Kava

* 10. loeng

K: laisk väärtustamine
V: Lihtsalt tüübitud λ-arvutus

Laiskus

Meil on kaks standardset reduktsioonijärjekorda:

- normaaljärjekord ja
- aplikatiivjärjekord.

Mis on eelised ja puudused? Kumba eelistada?

Normaaljärjekord on parem!

- Normaaljärjekord leiab alati normaalkuju, kui see eksisteerib! Normaalkuju ei ole näiteks: $(\lambda\, x.\ x\ x)\ (\lambda\, x.\ x\ x)$
- Ei väärtusta argumente mida ei kasutata. Näiteks $(\lambda\, x.\ 0)$ (fact 100)
- Funktsiooni argumendiks võib anda avaldise, millel pole normaalkuju.
  - Praktikas näiteks lõpmatu list. Näiteks Haskellis:

```
sieve (p:xs) = [x | x←xs, x `mod` p ≠ 0]
primes = map head (iterate sieve [2..])

Main> take 3 primes
[2,3,5]
```

- Kui argumenti kasutatakse mitu korda, väärtustatakse seda mitu korda :(

Laisk väärtustamine on veel parem!

- Nagu normaaljärjekord aga argumenti väärtustatakse maksimaalselt üks kord.
- Graafireduktsioon, mitte puu-redutkstioon.
- (Laisal) väärtustamisel on ka kitsam tähendus, mis ei kasuta reduktsioonisammu vaid on lähedasem kompileerimisele. Sellest räägime mõnes teises loengus/kursusel.

Näide (Haskell)

```
type Queen = (Int, Int)

attacking :: Queen → Queen → Bool
attacking (x1, y1) (x2, y2) =
    (x1 == x2) || (y1 == y2) || abs (x1 - x2) == abs (y1 - y2)

placeable :: Queen → [Queen] → Bool
placeable p qs = and [ (not (attacking p q)) | q ← qs ]

nQueens :: Int → [[Queen]]
nQueens n = nQueens' n where
    nQueens' 0 = [[]]
    nQueens' k =
        [ q:qs   | qs ← nQueens' (k - 1)
                 , q  ← [(k, y) | y ← [1..n]]
                 , placeable q qs]
```

Näide (Haskell)

```
*Main> time $ queens 10
```

```
♛ . . . . . . . . .
. . ♛ . . . . . . .
. . . . . ♛ . . . .
. . . . . . . ♛ . .
. . . . . . . . . ♛
. . . . ♛ . . . . .
. . . . . . . . ♛ .
. ♛ . . . . . . . .
. . . ♛ . . . . . .
. . . . . . ♛ . . .
```

```
0.007199s
```

Näide (Haskell)

```
fib :: Integer → Integer
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

*Main> time $ print $ fib 30
1346269
1.183209s
```

```
fibs :: [Integer]
fibs = 1 : 1 : zipWith (+) fibs (tail fibs)

*Main> time $ print $ fibs !! 30
1346269
0.000059s
```

Näide (Haskell)

```
foldl (+) 0 [10,20,30]
⟶ foldl (+) (0+10) [20,30]
⟶ foldl (+) (0+10+20) [30]
⟶ foldl (+) (0+10+20+30) []
⟶ 0+10+20+30
⟶ 10+20+30
⟶ 30+30
⟶ 60
```

vs.

```
foldl' (+) 0 [10,20,30]
⟶ foldl' (+) (0+10) [20,30]
⟶ foldl' (+) 10 [20,30]
⟶ foldl' (+) (10+20) [30]
⟶ foldl' (+) 30 [30]
⟶ foldl' (+) (30+30) []
⟶ foldl' (+) 60 []
⟶ 60
```

Sama arv samme.

Näide (Haskell)

```
*Main> time $ print $ foldl  (+) 0 [0..10000000]
50000005000000
2.895953s

*Main> time $ print $ foldl' (+) 0 [0..10000000]
50000005000000
0.268244s
```

- Laiskust mittekasutava koodi jaoks aeglasem ja raskemini optimeeritav kui agar väärtustamine. :(

Aplikatiivjärjekord on parem!

- Parem jõudlus lihtsamate optimisatsioonide abil.
  - `Int` parameeter protsessori registris.
- Jõudluse analüüs lihtsam. Vähem üllatusi.
- Aga lõpmatud listid???

