The Monad.Reader/Issue3/SoE Review

From HaskellWiki

< The Monad.Reader | Issue3

Isaac Jones (http://www.syntaxpolice.org) ' review of **The Haskell School of Expression** was originally published on Slashdot (http://books.slashdot.org/books/04/03/12/221232.shtml?tid=126&tid=156&tid=188&tid=192) , and is reprinted here with permission from the author.

Review: The Haskell School of Expression

Andrew Cooke reviewed for Slashdot (http://books.slashdot.org/article.pl?sid=04/02/19/2257203) *Purely Functional Data Structures*, which is on my book shelf next to The Haskell School of Expression: Learning Functional Programming through Multimedia (http://www.haskell.org/soe/) by Paul Hudak (http://www.cs.yale.edu/homes/hudak-paul.html) from the Yale Haskell Group (http://haskell.cs.yale.edu/yale/) . In his review, Cooke presented an overview of some available functional programming languages, such as OCaml (http://caml.inria.fr/) , SML (http://www.standardml.org/) , and of course Haskell (http://www.haskell.org/) . Havoc Pennington once called Haskell (http://ometer.com/books.html) "the least-broken programming language available today." Haskell is a purely functional, lazy, statically typed programming language. You can read more about Haskell itself here

As the title implies, **The Haskell School of Expression** introduces functional programming through the Haskell programming language and through the use of graphics and music. It serves as an effective introduction to both the language and the concepts behind functional programming. This text was published in 2000, but since Haskell '98[1] (http://www.haskell.org/onlinereport/) is the current standard, it is still a very relevant book.

Haskell's standardization process gives us a window into two different facets of the community: Haskell is designed to be both a stable, standardized language (called Haskell 98), and a platform for experimentation in cutting-edge programming language research. So though we have a standard from 1998, the implementations (both compilers and interpreters) are continually evolving to implement new, experimental features which may or may not make it into the next standard.

For instance, the Glasgow Haskell Compiler has implemented a meta-programming environment called Template Haskell. Haskell is also easy to extend in directions that don't change the language itself, through the use of "[[Embedded Domain Specific Languages]]" (EDSLs (http://citeseer.ist.psu.edu/hudak96building.html)) such as WASH for web authoring, Parsec for parsing and Dance (http://haskell.cs.yale.edu/yale/papers

/dance-30-tr/index.html) (more of Paul Hudak's work) for controlling humanoid robots.

Before we get too far, I should offer a disclaimer. The Haskell community is rather small, and if you scour the net, you may find conversations between myself and Paul Hudak or folks in his research group, since I use some of their software. That said, I don't work directly with Hudak or his research group.

In fact, the small size of the Haskell community is a useful feature. It is very easy to get involved, and folks are always willing to help newbies learn, since we love sharing what we know. You may even find that if you post a question about an exercise in *The Haskell School of Expression*, you'll get a reply from the author himself.

I consider this book to be written in a "tutorial" style. It walks the reader through the building of applications, but doesn't skimp on the concepts (indeed, the chapters are meant to alternate between "concepts" and "applications.") In some ways, the code examples make it a little difficult to jump around, since you are expected to build upon previous code. The web site provides code, however, so you can always grab that and use it to fill in the missing pieces.

For readers who wish to use this book as a tutorial, and to implement all of the examples (which is highly recommended), I suggest that you grab (http://cvs.haskell.org /Hugs/pages/downloading-Nov2002.htm) the Hugs interpreter and read the User's Guide (http://cvs.haskell.org/Hugs/pages/users_guide/index.html) while you're reading the first few chapters of *The Haskell School of Expression*. Hugs is very portable, free, and easy to use. It also has an interface with Emacs. Unfortunately, some of the example code has bit-rotted a little, and certain things don't work out-of-the-box for X11-based systems. The bit-rot can be solved by using the "November 2002" version of Hugs. This is all explained (http://www.haskell.org/soe/source.htm) on *School of Expression*'s web page (http://www.haskell.org/soe/) .

