HPC ja Hat

Aivar Annamaa

25. mai 2009

HPC - Haskell Program Coverage

- HPC aitab tuvastada, millist osa koodist programmi täitmise (testimise) ajal kasutati
  - programm kompileeritakse erilisel moel
  - programmi jooksutatakse, selle käigus genereeritakse "logifailid"
  - saadud faile uuritakse HPC vahenditega
- Kõik vajalik on olemas GHC standardvarustuses (alates versioonist 6.8.1)

HPC - Näide

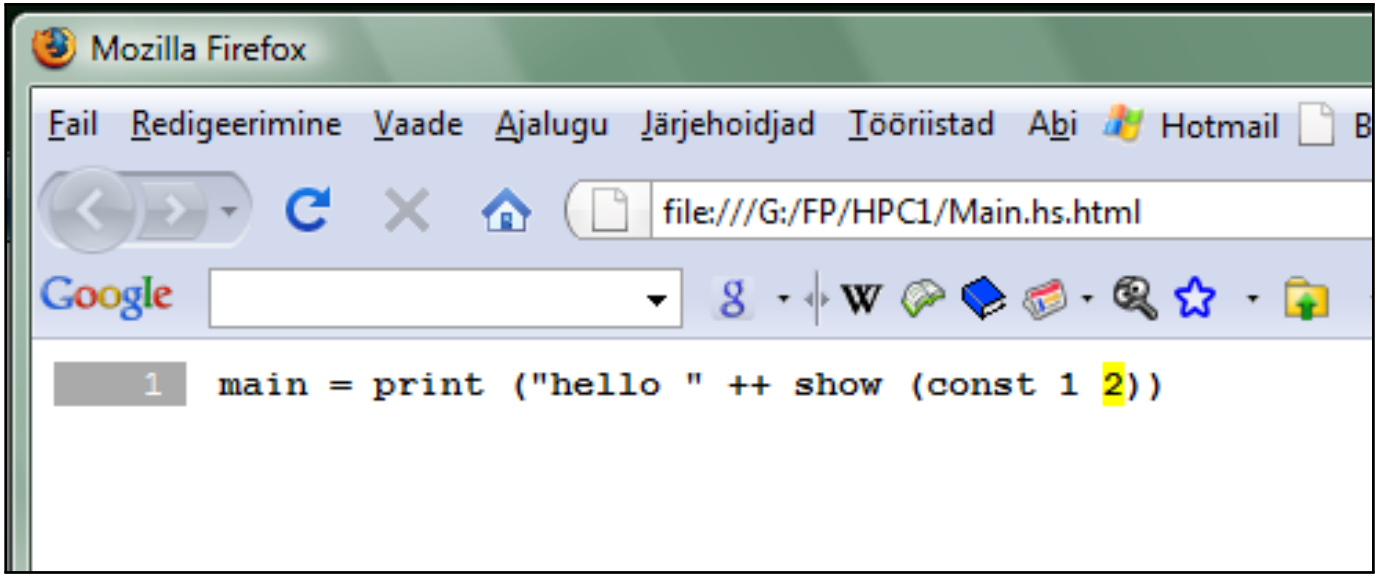
```
> ls
hello.hs
> cat hello.hs
main = print ("hello " ++ show (const 1 2))
> ghc -fhpc hello.hs --make
[1 of 1] Compiling Main             ( hello.hs, hello.o )
Linking hello ...
> ls -a
. .. hello  hello.hi  hello.hs  hello.o .hpc
> ls .hpc
Main.mix
> ./hello
"hello 1"
> ls
. .. hello  hello.hi  hello.hs  hello.o  hello.tix .hpc
```

HPC - Näide

```
> hpc report hello
 85% expressions used (6/7)
100% boolean coverage (0/0)
     100% guards (0/0)
     100% 'if' conditions (0/0)
     100% qualifiers (0/0)
100% alternatives used (0/0)
100% local declarations used (0/0)
100% top-level declarations used (1/1)
```