Lazy tüübid

Idrises on (lihtsustatult) selline andmestruktuur:

```
data Lazy : Type → Type where
     Delay : (val : a) → Lazy a
```

... ja funktsioon

```
Force : Lazy a → a
```

- `Delay` argumenti ei väärtustata.
- `Force (Delay x) = x`
- Näiteks `and : List (Lazy Bool) → Bool`

Laisad listid

Lõpmatud laisad listid ehk striimid:

```
data Stream : Type → Type where
     (::) : a → Lazy (Stream a) → Stream a
```

ja laisad listid

```
mutual
    data LList : Type → Type where
        Nil : LList a
        (::) : a → LazyList a → LList a

    LazyList : Type → Type
    LazyList a = Lazy (LList a)
```

Näide (Idris)

```
Queen : Type
Queen = (Int, Int)

attacking : Queen → Queen → Bool
attacking (x1, y1) (x2, y2) =
    (x1 == x2) || (y1 == y2) || abs (x1 - x2) == abs (y1 - y2)

placeable : Queen → List Queen → Bool
placeable p qs = and [ delay (not (attacking p q)) | q ← qs ]
-- erinevus Haskellist:
--    and töötab List (Lazy a) peal, peame lisama delay

nQueens : Int → LazyList (List Queen)
nQueens n = nQueens' n where
    nQueens' : Int → LazyList (List Queen)
    nQueens' 0 = [[]]
    nQueens' k =
        [ q::qs | qs ← nQueens' (k - 1)
                , q  ← cast [(k, y) | y ← [1..n]]
                , placeable q qs]
-- erinevus Haskellist:
--    listikomprehensioon töötab List-i peal,
--    peame konverteerima LazyList-iks (cast).
```

- Esimene element: 10 korda kiirem kui Haskellis!

Tüübisüsteemid

- Tüübid spetsifitseerivad programmide omadusi.
  - λ-arvutuses klassifitseerivad normaalkujusid.
  - Kui term on tüüpi Nat, ja ta on normaalkujul, siis see term esitab naturaalarvu.
- Tüübisüsteem koosneb kolmest komponendist:
  - tüüpide hulk;
  - programmide hulk;
  - tüüpimisreeglid.

Lihtsad tüübid

- Olgu antud loenduv hulk tüübimuutujaid (baastüübid).
- Lihtsad tüübid = ei ole polümorfsed.
- Lihtsate tüüpide süntaks:

$$\begin{array}{rcll} \tau & ::= & \alpha & \text{tüübimuutuja} \\ & | & \tau_1 \to \tau_2 & \text{funktsioonitüüp} \end{array}$$

- Tüübikonstruktor $\to$ on paremassotsiatiivne.

$$\tau_1 \to \tau_2 \to \tau_3 \equiv \tau_1 \to (\tau_2 \to \tau_3)$$

Church vs. Curry

- Tüüpidega λ-avaldiste esitamiseks on kaks peamist stiili.
- Church'i stiil: funktsiooni parameetrite tüübid esitatakse ilmutatult λ-termis $\lambda x : \tau.\ e$.
  - Näiteks: $\lambda x : \mathsf{Nat}.\ x + x : \mathsf{Nat} \to \mathsf{Nat}$.
  - Igal termil on unikaalne tüüp.
- Curry stiil: termid on tavalised (s.o. ilma tüüpideta) ja kasutatakse tüübikorrektsuse predikaati (tüübituletus).
  - Näide: $\lambda x.\ x + x : \mathsf{Nat} \to \mathsf{Nat}$.
  - Termidel on palju tüüpe (näit. $\lambda x.\ x : \mathsf{Nat} \to \mathsf{Nat}$, kuid ka $\lambda x.\ x : \mathsf{Bool} \to \mathsf{Bool}$).
- Meie kasutame põhiliselt esimest stiili.

Termid ja redutseerimine

- Termide süntaks:

$$
\begin{array}{rcll}
e & ::= & x & \text{muutuja} \\
 & | & e_1\ e_2 & \text{aplikatsioon} \\
 & | & \lambda x : \tau.\ e & \text{abstraktsioon}
\end{array}
$$

- Samad süntaktilised konventsioonid kui puhtas λ-arvutuses.
- Substitutsioon, reduktsioon jmt on defineertud analoogselt puhta λ-avutusega ignoreerides tüübiannotatsioone.

