An Invitation to Haskell

Emily Pillmore

March 19, 2023

Table of Contents

My name is Emily Pillmore.

I am a programmer, and a math enthusiast.

- Author/Maintainer of more than 30 packages, some bigger than others
- Served on the Haskell Core Libraries and .Org committees
- Twitter (@yandereidiot)
- Meetups in NYC: NY Homotopy Type Theory, NY Category Theory, and the NY Haskell User Group.
- All of my slides, general scribbles, research, and meetup content are hosted at cohomolo.gy.

If you ever want to talk math or programming, I'm around.

I helped start the Haskell Foundation and served on the executive leadership team as a duo (CTO) with Andrew Boardman (ED).

I now work at a company called **Kadena**, as the lead of the language, its ecosystem, and its execution layers.

Table of Contents

So what is Functional Programming?

So what is Functional Programming?

- A collection of features? (lambdas, first class HOF's, static type system...)

So what is Functional Programming?

- A collection of features? (lambdas, first class HOF's, static type system...)
- A programming style? (emphasis on recursion, "math", small static combinators, shunting as many errors to the compiler as possible)

So what is Functional Programming?

- A collection of features? (lambdas, first class HOF's, static type system...)
- A programming style? (emphasis on recursion, "math", small static combinators, shunting as many errors to the compiler as possible)
- A cult?

Table of Contents

In 1977, John Backus wrote everything we needed to know about FP.

Compositionality! Equational Reasoning! Sound foundational principles!

Table of Contents

Haskell builds a equational foundation on purity.

This means that functions may not have *side effects*. In conjunction with not allows side effects anywhere, this allows expressions to be completely deterministic, and therefore *referentially transparent*.

Table of Contents

Famously, Haskell has a type system.

Famously, Haskell has a type system.

- It has functions (read: function definitions, lambdas)

Famously, Haskell has a type system.

- It has functions (read: function definitions, lambdas)
- It has builtins (integers, IEEE floating points, machine words, characters etc.)

Famously, Haskell has a type system.

- It has functions (read: function definitions, lambdas)
- It has builtins (integers, IEEE floating points, machine words, characters etc.)
- It has generics

Haskell has a global notion of parametricity everywhere you want it which may be reasoned about equationally, and therefore free theorems you can reason about.

It has a form of ad-hoc polymorphism for generics called "Typeclasses".

For more, see:

- Wadler - Theorems for Free
- My talk - Type Arithmetic

Table of Contents

Since purity undermines destructive updates, our Haskell data structures are immutable. This creates some issues.

Since purity undermines destructive updates, our Haskell data structures are immutable. This creates some issues.

- The amortized complexity theory needed to talk about the best/average/worst case of operations goes out the window (your average case becomes your worst case for many operations).

Since purity undermines destructive updates, our Haskell data structures are immutable. This creates some issues.

- The amortized analysis (cheap small steps paying off a more expensive larger step) needed to talk about the best/average/worst case of operations goes out the window (your amortized cost becomes your worst case for many operations).
- Laziness (a limited form of mutation) turns out to be enough to recover amortized analysis

Since purity undermines destructive updates, our Haskell data structures are immutable. This creates some issues.

- The amortized analysis (cheap small steps paying off a more expensive larger step) needed to talk about the best/average/worst case of operations goes out the window (your amortized cost becomes your worst case for many operations).
- Laziness (a limited form of mutation) turns out to be enough to recover amortized analysis.
- This requires a different take on analysis (thunk counting techniques etc.) which causes you to think in a whole new paradigm.

Since purity undermines destructive updates, our Haskell data structures are immutable. This creates some issues.

- The amortized analysis (cheap small steps paying off a more expensive larger step) needed to talk about the best/average/worst case of operations goes out the window (your amortized cost becomes your worst case for many operations).
- Laziness (a limited form of mutation) turns out to be enough to recover amortized analysis.
- This requires a different take on analysis (thunk counting techniques etc.) which causes some tension.

Immutability + Laziness, though, is a super power. Friedman-Wise posed an important question back in 1976.

Inherently easy to spread about on multiple cores. With commutative, associative, and unital (see: commutative monoidal) functions, map-reduce is possible.

It also makes scheduling parallelism and concurrency a simpler.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents