Quiz (Week 3)

Properties for Functions

Question 1

The ROT13 Cipher is a simple substitution cipher which rotates the alphabet by thirteen places, that is, `A` becomes `N` and `Z` becomes `M`. Here is a simple (rather inefficient) Haskell implementation:

```
rot13 :: String -> String
rot13 = map $ \x ->
          case lookup x table of
            Just y  -> y
            Nothing -> x
  where
    table = table' 'A' 'Z' ++ table' 'a' 'z'
    table' a z = zip [a..z] (drop 13 (cycle [a..z]))
```

Select all the properties that this function satisfies (assuming ASCII strings).

1. ☑ `length x == length (rot13 x)`
2. ☑ `rot13 (map toUpper x) == map toUpper (rot13 x)`
3. ☐ `rot13 (map f x) == map f (rot13 x)`
4. ☐ `all (not . isAlpha) x ==> rot13 x == x`
5. ☑ `rot13 (a ++ b) == rot13 a ++ rot13 b`
6. ☐ `not (null x) ==> ord (head x) + 13 == ord (head (rot13 x))`
7. ☑ `rot13 (rot13 x) == x`

Question 2

Here is a function that fairly merges ordered lists, e.g. `merge [2,4,6,7] [1,2,3,4] == [1,2,2,3,4,4,6,7]`.

```
merge :: (Ord a) => [a] -> [a] -> [a]
merge (x:xs) (y:ys) | x < y     = x:merge xs (y:ys)
                    | otherwise = y:merge (x:xs) ys
```

```
merge xs [] = xs
merge [] ys = ys
```

Select all the properties that this function satisfies.

1. ☑ `merge (sort a) (sort b) == sort (merge a b)`
2. ☑ `merge a b == sort (a ++ b)`
3. ☑ `length (merge a b) == length a + length b`
4. ☑ `merge (filter f a) (filter f b) == filter f (merge a b)`
5. ☑ `merge (map f a) (map f b) == map f (merge a b)`
6. ☑ `sort (merge a b) == sort (a ++ b)`

Question 3

The following code converts Haskell `Int` values to and from strings containing their binary representation (as a sequences of `'1'` and `'0'` characters).

```
toBinary :: Int -> String
toBinary 0 = ""
toBinary n = let (d,r) = n `divMod` 2
             in toBinary d
                  ++ if r == 0 then "0"
                               else "1"

fromBinary :: String -> Int
fromBinary = fst . foldr eachChar (0,1)
  where
    eachChar '1' (sum, m) = (sum + m, m*2)
    eachChar _   (sum, m) = (sum    , m*2)
```

Select all properties that these functions satisfy.

1. ☑ `i >= 0 ==> fromBinary (toBinary i) == i`
2. ☐ `all (`elem` "01") s ==> toBinary (fromBinary s) == s`
3. ☐ `all (`elem` "01") s ==> read s >= fromBinary s`
4. ☑ `i > 0 ==> length (toBinary i) >= length (show i)`
5. ☐ `all (`elem` "01") s ==> fromBinary s == fromBinary ('0':s)`

Question 4

The following function removes adjacent duplicates from a list.

```haskell
dedup :: (Eq a) => [a] -> [a]
dedup (x:y:xs) | x == y = dedup (y:xs)
               | otherwise = x : dedup (y:xs)
dedup xs = xs
```

Assume the presence of the following `sorted` predicate:

```haskell
sorted :: (Ord a) => [a] -> Bool
sorted (x:y:xs) = x <= y && sorted (y:xs)
sorted xs = True
```

Select all properties that `dedup` satisfies.

