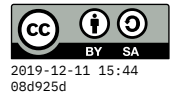


Assignment 10



due *Tue 26 Nov*

module A10 **where**

import Test.QuickCheck

*-- This will be a program to represent integers as strings of binary
-- digits, so this produces a single bit 0 or 1, depending on whether
-- the integer is even or odd.*

bit :: **Integer** -> **Char**

bit k = **if** even k **then** '0' **else** '1'

*-- Convert an integer to a string of bits. Give this a test! Here are
-- some working examples:*

*-- intToBinary 10 --> "1010"
-- intToBinary 15 --> "1111"
-- intToBinary 17 --> "10001"
-- intToBinary 18 --> "10010"*

intToBinary :: **Integer** -> **String**

intToBinary k = reverse (loop k)

where

loop 0 = []

loop n = bit n : loop (n `div` 2)

*-- Convert a string of bits to an integer, assuming it's well-formatted,
-- or Nothing if it's not a valid bit string.*

*-- binaryToInt "10010" --> Just 18
-- binaryToInt "1001101" --> Just 77
-- binaryToInt "z100 10" --> Nothing*

binaryToInt :: **String** -> **Maybe Integer**

binaryToInt str = loop 1 (reverse str)

where

loop _ [] = **Just** 0

loop k (b:bs) = **do**

n <- loop (k*2) bs

case b **of**

```

    '0' -> Just n
    '1' -> Just (k+n)
    _   -> Nothing

-- Here's a QuickCheck property for starting with an integer and doing
-- a round-trip to binary. The triple-equals (===) tests equality, but
-- also allows QuickCheck to print both sides whenever there's a
-- failure. Run it like this:
--   □> quickCheck toBinaryRoundTrip
--   *** OK, passed 100 tests; 107 discarded.

toBinaryRoundTrip :: Integer -> Property
toBinaryRoundTrip i =
    i > 0 ==>
    binaryToInt (intToBinary i) === Just i

-- TODO #1: The precondition i > 0 in the above property is there
-- because the intToBinary actually fails on negative numbers. It
-- generates an infinite loop!
--   □> intToBinary (-4)
--   "Interrupted.           -- I had to hit Control-C
--
-- So try to fix intToBinary to do something reasonable for negative
-- numbers. We're not going to attempt to do a two's-complement
-- representation, but instead we'll just prefix the string with a
-- minus sign. So here's what the correct behavior would look like:
--   □> intToBinary (-4)
--   "-100"

-- TODO #2: Once negatives are fixed in intToBinary, you can comment
-- out the "i > 0 ==>" in the property. Then you should find that
-- quickCheck toBinaryRoundTrip generates negative numbers as
-- counter-examples:
--   □> quickCheck toBinaryRoundTrip
--   *** Failed! Falsifiable (after 4 tests):
--   -1
--   Nothing /= Just (-1)
--
-- So the solution is to revise binaryToInt so that it can accept the
-- negative sign.

```

```

-- Next we have the QuickCheck property for starting from a string and
-- trying a round-trip. However, using this with the regular QuickCheck
-- means it will mostly try things that are not valid bit strings:
--   □> quickCheck fromBinaryRoundTrip
--   *** Failed! Falsifiable (after 3 tests and 2 shrinks):
--   "a"
--   Nothing /= Just "a"

```

```

fromBinaryRoundTrip :: String -> Property
fromBinaryRoundTrip s =
  (intToBinary <$> binaryToInt s) == Just s

```

```

-- So here's a special string generator that only generates valid bit
-- strings. You can try it like this:
--   □> sample genBits
--   "0"
--   "1"
--   "101"
--   "1000"
--   "100011101"
--   (etc)

```

```

genBits :: Gen String
genBits = fix . map bit <$> arbitrary
  where
    fix [] = "0"
    fix ('0':s) = "10" ++ s
    fix s = s

```

```

-- The 'fix' function makes sure that the string isn't empty, and that
-- it doesn't start with leading zeroes (unless it's JUST 0).

```

```

-- TODO #3: You can run a test incorporating the genBits generator like
-- this:

```

```

--   □> quickCheck (forall genBits fromBinaryRoundTrip)
--   *** Failed! Falsifiable (after 1 test):
--   "0"
--   Just "" /= Just "0"
--

```

```

-- And you may find that it flags the string "0" as a counter-example!
-- The problem is that intToBinary 0 produces "" (the empty string)

```

-- rather than "0" as we had initially. Try to fix that!

```
main :: IO ()
main = do
  putStrLn "TEST int <-> binary"
  quickCheck toBinaryRoundTrip
  putStrLn "TEST binary <-> int"
  quickCheck (forall genBits fromBinaryRoundTrip)
```