Assignment 2

Ruofei HUANG(z5141448) Anqi ZHU(z5141541)

April 30, 2018

1 Task 1

1.1 Prime

We define a number n to be a prime if it is a natural number greater than 1 and cannot be formed by mutiplying two natural numbers (bigger than 1) smaller than itself[1]. Hence we can describe the set containing all primes as:

$$Prime = \{n \in \mathbb{N} | \neg(\exists x \in (1..n-1)\,(x|n)) \wedge n > 1\}$$

GMP provides a function called ISPRIME() to check if a certain number n is a prime or not. The procedure of this function can be expressed as:

$$\textbf{proc}\ ISPRIME(\textbf{value}\ n, \textbf{result}\ p)\cdot$$
$$n, p : [TRUE, (n_0 \in Prime \wedge p > 0) \vee (n_0 \notin Prime \wedge p <= 0)]$$

We shall use its procedure call sugar, **result** $p = ISPRIME(\textbf{value}\ n)$, to verify primes later in our refinement.

1.2 Reverse

By the spec in verifying a number v which is the reverse of the number n [2], we can have the following mathematical definition:

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

[1] Reference from Wikipedia: https://en.wikipedia.org/wiki/Prime_number

[2] A proof provided by Lecturer in Control of this course on https://www.cse.unsw.edu.au/~cs2111/18s1/lec/reverse.pdf

where:

$$c(n) = \lfloor log_{10}(n) \rfloor,$$
$$S = [10]^*,$$
$$n \in \mathbb{N} \wedge n = \sum_{i=0}^{c(n)} (S_i 10^{(c(n)-i)})$$

Then we can simplify the spec of **proc** $reversen$(**value** $n : \mathbb{N}$, **result** $v : \mathbb{N}$) given by the lecturer, as

$$\textbf{proc}\ reversen(\textbf{value}\ n : \mathbb{N}, \textbf{result}\ v : \mathbb{N}) \cdot$$
$$r, v : [\text{TRUE}, v = rev(r_0)]$$

1.3 Emirp

An emirp n is a prime number that results in a different prime when its decimal digits are reversed[3]. The definition of emirp can be construct in mathematical semantics as follows:

$$n \in Emirp \iff n \in Prime \wedge rev(n) \in Prime \wedge n \neq rev(n)$$

Then we can define a function to check if the specified number n is an emirp. The function is such as:

$$isEmirp(n) = \begin{cases} 1 & \text{if } n \in Prime \wedge rev(n) \in Prime \wedge n \neq rev(n) \\ 0 & \text{else} \end{cases}$$

We use 0 and 1 as our returning value of the function, so that we can find out how many emirps are found in the range of 2 .. n according to the following mathematics semantics:

$$\textbf{the number of emirps found} = \sum_{i=0}^{n} isEmirp(i)$$

where:

$$n \in \mathbb{N}_{>1}$$

1.3.1 Derivation of ISEMIRP() Procedure Call

We want to transfer the isEmirp() function into a procedure so that we can use it in our later refinement. We start with a spec of the procedure.

$$\textbf{proc}\ ISEMIRP(\textbf{value}\ n : \mathbb{N}, \textbf{result}\ w) \cdot$$
$$\llcorner n, w : \left[\text{TRUE}, \left(\begin{array}{l} (w = 1 \wedge rev(n_0) \neq n_0 \wedge n_0 \in Prime \wedge rev(n_0) \in Prime) \vee \\ (w = 0 \wedge \neg(rev(n_0) \neq n_0 \wedge n_0 \in Prime \wedge rev(n_0) \in Prime)) \end{array} \right)\right] \lrcorner_{(A)}$$

[3] Another reference from Wikipedia :https://en.wikipedia.org/wiki/Emirp

$(A) \sqsubseteq$ $\langle$**c-frame**$\rangle$

$$w : [\text{TRUE}, \left(\begin{array}{l}(w = 1 \wedge rev(n) \neq n \wedge n \in Prime \wedge rev(n) \in Prime) \vee \\ (w = 0 \wedge \neg(rev(n) \neq n \wedge n \in Prime \wedge rev(n) \in Prime))\end{array}\right)]$$

$\sqsubseteq$ $\langle$**i-loc**$\rangle$

$$\textbf{var } r \cdot r, w : [\text{TRUE}, \left(\begin{array}{l}(w = 1 \wedge rev(n) \neq n \wedge n \in Prime \wedge rev(n) \in Prime) \vee \\ (w = 0 \wedge \neg(rev(n) \neq n \wedge n \in Prime \wedge rev(n) \in Prime))\end{array}\right)]$$

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

$\llcorner r : [\text{TRUE}, r = rev(n)];_{\lrcorner(B)}$

$$\llcorner r \cdot w : [r = rev(n), \left(\begin{array}{l}(w = 1 \wedge rev(n) \neq n \wedge n \in Prime \wedge \\ rev(n) \in Prime) \vee (w = 0 \wedge \neg(rev(n) \neq n \\ \wedge n \in Prime \wedge rev(n) \in Prime))\end{array}\right)]_{\lrcorner(C)}$$

