1 Problem

The number 3797 has an interesting property. Being prime itself, it is possible to continuously remove digits from left to right, and remain prime at each stage: 3797, 797, 97, and 7. Similarly we can work from right to left: 3797, 379, 37, and 3.

Find the sum of the only eleven primes that are both truncatable from left to right and right to left.

NOTE: 2, 3, 5, and 7 are not considered to be truncatable primes.

2 Solution

```
import Data.List
import qualified Data.Map as Map
import Data.Maybe
import System.Environment
import Data.Numbers.Primes

onePrimes = [1, 2, 3, 5, 7] :: [Int]

buildTruncLeft :: Int → [Int]
buildTruncLeft x =
  if isPrime (fromIntegral x)
    then (x : recLefts)
    else []
      where addLefts = map (λz → read ((show z) ++ (show x))) [1 . . 9] :: [Int]
        primeLefts = filter (isPrime2 ∘ fromIntegral) addLefts
        recLefts      = concat $ map buildTruncLeft primeLefts

buildTruncRight :: Int → [Int]
buildTruncRight x   =
  if isPrime (fromIntegral x)
    then (x : recRights)
    else []
      where addRights = map (λz → read ((show x) ++ (show z))) [0 . . 9] :: [Int]
        primeRights = filter (λz → (isPrime2 ∘ fromIntegral) z ∧
          isTruncLeft z ∧ isTruncRight z) addRights
        recRights      = concat $ map buildTruncRight primeRights

isTruncLeft :: Int → Bool
isTruncLeft x
  | (length $ show x) ≡ 1 = isPrime2 $ fromIntegral x
  | otherwise = (isPrime2 $ fromIntegral x) ∧ (isTruncLeft x′)
    where x′ = read $ (tail ∘ show) x :: Int

isTruncRight :: Int → Bool
isTruncRight x
  | (length $ show x) ≡ 1 = isPrime2 $ fromIntegral x
  | otherwise = (isPrime2 $ fromIntegral x) ∧ (isTruncRight x′)
    where x′ = read $ (reverse ∘ tail ∘ reverse ∘ show) x :: Int

buildAllTruncs :: Int → [Int]
buildAllTruncs x = concat [[x], recTruncs]
  where addRights = map (λz → read ((show z) ++ (show x))) onePrimes :: [Int]
      primeRecs    = filter isTruncLeft $ filter (isPrime ∘ fromIntegral) addRights
      recTruncs    = concat $ map buildAllTruncs primeRecs

primes2 :: [Integer]
primes2 = 2 : filter ((≡ 1) ∘ length ∘ primeFactors) [3, 5 . .]

primeFactors :: Integer → [Integer]
primeFactors n = factor n primes
    where
      factor _ [] = []
      factor m (p : ps) | p ∗ p > m  = [m]
                        | m ‘mod‘ p ≡ 0 = p : factor (m ‘div‘ p) (p : ps)
                        | otherwise = factor m ps

isPrime2 :: Integer → Bool
isPrime2 1 = False
isPrime2 n = case (primeFactors n) of
          (_: _ : _) → False
                  → True
```

3 Result

```
runhaskell problem37.lhs
```