Larry **Diehl**

Formal Methods Researcher & Engineer

2614 SW Water Ave – Portland, OR – 97201

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

Job Application for Post-Doc/Researcher Position *August 31, 2018*

Trustworthy Software Technologies - Tallinn University of Technology

Thesis Work

I earned a Ph.D. in Computer Science from Portland State University, with a research focus on datatype-generic programming (within a closed predicative dependent type theory), culminating in my thesis, "Fully Generic Programming over Closed Universes of Inductive-Recursive Types". With the advent of formal proof assistants (or dependently typed languages), constructive type theory can increasingly be used as a programming language for writing certified software, rather than just pen-and-paper mathematics. However, existing languages (e.g. Agda, Coq, Idris, and Lean) are still far from being easy or practical enough to be used by non-specialist programmers.

My Ph.D. work focused on extending dependently typed languages to support writing generic functions (and generic correctness proofs about them) once-and-for-all, that can be used with any datatype the user could possibly define. In industry, such generic functions have been popular in dynamic languages with runtime metaprogramming facilities (e.g. Ruby), but without the ability to reason about their correctness. One such example is generic marshalling and unmarshalling functions (e.g. toJSON and fromJSON), and generic proofs about the inverse properties about such functions (e.g. fromJSON . toJSON = id). Such functions can be defined for any datatype the user may declare, which introduces significant complexity in dependently typed languages due to the presence of infinitary, indexed, and inductive-recursive datatypes, which lie beyond the popular fragment of algebraic datatypes that non-dependent functional languages support.

Besides pragmatic motivations of making dependently typed languages more usable, my interests also include the theory behind such languages and other type theories. For example, my thesis (and papers leading up to it) introduced a model of closed type theory whose universe not only reflected codes for primitive types (pi, sigma, fix, etc.), but also mutually closed codes for signature functors, whose presence is crucial for the generic programming I achieved. My theoretical interests also includes model theory and proof theory, and their relationship to programming language semantics. For example, I proved termination of hereditary substitution for canonical System T via logical relations, whose inductive naturals cause the standard proof-theoretic termination argument to fail.

Postdoc Work

During my postdoc at the University of Iowa, my focus shifted from predicative Church-style type theories to impredicative Curry-style type theories (e.g. Cedille). This required me to reset many mental assumptions about what types of datatype encodings and theorems are possible in type theory. In particular, the work on such theories has led to beautiful impredicative semantic accounts of datatypes as initial algebras. In predicative type theories, initial algebras are either phrased syntactically (via a datatype of descriptions for signature functors, as in my thesis), or semantically by W-types, but this requires moving from intentional type theory to extensional type theory to preserve adequacy. The group at Iowa was already in the process of discovering such semantic accounts before I came, but I have been involved in extending them to larger classes of datatypes, including inductive-inductive and inductive-recursive ones. Additionally, I have worked on generic programming within this semantic setting, focusing on zero-runtime-cost reuse of programs and proofs between related datatypes, achieving software modularity without performance penalties.

Software Engineering Work

While the theory behind dependently typed languages is well understand by researchers, the practical implementation of non-toy implementations is a bit of a dark art. This is especially true when it comes to features such as dependent pattern matching, and inferring implicit arguments via dynamic higher-order unification. I am fortunate enough to have worked on the implementation of dependently typed languages during my Ph.D. (Spire and Ditto), and during my postdoc (Cedille).

Before entering my Ph.D. program, I also acrued experience in industry as a software engineer, working for startups in Florida and San Francisco. At this time I primarily worked with dynamic languages (Ruby) on problems in the medical space (Bear Den Designs), online advertising space (IZEA), and cloud space (Engine Yard). The problem areas I primarily worked on involved web applications, web services, and distributed programming.

Sincerely,

Larry Diehl