CSC236 Problem Set 1

Xuanqi Wei 1009353209

September 18, 2023

Contents

1 Question 1 **1**

2 Question 2 **2**

3 Question 3 **5**

1 Question 1

(a) According to the definition of P:

$$\forall g_1 \in G_1,\ \exists t_1 \in T_1,\ t_1\ tiles\ g_1 \implies \forall g_2 \in G_2,\ \exists t_2 \in T_2,\ t_2\ tiles\ g_2$$

(b) Firstly, assume

$$\forall g_1 \in G_1,\ \exists t_1 \in T_1,\ t_1\ tiles\ g, \text{ which is the antecedent.}$$

Secondly, I will do the consequent part, which:

$$Let\ g_2\ be\ an\ arbitrary\ element\ from\ G_2$$

Then, I want to prove that

$$\exists t_2 \in T_2,\ t_2\ tiles\ g_2$$

by selecting a satisfying element t_2 from T_2 and prove the element t_2 satisfies t_2 *tiles* g_2.

(c) The diagram above illustrates one instance of G_2 grids, which being tiled by triominoes.

Firstly, we already know that for P(1), the statement $\forall g_1 \in G_1,\ \exists t_1 \in T_1,\ t_1\ tiles\ g$ is true which is the antecedent of this direct proof.

Secondly, the above diagram is an element of the set of all $2^2 \times 2^2$ grid with one square removed, which is an element of G_2. By visulising those colorful triominoes, we see a combination triominoes, t_2, which is an element of the set of all tilings of elements of G_2 using triominoes, belonging to T_2, exists and tiles g_2.

Therefore, the diagram above illustrates an instance of that direct proof.

(d) Given the statement to prove: $\forall n \in \mathbb{N},\ P(n)$, which for each natural n you can tile any $2^n \times 2^n$ grid with one cell missing using only triominoes.

Proof: We prove this by Simple Induction on n.

Base Case: Let $n = 1$.

Since G_1 is the set of all $2^1 \times 2^1$ grids with one cell removed, which by definition is a single triominoe.

Therefore, $\forall g_1 \in G_1,\ \exists t_1 \in T_1,\ t_1\ tiles\ g_1$ is true, which P(1) is true.

Induction Step: Let $k \in \mathbb{N}$.

Induction Hypothesis: Assume that $P(k)$ is true.

By Induction Hypothesis, we know that $P(k)$ is true, which $\forall g_k \in G_k,\ \exists t_k \in T_k,\ t_k\ tiles\ g_k$ is true. I will take 3 different g_ks, the first with right button corner square missing, the second with right top corner square missing, and the third with left top corner square missing. I will make the missing corners in these 3 g_ks face inwards and add a triomino which will result in getting a 'L' shape. The remaining $\frac{1}{4}$ place is missing a cell to form a g_{k+1}, which can actually be an arbitraty element from G_k. By Induction Hypothesis, since $\forall g_k \in G_k,\ \exists t_k \in T_k,\ t_k\ tiles\ g_k$ is true, the remaining G_k place can be covered by trimonoes, proving the $P(k+1)$ is true.

Therefore, we've proved $\forall n \in \mathbb{N},\ P(n)$ is true.

$\blacksquare$

2 Question 2

```
1   from typing import Any
2
3
    3 usages   Henry-wxq +1
4   def q_2(n: int, x: Any) -> Any:
5       """Implement a Python function with parameters x and n that (ignoring floating-point issues) returns c_n.
6
7       Precondition:
8       1. x represents a non-zero real number.
9       2. n is a natural number
10      """
11      # Since c_1 is used in both n = 1 and recursion, I will put it at the front.
12      c_1 = x + 1 / x
13
14      if n == 1:
15          # From the definition of c_n, when n is 1, return the corresponding c_1
16          return c_1
17      elif n == 2:
18          # From the definition of c_n, when n is 2, return the corresponding c_2
19          c_2 = x * x + (1 / x) * (1 / x)
20          return c_2
21      else:
22          """This is the recursion part. According to the discovery from hint which will be stated below, I come up with a
23          general function for c_n.
24          """
25          # Aim at returning the recursive value of c for n minus 1 after reaching the case when n equals to 1.
26          c_minus1 = q_2(n-1, x)
27          # Aim at returning the recursive value of c for n minus 2. Since we don't know whether n is an even number or
28          # an odd number, we need to add both n equals to 1 and n equals to 2 to our base case.
29          c_minus2 = q_2(n-2, x)
30          # Calculate the c_n based on the discovery.
31          c_n = c_1 * c_minus1 - c_minus2
32          return c_n
```

Figure 1: Python function for Q2-a

(a) The above code is the Python function with parameter x and n that (ignoring floating-point issues) returns c_n, the comments are both in the code above and below.

