TARTU ÜLIKOOL
1632

Monaadid ja kõrvalefektid

Danel Ahman ja Varmo Vene

Funktsioonid Idrises

TARTU ÜLIKOOL

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund

Funktsioonid Idrises

TARTU ÜLIKOOL 1632

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine
  - mälu kasutus

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine
  - mälu kasutus
  - muudetavad (ingl *mutable*) muutujad

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine
  - mälu kasutus
  - muudetavad (ingl *mutable*) muutujad
  - mittedeterminism

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine
  - mälu kasutus
  - muudetavad (ingl *mutable*) muutujad
  - mittedeterminism
  - tõenäosused

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine
  - mälu kasutus
  - muudetavad (ingl *mutable*) muutujad
  - mittedeterminism
  - tõenäosused
  - ...

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus
  - muudetavad (ingl *mutable*) muutujad
  - mittedeterminism
  - tõenäosused
  - ...

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad
  - mittedeterminism
  - tõenäosused
  - ...

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad (`f : Nat -> ???`)
  - mittedeterminism
  - tõenäosused
  - ...

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad (`f : Nat -> ???`)
  - mittedeterminism (`f : Nat -> ???`)
  - tõenäosused
  - ...

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad (`f : Nat -> ???`)
  - mittedeterminism (`f : Nat -> ???`)
  - tõenäosused (`f : Nat -> ???`)
  - ...

Funktsioonid Idrises

TARTU ÜLIKOOL 1632

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad (`f : Nat -> ???`)
  - mittedeterminism (`f : Nat -> ???`)
  - tõenäosused (`f : Nat -> ???`)
  - ...

} **monaadid**

Funktsioonid Idrises

- Vaikimisi on Idrise funktsioonid **puhtad**, **ilma kõrvalefektideta**, **matemaatilised**
  - `f : Nat -> Bool`
- Samas on päris programmides aga tihti kasulikud ka **kõrvalefektid**
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (ingl *exceptions*) ja nende töötlemine (`f : Nat -> ???`)
  - mälu kasutus (`f : Nat -> ???`)
  - muudetavad (ingl *mutable*) muutujad (`f : Nat -> ???`)
  - mittedeterminism (`f : Nat -> ???`)
  - tõenäosused (`f : Nat -> ???`)
  - ...

monaadid

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetilised avaldised

```
infixl 10 :+:, :-:
infixl 11 :*:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetilised avaldised

```
infixl 10 :+:, :-:
infixl 11 :*:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (nende semantika arvutamine)

```
eval : Expr -> Int

eval (Num i) = i

eval (e1 :+: e2) = eval e1 + eval e2
eval (e1 :-: e2) = eval e1 - eval e2
eval (e1 :*: e2) = eval e1 * eval e2
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (nende semantika arvutamine)

```
eval : Expr -> Int

eval (Num i) = i

eval (e1 :+: e2) = eval e1 + eval e2
eval (e1 :-: e2) = eval e1 - eval e2
eval (e1 :*: e2) = eval e1 * eval e2
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (nende semantika arvutamine)

```
eval : Expr -> Int

eval (Num i) = i

eval (e1 :+: e2) = eval e1 + eval e2
eval (e1 :-: e2) = eval e1 - eval e2
eval (e1 :*: e2) = eval e1 * eval e2
```

- Väljund

```
Loeng14a> eval exp1
3
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (nende semantika arvutamine)

```
eval : Expr -> Int

eval (Num i) = i

eval (e1 :+: e2) = eval e1 + eval e2
eval (e1 :-: e2) = eval e1 - eval e2
eval (e1 :*: e2) = eval e1 * eval e2
```

- Väljund

```
Loeng14a> eval exp1
3
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)
```

Näide 1: Aritmeetiliste avaldiste väärtustamine

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (nende semantika arvutamine)

```
eval : Expr -> Int

eval (Num i) = i

eval (e1 :+: e2) = eval e1 + eval e2
eval (e1 :-: e2) = eval e1 - eval e2
eval (e1 :*: e2) = eval e1 * eval e2
```

- Väljund

```
Loeng14a> eval exp1
3

Loeng14a> eval exp2
6
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL 1632

- Aritmeetilised avaldised (koos **jagamisega**)

```
infixl 10 :+:, :-:
infixl 11 :*:, :/:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
          | (:/:) Expr Expr
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL 1632

- Aritmeetilised avaldised (koos **jagamisega**)

```
infixl 10 :+:, :-:
infixl 11 :*:, :/:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
          | (:/:) Expr Expr
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

- Aritmeetilised avaldised (koos **jagamisega**)

```
infixl 10 :+:, :-:
infixl 11 :*:, :/:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
          | (:/:) Expr Expr
```

- Näited

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)
```

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

Näide 2: Avaldiste väärtustamine (koos jagamisega)

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

- Näited

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL 1632

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

- Väljund

```
Loeng14b> eval1 exp3
6
```

- Näited

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

- Väljund

```
Loeng14b> eval1 exp3
6
```

- Näited

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL 1632

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

- Väljund

```
Loeng14b> eval1 exp3
6

Loeng14b> eval1 exp4
let False = True in
prim__div_Int 12 0
```

- Näited

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e
```

Viga, mida ei saa edasi töödelda!

(Error, mitte Exception)

- Väljund

```
Loeng14b> eval1 exp3
6

Loeng14b> eval1 exp4
let False = True in
prim__div_Int 12 0
```

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)
```

Näide 2: Avaldiste väärtustamine (koos jagamisega)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (koos **jagamisega**)

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + ev
eval1 (e1 :-: e2) = eval1 e1 - ev
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e
```

```
Integral Integer where
  div x y
      = case y == 0 of
              False => prim__div_Integer x y
```

Viga, mida ei saa edasi töödelda!

