Assignment 2

z5019338
z5131048

March 3, 2019

1 Introduction

Given a positive integer number n, return the n'th emirp. An emirp is a prime whose (base 10) reversal is also prime, but which is not a palindromic prime.
Firstly, let the set of all prime numbers be as *prime*, where *prime* denote as:

$$prime = \{p \in \mathbb{N}^+ \mid p > 1 \wedge \neg\exists k \in \mathbb{N}^+.\{1, n\} : \frac{p}{k} \in \mathbb{N}^+\}$$

Let the sequence of all the prime numbers be as P_n, where P_n is denoted as

$$P_n = \langle 2, 3, 5, 7, 11, ... \rangle_{n \in \mathbb{N}^+}$$

Given proc *reversen* as defined in the specification, it is going to be denoted as the mathematical function $res : \mathbb{N} \to \mathbb{N}$ as following:

$$res(n) = \sum_{i=0}^{\lfloor \log n \rfloor} S_i 10^i$$

where $n = \sum_{i=0}^{\lfloor \log n \rfloor} S_i 10^{\lfloor \log n \rfloor - i}$

2 The Derivation

Before deriving **proc** $\textsc{emirp}$(**value** $n : \mathbb{N}^+$, **result** $r : \mathbb{N}^+$), it needs to develop 2 procedure that help with proc $\textsc{emirp}$
proc $\textsc{isPrime}$(**value** $n : \mathbb{N}^+$, **result** $b : \mathbb{B}$)
proc $\textsc{nextPrime}$(**value** $p : \mathbb{N}^+$, **result** $q : \mathbb{N}^+$).

$$\textbf{proc } \textsc{nextPrime}(\textbf{value } p, \textbf{result } q) \cdot$$
$$\llcorner p, q : \left[\ p \in P_i, i := i_0 + 1 \wedge q := p_i\ \right] \lrcorner_{(A)}$$

$(A) \sqsubseteq \quad \langle \textbf{i-con} \rangle$

$\llcorner var\ con\ b \cdot p, q : [\ p \in P_i \wedge i = b, q := p_i \wedge i := i_0 + 1\ ]\lrcorner_{(B)}$

$(B) \sqsubseteq \quad \langle \textbf{s-post} \rangle$

$p, q : [\ p \in P_i \wedge i = b, q := P_i \wedge i = b + 1\ ]$

$\sqsubseteq \quad \langle \textbf{seq2} \rangle$

$i := i + 1;\ q := P_i$

$\textbf{proc}\ \textsc{isPrime}(\textbf{value}\ n, \textbf{result}\ r) \cdot$

$\llcorner n, r : [\ n > 0, b \Leftrightarrow n \in prime\ ]\lrcorner_{(I)}$

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

$\llcorner var\ k \cdot n, b, k : [\ n > 0, b \Leftrightarrow n \in prime\ ]\lrcorner_{(II)}$

$(II) \sqsubseteq \quad \langle \textbf{c-frame,seq}$, split into initialisation and loop$\rangle$

$\llcorner b : [\ n > 0, Inv\ ]\lrcorner_{(III)}$

$\llcorner n, b, k : [\ Inv, b \Leftrightarrow n \in prime\ ]\lrcorner_{(IV)}$

Where the loop invariant is defined as:

$$Inv = \left(\begin{array}{l} (n = 1 \Rightarrow b = False) \vee \\ 2 < k \leq n \wedge (\neg \exists k \in 2..(n-1)\ (p \mid k) \Leftrightarrow n \in prime \wedge b = True) \\ \vee (\exists k \in 2..(n-1)\ (p \mid k) \Leftrightarrow n \notin prime \wedge b = False) \end{array} \right)$$

$(III) \sqsubseteq \quad \langle \textbf{seq,ass} \rangle$

$\llcorner b, k : [\ n > 0, Inv[^{2,True}/_{k,b}]\ ]\lrcorner_{\spadesuit}$

$\textbf{if}\ n = 1\ \textbf{then}$

$\quad \llcorner n, b, k : [\ n > 0 \wedge n = 1, Inv[^{1,False}/_{n,b}]\ ]\lrcorner_{\heartsuit}$

