Problem Sheet 3

Henri Sota
h.sota@jacobs-university.de
Computer Science 2022

October 4, 2019

Problem 3.1

If we have a set $X = \{x_1, x_2, x_3, \dots, x_n\}$ with cardinality n, then $\mathcal{P}(X)$ has 2^n elements. (1)

Proof. We use induction. The induction hypothesis will be theorem 1.
Base Case: Our base case is $\mathcal{P}(A)$ when A has 0 elements ($n = 0$).
In this case, $A = \{\}$ is an empty set and its power set, $\mathcal{P}(A) = \{\{\}\}$, is a set with one element.
We want to claim that $|\mathcal{P}(B)| = 2 * |\mathcal{P}(A)|$, where set B has one more element than set A.
Induction Step: Assume that theorem 1 holds.

$$B = A \cup \{a_{n+1}\} \quad | \quad A = \{a_1, a_2, a_3, \dots, a_n\}, B = \{a_1, a_2, a_3, \dots, a_n, a_{n+1}\}$$

According to the inductive hypothesis and the union of set A with element a_{n+1}, $\mathcal{P}(B)$ has all the subsets of A, which are 2^n, and all subsets of the form $A_{subset} \cup \{a_{n+1}\}$ since every new combination of the power set is formed by appending the new element at each existing subset. Therefore there are 2^n subsets, which do not include a_{n+1} and who are also subsets of A, and there are 2^n subsets who have a_{n+1}. Total number of subsets of B:

$$|\mathcal{P}(B)| = 2^n + 2^n = 2 \cdot 2^n = 2^{n+1}$$

So it follows by induction that $\mathcal{P}(B)$ contains twice the amount of elements that $\mathcal{P}(A)$ contains, when $B = A \cup \{a_{n+1}\}$.

■

Problem 3.2

a) $R = \{(a,b)a, b \in \mathbb{Z} \land a \neq b\}$

- Not Reflexive: $\forall a \in \mathbb{Z} \quad (a,a) \notin \mathbb{Z}$ because $a = a$ i.e. a = 3
- Symmetric: $\forall a, b \in \mathbb{Z} \quad (a,b) \in \mathbb{Z}$ because if $a \neq b$ then $b \neq a$ i.e. a = 3 , b = 4
- Not Transitive:
  $\forall a, b \in \mathbb{Z} \quad (a,b) \in \mathbb{R}, a \neq b$
  $\forall b, c \in \mathbb{R} \quad (a,b) \in \mathbb{R}, b \neq c$
  This doesn't mean that $a \neq c$ because $a = c$ i.e. a = 3, b = 4, c = 3

b) $R = \{(a,b)a, b \in \mathbb{Z} \land |a - b| \leq 3\}$

- Reflexive: $\forall a \in \mathbb{Z} \quad (a,a) \in \mathbb{Z}$ because $|a - a| = 0 \leftrightarrow 0 \leq 3$

- Symmetric: $\forall a, b \in \mathbb{Z} \quad (a, b) \in \mathbb{Z}$

$$
\begin{aligned}
|a-b| &\leq 3 \\
-3 < a &- b < 3 \\
b - 3 < &a < b + 3 \\
a - 3 < &b < a + 3
\end{aligned}
$$

  From definition of absolute value: $|a-b| = |b-a|$

- Not Transitive:
  $\forall a, b \in \mathbb{Z} \quad (a, b) \in \mathbb{Z},\ a \neq b$
  $\forall b, c \in \mathbb{Z} \quad (b, c) \in \mathbb{Z},\ b \neq c$
  This doesn't mean that $a = c$ in case that $|a-b| = |b-c|$ i.e. $a = 3, b = 4, c = 5$

c) $R = \{(a, b) a, b \in \mathbb{Z} \wedge (a \bmod 10) = (b \bmod 10)\}$

- Reflexive: $\forall a \in \mathbb{Z} \quad (a, a) \in \mathbb{Z}$ because $a \bmod 10 = a \bmod 10$
- Symmetric:
  $\forall a \in \mathbb{Z} \quad (a, b) \in \mathbb{Z}$

$$
a \bmod 10 = b \bmod 10 \leftrightarrow b \bmod 10 = a \bmod 10
$$

  i.e. $a = 7, b = 17$

- Transitive:
  $\forall a, b \in \mathbb{Z} \quad (a, b) \in \mathbb{Z}, \quad a \mod 10 = b \mod 10$
  $\forall b, c \in \mathbb{Z} \quad (b, c) \in \mathbb{Z}, \quad b \mod 10 = c \mod 10$
  As a result, $a \mod 10 = c \mod 10$ i.e. $a = 7,\ b = 17,\ c = 27$

Problem 3.3

a)
```
-- a) Code isPrime
-- Pattern matching combined with guards in order to test primality of input
-- Return True if it is prime or return False otherwise
-- We rule out case 1 and 2; Entering a negative input also returns False
-- Next we check if any division of our input with each number from
-- 2 to sqrt(input) produces a remainder of 0
-- If yes, our input is not prime and we return False
-- Otherwise, our input is a prime number and return True
isPrime :: Integer -> Bool
isPrime 1 = False
isPrime 2 = True
isPrime n
    | (n < 1) = False
    | (length [x | x <- [2 .. ((truncate(sqrt(fromIntegral n))))], mod n x ==
    0]) > 0 = False
    | otherwise = True
```

b)
```
-- b) Code isCircPrime
-- Guards are used to check if our input is a circular prime number
-- First we convert our input x to String, on which we apply function circle
   to
-- produce all possible rotations of it. Then we map each string from the list
-- of strings we got to function read which converts each string to Integer
-- We map each Integer with isPrime to produce a list of Boolean values
-- Using all function we check if all Booleans are True
-- If all Booleans are True, then x is circular prime, we return True
-- Otherwise x is not a circular prime, we return False
isCircPrime :: Integer -> Bool
isCircPrime x
  | (all (==True) (map isPrime (map (read :: String->Integer) (circle (show x)
   )))) == True = True
  | otherwise = False
```

In order to test these two functions, I've used multiple calls to these functions with different input, namely -1, list of all numbers between 2 and 100 for both of them, which respectively produced:

- isPrime: -1 -> False; [2..100] -> [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
- isCircPrime -1 -> False; [2..100] -> [2,3,5,7,11,13,17,31,37,71,73,79,97]