Funktsionaalprogrammeerimise meetod

Varmo Vene

Arvutiteaduse Instituut
Tartu Ülikool

EMAIL: varmo@cs.ut.ee
WWW: http://www.cs.ut.ee/~varmo/MFP2008/
LIST: ati.funprog@lists.ut.ee

Üldinfo

Hinde kujunemine

- Praktikumid (50 punkti)
  - punkte annab praktikumi juhendaja (H.Nestra või K.Apinis) koduülesannete lahenduste ning aktiivse osavõtu eest
- Eksam (50 punkti)
  - eksam on kirjalik, aega 2 tundi, täpsemad tingimused viimases loengus
- Boonusülesanded (50 punkti)
  - ülesanded koostab ning punkte annab H.Nestra
- Individuaalprojekt (50 punkti)
  - projektikirjeldused koostab ning punkte annab K.Apinis
- Hindeskaala traditsiooniline (so, 91–...=A, 81–90=B, jne)

Üldinfo

Kirjandus

1. B. O'Sullivan, D. Stewart, J. Goerzen. *Real World Haskell*. O'Reilly Media, Inc., 2008.
2. G. Hutton. *Programming in Haskell*. Cambridge University Press, 2007.
3. S. Thompson. *Haskell: The Craft of Functional Programming*. 2nd ed. Addison-Wesley. 1999.
4. P. Hudak. *The Haskell School of Expression: Learning Functional Programming through Multimedia*. Cambridge University Press, 2000.
5. R. Bird. *Introduction to Functional Programming using Haskell*. 2nd ed. Prentice Hall, 1998.
6. R. L. Page. *Two Dozen Short Lessons in Haskell*. Uni. of Oklahoma, 1997.

Programmeerimiskeelte paradigmad

Imperatiivsed keeled

Protseduursed

Fortran, Pascal, C, Ada

Objektorienteeritud

Smalltalk, C++, Java

Deklaratiivsed keeled

Funktsionaalsed

Scheme, ML, Haskell

Loogilised

Prolog, Gödel, Mercury

Funktsionaalsed keeled

Funktsionaalsete keelte põhiomadused

- Puudub programmi oleku mõiste
  - Puuduvad kõrvalefektid, muutujad
  - *Ilmutatud viidatavus* (referential transparency)
- Rikkad abstraktsioonivahendid
  - Andmeabstraktsioon
  - Kõrgemat-järku funktsioonid
  - Laisk väärtustamine
- Võimas tüübisüsteem
  - Staatiline tüübikontroll, tüüpide tuletamine
  - Polümorfism (parameetriline, 'ad-hoc')

Funktsionaalsed keeled

Põhjusi FP õppimiseks

- Erinev lähenemine programmeerimisele
  - annab uue vaatenurga tuntud probleemide lahendamiseks
- FP on "kõrgtaseme" paradigma
  - lubab probleeme lahendada väga abstarktsel tasemel
  - kergendab vigade leidmist/vältimist
  - võit produktiivsuses tihti mitmekordne
- Mõju teiste keelte arengule
  - automaatne mäluhaldus (Java)
  - polümorfism (C++, Ada)
- Lihtne semantika
  - formaalne keele kirjeldamine
  - programmide analüüs ja teisendamine

Funktsionaalsed keeled

Funktsionaalsete keelte puudusi

- Mõne ülesande lahendamisel võib omistamine olla vajalik
- Mõnikord me tahame modellerida väliskeskonna olekut
- Kasutajaliidese programmeerimine võib olla keerulisem
- Programmide efektiivsus ei ole mitte alati piisav
- Aja- ja mälukeerukus võib olla raskesti ennustatav

Funktsionaalsed keeled

FP lühiajalugu

- Kombinaatorloogika (M. Schönfinkel 1924, H. Curry 1927)
- Lambda-arvutus (A. Church 1936)
- Lisp (J. McCarthy 1958)
- ISWIM (P. Landin 1965)
- FP (J. Backus 1977)
- ML ja polümorfne tüübisüsteem (R. Milner 1978)
- Hope, Sasl, Miranda, . . . (1980 – 85)
- Haskell (1988)

Funktsionaalsed keeled

Funktsionaalne keel Haskell

- Lühiajalugu

1987	loodi Haskell'i komitee
1988	esimene keelekirjeldus (v. 1.0)
1999	Haskell98 (Standard Haskell)
(2008)	Haskell' (Haskell-prime)

- Koduleheküg: `http://www.haskell.org/`
- Realisatsioonid

hugs	interpretaator; Portland, Yale
ghc	kompilaator; Glasgow, Cambridge

Haskell

Hugs 98

- Hugs = Haskell Users Gofer System
- `http://www.haskell.org/hugs`
  - Win95/NT/XP, Linux, FreeBSD, MacOS X, ...
- Käivitamiseks käsurealt:

  `hugs [options] [file]`

GHC

- GHC = Glasgow Haskell Compiler
- `http://www.haskell.org/ghc`
  - Win95/NT/XP, Linux, Solaris, FreeBSD, MacOS X
- Käivitamiseks käsurealt:

