Sandy Maguire

343.262.3363 • sandy@sandymaguire.me • https://reasonablypolymorphic.com

Research Interests

Functional programming, compiler design, free monads

Education

2010-2015 **BSE, Software Engineering, Honours, Co-operative Program**
University of Waterloo, Waterloo, ON

- 77.16 Cumulative GPA
- Graduated in Excellent Standing

Classes and Grades

2015 Compiler Construction (92) • Design Project 2 (89) • Introduction to Artificial Intelligence (84) • Software Testing and Quality Assurance (76) • Introduction to Microeconomics (74) • Cybernetics and Society (66)

2014 Design Project 1 (92) • Quantum Physics 1 (83) • Computer Networks (78) • Software Design and Architectures (73) • Cooperative and Adaptive Algorithms (72)

2013 Design Project Planning (96) • Operating Systems (87) • The Use of English (86) • Software Requirements Specification and Analysis (83) • Concurrent and Parallel Programming (75) • User Interfaces (74) • Introduction to Database Management (71) • Introduction to the Theory of Computing (70) • Numerical Computation (68) • Introduction to Feedback Control (65) • Algorithms 1 (58)

2012 Data Structures and Data Management (80) • Software Engineering Principles (75) • Introduction to Combinatorics (74) • Waves, Electricity and Magnetism (72) • Advanced Mathematics for Software Engineers (67) • Engineering Economics (67)

2011 Foundations of Sequantial Programs (85) • Chemistry for Engineers (81) • Digital Circuits and Systems (80) • Functional Programming and Data Abstraction (79) • Calculus 2 for Engineering (76) • Digital Computers (75) • Statistics for Software Engineering (75) • Physics of Electrical Engineering 2 (73) • Introduction to Philosophy (70) • Algebra for Honours Mathematics (62) • Logic and Computation (62)

2010 Linear Algebra for Engineering (96) • Programming Principles (93) • Physics of Electrical Engineering 1 (86) • Introduction to Methods of Software Engineering (85) • Calculus 1 for Engineers (78) • Linear Circuits (73)

Professional Experience

2016-2018 **Senior Haskell Software Engineer**
Takt

- Led a team of four to reimplement a core subcomponent of the product --- increased the cadence of new feature development from months to days.
- Directed a team of three to implement a high-throughput, low-latency brokered streaming library. Resulting library became the company's core inter-service communication protocol.

2015-2016 **Engineer III (Identity and Access Management)**
Google

- Led the architectural design effort of a user-defined permission model for the cloud --- the team's only project for the subsequent quarter.
- Took ownership over an unmaintained, service-critical internal compiler; improved compile times by 96% and test coverage by 65%.

2014 **Engineering Intern (Ads Ranking)**
Facebook

- Analyzed the advertising platform's spending behaviors, and subsequently implemented algorithmic changes --- resulting in a 0.5% revenue increase.
- Parallelized the back-end graph ranker, resulting in site-wide response time gains of 0.4%.

2013 **Engineering Intern (FIFA Mobile)**
Electronic Arts Canada
- Single-handedly cleaned up 60% of the project's technical debt alongside assigned tasks.

2012 **Gameplay Programmer Intern**
Digital Extremes
- Architected and implemented the game's entire voice-over-IP stack.
- Rewrote a significant portion of the AI and PhysX engines to support arbitrary 3-space rotation.
- Designed and implemented a type-system for the game's proprietary scripting language runtime.

2011 **.NET Consultant**
Systemgroup Consulting Inc,
- Found and closed over ten highly-exploitable vulnerabilities in a multi-billion-dollar company's website.

Publications

Books

2018 **Thinking with Types: Type-Level Programming in Haskell** [link]
Sandy Maguire, self-published
- Favourably reviewed; described as being "well written and very approacable", "absolutely brilliant," and "a great read."
- Sold over 1,000 copies --- a significant portion of the estimated 10,000 members in the English-speaking Haskell community.

2016 **How These Things Work** [link]
Sandy Maguire, self-published

Blog

2015- **Reasonably Polymorphic** [link]
Sandy Maguire
- Professional and research blog.

Presentations

2019 **Chasing the Performance of Free Monads**
Lambda Montreal • Montreal, QC

2018 **Existentialization: What Is It Good For?** [slides]
Denver FP • Denver, CO

2017 **Some1 Like You: Dependent Pairs in Haskell** [video]
LambdaConf • Boulder, CO

2017 **Don't Eff It Up: Freer Monads in Action** [video]
BayHac • San Francisco, CA

Open Source Contributions

Maintainer

2019- **isovector/polysemy** [link]
Higher-order, no-boilerplate, zero-cost free monad library.

2019- **isovector/suavemente** [link]
An applicative functor capable of talking directly to HTML inputs.

2018- **isovector/do-notation** [link]
Generalized do-notation for indexed monads.

2018- **isovector/latex-live-snippets** [link]
Keep code snippets up to date in LaTeX documents.

2018- **isovector/constraints-emerge** [link]
Runtime reification of Haskell's instance resolution machinery.

