COMS W6156 - Topics in Software Engineering

Spring 2019

Preliminary Proposal

Bruno Martins - bm2787

Rust is a new (version 1.0 was released in 2015) systems programming language developed by Mozilla as a fast and safe alternative to the dominant C and C++ languages. In their words: "Rust is a systems programming language that runs blazingly fast, prevents almost all crashes, and eliminates data races". While most of the language features borrows judiciously from other languages, there is one central novel feature that seems to be unique to Rust: the borrow checker. The borrow checker **statically** evaluates the ownership and the lifetime of all pieces of data in a program to find and prevent entire classes of bugs like data races and use-after-free.

My proposal is to put their claims to test.

First, I'd need to familiarize myself with Rust. Towards that end, I plan on reading most of "The Rust Programming Language" book [1] in order to learn how to program in it.

These are the questions I'd like to answer or investigate:

Would Rust prevent bugs in an actual, production codebase?

Towards this end, I will pick a production codebase that I am fairly familiar with, EPICS (Experimental Physics and Industrial Control System, written in C and C++), and:

1) Comb through their bugs [2] (their bug tracker goes back 16 years)
2) Select a few interesting bugs that are reasonably easy to isolate and reproduce
3) Isolate and reproduce the bug in their source language
4) Try to reproduce the bug in Rust, attempting to stay as close as possible to the original code structure
5) If the bug already has a fix, investigate if the fix would be applicable, or even necessary, in Rust

How easy is it to port code from C or C++ to Rust?

This question will be answered in part by the work mentioned in the previous question, in part by reading about other people's experiences [3] [4] [5], and in part by doing a small survey. I have a friend that works with Rust as a full time job and I think I can get him and a few of his coworkers to fill out a survey.

Resistance to mutations

I'd also like to incorporate the professor's suggestions on investigating mutation analysis. EPICS currently has a fair amount of tests, so I could start by using a mutation tool to introduce bugs there, but I am not quite sure how to do the same on the Rust side. The EPICS codebase is large. Maybe I could look into the snippets that I'd have in both languages from the "Would Rust prevent bugs in an actual, production codebase?" work, write tests to them and then evaluate the effects of similar mutations on both of them.

My plan for the final project is to reimplement a whole module of EPICS into Rust. Maybe I could investigate mutations then?

Since the academic literature on Rust is scarce (as far as I could tell), I'd be reading other documents that I found on Rust too, like [6] [7] . One particular paper that might be very interesting is "RustBelt: Securing the Foundations of the Rust Programming Language" [8] in which the authors claim to provide formal, machine-checked proofs to Rust's safety claims, apparently using Separation Logic - one of the theoretical foundations of the paper that I presented this week on Infer and FootPatch. If this paper turns out to be important for my paper, I'd also try to get a better grip on the theoretical underpinnings of Separation Logic, starting with [9] (which seems to be the seminal paper on this topic?).

Following the professor's suggestions, I will include a primer on Separation Logic section, and consider submitting the paper to OOPSLA if I get good results.

[1] J. Blandy, *The Rust Programming Language: Fast, Safe, and Beautiful*. O'Reilly Media, Inc., 2015.

[2] "Bugs : EPICS Base." [Online]. Available: https://bugs.launchpad.net/epics-base/+bugs?field.searchtext=&search=Search&field.status%3Alist=NEW&field.status%3Alist=CONFIRMED&field.status%3Alist=TRIAGED&field.status%3Alist=INPROGRESS&field.status%3Alist=FIXCOMMITTED&field.status%3Alist=FIXRELEASED&field.status%3Alist=INCOMPLETE_WITH_RESPONSE&field.status%3Alist=INCOMPLETE_WITHOUT_RESPONSE&assignee_option=any&field.assignee=&field.bug_reporter=&field.bug_commenter=&field.subscriber=&field.structural_subscriber=&field.tag=&field.tags_combinator=ANY&field.has_cve.used=&field.omit_dupes.used=&field.omit_dupes=on&field.affects_me.used=&field.has_patch.used=&field.has_branches.used=&field.has_branches=on&field.has_no_branches.used=&field.has_no_branches=on&field.has_blueprints.used=&field.has_blueprints=on&field.has_no_blueprints.used=&field.has_no_blueprints=on&orderby=datecreated&start=0. [Accessed: 06-Feb-2019].

[3] A. Zeng and W. Crichton, "Identifying Barriers to Adoption for Rust through Online Discourse," *ArXiv190101001 Cs*, Jan. 2019.

[4] B. Anderson *et al.*, "Experience Report: Developing the Servo Web Browser Engine using Rust," *ArXiv150507383 Cs*, May 2015.

[5] Y. Lin, S. M. Blackburn, A. L. Hosking, and M. Norrish, "Rust As a Language for High Performance GC Implementation," in *Proceedings of the 2016 ACM SIGPLAN International Symposium on Memory Management*, New York, NY, USA, 2016, pp. 89–98.

[6] A. Balasubramanian, M. S. Baranowski, A. Burtsev, A. Panda, Z. Rakamarić, and L. Ryzhyk, “System Programming in Rust: Beyond Safety,” in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, New York, NY, USA, 2017, pp. 156–161.

[7] A. Levy, B. Campbell, B. Ghena, P. Pannuto, P. Dutta, and P. Levis, “The Case for Writing a Kernel in Rust,” in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, New York, NY, USA, 2017, pp. 1:1–1:7.

[8] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer, “RustBelt: Securing the Foundations of the Rust Programming Language,” *Proc ACM Program Lang*, vol. 2, no. POPL, pp. 66:1–66:34, Dec. 2017.

[9] J. C. Reynolds, “Separation Logic: A Logic for Shared Mutable Data Structures,” in *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, Washington, DC, USA, 2002, pp. 55–74.