Larry **Diehl**

Software Engineer · Researcher

2614 SW Water Ave – Portland, OR – 97201

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

Company Recruitment Team

September 2, 2016

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

Job Application for Software Engineer/Researcher

Software Engineering Background

I got into Ruby programming early around the time Ruby on Rails started getting popular. I was impressed by how small and elegant Ruby programs were compared to their C and Java counterparts. Although a dynamic language like Ruby makes more runtime errors possible, I consider myself fortunate to have spent my nascent programming years with the language. This is because the Ruby community emphasizes good software engineering practices, like code quality, unit and and integration testing, Test-Driven Development, and Agile development. In other words, perhaps because Ruby is such a brittle language in the first place, the community highly values software correctness.

After graduating with my Bachelor's, I had already worked for two Ruby companies in Florida and moved to California to work for Engine Yard. Engine Yard was (and perhaps still is) one of the most prestigious Ruby companies. There I worked on the main product, a cloud hosting platform for 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. This is around the point where 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 (in a setting like I experienced at Engine Yard) because proofs are inherently compositional.

To learn more with the hope making such work more practical, I decided to a do a Ph.D. in Computer Science. 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 with them and implementing 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. 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

Attached: Résumé