

Funktsionaalprogrammeerimise meetod

Varmo VENE

Arvutiteaduse Instituut
Tartu Ülikool

EMAIL: varmo@cs.ut.ee
WWW: <http://www.cs.ut.ee/~varmo/MFP2007/>
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)

Kirjandus

- ① G. Hutton. *Programming in Haskell*. Cambridge University Press, 2007.
- ② S. Thompson. *Haskell: The Craft of Functional Programming*. 2nd ed. Addison-Wesley. 1999.
- ③ P. Hudak. *The Haskell School of Expression: Learning Functional Programming through Multimedia*. Cambridge University Press, 2000.
- ④ R. Bird. *Introduction to Functional Programming using Haskell*. 2nd ed. Prentice Hall, 1998.
- ⑤ 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
 - Staatileine 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 abstraktsel 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 (Schönfinkel, H. Curry)
- Lambda-arvutus (A. Church)
- 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)
 - (2007) Haskell' (Haskell-prime)
- Kodulehekülg: <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, ...)
- Definiitsioonid (andmetüüpide, funktsioonide)

Näide

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

Haskell

Tüüpide deklareerimine

$fname_1, fname_2, \dots, fname_n :: type$

- 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

$fname\ arg_1\ \dots\ arg_n = expr$

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

Näide — faktoriaal

```
-- fact1 on defineeritud tingimusavaldise abil  
fact1 n = if n == 0 then 1  
         else n * fact1 (n - 1)
```

NB!

Võrdusoperaator $\equiv$ kirjutatakse programmifailis `==`

Haskell

Avaldise väärtustamine (reduktsioon)

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

Näide

$$\begin{aligned} fact1\ 2 &\implies \text{if } 2 \equiv 0 \text{ then } 1 \text{ else } 2 * fact1\ (2 - 1) \\ &\implies 2 * fact1\ 1 \\ &\implies 2 * (\text{if } 1 \equiv 0 \text{ then } 1 \text{ else } 1 * fact1\ (1 - 1)) \\ &\implies 2 * (1 * fact1\ 0) \\ &\implies 2 * (1 * (\text{if } 0 \equiv 0 \text{ then } 1 \text{ else } 0 * fact1\ (0 - 1))) \\ &\implies 2 * (1 * 0) \\ &\implies 2 \end{aligned}$$

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 -}
```

```
fact2 :: Int → Int
```

```
fact2 0 = 1
```

```
fact2 n = n * fact2 (n - 1)
```

```
-- fact3 on defineeritud "valvurite" abil
```

```
fact3 :: Integer → Integer
```

```
fact3 n | n ≡ 0      = 1
```

```
        | otherwise = n * fact3 (n - 1)
```

Haskell

Variatsioonid faktoriaalile

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

NB!

Võrdlusoperaator $\geq$ kirjutatakse programmfailis $>=$

Haskell

Variatsioonid 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)

-- standardne 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 }
```