Assignment 2

System Modelling and Design

Charlie Bradford
z5114682

Julian Zihao
z5135085

May 2, 2018

1 Specification

1.1 Definitions

To specify our n$^{\text{th}}$ emirp procedure we make use of the following mathematical functions:

$$\textbf{fReversen} : \mathbb{N} \to \mathbb{N}$$

$$\textbf{fReversen}(i) ::= i * 10^{\lfloor log_{10}\ i \rfloor} - 99 * \sum_{j=1}^{\lfloor log_{10}\ i \rfloor} \lfloor x * 10^{-j} \rfloor * 10^{\lfloor log_{10}\ i \rfloor - j}$$

fReversen maps each natural number to its base 10 reverse.

$$\textbf{fIsPrime} : \mathbb{N} \to \mathbb{B}$$

$$\textbf{fIsPrime}(i) ::= i\ \in\ \{n | \forall m\ \in\ 2..(n-1).(n \nmid m)\}$$

fIsPrime returns whether or not a number is in the set of primes.

$$\textbf{fIsEmirp} : \mathbb{N} \to \mathbb{B}$$

$$\textbf{fIsEmirp}(i) ::= \textbf{fIsPrime}(i)\ \wedge\ \textbf{fIsPrime}(\textbf{Reversen}(i))\ \wedge\ i\ \neq\ \textbf{fReversen}(i)$$

IsEmirp checks the three conditions of an emirp: primality, primality of the reverse, lack of palindromity.

In our implementation we will need to make use of equivalent procs and funcs which will respectively be The following equivalent procs and funcs will be used

reversen A proc that takes a value parameter n and assigns the reverse to a result parameter r thus the specification "$r\ :=\ \textbf{fReversen}(n)$" is equivalent to the procedure "var r; $\textbf{reversen}(n,\ r)$"

isPrime A probabilistic function for the purpose of simplicity. May be weak to strong psuedo primes, it is very unlikely to affect the programme.

Rather than using a func equivalent for **fIsEmirp** we will simply test each of the above-listed qualities separately.

Define the predicate P to mean

$$x, n : [\text{true}, \exists! i \in [1..x]((\textbf{fIsEmirp}(i)) \wedge (n = 1 \oplus \exists! j \in [1..(i-1)](P(j, n-1))))]$$

Wherein the unique i must be the highest emirp less than or equal to x and the unique j must be the next emirp less than i. Thus if $P(x, n)$ is true then x is greater than or equal to the n^{th} but less than the $(n+1)^{th}$ emirp.
Now we have the specification of our statement:

$$\text{emirp} : [n > 0, P(p, n) \wedge \textbf{fIsEmirp}(p)]$$

2 Derivation

$$
\begin{array}{ll}
 & p, n : [n > 0, P(p,n) \wedge \textbf{fIsEmirp}(p)] \quad (0) \\
(0) \sqsubseteq & \textbf{var } i := 1; \\
 & p, n, i : [n > 0 \wedge i = 1, P(p,n) \wedge \textbf{fIsEmirp}(p)] \quad (1) \\
(1) \sqsubseteq & p : [n > 0 \wedge i = 1, i = 1 \wedge P(p,i)] \quad (2) \\
 & p, i : [i = 1 \wedge P(p,i), P(p,n)] \quad (3) \\
(2) \sqsubseteq & p := 13; \\
(3) \sqsubseteq & \textbf{while } i < n \textbf{ do} \\
 & \quad \textbf{con } T \bullet p,\ i : [i < n \wedge P(p,i) \wedge p = T, p = T + 2 \wedge P(p,i)] \quad (4) \\
 & \textbf{od} \\
(4) \sqsubseteq & p := p + 2; \\
 & p,\ i : [i < n \wedge P(p-2,i), P(p,i)] \quad (5) \\
(5) \sqsubseteq & \textbf{reversen}(p, r); \\
 & \textbf{if isPrime}(p) \ \wedge\ \textbf{isPrime}(r) \ \wedge\ p \neq r \\
 & \quad \textbf{then con } S \bullet i : [i < n \wedge P(p-2,i) \wedge i = S \wedge \textbf{fIsEmirp}(p), \\
 & \qquad\qquad i = S + 1 \wedge P(p-2,i) \wedge \textbf{fIsEmirp}(p)] \quad \sqsubseteq \quad i := i + 1; \\
 & \quad \textbf{else } i : [i < n \wedge P(p-2,i) \wedge \neg\textbf{fIsEmirp}(p), P(p,i)] \quad \sqsubseteq \quad \textbf{SKIP}; \\
 & \textbf{fi}
\end{array}
$$