Firstly, I clearly stated the pre-conditions on x and n in a header comment, which $x \in \mathbb{R}/\{0\}$ and $n \in \mathbb{N}$.

Secondly, at line 12, I write the calculation of c_1 because it will be used in both 'if' statement at line 14 and 'else' statement at line 21, avoiding redundancy.

Thirdly, I implemented the based case when n equals to 1 and n equals to 2 according to the definition of c_n.

Fourthly, I implemented the recursion based on the discovery from hint.

$$
\begin{aligned}
(x + \frac{1}{x}) \cdot (x^n + \frac{1}{x^n}) &= x^{n+1} + \frac{1}{x^{n-1}} + x^{n-1} + \frac{1}{x^{n+1}} \\
&= (x^{n+1} + \frac{1}{x^{n+1}}) + (x^{n-1} + \frac{1}{x^{n-1}})
\end{aligned}
$$

According to the definition of c_n, gives:

$$c_1 \cdot c_n = c_{n+1} + c_{n-1}$$
$$\implies c_{n+1} = c_1 \cdot c_n - c_{n-1}$$

Thus, we generalize the above equation into: $c_n = c_1 \cdot c_{n-1} - c_{n-2}$, which is the core of our recursive part, at line 31.

Fifthly, aiming at returning the recursive value of c_{n-1} after reaching the case when n eqials to 1, I write the code line 26. Aiming at return the recursive value of c_{n-2}, I write the code at line 29. Since we don't know whether n is an even number or an odd number, we need to add both n equals to 1 and n equals to 2 to our base case at line 16 and at line 19.

Finally, we can obtain the c_n using the recursive function without use any loops, or any helper functions, nor call any exponentiation functions.

(b) To state a recurrence for the sequence c, I will take $n = 3$, which a natural number satisfying the precondition, and I will take $x = 2$, which is a non-zero real number.

Firstly, it will calculate the c_1 which $c_1 = 2 + \frac{1}{2} = 2.5$.

Then it will go directly into the 'else' statement which will get into calculate c_{n-1} calling the function q_2 recursively. We put $n - 1 = 2$ and $x = 2$ into the function again which this time goes into the 'elif' statement and return $c_2 = 4.25$. Thus we get, in this case, $c_{n-1} = 4.25$. Continuing, it will get into calculate c_{n-2} calling the function recursively. We put $n - 2 = 1$ and $x = 2$ into the function agiain which this time go into the 'if' statement and return $c_1 = 2.5$ Thus we get, in this case, $c_{n-2} = 2.5$

Finally, the above values goes into our equation which gives:

$$c_n = c_1 \cdot c_{n-1} - c_{n-2}$$
$$= 2.5 \times 4.25 - 2.5 = 8.125$$

(c) If $x + \frac{1}{x}$ is an integer, then $x^n + \frac{1}{x^n}$ is an integer for each $n \in \mathbb{N}$

Given statement to prove: $\forall n \in \mathbb{N},\ P(n)$, which $P(n)$: $x^n + \frac{1}{x^n}$ is an integer where $x \in \mathbb{R}/\{0\}$

Proof: We prove this by complete induction on n.

Base Case: Let $n = 1$

By assumption, $P(n) = x + \frac{1}{x}$ is an integer.

We've proved that $P(1)$ is true.

Induction Step: Let $n > 1$

Induction Hypothesis: Assume $\forall k,\ 1 \leq k < n,\ P(k)$

WTS: $P(n)$

From induction hypothesis, when $k_1 = 1,\ 1 \leq k_1 < n,\ P(1)$ is true, which $x + \frac{1}{x}$ is an integer.

From induction hypothesis, when $k_{n-1} = n - 1,\ 1 \leq k_{n-1} < n$, $P(n-1)$ is true, which $x^{n-1} + \frac{1}{x^{n-1}}$ is an integer.

From induction hypothesis, when $k_{n-2} = n - 2,\ 1 \leq k_{n-2} < n$, $P(n-2)$ is true, which $x^{n-2} + \frac{1}{x^{n-2}}$ is an integer.

Thus, we obtain that

$$\begin{aligned}
&(x + \frac{1}{x}) \cdot (x^{n-1} + \frac{1}{x^{n-1}}) - (x^{n-2} + \frac{1}{x^{n-2}}) \text{ is an integer.} \\
&= x^n + \frac{1}{x^{n-2}} + x^{n-2} + \frac{1}{x^n} - x^{n-2} - \frac{1}{x^{n-2}} \\
&= x^n + \frac{1}{x^n} \text{ is an integer}
\end{aligned}$$

I've proved that $P(n)$ is true.

To conclude, I've proved that $x + \frac{1}{x}$ is an integer, then $x^n + \frac{1}{x^n}$ is an integer for each $n \in \mathbb{N}$.

