Funktsionaalprogrammeerimise meetod

Varmo VENE
Arvutiteaduse Instituut
Tartu Ülikool

EMAIL: varmo@cs.ut.ee
WWW: http://www.cs.ut.ee/~varmo/MFP2003/
LIST: ati.funprog@lists.ut.ee

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, Gödel, Mercury

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')

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)

Funktsionaalne keel Haskell

- Lühiajalugu

  1987 loodi Haskell'i komitee

  1988 esimene keelekirjeldus (v. 1.0)

  1999 Haskell98 (Standard Haskell)

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

- Realisatsioonid

  hugs interpretaator, Portland, Yale

  ghc kompilaator, Glasgow

  nhc kompilaator, York

  hbc kompilaator, Chalmers

Hugs 98

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

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

- Keskkonnamuutujad:

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

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

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

  - 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

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

Haskell

- Ühe funktsiooni 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)
```

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
-- kasutamise 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

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

- Ilmutatud grupeerimine

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