Getting Started With Functional Programming in Haskell

Jan van Eijck

CWI, Amsterdam and Uil-OTS, Utrecht

jve@cwi.nl

April 21, 2009

Abstract

The purpose of this lecture is to give a lightning introduction to the functional programming language Haskell, and to make preparations for using Haskell for understanding more about language, maths and logic.

Learning Something New: Key ingredients

New Facts You will learn a few facts about how functional programs are written.

New Skills The main focus of this lecture.

- skills in (functional) computation, in learning to think functionally
- skills in representation, in getting from definitions to programs, in 'seeing' the program hidden in a definition.
- skills in working with 'the stuff of language, maths and logic'.

Attitude The most important thing. But how do you acquire it? Once you have acquired the correct attitude you can learn to do anything.

Types in Grammar and Types in Programming

Adjectives are words that combine with nouns to form complex nouns: nice combines with the noun guy to form the noun nice guy. In the grammar formalism of categorial grammar, one calls A/B the type of a word that needs a type B word to its right in order to produce a type A word, according to the rule:

$$A/B + B = A.$$

Applying this to the case of adjectives and nouns: if nouns have type N, then adjectives have type N/N, and the complex noun nice guy is produced by the rule:

$$\text{nice}_{N/N} + \text{guy}_N = (\text{nice guy})_N.$$

Here, N/N is the grammatical function, N the grammatical argument.

Adverbials are words that map adjectives into complex adjectives:

$$\begin{aligned}&\text{very}_{(N/N)/(N/N)} + \text{nice}_{N/N} + \text{guy}_N \\ &= (\text{very nice})_{N/N} + \text{guy}_N \\ &= (\text{very nice guy})_N.\end{aligned}$$

Similarly, $B\backslash A$ is the type of a word that needs a B type word on its lefthand side to produce an A type word. In English, an adjective like emeritus behaves like this:

$$\text{professor}_N + \text{emeritus}_{N\backslash N} = (\text{professor emeritus})_N.$$

Here $N\backslash N$ is the function, N the argument.

Without the directional information

$\text{very}_{(N\to N)\to(N\to N)}\ \text{nice}_{N\to N}\ \text{guy}_N = (\text{very nice})_{N\to N}\ \text{guy}_N = (\text{very nice guy})_N$

$(\text{emeritus})_{N\to N}\ \text{professor}_N = (\text{professor emeritus})_N.$

Use of types in programming

If integer numbers have type `Int`, then we get:

- addition of integer numbers has type `Int -> Int -> Int`.
- incrementing integers by 1 has type `Int -> Int`.
- squaring integers has type `Int -> Int`.

This information is very useful to check whether a program is well typed.

Checking types is a handy way to spot common mistakes in programming.

Functions and Functional Programming

Functional Programming is programming with functions.

A well-known functional programming language is Haskell, named after the logician Haskell B. Curry.

Hugs is the implementation of Haskell that we are going to use. See http://www.haskell.org/hugs/.

A textbook on Haskell that bridges the gap between reasoning and programming is [2].

In this course we will illustrate material from [10] by means of Haskell implementations.

Functions

A function from a set A to a set B is an instruction to link each element of A to an element of B.

See Chapter 2 in [10].

Notation:

$f : A \to B$.

In the context of programming the sets are called types, and the notation is as follows:

```
f :: a -> b
```

This is called: the type of the program f.

The instruction for the function itself is the program for f.

Example

A program reversal for the reversal of a string.

The type is `String -> String`.

The instruction explains how lists of characters are reversed.

For you to do: write the instruction.