(Error, mitte Exception)

- Väljund

```
Loeng14b> eval1 exp3
6

Loeng14b> eval1 exp4
let False = True in
prim__div_Int 12 0
```

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

TARTU ÜLIKOOL

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

```
eval2 : Expr -> Option Int
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) = case eval2 e1 of
                      None    => None
                      Some v1 => case eval2 e2 of
                                   None    => None
                                   Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) = ...
eval2 (e1 :*: e2) = ...
eval2 (e1 :/: e2) = ...
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

```
eval2 : Expr -> Option Int

...
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtustamine** (koos jagamise ja **eranditega**)

```
eval2 : Expr -> Option Int

...

eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => if v2 == 0
                            then None
                            else Some (v1 `div` v2)
```

Näide 3: Avaldiste väärtustamine (koos eranditega)

TARTU ÜLIKOOL

- **Osaliste väärtuste** andmetüüp (isomorfne Maybe andmetüübiga standardteegis)

```
data Option a = None
              | Some a
```

- Aritmeetiliste avaldiste **väärtust**

```
eval2 : Expr -> Option Int

...

eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 o
                 None    => Nor
                 Some v2 => if v2 == 0
                            then None
                            else Some (v1 `div` v2)
```

```
exp3 : Expr
exp3 = Num 4 :*: Num 3 :/: Num 2

Loeng14b> eval2 exp3
Some 6

exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)

Loeng14b> eval2 exp4
None
```

Palju koodi duplitseerimist

TARTU ÜLIKOOL

Palju koodi duplitseerimist

TARTU ÜLIKOOL

- **Ilma eranditeta** väärtustaja

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

Palju koodi duplitseerimist

TARTU ÜLIKOOL

- **Ilma eranditeta** väärtustaja

```
eval1 : Expr -> Int

eval1 (Num i) = i

eval1 (e1 :+: e2) = eval1 e1 + eval1 e2
eval1 (e1 :-: e2) = eval1 e1 - eval1 e2
eval1 (e1 :*: e2) = eval1 e1 * eval1 e2
eval1 (e1 :/: e2) = eval1 e1 `div` eval1 e2
```

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL

- **Korduvad mustrid**

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL 1632

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
  - kui üks ebaõnnestub, siis ebaõnnestub kogu avaldise väärtustamine

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL 1632

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
  - kui üks ebaõnnestub, siis ebaõnnestub kogu avaldise väärtustamine

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
  - kui üks ebaõnnestub, siis ebaõnnestub kogu avaldise väärtustamine
  - õnnestumise korral antakse tulemus edasi järgnevale arvutusele
- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
  - kui üks ebaõnnestub, siis ebaõnnestub kogu avaldise väärtustamine
  - õnnestumise korral antakse tulemus edasi järgnevale arvutusele

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Samas palju sarnaseid korduvaid mustreid

TARTU ÜLIKOOL

- **Korduvad mustrid**
  - väärtuse tagastamine ilma erandita
  - alamavaldiste väärtustamine toimub järjestikku
  - kui üks ebaõnnestub, siis ebaõnnestub kogu avaldise väärtustamine
  - õnnestumise korral antakse tulemus edasi järgnevale arvutusele
  - vajadusel erandi tõstatamine

- **Eranditega** väärtustaja

```
eval2 : Expr -> Option Int

eval2 (Num i) = Some i

eval2 (e1 :+: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 + v2)
eval2 (e1 :-: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 - v2)
eval2 (e1 :*: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 => Some (v1 * v2)
eval2 (e1 :/: e2) =
  case eval2 e1 of
    None    => None
    Some v1 => case eval2 e2 of
                 None    => None
                 Some v2 =>
                   if v2 == 0 then None else Some (v1 `div` v2)
```

Korduvad mustrid abstraktsemalt

TARTU ÜLIKOOL

Korduvad mustrid abstraktsemalt

TARTU ÜLIKOOL 1632

- **Väärtuse tagastamine** ilma erandit tõstatamata

```
oReturn : a -> Option a
oReturn x = Some x
```

Korduvad mustrid abstraktsemalt

- **Väärtuse tagastamine** ilma erandit tõstatamata

```
oReturn : a -> Option a
oReturn x = Some x
```

- Eranditega arvutuste **järjestikku jooksutamine**

```
oBind : Option a -> (a -> Option b) -> Option b

oBind comp f = case comp of
                 None   => None
                 Some x => f x
```

Korduvad mustrid abstraktsemalt

- **Väärtuse tagastamine** ilma erandit tõstatamata

```
oReturn : a -> Option a
oReturn x = Some x
```

- Eranditega arvutuste **järjestikku jooksutamine**

```
oBind : Option a -> (a -> Option b) -> Option b

oBind comp f = case comp of
                 None   => None
                 Some x => f x
```

- Vajadusel **erandi tõstatamine**

```
oThrow : Option a
oThrow = None
```

Näide 3: Avaldiste väärtustamine (abstraktsemalt)

TARTU ÜLIKOOL

Näide 3: Avaldiste väärtustamine (abstraktsemalt)


```
eval3 : Expr -> Option Int

eval3 (Num i) = oReturn i

eval3 (e1 :+: e2) = eval3 e1 `oBind` \ v1 =>
                    eval3 e2 `oBind` \ v2 =>
                    oReturn (v1 + v2)

eval3 (e1 :-: e2) = eval3 e1 `oBind` \ v1 =>
                    eval3 e2 `oBind` \ v2 =>
                    oReturn (v1 - v2)

eval3 (e1 :*: e2) = eval3 e1 `oBind` \ v1 =>
                    eval3 e2 `oBind` \ v2 =>
                    oReturn (v1 * v2)

eval3 (e1 :/: e2) = eval3 e1 `oBind` \ v1 =>
                    eval3 e2 `oBind` \ v2 =>
                    if v2 == 0 then oThrow else oReturn (v1 `div` v2)
```

Option tüüp on monaad!

TARTU ÜLIKOOL

Option tüüp on monaad!

