Motivation

```haskell
tossDice :: Rand Int
tossDice = do
    d1 <- dice
    d2 <- dice
    return $ d1 + d2
```

```
[11,8,10,7,11,5,8,4,6,7]
```

```haskell
tossDicePrime :: Rand [Int]
tossDicePrime = weighted $ do
    d <- tossDice
    score $ if prime d then 1 else 0
    return $ d
```

```
[11,7,3,7,7,7,5,5,11,11]
```

```haskell
data Rand x where
    Ret :: x -> Rand x
    Sample01 :: (Float -> Rand x) -> Rand x
    Score :: Float -> Rand x -> Rand x
    Ap :: Rand (a -> x) -> Rand a -> Rand x
instance Functor Rand where
  fmap f (Ret x) = Ret (f x)
  fmap f (Sample01 r2mx) = Sample01 (\r -> fmap f (r2mx r))
  fmap f (Score s mx) = Score s (fmap f mx)
  fmap f (Ap m2x ma) = Ap ((f .) <$> m2x) ma
instance Applicative Rand where
  pure = Ret
  pa2b <*> pa = Ap pa2b pa
instance Monad Rand where
  return = Ret
  (Ret x) >>= x2my = x2my x
  (Sample01 r2mx) >>= x2my = Sample01 (\r -> r2mx r >>= x2my)
  (Score s mx) >>= x2my = Score s (mx >>= x2my)
  (Ap m2x ma) >>= x2my =
    m2x >>= \a2x -> ma >>= \a -> x2my (a2x a)
```