Internship proposal:
Formal verification of Rust programs

Advisor:
Son Ho
Team Prosecco – Inria
son.ho@inria.fr

Head of department:
Karthikeyan Bhargavan
Team Prosecco – Inria
karthikeyan.barghavan@inria.fr

Location of the internship: Paris, France

Context Program verification is a notoriously difficult and time-consuming task, in particular when reasoning about pointer aliasing and mutable state. This indeed requires dedicated tooling and logics [15, 9, 12, 16], an expert knowledge of the verification tools and the target languages, as well as a huge amount of work [10, 15, 9], impeding large-scale verification efforts.

Rust is a young programming language with an affine type system and an ownership mechanism, which allow fine control of memory with a low overhead like C, while providing strong type and memory safety guarantees similarly to high-level languages typically relying on a garbage collector like OCaml [3]. This combination of fine control and strong guarantees has made it a language of choice for more and more industrial projects [4, 1], and has triggered a growing trend to leverage its type system to simplify reasoning about stateful programs [5, 13].

Goals While the growing interest for Rust has led to the development of several dedicated verification frameworks [5, 11, 14, 8, 17, 2], the successful application of such frameworks to the verification of large real-world Rust programs is yet to be explored.

Accordingly, the goal of this internship will be to apply verification techniques to programs written in Rust and targeting real world applications, a potential candidate being the efficient implementation of a cryptographic protocol such as MLS [6]. A first step might be to use the Hacspec [14] subset of Rust to write and specify such a program, then expand this analysis with more advanced tooling to use a larger subset of Rust, or reason about non-functional properties such as security guarantees [7].

As progress is made on the verification work, the intern will be encouraged to identify shortcomings in the existing techniques as well as possible improvements, potentially leading to contributions to the tools themselves if time allows.

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

- fluency with the Rust programming language;
- fluency with a functional programming language like OCaml;
- some experience with at least one theorem prover (Coq, F*, LEAN, HOL4, Isabelle/HOL, ...);

References

[1] 9 companies that use rust in production. https://serokell.io/blog/rust-companies.

[2] Project oak. https://github.com/project-oak/rust-verification-tools.

[3] The rust programming language. https://doc.rust-lang.org/book/title-page.html.

[4] What is rust and why is it so popular? https://stackoverflow.blog/2020/01/20/what-is-rust-and-why-is-it-so-popular/.

[5] Vytautas Astrauskas, Peter Müller, Federico Poli, and Alexander J Summers. Leveraging rust types for modular specification and verification. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–30, 2019.

[6] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The messaging layer security (mls) protocol. https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/.

[7] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. Dy*: a modular symbolic verification framework for executable cryptographic protocol code. In *EuroS&P 2021-6th IEEE European Symposium on Security and Privacy*, 2021.

[8] Xavier Denis. Creusot. https://github.com/xldenis/creusot.

[9] AYMERIC FROMHERZ, ASEEM RASTOGI, NIKHIL SWAMY, SYDNEY GIBSON, GUIDO MARTÍNEZ, DENIS MERIGOUX, and TAHINA RAMANANANDRO. Steel: Proof-oriented programming in a dependently typed concurrent separation logic. 2021.

[10] David Greenaway, June Andronick, and Gerwin Klein. Bridging the gap: Automatic verified abstraction of c. In *International Conference on Interactive Theorem Proving*, pages 99–115. Springer, 2012.

[11] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: Securing the foundations of the rust programming language. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–34, 2017.

[12] Robbert Krebbers, Ralf Jung, Aleš Bizjak, Jacques-Henri Jourdan, Derek Dreyer, and Lars Birkedal. The essence of higher-order concurrent separation logic. In *European Symposium on Programming*, pages 696–723. Springer, 2017.

[13] Yusuke Matsushita, Takeshi Tsukada, and Naoki Kobayashi. Rusthorn: Chc-based verification for rust programs (full version).

[14] Denis Merigoux, Franziskus Kiefer, and Karthikeyan Bhargavan. *Hacspec: succinct, executable, verifiable specifications for high-assurance cryptography embedded in Rust.* PhD thesis, Inria, 2021.

[15] Jonathan Protzenko, Jean Karim Zinzindohoué, Aseem Rastogi, Tahina Ramananandro, Peng Wang, Santiago Zanella Béguelin, Antoine Delignat-Lavaud, Catalin Hritcu, Karthikeyan Bhargavan, Cédric Fournet, et al. Verified low-level programming embedded in f. *Proc. ACM program. lang.*, 1(ICFP):17–1, 2017.

[16] Michael Sammler, Rodolphe Lepigre, Robbert Krebbers, Kayvan Memarian, Derek Dreyer, and Deepak Garg. Refinedc: automating the foundational verification of c code with refined ownership types. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 158–174, 2021.

[17] Sebastian Ullrich. Electrolysis. https://github.com/Kha/electrolysis.