Combining Query Languages and Semantic Analysis by Abstract Interpretation to Verify Program Correctness

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:	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 query language for static code analysis that can express constraints on semantic information (such as variable values), and to develop a static analysis based on abstract interpretation [1] to check these queries. We aim at going beyond mostly syntactic languages and checkers, such as the popular tool CodeQL [5], to provide instead semantic-based queries that a programmer can exploit to customize the correctness properties an abstract interpreter should check. 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 static analysis platform written in OCaml to extend its existing C analysis [4], and evaluated experimentally on representative code fragments and security-related properties.

Context and Motivation

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. Such correctness properties are called *non-functional*, as they do not depend on the behavior expected for a specific program, but on requirements related to the language and the environment (such as constraints on the arguments of library calls). Static analyzers by abstract interpretation for run-time errors and undefined behaviors are now used routinely in the industry (in particular in the embedded industry), as witnessed by the success of tools such as Astrée and Frama-C Eva. This is also the current scope of the MOPSA [2] academic platform developed in the APR team to check C and Python programs. Nevertheless, there is an incentive to go beyond run-time error analysis.

One possibility is to focus instead on *functional* correctness. Traditional functional correctness verification requires the programmer to write program specifications. These generally take the form of pre and post-conditions attached to each function. They are costly to develop and difficult to reuse across programs as they are program- and location-specific. Traditional tools to check functional correctness employ logic-based deductive methods (such as Dijkstra's predicate transformers) and automate the verification of the specification through SMT solvers, which can require additional annotations (such as loop invariants). Static analysis tools for run-time errors can also support a form of functional verification by incorporating the specification as *assertions*. The resulting analysis does not

require annotations in addition to the specification, but it can be limited by the expressiveness of the assertion language and the precision of its abstract domains. This is used, for instance, in MOPSA to check the preconditions of calls to the C standard library [3], with the addition of a specific contract language and specific abstract domains.

This internship proposes to explore another way to go beyond run-time error checking: develop a query language for the programmers to express declaratively properties to check. The properties to check are thus provided externally, and checked globally on the whole code, without requiring the programmer to annotate it. We take inspiration from CodeQL [5], which is a popular SAST (Static Application Security Testing) tool, integrated for instance with GitHub. Large libraries of general-purpose CodeQL queries to check for common vulnerabilities are already available, and it is possible to create new queries adapted to specific software. However, CodeQL works mostly at the syntactic level, providing a powerful language for of pattern matching, with some support for data flow and taint analysis, but it does not support deep semantic analyses, such as value analyses. The goal of the internship is to develop a static analysis by abstract interpretation that can check for code queries that, contrary to CodeQL, leverage the result of deep semantic analyses and have soundness guarantees.

Expected Work

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

A first step will be to study code query languages, such as CodeQL [5], and extract a small, meaningful set of constructions that suffice to express common queries. The language will then be extended to include semantic information about the program, notably variable values. In addition to variable values, the query language could also leverage the taint analysis, which has been included recently in MOPSA [6]. The internship will focus on the ability to express as queries security vulnerabilities that are out of the scope of current query languages. A second step will be the development of a concrete semantics for this language, and provide a formal meaning to the fact that a program fragment matches a query. The last step will be the design and implementation of a sound static analysis parameterized by a query, that leverages the MOPSA analyzer. Two different approaches seem possible. Firstly, the analyzer could be modified to export a database of abstract facts at the end of a regular analysis. Then, an external tool is developed to take this database and the query as arguments, and report whether and where the query is matched. Alternatively, the analyzer could be enhanced to take the query as argument and perform a static analysis specific to the query. The second option is more involved and requires runinning the static analysis for each query, but presents better opportunities for speed and precision improvements.

The target language will be C, as it is well supported by the MOPSA analyzer [3]. The implementation work will be complemented by an experimentation on representative fragments of C programs and security-relevant examples of queries.

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] O. de Moor, M. Verbaere, E. Hajiyev, P. Avgustinov, T. Ekman, N. Ongkingco, D. Sereni, J. Tibble. QL for Source Code Analysis. In Source Code Analysis and Manipulation (SCAM). 2007.

[6] Francesco Parolini, Antoine Miné. Sound abstract nonexploitability analysis. In VMCAI 2024.