$\textbf{else};$

$\quad \llcorner n, b, k : [\ n > 0 \wedge n \neq 1, Inv[^{1,False}/_{n,b}]\ ]\lrcorner_{\diamondsuit}$

$\textbf{fi}$

$\spadesuit \sqsubseteq \quad \langle \textbf{ass}$, justified below in Sect. 2.1$\rangle$

$b := True;$

$k := 2;$

$\heartsuit \sqsubseteq \quad \langle \textbf{ass}$, justified below in Sect. 2.2$\rangle$

$b := False$

$\diamondsuit \sqsubseteq \quad \langle \textbf{skip} \rangle$

$skip$

$$
\begin{array}{rl}
(IV) \sqsubseteq & \langle \textbf{s-post}, \text{justified below in Sect. 2.3} \rangle \\
& n, b, k : [\ Inv, Inv \wedge k = n\ ] \\
\sqsubseteq & \langle \textbf{c-frame,while} \rangle \\
& \textbf{while } k < n \textbf{ do} \\
& \quad \llcorner n, b, k : \left[\ Inv \wedge k < n, Inv[^{k+1}/_{k}]\ \right] \lrcorner_{(V)} \\
& \textbf{od}; \\
(V) \sqsubseteq & \langle \textbf{seq,c-frame} \rangle \\
& \llcorner n, b, k : \left[\ Inv \wedge k < n, Inv[^{k+1}/_{k}]\ \right] \lrcorner_{(VI)} \\
& \llcorner k : \left[\ Inv \wedge k < n, Inv[^{k+1}/_{k}]\ \right] \lrcorner_{(VII)} \\
(VI) \sqsubseteq & \langle \textbf{if} \rangle \\
& \textbf{if } n|k \textbf{ then} \\
& \quad \llcorner n, b, k : \left[\ n|k \wedge Inv \wedge k < n, Inv[^{k+1}/_{k}]\ \right] \lrcorner_{(VIII)} \\
& \textbf{else}; \\
& \quad \llcorner n, k : \left[\ n \nmid k \wedge Inv \wedge k < n, Inv[^{k+1}/_{k}]\ \right] \lrcorner_{(IX)} \\
& \textbf{fi} \\
(VIII) \sqsubseteq & \langle \textbf{ass} \rangle \\
& b := False; \\
(IX) \sqsubseteq & \langle \textbf{skip} \rangle \\
& skip; \\
(VII) \sqsubseteq & \langle \textbf{ass} \rangle \\
& k := k + 1;
\end{array}
$$

2.1 Proof of $\spadesuit \sqsubseteq b := True; k := 2;$

We need to prove validity

$$n > 0 \Rightarrow Inv[^{2, True}/_{k,b}]$$

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

$$
n > 0 \Rightarrow \\
\left(\begin{array}{l}
(n = 1 \Rightarrow b = False) \vee \\
2 < 2 \leq n \wedge (\neg \exists 2 \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge True = True) \\
\vee (\exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \notin prime \wedge b = False)
\end{array} \right)
$$

Clearly, we have established validity of the second conjunct of the RHS. It's obviously true. As long as any disjunct is true, the rest is true

2.2 Proof of $\heartsuit \sqsubseteq b := False;$

We need to prove validity

$$n > 0 \wedge n = 1 \Rightarrow Inv[^{1,False}/_{n,b}]$$

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

$$n > 0 \wedge n = 1 \Rightarrow$$
$$\left(\begin{array}{l} (n = 1 \Rightarrow False = False) \vee \\ 2 < k \leq n \wedge (\neg \exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge b = True) \\ \vee (\exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \notin prime \wedge b = False) \end{array}\right)$$

