HW5

Q1. $\forall n$.. P(n) if P(0) and P(n-1) $\rightarrow$ P(n)

P(n) is power n x = x^n

Case P(0): Show that power 0 x = x^0 = 1.0

power 0 x = 1.0 , by def of power #

Case P(n): Show power n x = x^n, Given power n-1 x = x^{n-1} #

power n x = x *. power n-1 x , by def. of power

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

= x^n , by rules of exponent arithmetic.

#

Q2

Principle : $\forall n$, P(n) if P(zero) and P(Succ n) if P(n) #

P(n) is power n x = $x^{toInt(n)}$

Case P(Zero): Show power Zero x = $x^{toInt(Zero)}$

power Zero x = 1.0 , by def of power

= x^0 , by exponent arithmetics

= $x^{toInt(Zero)}$, by def of toInt. #

Case P(Succ n): Show power (Succ n) x = $x^{toInt(Succ\ n)}$, Given 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 exponent arithmetics

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

Q3. Principle: $\forall l$, $P(l)$ if $P([])$ and $P(x::xs)$ if $P(xs)$ #

$P(l, r)$ is length $(l @ r)$ = length l + length r

Case $P([], r)$: Show length $([] @ r)$ = length $[]$ + length r

length $([] @ r)$ = length r , by property of lists

= 0 + length r , by arithmetics

= length $[]$ + length r, by def of length

Case $P(x::xs, r)$: Show length $(x::xs @ r)$ = length $(x::xs)$ + length r #

Given length $(xs @ r)$ = length xs + length r

length $(x::xs @ r)$ = length $(x :: (xs @ r))$, by principles of community

= 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 #

Q4 $P(l)$ is length (reverse l) = length l

Case $P([])$: Show length (reverse $[]$) = length $[]$

length (reverse $[]$) = length $[]$, by def of reverse #

Case $P(x::xs)$: Show length (reverse $x::xs$) = length $(x::xs)$,

Given length (reverse xs) = length xs

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

= length (reverse xs) + length $[x]$, by proof in Q3

= length xs + length $[x]$, by inductive hypothesis

= length $[x]$ + length xs , by commutativity of addition

= length $([x] @ xs)$, by proof in Q3

= length $(x::xs)$, by property of lists #

Q5. P(l_1, l_2) is reverse (l_1 @ l_2) = reverse l_2 @ reverse l_1

Case P([], l_2): Show reverse ([] @ l_2) = reverse l_2 @ reverse []

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. #

Case P(x::xs, l_2): Show reverse (x::xs @ l_2) = reverse l_2 @ reverse (x::xs)

Given reverse (xs @ l_2) = reverse l_2 @ reverse xs

reverse (x::xs @ l_2) = reverse (x::(xs @ l_2)), by principles of community

= 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 #

Qb. P(l) is sorted(l) => sorted(place e l)

Case P([], e): Show sorted [] => sorted(place e [])

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

= true, by def of sorted #

Case P(x::xs, e): Given sorted xs => sorted(place e xs),

Show sorted (x::xs) => sorted place e (x::xs)

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

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

= true && true,

↑ by case assumption ↑ by inductive hypothesis

Case e ≥ x:

A. if place e xs = [e]

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

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

= x <= e && sorted (e::[]), by def of sorted.

(by case assumption; by def of sorted)

= true && true #

B. if place e xs = e::xs

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

= sorted (place e xs), by case assumption

= true, by inductive hypothesis. #

C. if place e xs = x1::(place e x') where xs = x1::x',

sorted (place e (x::xs)) = sorted (x::x1::(place e x')), by def of place

= x <= x1 && sorted (x1::(place e x')), by def of sorted

= true && sorted (place e xs)

(by inductive hypothesis: sorted (x::xs) = sorted (x::x1::x') = x <= x1 && sorted (x1::x') holds; by case assumption)

#

Q7.

① Whether or not l is a sorted list, is_element will linearly scan through the whole list until it finds the specified element e, so it doesn't matter if l is sorted or not, is_element will work as long as it is given a list.

② Yes. ∵ In the proof of Q6 we used the assumption sorted (x::xs) = sorted (e), ∴ we need it to hold in order for the proof work.

Q8. loop invariant: $z = y^i$

① Before the loop, Preconditions $x \geq 0$, $z = 1$, $i = 0$ implies $z = 1 = y^0 = y^i$

$\Rightarrow$ implies loop invariant.

So loop invariant holds before the loop #

② After the nth iteration:

If loop invariant holds before the loop body,

ie, $z = y^n$ holds,

Then after the loop body, $z = y^n * y = y^{n+1}$, $i = n+1$

$\Rightarrow z = y^i$ loop invariant holds for the n+1 th iteration.

By induction, loop invariant holds for every cycle of the loop #

③ When the loop terminates, $i = x$ to break the while loop.

$\because$ loop invariant holds, $z = y^i = y^x$ implies post condition #

Q9.

```
lower = 1.0
upper = n
while (upper - lower > accuracy) {
    guess = (lower + upper) / 2.0
    if (guess * guess > n)
        upper = guess
    else
        lower = guess
}
```

loop invariant: $lower \leq \sqrt{n} \leq upper$

① Before the loop: Preconditions $n \geq 1.0$, $lower = 1.0$, $upper = n$ implies $n = upper \geq \sqrt{n} \geq lower = 1.0$ #

② After the kth iteration.

If loop invariant holds before the loop body, ie, $lower \leq \sqrt{n} \leq upper$ holds

then after the loop body,

Case1 $guess * guess > n \Rightarrow guess \geq \sqrt{n}$

$\therefore$ after the assignment, $upper = guess \geq \sqrt{n} \geq lower$, loop invariant holds.

Case2 $guess * guess \leq n \Rightarrow guess \leq \sqrt{n}$

$\therefore$ after the assignment, $lower = guess \leq \sqrt{n} \leq upper$, loop invariant holds.

$\therefore$ loop invariant holds for the k+1th iteration.

By induction, loop invariant holds for every cycle of the loop. #

② When the loop terminates, upper - lower ≤ accuracy to break while loop.

∴ loop invariant holds, lower ≤ $\sqrt{n}$ ≤ upper hold.

∴ implies post conditions upper - lower ≤ accuracy ∧ lower ≤ $\sqrt{n}$ ≤ upper #