Exercise 1

Exercise 1a

Proof whether the following function is primitive recursive.

$$f(x) = x!$$

In order to do so we simply define f as

$$f(x) = \begin{cases} f(0) = 1 \\ f(x+1) = s(x) \cdot f(x) \end{cases}$$

or to be more precise

$$f(x) = \begin{cases} f(0) = s(o(0)) \\ f(x+1) = g(s(x), f(x)) \end{cases}$$

Since, we have already proven that $g(x, y) = x \cdot y$ is primitive recursive and since $s(x)$ and $o(x)$ are primitive recursive. We have now constructed our function f by means of *composition* and *primitive recursion* from primitive recursive functions. Thus, we conclude f is primitive recursive.

Exercise 1b

Proof whether the following function is primitive recursive.

$$\overline{sg}(x) = \begin{cases} 0 & x > 0 \\ 1 & x = 0 \end{cases}$$

In order to do so we simply define $\overline{sg}$ as

$$\overline{sg}(x) = \begin{cases} \overline{sg}(0) = 1 \\ \overline{sg}(x+1) = 0 \end{cases}$$

or to be more precise

$$\overline{sg}(x) = \begin{cases} \overline{sg}(0) = s(o(0)) \\ \overline{sg}(x+1) = o(x) \end{cases}$$

However, the problem with this is, that I am not sure whether this is an appropriate use of primitive recursion. Therefore, I would propose another construct, i.e.

$$\overline{sg}(x) = \begin{cases} \overline{sg}(0) = s(o(0)) \\ \overline{sg}(x+1) = g(o(x), \overline{sg}(x)) \end{cases}$$

with $g(x, y) = x \dot{-} y$. Since, we have already proven that $g(x, y) = x \dot{-} y$ is primitive recursive and since $s(x)$ and $o(x)$ are primitive recursive. We have now constructed our function $\overline{sg}$ by means of *composition* and *primitive recursion* from primitive recursive functions. Thus, we conclude $\overline{sg}$ is primitive recursive.

Exercise 1c

Proof whether the following function is primitive recursive.

$$Rem(x, y) = \begin{cases} 0 & x = 0 \\ \text{the reminder of dividing } y \text{ by } x & otw. \end{cases}$$

In order to do so we simply define *Rem*. Firstly, we observe that $y - zx = r$ for the first z s.t. $y - zx < x$ Hence, if we use bounded minimisation we can construct something like

$$h(x, y) = \mu z \leq Proj_1^1(y)[y \dot{-} s(z) \cdot x = 0]$$

Hence, h will return either the last z before the equation $y - zx = 0$ is fulfilled or $y + 1$. Furthermore, y was chosen as a bound, because $y \dot{-} (y + 1)x > 0$ then x must be smaller than 1. However, if were to construct our function as naively as presented below

$$f(x, y) = y \dot{-} h(x, y)x$$

we would have the following problems:

1. if $y \dot{-} h(x, y)x = x$, i.e. if x is a devisor of y it should return 0 instead of x;
2. if $x = 0$;

Therefore, we simply adapt f such that,

$$f'(x, y) = sg(x \dot{-} (y \dot{-} h(x, y) \cdot x))(y \dot{-} h(x, y) \cdot x)$$

Since, if the remainder is greater or equal to the divisor one can simply continue the division. However, $y \dot{-} h(x, y)x > x$ is impossible, thus only the case $y \dot{-} h(x, y)x = x$ is relevant. Therefore, $x \dot{-} y - h(x, y)x = 0$ if and only if either the divisor is equal to the remainder or $x = 0$. In the first case, we could simply subtract x once more, thus resulting in 0 anyway and in the second case we want to return 0 by definition. Hence, if we multiply our result simply by 0 if the equation $x \dot{-} y - h(x, y)x$ returns 0 and 1 otherwise.
Hence, we can define

$$Rem(x, y) = f'(x, y) = sg(x \dot{-} (y \dot{-} h(x, y) \cdot x))(y \dot{-} h(x, y) \cdot x)$$

Since, we only used bounded minimisation, which is still primitive recursive, and since s, $\cdot$, $\dot{-}$ and sg are also primitive recursive, we have now constructed our function *Rem* by means of *composition*, *bounded minimisation* and *primitive recursion* from primitive recursive functions. Thus, we conclude *Rem* is primitive recursive.