The Haskell School of Expression should be very effective for programmers who have experience in more traditional languages, and programmers with a Lisp background can probably move quickly through some of the early material. If you've never learned a functional language, I definitely recommend Haskell: Since Haskell is *purely* functional (unlike Lisp), it will more or less prevent you from "cheating" by reverting to a non-functional style. In fact, if you've never really looked at functional programming languages, it may surprise you to learn that Haskell has no looping constructs or destructive assignment (no `x = x + 1`). All of the tasks that you would accomplish through the use of loops is accomplished instead through recursion, or through higher-level abstractions upon recursion.

Since I was already comfortable with recursion when I started this book, it is hard for me to gauge how a reader who has never encountered recursion would find this book's explanation of the concept. *The Haskell School of Expression* introduces recursion early on, in section 1.4. It is used in examples throughout the book, and if you follow along with these examples, you will most certainly be using it a lot. The introduction seems natural enough to me, but I note that Hudak does not give the reader any extra insight or tricks to help them along. Not to worry, though; recursion is very natural in Haskell and the reader may not even notice that they are doing something a little tricky.

The use of multimedia was a lot of fun for me, and should quickly dispel the myth that IO

is difficult in Haskell. For instance, Hudak has the reader drawing fractals by page 44, and throughout the book, the reader will be drawing shapes, playing music, and controlling animated robots.

Any book on Haskell must be appraised for its explanation of monads in general and IO specifically. Monads are a purely functional way to elegantly carry state across several computations (rather than passing state explicitly as a parameter to each function). They are a common stumbling block in learning Haskell, though in my opinion, their difficulty is over-hyped.

Since input and output cause side-effects, they are not purely functional, and don't fit nicely into a function-call and recursion structure. Haskell has therefore evolved a way to deal safely and logically with IO through the use of monads, which encapsulate mutable state. In order to perform IO in Haskell, one must use monads, but not necessarily understand them.

Some people find monads confusing; I've even heard a joke that you need a Ph.D. in computer science in order to perform IO in Haskell. This is clearly not true, and this book takes an approach which I whole-heartedly agree with. It gets the reader using monads and IO in chapter 3 without explaining them deeply until chapters 16 (IO) and 18 (monads). By the time you get there, if you have heard that monads are confusing, you might be inclined to say "how is this different from what we've been doing all along?" Over all, I was pleased with the explanation of monads, especially state monads[2] (http://web.archive.org/web/20041213163105/http://www.haskell.org/hawiki/StateMonad) in chapter 18, but I felt that the reader is not given enough exercises where they implement their own monads.

If you're worried that drawing shapes and playing music will not appeal to your mathematic side, you will be pleased by the focus on algebraic reasoning for shapes (section 8.3) and music (section 21.2), and a chapter on proof by induction (chapter 11).

After reading this book you will be prepared to take either of the two paths that Haskell is designed for: You can start writing useful and elegant tools, or you can dig into the fascinating programming language research going on. You will be prepared to approach *arrows*, a newer addition to Haskell which, like monads, have a deep relationship to category theory. Arrows are used extensively in some of the Yale Haskell group's recent work (http://www.haskell.org/yale/publications.html) . You will see a lot of shared concepts between the animation in *The Haskell School of Expression* and Yale's "Functional Reactive Programming" framework, Yampa. If you like little languages (http://c2.com/cgi/wiki?LittleLanguage) , you'll appreciate how useful Haskell is for embedded domain specific languages. It may be even more useful (http://www.cse.unsw.edu.au/~sseefried/files/sseefried03th-pan.pdf) now that Template Haskell is in the works. Andrew Cooke described (http://books.slashdot.org /article.pl?sid=04/02/19/2257203) *Purely Functional Data Structures* as a great *second* book on functional programming. In my opinion, *The Haskell School of Expression* is the great first book you're looking for.

Category: Article
