Rust's Borrow Checker

This is based on the work described in a paper!

“RustBelt: Securing the Foundations of the RustProgramming Language”

https://people.mpi-sws.org/~dreyer/papers/rustbelt/paper.pdf

Code is awful

Code is awful

- Our favorite topic, the failings of traditional language paradigms!

Code is awful

- The Billion Dollar Mistake
- Threading
  - ESR calls this an ugly hack, favors IPC

Code is awful

- The Billion Dollar Mistake
- Threading
  - ESR calls this an ugly hack, favors IPC

The real problem is managing your search space.

Code is awful

- How much can you prove about your code?
- How much does your type system give you?
- How quickly do you identify that a problem exists?

Code is awful

Java!

Code is awful

Java!

http://wouter.coekaerts.be/2018/java-type-system-broken

https://www.cs.rice.edu/~javaplt/papers/oopsla2008.pdf

Code is awful

C#!

Code is awful

C#!

https://stackoverflow.com/questions/123506/c-sharp-compiler-incorrectly-optimizes-code

Code is awful

C!

Code is awful

C!

https://llvm.org/pubs/2008-10-EMSOFT-Volatiles.pdf

http://www.cs.utah.edu/~regehr/papers/pldi11-preprint.pdf

https://www.youtube.com/watch?v=tU5HfVc2nR8

Code is awful

Go!

Code is awful

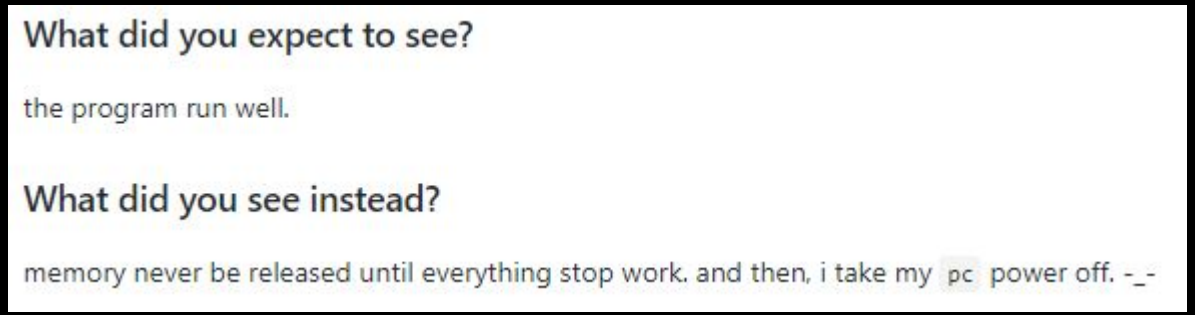
Go!

https://github.com/golang/go/issues/22350

https://twitter.com/SusanPotter/status/1129593107710922752

Code is awful

Go!



What did you expect to see?

the program run well.

What did you see instead?

memory never be released until everything stop work. and then, i take my `pc` power off. -_-

Code is awful

Go!

Code is awful

Haskell!

Code is awful

Haskell!

(Yes, really)

Code is awful

Haskell!

https://gitlab.haskell.org/ghc/ghc/issues/163

https://replit.canny.io/bug-reports/p/haskell-getline-bug-again

Code is awful

Nevermind…

- Security Vulnerabilities
- Risk to Life
  - Medical Device Firmware
  - Vehicle Firmware
- Lost Dev Time
- Site Reliability...

Is there hope?

Is there hope?

- Tests?
- Types?
- Oncall Rotations?
- Code Reviews?
- Continuous Integration?
- Automatic push?
- Health checks?
- Prayer?
- Sacrifices?


Be Brave. There is yet hope.

Why do bad things happen to good processes?

Part one: Multiple Aliases

In C/C++, you would say that you have a memory address, with multiple pointers to that address across different threads.

Without any kind of locking or mutexing, this gives rise to a race condition.

Part one: Multiple Aliases

More formally:

- Multiple Aliases (pointers)
- To a resource (memory address/variable)

—

Part one: Multiple Aliases

More formally:

- Multiple Aliases (pointers)
- To a resource (memory address/variable)

This still leaves us in a data race. Rust’s locking mechanism is Ownership.

—

Part one: Multiple Aliases

- A lock enforces that only one code location has access to … something.
- Critically, this is opt-in. You have to wrap the operations you're doing to avoid a data race
- Ownership, being a type system mechanism, is opt-out.
    - unsafe is available if you need to do black magic.
    - However, obviously, you're on your own.

Conclusion one!

Conclusion one!

- We need to prove that the Ownership model is sound.
- We need to prove that following the type system in rust yields a program without unsafe/undefined behaviors.

unsafe considered… unsafe

unsafe considered... unsafe

There are many cases where the performant version of the algorithm flagrantly defies the borrow checker.

Many of those cases are in the Rust standard library.

unsafe considered... unsafe

This has lead to data races that only show up if a particular combination of standard libraries are used together.

The libraries in question use unsafe internally. They are well behaved in isolation.

unsafe considered... unsafe

This has lead to data races that only show up if a particular combination of standard libraries are used together.

The libraries in question use unsafe internally. They are well behaved in isolation.

unsafe considered... unsafe

There are at least two papers documenting multi-library data race bugs.

This gives you some idea of how much fun it was to troubleshoot and fix. :-)

Conclusion Two!

Conclusion Two!

- We need to prove that the Ownership model is sound.
- We need to prove that following the type system in rust yields a program without unsafe/undefined behaviors.

Conclusion Two!

- We need to prove that the Ownership model is sound.
- We need to prove that following the type system in rust yields a program without unsafe/undefined behaviors.

- We also need to prove that the use of unsafe in the stdlib is…. Safe.

Formal Verification

Formal Verification

Formal Verification is unique.

- It tests a *desgin specification*.
- It’s common to be done before a single line of code is written

Formal Verification

- “If implemented correctly, will this design work?”

Formal Verification

There are lots of ways this is done.

- TLA+
  - Temporal Logic of Actions
- Coq, Agda, QuickCheck, others.
- Iris
  - https://iris-project.org/

—

Iris

- Iris is written in Coq
- It's specifically designed to reason about the problem domain we're working in!

From the website:

Iris is a Higher-Order Concurrent Separation Logic Framework implemented and verified in the proof assistant Coq.

Example Programs

Live coding because what could go wrong! :-D