Tüüpimisrelatsioon

- Et teha kindlaks mis termid on millist tüüpi, defineerime tüüpimisrelatsiooni.
- Baasrelatsioon on kujul $\Gamma \vdash e : \tau$
  - *"term e on tüüpi τ kontekstis Γ"*.
- Kontekst Γ on muutujast ja tüübist koosnevate paaride jada $\Gamma = \{x_1 : \tau_1, \ \ldots, \ x_n : \tau_n\}$.
  - Konteksti Γ doomenit tähistame $\mathrm{dom}(\Gamma) = \{x_1, \ \ldots, \ x_n\}$.
  - Korrektses tüüpimisrelatsioonis peab olema $\mathrm{FV}(e) \subseteq \mathrm{dom}(\Gamma)$.
- $\Gamma \leq \Delta$ tähendab, et kui $x \in \mathrm{dom}(\Gamma)$, siis $x \in \mathrm{dom}(\Delta)$ ja $\Gamma(x) = \Delta(x)$.
- Programm (s.o. kinnine term) e on tüüpi τ, kui tal on see tüüp tühjas kontekstis $\vdash e : \tau$.

Tüüpimisrelatsioon

- Tüüpimisrelatsioon on defineeritud tüüpimisreeglite abil.
- Lihtsalt tüübitud λ-arvutuse tüüpimisreeglid:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ var \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma.\, e : \sigma \to \tau}\ abs$$

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}\ app$$

  kus $x \notin \mathrm{dom}(\Gamma)$.
- Edaspidi tähistame lihtsalt tüübitud λ-arvutust $\lambda\!\to$.

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (1):

$$\frac{}{\vdash \lambda x{:}\tau,\ y{:}\sigma.\ x\ :}$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (1):

$$\dfrac{\dfrac{}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x :}}{\vdash \lambda x{:}\tau,\ y{:}\sigma.\, x : \tau \to}\ abs$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x :}}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to}\ abs}{\vdash \lambda x{:}\tau,\ y{:}\sigma.\, x : \tau \to \sigma \to}\ abs$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x : \tau}\ \mathit{var}}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to \tau}\ \mathit{abs}}{\vdash \lambda x{:}\tau,\ y{:}\sigma.\, x : \tau \to \sigma \to \tau}\ \mathit{abs}$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (2):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash x : \sigma \to \tau \qquad \dfrac{\Gamma \vdash y : \rho \to \sigma \quad \Gamma \vdash z : \rho}{\Gamma \vdash yz : \sigma}}{\Gamma \vdash x(yz) : \tau}}{\Gamma_2 \vdash \lambda z{:}\rho.\, x(yz) : \rho \to \tau}}{\Gamma_1 \vdash \lambda y{:}\rho\to\sigma,\ z{:}\rho.\, x(yz) : (\rho \to \sigma) \to \rho \to \tau}}{\vdash \lambda x{:}\sigma\to\tau,\ y{:}\rho\to\sigma,\ z{:}\rho.\, x(yz) : (\sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau}
$$

kus

$$
\begin{array}{lcl}
\Gamma_1 & = & \{x{:}\sigma\to\tau\} \\
\Gamma_2 & = & \{x{:}\sigma\to\tau,\ y{:}\rho\to\sigma\} \\
\Gamma & = & \{x{:}\sigma\to\tau,\ y{:}\rho\to\sigma,\ z{:}\rho\}
\end{array}
$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (3):

$$\frac{}{\vdash \lambda x{:}\tau.\, x\ x\ :}$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (3):

$$\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x\ x\ :}}{\vdash \lambda x{:}\tau.\ x\ x : \tau \to}\ abs$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \sigma \to} \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \sigma}}{\{x{:}\tau\} \vdash x\ x :}\ app}{\vdash \lambda x{:}\tau.\, x\ x : \tau \to}\ abs$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to} \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\,var}{\{x{:}\tau\} \vdash x\,x :}\,app}{\vdash \lambda x{:}\tau.\, x\,x : \tau \to}\,abs$$

Tüüpimispuu konstrueerimine

Tüüpimispuu konstrueerimise näide (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to \rho}\,var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\,var}{\{x{:}\tau\} \vdash x\,x : \rho}\,app}{\vdash \lambda x{:}\tau.\, x\,x : \tau \to \rho}\,abs$$

Tüüpimispuu konstrueerimine ebaõnnestub!

