1/2 = \text{size}(T)/2 + 1/2 \). This shows the base case holds.

**Inductive Hypothesis:** Suppose that \( P(j) \) holds for all integers \( j = 1, 2, \ldots, k \) for some arbitrary integer \( k \geq 1 \).

**Inductive Step:** Let \( T \) be an arbitrary tree of size \( k + 1 \). Since \( k + 1 > 1 \), we must have \( T \neq \bullet \). It follows from the definition of a tree that \( T = \text{Tree}(\bullet, L, R) \) for some trees \( L \) and \( R \). By definition, we have \( \text{size}(T) = 1 + \text{size}(L) + \text{size}(R) \). Since sizes are non-negative, this equation shows \( \text{size}(T) > \text{size}(L) \) and \( \text{size}(T) > \text{size}(R) \) meaning we can apply the inductive hypothesis. This says that \( \text{leaves}(L) \geq \text{size}(L)/2 + 1/2 \) and \( \text{leaves}(R) \geq \text{size}(R)/2 + 1/2 \).

We have,

\[
\text{leaves}(T) = \text{leaves}(\text{Tree}(\bullet, L, R)) = \text{leaves}(L) + \text{leaves}(R) \geq (\text{size}(L)/2 + 1/2) + (\text{size}(R)/2 + 1/2) = (1/2 + \text{size}(L)/2 + \text{size}(R)/2) + 1/2 = 1 + \text{size}(L)/2 + \text{size}(R)/2 + 1/2 = \text{size}(T)/2 + 1/2.
\]

This shows \( P(k + 1) \).

**Conclusion:** \( P(n) \) holds for all integers \( n \geq 1 \) by the principle of strong induction.

Note, this proves the claim for all trees because every tree \( T \) has some size \( s \geq 1 \). Then \( P(s) \) says that all trees of size \( s \) satisfy the claim, including \( T \).
3. Reversing a Binary Tree

Consider the following definition of a (binary) Tree.

**Basis Step** Nil is a Tree.

**Recursive Step** If L is a Tree, R is a Tree, and x is an integer, then Tree(x, L, R) is a Tree.

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

\[
\begin{align*}
\text{sum(Nil)} &= 0 \\
\text{sum(Tree(x, L, R))} &= x + \text{sum}(L) + \text{sum}(R)
\end{align*}
\]

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

\[
\begin{align*}
\text{reverse(Nil)} &= \text{Nil} \\
\text{reverse(Tree(x, L, R))} &= \text{Tree}(x, \text{reverse}(R), \text{reverse}(L))
\end{align*}
\]

Show that, for all Trees T that

\[
\text{sum}(T) = \text{sum}(\text{reverse}(T))
\]

**Solution:**

For a Tree T, let \( P(T) \) be “\( \text{sum}(T) = \text{sum}(\text{reverse}(T)) \)”. We show \( P(T) \) for all Trees T by structural induction.

**Base Case:** By definition we have \( \text{reverse}(\text{Nil}) = \text{Nil} \). Applying \text{sum} to both sides we get \( \text{sum}(\text{Nil}) = \text{sum}(\text{reverse}(\text{Nil})) \), which is exactly \( P(\text{Nil}) \), 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. Goal: Show \( P(\text{Tree}(x, L, R)) \) holds.

We have,

\[
\begin{align*}
\text{sum}(\text{reverse}(\text{Tree}(x, L, R))) &= \text{sum}(\text{Tree}(x, \text{reverse}(R), \text{reverse}(L))) \quad \text{[Definition of reverse]} \\
&= x + \text{sum}(\text{reverse}(R)) + \text{sum}(\text{reverse}(L)) \quad \text{[Definition of sum]} \\
&= x + \text{sum}(R) + \text{sum}(L) \quad \text{[Inductive Hypothesis]} \\
&= x + \text{sum}(L) + \text{sum}(R) \quad \text{[Commutativity]} \\
&= \text{sum}(\text{Tree}(x, L, R)) \quad \text{[Definition of sum]}
\end{align*}
\]

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

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

4. 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.

**Solution:**

\[
\begin{align*}
\text{Basis}: & \quad \varepsilon \in S. \\
\text{Recursive Step}: & \quad \text{If } x \in S, \text{ then } x00, x01, x10, x11 \in S. \\
\text{Exclusion Rule}: & \quad \text{Each element of } S \text{ is obtained from the basis and a finite number of applications of the recursive step.}
\end{align*}
\]
“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 \( x_1x_2...x_n \), where each \( x_i \in \{00, 01, 10, 11\} \). In that case, we can see that \(|x| = |x_1| + |x_2| + \cdots + |x_n| = 2n\), 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 \( x_1x_2...x_n \), 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.

Solution:

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 \( 0^m1^n \) for some \( m, n \in \mathbb{N} \).

**Basis:** \( \varepsilon \in S \).

**Recursive Step:** If \( x \in S \), then \( 0x \in S \) and \( x1 \in S \).

**Exclusion Rule:** Each element of \( S \) is obtained from the basis and a finite number of applications of the recursive step.

*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 \( y = 0^m1^n \) for some \( m, n \in \mathbb{N} \). We can build up the string \( y \) from the empty string by applying the rule \( x \rightarrow 0x \) \( m \) times and then applying the rule \( x \rightarrow x1 \) \( n \) 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.

**Solution:**

These must be of the form \( 0^m1^n \) for some \( m, n \in \mathbb{N} \) with \( m \leq n \). 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 \( x1 \in S \).

**Exclusion Rule:** Each element of \( S \) is obtained from the basis and a finite number of applications of the recursive step.

*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 \( y = 1^{n-m} \), since \( n \geq m \). We can build up the string \( x \) from the empty string by applying the rule \( x \rightarrow 0x1 \) \( m \) times and then produce the string \( xy \) by applying the rule \( x \rightarrow x1 \) \( n-m \) times, which shows that the string is in \( S \).

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

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

000, 1000, 0100, 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.

5. Regular Expressions

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

Solution:

0 ∪ (1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9)0 ∪ 1 ∪ 2 ∪ 3 ∪ 4 ∪ 5 ∪ 6 ∪ 7 ∪ 8 ∪ 9)∗

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

Solution:

0 ∪ ((1 ∪ 2)(0 ∪ 1 ∪ 2)∗0)

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

Solution:

(01 ∪ 001 ∪ 1∗)∗(0 ∪ 00 ∪ ε)111(01 ∪ 001 ∪ 1∗)∗(0 ∪ 00 ∪ ε)