Using the Hugs Haskell Interpreter

```
jve@vuur:~/courses/twl2008$ hugs
___   __ __  __  ____   ___      ----------------------------------------
||   || ||  || ||  || ||__      Hugs 98: Based on the Haskell 98 standard
||___|| ||__|| ||__||  __||     Copyright (c) 1994-2005
||---||         ___||           World Wide Web: http://haskell.org/hugs
||   ||                         Report bugs to: hugs-bugs@haskell.org
||   || Version: March 2005     ----------------------------------------

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
Hugs.Base>
```

http://haskell.org/hugs

Using the GHCI Haskell Interpreter

```
jve@vuur:~/courses/twl2008$ ghci
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.6, for Haskell 98.
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
Prelude>
```

http://www.haskell.org/ghc/

Haskell

These slides form a literate program. The text you are reading is the documentation. The actual code is the part typeset in frames. This is how the code begins:

```
module GSWH

where
import List
import Char
```

This declares a module and imports two other modules. The code of the module consists of the text in frames.

Loading the module

```
jve@vuur:~/courses/twl2008$ ghci GSWH
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.6, for Haskell 98.
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
[1 of 1] Compiling GSWH             ( GSWH.lhs, interpreted )
Ok, modules loaded: GSWH.
*GSWH>
```

About Haskell

Haskell was named after the logician Haskell B. Curry. Curry, together with Alonzo Church, laid the foundations of functional computation in the era BC (Before the Computer), around 1940.

Haskell is a functional programming language, and a member of the Lisp family. Others family members are Scheme, ML, Occam, Clean. Haskell98 is intended as a standard for lazy functional programming.

With Haskell, the step from formal definition to program is particularly easy. This presupposes, of course, that you are at ease with formal definitions.

Our reason for combining training in reasoning with an introduction to functional programming is that your programming needs will provide motivation for improving your reasoning skills.

Implementation of a Prime Number Test

```
ld n = ldf 2 n

divides d n = rem n d == 0

ldf k n | divides k n = k
        | k^2 > n     = n
        | otherwise   = ldf (k+1) n

prime n | n < 1     = error "not a positive integer"
        | n == 1    = False
        | otherwise = ld n == n
```

Trying it out

```
somePrimes    = filter prime [1..1000]

primesUntil n = filter prime [1..n]

allPrimes     = filter prime [1..]
```

More on the `filter` function below.

```
GSWH> primesUntil 50
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
GSWH>
```

Reading

- On Logic and Maths in Linguistics: [10].
- On Logic, Maths and Functional Programming: [2].
- On Computational Linguistics and Functional Programming: [3], available from http://www.cwi.nl/~jve/cs.
- On Haskell: [1], [11],[4],[9], [8],[5],[7].
- On Grammars and Parsing: [6], available from http://www.cs.uu.nl/docs/vakken/gont/diktaat.pdf.

References

[1] Hal Daume. Yet another Haskell tutorial. www.cs.utah.edu/~hal/docs/daume02yaht.pdf.

[2] K. Doets and J. van Eijck. The Haskell Road to Logic, Maths and Programming, volume 4 of Texts in Computing. College Publications, London, 2004.

[3] Jan van Eijck and Christina Unger. Computational Semantics with Functional Programming. To appear with Cambridge University Press, 2009.

[4] The Haskell Team. The Haskell homepage. http://www.haskell.org.

[5] P. Hudak, J. Fasel, and J. Peterson. A gentle introduction to

Haskell. Technical report, Yale University, 1996. Online version: http://www.haskell.org/tutorial/.

[6] J. Jeuring and D. Swierstra. Grammars and parsing. Lecture Notes, Utrecht University, 2001.

[7] Mark P. Jones, Alastair Reid, et al. The Hugs98 user manual. http://cvs.haskell.org/Hugs/pages/hugsman/index.html.

[8] S. Peyton Jones, editor. Haskell 98 Language and Libraries; The Revised Report. Cambridge University Press, 2003.

[9] S. Peyton Jones, J. Hughes, et al. Report on the programming language Haskell 98. Available from the Haskell homepage: http://www.haskell.org, 1999.

[10] Barbara H. Partee, Alice ter Meulen, and Robert E. Wall. Mathematical Methods in Linguistics. Kluwer Academic Publishers, 1993.

[11] The GHC Team. The Glasgow Haskell compiler (GHC). http://www.haskell.org/ghc/.