TARTU ÜLIKOOL

- **Option tüüp** kirjeldab potentsiaalselt erandeid tõstatavaid arvutusi
  - kus saab ilma erandita **väärtusi tagastada**
  - mida saab **järjestikku käivitada**
  - kus saab **vajadusel erandit tõstatada**

Option tüüp on monaad!

TARTU ÜLIKOOL

- **Option tüüp** kirjeldab potentsiaalselt erandeid tõstatavaid arvutusi
  - kus saab ilma erandita **väärtusi tagastada**
  - mida saab **järjestikku käivitada**
  - kus saab **vajadusel erandit tõstatada**
- **Monaadid** kirjeldavad üldisemalt arvutusi
  - kus saab **väärtusi tagastada**
  - mida saab **järjestikku käivitada**
  - kus võib saada **käivitada spetsiifilisi operatsioone**

Monaadid

Monaadid

- **Monaadid** üldiselt (Idrise tüübiklassina)

```
interface Monaad (0 m : Type -> Type) where
  return : a -> m a
  bind   : m a -> (a -> m b) -> m b                    (>>=, do-notatsioon)
```

Monaadid

- **Monaadid** üldiselt (Idrise tüübiklassina)

```
interface Monaad (0 m : Type -> Type) where
  return : a -> m a
  bind   : m a -> (a -> m b) -> m b                    (>>=, do-notatsioon)
```

- + **võrduslikud seadused** `return`i ja `bind`i kohta, mida FP keeltes tihti ei kontrollita
  - `bind (return x) f = f x`
  - `bind c (\ x => return x) = c`
  - `bind (bind c f) g = bind c (\ x => bind (f x) g)`

Monaadid

- **Monaadid** üldiselt (Idrise tüübiklassina)

```
interface Monaad (0 m : Type -> Type) where
  return : a -> m a
  bind   : m a -> (a -> m b) -> m b                    (>>=, do-notatsioon)
```

- + **võrduslikud seadused** `return`i ja `bind`i kohta, mida FP keeltes tihti ei kontrollita
  - `bind (return x) f = f x`
  - `bind c (\ x => return x) = c`
  - `bind (bind c f) g = bind c (\ x => bind (f x) g)`

- Option monaad **eranditega programmeerimiseks**

```
Monaad Option where
  return = oReturn
  bind   = oBind
```

Monaadid Idrises

Monaadid Idrises

- Idrises on ka **sisseehitatud monaadide tüübiklass**

```
interface Functor f where
  map : (a -> b) -> f a -> f b

interface Functor f => Applicative f where
  pure : a -> f a
  (<*>) : f (a -> b) -> f a -> f b

interface Applicative m => Monad m where
  (>>=) : m a -> (a -> m b) -> m b
```

Monaadid Idrises

- Idrises on ka **sisseehitatud monaadide tüübiklass**

```
interface Functor f where
  map : (a -> b) -> f a -> f b

interface Functor f => Applicative f where
  pure : a -> f a
  (<*>) : f (a -> b) -> f a -> f b

interface Applicative m => Monad m where
  (>>=) : m a -> (a -> m b) -> m b
```

sisseehitatud monaadide tüübiklass

Monaadid Idrises

- Idrises on ka **sisseehitatud monaadide tüübiklass**

```
interface Functor f where
  map : (a -> b) -> f a -> f b

interface Functor f => Applicative f where
  pure : a -> f a
  (<*>) : f (a -> b) -> f a -> f b

interface Applicative m => Monad m where
  (>>=) : m a -> (a -> m b) -> m b
```

← sisseehitatud monaadide tüübiklass

- Niimoodi mitmekihiliselt monaadide konstrueerimine on **tihti liiga kohmakas**!

Monaadid Idrises

- Idrises on ka **sisseehitatud monaadide tüübiklass**

```
interface Functor f where
  map : (a -> b) -> f a -> f b

interface Functor f => Applicative f where
  pure : a -> f a
  (<*>) : f (a -> b) -> f a -> f b

interface Applicative m => Monad m where
  (>>=) : m a -> (a -> m b) -> m b
```

sisseehitatud monaadide tüübiklass

- Niimoodi mitmekihiliselt monaadide konstrueerimine on **tihti liiga kohmakas**!
- Selle aine näidetes ja ülesannetes defineerime monaadid **oma tüübiklassi abil**

```
Monaad m => Functor m     where ...
Monaad m => Applicative m where ...
Monaad m => Monad m       where ...
```

```
interface Monaad (0 m : Type -> Type) where
  return : a -> m a
  bind   : m a -> (a -> m b) -> m b
```

Näide 3: Avaldiste väärtustamine (do-notatsiooniga)

TARTU ÜLIKOOL

Näide 3: Avaldiste väärtustamine (do-notatsiooniga)

TARTU ÜLIKOOL 1632

- Kuna
  - näitasime, et Option on Monaad
  - iga Monaad on Monad
  - iga Monad'i puhul saame kasutada **do-notatsiooni**!
  - täpselt nagu IO programmides

Näide 3: Avaldiste väärtustamine (do-notatsiooniga)

TARTU ÜLIKOOL

```
eval4 : Expr -> Option Int

eval4 (Num i) = return i

eval4 (e1 :+: e2) = do
  v1 <- eval4 e1
  v2 <- eval4 e2
  return (v1 + v2)

eval4 (e1 :-: e2) = do
  v1 <- eval4 e1
  v2 <- eval4 e2
  return (v1 - v2)

eval4 (e1 :*: e2) = do
  v1 <- eval4 e1
  v2 <- eval4 e2
  return (v1 * v2)

eval4 (e1 :/: e2) = do
  v1 <- eval4 e1
  v2 <- eval4 e2
  if v2 == 0 then throw else return (v1 `div` v2)
```

- Kuna
  - näitasime, et Option on Monaad
  - iga Monaad on Monad
  - iga Monad'i puhul saame kasutada **do-notatsiooni**!
  - täpselt nagu IO programmides

