Assignment 2

Nathan Ellis, Rishad Mahbub

May 2, 2018

1 Task 1

Define a function c: $\mathbb{R} \longrightarrow \mathbb{R}$ as

$$c(n) = \lfloor \log_{10} n \rfloor$$

Define the set of all primes $\mathbb{P}$ as

$$\mathbb{P} = \{\ x \in \mathbb{N} \mid \forall (1 < i < x)\ .\ x mod(i) \neq 0\ \}$$

Define the set of all emirps $\mathbb{E}$ as

$$\mathbb{E} = \{\ x \in \mathbb{N} \mid x = \sum_{i=0}^{c(x)} S_i 10^{c(x)-i} \wedge r = \sum_{i=0}^{c(x)} S_i 10^i\ \wedge x \in \mathbb{P} \wedge r \in \mathbb{P} \wedge x \neq r\ \}$$

where S is some constant.

Define a function f: $\mathbb{N} \longrightarrow \mathbb{N}$ as

$$f(y) = \begin{cases} 0 & \textbf{if } y \notin \mathbb{E} \\ 1 & \textbf{if } y \in \mathbb{E} \end{cases}$$

We specify our function as required by the assignment 2 specification as

$$\llcorner n : \left[\ \text{emirpNumber} > 0,\ \ f(n) = 1 \wedge \text{emirpNumber} = \textstyle\sum_{k=0}^{n} f(k)\ \right] \lrcorner_{(1)}$$

As the precondition states, the user input number called *emirpNumber* is greater than 0. The program specification then states that n is changed such that the number of emirps between 0 and n inclusive is equal to *emirpNumber* and n is itself an emirp.

2 Task 2

We will use two functions and so derive both using refinement calculus.

2.1 Emirp Derivation

We start with a spec of the procedure EMIRP.

$$\textbf{proc}\ \textsc{emirp}(tobeadded)\ \cdot$$

$$n : \left[\ \text{emirpNumber} > 0,\ \textstyle\sum_{k=1}^{n} f(k) = \text{emirpNumber} \wedge f(n) = 1\ \right]$$

$$\sqsubseteq \quad \langle \textbf{i-loc} \rangle$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\ \text{emirpNumber} > 0,\ \textstyle\sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = 1\ \right] ; \lrcorner(1)$$

$$(1) \sqsubseteq \quad \langle \textbf{seq - see justification below} \rangle$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\ \text{emirpNumber} > 0,\ \textstyle\sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m\ \right] ; \lrcorner(2)$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \\ \wedge\ \text{emirpsCounted} = \text{emirpNumber} \wedge f(n) = 1 \end{array} \right] ; \lrcorner(3)$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \\ \wedge\ \text{emirpsCounted} = \text{emirpNumber} \wedge f(n) = 1 = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpNumber} \wedge f(n) = 1 = m \end{array} \right] \lrcorner(4)$$

$$(2) \sqsubseteq \quad \langle \textbf{ass - see justification below} \rangle$$

$$n := 0;\ \text{emirpsCounted} := 0;\ m := 0$$

$$(3) \sqsubseteq \quad \langle \textbf{while} \rangle$$

$$\textbf{while}\ (\text{emirpsCounted} \neq \text{emirpNumber} \vee f(n) \neq 1)\ \textbf{do}$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \\ \wedge(\text{emirpsCounted} \neq \text{emirpNumber} \vee f(n) \neq 1), \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right] \lrcorner(5)$$

$$\textbf{od}$$

$$(5) \sqsubseteq \quad \langle \textbf{seq2} \rangle$$

$$\llcorner n : \left[\begin{array}{l} \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m, \\ \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n-1) = m \end{array} \right] ; \lrcorner(6)$$

$$\llcorner n, m, \text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n-1) = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right] \lrcorner(7)$$

$$(6) \sqsubseteq \quad \langle \textbf{ass} \rangle$$

$$n := n + 1;$$

$$(7) \sqsubseteq \quad \langle \textbf{seq2} \rangle$$

$$\llcorner m : \left[\begin{array}{l} \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n-1) = m, \\ \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right] ; \lrcorner(8)$$

$\llcorner n, m, \text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n) = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right]\lrcorner_{(9)}$

(8) $\sqsubseteq$ $\langle$**ass**$\rangle$

$m := \text{ISEMIRP}(n);$

(9) $\sqsubseteq$ $\langle$**if**$\rangle$

if m=1 **then**

$\text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n) = 1 = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right]$