2018- **isovector/ecstasy** [link]
A boilerplate-free entity component system library.

2017- **isovector/th-dict-discovery** [link]
Compile-time reification of every typeclass instance in scope.

2017 **TaktInc/freer** [link]
Corporate fork of a freer monad library.

Contributor

2019 **Lysxia/first-class-families** [link]
A library for higher-order type-level programming in Haskell.

2017 **kim/opentracing** [link]
Haskell implementation of the OpenTracing instrumentation API.

2017 **google/proto-lens** [link]
API for protocol buffers using modern Haskell language and library patterns.

2017 **IxpertaSolutions/freer-effects** [link]
A free monad library in the style of "Freer Monads, More Extensible Effects."

2017 **k0001/exinst** [link]
An implementation of sigma types in Haskell.

Committer

2018 **nomeata/inspection-testing** [link]
Automated testing of Haskell's Core representations.

2018 **AshleyMoni/QuadTree** [link]
QuadTree library for Haskell, with lens support.

2018 **conal/concat** [link]
A GHC plugin to compile Haskell to constrained categories.

2018 **diagrams/diagrams-contrib** [link]
User contributed extensions to diagrams.

Honours, Awards and Scholarships

2015 **SoftEng Capstone Design Symposium (2nd place)**
Yelp

2010 **President's Scholarship**
University of Waterloo

Research Statement

My program in university was co-operative; after each school term I was expected to find and work a job for four months. Four months is just about as long as it takes most people to start feeling comfortable in a new codebase, and as a result, I needed to learn how to learn fast.

It became clear to me that a great deal of human capital was being wasted simply trying to reverse-engineer these codebases. The industry prioritizes writing "readable" code, characterized by being verbose and "doing nothing clever". While each step along the way is indeed comprehensible, too often this desideratum manages to lose the original *intention* behind the code.

Some friends and I decided to solve this. As our graduation capstone project, we built an analytics tool to determine *who knows what* in a codebase. By correlating commit information with between what code was physically on-screen, we found that we could outperform other contributors on both debugging and refactoring. Our results won us second-place in a project competition hosted by Yelp.

At my last company our code was an impressively tangled, untestable ball of spaghetti. A stupid bug made it into production, and it cost us a few million dollars. I started looking for a preventative solution, and came across free monads.

Free monads are a technique that allow for a separation of syntax from semantics. They allow you to quickly define a domain-specific language (DSL) appropriate for the problem at hand, and later provide interpretations for what that DSL means. These interpretations need not be unique, and can be composed at will. Different interpretations can provide dramatically different artifacts from the same code.

Free monads, as they exist today, have a problem in the face of polymorphism. While nothing in principle prevents their DSLs from being polymorphic, the ergonomics around doing so are frustrating. Usages that are perfectly clear to humans are nevertheless ambiguous to the compiler. I want to develop a compiler plugin which is capable of resolving these ambiguities, by extending the unification algorithm to play nicely with polymorphic DSLs.

In 2018, I realized I could take the approach of carefully evaluating only as much of a program as was necessary. In doing so, I could plumb through free monads, and perform static analysis on many things previously considered impossible.

Historically, free monads have had problems with performance. Wu and Schrijvers have shown that efficient free monads are possible, though their technique requires a significant amount of boilerplate. I came up with a solution that could give free implementations of the boilerplate, at the expense of making the compiler's optimiser work harder. My work allows for free monads to be a true zero-cost abstraction, without the need for any tedious boilerplate.

The zero-cost performance gains found here are impressive, though unreliable. The process requires the compiler to have total knowledge of the whole program, and requires hand-written annotations in order of having any chance of successfully optimizing the problem away. My results are a promising proof-of-concept; a robust solution will likely require finding an object-code format amenable to describing the DSLs as first-class concepts.

Last year I wrote a book on type-level programming in Haskell. I wrote the copy in LaTeX, which came back to haunt me when I decided to publish an ebook alongside the print version. While LaTeX provides an ergonomic writing environment, and makes it easy to spin up new abstractions, it doesn't give any tools to allow for different interpretations. For example, an ebook simply doesn't have the screen real-estate for a sidebar. This is an area in which I'd love to use free monads, but doing so will require better ergonomics for using them in non-programming domains.

HTML is a successful document format, which makes the distinction between block and inline elements. Block elements may contain inline elements and other blocks, but inline ones may contain only other inline elements. Despite their advances over first-order effects, scoped effects are still not yet capable of modeling HTML elements. In my view, this restriction is rather embarrassing. It's caused by a lacking reification of the DSL capabilities *inside* a scoped effect. I don't think this is a fundamental limitation of the approach, merely one that could use some thinking.

Finally, free monads strike me as having many desirable properties in knowledge work. The recent popularity of Jupyter notebooks suggests to me that not only do people want to give answers, but also show the means by which they were computed. But these are just two interpretations of the same thing! I dream of a day when all of our ad-hoc decision making is reified into data, to be reinterpreted at will. The implications for policy-making and transparent governance are especially staggering, and I'd like to do everything in my power to help make that future a reality.