HPC - Näide

```
> hpc markup hello
Writing: Main.hs.html
Writing: hpc_index.html
Writing: hpc_index_fun.html
Writing: hpc_index_alt.html
Writing: hpc_index_exp.html
> firefox Main.hs.html
```



HPC Suurema programmi näide

```
reciprocal :: Int -> (String, Int)
reciprocal n | n > 1 = ('0' : '.' : digits, recur)
  where
  (digits, recur) = divide n 1 []

divide :: Int -> Int -> [Int] -> (String, Int)
divide n c cs | c `elem` cs = ([], position c cs)
              | r == 0      = (show q, 0)
              | r /= 0      = (show q ++ digits, recur)
  where
  (q, r) = (c*10) `quotRem` n
  (digits, recur) = divide n r (c:cs)

position :: Int -> [Int] -> Int
position n (x:xs) | n==x      = 1
                  | otherwise = 1 + position n xs

showRecip :: Int -> String
showRecip n =
  "1/" ++ show n ++ " = " ++
  if r==0 then d else take p d ++ "(" ++ drop p d ++ ")"
  where
  p = length d - r
  (d, r) = reciprocal n

main = do
  number <- readLn
  putStrLn (showRecip number)
  main
```

HPC Suurema programmi näide

module	Top Level Definitions		Alternatives		Expressions	
	%	covered / total	%	covered / total	%	covered / total
module CSG	100 %	0/0	100 %	0/0	100 %	0/0
module Construct	48 %	17/35	52 %	25/48	60 %	381/635
module Data	24 %	6/25	13 %	11/81	39 %	254/646
module Eval	70 %	22/31	60 %	65/108	57 %	361/628
module Geometry	75 %	42/56	69 %	45/65	70 %	300/427
module Illumination	61 %	11/18	49 %	46/93	46 %	279/600
module Intersections	63 %	14/22	38 %	83/213	38 %	382/1001
module Interval	47 %	8/17	41 %	16/39	41 %	69/165
module Main	100 %	1/1	100 %	1/1	100 %	6/6
module Misc	0 %	0/1	0 %	0/1	0 %	0/10
module Parse	80 %	16/20	68 %	26/38	72 %	192/264
module Primitives	16 %	1/6	16 %	1/6	20 %	5/24
module Surface	36 %	4/11	24 %	13/53	18 %	43/231

hpc programm

```
> hpc
Usage: hpc COMMAND ...

Commands:
  help         Display help for hpc or a single command
Reporting Coverage:
  report       Output textual report about program coverage
  markup       Markup Haskell source with program coverage
Processing Coverage files:
  sum          Sum multiple .tix files in a single .tix file
  combine      Combine two .tix files in a single .tix file
  map          Map a function over a single .tix file
Coverage Overlays:
  overlay      Generate a .tix file from an overlay file
  draft        Generate draft overlay that provides 100% coverage
Others:
  show         Show .tix file in readable, verbose format
  version      Display version for hpc
```

Hat - Haskell Tracer

- Aitab jälgida avaldiste väärtustamist programmi täitmise ajal
  - programm komipleeritakse eriliselt
  - jooksutamise käik salvestatakse logifaili
  - logifaili uuritakse Hat vahenditega
- `http://www.haskell.org/hat`
- Paigaldamine
  - vajab hmake-i (ei tule GHC-ga kaasa)
  - 2.06 andis vigu, tööversioon (21.05.09 seisuga) toimis
  - Vanem versioon olemas ka Windows-i jaoks (aga ei tööta GHC 6.10-ga)

Hat - näide

Insort.hs

```
sort :: Ord a => [a] -> [a]
sort []     = []
sort (x:xs) = insert x (sort xs)

insert :: Ord a => a -> [a] -> [a]
insert x []     = [x]
insert x (y:ys) = if x <= y then x : y : ys
                  else y : insert x ys

main = putStrLn (sort "program")
```

