

Haskell for Lisp Programmers

Erik Charlebois

March 4th, 2009

Outline

Differences between Lisp and Haskell

- Syntax

- Static Types

- Laziness

- Purity

The Haskell Platform

- cabal

- Hackage

- cabal-install

- Other tools

What I'm up to...

- Generating bindings to C++

Differences between Lisp and Haskell

Differences between Lisp and Haskell

Lisp and Haskell are opposites.

Haskell	Lisp
Syntax	No Syntax
Statically Typed	Dynamically Typed
Pure Functions	Impure Functions
Lazy Evaluation	Eager Evaluation
Compile and Run	Image-Based

Syntax

Lists and Tuples

```
> 1 : [2,3]
[1, 2, 3] :: [Integer]
> (1, "Foo")
(1, "Foo") :: (Integer, [Char])
```

Function Definition

```
cons x y = (:) x y
cons = (:)
:type cons
cons :: forall a. a -> [a] -> [a]
```

Function Application

```
cons 1 (cons 2 [])
cons 1 $ cons 2 []
> :type ($)
($) :: forall a b. (a -> b) -> a -> b
```

Syntax

Infix Functions

```
> :info (:)
data [] a = [] | a : [a]
infixr 5 :

infixr 6 `cons`
1 `cons` 2 : [] -- error, ((:) (cons 1 2) [])

infixr 4 `cons`
cons = (:)
1 `cons` 2 : [] -- works, (cons 1 ((:) 2 []))
```

Syntax

Let and Where

```
add x y = let a = x
           b = y
           in x + y
```

```
add x y = a + b
  where a = x
        b = y
```

Case and Guards

```
foo x y = case x of
           0 -> bar y
           n -> baz x y
```

```
foo x y | x == 0 = bar y
foo x y          = baz x yeleganKin
```

Static Types

Types and Kinds

```
> :kind Char
Char :: *
> :kind []
[] :: * -> *
> :kind [] Char
[] Char :: *
> :type "foo"
"foo" :: [Char]
```

Algebraic Data Types

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

```
> :kind Tree
Tree :: * -> *
> :type Leaf
Leaf :: forall a. a -> Tree a
> :type Node
Node :: forall a. Tree a -> Tree a -> Tree a
```

Static Types

Pattern Matching

```
map fn []      = []  
map fn (x:xs) = fn x : map fn xs
```

Type Classes

```
class Monoid a where
```

```
  mempty  :: a
```

```
  mappend :: a -> a -> a
```

```
class Foldable t where
```

```
  fold :: Monoid m => t m -> m
```

```
instance Foldable Tree where
```

```
  fold (Leaf a)    = a
```

```
  fold (Node a b) = fold a `mappend` fold b
```

```
> fold (Node (Node (Leaf [3]) (Leaf [4])) (Leaf [5]))  
[3,4,5] :: [Integer]
```

Laziness

Custom control flow

```
myif True tp _ = tp  
myif False _ fp = fp
```

Out of order initialization

```
let x = y + 1; y = 3 in x
```

Infinite lists

```
> let x = 1 : 2 : 3 : x  
> x  
[1,2,3,1,2,3,1,2,3,1,2,3,1,...]  
> take 10 x  
[1,2,3,1,2,3,1,2,3,1]
```

Purity

- ▶ Functions are pure – no side effects
- ▶ Sharing state or enforcing ordering requires threading a value through calls
- ▶ Threading is programmable – Monad type class
- ▶ Do notation makes things more readable

Monads

```
class Monad m where  
  return :: a -> m a  
  (>>=)  :: m a -> (a -> m b) -> m b
```

```
data State s a = State (s -> (s, a))
```

```
instance Monad (State s) where  
  return a = State (\s -> (s, a))  
  (>>=) (State lhs) rhs = State (\s -> case lhs s of  
    (s', a) -> case rhs a of  
      (State fn) -> fn s')
```

```
getState :: State s s  
getState = State (\s -> (s, s))
```

```
putState :: s -> State s ()  
putState s' = State (\s -> (s', ()))
```

```
runState :: State s a -> s -> (s, a)  
runState (State fn) a = fn a
```

