Programmeerimise paradigmad

Saab jagada kaheks:

- imperatiivsed e. mis operatsioone teha
  - Lisaks jaotatakse: protseduuraalsed ja objektorienteeritud
- deklaratiivsed e. lahenduse (tõe) kirjeldamine
  - Lisaks jaotatakse: funktsionaalne ja loogiline

Erinevused:

- Imperatiivne on lähedane protsessori käsustikule.
- Deklaratiivne on lähedane matemaatikale.

Matemaatik/loogik tahab mõelda kõrgemal tasemel:

- algebralistest struktuuridest nagu rühmad, monoidid, ringid jne.
- funktsioonidest, hulkadest ja relatsioonidest

Paljud protsessori instruktsioonid pole (matemaatikule) intuitiivsed.
Näiteks Javas: `Math.abs(-2147483648) == -2147483648`

Funktsionaalne keel

- Matemaatiline funktsioon: tulemus sõltub ainult funktsiooni parameetritest.
- Ilma funktsioonideta (meetodite, protseduurideta) ei saa!

  Java: `Math.max(4, 7)`
  Python: `max(4,7)`
  Idris: `max 4 7`

- Sõna „funktsioon“ asemel kasutatakse ka *abstraktsioon*.
  (Kuna enamasti ei huvita, kuidas `max` on implementeeritud.)

Funktsionaalne keel

- Matemaatiline funktsioon: tulemus sõltub ainult funktsiooni parameetritest.
- Ilma funktsioonideta (meetodite, protseduurideta) ei saa!

  Java: `Math.max(4, 7)`
  Python: `max(4,7)`
  Idris: `max 4 7`

- Sõna „funktsioon“ asemel kasutatakse ka *abstraktsioon*.
  (Kuna enamasti ei huvita, kuidas `max` on implementeeritud.)
- FP eelistab olemasolevate vahendite üldistamist.

  Näiteks: Samamoodi, kuidas programmeerimiskeeles saab käsitleda andmeid, peab saama käsitleda ka alamprogramme.

  $\Longrightarrow$ `max 3` saab võtta kõigist listi `[2,3,4]` elementidest (3. loeng)

  `map (max 3) [2,3,4] == [max 3 2, max 3 3, max 3 4]`

Funktsionaalne keel

- Matemaatiline funktsioon: tulemus sõltub ainult funktsiooni parameetritest.
- Ilma funktsioonideta (meetodite, protseduurideta) ei saa!

  Java: `Math.max(4, 7)`
  Python: `max(4,7)`
  Idris: `max 4 7`

- Sõna „funktsioon“ asemel kasutatakse ka *abstraktsioon*.
  (Kuna enamasti ei huvita, kuidas `max` on implementeeritud.)
- FP eelistab olemasolevate vahendite üldistamist.

  Näiteks: Samamoodi, kuidas programmeerimiskeeles saab käsitleda andmeid, peab saama käsitleda ka alamprogramme.

  $\Longrightarrow$ `max 3` saab võtta kõigist listi `[2,3,4]` elementidest (3. loeng)

  `map (max 3) [2,3,4] == [max 3 2, max 3 3, max 3 4] == [3,3,4]`

Puhas keel

- Puhas ehk kõrvaltoimevaba — funktsiooni kutse tulemus sõltub ainult parameetrite väärtustest.

  $\Longrightarrow$ igat funktsiooni saab testida teistest eraldi

Puhas keel

- Puhas ehk kõrvaltoimevaba — funktsiooni kutse tulemus sõltub ainult parameetrite väärtustest.

  $\Longrightarrow$ igat funktsiooni saab testida teistest eraldi

- Mittepuhtaid funktsioone (nagu juhuarvude genereerimine) pole võimalik funktsioonidena defineerida

  aga saab *modelleerida* kasutades puhtaid funktsioone.

