Krishnamoorthy V. Iyer, Soumen Ganguly and Professor Amitabha Sanyal

December 15, 2015

Video Tutorials for Self-Learners and Enthusiasts: Functional Data Structures and Algorithms in Haskell

**Abstract**

Functional programming paradigms hold promise to effectively manage the increasing complexity of software, accounting for their recent – and increasing – popularity. It would be worthwhile to train a new generation of mathematically savvy programmers with backgrounds in Mathematics, Engineering, and the hard sciences (especially Physics) in a functional language like Haskell. Our proposal is a first step in that direction.

# 1 Motivation

As software becomes more complex, code reusability (and comprehensibility!) becomes an ever more pressing issue. Consequently, functional programming paradigms will gain in popularity. Haskell offers many advantages in this regard, chief among them being purity and equational reasoning. Together, they:

- Simplify code reusability and comprehensibility, and
- Enable mathematicians to effectively leverage their skills and knowledge.

Haskell is often favored by theorists and mathematicians who are less interested in the implementation details of a program i.e. they would rather that the program(mer) tell the computer what needs to be done rather than telling the computer how to do it.[1]

Imperative programming paradigms (and languages based thereon) have a decided advantage over functional paradigms when it comes to memory usage – in both theory and practice. In the early days (circa 1960s) of programming, memory was at a premium. Functional paradigms' inefficient utilisation of memory (in comparison with imperative paradigms) was a central issue of concern.

*This historical accident – which is no longer as much of an issue as it once was (since nowadays, memory is one of the least expensive resources) – has led to the overwhelming dominance of imperative paradigms in industry and academia.* As a by-product, students – both Computer Science (CS) and non-CS majors – are mostly exposed to imperative paradigms in their undergraduate studies.

[1] A surprising and pleasing consequence of this is that Haskell programs often better reflect thinking styles that seem more natural to mathematicians and mathematically oriented scientists from non-CS backgrounds. Two (non)-examples are Haskell Quicksort and Meertens' Prime Number generator [1]. Non-examples because despite their subtle inadequacies and flaws, they are often used to "sell" Haskell. But there exist better ones.

In the present day, one reason for the continuing popularity of imperative languages and paradigms is that there exist multifarious online tutorials (text and video) addressed to a wide variety of audiences and communities. These tutorials do not merely teach language constructs. They also teach language specific implementations of data structures (DS) and algorithms, making it easy for self-learners to come upto speed and gain confidence.

Learning an imperative language's syntax is by no means sufficient to employ it effectively. The same holds for functional languages as well. Keeping this in mind, we aim to create a set of tutorials that enable self-learners in Haskell – and functional programmers in general – to have easy access to resources on Haskell implementations of data structures and algorithms.

The tutorials are not aimed (not primarily at least) at teaching Haskell syntax – that job is already done effectively by many online resources such as:

- Textbooks:
  - Miran Lipovaca's *Learn You a Haskell for Great Good* [2]
  - O'Sullivan, Stewart and Goerzen's *Real World Haskell* [3]
- Video Lecture Courses:
  - Professor Erik Meijer [4]
  - Professor Jurgen Giesl [5]

## 2 Intended Audience and Target Group

Functional programming in general, and Haskell in particular, is perceived as an elite paradigm and language, meant for mathematically able students of CS at the Masters' level and beyond. As taught in our universities, M.S and M. Tech students of CS have already had exposure to the fundamentals of CS, and have many courses on data structures and algorithms under their belt by the time they take course(s) in functional programming, *if they do*. It is relatively easy for them to understand, implement, and be cognizant of the advantages and pitfalls of DS and Algorithms in a functional setting.

CS instructors at the undergraduate level in any good university will first explain the DS/Algorithm and then discuss the implementation in a specific language. *Without gaining a good deal of practice with implementation, a student's knowledge is not well-grounded.*

Indeed, as one of us – Krishnamoorthy Iyer – can attest, mathematically able students from non-CS backgrounds such as Electrical Engineering, Mathematics, and Physics are often not even aware of the existence of the functional programming paradigm, and the many advantages it can bestow upon those who are willing and able to negotiate the initially steeper learning curve of a language such as Haskell.[2]

The aim of this projected sequence of video tutorials is to address and popularise functional programming among an audience of mathematically savvy students from Mathematics, Statistics, (Electrical) Engineering and Physics whose exposure to DS and Algorithms is perhaps not so strong as that of a classical

[2]Krishnamoorthy Iyer learnt of functional programming from his friend Prasanna Kumar, a Ph.D student of Prof. Amitabha Sanyal, during the course of a casual conversation.

CS student. One of us, Krishnamoorthy Iyer, can testify that Prof. Amitabha Sanyal's course on functional programming came as an eye-opener.

But for members of this group, what acts as a disincentive even to those who are willing to put in the effort to negotiate the steep learning curve is the absence of good video tutorials discussing the implementations of functional DS and algorithms.

This is the lacuna we would like to address.

