Assignment 2 (Written)

Due: Monday April 12, 2021 11:59 PM (Australian Eastern Standard Time)

Submit your assignment

Help

After you have completed the assignment, please save, scan, or take photos of your work and upload your files to the questions below. Crowdmark accepts PDF, JPG, and PNG file formats.

Q1 (10 points)

The following is a tail-recursive definition for the exponential function that **counts down**

```
expDown :: Int -> Int -> Int
expDown x y = h_exp x y 1

h_exp :: Int -> Int -> Int -> Int
h_exp x 0 ans = ans
h_exp x y ans = h_exp x (y-1) (ans*x)
```

A) Give the tail-recursive definition for `expUp` via `h_expUp` -- an implementation of the exponential function that **counts up**. You may presume $y \geq 0$. [3 points]

B) Find the **iteration invariant** for `h_expUp` and prove each line of your it satisfies it. [4 points]

C) Use your invariant to prove the correctness of `expUp`. [2 points]

D) State the **bound value** for this program. [1 points]

Q2 (5 points)

Define a function `iter` such that for all $n \geq 0$ and any `f :: a -> a`

```
iter n f
```

is the n-fold composition of f with itself. That is, $f \circ f \circ \cdots \circ f$ with the f written n-times.

E.g. 2-fold composition is $f \circ f$, 3-fold $f \circ f \circ f$, and so on.

Do not discuss the base case on Piazza --- determining the non-obvious base case is part of the assessment

Time left Hide

3 days, 19 hours

Q3 (5 points)

Given the code

```
foo :: a -> [a] -> a
foo = foldl (curry snd)
```

A) **Very briefly** explain (in English) what `foo` ultimately does. [2 points]

B) Provide a more straightforward implementation. [3 points]

Q4 (10 points)

Consider the following definitions for multiplying Natural numbers.

```
data Nat = Zero | Succ Nat deriving Show

emb :: Nat -> Int
emb Zero     = 0
emb (Succ n) = 1 + emb n

plus :: Nat -> Nat -> Nat
plus n Zero     = n
plus n (Succ m) = plus (Succ n) m

times :: Nat -> Nat -> Nat
times _ Zero = Zero
times n (Succ m) = plus n $ times n m
```

Using induction over `n` prove that

```
emb( times m n ) = emb(m) * emb(n)
```