Näide 3: Avaldiste väärtustamine (!-notatsiooniga)

Näide 3: Avaldiste väärtustamine (!-notatsiooniga)

TARTU ÜLIKOOL

- Kuna
  - näitasime, et Option on Monaad
  - iga Monaad on Monad
  - iga Monad'i puhul saame kasutada ka **!-notatsiooni**

Näide 3: Avaldiste väärtustamine (!-notatsiooniga)

TARTU ÜLIKOOL 1632

```
eval5 : Expr -> Option Int

eval5 (Num i) = return i

eval5 (e1 :+: e2) =
  return (!(eval5 e1) + !(eval5 e2))

eval5 (e1 :-: e2) =
  return (!(eval5 e1) - !(eval5 e2))

eval5 (e1 :*: e2) =
  return (!(eval5 e1) * !(eval5 e2))

eval5 (e1 :/: e2) =
  if !(eval5 e2) == 0 then throw else return (!(eval5 e1) `div` !(eval5 e2))
```

- Kuna
  - näitasime, et Option on Monaad
  - iga Monaad on Monad
  - iga Monad'i puhul saame kasutada ka **!-notatsiooni**

Näide 3: Avaldiste väärtustamine (!-notatsiooniga)

TARTU ÜLIKOOL

```
eval5 : Expr -> Option Int

eval5 (Num i) = return i

eval5 (e1 :+: e2) =
  return (!(eval5 e1) + !(eval5 e2))

eval5 (e1 :-: e2) =
  return (!(eval5 e1) - !(eval5 e2))

eval5 (e1 :*: e2) =
  return (!(eval5 e1) * !(eval5 e2))

eval5 (e1 :/: e2) =
  if !(eval5 e2) == 0 then throw else return (!(eval5 e1) `div` !(eval5 e2))
```

- Kuna
  - näitasime, et Option on Monaad
  - iga Monaad on Monad
  - iga Monad'i puhul saame kasutada ka **!-notatsiooni**

- **!-notatsioon:** kui defineerime arvutust `f : m a`, kus `m` on Monad, siis
  - kui `g : m b`, siis `!g : b` (aga ainult monaadilise arvutuse sees, st., `m`'i all)
  - taustal teisendatakse ! kasutamised do-notatsiooniga järjestatud arvutusteks

Vigade töötlemine

Vigade töötlemine

TARTU ÜLIKOOL

- Saame modelleerida ka **Java/Pythoni-stiilis erandite töötlemist**

```
tryCatch : Option a -> Option a -> Option a

tryCatch comp excHandler = case comp of
                             None   => excHandler
                             Some x => return x
```

Vigade töötlemine

TARTU ÜLIKOOL

- Saame modelleerida ka **Java/Pythoni-stiilis erandite töötlemist**

```
tryCatch : Option a -> Option a -> Option a

tryCatch comp excHandler = case comp of
                             None   => excHandler
                             Some x => return x
```

- Näiteks

```
exp4 : Expr
exp4 = Num 4 :*: Num 3 :/: (Num 2 :-: Num 2)

Loeng14b> eval2 exp4
None

Loeng14b> tryCatch (eval2 exp4) (return 42)
Some 42
```

Vigade töötlemine

- Saame modelleerida ka **Java/Pythoni-stiilis erandite töötlemist**

```
tryCatch : Option a -> Option a -> Option a

tryCatch comp excHandler
```

- Näiteks

```
exp4 : Expr
exp4 = Num 4 :*: Num 3 :/

Loeng14b> eval2 exp4
None

Loeng14b> tryCatch (eval2
Some 42
```

Option on erijuht **rohkemate eranditega monaadist**

```
data Exc e a = MkExc (Either e a)

eReturn : a -> Exc e a
eReturn x = MkExc (Right x)

eBind : Exc e a -> (a -> Exc e b) -> Exc e b
eBind comp f = case comp of
                    MkExc (Left e)  => MkExc (Left e)
                    MkExc (Right x) => f x
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

TARTU ÜLIKOOL

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

- Vaatame jälle **ilma jagamiseta** (ilma eranditeta) aritmeetilisi avaldisi

```
infixl 10 :+:, :-:
infixl 11 :*:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

- Vaatame jälle **ilma jagamiseta** (ilma eranditeta) aritmeetilisi avaldisi

```
infixl 10 :+:, :-:
infixl 11 :*:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
```

- Eesmärk: defineerida väärtustaja, mis **tagastab **ka** väärtustatud avaldiste arvu**
  - lihtne näide arvutuste/programmide instrumenteerimisest
  - teised sarnased näited: tegevuste logimine, autentimine, dünaamiline verifitseerimine

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

- Vaatame jälle **ilma jagamiseta** (ilma eranditeta) aritmeetilisi avaldisi

```
infixl 10 :+:, :-:
infixl 11 :*:

data Expr = Num Int
          | (:+:) Expr Expr
          | (:-:) Expr Expr
          | (:*:) Expr Expr
```

- Eesmärk: defineerida väärtustaja, mis **tagastab **ka** väärtustatud avaldiste arvu**
  - lihtne näide arvutuste/programmide instrumenteerimisest
  - teised sarnased näited: tegevuste logimine, autentimine, dünaamiline verifitseerimine
- Näitame, et selline väärtustamine on ka **näide monaadilistest arvutustest**!

