Verified interoperability between OCaml and C

Research internship proposal (Master 2 level)

2022

Supervisor

Armaël Guéneau, researcher at LMF, Inria Saclay (armael.gueneau@inria.fr)

Summary

It is not uncommon for OCaml libraries to interoperate with low-level libraries implemented in C. This is typically achieved by writing additional "glue" code, implemented in C but subject to additional restrictions imposed by the implementation of the OCaml language. As of today, there does not exist a formal specification of those restrictions; and there is thus no fully rigorous way of checking the safety of such glue code. In this internship, I propose to work on designing suitable reasoning rules for verified interoperability between OCaml and C.

Overview

There exist today many approaches for formally verifying the safety or functional correctness of programs; those approaches typically focus on verification of programs implemented in one programming language.

However, in practice, a non-negligible number of real-world libraries are implemented using a combination of not one but several languages. This is typically the case for libraries in a high-level language (such as OCaml) partially implemented using low-level code (typically C)—for either increased efficiency, the need to access primitive operations not available in the high-level language, interacting with the operating system, or calling into third-party libraries implemented in a different language.

The so-called *Foreign Function Interface* (FFI) defines the rules of how a (e.g. high-level) language can be linked with low-level code. In practice, the exact details of its FFI then vary from one language and the other. In the case of OCaml, the user is provided with a set of C headers, that they can then use to write C code to act as "glue" between their OCaml code and external C code. This provides the programmer with a lot of control and freedom to implement very efficient FFI code; but in turn, they have to obey a number of subtle restrictions when interacting with OCaml code. These restrictions are at the moment only documented informally in the OCaml

manual [man]. Writing correct interface code between OCaml and C is still the priviledge of experts, and bug-finding is often very tricky.

My current research project is to tackle the specification and *formal verification* of these restrictions and rules for cross-language interoperability; I am leading this project in collaboration with colleagues from Aarhus University (Denmark) and MPI-SWS (Germany). One of the key ideas is to develop rules based on *Separation Logic* that allow verifying the correctness of any "glue" code used to interoperate between OCaml and C code. These reasoning rules are then proved sound with respect to a more elementary operational semantics modeling the interoperating languages. The whole setup is being formalized in Coq using the Iris Separation Logic framework [JKJ+18, iri].

Current work has been focusing on grounding the more formal parts of the approach, and is so far restricted to a minimal subset of the possible interactions between simplified mini-Caml and mini-C languages. As such, we have not yet modelized the behavior of many of the primitives used in practice by C code interacting with OCaml code.

In this internship, I propose to take inspiration from the minimal model and rules that we formalized in Coq, and design an extended set of rules. This extended set of Separation Logic rules should cover a larger subset of the C-OCaml FFI and allow verifying interesting examples of "glue" C code. Then, in order to validate these new rules, one option is to show that they are *useful*, by integrating them in an existing verification tool for C [SLK+21, JP08], and applying the tool to real life mixed OCaml-C libraries. Alternatively, one could look into proving that the new rules are *correct*, by extending the Coq formalization to show the soundness of the new rules.

Roadmap

1. Identify a number of representative OCaml libraries that use C code through the OCaml FFI. One might for instance look at the zarith [zar] library, or libraries exporting an OCaml interface for third-party C libraries (tsdl [tsd], etc). For these libraries, identify the subset of the OCaml FFI they rely on.

2. Come up with reasoning rules in Separation Logic that enable reasoning about the "glue" C code appearing in these libraries.

3. Validate these new rules, for instance by adding them to an existing verification tool and applying it to the aforementioned C code.

Prerequisites

Familiarity with program verification and preferably with Hoare logic and Separation Logic (for instance the MPRI 2.36.1 course is relevant). A solid programming background, including fluency in OCaml and familiarity with C is desirable.

Practical details

The internship will happen at LMF, building 650, plateau de Saclay. (Campus Universitaire, Rue Raimond Castaing, Bâtiment 650, 91190 Gif-sur-Yvette)

References

[iri] The Iris website. https://iris-project.org.

[JKJ+18] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. 2018.

[JP08] Bart Jacobs and Frank Piessens. The VeriFast program verifier. Technical report, Technical Report CW-520, Department of Computer Science, Katholieke ..., 2008.

[man] The OCaml manual: Interfacing C with OCaml. https://v2.ocaml.org/manual/intfc.html.

[SLK+21] 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*, PLDI 2021, page 158–174, New York, NY, USA, 2021. Association for Computing Machinery.

[tsd] The tsdl library. https://github.com/dbuenzli/tsdl.

[zar] The zarith library. https://github.com/ocaml/Zarith.