Exercise 1d

Proof whether the following predicate is primitive recursive.

$$Fib(x) \text{ which is true iff } x \text{ is a Fibonacci number.}$$

Firstly, we define the function $fib(x)$, s.t.

$$fib(x) = x\text{-th Fibonacci number}$$

by only using primitive the primitive recursive functions s, $+$, $\dot{-}$ as well as *composition* and *primitive recursion*. That is,

$$fib(x) \begin{cases} fib(0) = 0 \\ fib(x + 1) = fib(x) + fib(x \dot{-} 1) + (2 \dot{-} s(x)) \end{cases}$$

Here $(2 \dot{-} s(x))$ is required to model the special case $fib(1) = 1$.
Now we can use fib to define χ_{Fib}, i.e.

$$\chi_{Fib}(x) = \overline{sg}(|fib(h(x)) - x|)$$

with h being

$$h(x) = \mu z \leq x + 5[|fib(z) - x| = 0]$$

We can select $x + 5$ as $\forall x \in \omega : x \geq 5 \implies fib(x) \geq x$. Hence, we simply ensure that the cases below 5 need not to be considered, as from there on $|fib(z) - x|$ can only increase if we have passed x, which means we can abort the search. Thus z will be the first value such that $|fib(z) - x| = 0$ if and only x is a fibonacci number. Otherwise, $|fib(z) - x|$ will at some point increase until we hit our bound. Then we simply use the $\overline{sg}$ function to turn 0 into 1 and $|fib(h(x)) - (x + 5 + 1)|$ into 0. Thus we have constructed the desired characteristic function for the predicate Fib, by only relying on primitive recursive functions and their composition together with bounded minimisation. Hence, our predicate is also primitive recursive.

Exercise 1e

Proof whether the following set is primitive recursive.

Any co-finite set.

Firstly, a co-finite subset of a set X is a subset A of X for which the relative complement $X \backslash A$ is finite. In our case the $X = \omega$.
Secondly, we begin by constructing a characteristic function for the finite set $B \coloneqq X \backslash A$. This finite set can be split into $B_0, B_1, ...B_k$ sets such that $\forall i \in [0, k] : |B_i| = 1$ and $X_i \cap X_j = \varnothing$ for $i \neq j$. Hence, for each B_i one can formulate the following characteristic function

$$\chi_{b_i}(x) = \overline{sg}(|e_i - x|)$$

In this case e_i is simply the constant function corresponding to the one element in B_i. Finding this function is primitive recursive, because apart from the fact that it is mentioned in the script that the characteristic function of a finite set is primitive recursive, one could argue as follows (However, I am not entirely sure whether this argument holds). One can understand this one element set as a sequence, i.e. we impose an order onto the set, we can apply the sequence to $Proj_1^1$ thus obtaining the constant function for the one element of the set.
Following this we can now construct the following function

$$f(x) = \chi_{b_0}(x) + \chi_{b_1}(x) + ... + \chi_{b_k}(x)$$

Do to the previously stated conditions, there can only be one i s.t. $\chi_{b_i}(x) = 1$. Hence, this sum is 1 if and only if $x \in B$ else it is 0. Thus, we have obtained the characteristic function for B, i.e $f(x) = \chi_B(x)$
Now we simply use composition to obtain the characteristic function of A. That is,

$$\chi_A(x) = \overline{sg}(\chi_B(x))$$

This is possible, since every element which is not in the finite set B must be in its complement, i.e. $A = X \backslash B$. Thus, if the characteristic function of B evaluates to 1 (0) then the characteristic function of A must therefore evaluate to 0 (1) respectively. Furthermore, since χ_B is primitive recursive (i.e. we only used composition of other primitive functions) and since $\overline{sg}$ is primitive recursive, their composition, i.e. χ_A is also primitive recursive.