Tüübiohutus

- Tüübiohutus:
  Korrektselt tüübitud programmid "ei lähe valesti".
- Hind:
  - Mõned mõistlikud programmid hüljatakse.
- Koosneb kahest osast:
  - Progress: Korrektselt tüübitud ei ole tupikus (ta on kas väärtus või saab teha ühe väärtustusreeglitele vastava sammu).
  - Säilitamine: Kui korrektselt tüübitud term teeb ühe sammu, siis ka resultaat on korrektselt tüübitud.

Tüüpide unikaalsus, säilimine ja progress

- Tüüpide unikaalsus:
  Kui $\Gamma \vdash e : \tau_1$ ja $\Gamma \vdash e : \tau_2$, siis $\tau_1 = \tau_2$.
- NB! Tüüpide unikaalsus ei kehti mitmete rikkamate keelte korral.

- Tüüpide säilimine (subjekt-reduktsioon):
  Kui $\Gamma \vdash e : \tau$ ja $e \twoheadrightarrow_\beta e'$, siis $\Gamma \vdash e' : \tau$.

- Progress:
  Kui $\Gamma \vdash e : \sigma$, siis $\exists e' : \sigma . e \rightarrow_\beta e'$ või $e \in \mathrm{Val}$, kus

$$v \in \mathrm{Val} \ ::= \ x\, v \ldots v \mid \lambda x{:}\tau . v$$

Tugev normaliseerimine

- Tugev normaliseerimine:
  Lihtsalt tüübitud λ-arvutuses $\lambda\!\rightarrow$ on iga β-reduktsiooni jada lõplik.
- Mõned lihtsad järeldused:
  - Termide võrdsuse küsimus on lahenduv.
  - Püsipunktikombinaatorid ei ole defineeritavad.

Laiendus: tõeväärtused

- Tüübid: $\tau ::= \ldots \mid \mathsf{Bool}$
- Termid: $e ::= \ldots \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$
- Väärtused: $v ::= \ldots \mid \mathsf{true} \mid \mathsf{false}$
- Tüüpimisreeglid:

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}}$$

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}$$

- Väärtustamisreeglid:

$$\begin{array}{lcl} \mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2 & \rightarrow & e_1 \\ \mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2 & \rightarrow & e_2 \end{array}$$

Laiendus: ühiktüüp

- Tüübid: $\tau ::= \ldots \mid \mathrm{Unit}$
- Termid: $e ::= \ldots \mid ()$
- Väärtused: $v ::= \ldots \mid ()$
- Tüüpimisreeglid:

$$\frac{}{\Gamma \vdash () : \mathrm{Unit}}$$

- Väärtustamisreegleid ei ole!

Laiendus: paarid

- Tüübid: $\tau ::= \ldots \mid \tau_1 \times \tau_2$
- Termid: $e ::= \ldots \mid (e_1, e_2) \mid \mathtt{fst} \mid \mathtt{snd}$
- Väärtused: $v ::= \ldots \mid (v_1, v_2)$
- Tüüpimisreeglid:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{fst}\, e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{snd}\, e : \tau_2}$$

- Väärtustamisreeglid:

$$\begin{array}{lcl} \mathtt{fst}\,(e_1, e_2) & \to & e_1 \\ \mathtt{snd}\,(e_1, e_2) & \to & e_2 \end{array}$$

Laiendus: summatüüp

- Tüübid: $\tau ::= \ldots \mid \tau_1 + \tau_2$
- Termid: $e ::= \ldots \mid \mathtt{inl} \mid \mathtt{inr} \mid \mathsf{case}\,(e_0;\ x_1.e_1;\ x_2.e_2)$
- Väärtused: $v ::= \ldots \mid \mathtt{inl}\ v_1 \mid \mathtt{inl}\ v_2$
- Tüüpimisreeglid:

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathtt{inl}\ e : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathtt{inr}\ e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e_0 : \tau_1 + \tau_2 \quad \Gamma,\ x_1 : \tau_1 \vdash e_1 : \sigma \quad \Gamma,\ x_2 : \tau_2 \vdash e_2 : \sigma}{\Gamma \vdash \mathsf{case}\,(e_0;\ x_1.e_1;\ x_2.e_2) : \sigma}$$

- Evaluation rules:

$$\begin{array}{lcl} \mathsf{case}\,(\mathtt{inl}\ e_0;\ x_1.e_1;\ x_2.e_2) & \to & e_1[x_1 \mapsto e_0] \\ \mathsf{case}\,(\mathtt{inr}\ e_0;\ x_1.e_1;\ x_2.e_2) & \to & e_2[x_2 \mapsto e_0] \end{array}$$