

# Assignment 2

Ruofei HUANG(z5141448)      Anqi ZHU(z5141541)

May 1, 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 multiplying two natural numbers (bigger than 1) smaller than itself<sup>1</sup>. 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:

```
proc ISPRIME(value  $n$ , result  $p$ ).  
 $n, p : [TRUE, (n_0 \in Prime \wedge p > 0) \vee (n_0 \notin Prime \wedge p \leq 0)]$ 
```

We shall use its procedure call sugar, **result**  $p = ISPRIME(\mathbf{value} \ n)$ , 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  $n$ <sup>2</sup>, we can have the following mathematical definition:

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

---

<sup>1</sup> Reference from Wikipedia: [https://en.wikipedia.org/wiki/Prime\\_number](https://en.wikipedia.org/wiki/Prime_number)

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

where:

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

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

$$\begin{aligned} &\mathbf{proc} \text{ reversen}(\mathbf{value} \ n : \mathbb{N}, \mathbf{result} \ v : \mathbb{N}). \\ &r, v : [\text{TRUE}, v = \text{rev}(r_0)] \end{aligned}$$

### 1.3 Emirp

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

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

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

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

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:

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

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.

$$\begin{aligned} &\mathbf{proc} \text{ ISEMIRP}(\mathbf{value} \ n : \mathbb{N}, \mathbf{result} \ w). \\ &\llbracket n, w : [\text{TRUE}, \left( \begin{aligned} &(w = 1 \wedge \text{rev}(n_0) \neq n_0 \wedge n_0 \in \text{Prime} \wedge \text{rev}(n_0) \in \text{Prime}) \vee \\ &(w = 0 \wedge \neg(\text{rev}(n_0) \neq n_0 \wedge n_0 \in \text{Prime} \wedge \text{rev}(n_0) \in \text{Prime})) \end{aligned} \right) ] \rrbracket^{-(A)} \end{aligned}$$

---

<sup>3</sup> Another reference from Wikipedia :<https://en.wikipedia.org/wiki/Emirp>

(A)  $\sqsubseteq$   $\langle \mathbf{c\text{-}frame} \rangle$

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

$\sqsubseteq$   $\langle \mathbf{i\text{-}loc} \rangle$

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

$\sqsubseteq$   $\langle \mathbf{seq2} \rangle$

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

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

(B)  $\sqsubseteq$   $\langle \mathbf{proc} \rangle$

$\text{reversen}(n, r);$

(C)  $\sqsubseteq$   $\langle \mathbf{c\text{-}frame} \rangle$

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

$\sqsubseteq$   $\langle \mathbf{s\text{-}post, justified below in 1.3.2} \rangle$

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

(C')  $\sqsubseteq$   $\langle \mathbf{if} \rangle$

$\mathbf{if} \ r \neq n$

$\mathbf{then} \ \llcorner w : [r \neq n \wedge \text{pre}(C'), \text{post}(C')] \llcorner(D)$

$\mathbf{else} \ \llcorner w : [r = n \wedge \text{pre}(C'), \text{post}(C')] \llcorner(E)$

$\mathbf{fi}$

(D)  $\sqsubseteq$   $\langle \mathbf{if} \rangle$

$\mathbf{if} \ \text{ISPRIME}(n) > 0$

$\mathbf{then} \ \llcorner w : [\text{pre}(D) \wedge n \in \text{Prime}, \text{post}(C')] \llcorner(F)$

$\mathbf{else} \ \llcorner w : [\text{pre}(D) \wedge n \notin \text{Prime}, \text{post}(C')] \llcorner(G)$

$\mathbf{fi}$

(F)  $\sqsubseteq$   $\langle \mathbf{if} \rangle$

$\mathbf{if} \ \text{ISPRIME}(r) > 0$

$\mathbf{then} \ \llcorner w : [\text{pre}(F) \wedge r \in \text{Prime}, \text{post}(C')] \llcorner(H)$

$\mathbf{else} \ \llcorner w : [\text{pre}(F) \wedge r \notin \text{Prime}, \text{post}(C')] \llcorner(I)$

$\mathbf{fi}$

(E)(G)(I)  $\sqsubseteq$   $\langle \mathbf{ass, justified below in 1.3.3} \rangle$