Tugevalt ja staatiliselt tüübitud

- Mida rohkem programmeerimiskeel lubab seda parem?
  - Ei! Näiteks, Portable Document Format (1993) on suuresti vähendatud võimalustega PostScript (1982).
- Piiramise üks võimalus on *tüübisüsteemiga*.
  - Tüüp kirjeldab (ja kitsendab) programmi alamosa ja tüübikontroll otsustab, kas neid osi tohib niimoodi kombineerida.
  - Näiteks: Kui `a` on tüüpi `Int` siis seda saab anda `max` argumendiks.
  - Testimise vajadus mingil määral väiksem?

Tugevalt ja staatiliselt tüübitud

- Mida rohkem programmeerimiskeel lubab seda parem?
  - Ei! Näiteks, Portable Document Format (1993) on suuresti vähendatud võimalustega PostScript (1982).
- Piiramise üks võimalus on *tüübisüsteemiga*.
  - Tüüp kirjeldab (ja kitsendab) programmi alamosa ja tüübikontroll otsustab, kas neid osi tohib niimoodi kombineerida.
  - Näiteks: Kui `a` on tüüpi `Int` siis seda saab anda `max` argumendiks.
  - Testimise vajadus mingil määral väiksem?
- Staatiline kontroll — kompileerimise käigus
- Tugevalt tüübitud — väljastatakse kohe veateade.

Listifunktsioonid: prod

Implementeerime funktsiooni, mis korrutab kokku kõik parameetrina antud listi elemendid. Näiteks: `prod [2,3,4] = 2*3*4 = 24`

```
prod : List Int → Int
prod = ?rhs_prod
```

Listifunktsioonid: concat

Implementeerime funktsiooni, mis konkateneerib (++) kokku kõik parameetrina antud listi elemendid. Näiteks:

```
concat [[2,3], [4], [], [5]] = [2, 3, 4, 5]
```

```
concat : List (List a) → List a
concat = ?rhs_concat
```

Listifunktsioonid: head

Implementeerime funktsiooni, mis tagastab listi pea. Näiteks:

```
head [2,3,4,5] = 2
```

```
head : List a → a
head = ?rhs_head
```

Listifunktsioonid: tail

Implementeerime funktsiooni, mis tagastab listi saba. Näiteks:

```
tail [True, False, True] = [False, True]
```

```
tail : List a → List a
tail = ?rhs_head
```

Listifunktsioonid: kolmas

Implementeerime funktsiooni, mis tagastab listi kolmanda elemendi.
Näiteks: `kolmas [True, False, True] = True`

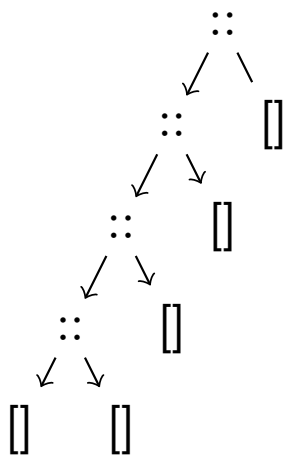
```
kolmas : List a → a
kolmas = ?rhs_head
```

Listi esitus puuna

Mis on järgneva listi esitus puuna: `[4,5,6]`

Küsimus

Kas järgnev on valiidne struktuur? Kui jah, siis mis on selle tüüp.



Konstantidega λ-arvutus

- Termid:

e	$::=$	x	muutuja
	$\mid$	c	konstant
	$\mid$	$(e_1\ e_2)$	aplikatsioon
	$\mid$	$(\lambda x.\ e)$	abstraktsioon

- Puhas λ-arvutus on λ-arvutus ilma konstantide ja muude lisadeta
- Sulgudest hoidumine:

$$
\begin{array}{lcl}
e_1\ e_2\ \ldots\ e_n & \equiv & ((\ldots(e_1\ e_2)\ldots)e_n) \\
\lambda x.\ e_1\ e_2\ \ldots\ e_n & \equiv & (\lambda x.\ (e_1\ e_2\ \ldots\ e_n)) \\
\lambda x_1\ x_2\ \ldots\ x_n.\ e & \equiv & (\lambda x_1.\ (\lambda x_2.\ (\ldots(\lambda x_n.\ e)\ldots)))
\end{array}
$$

- Näited:

$\lambda x.\ x$ $\qquad$ $((\lambda x.\ (\lambda f.\ f\ x))\ y)\,(\lambda z.\ z)$

add 2 3 $\qquad$ $(\lambda x.\ \mathsf{mul}\ x\ 2)\ 5$

Lambda-arvutuse intuitsioon

- Term kodeerib mingit arvutusprobleemi.
- Termi teisendatakse vastavalt lihtsustusreeglitele.
- Kui rohkem lihtsustada ei saa, oleme leidnud tulemuse.