Monads

```
incrementBy1 = getState >>= \s -> putState (s + 1)
```

```
morefun_nosugar = incrementBy1 >>= \_ ->
  getState >>= \s ->
  incrementBy1 >>= \_ -> -- has no effect
  putState s -- because we put an old state here
```

```
morefun_withsugar = do
  incrementBy1
  s <- getState
  incrementBy1
  putState s
```

```
runfun = runState morefun_withsugar 10
```

```
> runfun
11 :: Integer
```

The Haskell Platform

Haskell Platform

A 2008 proposal for a dependable Haskell platform

- ▶ cabal – a standard build system
- ▶ hackage – a single site for package hosting
- ▶ cabal-install – a standard package management tool

cabal

- ▶ Allow developers to easily build conforming packages
- ▶ Operating system independent – works on Windows!
- ▶ Capable of building mixtures of C and Haskell

```
Name:          interpolatedstring-qq
Version:       0.1
License:       BSD3
License-file:  LICENSE
Category:      Data
Author:        Erik Charlebois
```

library

```
extensions:    TemplateHaskell
build-depends: base, template-haskell, haskell-src-meta
hs-source-dirs: src
exposed-modules: Text.InterpolatedString.QQ
```

Hackage

- ▶ Repository of published packages
- ▶ Similar to CPAN, CTAN, Boost, etc
- ▶ Haddock documentation for all packages is hosted
- ▶ Anyone can upload via web or cabal

.NET

- [hs-dotnet](#) library: Pragmatic .NET interop for Haskell

AI

- [hfann](#) library and program: Haskell binding to the FANN library
- [hgalib](#) library: Haskell Genetic Algorithm Library
- [hpylos](#) program: AI of Pylos game with GLUT interface.
- [mines](#) program: Minesweeper simulation using neural networks

Algorithms

- [binary-search](#) library: Binary and exponential searches
- [DecisionTree](#) library: A very simple implementation of decision trees for discrete attributes.
- [Diff](#) library: $O(ND)$ diff algorithm in haskell.
- [edit-distance](#) library and programs: Levenshtein and restricted Damerau-Levenshtein edit distances
- [funsat](#) library and program: A modern DPLL-style SAT solver
- [garsia-wachs](#) library: A Functional Implementation of the Garsia-Wachs Algorithm
- [Graphalyze](#) library: Graph-Theoretic Analysis library.
- [GraphSCC](#) library: Tarjan's algorithm for computing the strongly connected components of a graph.
- [hgal](#) library: library for computation automorphism group and canonical labelling of a graph
- [hmm](#) library: Hidden Markov Model algorithms
- [incremental-sat-solver](#) library: Simple, Incremental SAT Solving as a Library

cabal-install

- ▶ Tool for automatically pulling dependencies and building them
- ▶ Similar to apt-get, emerge, etc

```
erikc@emac:~$ sudo cabal install hint --global
Resolving dependencies...
Downloading hint-0.3.1.0...
Configuring hint-0.3.1.0...
Preprocessing library hint-0.3.1.0...
Building hint-0.3.1.0...
[ 1 of 21] Compiling Hint.Util
...
[21 of 21] Compiling Language.Haskell.Interpreter.GHC
ar: creating archive dist/build/libHShint-0.3.1.0.a
Installing library in /usr/local/lib/hint-0.3.1.0/ghc-6.10.1
Registering hint-0.3.1.0...
Reading package info from "dist/installed-pkg-config" ... done.
Writing new package config file... done.
```

Other tools

darcs	Distributed source control
quickcheck	Random testing for pure functions
hunit	Unit testing for non-pure functions
hs-lint	Lint for Haskell
haddock	Javadoc-style documentation
hscolor	Syntax highlighting
hoogle	Type-based API search
ghc-core	Dumps internal core language used by GHC compiler

Community Activity

- ▶ Research
 - ▶ Formal verification
 - ▶ Parallelism
 - ▶ Type theory
- ▶ Commercial
 - ▶ Financial sector (Credit Suisse)
 - ▶ Security and encryption (Galois)
 - ▶ Hardware design (Bluespec)
 - ▶ Procedural city generation (gamr7)
 - ▶ Multimedia content creation (Anygma)
- ▶ Open Source
 - ▶ Languages (GHC, Pugs)
 - ▶ Window Manager (xmonad)
 - ▶ Source Control (darcs)
 - ▶ Build System, Packaging (cabal)
 - ▶ Web Applications (happs, hoogle)

What I'm up to...