$w := 0$

3

(H)  $\sqsubseteq$   $\langle \mathbf{ass, justified below in 1.3.4} \rangle$

$w := 1$

We gather the code for the procedure body of ISEMIRP:

```

var  $r$ ;
 $reversen(n, r)$ ;
if  $r \neq n$  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 $r = rev(n)[^{w_0}/_w] \wedge post(C*)$

$$\begin{aligned}
& r = rev(n)[^{w_0}/_w] \wedge post(C') \\
\Leftrightarrow & \quad \langle \text{Substitue and expand } post(C') \rangle \\
& r = rev(n) \wedge \left( \begin{array}{l} (w = 1 \wedge rev(n) \neq n \wedge n \in Prime \wedge \\ rev(n) \in Prime) \vee (w = 0 \wedge \neg(rev(n) \neq n) \\ \wedge n \in Prime \wedge rev(n) \in Prime) \end{array} \right) \\
\Leftrightarrow & \quad \langle \text{Substitute } r = rev(n) \rangle \\
& \left( \begin{array}{l} (w = 1 \wedge r \neq n \wedge n \in Prime \wedge r \in Prime) \vee \\ (w = 0 \wedge \neg(r \neq n \wedge n \in Prime \wedge r \in Prime)) \end{array} \right) \\
\Leftrightarrow & \quad \langle \text{definition of } post(C) \rangle \\
& post(C)
\end{aligned}$$

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

Before proving this assertion, we observe that in the  $post(C')$ , we have

$$\begin{aligned}
& \neg(r \neq n \wedge n \in Prime \wedge r \in Prime) \\
\Leftrightarrow & r = n \vee n \notin Prime \vee r \notin Prime \\
\Rightarrow & w = 0
\end{aligned}$$

Then this assertion is valid:

$$r = n \vee n \notin Prime \vee r \notin Prime \Rightarrow post(C')[^0/_w]$$

By observe the  $pre(E)$  , $pre(G)$  , $pre(I)$ , we have:

$$\begin{aligned} pre(E) &\Rightarrow r = n \Rightarrow post(C')^0/w] \\ pre(G) &\Rightarrow n \notin Prime \Rightarrow post(C')^0/w] \\ pre(I) &\Rightarrow r \notin Prime \Rightarrow post(C')^0/w] \end{aligned}$$

Which can all prove the validity.

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

Similar to 1.3.3 we have:

$$pre(H) \Leftrightarrow (r \neq n \wedge n \in Prime \wedge r \in Prime) \Rightarrow post(C')^1/w]$$

## 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:

$$\begin{aligned} &\text{proc } EMIRP(\text{value } n : \mathbb{N}, \text{result } r) \cdot \\ &\quad \llbracket n, r : [n > 0, \sum_{i=2}^r isEmirp(i) = n_0 \wedge r \in Emirp] \rrbracket_{(1)} \end{aligned}$$

## 2 Task 2

$$\begin{aligned} (1) &\sqsubseteq \langle \text{c-frame} \rangle \\ &\quad r : [n > 0, \sum_{i=2}^r isEmirp(i) = n \wedge r \in Emirp] \\ &\sqsubseteq \langle \text{i-loc} \rangle \\ &\quad \text{var } count \cdot count, r : [n > 0, \sum_{i=2}^r isEmirp(i) = n \wedge r \in Emirp] \\ &\sqsubseteq \langle \text{seq} \rangle \\ &\quad \llbracket count, r : [n > 0, Inv] \rrbracket_{(2)}; \\ &\quad \llbracket count, r : [Inv, \sum_{i=2}^r isEmirp(i) = n \wedge r \in Emirp] \rrbracket_{(3)} \end{aligned}$$

Where the loop invariant is defined by:

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

(2)  $\sqsubseteq$        $\langle \text{Routine work: initialize the variables in the loop} \rangle$   
 $r := 1; \text{count} := 0;$   
 (3)  $\sqsubseteq$        $\langle \text{s-post, justified below} \rangle$   
 $\text{count}, r : [\text{Inv}, \text{Inv} \wedge (\text{count} = n \wedge r \in \text{Emirp})]$   
 $\sqsubseteq$        $\langle \text{while, a procedure call sugar, similar proof in 1.1.1} \rangle$   
**while**  $\text{count} \neq n \vee \text{ISEMIRP}(r) \neq 1$  **do**  
 $\quad \sqcup \text{count}, r : [\text{Inv} \wedge (\text{count} \neq n \vee r \notin \text{Emirp}), \text{Inv}] \sqcup (4)$   
**od**  
 (4)  $\sqsubseteq$        $\langle \text{f-ass} \rangle$   
 $\sqcup \text{count}, r : [\text{Inv} \wedge (\text{count} \neq n \vee r \notin \text{Emirp}), \text{Inv}^{[\text{count} + \text{ISEMIRP}(r) / \text{count}]}]; \sqcup (5)$   
 $\text{count} := \text{count} + \text{ISEMIRP}(r);$   
 (5)  $\sqsubseteq$        $\langle \text{c-frame, ass, justified below} \rangle$   
 $r := r + 1;$

We gather the code for the procedure body of EMIRP:

```

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

```

### 3 Task 3

#### 3.1 Proof of

$$pre(3)[^{count_0}/_{count}][^{r_0}/_r] \wedge (Inv \wedge (count = n \wedge r \in Emirp)) \Rightarrow post(3)$$

$$\begin{aligned}
& pre(3) \wedge Inv \wedge (count = n \wedge r \in Emirp) \\
\Leftrightarrow & \quad \langle \text{Expand } pre(3) \rangle \\
& Inv[^{count_0}/_{count}][^{r_0}/_r] \wedge (Inv \wedge (count = n \wedge r \in Emirp)) \\
\Leftrightarrow & \quad \langle \text{Substitute } Inv \text{ and } Inv[^{count_0}/_{count}][^{r_0}/_r] \rangle \\
& \left( \begin{array}{l} (count_0 = \sum_{i=2}^{r_0} isEmirp(i) \wedge n > 0 \wedge 0 \leq count_0 \leq n \wedge r_0 \geq count_0) \wedge \\ (count = \sum_{i=2}^r isEmirp(i) \wedge n > 0 \wedge 0 \leq count \leq n \wedge r \geq count \wedge \\ (count = n \wedge r \in Emirp)) \end{array} \right) \\
\Rightarrow & \quad \langle \text{Simplify and remove } count. \rangle \\
& \left( \begin{array}{l} (count_0 = \sum_{i=2}^{r_0} isEmirp(i) \wedge n > 0 \wedge 0 \leq count_0 \leq n \wedge r_0 \geq count_0) \wedge \\ (n = \sum_{i=2}^r isEmirp(i) \wedge r \geq n \wedge r \in Emirp) \end{array} \right) \\
\Rightarrow & \quad \langle \text{Directly imply from second line.} \rangle \\
& \sum_{i=2}^r isEmirp(i) = n \wedge r \in Emirp \\
\Leftrightarrow & \quad \langle \text{Definition of } post(3) \rangle \\
& post(3)
\end{aligned}$$

#### 3.2 Proof of (5) $\sqsubseteq r := r + 1$

We need to prove the validity of:

ss

### 4 Task 4