Funktsionaalne Programmeerimine

Varmo Vene
Arvutiteaduse Instituut
Tartu Ülikool

EMAIL: varmo@cs.ut.ee

WWW: http://www.cs.ut.ee/~varmo/FP/

LIST: ati.funprog@lists.ut.ee

Kursuse plaan

- Ülevaade keelest Haskell
- λ-arvutus
  - Reduktsioon, Church-Rosseri teoreemid
  - Kombinaatorloogika
  - Arvutatavus, Churchi tees
- Tüübiteooria elemendid
  - Tüübitud λ-arvutus, Curry-Howard isomorfism
  - 2-järku loogikad, polümorfne λ-arvutus
  - Hindley-Milneri tüübisüsteem

Kirjandus

1. P. Hudak. *The Haskell School of Expression*. Cambridge University Press, 2000.
2. R. L. Page. *Two Dozen Short Lessons in Haskell*. Uni. of Oklahoma, 1997.
3. C. Hankin. *Lambda Calculi - A guide for computer scientists*. Oxford University Press, 1994.

Programmeerimiskeelte paradigmad

- Imperatiivsed keeled
  - Protseduursed keeled — Fortran, Pascal, C, Ada
  - Objektorienteeritud keeled — Smalltalk, C++, Java
- Deklaratiivsed keeled
  - Funktsionaalsed keeled — Scheme, ML, Haskell
  - Loogilised keeled — Prolog, Parlog, Gödel, Mercury

Funktsionaalsete keelte põhiomadused

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

Põhjusi FP õppimiseks

- Erinev lähenemine programmeerimisele
  - annab uue vaatenurga tuntud probleemide lahendamiseks

Põhjusi FP õppimiseks

- Erinev lähenemine programmeerimisele
- FP on “kõrgtaseme” paradigma
  - lubab probleeme lahendada väga abstarktsel tasemel
  - kergendab vigade leidmist/vältimist
  - võit produktiivsuses tihti mitmekordne

Põhjusi FP õppimiseks

- Erinev lähenemine programmeerimisele
- FP on “kõrgtaseme” paradigma
- Mõju teiste keelte arengule
  - automaatne mäluhaldus (Java)
  - polümorfism (C++, Ada)
  - algebralised andmetüübid (Pizza)

Põhjusi FP õppimiseks

- Erinev lähenemine programmeerimisele
- FP on “kõrgtaseme” paradigma
- Mõju teiste keelte arengule
- Lihtne semantika
  - formaalne keele kirjeldamine
  - programmide analüüs
  - programmide teisendamine

Funktsionaalsete keelte puudusi

- Mõnede ülesannete 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

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 (Milner 1978)
- Hope, Sasl, Miranda, ... (1980–85)
- Haskell (1988)

Funktsionaalne keel Haskell

- Lühiajalugu

  1987 loodi Haskell'i komitee

  1988 esimene keele kirjeldus (v. 1.0)

  1999 Haskell98 (Standard Haskell)

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

  hugs interpretaator, Portland, Yale

  ghc kompilaator, Glasgow

  hbc kompilaator, Chalmers

  nhc kompilaator, York

Hugs 98

- Hugs = Haskell Users Gofer System
- `http://www.haskell.org/hugs`
  - Win95/NT, Unix, Linux, ...
- Käivitamiseks shelli käsurealt:

```
hugs [options] [file]
```

- Keskkonna muutujad

```
setenv HUGSFLAGS '-E"emacsclient +%d %s"'
```


Haskell

- Programmi struktuur
  - Programm koosneb moodulitest
  - Peamooduli nimi `Main`
  - Vaikimisi laaditakse sisse moodul `Prelude`
- Mooduli struktuur
  - Mooduli päis (mooduli nimi ja eksportlist)
  - Deklaratsioonid (impodid, funktsioonide tüübid, ...)
  - Definitsioonid (andmetüüpide, funktsioonide)
- Näide

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

Haskell

- Tüüpide deklareerimine

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

  - 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

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

Haskell

- Avaldise väärtustamine (reduktsioon)
  - Leitakse funktsiooni defineeriv võrdus
  - Avaldis asendatakse võrduse parema poolega
  - Formaalsed parameetrid asendatakse tegelike argumentidega
  - Kogu protsessi korratakse kuni rohkem ei saa

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

Haskell

- Ühe funktsiooni võib defineerimiseks võib kasutada mitut võrdust
- Väärtustamisel kasutatakse (tekstuaalselt) esimest “sobivat”

```
{- 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)
```