■

3 Question 3

(a)

```python
def q3_a_func(n: int) -> int:
    """Implement a Python function that takes a positive natural number n and returns a_n.

    Precondition: n is a positive natural number
    """
    if n == 1:
        # From the definition of a_n, when n is 1, return a_n equals to 1.
        return 1
    else:
        """ This is the recursion. Aim at returning the recursive value of a_n after reaching the base case when
        a_n equals to 1.
        """
        return q3_a_func(math.floor(math.sqrt(n))) * q3_a_func(math.floor(math.sqrt(n))) \
            + 2 * q3_a_func(math.floor(math.sqrt(n)))
```

Figure 2: Python function for Q3-a

(b)

```python
def q3_b_func(n: int) -> int:
    """Implement a Python function that takes a positive natural number n and raises an exception if n is 1, otherwise
    it returns a_n.

    Precondition: n is a positive natural number
    """
    if n == 1:
        # By question requirement, when n is 1, raises an Exception.
        raise Exception("Sorry, n must be greater than 1")
    elif n == 2 or n == 3:
        """Since when n equals to 2 or n equals to 3, the floor of square root of n is 1, and, in this function, we
        don't have the value of a_n when n equals 1. Thus, we need to manually add the value of a_n when n equals to 2
        and n equals to 3 to prevent the error when calling the recursive.
        """
        return 3
    else:
        """This is the recursion. Aim at returning the recursive value of a_n after reaching the case when a_n equals
        to 2 or a_n equals to 3.
        """
        return q3_b_func(math.floor(math.sqrt(n))) * q3_b_func(math.floor(math.sqrt(n))) \
            + 2 * q3_b_func(math.floor(math.sqrt(n)))
```

Figure 3: Python function for Q3-b

(c) When $n_0 = 2$, n_0 is the smallest natural n_0 so that a_n is a multiple of 3 for each natural $n \geq n_0$.

Given statement to prove: $\forall n \in \mathbb{N},\ n \geq n_0,\ P(n)$, which $P(n) : a_n$ is a multiple of 3.

Let $n \in \mathbb{N}$.

Proof: We prove this by complete induction on n.

Base Case: Let $2 \leq n < 4$.

$$\begin{aligned} P(2): \ a_2 &= (a_{\lfloor\sqrt{2}\rfloor})^2 + 2 \cdot a_{\lfloor\sqrt{2}\rfloor} \\ &= a_1^2 + 2 \cdot a_1 = 1^2 + 2 = 3 \text{is a multiple of } 3. \end{aligned}$$

$$
\begin{aligned}
P(3):\ a_3 &= (a_{\lfloor\sqrt{3}\rfloor})^2 + 2\cdot a_{\lfloor\sqrt{3}\rfloor} \\
&= a_1^2 + 2\cdot a_1 = 1^2 + 2 = 3 \text{is a multiple of 3.}
\end{aligned}
$$

Thus, I've proved the base case is true.

Induction Step: Let $n \geq 4$.

Induction Hypothesis: Assume $\forall k,\ 2 \leq k < n,\ P(k)$

Since $n \geq 4$, gives $\lfloor\sqrt{n}\rfloor < n$.

Since $\lfloor\sqrt{n}\rfloor < n$ and $4 \leq n$, gives $2 \leq \lfloor\sqrt{n}\rfloor$ as 2 is the smallest value of $\lfloor\sqrt{n}\rfloor$, which gives,

$$2 \leq \lfloor\sqrt{n}\rfloor < n$$

Since $\lfloor\sqrt{n}\rfloor$ is an integer which $\lfloor\sqrt{n}\rfloor \geq 2$, from induction hypothesis, we can always find $k' = \lfloor\sqrt{n}\rfloor$, which $P(k')$ is true and $a_{k'} = 3p,\ p \in \mathbb{N}$.

Thus gives,

$$
\begin{aligned}
a_n &= (\lfloor\sqrt{n}\rfloor)^2 + 2\cdot a_{\lfloor\sqrt{n}\rfloor} \\
&= (a_{k'})^2 + s\cdot a_{k'} \\
&= (3p)^2 + 2\cdot(3\cdot p) \\
&= 9\cdot p^2 + 6\cdot p \\
&= 3\cdot(3\cdot p^2 + 2p)
\end{aligned}
$$

Let $q = 3\cdot p^2 + 2\cdot p$. Since $p \in \mathbb{N}$, gives $q \in \mathbb{N}$, which

$$a_n = 3q,\ q \in \mathbb{N} \text{ , where } a_n \text{ is a multiple of 3.}$$

I've proved that $P(n)$ is true.

To conclude, I've proved when $n_0 = 2$, n_0 is the smallest natural n_0 so that a_n is a multiple of 3 for each natural $n \geq n_0$.

■