Borrow Newt: Modeling Rust Borrow Checking

May 12, 2023

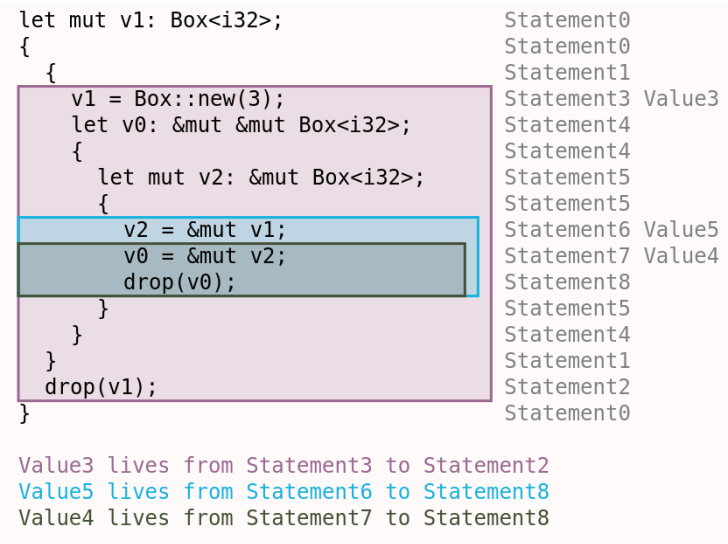
Thomas Castleman

For my capstone project, I worked with Ria Rajesh to model the borrow checking rules of the Rust programming language in Forge, a specification language similar to Alloy. Rust is a language aiming to target the same application space as C/C++, but with stronger guarantees around memory safety. It does so through a system of ownership and borrowing that governs how resources are passed around and ultimately destroyed. This allows Rust to safely handle memory management without the overhead of a garbage collector or the common pitfalls of manual memory management.

Rust makes strong claims about the guarantees it provides, which could have substantial implications for how we design secure, reliable systems, given the prevalence of memory safety errors in vulnerabilities. But, how do we know that Rust can actually deliver on these guarantees? This is where formal methods tools can play a key role, giving us confidence that the promises Rust makes are indeed fulfilled by the rules it enforces.

This project models these rules to produce instances of Rust programs (for a very small subset of the language) that follow or violate borrow checking, with the goal of verifying that memory safety properties are upheld when a program passes the borrow checker.

We were able to use Forge to generate and visualize simple Rust programs, and verify (for a bounded program size) that borrow checking prevents the creation of dangling pointers. Further work could involve enriching the model to enable it to express other properties relevant to memory safety, and checking that these too follow from borrow checking.



An example instance from our model

Repository

The model [can be found at this repository.]

Faculty Sponsor

Tim Nelson