Hat - näide

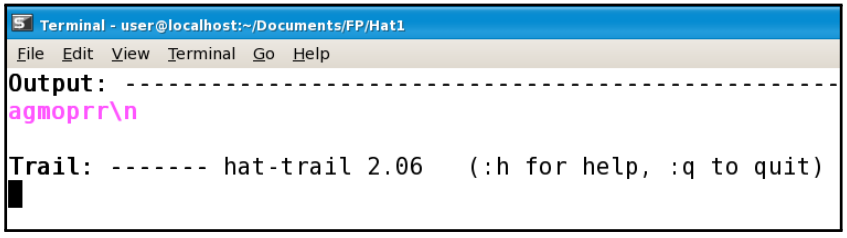
```
> hmake -hat Insort
hat-trans   Insort.hs
Creating directories Hat
Wrote Hat/Insort.hs
ghc     -c -package hat -o Hat/Insort.o Hat/Insort.hs
ghc     -package hat  -o Insort Hat/Insort.o
> ./Insort
agmoprr
> ls
Hat     Insort.hat          Insort.hat.output  Insort.hx
Insort  Insort.hat.bridge   Insort.hs
```

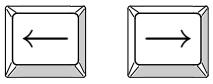
hat-trail

- `hat-trail` on interaktiivne konsooliprogramm
- vastab küsimusele "Kust see tuli?"
- "see" võib olla:
  - Viga
  - Väljund
  - non-value expr
  - väärtused
-

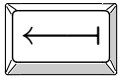
hat-trail

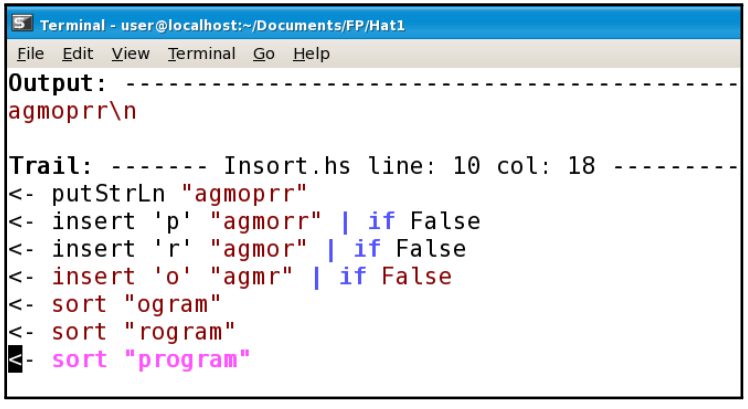
```
> hat-trail Insort
```



 - aktiivse alamavaldise valimiseks

 - läheb valitud avaldise "sisse"

 (backspace) - läheb tagasi vanem-avaldisse



hat-observe

- `hat-observe` on käsurea põhine konsooliprogramm
- vastab küsimusele "Kuidas see funktsioon käitus?"

hat-observe

```
> hat-observe Insort
hat-observe> :info
Main
20 insert   1 main     8 sort
Prelude (trusted)
17 <=         18 ==          1 error       1 putStrLn

hat-observe> sort
1 sort "program" = 'a':'g':'m':'o':_|_
2 sort "rogram" = 'a':'g':'m':'o':_|_
3 sort "ogram" = "agmor"
4 sort "gram" = "agmr"
5 sort "ram" = "amr"
6 sort "am" = "am"
7 sort "m" = "m"
8 sort [] = []
hat-observe>
```

hat-observe näidistega

```
hat-observe> insert 'g' _
1 insert 'g' "amr" = "agmr"
2 insert 'g' "mr" = "gmr"
hat-observe> insert _ _ = [_]
1 insert 'm' [] = "m"
2 insert 'r' [] = "r"
hat-observe> sort in main
1 sort "program" = 'a':'g':'m':'o':_|_
hat-observe> sort in sort
1 sort "rogram" = 'a':'g':'m':'o':_|_
2 sort "ogram" = "agmor"
3 sort "gram" = "agmr"
4 sort "ram" = "amr"
5 sort "am" = "am"
6 sort "m" = "m"
7 sort [] = []
```

teised hat vahendid

- hat-explore: võimaldab läbi arvutuse sammuda
- hat-stack: näitab katkestatud arvutuse funktsioonide magasini
- hat-detect: küsib kasutajalt "kas see reduktsioon on õige?"