A History of Haskell: being lazy with class
By Hudak, Hughes, Jones, Wadler

Louis Hyde, Peter Wood
Section 1

Creating Haskell
Before Haskell

- Late 1950s: Lisp, the first functional language
- 1960s: importance of lambda calculus identified
- Late 1970s: Scheme, like Lisp but with more lambda calculus; lexical scoping and other features
- Late 1970s and early 1980s: lazy evaluation comes into prominence
Creating Haskell

Why create Haskell?

By the mid-1980s, functional programming had become popular, but everyone used their own languages, many of which were proprietary.

Figure: xkcd courtesy of Randall Munroe
Creating Haskell

How Haskell was created

- FPCA conference
- Simon Petyon Jones met with Paul Hudak
- First meeting of the FPLang Committee
Goals of the FPLang committee

1. Usable for real-world applications
2. Published formal syntax and semantics
3. Free
4. Usable for language research
5. Consensus-based
6. Reduce unnecessary diversity
Choosing a name

- All proposed names were written on a board
- Each person removed one name they didn’t want
- They were left with “Curry”, after Haskell Curry
- Potential for stupid jokes
- Eventually settled on “Haskell”
Section 2

Evolution of Haskell
Publication of the first Haskell Report

- April 1, 1990
- Standard Prelude
- Further versions published over next seven years
Later Haskell Reports

- Haskell 1.1, August 1991: adding `let`
- Haskell 1.2, March 1992: extremely minor changes
- Haskell 1.3, May 1996
  - Standard libraries
  - Monadic IO
  - Higher-kind polymorphism, important for monads
  - Improvements to algebraic data types
- Haskell 1.4, April 1997: more minor changes
Haskell 98

- February 1999
- Intended to be a standard dialect but not the only
- Definition of the language and a few small libraries
- Marked the dissolution of the Committee
- Revised version published as a book in December 2002
Input and output

- A pure language can’t have side effects, but input and output are side effects
- Streams
  - A program is a function from a list of responses to a list of requests
  - Requires lot of confusing-looking pattern-matching
- Continuations
  - Failure continuations and success continuations
  - The appropriate one is applied to the result of the IO
  - Usually defined in terms of streams
Monadic IO

- The IO monad represents a computation that, when performed, does some IO and returns a value.
- For example, a function that reads a file could be `String -> IO String`.
- The monadic `return` performs no IO and simply returns its value.
- The monadic `bind` composes two IO actions simultaneously.
- do-notation, added in Haskell 1.3.
Once the IO monad emerged, many people started using it to call C code.

“Blessed Addendum” to Haskell 98 Report published in 2003, specifying a standard for how to do this.
Section 3

Significant Features and Applications of Haskell
Monads: More than I/O

- Lists, Maybes, Eithers
- Functions
- ST
Monads: More than I/O

- Lists, Maybes, Eithers
- Functions
  - `return :: b -> (a -> b)`
  - `(>>=) :: (a -> b) -> (b -> (a -> c)) -> (a -> c)`
- ST
Monads: More than I/O

- Lists, Maybes, Eithers
- Functions
  - return :: $b \rightarrow (a \rightarrow b)$
  - $(\gg=) :: (a \rightarrow b) \rightarrow (b \rightarrow (a \rightarrow c)) \rightarrow (a \rightarrow c)$
- ST
  - runST :: (forall s. ST s a) -> a
Monads: a Plethora of Applications

- Randomness
- Concurrency
- Exceptions
- Parsers
- etc.
Arrows: a Brief Mention

class Arrow a where
    arr :: (b -> c) -> a b c
    (>>>) :: a b c -> a c d -> a b d
    first :: a b c -> a (b,d) (c,d)
Functional Reactive Programming

- Behaviors
  - A time varying value
- Event Stream
  - Stream of discrete events
  - Time value pair
  - Potentially infinite
FRP Applications

- Animation
- Computer Music
- GUI
Brief Mentions

- Typeclasses
- QuickCheck
Section 4

Influence of Haskell
Functional Programming Languages

Curry: a combined functional-logical programming language
Cayenne: a dependently typed language
Isabelle: a theorem prover with typeclasses
Elm: Functional web frontend language
Imperative Programming Languages

Python and Javascript: list comprehensions

Java: Generic types are based on Hindley-Milner

C# and Visual Basic: LINQ (Language INtegrated Query) is based on monad comprehensions
Academia and Education

- Designed to be suitable for teaching
- Lots of introductory textbooks
  - As of the paper’s writing few people learning it as their first language
- Widely used in classes on functional programming, advanced programming, and programming languages
- Academia is Haskell’s main base (as of 2007)
Industry

- Prized for its reliability and extensibility
- Problems with efficiency and finding programmers

Galois Connections: high confidence software
BlueSpec: hardware design
Aetion: Artificial intelligence software for decision support
Linspire: Linux distribution
Section 5

Commentary on the Paper
Positive Thoughts

- Easy to read
- Explains reasoning behind design choices well
- Lots of interesting references
Criticisms

- Descriptions of companies use of Haskell lack substantive details
- Suffers some superfluous topics in an already lengthy paper
Section 6

Questions?