Personal Preferences

Why I prefer Haskell to Lisp...

- ▶ Platform and community
- ▶ A good, free implementation
- ▶ Strict typing
- ▶ Works the same across Mac, Linux and Windows

What I miss from Lisp...

- ▶ Macros
- ▶ Mixture of compilation and interpretation
- ▶ Easy runtime extension

Generating bindings to C++

- ▶ Use Dwarf debug information in C++ object files to generate Lua bindings that can access C++ code
- ▶ A Lua debugger expression evaluator for C++
- ▶ Tool Components
 - ▶ Data.Elf
 - ▶ Data.Dwarf
 - ▶ Data.Macho
 - ▶ Data.Pecoff
 - ▶ InterpolatedString.QQ
- ▶ Runtime Components
 - ▶ "Glua" tool
 - ▶ Lua runtime
 - ▶ C runtime
- ▶ Interpolated strings uses Template Haskell and Quasiquote to do custom syntax for Ruby-style string interpolation

```
[$istr|x + y is #{x + y}||] -> "x + y is " ++ show (x + y)
```

Questions?

Interpolated Strings

```
{-# LANGUAGE TemplateHaskell #-}
module Text.InterpolatedString.QQ (istr) where

import qualified Language.Haskell.TH as TH
import Language.Haskell.TH.Quote
import Language.Haskell.Meta.Parse

-- parseExp :: String -> Either String Exp
-- appE :: ExpQ -> ExpQ -> ExpQ

data Part = Lit String
          | Quote String
          deriving Show

parseHaskell :: String -> String -> String
parseHaskell a [] = [Lit (reverse a)]
parseHaskell a ('\\\' : x:xs) = parseHaskell (x:a) xs
parseHaskell a ('\\\' : []) = parseHaskell ('\\\' : a) []
parseHaskell a ('}':xs) = Quote (reverse a) : parseStr [] xs
parseHaskell a (x:xs) = parseHaskell (x:a) xs
```

Interpolated Strings

```
parseStr :: String -> String -> [Part]
parseStr a [] = [Lit (reverse a)]
parseStr a ('\\\' :x:xs) = parseStr (x:a) xs
parseStr a ('\\\' :[]) = parseStr ('\\\' :a) []
parseStr a ('#': '{':xs) = Lit (reverse a) : parseHaskell "" xs
parseStr a (x:xs) = parseStr (x:a) xs
```

```
makeExpr :: [Part] -> TH.AppE
makeExpr [] = [| "" |]
makeExpr ((Lit a):xs) = TH.appE [| (++) a |] (makeExpr xs)
makeExpr ((Quote a):xs) =
  TH.appE [| (++) (show $(strToHaskell a)) |] (makeExpr xs)
```

```
strToHaskell :: String -> Exp
strToHaskell s = case parseExp s of
  Left s -> TH.report True s >> [| "" |]
  Right e -> return e
```

```
istr :: QuasiQuoter
istr = QuasiQuoter (makeExpr $ parseStr "") undefined
```

Interpolated Strings

```
luaBind dig di | infoTag di == kDW_TAG_member =
  let attributes = infoAttributes di
      typeName   = getTypeName dig (attributes ! kDW_AT_type)
  in [$istr|
    [{"#{attributes ! kDW_AT_name}"}] = {
      typeof   = "#{typeName}",
      offset   = #{attributes ! kDW_AT_data_member_location},
      bitsize  = #{attributes ! kDW_AT_bit_size},
      bitoff   = #{attributes ! kDW_AT_bit_offset}, },|]

luaBind dig di | infoTag di == kDW_TAG_structure_type =
  let fullName = getTypeName dig di
      sizeof    = infoAttrs di ! kDW_AT_byte_size
      members   = filter (== kDW_TAG_member) (infoChilds di)
  in [$istr|
    [{"#{fullName}"}] = {
      mtable = glua.struct_table,
      sizeof = #{sizeof},
      member = { #{concat (map (luaBind dig) members)} },
    }|]
```