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

TARTU ÜLIKOOL

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
Counter = Int

eval1 : Expr -> (Int,Counter)
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
Counter = Int

eval1 : Expr -> (Int,Counter)

eval1 (Num i) = (i,0)

eval1 (e1 :+: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 + v2 , c1 + c2 + 1)

eval1 (e1 :-: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 - v2 , c1 + c2 + 1)

eval1 (e1 :*: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 * v2 , c1 + c2 + 1)
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
Counter = Int

eval1 : Expr -> (Int,Counter)

eval1 (Num i) = (i,0)

eval1 (e1 :+: e2) = case eval1 e1 of
                      (v1,c1) => case
                                   (v2

eval1 (e1 :-: e2) = case eval1 e1 of
                      (v1,c1) => case
                                   (v2

eval1 (e1 :*: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 * v2 , c1 + c2 + 1)
```

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)

Loeng14c> eval1 exp1
(3, 1)

Loeng14c> eval1 exp2
(6, 2)
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

TARTU ÜLIKOOL

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
Counter = Int

eval1 : Expr -> (Int,Counter)

eval1 (Num i) = (i,0)

eval1 (e1 :+: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 + v2 , c1 + c2 + 1)

eval1 (e1 :-: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 - v2 , c1 + c2 + 1)

eval1 (e1 :*: e2) = case eval1 e1 of
                      (v1,c1) => case eval1 e2 of
                                   (v2,c2) => (v1 * v2 , c1 + c2 + 1)
```

Näide 4: Avaldiste väärtustamine (instrumenteeritult)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
Counter = Int

eval1 : Expr -> (Int,Counter)

eval1 (Num i) = (i,0)

eval1 (e1 :+: e2) = case eval1
                       (v1,c1) =


eval1 (e1 :-: e2) = case eval1 e1 of
                       (v1,c1) => case eval1 e2 of
                                    (v2,c2) => (v1 - v2 , c1 + c2 + 1)

eval1 (e1 :*: e2) = case eval1 e1 of
                       (v1,c1) => case eval1 e2 of
                                    (v2,c2) => (v1 * v2 , c1 + c2 + 1)
```

- Samasugused **korduvad mustrid** nagu varem:
  - väärtuste tagastamine
  - arvutuste järjestikku käivitamine
  - instrueerimine (loendurile +1 tegemine)

Korduvad mustrid instrumenteerimisel

TARTU ÜLIKOOL

Korduvad mustrid instrumenteerimisel

TARTU ÜLIKOOL 1632

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

Korduvad mustrid instrumenteerimisel

TARTU ÜLIKOOL 1632

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi väärtustamata (loendur on 0)

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

Korduvad mustrid instrumenteerimisel

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi väärtustamata (loendur on 0)

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

- Instrumenteeritud **arvutuste järjestikku jooksutamine**

```
cBind : Ctr a -> (a -> Ctr b) -> Ctr b
cBind comp f = case comp of
                  MkCtr (x,c1) => case (f x) of
                                    MkCtr (y,c2) => MkCtr (y , c1 + c2)
```

Korduvad mustrid instrumenteerimisel

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi väärtustamata (loendur on 0)

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

- Instrumenteeritud **arvutuste järjestikku jooksutamine**

```
cBind : Ctr a -> (a -> Ctr b) -> Ctr b
cBind comp f = case comp of
                  MkCtr (x,c1) => case (f x) of
                                    MkCtr (y,c2) => MkCtr (y , c1 + c2)
```

- Avaldiste **väärtustamise arvu loendamine**

```
cCount : Ctr ()
cCount = MkCtr ((), 1)
```

Korduvad mustrid instrumenteerimisel

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi väärtustamata (loendur on 0)

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

- Instrumenteeritud **arvutuste järjestikku jooksutamine**

```
cBind : Ctr a -> (a -> Ctr b) -> Ctr b
cBind comp f = case comp of
                 MkCtr (x,c1) => case (f x) of
                                   MkCtr (y,c2) => MkCtr (y , c1 + c2)
```

- Avaldiste **väärtustamise arvu loendamine**

```
cCount : Ctr ()
cCount = MkCtr ((), 1)
```

```
Monaad Ctr where
  return = cReturn
  bind   = cBind
```

Korduvad mustrid instrumenteerimisel

TARTU ÜLIKOOL

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi vä

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

- Instrumenteeritud **arvutuste järjestik**

```
cBind : Ctr a -> (a -> Ctr b) -> Ctr
cBind comp f = case comp of
                 MkCtr (x,c1) => case
                                   Mk
```

- Avaldiste **väärtustamise arvu loenda**

```
cCount : Ctr ()
cCount = MkCtr ((), 1)
```

`Counter` ei pea olema ainult `Int`

Võib parametriseerida suvalise Monoid-iga

```
interface Semigroup ty where
  (<+>) : ty -> ty -> ty

interface Semigroup ty => Monoid ty where
  neutral : ty
```

Saame nn üldise **kirjutajamonaadi**
(vt Idrise koodi)

Korduvad mustrid instrumenteerimisel

- Instrumenteeritud arvutuste tüüp

```
data Ctr a = MkCtr (a,Counter)
```

- **Väärtuse tagastamine** ilma avaldisi väärtustamata (loendur on 0)

```
cReturn : a -> Ctr a
cReturn x = MkCtr (x,0)
```

- Instrumenteeritud **arvutuste järjestikku jooksutamine**

```
cBind : Ctr a -> (a -> Ctr b) -> Ctr b
cBind comp f = case comp of
                 MkCtr (x,c1) => case (f x) of
                                   MkCtr (y,c2) => MkCtr (y , c1 + c2)
```

- Avaldiste **väärtustamise arvu loendamine**

```
cCount : Ctr ()
cCount = MkCtr ((), 1)
```

```
Monaad Ctr where
  return = cReturn
  bind   = cBind
```

Näide 4: Avaldiste väärtustamine (do-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

Näide 4: Avaldiste väärtustamine (do-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count                    ← avaldise väärtustamise loendamine
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count                    ← avaldise väärtustamise loendamine
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count                    ← avaldise väärtustamise loendamine
  return (v1 * v2)
```

Näide 4: Avaldiste väärtustamine (do-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

Näide 4: Avaldiste väärtustamine (do-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

```
exp1 : Expr
exp1 = Num 1 :+: Num 2

Loeng14c> eval3 exp1
MkCtr (3, 1)

exp2 : Expr
exp2 = Num 2 :*: (Num 4 :-: Num 1)

Loeng14c> eval3 exp2
MkCtr (6, 2)
```

