Static Value Analysis by Abstract Interpretation of Functional Languages and Application to OCaml Analysis

Master 2 research internship proposal, 2021–2022

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

The goal of the internship is to develop in the MOPSA analysis platform a static value analysis by abstract interpretation for a realistic subset of OCaml, featuring at least higher order functions, recursion, references, structure and variant types, and exceptions. The analysis should target run-time errors, such as arithmetic overflows, invalid operations, and unhandled exceptions.

Related Work

Static value analysis by abstract interpretation has been applied with some success to the verification of imperative languages (such as C [3]) and object-oriented languages (such as Java). The scope of these analyses has been extended recently to support more dynamic languages, such as JavasScript and Python [2]. These analyses infer (an over-approximation of) the possible values of variables at all program points in order to detect soundly and statically run-time errors, such as arithmetic overflows, invalid operations, invalid pointer or array access, etc. *Functional languages* have been much less targeted by value analyses.

Indeed, while there already exists a large body of work on the static analysis of functional languages, including analyses by abstract interpretation, these focus mostly on static typing, control-flow analysis [5, 6], and strictness analysis [7]. These analyses abstract away variable values, and so are not expressive enough to prove the absence of run-time errors. The goal of the internship is to develop a value analysis by abstract interpretation for OCaml that is expressive enough to prove the absence of run-time errors and general enough to support complex value domains, such as relational numeric domains.

Historically, abstract interpreters for functional languages are based on higher-order abstract domains [7] designed to abstract concrete functions as abstract functions or abstract relations, and are limited to finite or non-relational abstractions.

There are two notable exceptions, which could provide interesting ideas for this internship. Firstly, an extension of classic higher-order abstract interpretation considers using relational numeric domains locally [8]. While it targets a lambda-calculus, it could provide ideas for an analysis of OCaml. Secondly, Jhala et al. [4] propose an analysis for OCaml that is precise enough to infer arithmetic relations, but uses a hybrid approach combining refinement types with predicate abstraction, followed by a conversion to first-order programs that are analyzed with an off-the-shelf analyzer.

In the internship, we will explore solutions based purely on abstract interpretation and the design of suitable abstract domains that can be lifted to a native analysis of OCaml, including its functional and imperative features and its rich data-types. Instead of considering higher-order abstractions [7] that are the basis of control-flow and strictness analyses, but are theory-heavy, the internship will consider a more practical route. We will study whether domains that are currently successful in value analyses of imperative programs [1, 2] could be reused and extended to analyze functional ones. For instance, memory and pointer abstractions could be exploited to abstract closures.

Expected Work

The intended work will include a theoretical side: developing abstract domains adapted to the analysis of functional languages and proving formally their soundness. It will also include a practical side: implementing the analysis for a subset of OCaml and validating its benefit experimentally. The host team is developing an open-source static analysis platform, MOPSA [1], that includes an analysis of C and Python programs using several, ready-to-use abstractions, and a framework to easily extend it to new abstractions and new languages.

In practice, a first step will be to extend the abstract syntax tree (AST) used by MOPSA to support OCaml programs, and to provide a front-end to parse OCaml programs into this AST. It will be important to choose the right intermediate language as input to MOPSA, to benefit from type inference and removal of syntactic sugar.

A second step will be to design an analysis while relying as much as possible on the domains currently existing in MOPSA and adding only the domains and iterators for the constructions that are novel compared to C and Python. We expect that these new domains and iterators would include support for closures (to represent functions as values), variant types, and recursive functions (at the moment, recursion is not supported for the analysis of C and Python). One possible idea is to exploit entangled domains [8] proposed in the context of relational numeric analysis of lambda-calculus. The analysis will be experimented on small but realistic OCaml programs that make use of functional features.

After a basic analysis supporting a simple subset of OCaml is achieved, further work can include either extensions to more complex OCaml features, or the design of abstract domains more adapted to the analysis of functional languages than the ones built in MOPSA (that target C and Python).

Requested Skills

- The internship requires a strong knowledge of static analysis by abstract interpretation.
- The intern should have followed one of the following Master 2 courses: “Abstract interpretation: application to verification and static analysis” from MPRI, or “Typage et analyse statique” from the STL Master at Sorbonne Université, or an equivalent course.

- Knowledge of the OCaml language is required as it is the analysis target in this internship, but also to carry the implementation effort within the MOPSA platform [1].

Context of the Internship

The internship will take place in the APR team, in the LIP6 laboratory, Jussieu Campus, Sorbonne Université, Paris. It is proposed in the scope of the MOPSA research project.

References

[1] M. Journault, A. Miné, M. Monat, and A. Ouadjaout. Combinations of reusable abstract domains for a multilingual static analyzer. In Proc. of the 11th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE19), pages 1–17, Jul. 2019.

[2] R. Monat, A. Ouadjaout, A. Miné. Static type analysis by abstract interpretation of Python programs. In Proc. of the 34th European Conference on Object-Oriented Programming (ECOOP'20), volume 166 of LIPIcs, 2020, Dagstuhl Publishi, 17 pages.

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03), pages 196–207. ACM, Jun. 2003.

[4] R. Jhala, R. Majumdar, and A. Rybalchenko. HMC: Verifying Functional Programs Using Abstract Interpreters. In Computer Aided Verification (CAV 2011), pp. 470–485, 2011.

[5] F. Nielson and H. R. Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In POPL'97: Proc. of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1997, ACM, pp. 332–345.

[6] B. Montagu and T. Jensen. Trace-based control-flow analysis. In PLDI'21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, ACM, pp .482–496.

[7] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In Proc. of the 1994 International Conference on Computer Languages, pp. 95—112, IEEE Press, 1994.

[8] S. Liang, M. Might. Entangled abstract domains for higher-order programs. In the Annual Workshop on Scheme and Functional Programming, 2013.