Formally verifying properties of a toy language

Formal Verification, EPFL

Guillem Bartrina I Moreno
guillem.bartrinaimoreno@epfl.ch

Franco Sainas
franco.sainas@epfl.ch

Marcin Wojnarowski
marcin.wojnarowski@epfl.ch

January 8, 2024

Abstract

Many of the available mainstream programming languages were created without much of consideration of their formal foundation. This makes it difficult to reason about safety guarantees in retrospect. Only a subset of C programs can be formally proven to be memory safe. On the other hand, memory safety guarantees of `unsafe`-free Rust can be formally proven on the language-level[2]. In a similar vein we aim to tackle the issue of memory safety within a language. We define a toy imperative programming language with semantics that allow us to formally show that any correct program written in this language will not violate memory safety properties. We derive and prove properties from the type checking stage and then show that the interpreter preserves said properties. The memory safety properties of interest are: no dangling references, no overlapping memory regions, and no access to unallocated/unowned memory. As a backing paper we use the formal definition of the static analysis of the Move language[1] which concerns itself with similar memory issues.

References

[1] Sam Blackshear et al. *The Move Borrow Checker*. 2022. arXiv: 2205.05181 [cs.PL].

[2] Ralf Jung et al. "RustBelt: Securing the Foundations of the Rust Programming Language". In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017). DOI: 10.1145/3158154. URL: https://doi.org/10.1145/3158154.