$\sqsubseteq$ $\langle$**ass**$\rangle$

$\text{emirpsCounted} := \text{emirpsCounted} + 1;$

else

$\text{emirpsCounted} : \left[\begin{array}{l} \sum_{k=1}^{n-1} f(k) = \text{emirpsCounted} \wedge f(n) = 0 = m, \\ \sum_{k=1}^{n} f(k) = \text{emirpsCounted} \wedge f(n) = m \end{array} \right]$

$\sqsubseteq$ $\langle$**ass**$\rangle$

skip;

fi

2.2 IsEmirp Derivation

For this program specification of ISEMIRP, we use our definition

$$\mathbb{E} = \{\ x \in \mathbb{N} \mid x = \sum_{i=0}^{c(x)} S_i 10^{c(x)-i} \wedge r = \sum_{i=0}^{c(x)} S_i 10^{i}\ \wedge x \in \mathbb{P} \wedge r \in \mathbb{P} \wedge x \neq r\ \}$$

where S is some constant.

We will also use our definition of function f:

$$f(x) = \begin{cases} 0 & \textbf{if } y \notin \mathbb{E} \\ 1 & \textbf{if } y \in \mathbb{E} \end{cases}$$

We start with a spec of the procedure ISEMIRP.

proc $\text{ISEMIRP}(\textbf{result}\ x : \mathbb{N}, \textbf{value}\ y : \mathbb{N}) \cdot$

$\llcorner x : [\ y > 0,\ x = f(y)\ ]\lrcorner_{(1)}$

(1) $\sqsubseteq$ $\langle$**i-loc**$\rangle$

$\llcorner \textbf{var}\ r\ \cdot\ x, r : [\ y > 0,\ x = f(y)\ ]\lrcorner_{(2)}$

(2) $\sqsubseteq$ $\langle$**seq2**$\rangle$

$\llcorner r : [\ y > 0,\ y > 0 \wedge r = 0\ ]; \lrcorner_{(3)}$

$\llcorner x, r : [\ y > 0 \wedge r = 0,\ x = f(y)\ ]_{\lrcorner 4)}$

(3) $\sqsubseteq$ $\langle$**ass**$\rangle$

$r := 0$

(4) $\sqsubseteq$ $\langle$**i-con**$\rangle$

con $S : [10]^* \cdot\ x, r : \left[\ y > 0 \wedge r = 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i},\ x = f(y)\ \right]$

$\sqsubseteq$ $\langle$**seq2**$\rangle$

$\llcorner$**con** $S : [10]^* \cdot\ r : \left[\begin{array}{l} y > 0 \wedge r = 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i}, \\ \\ y > 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \end{array}\right];_{\lrcorner(5)}$

$\llcorner$**con** $S : [10]^* \cdot\ x, r : \left[\ y > 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i,\ x = f(y)\ \right];_{\lrcorner(6)}$

(5) $\sqsubseteq$ $\langle$**proc - see justification below**$\rangle$

reversen(y, r)

(6) $\sqsubseteq$ $\langle$**i-loc**$\rangle$

var $e : \mathbb{B}\ .\ x, r, e : \left[\ y > 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i, x = f(y)\ \right]$

$\sqsubseteq$ $\langle$**c-frame r nor $\mathbf{r_0}$ in post**$\rangle$

$x, e : \left[\ y > 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i,\ x = f(y)\ \right]$

$\sqsubseteq$ $\langle$**seq and c-frame on (8) where e nor $\mathbf{e_0}$ in post**$\rangle$

$\llcorner e : \left[\begin{array}{l} y > 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i, \\ y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \end{array}\right];_{\lrcorner(7)}$

$\llcorner x : \left[\begin{array}{l} y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y), \\ x = f(y) \end{array}\right]_{\lrcorner(8)}$

(7) $\sqsubseteq$ $\langle$**proc - see justification below**$\rangle$

$e :=$ mpz_probab_prime_p$(y) \wedge$ mpz_probab_prime_p$(r) \wedge$ mpz_cmp(y, r)

(8) $\sqsubseteq$ $\langle$**if**$\rangle$

if e **then**

$\llcorner x : \left[\begin{array}{l} e \Leftrightarrow \text{TRUE} \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i, \\ x = f(y) \end{array}\right]_{\lrcorner(9)}$

else

$\llcorner x : \left[\begin{array}{l} e \Leftrightarrow \text{FALSE} \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i, \\ x = f(y) \end{array}\right]_{\lrcorner(10)}$

(9) $\sqsubseteq$ $\langle$**ass - see justfication below**$\rangle$

$x := 1$

(10) $\sqsubseteq$ $\langle$**ass - see justfication below**$\rangle$

$x := 0$

There are still some outstanding proofs in the above derivation. We will discharge them below in the next subsection.