Exercise 2

Describe the function which is the result of application of the minimization operator to

$$g(x, y, z) = |zy - x|$$

If we apply the minimsation operator we obtain

$$f(x,y) = \mu z[g(x,y,z) = 0] = \mu z[|zy - x| = 0]$$

this can further be rewritten into

$$f(x,y) = \begin{cases} z & \text{if } |0y-x|, ...|(z-1)y-x| \text{ are defined and not equal to 0 and } |zy-x| = 0 \text{ and defined} \\ \uparrow & otw. \end{cases}$$

Hence, z is the first z s.t. $|zy - x| = 0$. This means the function f converges, if there is an z such that $zy = x$, which is $z = \frac{x}{y}$ such that the remainder of this division is 0. Hence, if f converges we know y is a divisor (in $\mathbb{Z}$) of x and its division results in z. Therefore, f is the devision defined in $\mathbb{Z}$, i.e.

$$f(x,y) = \begin{cases} \frac{x}{y} & y \text{ divisor of } x \text{ in } \mathbb{Z} \\ \uparrow & otw. \end{cases}$$

Exercise 3

Let $f(x,y) = x \dot{-} y$. Write a PRL-program P such that $Func_{2,1}(P,x)$ computes f.

Firstly, we define the function $g(x) = x \dot{-} 1$. Therefore, we define the Program P_g such that $Func_{2,1}(P_g,x)$ computes g.

```
loop X1 ;
X4 ← X3 ;
X3 ← X3 + 1 ;
end ;
X1 ← X4 ;
X3 ← 0 ;
X4 ← 0 ;
```

Here we simply add X_1 times a 1 to X_3, but before the increment is made, we copy the current value of X_3 to X_4. Thus after X_1 iterations the value of $X_4 = X_1 - 1$. In a last step we simply output the calculated value, by setting the value of X_1 to the value of X_4 and clear the used register. The special case $0 \dot{-} 1$ is also considered, since if we do not enter the loop X_4 remains 0, and therefore we simply have $X_1 = 0$, which is exactly the behaviour we desire.
Now we can simply wrap P_g into another loop, i.e.

```
loop X2 ;
Pg ;
end ;
X2 ← 0 ;
```

Which expanded results in,

```
loop X2 ;
loop X1 ;
X4 ← X3 ;
X3 ← X3 + 1 ;
end ;
X1 ← X4 ;
X3 ← 0 ;
X4 ← 0 ;
end ;
X2 ← 0 ;
```

Exercise 4

For this exercise we use the definition of the $-$-function presented in the script:

$$-(x,y) = \begin{cases} x-y & x \geq y \\ \uparrow & x < y \end{cases}$$

The basic procedure of the program is as follows:

1. Go to the first 0 between x and y
2. Check if $y = 0$, i.e if there is a 0 to the right
3. If $y = 0$, go back to the first 0 before x and end the program
4. Else go to the end of y
5. Remove the last 1 from y
6. Go to the 0 separating x and y
7. Go to the beginning of x
8. If the first 1 of x does exist, remove it
9. Else loop
10. If the second 1 of x does exist, remove it
11. Else loop
12. Now we have the problem $f(x-2, y-1) = (x-2) - 2 * (y-1)$ and we start again

```
# Go to the first element of the first input
1: R then 2

# Loop over first input until 0 separator is reached
2: if t then 4 else 3
3: R then 2

# Go to the first element of the second input
4: R then 5

# if there is no first element of the second input (i.e. we encounter a 0)
# initialise the end of the program
5: if t then 21 else 6

# else loop until the end of the second input is reached
6: R then 7
7: if t then 8 else 6

# Go to the last element of the second input
8: L then 9

# Remove the last element of the second input (i.e. set the last 1 to 0)
# (there must be at least 1)
```

```
9: d0 then 10

# Loop over the second input until the 0 separator is reached
10: L then 11
11: if t then 12 else 10

# Loop over the first input until the first 0 is reached
12: L then 13
13: if t then 14 else 12

# Go to the first element of the first input
14: R then 15

# if there is no first element of the first input (i.e. we encounter a 0)
# diverge
15: if t then 19 else 16

# Remove the first element of the first input (i.e. set the last 1 to 0)
16: d0 then 17

# Go to the second element of the first input
17: R then 18

# if there is no second element of the first input (i.e. we encounter a 0)
# diverge
18: if t then 19 else 20
19: R then 19

# Remove the second element of the first input (i.e. set the last 1 to 0)
# and start all over again
20: d0 then 1

# If there is no first element of the second input
# we can stop the program.

# Go to the last element of the first input
21: L then 22

# Loop over the first input until the first 0 is reached
22: L then 23
23: if t then 0 else 22
```

Exercise 5

Prove that if $f(x) = \mu y[g(x,y) = 0]$ and $g(x,y)$ is register-computable, then $f(x)$ is also register-computable.
Before we continue with the proof we define a macro $copy(x,y,z)$ that computes copies the value of X_x to X_y by using X_z.

```
copy(x,y,z):
1: if t_x then 5 else 2
```

