Daniel Fedorin

Portland, OR **(000) 000 0000** **danila.fedorin@gmail.com**

Education

Master of Science, Oregon State University, Computer Science - Completed June 2022 • **4.0 GPA**
Thesis: *Guided Translation of Algorithmic Notation into Functional Programs*

Bachelor of Science, Oregon State University, Major: Computer Science | Minor: Mathematics - Completed June 2020 • **4.0 GPA**

Projects

agda-spa — Framework for formally-verified lattice-based program analysis in **Agda**, explained in-depth in a series of posts.
bloglang — Compiler for a functional, lazily evaluated language using **C++** and **LLVM**, explained in-depth on personal blog.
maypop — Dependently typed functional programming language capable of formal proofs, written in **Haskell**.
matrix-highlight — **TypeScript**-based browser extension for collaborative web annotation based on the Matrix protocol.

Publications

Divya Bajaj, Martin Erwig, **Daniel Fedorin**: *A Visual Notation for Succinct Program Traces (journal paper)*, COLA 2023
Divya Bajaj, Martin Erwig, **Daniel Fedorin**, Kai Gay: *Adaptable Traces for Program Explanations*, APLAS 2021
Divya Bajaj, Martin Erwig, **Daniel Fedorin**, Kai Gay: *A Visual Notation for Succinct Program Traces*, VL/HCC 2021
Jácome Cunha, Mihai Dan, Martin Erwig, **Daniel Fedorin**, Alex Grejuc: *Explaining spreadsheets with spreadsheets (short paper).* GPCE 2018: 161-167

Work Experience

Senior Programming Language Engineer, Chapel
Hewlett Packard Enterprise | Summer 2022 - Present

- Implemented compile-time reflection, parallel iterators, and trait-like interfaces in Chapel's **C++**-based compiler.
- Led development of **Python** bindings for compiler, accelerating development of a linter and a language server by over 10x.
- Optimized scope resolution and type checking compiler passes, improving performance by 30% and 60% respectively.
- Designed a type-safe error reporting API, improving developer experience and enabling 100+ custom error messages.
- Supported community growth by designing, launching, and authoring articles for the Chapel language technical blog.
- Laid groundwork for compatibility with leading-edge supercomputers by implementing initial **AMD GPU** programming support using **Clang** and **ROCm** tooling.

Research and Teaching Assistant, Programming Language Theory
Oregon State University, Corvallis, OR | Spring 2018 - Summer 2022

- Formalized denotational and operational semantics of new explanation-oriented programming languages.
- Developed tooling in **Haskell** to interpret, verify, generate, and debug programming languages.
- Contributed to research papers published to the GPCE and VL/HCC.
- Proctored quizzes and exams for over 200 students.
- Aided students in implementing a final project in the form of a custom programming language.
- Suggested and organized independent review sessions attended by over 70 students, with 50% attendance growth between sessions.

Front-End Intern, Hydrogen
Element.io | June 2021 - September 2021

- Spearheaded migration of codebase to **TypeScript**, improving documentation and discovering hidden bugs.
- Leveraged advanced type system features to precisely specify nontrivial program properties.
- Developed a mocking system to help specify and test corner cases in a distributed communication system.
- Independently implemented user-facing features including offline-first replies and sanitized HTML rendering.
- Engaged in **open-source development**, interacting with community to respond to bug reports and feature requests.

Additional Experience

Technical Writer
Independent | Spring 2015 - Present

- Designed and published website currently live at danilafe.com, peaking at 27k daily unique visitors.
- Authored blog posts on topics spanning data structures, web development, programming languages, and compilers.
- Formalized and described solutions to select Advent of Code problems using the **Coq proof assistant**.
- Created 14-part series on compiler development, walking readers through lexing, parsing, compilation using LLVM, garbage collection, and polymorphic type checking.