Advanced Programming 2017
Assignment 1

Tom Jager
dgr418@alumni.ku.dk

Tobias Ludwig
fqj315@alumni.ku.dk

September 18, 2017

1 Defining SubsM ...

1.1 ...to be a Functor

```
instance Functor SubsM where
  fmap f fra = SubsM (\c -> case runSubsM fra c of
    (Left e)         -> Left e
    (Right (a,env)) -> Right (f a,env))
```

1.2 ...to be an Apllicative

```
instance Applicative SubsM where
  pure a = return a
  ff <*> fa = SubsM (\c0 -> case runSubsM ff c0 of
    (Left e)          -> Left e
    (Right (f, env)) -> case runSubsM fa (env,snd c0) of
      (Left e')          -> Left e'
      (Right (a, env')) -> Right (f a, env'))
```

1.3 ...to be a Monad

```
instance Monad SubsM where
  return x = SubsM (\c -> Right (x, fst c))
  m >>= f  = SubsM (\c0 -> case runSubsM m c0 of
     (Left e)          -> Left e
     (Right (x,env)) -> case runSubsM (f x) (env,snd c0) of
       (Left e')          -> Left e'
       (Right (x',env')) -> Right (x',env'))
  fail s = error s
```

2 Implementing the Primitives

Primitives are functions from a list of values to an `Either` type. For each operator we generate Errors if the arguments supplied do not match the expected number or types.