Advanced Programming Principles

Ming-Chang Chiu 5123394

Q1. power n x = x^n

$\forall n$, P(n) if P(0) and P(n-1) $\Rightarrow$ P(n), where P(n) is power n x = x^n

Base: P(0) = x^0

power 0 x = 1 , by definition of power

= x^0 , by property of exponential, $x^0 = 1$

Inductive case: P(n) = x^n , where P(n-1) = power (n-1) x = x^{n-1} holds.

power n x = x * power (n-1) x , by def of power

= x * x^{n-1} , by inductive hypothesis

= x^{n-1} * x , by commutativity of multiplication

= x^n , by exponential property

Q2. $\forall n$, P(n) if P(zero) and P(Succ n) when P(n) holds.

P(zero) : power zero x

= 1.0 , by def of power

= x^0 , by property of exponential, $x^0 = 1$

= $x^{toInt(zero)}$, by def of toInt

P(Succ n) : Given P(n) = power n x = $x^{toInt(n)}$

Power (Succ n) x

= x * power n x , by def of power

= x * $x^{toInt(n)}$, by inductive hypothesis

= $x^{toInt(n)+1}$, by property of exponential, $x^a \cdot x^b = x^{a+b}$

= $x^{toInt(Succ\ n)}$, by def of toInt

Q3. $\forall l$, P l if P([]) and

P(v::l'), if P(l')

P([]) : Length([]@r)

= length (r) , by property of lists

= 0 + Length r , by identity of addition of natural number

= Length [] + Length r , by def of Length

$P(l)$: Given $l = x::xs$ and $length(xs@r) = length\ xs + length\ r$

$length((x::xs)@r)$
$= length([x]@xs@r)$, by property of lists
$= length([x]@(xs@r))$, by property of lists
$= length(x::(xs@r))$, — " —
$= 1 + length(xs@r)$, by def of length
$= 1 + length\ xs + length\ r$, by inductive hypothesis
$= length(x::xs) + length\ r$, by def of length
$= length\ l + length\ r$, by def of l

Q4 $length(reverse\ l) = length\ l$.

$P([\,])$:
$length(reverse\ [\,]) = length([\,])$, by def of reverse

$P(l)$: Given $l = x::xs$ and $length(reverse\ xs) = length\ xs$

$length(reverse\ x::xs) = length(reverse\ xs@[x])$, by def of reverse
$= length(reverse\ xs) + length\ x$ by def of length
$= length\ xs + length\ x$, by inductive hypothesis
$= length(xs@[x])$, by def of length
$= length\ l$, by property of lists

Q5. $reverse(l_1@l_2) = reverse\ l_2 @ reverse\ l_1$

$P([\,], l_2)$:
$reverse([\,]@l_2) = reverse\ l_2$, by property of lists
$= reverse\ l_2 @ [\,]$, by property of lists
$= reverse\ l_2 @ reverse\ [\,]$, by def of reverse

$P(l_1, l_2)$: Given $l_1 = x::xs$ and reverse $(xs@l_2)$ = reverse l_2 @ reverse xs

reverse $(x::xs)@l_2)$ = reverse $([x]@xs@l_2)$, by property of lists

= reverse $(x::(xs@l_2)$, ——— " ———

= reverse $(xs@l_2)@[x]$, by def of reverse

= reverse l_2 @ reverse xs @ $[x]$, by inductive hypothesis

= reverse l_2@ reverse$(x::xs)$, by def of reverse

= reverse l_2 @ reverse l, by def of l_1

Q6 Assume sorted l is true then sorted (place e l) is also true

$P([\,])$:

sorted(place e [ ]) = sorted ([e]) , by def of place

= sorted (e::[ ]), by property of lists

= true , by def of sorted

$P(l)$: Given $l = x::xs$ and sorted$(xs) \Rightarrow$ sorted(place e xs)

case $x < e$ and $x = e$

sorted(place e x::xs) = sorted(x::(place e xs)), by def of place

= x<= (any element in (place e xs) && sorted(place e xs), by def of sorted

= sorted(place e xs), by case assumption and the premise, sorted l

= true, by inductive hypothesis

case $x > e$:

sorted(place e x::xs) = sorted(e::x::xs), by def of place

= e<=x && sorted(x::xs) , by def of sorted

= sorted(x::xs) , by case assumption

= sorted (l) , by def of l

= true , by the premise, l is sorted.

Q7 (i) In (place e ℓ) e will be placed ~~at~~ before the first element that is larger in ℓ. If there is no element larger than e in ℓ, then e will be put at the end of ℓ. So, is_elem e (place e ℓ) will always true even if ℓ is not sorted.

(ii) The premise (sorted ℓ) is needed in Q6. If the premise doesn't exist, then given $\ell = [4:3:1]$ and $e=2$, sorted (place e ℓ) will be false. There are so many counter example can show that sorted(place e ℓ) is false when the premise (sorted ℓ) doesn't exist. So, sorted ℓ is necessary.

Q8 Loop invariant: $z = y^i$

1. Since $i=0$ and $z=1$ before the loop starts, by exponential property, $z = 1 = y^0 = y^i$. The loop invariant holds.

2. If $z = y^i$ holds before the loop, after an iteration
   $z = y$ and $i = 1$,
   So, $z = y = y^1 = y^i \Rightarrow$ look invariant is also true

3. Since the loop terminates when $i = x$, we know $z = y^x$ holds because $z = y^i = y^x$

Q9.

Note: $n \geq 1$ and accuracy=0.01 and loop invariant: lower $\leq \sqrt{n} \leq$ upper

```
upper = n
lower = 0
while (upper - lower > accuracy) {
    if (((upper+lower)/2)^2 > n)
        up = (upper+lower)/2
    else
        lower = (upper+lower)/2
}
```

1. Before entering the loop

upper $= n \geq \sqrt{n}$ for $n \geq 1$

lower $= 0 \leq \sqrt{n}$ for $n \geq 1$

so the loop invariant holds

2. Assume loop invariant holds before entering the loop.

After the first iteration,

case $\left(\frac{upper+lower}{2}\right)^2 > n$:

upper $= \frac{upper+lower}{2} = \frac{n+0}{2} = \frac{n}{2}$.

lower $= 0 \leq \sqrt{n}$ still holds, and by case assumption, $upper^2 = \left(\frac{n}{2}\right)^2 = \frac{n^2}{4} > n$

so upper $= \frac{n}{2} > \sqrt{n}$ also hold.

case $((upper+lower)/2)^2 \leq n$:

lower $= (upper+lower)/2 = \frac{n}{2}$

upper $= n \geq \sqrt{n}$ still holds, and by case assumption, lower $= \frac{n}{2} \leq \sqrt{n}$

so lower $\leq \sqrt{n}$ holds as well.

3. The loop terminates when (upper-lower $\leq$ accuracy), thus upper-lower $\leq$ accuracy. And we know loop invariant holds after loop is terminated. so,

$$(upper - lower \leq accuracy) \land (lower \leq \sqrt{n} \leq upper)$$