Monádtranszformátorok

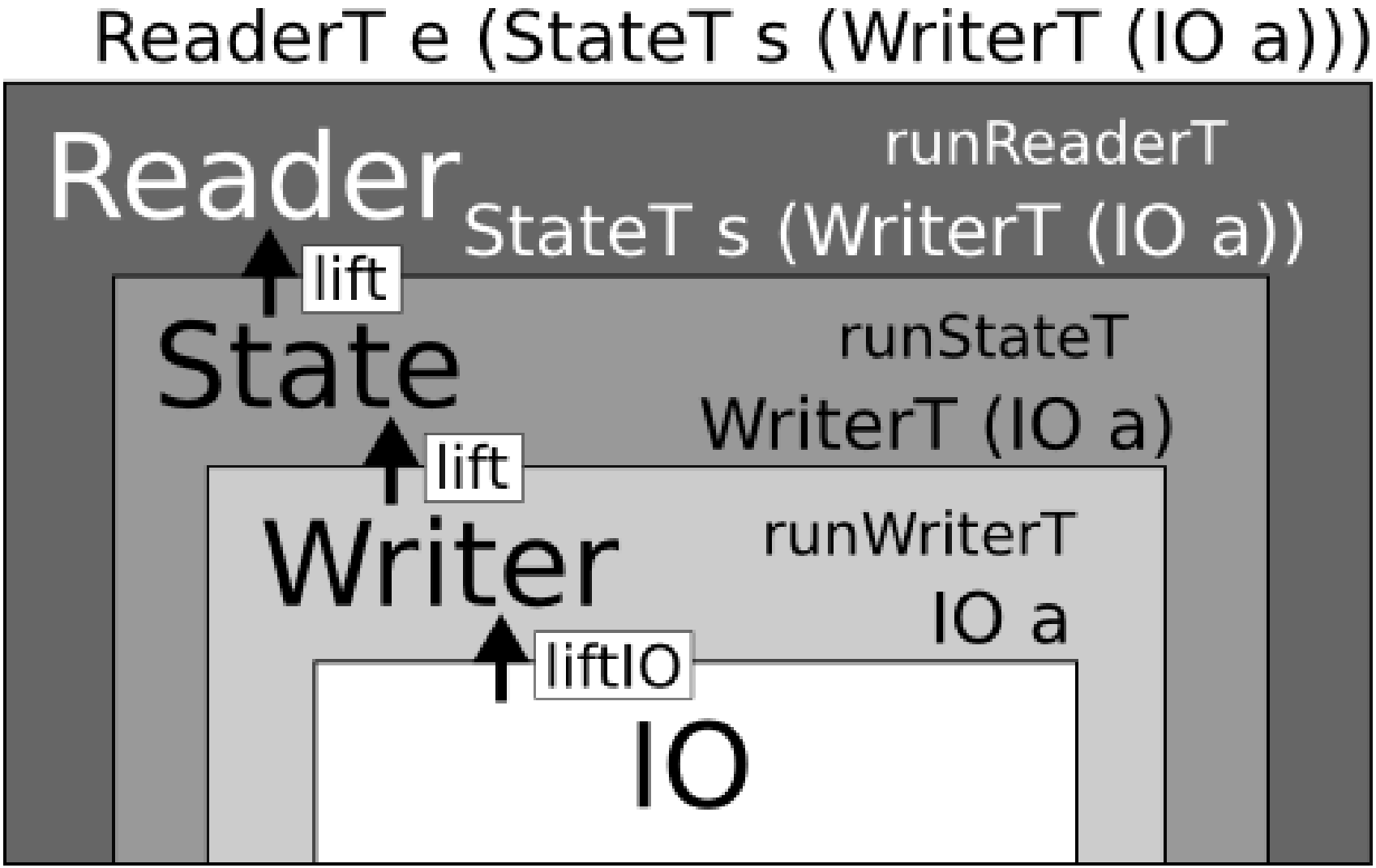
Motiváció: Monádok kombinációja

```
countEntries :: FilePath → K ()
countEntries path = f path 0 where
  f p n = do
    deepestReached ← get
    when (deepestReached < n) (put n)
    c ← liftIO (System.Directory.getDirectoryContents p)
    let contents = filter (notElem` [".", ".."]) c
    tell [(p, length contents)]
    forM_ contents (λ name . do
      let newName = p </> name
      isDir ← liftIO (System.Directory.doesDirectoryExist newName)
      maxDepth ← ask
      when (isDir ∧ (n < maxDepth)) (f newName (n + 1)))

type S = Int
type C = Int
type K = ReaderT C (StateT S (WriterT [(FilePath, Int)] IO))

runStack :: K α → Int → IO ((α, S), [(FilePath, Int)])
runStack k max = runWriterT (runStateT (runReaderT k c) s)
  where (s, c) = (0, max)
```

Monádok rétegelése (stacking)



Részletek: A *MonadTrans* és *MonadIO* osztályok

```
-- Control.Monad.Trans
class MonadTrans (τ :: (⋆ → ⋆) → ⋆ → ⋆) where
  lift :: Monad μ ⇒ μ α → τ μ α
```

