Larry **Diehl**

Software Engineer · Researcher

2614 SW Water Ave – Portland, OR – 97201

407-718-7665 | larrytheliquid@gmail.com | www.larrytheliquid.com | larrytheliquid

Job Application for Software Engineer/Researcher *September 2, 2016*

Galois Inc.
421 SW 6th Avenue, Suite 300
Portland, Oregon 97204

Software Engineering Background

My major professional software development experience has involved the Ruby programming language. The Ruby community emphasizes good software engineering practices, like code quality, unit testing, integration testing, Test-Driven Development, and Agile development.

The most most prestigious Ruby company I worked for was Engine Yard, where I worked on a cloud hosting platform built on top of Amazon AWS. A monolithic application could not support such a large product, so instead it was made of many isolated microservices to tame the complexity. The main product involved the consumption of these microservices, and careful orchestration was required to keep the entire apparatus running smoothly. There I realized the software engineering limits of a language like Ruby for large software endeavours. While testing a single application or microservice is easy, the combinatorial explosion of test cases resulting from the interaction of microservices makes exhaustive testing impossible, and reasonable coverage unreasonably difficult.

Formal Methods Background

My experience working for Engine Yard made me look for other ways to achieve better software engineering, ultimately settling on formal methods. While unit tests can never exhaustively show correctness over an infinite domain, a proof of a universally quantified proposition can. This idea led me to write a proof of concept web framework (Lemmachine) in the dependently typed Agda language. The idea was that proofs of correctness for individual microservices could be reused by correctness proofs of the consumer application because proofs are inherently compositional.

I decided to a do a Ph.D. in Computer Science, hoping to make such work more practical. I specialized in researching dependently typed languages. While writing Lemmachine I encountered code reuse issues with modern dependently typed languages, so my research focused on generic programming to overcome such issues. Close to graduating, I am now very comfortable with dependently typed languages, both programming (and proving) with them and implementing (using Haskell) custom versions of them to cater to specific problems.

Why Galois?

At the current point of my career I have experienced two extremes. The first extreme was software engineering in Ruby for ordinary business applications. The second extreme was academic formal methods research using Haskell, Agda, and dependent types. I'm now ready to find the middle ground between the two, finding the right compromises between engineering effort and desired correctness criteria for specific projects. I believe Galois is a fantastic fit for doing this.

Sincerely,

Larry Diehl