3 Implementation

```
                                         {$n \geq 1$} $\Rightarrow I[^{13}/_{p}][^{1}/_{i}]$
$i := 1$;                                $I[^{13}/_{p}]$
$p := 13$;                               $I$
while $i < n$ do                         $I \wedge i < n \wedge \neg\textbf{fIsEmirp}(p+2) \Rightarrow I[^{p+2}/_{p}]$
                                         $I \wedge i < n \wedge \textbf{fIsEmirp}(p+2) \Rightarrow I[^{p+2}/_{p}][^{i+1}/_{counter}]$
    $p := p + 2$;
    REVERSEN($p$, $r$)
    if ISPRIME($p$) $\wedge$ ISPRIME($r$) $\wedge$ $p \neq r$ then
        $i := i + 1$;
    end if;
end while
                                         $I \wedge counter \geq n \wedge \textbf{fIsEmirp}(p) \Rightarrow P(p,n) \wedge \textbf{fIsEmirp}(p)$
```

Where isPrime is defined as in the GMP library mpz_probab_prime_p.

3.1 Invariant

Our single loop invariant is as such:

$I = \{P(p,i) \wedge i \leq n\}$

3.2 Implication 1

$$n \geq 1 \Rightarrow I[^{13}/_{p}][^{1}/_{i}]$$

We begin by unpacking the invariant and performing substitutions.

$$
\begin{aligned}
& n \geq 1 \\
\Rightarrow & \{P(p,i) \wedge i \leq n\}[^{13}/_{p}][^{1}/_{i}] \\
\Leftrightarrow & P(13,1) \wedge 1 \leq n
\end{aligned}
$$

As $n \geq 1 \Leftrightarrow 1 \leq n$ we can simplify the implication to **True** $\Rightarrow P(13,1)$. Expanding this gives:

$$
\begin{aligned}
& P(13,1) \\
\Leftrightarrow & \exists i \in [1..13](\textbf{fIsEmirp}(i) \wedge (1 = 1 \oplus \exists! j \in [1..(i-1)](P(j,0)))) \\
\Leftrightarrow & \textbf{fIsEmirp}(13) \wedge (\textbf{True} \oplus \exists! j \in [1..12](P(j,0)))
\end{aligned}
$$

The simplest way to prove "$\exists! j \in [1..12](P(j,0)$" false is to show there are no values between 1 and 12 inclusive that are both prime and not equal to their reverse. The primes in that range are $L = \langle 2,3,5,7,11 \rangle$, and their respective reverses are $\langle 2,3,5,7,11 \rangle$. All these numbers are equal to their reverses there for there is no $x \in L$ such that $\mathbf{fIsEmirp}(x)$ is true. This gives us:

$$\begin{aligned} & P(13,1) \\ \Leftrightarrow & \mathbf{fIsEmirp}(13) \wedge (\mathbf{True} \oplus \mathbf{False}) \\ \Leftrightarrow & 12! \nmid 13 \wedge 30! \nmid 31 \wedge 12 \neq 31 \end{aligned}$$

By calculation we can show $12! \equiv 12(\mathbf{Mod}\ 13)$ and $30! \equiv 30(\mathbf{Mod}\ 31)$ and it is clear that $13 \neq 31$ thus the RHS is true and so is the implication.

3.3 Implication 2

$$I \wedge i < n \wedge \neg\mathbf{fIsEmirp}(p+2) \Rightarrow I[{}^{p+2}/_{p}]$$

We begin by unpacking the invariant and performing substitutions.

$$\begin{aligned} & P(p,i) \wedge i < n \wedge \neg\mathbf{fIsEmirp}(p+2) \\ \Rightarrow & \{P(p,i) \wedge i \leq n\}[{}^{p+2}/_{p}] \\ \Leftrightarrow & P(p+2,i) \wedge i \leq n \end{aligned}$$