$(B) \sqsubseteq$ $\langle$**proc**$\rangle$

$reversen(n, r);$

$(C) \sqsubseteq$ $\langle$**s-post**, justified below in 1.3.2$\rangle$

$$\llcorner r \cdot w : [r = rev(n), \left(\begin{array}{l}(w = 1 \wedge r \neq n \wedge n \in Prime \wedge r \in Prime) \vee \\ (w = 0 \wedge \neg(r \neq n \wedge n \in Prime \wedge r \in Prime))\end{array}\right)]_{\lrcorner(C')}$$

$(C') \sqsubseteq$ $\langle$**if**$\rangle$

if $r \neq n$
then $\llcorner r \cdot w : [r \neq n \wedge pre(C'), post(C')]_{\lrcorner(D)}$
else $\llcorner r \cdot w : [r = n \wedge pre(C'), post(C')]_{\lrcorner(E)}$
fi

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

if $ISPRIME(n) > 0$
then $\llcorner r \cdot w : [pre(D) \wedge n \in Prime, post(C')]_{\lrcorner(F)}$
else $\llcorner r \cdot w : [pre(D) \wedge n \notin Prime, post(C')]_{\lrcorner(G)}$
fi

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

if $ISPRIME(r) > 0$
then$\llcorner r \cdot w : [pre(F) \wedge r \in Prime, post(C')]_{\lrcorner(H)}$
else$\llcorner r \cdot w : [pre(F) \wedge r \notin Prime, post(C')]_{\lrcorner(I)}$
fi

$(E)(G)(I) \sqsubseteq$ $\langle$**ass**, justfied below in 1.3.3$\rangle$

$w := 0$

$(H) \sqsubseteq$ $\langle$**ass**, justified below in 1.3.4$\rangle$

$w := 1$

We gather the code for the procedure body of ISEMIRP:

```
var r;
reversen(n, r);
if r ≠ n then
    if ISPRIME(n) > 0 then
        if ISPRIME(r) > 0
        then w := 1;
        else w := 0;
        fi
    else w := 0;
    fi
else w := 0;
fi
```

1.3.2 Proof of $pre(C)[^{r_0}/_r][^{w_0}/_w] \wedge post(C') \Rightarrow post(C)$

1.3.3 Proof of $(E)(G)(I) \sqsubseteq w := 0$

1.3.4 Proof of $(H) \sqsubseteq w := 1$

1.4 Specification of the Main Procedure

The job of our main procedure, **proc** $EMIRP$(**value** $n : \mathbb{N}$, **result** r), is to find and return the n^{th} emirp, where n is a given positive parameter. Using the function $isEmirp()$ which is defined and proved above, we can specify our main procedure as:

proc $EMIRP$(**value** $n : \mathbb{N}$, **result** r)$\cdot$

$$\llcorner n, r : [n > 0, \sum_{i=2}^{r} isEmirp(i) = n_0 \wedge r \in Emirp] \lrcorner_{(1)}$$

2 Task 2

(1) $\sqsubseteq$ $\langle$**c-frame**$\rangle$

$$r : [n > 0, \sum_{i=2}^{r} isEmirp(i) = n \wedge r \in Emirp]$$

$\sqsubseteq$ $\langle$**i-loc**$\rangle$

$$\textbf{var } count \cdot count, r : [n > 0, \sum_{i=2}^{r} isEmirp(i) = n \wedge r \in Emirp]$$

$\sqsubseteq$ $\langle$**seq**$\rangle$

$\llcorner count, r : [n > 0, Inv]\lrcorner_{(2)};$

$$\llcorner count, r : [Inv, \sum_{i=2}^{r} isEmirp(i) = n \wedge r \in Emirp]\lrcorner_{(3)}$$

where the loop invariant is defined by:

$$Inv = \left(count = \textstyle\sum_{i=2}^{r} isEmirp(i) \wedge n > 0 \wedge 0 \leq count \leq n \wedge r \geq count \right)$$

(2) $\sqsubseteq$ $\langle$Routine work: initialize the variables in the loop$\rangle$

$r := 1; count := 0;$

(3) $\sqsubseteq$ $\langle$**s-post**, justified below$\rangle$

$count, r : [Inv, Inv \wedge (count = n \wedge r \in Emirp)]$

$\sqsubseteq$ $\langle$**while**$\rangle$

while $count \neq n \vee r \notin Emirp$ **do**

$\quad \llcorner count, r : [Inv \wedge (count \neq n \vee r \notin Emirp), Inv]\lrcorner_{(4)}$

od

(4) $\sqsubseteq$ $\langle$**f-ass**, justified below$\rangle$

$\llcorner count, r : [Inv \wedge (count \neq n \vee r \notin Emirp), Inv[^{r+1}/_{r}]]\lrcorner_{(5)}$

$r := r + 1;$

(5) $\sqsubseteq$ $\langle$**c-frame, ass**, justified below$\rangle$

$count := count + ISEMIRP(r);$

We gather the code for the procedure body of EMIRP:

> **var** *count*;
> $r := 2$;
> $count := 0$;
> **while** $count \neq n \vee r \notin Emirp$ **do**
> $r := r + 1$;
> $count := count + ISEMIRP(r)$;
> **od**

2.1 Task 3

2.2 Task 4