Clearly, we have established validity of the first conjunct of the RHS. It's obviously true. As long as any disjunct is true, the rest is true

2.3 Proof of $(IV) \sqsubseteq Inv \Rightarrow Inv \wedge k = n$

Expanding the definitions and performing the substitution yields

$$\left(\begin{array}{l} (n = 1 \Rightarrow b = False) \vee \\ 2 < k \leq n \wedge (\neg \exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge b = True) \\ \vee (\exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \notin prime \wedge b = False) \end{array}\right) \Rightarrow$$
$$\left(\begin{array}{l} (n = 1 \Rightarrow b = False) \vee \\ 2 < k \leq n \wedge (\neg \exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge b = True) \\ \vee (\exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \notin prime \wedge b = False) \wedge k = n \end{array}\right)$$

According there is 3 disconjuction of the LHS, there are 3 cases to consider:

1. $n > 0 \wedge n = 1 \Rightarrow Inv[^{1,2,False}/_{n,2,b}]$: as prove above, the fist disjunct of the RHS follows immediately.

2. $2 \leq k \leq n-1 \wedge p \nmid k \Rightarrow Inv[^{k+1,True}/_{k,b}]$: we prove the second disjunct of the RHS. As $2 \leq k \leq n-1 \wedge p \nmid k \Rightarrow \neg \exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge b = True$

3. $2 \leq k \leq n-1 \wedge p \mid k \Rightarrow Inv[^{k+1,False}/_{k}]$: similarly, $2 \leq k \leq n-1 \wedge p \mid k \Rightarrow \exists k \in 2..(n-1)\,(p \mid k) \Leftrightarrow n \in prime \wedge b = False$

Clearly, we have established validity of the RHS. It's obviously true.

We gather the code for the procedure body of isPrime:

$$\begin{array}{l} b := True; \\ \textbf{if } n = 1 \textbf{ then} \\ \quad b := False; \\ \textbf{else} \end{array}$$

$$
\begin{array}{l}
\quad skip \\
\textbf{fi} \\
k := 2 \\
\textbf{while } k < n \textbf{ do} \\
\quad \textbf{if } n = 1 \textbf{ then} \\
\quad\quad b := False; \\
\quad \textbf{else} \\
\quad\quad skip \\
\quad \textbf{fi} \\
\textbf{od};
\end{array}
$$

$$
\begin{array}{ll}
& \textbf{proc } \text{EMIRP}(\textbf{value } n, \textbf{result } r) \cdot \\
& \quad \llcorner n, r : \left[\begin{array}{l} n > 0, \\ n = 0 \wedge r \in prime \wedge rev(r) \in prime \wedge r \neq rev(r) \end{array} \right] \lrcorner_{(1)} \\
(1) \sqsubseteq & \quad \langle \textbf{i-loc} \rangle \\
& \llcorner var\ s \cdot n, r, s : \left[\begin{array}{l} n > 0, \\ n = 0 \wedge r \in prime \wedge rev(r) \in prime \wedge r \neq rev(r) \end{array} \right] \lrcorner_{(2)} \\
(2) \sqsubseteq & \quad \langle \textbf{seq}, \text{ split into initialisation and loop} \rangle \\
& \llcorner s : [\ n > 0, Inv\ ] \lrcorner_{(3)} \\
& \llcorner n, r, s : [\ Inv, n = 0 \wedge r \in prime \wedge rev(r) \in prime \wedge r \neq rev(r)\ ] \lrcorner_{(4)}
\end{array}
$$

Where the loop invariant is defined as:

$$
Inv = (\ 0 \leq i \leq n \wedge s = P_i \wedge rev(s) \in prime \wedge s \neq rev(s) \wedge r = P_{(n-i)}\ )
$$

where P is the prime numbers sequence.

