Internship proposal: Soundness of Rust Verification in the Presence of Safe and Unsafe Code

Advisor:
Yannick Zakowski
Team Cambium – Inria
yannick.zakowski@inria.fr

Co-Advisor:
Son Ho
Microsoft Azure Research
t-sonho@microsoft.com

Location of the internship: Paris, France

Context Over the past decade, the Rust programming language has gained traction both in academia and in industry [4, 1, 5, 13, 2, 9, 15], consistently ranking as the most beloved language by developers [18, 14] for the past 8 years. A large part of this success stems from several key features of the language: Rust provides both the high performance and low-level idioms commonly associated to C or C++, as well as memory-safety by default thanks to its rich, borrow-based type system. This makes Rust suitable for a wide range of applications: both Windows [16] and the Linux kernel [3] now support Rust, the latter marking the first time a language beyond C was ever approved for Linux. The safety guarantees of Rust are particularly appealing for security-critical systems: leading governments now also recommend Rust [17, 12].

Of course, despite being safer than C or C++, Rust programs are not immune to bugs and vulnerabilities. Case in point, between January 1, 2024 and January 1, 2025, 137 security advisories against Rust crates were filed on RUSTSEC, a vulnerability database for the Rust ecosystem maintained by the Rust Secure Code working group. These vulnerabilities arose due to several reasons: runtime errors leading to aborted executions (`panic` in Rust, e.g., after an out-of-bounds array access), implementation or design flaws, or even memory vulnerabilities when using Rust's `unsafe` escape hatch, which allows the use of C-like, unchecked pointer operations when Rust's borrow-based type system is too restrictive.

To enforce those properties that fall outside the scope of Rust's borrow-checker, a solution is to use formal verification tools. One such tool, Aeneas [7], leverages Rust's rich region-based type system to generate a lightweight, pure translation of a large class of Rust programs. This model is then extracted to the Lean theorem prover, allowing one to formally reason about it to prove properties such as functional correctness. Doing so, Aeneas allows a lightweight verification by eliminating memory reasoning, allowing the proof engineers to instead focus on *functional* properties of their code. This approach has proved successful enough for Aeneas to be applied to the ongoing verification of SymCrypt, Microsoft's open source cryptographic library, which is notably used in the Windows kernel and in Azure Linux [11]. Moreover, an effort is under way to allow combining Aeneas' pure translation with a stateful embedding, which would allow leveraging separation logic to reason about code containing features such as unsafe, concurrency, I/O or interior mutability, and which are today outside the scope of the translation.

The translation performed by Aeneas, which is based on a symbolic execution, is however complex, making its correctness a non-trivial result. Some previous work [6], which is in the process of being mechanized, has shown that (a subset of) Aeneas' translation actually implements a borrow-checker, that is: a program which is successfully translated by Aeneas is memory safe. This result, by focusing on the memory safety, leaves aside the question of formally linking the generated model, and the theorems which apply to it, to the original Rust code. Moreover, it is unclear how to extend this proof to a subset of Rust which combines features which can be functionalized by using the pure translation, with features which can not and would be reasoned about using separation logic.

Goals The goal of the internship is to reason about the soundness of Aeneas' translation in the presence of code which can not be functionalized. A possible direction would be to use a semantic model of the Rust language as successfully performed by RustBelt and RustHornBelt [8, 10]. By using separation logic to introduce a semantic model of LLBC environments [6], one could combine those environments with separation logic predicates which do not fit into LLBC (e.g., "points to" predicates). This would allow studying Rust functions which are currently not covered by the translation (e.g., `mut_to_raw`, which converts a mutable borrow to a raw pointer), and on the longer term prove a correctness theorem for the translation in the presence of code which can not be functionalized.

Qualifications This internship of 6 months or less would be hosted by the team Cambium at Inria Paris. The preferred qualifications for the student at the beginning of the internship would be:

- familiarity with the Rust programming language;
- fluency with a functional programming language like OCaml;
- some experience with at least one theorem prover (Rocq, F*, LEAN, Isabelle/HOL, ...);

References

[1] Vytautas Astrauskas, Peter Müller, Federico Poli, and Alexander J. Summers. Leveraging Rust types for modular specification and verification. 2019.

[2] Yechan Bae, Youngsuk Kim, Ammar Askar, Jungwon Lim, and Taesoo Kim. Rudra: finding memory safety bugs in rust at the ecosystem scale. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 84–99, 2021.

[3] Kees Cook. [GIT PULL] Rust introduction for v6.1-rc1. `https://lore.kernel.org/lkml/202210010816.1317F2C@keescook/`.

[4] Xavier Denis, Jacques-Henri Jourdan, and Claude Marché. Creusot: a foundry for the deductive verification of rust programs. In *International Conference on Formal Engineering Methods*, pages 90–105. Springer, 2022.

[5] Lennard Gäher, Michael Sammler, Ralf Jung, Robbert Krebbers, and Derek Dreyer. RefinedRust: A type system for high-assurance verification of Rust programs. *Proceedings of the ACM on Programming Languages*, 8(PLDI):1115–1139, 2024.

[6] Son Ho, Aymeric Fromherz, and Jonathan Protzenko. Sound borrow-checking for Rust via symbolic semantics. *Proceedings of the ACM on Programming Languages*, 8(ICFP):426–454, 2024.

[7] Son Ho and Jonathan Protzenko. Aeneas: Rust verification by functional translation. *Proceedings of the ACM on Programming Languages*, 6(ICFP):711–741, 2022.

[8] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: Securing the foundations of the Rust programming language. 2018.

[9] Andrea Lattuada, Travis Hance, Chanhee Cho, Matthias Brun, Isitha Subasinghe, Yi Zhou, Jon Howell, Bryan Parno, and Chris Hawblitzel. Verus: Verifying Rust programs using linear ghost types. 2023.

[10] Yusuke Matsushita, Xavier Denis, Jacques-Henri Jourdan, and Derek Dreyer. Rusthornbelt: a semantic foundation for functional verification of rust programs with unsafe code. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, page 841–856, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Microsoft Research. Rewriting symcrypt in rust to modernize microsoft's cryptographic library, 2025.

[12] National Security Agency. Software memory safety. https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF, 2022.

[13] Vikram Nitin, Anne Mulhern, Sanjay Arora, and Baishakhi Ray. Yuga: Automatically detecting lifetime annotation bugs in the rust language. *IEEE Transactions on Software Engineering*, 2024.

[14] StackOverflow. 2023 developer survey. https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages, 2023.

[15] The Register. In Rust we trust: Microsoft Azure CTO shuns C and C++. https://www.theregister.com/2022/09/20/rust_microsoft_c/, 2022.

[16] The Register. Microsoft is busy rewriting core Windows code in memory-safe Rust. https://www.theregister.com/2023/04/27/microsoft_windows_rust/, 2023.

[17] The White House. Back to the Building Blocks: a Path Toward Secure and Measurable Software. https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf.

[18] Sara Verdi. Why rust is the most admired language among developers. https://github.blog/2023-08-30-why-rust-is-the-most-admired-language-among-developers/, 2023.