Näide 4: Avaldiste väärtustamine (do-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

Näide 4: Avaldiste väärtustamine (!-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

```
eval4 : Expr -> Ctr Int

eval4 (Num i) = return i

eval4 (e1 :+: e2) = do
  count
  return (!(eval4 e1) + !(eval4 e2))

eval4 (e1 :-: e2) = do
  count
  return (!(eval4 e1) - !(eval4 e2))

eval4 (e1 :*: e2) = do
  count
  return (!(eval4 e1) * !(eval4 e2))
```

Näide 4: Avaldiste väärtustamine (!-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

```
eval4 : Expr -> Ctr Int

eval4 (Num i) = return i

eval4 (e1 :+: e2) = do
  count
  return (!(eval4 e1) + !(eval4 e2))

eval4 (e1 :-: e2) = do
  count
  return (!(eval4 e1) - !(eval4 e2))

eval4 (e1 :*: e2) = do
  count
  return (!(eval4 e1) * !(eval4 e2))
```

NB: `count` toimub enne `eval4` väljakutseid

Näide 4: Avaldiste väärtustamine (!-notatsiooniga)

- Aritmeetiliste avaldiste **väärtustamine** (**instrumenteeritult**)

```
eval3 : Expr -> Ctr Int

eval3 (Num i) = return i

eval3 (e1 :+: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 + v2)

eval3 (e1 :-: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 - v2)

eval3 (e1 :*: e2) = do
  v1 <- eval3 e1
  v2 <- eval3 e2
  count
  return (v1 * v2)
```

```
eval4 : Expr -> Ctr Int

eval4 (Num i) = return i

eval4 (e1 :+: e2) = do
  count
  return (!(eval4 e1) + !(eval4 e2))

eval4 (e1 :-: e2) = do
  count
  return (!(eval4 e1) - !(eval4 e2))

eval4 (e1 :*: e2) = do
  count
  return (!(eval4 e1) * !(eval4 e2))
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Idrise programmides esinevad **muutujad pole muudetavad ega ülekirjutatavad**

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Idrise programmides esinevad **muutujad pole muudetavad ega ülekirjutatavad**
- Seetõttu ei saa me Idrises järgmisel kujul olevaid programme esitada

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Idrise programmides esinevad **muutujad pole muudetavad ega ülekirjutatavad**
- Seetõttu ei saa me Idrises järgmisel kujul olevaid programme esitada

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {
  x = x + i;
  y = x + j;
  x = 42;
  return z
}
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Idrise programmides esinevad **muutujad pole muudetavad ega ülekirjutatavad**
- Seetõttu ei saa me Idrises järgmisel kujul olevaid programme esitada

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {
  x = x + i;
  y = x + j;
  x = 42;
  return z
}
```

- Probleemiks on Idrise funktsioonide puhtus ning mälu mittekasutamine

Näide 5: Muudetavad muutujad (nagu Javas, ...)

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Näitame, et **muudetavaid muutujaid saab monaadiga modelleerida**

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Näitame, et **muudetavaid muutujaid saab monaadiga modelleerida**
- Selle jaoks vaatleme funktsioonide `f : a -> b` asemel funktsioone `f : a -> St b`

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

- Näitame, et **muudetavaid muutujaid saab monaadiga modelleerida**
- Selle jaoks vaatleme funktsioonide `f : a -> b` asemel funktsioone `f : a -> St b`

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Näitame, et **muudetavaid muutujaid saab monaadiga modelleerida**
- Selle jaoks vaatleme funktsioonide `f : a -> b` asemel funktsioone `f : a -> St b`

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a
```

```
data Vars = X | Y | Z

State : Type
State = Vars -> Int

lookup : Vars -> State -> Int
lookup x s = s x

update : Vars -> Int -> State -> State
update x i s y = if x == y then i else s y
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Näitame, et **muudetavaid muutujaid saab monaadiga modelleerida**
- Selle jaoks vaatleme funktsioonide `f : a -> b` asemel funktsioone `f : a -> St b`

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a
```

```
data Vars = X | Y | Z

State : Type
State = Vars -> Int

lookup : Vars -> State -> Int
lookup x s = s x

update : Vars -> Int -> State -> State
update x i s y = if x == y then i else s y
```

- Funktsioonid `f : a -> St b` teisendavad **algoleku lõppväärtuseks ja lõppolekuks**

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')


Monaad St where
  return = sReturn
  bind   = sBind
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) ->
sBind f g = MkSt (\ s => case f
                           MkSt


Monaad St where
  return = sReturn
  bind   = sBind
```

State ei pea olema ainult `Vars -> Int`

```
ST' : Type -> Type -> Type
ST' s a = s -> (a,s)

data ST : Type -> Type -> Type where
  MkST : ST' s a -> ST s a

stReturn : a -> ST s a
stReturn x = ...

stBind : ST s a -> (a -> ST s b) -> ST s b
stBind f g = ...
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')


Monaad St where
  return = sReturn
  bind   = sBind
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')


prefix 8 ^!
(^!) : Vars -> St Int
(^!) x = MkSt (\ s => (lookup x s,s))
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL 1632

```
St' : Type -> Type
St' a = State -> (a,State)

data St : Type -> Type where
  MkSt : St' a -> St a


sReturn : a -> St a
sReturn x = MkSt (\ s => (x,s))

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')


prefix 8 ^!
(^!) : Vars -> St Int
(^!) x = MkSt (\ s => (lookup x s,s))

infix 7 ^=
(^=) : Vars -> Int -> St ()
(^=) x i = MkSt (\ s => ((),update x i s))
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

```
St' : Type -> Type
St' a = State -> (a

data St : Type ->
  MkSt : St' a ->

sReturn : a -> St
sReturn x = MkSt (\

sBind : St a -> (a -> St b) -> St b
sBind f g = MkSt (\ s => case f of
                           MkSt f' => case f' s of
                                        (x,s') => case g x of
                                                    MkSt g' => g' s')
```