2: S_x then 3
3: A_y then 4
4: A_z then 1
5: if t_z then 0 else 6
6: S_z then 7
7: A_x then 5

This program simply decreases the value of register X_x and increases the values of the registers X_y and X_z. If the value of $X_x = 0$ we just transfer the value of X_z back to X_x, by decrementing the prior and incrementing the latter, during every given step until $X_z = 0$. This macro greatly enhances readability of the program P_f, which we will define in a manner such that it computes f. Firstly, since g is register-computable, there must exist a register program P_g which computes the function g. To do so we first define $r \in \omega$ as a high number, s.t. $max\{i|X_i \text{ used in program } P_g\} < r$

1: if t_{r+1} then 3 else 2
2: S_{r+1} then 1
3: $copy(1, r, r+2)$ then 4
4: $copy(r+1, 2, r+2)$ then 5
5: P_g
6: if t_1 then 9 else 7
7: A_{r+1} then 8
8: $copy(r, 1, r+2)$ then 4
9: $copy(r+1, 1, r+2)$ then 0

Our program P_f performs as follows:
Firstly, we make sure that the register X_{r+1} representing y, is empty. This is accomplished by *Command 1* and *Command 2*. Following this we secure the input to function f, by copying it from register X_1 to register X_r (*Command 3*). Then we copy the value of register X_{r+1} to register X_2 (*Command 4*), thus we have x and y on register X_1 and X_2 respectively.
Important: I assume that P_g cleans its registers after computation, i.e. $\forall i \in [2, r-1] : X_i = 0$, otherwise we can simply run $\forall i \in [2, r-1]$

1: if t_i then 0 else 2
2: S_i then 1

before we execute P_g.
After, we have executed P_g (*Command 5*), we test whether it returned 0, i.e. $X_1 = 0$ (*Command 6*). If $X_1 = 0$ we return y, i.e. we copy the value of X_{r+1} to X_1 (*Command 9*). If $X_1 \neq 0$, however, we increment X_{r+1} (*Command 7*), reset the input, i.e. copy the value of X_r to X_1 (*Command 8*), and restart the process from *Command 4*. Hence, if there is a y such that P_g returns 0, it will be found by P_f. However, if it does not exist, P_f will diverge.
Hence, we have constructed a register program, which simulates minimisation. Therefore, it is shown that applying the minimisation function to a register computable function, preserves register computability. Moreover, since we have assumed that g is register computable, we can conclude $f(x)$ is also register computable.

Exercise 6

Proof the following statement:

For all k, x we have

a) $f_i(x) \geq x + 1$ for all i;

b) $f_i^k(x)$ is increasing in i, k, x;

c) $2f_i^k(x) \leq f_i^{k+1}(x)$ for $i \geq 1$;

d) $f_i^k(x) + x \leq f_i^{k+1}(x)$ for $i \geq 1$;

Exercise 6a

We solve this problem by induction over i.

IB: We set $i = 0$, thus

$$f_0(x) = x + 2 \geq x + 1 \quad \text{for all } i > 1$$

In the case of $x = 0$ and $x = 1$ we have

$$f_0(0) = 1 \geq 0 + 1$$
$$f_0(1) = 2 \geq 1 + 1$$

IH: $\forall i : f_i(x) \geq x + 1$

IS: Now we perform the inductive step:

$$\begin{aligned} f_{i+1}(x) &= f_i^x(1) \\ f_i^x(1) &= f_i(f_i^{x-1}(1)) \geq f_i^{x-1}(1) + 1 \\ f_i^{x-1}(1) + 1 &= f_i(f_i^{x-2}(1)) + 1 \geq f_i^{x-2}(1) + 2 \\ &\vdots \\ f_i^{x-(x-1)}(1) + (x-1) &= f_i(1) + (x-1) \geq 1 + 1 + (x-1) = x + 1 \end{aligned}$$

Hence, we can conclude that

$$f_{i+1}(x) \geq x + 1$$

Thus our claim holds.

Exercise 6b

We solve this problem by induction over i, k and x.
Before, we start I need to clarify that I assume that $f_i^0(x) = x$, since $f_i^k(x) = f_i(f_i(\dots f_i(x)\dots))$ with k-times nesting, which is simply $f_i(f_i(\dots f_i(x)\dots)) = (f_i \circ f_i \circ \cdots \circ f_i)(x)$ k-times. Hence, if $k = 0$ it should be $()(x)$. Therefore, I am not entirely sure, whether this is x or something else. However, intuitively the prior makes sense (and it seems to be consistent).

