Formal Verification for Rust: Where are we now?

Molly MacLaren
CASC
John Sarracino, Matt Sottile

COMPUTING

CASC
Center for Applied Scientific Computing

LLNL-PRES-867435
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Formal Verification for Code Correctness and Safety

CASC

Summer SCAM!

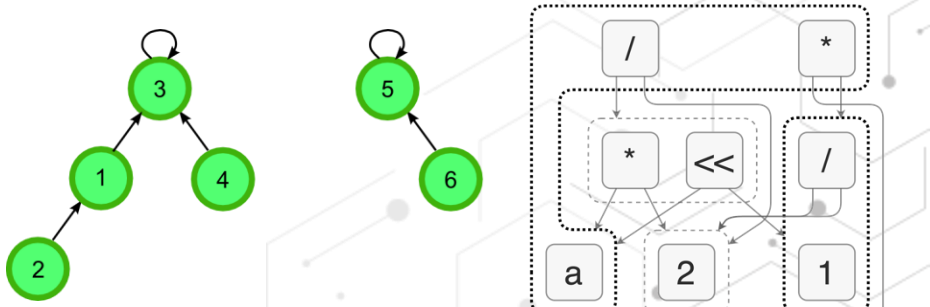
COMPUTING SCHOLAR PROGRAM
Lawrence Livermore National Laboratory

- What is Formal Verification?
  - An aspect of **software assurance**
  - Enables **proving mathematical models** of program execution
    - Testing → only checks for finite inputs
    - FV → checks for all possible inputs!
- Why Rust?
  - **Ownership** properties ensure memory safety at compile-time
    - Prevents bugs such as in the recent Crowdstrike incident
    - Simplifies proof logic!
  - **Functional programming** paradigms present in Rust
    - Data is immutable by default → **deterministic** functions
- Code correctness is essential in critical systems!

```
#[requires(x < i32::MAX)]
#[ensures(result == x + 1)]
fn add_1(x: i32) -> i32 {
        x + 1
}
```

- My contribution & Case Study

  Verification of the Union-Find data structure found in *e-graphs good*



Evaluation of Rust Verification tools

CASC

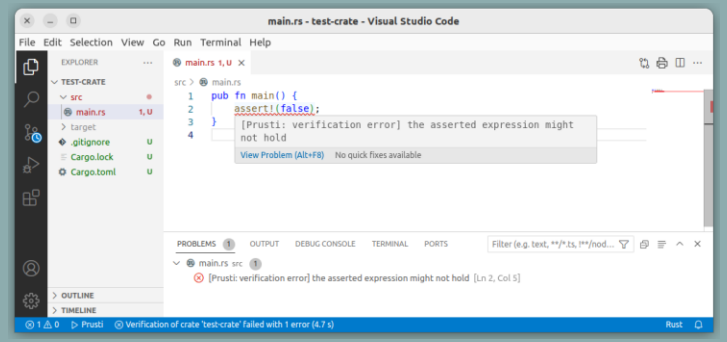
Summer SEM!

COMPUTING SCHOLAR PROGRAM
Lawrence Livermore National Laboratory

Prusti

Proved with Viper

Development entirely in VS Code



Int, Sequence, Map, Set

//TODO

∩

Allow functionally pure Rust code to be used in proofs

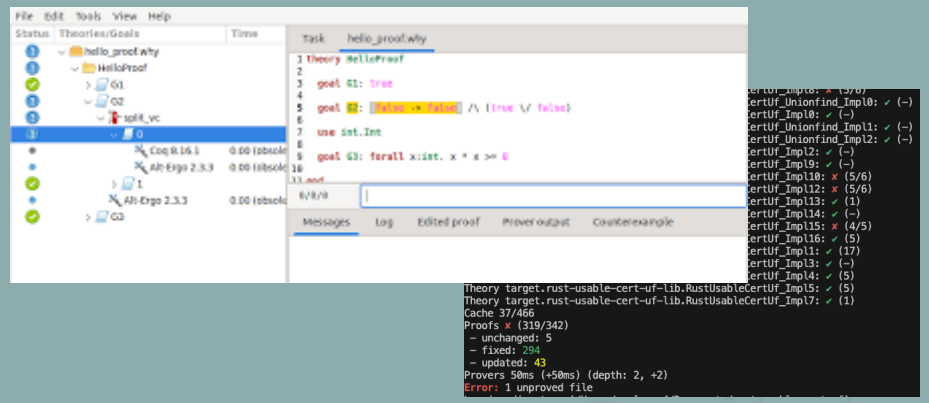
Model borrows of ownership with a “before” and “after” state

Syntactic similarities

Creusot

Proved with Why3

External environments to debug proofs



Int, Sequence, Map, Set

Where should we focus future work in Rust verification?

- My suggestion of today:
  - Start writing proofs with Prusti and “graduate” to Creusot
  - This is far from ideal.

Therefore, for all Rust verification tools, there exists demonstrated need for improvement!

Q.E.D.

- The ideal Rust verifier should have:
  - An ergonomic UX like **Prusti**
    - Seamless between writing code and proving code
    - Shows coordination between code and violated contracts
    - Comprehensive documentation and examples
  - Proving capabilities of **Creusot**
    - Ability to handle many datatypes in proof logic
    - Extended to write logical models for custom types
    - Accessible intermediate verification lang for debugging
  - Minimized friction during development
    - Stable support for proof counterexamples
    - Improved error handling