Slides - 7/12

Justin Restivo

Table of contents

Pitch

Problem Statement

- Large amount of legacy C code
- C is not memory safe. 70% of MSFT bugs are memory safety bugs.
- Goal: maintain legacy codebase in a closer to memory safe language
- Rust

Existing work

- FFI with C code is (1) difficult to maintain and (2) clunky
- C2Rust has limitations
  - Output code "probably" works on a 3 year old Rust compiler
  - Build: hacky python scripts
  - Doesn't handle edge cases
    - Syntactic bugs
    - UB
  - Compiles to nightly Rust

Existing work

- Existing unsafe -> safe lifting literature does not show lifting from unsafe to safe code is correct
  - CrustS
  - Crown
  - Translating C to safer Rust

Existing work

Why is this interesting?

- Rust mainstreams novel ideas to systems programming
  - lifetime and ownership semantics
  - XOR mutability, interior mutability
  - “unsafe” vs “safe”
- Formalization of Rust is relatively unexplored:
  - Rust specification does not exist
  - Existing work incomplete: minirust, rustbelt, ferrous
- High impact: Rust used extensively in industry

Vision of our tool

- Compile C to Rust
- Improve on C2Rust flaws
- Guarantee that Rust code matches or improves on behavior of C code
- Provide formalization of lifting literature

Implementation

High Level View

- CFRust: C “friendly” Rust IR. No lifetime or pointer reasoning. Handles control flow differences and types
- RustLight: bare minimum IR representing Rust subset that include lifetimes, ownership, and Rust-like pointers (*const, *mut, &)
- RustSafe: introduce all Rust pointer types, lifting passes would happen here

Current progress

- Three moving pieces:
  - CFRust IR
  - Translation Clight -> CFRustIR
  - Extraction

Plan

Within the next month

- Finalize CFRust IR/translation next week
- Test translation with identified edge cases
  - examples that C2Rust cannot handle
- RustLight IR, translation, semantics, extraction

Questions

- Should I rebase to nominal compcert?
- Handling of pointer rules + ownership