Need **muutujate lugemise ja kirjutamise operatsioonid**

on spetsiifilised olekumonaadile olekutüübiga `Vars -> Int`

```
prefix 8 ^!
(^!) : Vars -> St Int
(^!) x = MkSt (\ s => (lookup x s,s))

infix 7 ^=
(^=) : Vars -> Int -> St ()
(^=) x i = MkSt (\ s => ((),update x i s))
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Java programm

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {

  x = x + i;

  y = x + j;

  x = 42;

  return z
}
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Java programm do-notatsiooniga

```
prog1 : Int -> Int -> St Int

prog1 i j = do

  x <- ^! X
  X ^= x + i

  x' <- ^! X
  Y ^= x' + j

  X ^= 42

  ^! Z
```

- Java programm

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {

  x = x + i;

  y = x + j;

  x = 42;

  return z
}
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Java programm

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {

  x = x + i;

  y = x + j;

  x = 42;

  return z
}
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Java programm do- ja !-notatsiooniga

```
prog2 : Int -> Int -> St Int

prog2 i j = do

  X ^= !(^! X) + i

  Y ^= !(^! X) + j

  X ^= 42

  ^! Z
```

- Java programm

```
int x = 1;
int y = 2;
int z = 3;

int prog(int i, int j) {

  x = x + i;

  y = x + j;

  x = 42;

  return z
}
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Muudetavate muutujatega programmide jooksutamiseks defineerime **run funktsiooni**

```
run : St a -> State -> (a,State)
run (MkSt f) s = f s

stateToList : State -> List Int
stateToList s = [s X , s Y , s Z]

runToList : St a -> State -> (a,List Int)
runToList f s = let (x,s') = run f s in (x, stateToList s')
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Muudetavate muutujatega programmide jooksutamiseks defineerime **run funktsiooni**

```
run : St a -> State -> (a,State)
run (MkSt f) s = f s

stateToList : State -> List Int
stateToList s = [s X , s Y , s Z]

runToList : St a -> State -> (a,List Int)
runToList f s = let (x,s') = run f s in (x, stateToList s')
```

- Näiteks valime oma programmi **muutujate algolekuks** järgmise oleku

```
initialState : State
initialState X = 1
initialState Y = 2
initialState Z = 3
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

- Java programm do-notatsiooniga

```
prog1 : Int -> Int -> St Int

prog1 i j = do

  x <- ^! X
  X ^= x + i

  x' <- ^! X
  Y ^= x' + j

  X ^= 42

  ^! Z
```

- Algolek

```
initialState : State
initialState X = 1
initialState Y = 2
initialState Z = 3
```

Näide 5: Muudetavad muutujad (nagu Javas, ...)

TARTU ÜLIKOOL

- Java programm do-notatsiooniga

```
prog1 : Int -> Int -> St Int

prog1 i j = do

  x <- ^! X
  X ^= x + i

  x' <- ^! X
  Y ^= x' + j

  X ^= 42

  ^! Z
```

- Algolek

```
initialState : State
initialState X = 1
initialState Y = 2
initialState Z = 3
```

```
Loeng14d> run (prog1 4 7) initialState
(3, update X 42 (update Y 12 (update X 5 initialState)))


Loeng14d> runToList (prog1 4 7) initialState
(3, [42, 12, 3])


Loeng14d> runToList (prog1 4 7) (\ _ => 0)
(0, [42, 11, 0])
```

Näide 6: Puhtad, efekti-vabad funktsioonid

Näide 6: Puhtad, efekti-vabad funktsioonid

- **Puhtad funktsioonid** on samuti näide monaadilistest arvutustest!

Näide 6: Puhtad, efekti-vabad funktsioonid

- **Puhtad funktsioonid** on samuti näide monaadilistest arvutustest!

```
data Id a = MkId a

iReturn : a -> Id a
iReturn x = MkId x

iBind : Id a -> (a -> Id b) -> Id b
iBind (MkId x') f = f x'

Monaad Id where
  return = iReturn
  bind   = iBind
```

Näide 6: Puhtad, efekti-vabad funktsioonid

- **Puhtad funktsioonid** on samuti näide monaadilistest arvutustest!

```
data Id a = MkId a

iReturn : a -> Id a
iReturn x = MkId x

iBind : Id a -> (a -> Id b) -> Id b
iBind (MkId x') f = f x'

Monaad Id where
  return = iReturn
  bind   = iBind
```

```
pureprog : Int -> Id Int -> Id Int

pureprog x y =
  return (x * !y + 7)
```

Näide 6: Puhtad, efekti-vabad funktsioonid

TARTU ÜLIKOOL

- **Puhtad funktsioonid** on samuti näide monaadilistest arvutustest!

```
data Id a = MkId a

iReturn : a -> Id a
iReturn x = MkId x

iBind : Id a -> (a -> Id b) -> Id b
iBind (MkId x') f = f x'

Monaad Id where
  return = iReturn
  bind   = iBind
```

```
pureprog : Int -> Id Int -> Id Int

pureprog x y =
  return (x * !y + 7)
```

- `Id a` on **isomorfne** tüübiga `a`

```
i : Id a -> a
i (MkId x) = x
```

```
j : a -> Id a
j x = MkId x
```

Näide 6: Puhtad, efekti-vabad funktsioonid

- **Puhtad funktsioonid** on samuti näide monaadilistest arvutustest!

```
data Id a = MkId a

iReturn : a -> Id a
iReturn x = MkId x

iBind : Id a -> (a -> Id b) -> Id b
iBind (MkId x') f = f x'

Monaad Id where
  return = iReturn
  bind   = iBind
```

```
pureprog : Int -> Id Int -> Id Int

pureprog x y =
  return (x * !y + 7)
```