$$
\begin{array}{ll}
(3) \sqsubseteq & \quad \langle \textbf{ass} \rangle \\
& s : [\ n > 0, Inv[^{13,n}/_{s}], n\ ] \\
\sqsubseteq & \quad \langle \textbf{ass}, \text{ justified below in Sect. 2.4} \rangle \\
& s := 13 \\
(4) \sqsubseteq & \quad \langle \textbf{s-post}, \text{ justified below in Sect. 2.6} \rangle \\
& n, r, s : [\ Inv[^{n-1}/_{n}], Inv \wedge n = 0\ ] \\
\sqsubseteq & \quad \langle \textbf{while,seq,c-frame} \rangle \\
& \textbf{while } n \neq 0 \textbf{ do} \\
& \quad \llcorner var\ k, b \cdot n, r, s : [\ Inv \wedge n \neq 0, Inv[^{n-1}/_{n}]\ ] \lrcorner_{(A)} \\
& \textbf{od};
\end{array}
$$

$\llcorner r : [\ Inv \wedge n = 0, Inv[^0/_n]\ ]\lrcorner_{(B)}$

$A \sqsubseteq$ $\langle$**c-frame,seq**$\rangle$

$\llcorner k, b : [\ Inv \wedge n \neq 0, Inv[^{n-1}/_n]\ ]\lrcorner_{(A1)}$

$\llcorner n, k, b, s : [\ Inv \wedge n \neq 0, Inv[^{n-1}/_n]\ ]\lrcorner_{(A2)}$

$\llcorner s : [\ Inv \wedge n \neq 0, Inv[^{n-1}/_n]\ ]\lrcorner_{(A3)}$

$A1 \sqsubseteq$ $\langle$..., proc,seq$\rangle$

$\textsc{reverseN}(s, k); \textsc{isPrime}(k, b);$

$A2 \sqsubseteq$ $\langle$**if**, where $g = (n \neq 0 \wedge k = rev(s) \wedge b \wedge k \neq s \vee n = 0)$$\rangle$

if g **then**

$\llcorner n, k, s : [\ g \wedge Inv \wedge n \neq 0, Inv[^{n-1}/_n]\ ]\lrcorner_{(i)}$

else

$\llcorner n, k, s : [\ \neg g \wedge Inv \wedge n \neq 0, Inv[^{n-1}/_n]\ ]\lrcorner_{(ii)}$

fi

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

$n := n - 1;$

$ii \sqsubseteq$ $\langle$**skip**$\rangle$

$skip$

$A3 \sqsubseteq$ $\langle$..., proc$\rangle$

$\textsc{nextPrime}(s, s)$

$B \sqsubseteq$ $\langle$**ass**, justified below in Sect. 2.5$\rangle$

$r := s;$

2.4 Proof of $(3) \sqsubseteq s := 13$

We need to prove validity

$$n > 0 \Rightarrow Inv[^{13,n}/_{s,n}]$$

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

$$n > 0 \Rightarrow$$
$$\left(\ 0 \leq n \leq n \wedge s = 13 \wedge rev(13) \in prime \wedge s \neq rev(13) \wedge r = P_{n-n}\ \right)$$

Clearly, we have established the validity of those conjunct of the RHS with $s := 13 and n := n$. This whole conjuncts are obviously true when s initialises with 13 .

2.5 Proof of $(B) \sqsubseteq r := s$

We need to prove validity

$$Inv \wedge n = 0 \Rightarrow Inv[^0/_n]$$

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

$$\begin{array}{c}\left(\ 0 \le i \le n \wedge s = P_i \wedge rev(s) \in prime \wedge s \ne rev(s) \wedge r = P_{(n-i)} \wedge n = 0\ \right)\ \Rightarrow \\ \left(\ 0 \le i \le 0 \wedge s = P_0 \wedge rev(s) \in prime \wedge s \ne rev(s) \wedge r = P_n\ \right)\end{array}$$

Clearly, we should establish the validity of first, second and fifith conjunct of the RHS by replace n as 0 in the LHS. After substit n by 0 in the LHS, RHS is given with $n = 0$ implication.

