Static Analysis by Abstract Interpretation of Library Bindings in Multi-Language Applications

Master 2 research internship proposal, 2024–2025

Supervisor:	Antoine Miné (antoine.mine@lip6.fr)
Internship location:	APR team, LIP6 Sorbonne Université Jussieu Campus, Paris, France
Duration:	4 to 6 months
Related projects:	PEPR Secureval, MOPSA analyzer
Relevant courses:	MPRI 2.6: Abstract Interpretation: Application to verification and static analysis Master 2 STL: Typage et analyse statique (Sorbonne Université)

The goal of the internship is to develop a static analysis to verify language bindings, that is, "stub" functions that allow some code written in a "host" programming language to interact with a "guest" library written in another programming language. A common case is that of stubs written in the C programming language to call guest C library functions from higher-level host languages, such as OCaml, Python, or Java. Incorrect stubs can trigger errors in the C code, but also leave the run-time of the host language in an inconsistent state and cause errors that would not be otherwise possible in the host language (such as memory errors, type errors, etc.). Our goal is to develop a sound, semantic static analysis to check stub correctness. On the theoretical level, the methods will be developed and proved formally correct within the framework of abstract interpretation [1]. On the practical level, the techniques will be implemented within the MOPSA [3] open-source multi-language static analysis platform written in OCaml, extending its existing C analysis [4]. It will be evaluated experimentally on representative code fragments from library bindings.

Context and Motivation

Modern applications often combine sources written in multiple programming languages. It is quite frequent for a program to be written in a high-level language with a complex run-time and automated memory management, such as OCaml, Java, or Python, and still call libraries written in a much lower-level language, such as C. Direct calls from the high-level, so-called host language, to C is generally not supported as the data representation and call conventions differ too widely. Instead, the host language defines a set of C types and functions, so-called foreign function interface (FFI), that allows writting "stubs" C function to access and convert data-structures, and perform actual function calls to the C library. The stub functions must comply with the memory management requirements of the host language, such as the use of a garbage collector. Failure to apply the rules dictated by the FFI may result in difficult-to-spot errors. They may occur in the stub code, the guest library, but also at arbitrary points in the host code: as critical run-time invariants are not maintained anymore, it can trigger errors that, otherwise, would be impossible to observe in the host language (such as type or memory errors). The goal of the internship is to develop automated formal verification methods using abstract interpretation to ensure that stub functions obey the rules of the FFI and are free of errors.

Abstract interpretation allows the design of efficient static analyzers, able to compute approximations of program semantics and infer properties. A key application is the verification of the absence of run-time errors, such as arithmetic overflows and memory errors in C, uncaught exceptions in Python, etc. The large majority of static analyses target a single programming language

at a time and ignore stub requirements. Multi-language analysis by abstract interpretation has been studied on a theoretical level in [6]. It has been applied to the analysis of programs mixing C and Python in [5] using the MOPSA analyzer [2]. This multi-language analysis handles both the Python and C parts, but does not verify all the requirements of the FFI (such as proper reference counting). A prelimiary work analyzing programs mixing C and OCaml in MOPSA has also been conducted during an internship [9]. The Goblint static analyzer has been extended to analyze C bindings to OCaml [8], but it focuses on the specific problem of porting stubs from OCaml 4 to OCaml 5 and analyses properties related to multi-threading. Other formal methods have been proposed to analyze language interoperability; for instance, Melocoton [7] uses program logic to verify code mixing OCaml and C, which can also serve as a basis for a semantic formalization of the requirements of C stubs for OCaml.

The scope of the internship is the static analysis by abstract interpretation of C stubs for a high-level language of your choice, in order to check the adherence to the corresponding FFI. Natural candidates for the high-level language FFI are Python and OCaml (extending existing works in the MOPSA project), but other language FFI are also possible, such as Java.

Expected Work

The expected work includes both theoretical aspects (semantic development, abstraction design, proofs of correctness) and practical aspects (implementation, experimentation, evaluation on representative stub fragments).

A first step will be to select a high-level language FFI and study its requirements. A concrete semantics will then be developed to formalize this FFI (possibly based on previous works [7, 9]). The semantics will formalize the C calls to the FFI API and its relation to both the C data representation and the high-level language representation of data. In particular, when called, C stubs can make hypotheses guaranteed by the run-time of the host language, such as well-typing, variable range, etc. It will be necessary to take these hypotheses into account (e.g., leveraging the type information assigned to the stub by the host language) in order to prove the safety of the stub. Dually, the semantics shall formalize the correctness conditions the C stub code must obey when calling the FFI API, and when returning from the stub.

Secondly, a set of abstract domains able to express and infer the correctness properties must be developed. They must be proved sound and implemented in the MOPSA analyzer. Then, a set of analyses will be performed, using a well-chosen set of stubs. The analysis can be conducted completely at the C level. In this case, a "driver" must be constructed, that is, a main C function that simulates the use of the library by a high-level program by calling directly the C stubs. The simulation should cover as many cases as possible, leveraging the ability of abstract interpretation to compactly express at the abstract level a large (possibly infinite) set of memory states and program executions. An analysis thus comprises the C driver, the C stubs to analyze, as well as the guest C library it interfaces to and the FFI library provided by the host language. If needed, parts of the guest and the FFI libraries can be modeled using MOPSA analysis stubs [4], as currently used for instance to model the standard C library. The implementation work will be complemented by an experimentation on representative stub fragments.

If time permits, the internship can consider the analysis of programs in a host language supported by MOPSA (Python or OCaml) that call the stubs and the guest library. Another possible extension would be to consider the impact on the analysis of automated stub generators (such as OCamlIDL and ctypes for OCaml).

The internship will take place in the APR team, in the LIP6 laboratory, on the Jussieu Campus at Sorbonne Université, Paris. If successful, the internship could lead to a fully-funded PhD on a follow-up subject.

Required Skills

- Strong knowledge of abstract interpretation (having followed a Master-level course, e.g., at MPRI or at the STL Master at Sorbonne Université).
- Knowledge of C.

- Experience with OCaml (the language MOPSA is implemented in).
- Willingness to formalize semantics on paper, implement it in a computer, and perform experiments.

References

[1] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of POPL'77, pages 238-252, 1977.

[2] MOPSA - A Modular Open Platform for Static Analysis. https://mopsa.lip6.fr

[3] M. Journault, A. Miné, M. Monat, and A. Ouadjaout. Combinations of reusable abstract domains for a multilingual static analyzer. In VSTTE 2019, pages 1–17, Jul. 2019.

[4] A. Ouadjaout and A. Miné. A library modeling language for the static analysis of C programs. In SAS 2020, LNCS 12389, pages 223–246. Springer, Nov. 2020.

[5] R. Monat, A. Ouadjaout, A. Miné. A Multilanguage Static Analysis of Python Programs with Native C Extensions. In SAS 2021, LNCS 12913, pages 323–345. Springer, Oct. 2021.

[6] S. Buro, R. Crole, and I. Mastroeni. On Multi Language Abstraction. In SAS 2020, LNCS 12389, pages 310–332. Springer, Nov. 2020.

[7] A. Guéneau, J. Hostert, S. Spies, M. Sammler, L. Birkedal, and D. Dreyer. Melocoton: A program logic for verified interoperability between OCaml and C. In ACM Prog. Lang., Vol. 7, OOPSLA2, Oct. 2023

[8] E. Török. Targeted static analysis for OCaml C stubs: Eliminating gremlins from the code. In OCaml 2023 workshop.

[9] G. Maire and A. Miné. A multilanguage static analysis for programs operating between OCaml and C. Bachelor internship report, Sorbonne Université, 2024.