# Modellierung von Call-Time Choice als Effekt mittels freier Monaden mit Scope

Niels Bunkenburg

19. Dezember 2018

Arbeitsgruppe für Programmiersprachen und Übersetzerkonstruktion
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

# Motivation

# Grundlagen

## Programme

```
data Prog sig a = Return a
                | Op (sig (Prog sig a))

data Free f a = Pure a
              | Free (f (Free f a))
```

## Programmsignaturen

```
data (sig1 + sig2) cnt = Inl (sig1 cnt)
                       | Inr (sig2 cnt)

class (Functor sub, Functor sup) => sub <: sup where
  inj :: sub a -> sup a
  prj :: sup a -> Maybe (sub a)
```

# Beispieleffekte

Effektfreie Programme

```
data Void cnt
```

```
data VoidProg a = Return a
```

Nichtdeterministische Programme

```
data ND p = Fail | Choice p p
```

```
data NDProg a = Return a
              | Fail
              | Choice (NDProg a) (NDProg a)
```

# Handler

Effekte werden durch Handler verarbeitet.

```
run :: Prog Void a -> a
run (Return x) = x

runND :: (Functor sig)
      => Prog (ND + sig) a -> Prog sig (Tree a)
runND (Return a) = return (Leaf a)
runND Fail       = return Failed
runND (Choice m p q ) = do
  pt <- runND p
  qt <- runND q
  return (Choice m pt qt)
runND (Other op) = Op (fmap runND op)
```

# Beispielprogramme

```
coin :: Prog (ND + Void) Int
coin = Choice (return 0) (return 1)

λ> run . runND $ coin
Choice (Leaf 0) (Leaf 1)

λ> putStrLn . pretty . run . runND $ addM coin coin
?
|---- ?
       |---- 0
       |---- 1
|---- ?
       |---- 1
       |---- 2
```

# Weitere Effekte

```
data State s p = Get (s -> p)
               | Put s p
```


```
data State s p = Get (s -> p)
               | Put s p
```