IB: Firstly, we present some base cases (I think the first base case would suffice, but I am including the other just to be save):
For $i = 0$, $k = 0$ and $x = 0$ we obtain

$$\begin{aligned} f_i^k(x) = f_0^0(0) = 0 \leq f_{i+1}^k(x) &= f_1^0(0) = 0 \\ &\leq f_i^{k+1}(x) = f_0^1(0) = 1 \\ &\leq f_i^k(x+1) = f_0^0(1) = 1 \end{aligned}$$

For $i = 1$, $k = 0$ and $x = 0$ we obtain

$$\begin{aligned} f_i^k(x) = f_1^0(0) = 0 \leq f_{i+1}^k(x) &= f_2^0(0) = 0 \\ &\leq f_i^{k+1}(x) = f_1^1(0) = f_0^0(1) = 1 \\ &\leq f_i^k(x+1) = f_1^0(1) = 1 \end{aligned}$$

For $i = 0$, $k = 1$ and $x = 0$ we obtain

$$\begin{aligned} f_i^k(x) = f_0^1(0) = 1 \leq f_{i+1}^k(x) &= f_1^1(0) = f_0^0(1) = 1 \\ &\leq f_i^{k+1}(x) = f_1^2(0) = f_1(f_0^0(1)) = f_1(1) = f_0^1(1) = 2 \\ &\leq f_i^k(x+1) = f_1^1(1) = f_0^1(1) = 2 \end{aligned}$$

For $i = 0$, $k = 0$ and $x = 1$ we obtain

$$\begin{aligned} f_i^k(x) = f_0^0(1) = 1 \leq f_{i+1}^k(x) &= f_1^0(1) = 1 \\ &\leq f_i^{k+1}(x) = f_1^1(1) = f_0^1(1)) = 2 \\ &\leq f_i^k(x+1) = f_1^0(2) = 2 \end{aligned}$$

For $i = 1$, $k = 1$ and $x = 1$ we obtain

$$\begin{aligned} f_i^k(x) = f_1^1(1) = 2 \leq f_{i+1}^k(x) &= f_2^1(1) = f_1^1(1) = f_0^1(1) = 2 \\ &\leq f_i^{k+1}(x) = f_1^2(1) = f_1(f_0^0(1)) = f_1(1) = f_0^1(1) = 2 \\ &\leq f_i^k(x+1) = f_1^1(2) = f_0^2(1) = f_0(2) = 2+2 = 4 \end{aligned}$$

IH: Now that we have established the base cases we formulate our three induction hypotheses:

For all i, k, x:

$$\begin{aligned} f_i^k(x) &\leq f_{i+1}^k(x) \\ f_i^k(x) &\leq f_i^{k+1}(x) \\ f_i^k(x) &\leq f_i^k(x+1) \end{aligned}$$

IS(i): We increment i and use $f_i^k(x) \leq f_{i+1}^k(x)$:

$$\begin{aligned} f_{i+1}^k(x) = & f_{i+1}(f_{i+1}^{k-1}(x)) \\ & f_{i+1}(f_{i+1}^{k-1}(x)) = f_i^{f_{i+1}^{k-1}(x)}(1) \leq f_{i+1}^{f_{i+1}^{k-1}(x)}(1) = f_{i+2}(f_{i+1}^{k-1}(x)) \\ & f_{i+2}(f_{i+1}^{k-1}(x)) = f_{i+2}(f_{i+1}(f_{i+1}^{k-2}(x))) = f_{i+2}(f_i^{f_{i+1}^{k-2}(x)}(1)) \leq f_{i+2}(f_{i+1}^{f_{i+1}^{k-2}(x)}(1)) = f_{i+2}^2(f_{i+1}^{k-2}(x)) \\ & \quad \vdots \\ & f_{i+2}^{k-(k-1)}(f_{i+1}^{k-(k-1)}(x)) = f_{i+2}(f_{i+1}(x)) = f_{i+2}(f_i^x(1)) \leq f_{i+2}(f_{i+1}^x(1)) = f_{i+2}^{(k-1)}(f_{i+2}(x)) = f_{i+2}^k(x) \end{aligned}$$