2.3 Proof of $(5) \sqsubseteq$ reversen(y,r)

We use the definition of REVERSEN from the Assignment 2 Specification which reads:

$$\textbf{proc } \text{REVERSEN}(\textbf{value } n : \mathbb{N}, \textbf{result } r : \mathbb{N}) \cdot$$
$$\textbf{con } S : [10]^* \;.\; r : \left[\; n = \textstyle\sum_{i=0}^{c(n)} (S_i 10^{(c(n)-i)}) \wedge n > 0, r = \sum_{i=0}^{c(n)} (S_i 10^i) \;\right]$$

To prove this function's pre-condition and post-condition are identical to pre(5) and post(5) we we must follow the w-pre and s-post rules as described in the COMP2111 glossary. Upon insepction, we must adjust our function call variables to suit those of our specification (let n in the above become y). We begin the proof by weakening our pre(5), in which we must prove:

$$pre(5) \Rightarrow pre(\text{REVERSEN}(y, r))$$

so we do so

$$\begin{aligned}
& pre(5) \\
\Leftrightarrow \quad & \langle \text{Substitution} \rangle \\
& y > 0 \wedge r = 0 \wedge y = \sum_{i=0}^{c(y)} (S_i 10^{(c(y)-i)}) \\
\Rightarrow \quad & \langle \text{Treating B as the 2nd conjunct, we have A} \wedge \text{B} \wedge \text{C} \Rightarrow \text{A} \wedge \text{B} \rangle \\
& y = \sum_{i=0}^{c(y)} (S_i 10^{(c(y)-i)}) \wedge y > 0 \\
\Leftrightarrow \quad & \langle \text{Substitution and replacing } n \text{ with } y \rangle \\
& pre(\text{REVERSEN}(y, r))
\end{aligned}$$

We then have to strengthen our post(5), in which we must prove:

$$pre(5)[^{r_0}/_r] \wedge post(\text{REVERSEN}(y, r)) \Rightarrow post(\text{REVERSEN}(y, r))$$

so we do so

$$\begin{aligned}
& pre(5)[^{r_0}/_r] \wedge post(\text{REVERSEN}(y, r)) \\
\Leftrightarrow \quad & \langle \text{Substitution} \rangle \\
& y > 0 \wedge r_0 = 0 \wedge y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{c(n)} (S_i 10^i) \\
\Rightarrow \quad & \langle \text{Treating B as the final conjunct, we have A}
\end{aligned}$$

$$r = \sum_{i=0}^{c(n)} (S_i 10^i)$$

$$\Leftrightarrow \qquad \langle\text{Substitution}\rangle$$

$$post(\text{REVERSEN}(y, r))$$

Therefore, by w-pre and s-post, we have proven the function call to be a correct refinement of our specification's pre-condition and post-condition.

2.4 Proof of (7) $\sqsubseteq$ mpz-probab-prime-p(y); mpz-probab-prime-p(r); mpz-cmp(y,r)

We use the definition of MPZ-PROBAB-PRIME-P and MPZ-CMP from the Assignment 2 Specification which read:

$$\textbf{proc } \text{MPZ-PROBAB-PRIME-P}(\textbf{value } x : \mathbb{N}, \textbf{result } y : \mathbb{B}) \cdot$$
$$y : \left[\text{ TRUE}, x \Leftrightarrow (y \in \mathbb{P}) \right]$$

$$\textbf{proc } \text{MPZ-CMP}(\textbf{value } y : \mathbb{Z}, \textbf{value } z : \mathbb{Z}, \textbf{result } x : \mathbb{B}) \cdot$$
$$y : \left[\text{ TRUE}, x \Leftrightarrow (y = z) \right]$$

We need to strengthen our post(7) and we do so as follows

$$post(7) \Leftrightarrow \qquad \langle\text{Substitution}\rangle$$

$$y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \Rightarrow \qquad \langle\text{Expanding the final}$$

$$y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow (y \in \mathbb{P}) \wedge e \Leftrightarrow (r \in \mathbb{P}) \wedge e \Leftrightarrow (r \neq y)$$

We then need to weaken our pre(7) and we do so as follows

$$pre(7) \Leftrightarrow \qquad \langle\text{Substitution}\rangle$$

$$y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \Rightarrow \qquad \langle\text{Treating the previous statement as A in A}\wedge\text{TRUE} \Rightarrow \text{TRUE}\rangle$$

$$\text{TRUE}$$

By s-post and w-pre rules defined by the COMP2111 Glossary, we now have a pre-condition and post-condition

$$e : \left[\text{TRUE}, \; y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow (y \in \mathbb{P}) \wedge e \Leftrightarrow (r \in \mathbb{P}) \wedge e \Leftrightarrow (r \neq y) \right]$$

The post-condition can clearly be implied from the pre-condition, especially noting that the final three conjuncts are evidently the same as the definitions (above), with variable e simply replacing x.