Näited

- add (mul 2 3) 1 $\rightarrow$ add 6 1 $\rightarrow$ 7
- $(\lambda x.\ x\ 3\ 2)$ mul $\rightarrow$ mul 3 2 $\rightarrow$ 6

Lambda-arvutus

- Mõne aja pärast näeme, et konstante ei ole tegelikult "vaja". Kõik on võimalik kodeerida puhaste funktsioonide abil.
- Vähegi suuremate termide puhul kaob ülevaatlikus. Rakendame masinlikult reegleid.
- Selguse parandamiseks kasutame makro-definitsioone (peavad olema mitterekursiivsed).
- Näited:

$$
\begin{array}{lcl}
\mathsf{liida} & \equiv & \lambda x\, y.\ \mathsf{add}\ x\ y \\
\mathsf{dbl} & \equiv & \lambda x.\ \mathsf{add}\ x\ x \\
\mathsf{I} & \equiv & \lambda x.\ x \\
\mathsf{K} & \equiv & \lambda x\, y.\ x \\
\mathsf{S} & \equiv & \lambda f\, g\, x.\ f\, x\, (g\, x)
\end{array}
$$

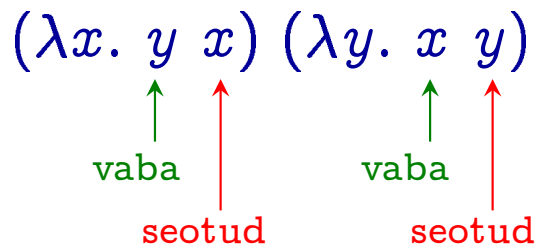
Termi esitus süntaksipuuna

$$\lambda x\ y.\ g\ \lambda z.\ (\lambda q.\ x)\ z$$

- Joonista term süntaksipuuna ehk pane sulud.
- Mis on selle termi vabad muutujad?

Vabad ja seotud muutujad

- Muutuja x on *vaba* λ-termis e (so. $x \in \mathrm{FV}(e)$), kui ta ei asu sümboli λx mõjupiirkonnas.
- Vastasel korral on x *seotud*.



Vabad ja seotud muutujad

- Vabade muutujate induktiivne definitsioon:

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\} \\ \mathrm{FV}(c) & = & \emptyset \\ \mathrm{FV}(e_1\ e_2) & = & \mathrm{FV}(e_1) \bigcup \mathrm{FV}(e_2) \\ \mathrm{FV}(\lambda x.\ e) & = & \mathrm{FV}(e) - \{x\} \end{array}$$

- Ilma vabade muutujateta λ-termid on kinnised.
- Näited:

$$\begin{array}{lcl} \mathrm{FV}(\lambda x\, y.\ x\ y) & = & \emptyset \\ \mathrm{FV}(\lambda x.\ (\lambda y.x)\,(\lambda z.y)) & = & \{y\} \end{array}$$

Leia termi vabad ja seotud muutujad!

$$(\lambda x\ x.\, x)\ x\ (\lambda x.\ x\ x)\ x$$

α-konversioon

- Seotud muutujate nimed ei oma tähtsust!
- λ-termid e_1 ja e_2 on α-kongruentsed (tähistus $e_1 =_\alpha e_2$) kui nad on identsed seotud muutujate ümbernimetamise täpsuseni.
- Näited:

$$\begin{array}{lcl} \lambda x.\ x & =_\alpha & \lambda y.\ y \\ \lambda x.\ f\ x & =_\alpha & \lambda z.\ f\ z \\ \lambda x.\ (\lambda y.\ y)\ x & =_\alpha & \lambda y.\ (\lambda y.\ y)\ y \\ \lambda x\ y.\ x + y & \neq_\alpha & \lambda y\ y.\ y + y \end{array}$$