(Here I implicitly used the fact that $f_i^k(x) \leq f_i^k(y)$ for $x \leq y$. Unfortunately, I recognised this before handing the HW in s.t. I did not had time to account for this.)
Hence, we have

$$f_{i+1}^k(x) \leq f_{i+2}^k(x)$$

IS(k): We increment k and use $f_i^k(x) \leq f_i^{k+1}(x)$:

$$f_i^{k+1}(x) = f_i^k(f_i(x)) \leq f_i^{k+1}(f_i(x)) = f_i^{k+2}(x)$$

Hence, we have

$$f_i^{k+1}(x) \leq f_i^{k+2}(x)$$

IS(x): We increment x and use $f_i^k(x) \leq f_i^k(x+1)$:
In order to present the idea we first set $k = 1$.

$$f_i(x+1) = f_{i-1}^{x+1}(1) = f_{i-1}(f_{i-1}^x(1)) \leq f_{i-1}(f_{i-1}(f_{i-1}^x(1))) = f_{i-1}^{x+2}(1) = f_i(x+2)$$

Here, we use $f_i^{k+1}(x) \leq f_i^{k+2}(x)$. Unfortunately, I was not able to proof this for any k as my initial argument (I found this mistake close to the deadline)

$$f_i^k(x+1) = f_i^{k-1}(f_i(x+1)) \leq f_i^{k-1}(f_{i-1}^{x+2}(1)) = f_i^k(x+2)$$

(Using the case of $k = 1$)
k Is not a valid argument, as our induction hypothesis only holds until x and not for f_{i-1}^x. That is, we want to prove that a greater argument results in a greater output, thus we cant use it as an argument.

Exercise 6c

We solve this problem by induction over i.

IB: We set $i = 1$, thus

$$2 \cdot f_1^k(x) = 2 \cdot 2^k x \leq 2^{k+1} x = f_1^{k+1}(x)$$

IH: $\forall i \geq 1 : 2 \cdot f_i^k(x) \leq f_i^{k+1}(x)$

IS: Now we perform the inductive step:

$$\begin{aligned} 2 \cdot f_{i+1}^k(x) &= 2 \cdot f_{i+1}(f_{i+1}^{k-1}(x)) = 2 \cdot f_i^{f_{i+1}^{k-1}(x)}(1) \\ 2 \cdot f_i^{f_{i+1}^{k-1}(x)}(1) &\leq f_i^{f_{i+1}^{k-1}(x)+1}(1) = f_i(f_i^{f_{i+1}^{k-1}(x)}(1)) \\ f_i(f_i^{f_{i+1}^{k-1}(x)}(1)) &= f_i(f_{i+1}^k(x)) \leq f_{i+1}(f_{i+1}^k(x)) = f_{i+1}^{k+1}(x) \end{aligned}$$

(The last inequality holds to to the fact that f is increasing)
Hence, we can conclude that

$$2 \cdot f_{i+1}^k(x) \leq f_{i+1}^{k+1}(x)$$

Thus our claim holds.

Exercise 6d

We solve this problem by induction over i.

IB: We set $i = 1$, thus

$$f_1^k(x) + x = 2^k x + x \leq 2^k x + 2^k x = 2 \cdot 2^k x = 2^{k+1} x = f_1^{k+1}(x)$$

IH: $\forall i \geq 1 : f_i^k(x) + x \leq f_i^{k+1}(x)$

IS: Now we perform the inductive step:

$$\begin{aligned} f_{i+1}^k(x) + x =& f_{i+1}(f_{i+1}^{k-1}(x)) + x = f_i^{f_{i+1}^{k-1}(x)}(1) + x \\ & f_i^{f_{i+1}^{k-1}(x)}(1) + x \leq f_i^{f_{i+1}^{k-1}(x)+1}(1) = f_i(f_i^{f_{i+1}^{k-1}(x)}(1)) \\ & f_i(f_i^{f_{i+1}^{k-1}(x)}(1)) = f_i(f_{i+1}^k(x)) \leq f_{i+1}(f_{i+1}^k(x)) = f_{i+1}^{k+1}(x) \end{aligned}$$

(The last inequality holds to to the fact that f is increasing)
Hence, we can conclude that

$$f_{i+1}^k(x) + x \leq f_{i+1}^{k+1}(x)$$

Thus our claim holds.