From the LHS we know that i is less than n so, as n is unchanged, it follows that n is less than or equal to i and thus both conjuncts can be discharged. This leaves us with:

$$\begin{aligned} & P(p,i) \wedge \neg\mathbf{fIsEmirp}(p+2) \\ \Rightarrow & P(p+2,i) \end{aligned}$$

So now we have that p is greater than or equal the i^{th} emirp and less than the $(i+1)^{th}$ and that $p+2$ is not an emirp. So we need only show that $p+2$ is less than the $(i+1)^{th}$ emirp. The only candidate for an emirp less than or equal to $p+2$ but greater than p is $p+1$. However, p is intiallised at 13 and only incremented by two so $p+1$ must be an even number. Therefore, as there is no prime number larger than two that is even, $p+1$ cannot be prime let alone an emirp. Thus the RHS is true and the implication holds.

3.4 Implication 3

$$I \wedge i < n \wedge \mathbf{fIsEmirp}(p+2) \Rightarrow I[{}^{p+2}/_{p}][{}^{i+1}/_{i}]$$

We begin by unpacking the invariant and performing substitutions.

$$\begin{aligned} & P(p,i) \wedge i \leq n \wedge i < n \wedge \mathbf{fIsEmirp}(p+2) \\ \Rightarrow & \{P(p,i) \wedge i \leq n\}[{}^{p+2}/_{p}][{}^{i+1}/_{i}] \\ \Leftrightarrow & P(p+2,i+1) \wedge i+1 \leq n \end{aligned}$$

From the LHS we know that $i < n$ so it follows that $i \leq n$ and both of those conjucts can be discharged. This leaves us with:

$$\begin{aligned} & P(p, i \wedge \mathbf{fIsEmirp}(p+2) \\ \Rightarrow & P(p+2,i+1) \end{aligned}$$

We know from the LHS that p is less than the $(i+1)^{th}$ emirp and the RHS asserts that $p+2$ is greater than or equal to the $(i+1)^{th}$ but less than the $(i+2)^{th}$ emirp. Given the LHS, this could only be false if $p+1$ were also an emirp, however, as above, we know that $p+1$ is even, cannot be a prime, and therefore cannot be an emirp.

3.5 Implication 4

$$I \wedge i \geq n \wedge \mathbf{fIsEmirp}(p) \Rightarrow P(p,n) \wedge \mathbf{fIsEmirp}(p)$$

Unpacking the invariant and simplifying:

$$\begin{aligned} & P(p,i) \wedge i \leq n \wedge i \geq n \wedge \mathbf{fIsEmirp}(p) \\ \Leftrightarrow & P(p,i) i = n \wedge \mathbf{fIsEmirp}(p) \\ \Leftrightarrow & P(p,n) \wedge \mathbf{fIsEmirp}(p) \\ \Leftrightarrow & \text{RHS} \end{aligned}$$

4 Program

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

typedef int bool;

#define TRUE 1
#define FALSE 0

void emirp (mpz_t n, mpz_t p);

int main (int argc, char *argv[])
{
    mpz_t n, p;
    mpz_inits(n, p, NULL);

    mpz_inp_str(n, stdin, 10);

    emirp(n, p);

    gmp_printf("%Zd\n", p);
  mpz_clear(p);
    return 0;
}

void emirp (mpz_t n, mpz_t p)
{
    mpz_t i, r;
    mpz_inits(r, NULL);
    mpz_init_set_ui(i, 1);
    mpz_set_ui(p, 13);

    while (mpz_cmp(i, n) < 0)
    {
        mpz_add_ui(p, p, 1);
        reversen(p, r);
        if (mpz_probab_prime_p(r, 50) > 0
            && mpz_probab_prime_p(p, 50) > 0
            && mpz_cmp(r, p) != 0)
        {
            mpz_add_ui(i, i, 1);
        }

    }
    mpz_clears(i, r, n, NULL);
}
```

4.1 Changes

For, gmp.h have been included in the C code and different GMP functions have been used in the C code as well, changes made during the translation to C.

```c
mpz_t i, r
```

The above step is to itnitialize i and r , and set their value to 0.

```c
mpz_inits(r, NULL);
```

The above step is to r's value to 0, however for there needs to be two variable, therefore NULL has been used to fill in.

```c
mpz_clears(i, r, n, NULL);
```

The above step is to free the space occupied by the variables i, r and n.