2.6 Proof of $(4) \sqsubseteq Inv[^{n-1}/_n] \Rightarrow Inv \wedge n = 0$

Expanding the definitions and performing the substitution yields

$$\begin{array}{c}\left(\ 0 \le i \le n-1 \wedge s = P_i \wedge rev(s) \in prime \wedge s \ne rev(s) \wedge r = P_{(n-1-i)}\ \right) \Rightarrow \\ \left(\ n = 0 \wedge 0 \le i \le n \wedge s = P_i \wedge rev(s) \in prime \wedge s \ne rev(s) \wedge r = P_{(n-i)}\ \right)\end{array}$$

According the conjuncts of LHS, there are 2 cases to consider:

1. $n = 0 \wedge Inv[^0/_n]$: its validity as the previour proof.
2. $n \ne 0 \wedge Inv[^{n-1}/_n]$: while $n \ne 0$, the first conjunct can remove in the RHS and replace the rest n with n-1. This become indentical with LHS, therefore, it is valid and true.

We gather the code for the procedure body of EMIRP:

```
s := 2;
while n ≠ 0 do
   reversen(s, k)
   if k ∈ prime ∧ k ≠ s then
      n = n − 1
   else
      skip
   fi
   nextPrime(s, s)
od;
```

3 The C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <gmp.h>
#define USEGMP

#include "reverse.h"
#define FALSE 0
#define TRUE 1

void emirp(mpz_t n, mpz_t r) {
    /* 1. Initialize the beginning prime number s as 13 */
    mpz_t s;
    mpz_init(s);
    mpz_set_ui(s, 13);

    // decrement n till n equal to 0
    while (mpz_cmp_ui(n, 0) != 0) {
        /* 3. reverse s put into r*/
        reversen(s, r);
        /* 4. check if r is prime number and r is not same as s */
        if (mpz_probab_prime_p(r, 40) && mpz_cmp(r, s)) {
            /* 5. decrement the n when r is prime*/
            mpz_sub_ui(n, n, 1);
        }
        /* 6. get the next prime number while n =! 0 */
```

```
        mpz_nextprime(s, s);
    }
    /* while n = 0, reverse r as nth prime is s and return r as result */
    reversen(r, r);
}

int main() {
    char inputStr[1024];
    // mpz_t is the type defined for GMP integers.
    // It is a pointer to the internals of the GMP integer data structure
    mpz_t n;
    int flag;

    printf(”Enter your number: ”);
    flag = scanf(”%1023s”, inputStr);
    // NOTE: never every write a call scanf (”%s”, inputStr);
    // You are leaving a security hole in your code.
    assert(flag > 0);
    // If flag is greater 0 then the operation /*

    // 1. Initialize the number
    mpz_init(n);
    mpz_set_ui(n, 0);

    // 2. Parse the input string as a base 10 number
    flag = mpz_set_str(n, inputStr, 10);
    assert(flag == 0);
    // If flag is not 0 then the operation /*

    // 3. Initialize the result number
    mpz_t r;
    mpz_init(r);

    // 4. find the nth number that reversed also is prime
    emirp(n, r);
    mpz_out_str(stdout, 10, r);
    printf(”\n”);
    // 6. Clean up the mpz_t handles or else we will leak memory
    mpz_clear(n);
    return EXIT_SUCCESS;
}
```

In our C implementation, we used the GMP library, allowing us to replace functions in our toy language:

isPrime is replaced with the function $mpz\_probab\_prime_p$,
nextPrime is replaced with *mpz_nextprime*,
and *reversen* is given to us in the spec.
Other functions are also used such as:
mpz_set_ui is a function which assigns a variable (instead of s=13)
mpz_init is an initialising function.
mpz_sub_ui is a function which allows us to subtract.
mpz_cmp is a function which allows us to compare (instead of r == s)
We use these functions as we are dealing with the type *mpz_t* instead of a regular long or int. We use these functions under the assumption that they work correctly as they are library functions.