# 3 Projected Work

About ourselves:

- Krishnamoorthy Iyer is a Ph.D candidate in Electrical Engineering (EE). He expects to finish in about six months. He has credited and audited basic courses in DS and Algorithms, and on Algorithm Design during his Ph.D studies. He is – and considers himself to be – primarily an EE guy who has been captivated by the beauty and elegance of the functional programming paradigm and would like to gain more programming expertise by writing programs.

- Soumen Ganguly is a CS major. He has been with FOSSEE these past six months, and has plenty of programming experience, having spent a Google Summer of Code at the end of his sophomore year. He does not know Haskell, but is extremely keen to learn and make it part of his programming language toolbox.

We bring to the table strengths that are complementary and are confident that we will be able to work together effectively.

## 3.1 Beginning

Learning:

- *Weeks 1-3*: Bringing Soumen upto speed with the basics of Haskell's language constructs. We envisage that the primary difficulty is that since a functional language like Haskell is non-trivially different from imperative programming languages, it may take a little while to get used to the programming constructs. We will make full use of available online text and video resources mentioned above, besides others we come across.

- *Weeks 4-6*: During this period, we will explore the use of arrays and graphs in Haskell. Haskell libraries currently support nine different kinds of array constructors [6], which are optimized for different needs. Learning to navigate through this embarrassment of riches is a *sine qua non* for the serious functional programmer. We will also explore the graph library documentation [7], [8] and the functional graph library (fgl) package [9], and also study some of the simpler graph algorithm implementations in Haskell, such as breadth-first search (BFS) and numbering [10] and depth-first search (DFS) [11], and expanding a graph [12].

Basic Tutorials:

- *Weeks 7-10*: Get started on:
  - Chris Okasaki's *Purely Functional Data Structures* [13] and his EdisonCore package [14] [15] – a library of efficient, purely functional data structures and their Haskell implementations, and create tutorials based on them.
  - Richard Bird's *Pearls of Functional Algorithm Design* [16] – a collection of elegant functional algorithms aka "pearls".

  We would also like to make tutorials for at least the following thirteen pearls from [16], to give a flavor of the power of functional algorithm design:
  - 2: A surpassing problem
  - 7: Building a tree with mimimum height (see also [17])
  - 8: Unraveling greedy algorithms
  - 9: Finding celebrities
  - 12: Ranking suffixes
  - 13: The Burrows-Wheeler transform
  - 14: The last tail
  - 15: All the common prefixes
  - 16: The Boyer-Moore algorithm
  - 17: The Knuth-Morris-Pratt algorithm
  - 23: Inside the convex hull
  - 24: Rational arithmetic coding
  - 25: Integer arithmetic coding

  Pearls 12-17 are popular string algorithms. Contingent on the project's success, they may constitute a good start to developing a library of tutorials on Haskell implementations of string algorithms. (An ambitious plan for the future would be to implement the algorithms in Gusfield [18], see also [19]).

  Pearls 13, 24 and 25 are (lossless) data compression algorithms. 24 and 25 deal with arithmetic coding (studied in introductory information theory).

  Other possibilities:
  - Standard Binary Heaps [20]
  - Algorithms on Braun Trees [21]
  - Mingling Streams [22]

### 3.2 Intermediate

Intermediate Tutorials:

- *Weeks 11-13*:
  - Examples of algorithms of interest to mathematicians:
    - Enumerating the Rationals [23]
    - Generating Primes [24]
    - Generating Fractals:
      - the Bird tree [25]
      - the Mandelbrot set [26]
  - Examples of algorithms of interest to (electrical) engineers (from a first course in Information Theory):
    - Huffman
    - Arithmetic (see the previous subsection)
    - Blahut-Arimoto
    - Lempel-Ziv
  - Functional DS of independent interest
    - Zipper [27], the generic zipper [28], and the monad zipper [29]
    - Finger Trees [30]
  - Ukkonen's algorithm: Online construction of suffix trees
  - The probability monad and probabilistic functional programming by [31]. This could segue into an exploration of the HLearn machine learning toolbox [32] (claimed to be better than WEKA [33]).
  - Information Bottleneck Method [34] implementation and tutorial
  - Backtracking: Sudoku [35], Tiling a chessboard [36]

## 4 Conclusion

Depending on the project's success, two further avenues that may be worth exploring:

- Prof. Prabhu Ramachandran mentioned to one of us (Krishnamoorthy Iyer) the need to create tutorials in Python with a functional flavor. We are confident that the experience gained in this project will be useful.
- Haskell may also act as a gateway to LISP, used widely in the Artificial Intelligence community. To take one example, Prof. David Touretzky uses LISP in his researches into AI and Computational Neuroscience.

It is our hope that the video tutorials will spark and sustain interest in functional programming paradigms that are becoming increasingly popular. Thus leading to an increase in the size of the community of dedicated Haskellers in particular and functional programmers in general.

# References


[1] L. Meertens, "Calculating the Sieve of Eratosthenes," *J. Functional Programming*, vol. 14, no. 6, pp. 759–763, 2004.