Transzformátortörvények:

$$\begin{aligned} \mathit{lift} \circ \mathit{return} &\equiv \mathit{return} \\ \mathit{lift}\,(x \gg\!= f) &\equiv \mathit{lift}\,x \gg\!= (\mathit{lift} \circ f) \end{aligned}$$

Az *IO* esetén ellenben:

```
-- Control.Monad.IO.Class
class Monad μ ⇒ MonadIO (μ :: ⋆ → ⋆) where
  liftIO :: IO α → μ α

instance MonadIO IO where
  liftIO m = m
```

Adaptált törvények:

$$\begin{aligned} \mathit{liftIO} \circ \mathit{return} &\equiv \mathit{return} \\ \mathit{liftIO}\,(x \gg\!= f) &\equiv \mathit{liftIO}\,x \gg\!= (\mathit{liftIO} \circ f) \end{aligned}$$

A *ReaderT* transzformátor (egyszerűsített)

```
class Monad μ ⇒ MonadReader (μ :: ⋆ → ⋆) where
  type EnvType μ -- "associated type (synonym)"
  ask :: μ (EnvType μ)
newtype ReaderT ε μ α = RT (ε → μ α)

runReaderT :: ReaderT ε μ α → ε → μ α
runReaderT (RT t) = t

type Reader ε = ReaderT ε Identity

runReader :: Reader ε α → ε → α
runReader m x = runIdentity (runReaderT m x)

instance Monad μ ⇒ Monad (ReaderT ε μ) where
  return = lift ∘ return
  x >>= f = RT $ λ e . do
    x1 ← runReaderT x e
    runReaderT (f x1) e)
```

A *ReaderT* transzformátor (egyszerűsített, folytatás)

```
instance MonadTrans (ReaderT ε) where
  lift m = RT $ λ e . m

instance Monad μ ⇒ MonadReader (ReaderT ε μ) where
  type EnvType (ReaderT ε μ) = ε
  ask = RT $ λ e . return e -- ask :: (ReaderT ε μ) ε

instance MonadState μ ⇒ MonadState (ReaderT ε μ) where
  type StateType (ReaderT ε μ) = StateType μ
  get = lift get -- get :: μ (StateType μ)
  put = lift ∘ put -- put :: (StateType μ) → μ ()

instance MonadWriter μ ⇒ MonadWriter (ReaderT ε μ) where
  type WriterType (ReaderT ε μ) = WriterType μ
  -- tell :: (WriterType μ) → μ ()
  tell = lift ∘ tell
  -- censor :: (WriterType μ → WriterType μ) → μ α → μ α
  censor f m = RT $ λ e . do
    let m1 = runReaderT m e
    censor f m1
instance MonadIO μ ⇒ MonadIO (ReaderT ε μ) where
  liftIO = lift ∘ liftIO
```

A *StateT* transzformátor (egyszerűsített)

```
class Monad μ ⇒ MonadState μ where
  type StateType μ
  get :: μ (StateType μ)
  put :: (StateType μ) → μ ()
newtype StateT σ μ α = ST (σ → μ (α, σ))

runStateT :: StateT σ μ α → σ → μ (α, σ)
runStateT (ST t) = t

type State σ = StateT σ Identity

runState :: State σ α → σ → (α, σ)
runState m x = runIdentity (runStateT m x)

instance Monad μ ⇒ Monad (StateT σ μ) where
  return x = ST $ λ s . return (x, s))
  x >>= f = ST $ λ s . do
    (x1, s1) ← runStateT x s
    runStateT (f x1) s1)
```

A *StateT* transzformátor (egyszerűsített, folytatás)

```
instance MonadTrans (StateT σ) where
  lift m = ST $ λ s . do
    x ← m
    return (x, s))

instance Monad μ ⇒ MonadState (StateT σ μ) where
  type StateType (StateT σ μ) = σ
  get = ST $ λ s . return (s, s)
  put s = ST $ λ _ . return ((), s)

instance MonadReader μ ⇒ MonadReader (StateT σ μ) where
  type EnvType (StateT σ μ) = EnvType μ
  ask = lift ask

instance MonadWriter μ ⇒ MonadWriter (StateT σ μ) where
  type WriterType (StateT σ μ) = WriterType μ
  tell = lift ∘ tell
  censor f m = ST $ λ s . do
    let m1 = runStateT m s
    censor f m1

instance MonadIO μ ⇒ MonadIO (StateT σ μ) where
  liftIO = lift ∘ liftIO
```

A *WriterT* transzformátor (egyszerűsített)

```
class (Monoid (WriterType μ), Monad μ) ⇒ MonadWriter μ where
  type WriterType μ
  tell    :: (WriterType μ) → μ ()
  censor  :: (WriterType μ → WriterType μ) → μ α → μ α
newtype WriterT ω μ α = WT (μ (α, ω))

runWriterT :: WriterT ω μ α → μ (α, ω)
runWriterT (WT m) = m

type Writer α = WriterT α Identity

runWriter :: Writer ω α → (α, ω)
runWriter m = runIdentity (runWriterT m)

instance (Monoid ω, Monad μ) ⇒ Monad (WriterT ω μ) where
  return x = WT $ return (x, mempty)
  x >>= f = WT $ do
    (x1, w1) ← runWriterT x
    (x2, w2) ← runWriterT (f x1)
    return (x2, w1 <> w2)
```