  `ghci [options] [file]`

Haskell

Programmi struktuur

- Programm koosneb moodulitest
- Peamooduli nimi *Main*
- Vaikimis laaditakse sisse moodul *Prelude*

Mooduli struktuur

- Mooduli päis (mooduli nimi ja eksportlist)
- Deklaratsioonid (impordid, funktsioonide tüübid, . . .)
- Definitsioonid (andmetüüpide, funktsioonide)

Näide

```
module Hello (hello) where
hello :: String
hello = "Hello, World!"
```

Haskell

Tüüpide deklareerimine

$$fname_1, fname_2, ..., fname_n :: type$$

- Ei ole kohustuslik, kuna süsteem suudab tavaliselt ise tüübid tuletada
- On soovitatav, kuna aitab vastavaid funktsioone dokumenteerida
- Mõningatel juhtudel ei saa süsteem ise hakkama
- Tüübideklaratsioonid võivad esineda ka avaldistes

$$3 * (5 :: Int) + 4$$

NB!

Tüüpide nimed algavad suurte tähtedega, funktsioonide nimed väikestega

Haskell

Funktsioonide defineerimine

$$fname\ arg_1 \ldots arg_n = expr$$

- Võrduse vasakpool koosneb funktsiooni nimest ja argumentide näidistest
- Võrduse parempool koosneb defineerivast avaldisest

Näide — faktoriaal

$$\begin{aligned} &\text{-- } fact1 \text{ on defineeritud tingimusavaldise abil} \\ &fact1\ n = \mathbf{if}\ n == 0\ \mathbf{then}\ 1 \\ &\qquad\qquad\qquad\qquad \mathbf{else}\ \ n * fact1\ (n-1) \end{aligned}$$

Haskell

Avaldise väärtustamine (reduktsioon)

- Leitakse funktsiooni defineeriv võrdus
- Avaldis asendatakse võrduse parema poolega
- Formaalparameetrid asendatakse tegelike argumentidega
- Kogu protsessi korratakse kuni rohkem ei saa

Näide

$$
\begin{aligned}
\mathit{fact1}\ 2 &\Longrightarrow \mathbf{if}\ 2 == 0\ \mathbf{then}\ 1\ \mathbf{else}\ 2 * \mathit{fact1}\ (2-1) \\
&\Longrightarrow 2 * \mathit{fact1}\ 1 \\
&\Longrightarrow 2 * (\mathbf{if}\ 1 == 0\ \mathbf{then}\ 1\ \mathbf{else}\ 1 * \mathit{fact1}\ (1-1)) \\
&\Longrightarrow 2 * (1 * \mathit{fact1}\ 0) \\
&\Longrightarrow 2 * (1 * (\mathbf{if}\ 0 == 0\ \mathbf{then}\ 1\ \mathbf{else}\ 0 * \mathit{fact1}\ (0-1))) \\
&\Longrightarrow 2 * (1 * 1) \\
&\Longrightarrow 2
\end{aligned}
$$

Haskell

Mitme võrdusega definitsioonid

- Funktsiooni defineerimiseks võib kasutada mitut võrdust
- Väärtustamisel kasutatakse esimest "sobivat"

Näited

{- *fact2* on defineeritud näidistega sobitamise abil;
töötab ainult 32bitiste täisarvudega -}

$$
\begin{aligned}
&fact2 :: Int \to Int \\
&fact2\ 0 = 1 \\
&fact2\ n = n * fact2\ (n-1)
\end{aligned}
$$

-- *fact3* on defineeritud "valvurite" abil

$$
\begin{aligned}
&fact3 :: Integer \to Integer \\
&fact3\ n \mid n == 0 \quad = 1 \\
&\qquad\quad \mid otherwise = n * fact3\ (n-1)
\end{aligned}
$$

Haskell

Variatsioone faktoriaalile

```
-- see ei lähe lõpmatusse tsüklisse
fact4 :: Integer -> Integer
fact4 n | n == 0 = 1
        | n >= 1 = n * fact4 (n - 1)


-- samad sõnad, kuid saavutatud n + k näidiste abil
fact5 :: Integer -> Integer
fact5 0       = 1
fact5 (n + 1) = (n + 1) * fact5 n
```

Haskell

Variatsioone faktoriaalile

```
-- akumulaatorit kasutav faktoriaal
-- defineeritud where konstruktsiooni abil
fact6 :: Integer → Integer
fact6 n = fact6' 1 n
  where fact6' a 0 = a
        fact6' a n = fact6' (a * n) (n - 1)


-- standartne iteratiivne definitsioon kasutades
-- eeldefineeritud funktsiooni product ja
-- aritmeetilise jada konstruktsiooni
fact8 :: Integer → Integer
fact8 n = product [1..n]
```

Haskell

Paigutusreeglid

- Paigutus määrab ära definitsioonide kuuluvuse
- Kõik sama taseme definitsioonid peavad algama täpselt samast veerust
- Võimalik on ka ”traditsiooniline” (ilmutatud) grupeerimine kasutades loogilisi sulge ja semikooloneid

Paigutusega grupeerimine

```
a = b + c
  where b = 1
        c = 2
d = a * 2
```

Ilmutatud grupeerimine

```
{ a = b + c
    where { b = 1;
            c = 2};
  d = a * 2}
```