[2] M. Lipovaca, "Learn You a Haskell for Great Good! A Beginner's Guide," http://learnyouahaskell.com/.

[3] B. O'Sullivan, D. Stewart, and J. Goerzen, "Real World Haskell," http://book.realworldhaskell.org/.

[4] E. Meijer, "Functional Programming Fundamentals," https://channel9.msdn.com/Series/C9-Lectures-Erik-Meijer-Functional-Programming-Fundamentals.

[5] J. Giesl, "Funktionale Programmierung," http://video.s-inf.de/#FP.2005-SS-Giesl.(COt).HD_Videoaufzeichnung.

[6] https://wiki.haskell.org/Arrays.

[7] M. Erwig, "Inductive Graphs and Functional Graph Algorithms," *J. Functional Programming*, vol. 11, no. 5, pp. 467–492, 2001.

[8] ——, "Fully Persistent Graphs - Which One To Choose?" in *9th Int. Workshop on Implementation of Functional Languages. LNCS 1467*, 1997, pp. 123–140.

[9] ——, "The fgl package," https://hackage.haskell.org/package/fgl.

[10] C. Okasaki, "Breadth-First Numbering: Lessons from a Small Exercise in Algorithm Design," *J. Functional Programming.*

[11] D. J. King and J. Launchbury, "Structuring Depth-First Search Algorithms in Haskell." ACM Press, 1995, pp. 344–354.

[12] R. S. Bird, "An in-situ algorithm for expanding a graph," *J. Functional Programming*, vol. 23, no. 2, pp. 174–184, 2013.

[13] C. Okasaki, *Purely Functional Data Structures.* Cambridge University Press, 1998.

[14] ——, "An Overview of Edison," in *Electronic Notes in Theoretical Computer Science.* Elsevier, 2000, pp. 34–54.

[15] ——, "The Edison Core package," https://hackage.haskell.org/package/EdisonCore.

[16] R. Bird, *Pearls of Functional Algorithm Design.* Cambridge University Press, 2010.

[17] R. S. Bird, "On building trees with minimum height," *J. Functional Programming*, vol. 7, no. 4, pp. 441–445, 1997.

[18] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[19] D. Wakeling, "Biological sequence similarity," *J. Functional Programming*, vol. 16, no. 1, pp. 1–12, 2006.

[20] V. Kostyukov, "A Functional Approach to Standard Binary Heaps," *CoRR*, vol. abs/1312.4666, 2013. [Online]. Available: http://arxiv.org/abs/1312.4666

[21] C. Okasaki, "Three Algorithms on Braun Trees," *J. Functional Programming*, vol. 7, no. 6, pp. 661–666, 1997.

[22] R. S. Bird, "How to mingle streams," *J. Functional Programming*, vol. 25, 2015.

[23] J. Gibbons, D. Lester, and R. Bird, "Enumerating the Rationals," *J. Functional Programming*, vol. 16, no. 3, pp. 281–291, 2006.

[24] M. E. O'Neill, "The Genuine Sieve of Eratosthenes," *J. Functional Programming*, vol. 19, no. 1, pp. 95–106, 2009.

[25] R. Hinze, "The Bird tree," *J. Functional Programming*, vol. 19, no. 5, pp. 491–508, 2009.

[26] M. P. Jones, "Composing fractals," *J. Functional Programming*, vol. 14, no. 6, pp. 715–725, 2004.

[27] G. Huet, "The Zipper," *J. Functional Programming*, vol. 7, no. 5, pp. 549–554, September 1997.

[28] M. D. Adams, "Scrap your zippers: A generic zipper for heterogeneous types," in *WGP '10: Proceedings of the 2010 ACM SIGPLAN workshop on Generic programming*. New York, NY, USA: ACM, 2010, pp. 13–24.

[29] T. Schrijvers and B. C. D. S. Oliveira, "The Monad Zipper," 2010.

[30] R. Hinze and R. Paterson, "Finger trees: a simple general-purpose data structure," *J. Functional Programming*, vol. 16, no. 2, pp. 197–217, 2006.

[31] M. Erwig and S. Kollmansberger, "Probabilistic Functional Programming in Haskell," *J. Functional Programming*, vol. 16, no. 1, pp. 21–34, 2006.

[32] M. Izbicki, "HLearn: A Machine Learning Library for Haskell," in *Proceedings of The Fourteenth Symposium on Trends in Functional Programming*, Brigham Young University, Utah, May 14-16, 2013.

[33] ——, "HLearn's code is shorter and clearer than Weka's," https://izbicki.me/blog/hlearns-code-is-shorter-and-clearer-than-wekas.

[34] N. Tishby, O. C. Pereira, and W. Bialek, "The information bottleneck method," in *University of Illinois*, 1999, pp. 368–377.

[35] R. Bird, "A program to solve Sudoku," *J. Functional Programming*, vol. 16, no. 6, pp. 671–679, 2006.

[36] R. S. Bird, "On tiling a chessboard," *J. Functional Programming*, vol. 14, no. 6, pp. 613–622, 2004.