1. [x] `sorted xs ==> sorted (dedup xs)`
2. [x] `sorted xs ==> dedup xs == nub xs`
3. [x] `sorted xs ==> dedup (dedup xs) == dedup xs`
4. [ ] `sorted xs && sorted ys ==> dedup xs ++ dedup ys == dedup (xs ++ ys)`
5. [ ] `sorted xs ==> length (dedup xs) < length xs`
6. [x] `(x `elem` xs) == (x `elem` dedup xs)`

Functions for Properties

Question 5

Here are a set of properties that the function `foo` must satisfy:

```haskell
foo :: [a] -> (a -> b) -> [b]
foo = undefined -- see below

prop_1 :: [Int] -> Bool
prop_1 xs = foo xs id == xs

prop_2 :: [Int] -> (Int -> Int) -> (Int -> Int) -> Bool
prop_2 xs f g = foo (foo xs f) g == foo xs (g . f)
```

Choose an implementation for `foo` that satisfies the above properties, and type-checks:

1. ( )

```haskell
foo xs f = []
```

2. ○

```
foo xs f = xs
```

3. ○

```
foo [] f = []
foo (x:xs) f = x : foo xs f
```

4. ◉

```
foo [] f = []
foo (x:xs) f = f x : foo xs f
```

5. ○

```
foo [] f = []
foo (x:xs) f = foo xs f
```

Question 6

```
bar :: [Int] -> [Int]
bar = undefined

prop_1 :: [Int] -> Bool
prop_1 xs = bar (bar xs) == xs

prop_2 :: [Int] -> Bool
prop_2 xs = length xs == length (bar xs)

prop_3 :: [Int] -> (Int -> Int) -> Bool
prop_3 xs f = bar (map f xs) == map f (bar xs)
```

Choose all implementations for `bar` that satisfy the above properties, and type-check:

1. ☐

```
bar []     = []
bar (x:xs) =      bar (filter (<=x) xs)
           ++ x : bar (filter (> x) xs)
```

2. ☑

```
bar xs = go xs []
  where go []     acc = acc
        go (x:xs) acc = go xs (x:acc)
```

3. ☐

```
bar []     = []
bar (x:xs) = xs ++ [x]
```

4. ☑

```
bar = id
```

5. ☐

```
bar xs = nub xs
```

6. ☐

```
bar xs = replicate (length xs) (maximum xs)
```

Question 7

```
baz :: [Integer] -> Integer
baz = undefined

prop_1 :: [Integer] -> [Integer] -> Bool
prop_1 xs ys = baz xs + baz ys == baz (xs ++ ys)

prop_2 :: [Integer] -> Bool
prop_2 xs = baz xs == baz (reverse xs)

prop_3 :: Integer -> [Integer] -> Bool
prop_3 x xs = baz (x:xs) - x == baz xs
```

Choose a law-abiding definition for `baz`, that type checks:

1. ◉

```
baz = foldr (+) 0
```

2. ○

```
baz []     = 0
baz (x:xs) = 1 + baz xs
```

3. ○

```
baz []     = 1
baz (x:xs) = x + baz xs
```

4.

```
baz xs = 0
```

Question 8

Here is a definition of a function `fun`, and properties for another function `nuf`:

```
fun :: [Integer] -> [Integer]
fun []       = []
fun [x]      = []
fun (x:y:xs) = (y-x):fun (y:xs)

nuf :: [Integer] -> Integer -> [Integer]
nuf = undefined

prop_1 :: [Integer] -> Integer -> Bool
prop_1 xs x = nuf (fun (x:xs)) x == (x:xs)

prop_2 :: [Integer] -> Integer -> Bool
prop_2 xs x = fun (nuf xs x) == xs
```

Choose a definition for `nuf` that type checks and satisfies the given properties:

1.

```
nuf [] i = []
nuf (x:xs) i = (i + x) : nuf xs (i + x)
```

2.

```
nuf [] i = [i]
nuf (x:xs) i = (i + x) : nuf xs (i + x)
```

3. Haskell

```
nuf xs i = scanl (\v x -> v + x) i xs
```

4.

```
nuf [] i = []
nuf (x:xs) i = (i + x) : nuf xs i
```

5.

```
nuf xs i = i : scanl (\v x -> v + x) i xs
```

Due: Friday, June 28, 11:59:59 pm

Upon clicking submit, you will be prompted for your zID and zPass. Please make sure that your answers are final and that you have answered every question.

If there is a problem, please contact the course administrator.

Submit

You can click here to check if you have submitted already.