Scott Olson

scott@solson.me https://solson.me https://github.com/solson

Experience

Google Oct 2016–present
Software Engineer Seattle, WA

- Worked on the Google Compute Engine Live Migration team, responsible for moving running customer VMs to new host machines without rebooting the VMs, and with minimal interruption to the customer
- Refactored an important internal interface which affected much of our C++11 codebase, reducing repetition and boosting maintainability
- Designed a process for recording and analyzing performance data from migrations in production with minimal performance impact
- Monitored and debugged issues with live migration in production

IBM Sept 2014–Aug 2015
Software Engineering Intern Markham, ON

- Began the porting effort of Clang to z/OS for IBM's XL C/C++ compiler
  - Involved low-level platform support and an understanding of UNIX and POSIX
- Diagnosed bugs in the compiler we were using to compile the Clang sources
  - Involved complex details of C++ features and an understanding of standards compliance
- Worked with C, C++, Makefiles, shell scripts, Python, and autoconf

Google May–Aug 2014
Software Engineering Intern Waterloo, ON

- Developed internal tools for monitoring ads infrastructure, with feedback from and meetings with the users
- Worked with C++, JS, SQL, HTML, CSS, Bigtable, and other internal tools

Human-Computer Interaction Lab [link] May–Dec 2013, Sept 2015–Present
Research Assistant Saskatoon, SK

- Developed several research projects and guided research participants through user studies
- Worked with C#, Java, C++, Android, Microsoft Kinect, and real-time networking

Education

Bachelor of Science, Computer Science (Honours) 2011–2016
University of Saskatchewan Saskatoon, SK

- Arts & Science Dean's List (every year)
- Graduated with High Honours and multiple awards
- Internship during May 2014–August 2015 (no classes)

Skills

Languages

- Expert in C++, C, Rust, and Ruby
- I learn programming languages and write compilers as a hobby and have a deep knowledge of programming language design and implementation
- Familiar with Haskell, x86 assembly, JS, HTML, CSS, LaTeX, Idris, Coq, C#, Java, Python, Clojure, Lua, Matlab, Racket, SML, Forth, Nix, and have played with many others

Tools

- Quite familiar with LLVM (some of my hobby compilers used LLVM, my IBM work involved LLVM as a Clang dependency, and it comes up a lot in Rust since rustc uses an LLVM back-end)
- Regularly use git, vim, ssh, tmux, mosh, bash, fish, and gdb
- Use NixOS as my main operating system. Also experienced with Arch Linux, Ubuntu, and others

Open-source Projects

Miri (Rust · 2015–2018)

- My undergraduate honours research project
- An experimental interpreter for the Rust compiler's mid-level intermediate representation (MIR)
- Different from most interpreters since it simulates differently-sized values, a virtual memory system, and unsafe memory operations (where invalid operations are detected and diagnosed)
- Based on a proposal for compile-time expression evaluation (like `constexpr` in C++) by one of the Rust compiler team members, who helped me throughout this project
- I got invited to a Rust compiler team workweek due to their interest in using Miri in rustc
- Later on, I collaborated with other contributors who were interested in the project, who ultimately took charge of the project when I had no free time for it
- As of early 2018, Miri has been integrated into the official Rust compiler and is being used for evaluation of compile-time constants just as originally planned

rustc (Rust · 2015–2018)

- I am a member of the Rust style guide team and language design team, and have been invited to multiple Rust team workweeks hosted by Mozilla
- I contribute to the open-source Rust compiler partly for my work on Miri but mainly because I am excited by Rust and its potential
- I have learned much about compiler internals through my work on rustc and Miri

Apricot (Ruby · 2012–2013)

- Compiled a simple Clojure-like language to Rubinius bytecode
  - Rubinius is an alternative Ruby implementation with a C++ bytecode VM
  - Apricot code was able to call functions defined in Ruby code and vice versa
- Collaborated remotely with a friend