A *WriterT* transzformátor (egyszerűsített, folytatás)

```
instance Monoid ω ⇒ MonadTrans (WriterT ω) where
  lift m = WT $ do
    x ← m
    return (x, mempty)

instance (Monoid ω, MonadState μ) ⇒ MonadState (WriterT ω μ) where
  type StateType (WriterT ω μ) = StateType μ
  get = lift get
  put = lift ∘ put

instance (Monoid ω, MonadReader μ) ⇒ MonadReader (WriterT ω μ) where
  type EnvType (WriterT ω μ) = EnvType μ
  ask = lift ask

instance (Monoid ω, Monad μ) ⇒ MonadWriter (WriterT ω μ) where
  type WriterType (WriterT ω μ) = ω
  tell w = WT $ return ((), w)
  censor f m = WT $ do
    (x, w) ← runWriterT m
    return (x, f w)

instance (Monoid ω, MonadIO μ) ⇒ MonadIO (WriterT ω μ) where
  liftIO = lift ∘ liftIO
```

A monádrétegek sorrendje (példa)

```
stack :: (MonadState Int μ, MonadWriter [String] μ, MonadError String μ) ⇒ μ ()
stack = do
  r ← get
  tell ["Hello!"]
  throwError "Error!"
  tell ["Value: " ++ show r]
  put (r + 1)

type A = ExceptT String (WriterT [String] (State Int))
type B = StateT Int (WriterT [String] (Either String))

stackA :: A ()
stackA = stack

stackB :: B ()
stackB = stack

runA = runState (runWriterT (runExceptT stackA)) 0
runB = runWriterT (runStateT stackB 0)
```

Tulajdonságalapú tesztelés

QuickCheck

A QuickCheck Haskell programok automatikus, tulajdonság-alapú tesztelésére használható.

- *Programspecifikáció*: program által teljesítendő tulajdonságok
- Nagy számú, a *tulajdonságok alapján generált* tesztesetek
- A specifikáció *Haskellben adható meg*, kombinátorokkal
- További *kombinátorok*: tulajdonságok definiálása, eredmények elemzése, tesztadatok előállítása
- Megvalósítás: beágyazott szakterületspecifikus nyelv (*embedded domain-specific language*), amely viszont tetszőleges nyelven írt program teszteléséhez használható!

"The coolest example of type classes that I know" – Simon Peyton Jones ("Adventure with Types in Haskell", `https://www.youtube.com/watch?v=6COvD8oynmI`)

A QuickCheck (*Test.QuickCheck*) használata

$$
\begin{aligned}
&\mathit{flickSort} :: \mathit{Ord}\ \alpha \Rightarrow [\alpha] \rightarrow [\alpha] \\
&\mathit{flickSort}\ [] = [] \\
&\mathit{flickSort}\ (x : y : \mathit{xs}) = (y : x : \mathit{xs}) \\
&\mathit{flickSort}\ \mathit{xs} = \mathit{sort}\ \mathit{xs} \\
&\text{-- prop\_idempotent :: [Int] } \rightarrow \text{ Bool} \\
&\mathit{prop\_idempotent}\ \mathit{xs} = \mathit{flickSort}\ (\mathit{flickSort}\ \mathit{xs}) \equiv \mathit{flickSort}\ \mathit{xs}
\end{aligned}
$$

Tesztelés:

```
--  quickCheck :: Testable prop => prop -> IO ()
GHCi> quickCheck prop_idempotent
+++ OK, passed 100 tests.
```

További tulajdonságok:

$$
\begin{aligned}
&\mathit{prop\_ordered}\ \mathit{xs} = \mathit{ordered}\ (\mathit{flickSort}\ \mathit{xs}) \\
&\quad \textbf{where}\ \mathit{ordered}\ \mathit{xs} = \mathit{and}\ (\mathit{zipWith}\ (<)\ \mathit{xs}\ (\mathit{tail}\ \mathit{xs})) \\
&\mathit{prop\_sortModel}\ \mathit{xs} = (\mathit{sort}\ \mathit{xs}) \equiv (\mathit{flickSort}\ \mathit{xs}) \\
&\mathit{prop\_append}\ \mathit{xs}\ \mathit{ys} = \mathit{not}\ (\mathit{null}\ \mathit{xs}) \supset \mathit{not}\ (\mathit{null}\ \mathit{ys}) \supset \\
&\quad \mathit{head}\ (\mathit{flickSort}\ (\mathit{xs} +\!\!+ \mathit{ys})) \equiv (\mathit{minimum}\ \mathit{xs})\ `\mathit{min}`\ (\mathit{minimum}\ \mathit{ys})
\end{aligned}
$$

Bővebben: `http://hackage.haskell.org/package/QuickCheck`