- `Id a` on **isomorfne** tüübiga `a`

```
i : Id a -> a
i (MkId x) = x
```

```
j : a -> Id a
j x = MkId x
```

```
ij : (x : a) -> i (j x) = x
ij x = Refl

ji : (x : Id a) -> j (i x) = x
ji (MkId x) = Refl
```

Kokkuvõtteks

Kokkuvõtteks

TARTU ÜLIKOOL

- **Monaad = järjestikuste, kõrvalefekte** sisaldavate **programmide abstraktsioon**

Kokkuvõtteks

- **Monaad = järjestikuste, kõrvalefekte** sisaldavate **programmide abstraktsioon**
- Palju näiteid
  - puhtad funktsioonid (`f : Nat -> Id Bool`)
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (`f : Nat -> Option Bool`)
  - mälu kasutus (`f : Nat -> St Bool`)
  - mittedeterminism (`f : Nat -> List Bool`)
  - tõenäosused (`f : Nat -> Dist Bool`)
  - termipuud (`f : Nat -> Term Bool`)
  - ...

} **monaadid** (`f : Nat -> m Bool`)

Kokkuvõtteks

- **Monaad = järjestikuste, kõrvalefekte** sisaldavate **programmide abstraktsioon**
- Palju näiteid
  - puhtad funktsioonid (`f : Nat -> Id Bool`)
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid (`f : Nat -> Option Bool`)
  - mälu kasutus (`f : Nat -> St Bool`)
  - mittedeterminism (`f : Nat -> List Bool`)
  - tõenäosused (`f : Nat -> Dist Bool`)
  - termipuud (`f : Nat -> Term Bool`)
  - ...

} **monaadid** (`f : Nat -> m Bool`)

- Veel rohkem näiteid saab **olemasolevate monaadide komponeerimisest**

Kokkuvõtteks

- **Monaad = järjestikuste, kõrvalefekte** sisaldavate **programmide abstraktsioon**
- Palju näiteid
  - puhtad funktsioonid (`f : Nat -> Id Bool`)
  - sisend-väljund (`f : Nat -> IO Bool`)
  - erandid
  - mälu ka
  - mittede
  - tõenäos
  - termipuud (`f : Nat -> Term Bool`)
  - ...

`-> m Bool)`

Küsimus: Mis on Curry-Howardi vastavus monaadide jaoks?

Vastus: Loogika arvutiteaduses (LTAT.03.021) kevadsemestril

- Veel rohkem näiteid saab **olemasolevate monaadide komponeerimisest**

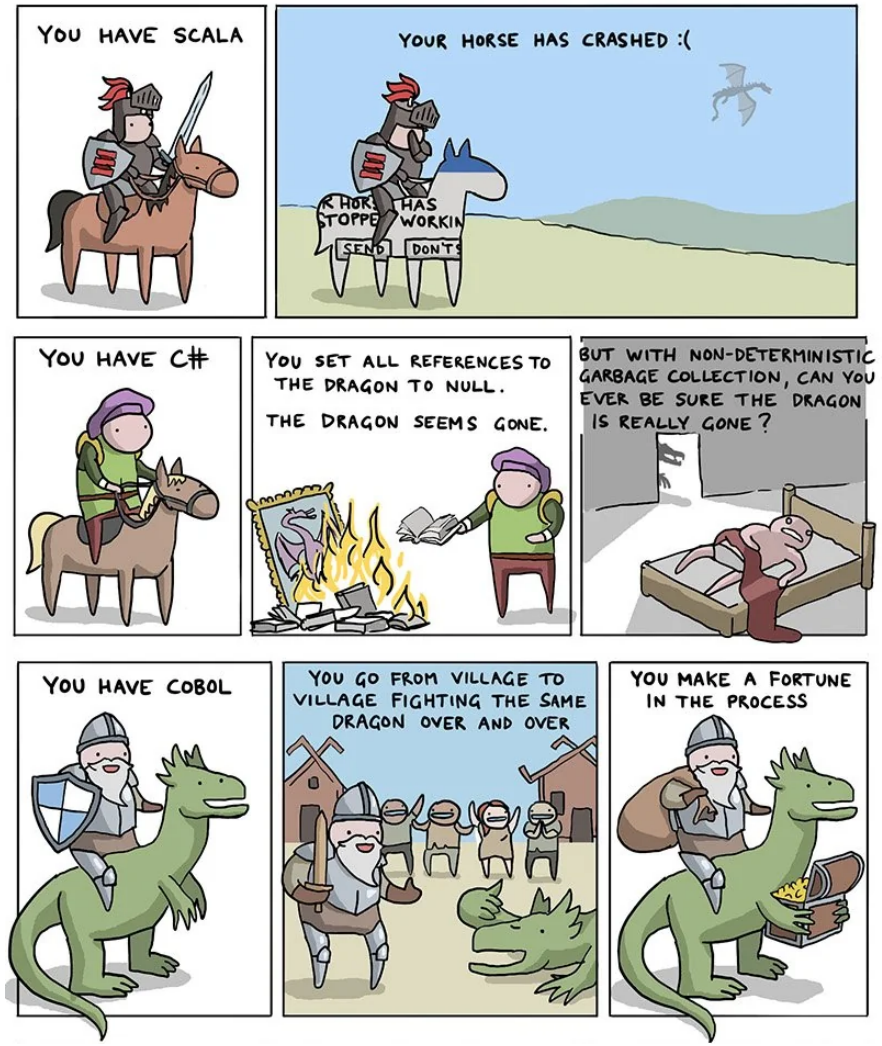
Kokkuvõtteks

- M… …oon
- Pa…
  - …
  - …
  - …
  - …
  - …ool)
  - …
  - …
  - …
  - …
  - …
- Vee…

GIT THE DRAGON!

HOW TO KILL THE DRAGON USING 9 PROGRAMMING LANGUAGES

BY toggl Goon Squad



MART VIRKUS '17 TOGGL.COM