2.5 Proof of $(9) \sqsubseteq x := 1$

We need to prove the validity of

$$pre(9) \Rightarrow (post(9))[^1/_x]$$

i.e., the prerequisite of the relevant instance of **ass**. Expanding the definitions and performing the substitution yields

$$\left(y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow \mathrm{TRUE} \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \right) \Rightarrow$$
$$f(y) = 1 \enspace .$$

The items in *blue* and *red* form the first two conjuncts of our definition of emirps $\mathbb{E}$ (see Task 1). The equivalence in *green* shows that $y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge x \neq r$ are all evaluated true in our pre-condition. These three conditions form the final three conjuncts from our definition of $\mathbb{E}$. Therefore, the variable x is contained within the set $\mathbb{E}$. We then notice that because $x \in \mathbb{E}$ and by our definition of $f(y)$ (see Task 1), $f(y) = 1$ as required.

2.6 Proof of $(10) \sqsubseteq x := 0$

We need to prove the validity of

$$pre(10) \Rightarrow (post(10))[^0/_x]$$

i.e., the prerequisite of the relevant instance of **ass**. Expanding the definitions and performing the substitution yields

$$\left(y = \sum_{i=0}^{C(y)} S_i 10^{C(y)-i} \wedge r = \sum_{i=0}^{C(y)} S_i 10^i \wedge e \Leftrightarrow \mathrm{FALSE} \Leftrightarrow (y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge r \neq y) \right) \Rightarrow$$
$$f(y) = 0 \enspace .$$

The items in *blue* and *red* form the first two conjuncts of our definition of emirps $\mathbb{E}$ (see Task 1). The equivalence in *green* shows that $y \in \mathbb{P} \wedge r \in \mathbb{P} \wedge x \neq r$ are collectively evaluated false in our pre-condition. This means one of these conditions is false. Looking at our definition of $\mathbb{E}$, the variable x is not contained within the set $\mathbb{E}$ for this reason. We then notice that because $x \notin \mathbb{E}$ and by our definition of $f(y)$ (see Task 1), $f(y) = 0$ as required.

3 Task 3

```c
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include <reverse.h>

int is_emirp(mpz_t z);
void emirp(mpz_t n, unsigned long emirpNumber);

int is_emirp(mpz_t y)
{
	mpz_t r;		// Declares a gmp number called r
	mpz_init(r);		// A gmp function call to initilize r and set it to 0
	reversen(y, r);		// Function call to set the value or r to the reverse value of n
	if ((mpz_cmp(y, r)) &&		// Check for emirp by emirp definition
		(mpz_probab_prime_p(y, 5)) &&
		(mpz_probab_prime_p(r, 5))) {
		return 1;
	}
	return 0;
}

void emirp(mpz_t n, unsigned long emirpNumber)
{
	// Set up the loop invariant
	mpz_init(n);		// A gmp function call to initialize n and set it to 0
	int emirpsCounted = 0;		// The number of emirps the program has counted so far
	int m = is_emirp(n);		// This is required for the loop invariant

	// Run the loop
	while (emirpsCounted != emirpNumber || m != 1) {
		mpz_add_ui(n, n, 1);		// Add unsigned long value '1' to mpz_t 'n'
		m = is_emirp(n);
		if (m == 1) {
			emirpsCounted += 1;
		}
	}
}

int main(void)
{
	int emirpNumber = 0;		// Get user input to get what number emirp is desired
```

```
    printf("Enter a number: ");
    if (!scanf("%d", &emirpNumber)) {
        printf("Invalid input\n");
        exit(1);
    }

    mpz_t n;        // Declares a gmp number called n
    mpz_init(n);        // A gmp function call to initialize n and set it to 0
    emirp(n, emirpNumber);      // Set n to the value of the emirpNumber'th emirp
    gmp_printf("%Zd\n", n);      // Prints the emirpNumber'th emirp

    return 0;
}
```

In the C code