Backward Abstract Interpretation using Over and Under-Approximations

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 2 STL: Typage et analyse statique

The goal of the internship is to develop backward value analyses for MOPSA, a modular static analyzer by abstract interpretation targeting realistic C and Python programs but currently only supporting forward analyses. The internship may consider both over-approximations and, if time permits, under-approximations.

Related Work

The theory of abstract interpretation allows the design of effective and efficient static analyzers able to compute approximations of program semantics.

Over-approximations. The theory and implementation of abstract interpretation is mostly concerned with over-approximations:

1. A classic forward analysis computes an over-approximation of the states reachable from the program entry.
2. A classic backward analysis computes the states co-reachable from some target states (i.e., it computes the entry states from which an execution can go into one of the target states). This set of entry states is also over-approximated. Thus, the analysis infers *necessary conditions* on entry states for the program to reach the target states. Any entry state that does not satisfy the conditions cannot possibly lead to the target states, while an entry state that does satisfy them may or may not lead to the target states.
3. Generally, a forward analysis must be performed before a backward analysis in order to achieve a sufficient precision. Over-approximating forward and backward computations can then be iterated to gain more precision or to specialize the analysis to specific properties. Applications include: providing automatically some context for the false positives of an imprecise analysis [6], refining automatically the analysis into an abstract testing procedure[4], or performing interactive abstract debugging [3].

In practice, backward over-approximating analysis is mostly supported in numeric abstract domains, while analyzers also rely on abstract domains representing pointers and memory structures, as well as complex compositions and interactions of several domains [7]. Moreover, it is challenging to make backward analysis scale up due to the need to store invariants. Thus, it is less frequently employed in industrial-strength analyzers that target real languages, such as C: these are generally limited to a simple forward pass. One aspect of the internship is to extend existing techniques for over-approximating backward analyses to address some of these limitations, and integrate them into an analyzer for C programs.

Under-approximations. In theory, an under-approximating backward analysis could be employed to provide *sufficient conditions* for a target program state to be reached (instead of necessary conditions). Applications include:

1. **Proving that an alarm is a true error** and not a false alarm by inferring a counter-example execution [1].
2. **Inferring procedure contracts**: sufficient assertions to insert at the beginning of a procedure to ensure that its execution will never fail [2].
3. **Evaluating the uncertainty of an analysis** by combining both over-approximations and under-approximations, and quantifying the distance between them. Such analyses may be iterated to improve their precision.

So far, the lack of effective under-approximating infinite-state abstract domains has limited the development of abstract interpretation techniques to solve these problems. Counter-example generation with formal verification has been widely explored in the context of model-checking, but requires reasoning in a finite or sufficiently regular world that can be exactly represented explicitly or symbolically by the model-checker, and is not directly applicable to general, infinite-state abstract interpretation. The internship will thus explore novel methods that can handle large state spaces and sound approximations. Procedure contract inference by abstract interpretation has been proposed by Cousot et al. [2], but employs over-approximations of necessary conditions to remove only erroneous executions. The internship will also consider under-approximated sufficient conditions to keep only correct executions.

Effective under-approximations have been proposed for classic numeric domains (intervals, polyhedra) in [1] to infer sufficient conditions for the absence of array bound errors in simple programs, and later found applications in proving liveness properties [5]. These can serve as the basis for the internship, but will require extensions to ensure precision, scalability, and support for non-numeric variables such as pointers.

Expected Work

The intended work will include a theoretical side: developing abstract semantics and proving formally their soundness. It will also include a practical side: implementing the semantics and validating their benefit experimentally.

The host team is developing an open-source static analysis platform, MOPSA [7], 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. However, the framework is currently limited to forward over-approximating analyses. A first step will be to add support for backward iterations in MOPSA, and evaluate classic over-approximating backward analyses, focusing for simplicity on small C programs featuring numeric and pointer variables. Then, the intern will implement and evaluate

the novel over-approximating and/or under-approximating backward operators developed during the internship. Feedback from experiments throughout the internship will guide the design of new abstractions tailored to concrete problems in realistic settings.

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 for the implementation effort within the MOPSA platform [7].

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] A. Miné. Backward under-approximations in numeric abstract domains to automatically infer sufficient program conditions. *Science of Computer Programming (SCP)*, 33 pages, Oct. 2013.

[2] P. Cousot, R. Cousot, & F. Logozzo. Precondition Inference from Intermittent Assertions and Application to Contracts on Collections. In *12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11)*, Austin, Texas, LNCS 6538, Springer, 2011, pp. 150–168.

[3] F. Bourdoncle. Assertion-Based Debugging of Imperative Programs by Abstract Interpretation. In Proc. of 4th European Software Engineering Conf. (ESEC'93), pp. 501–516, vol. 717 of LNCS. Springer, 1993.

[4] B. Yin, L. Chen, J. Liu, J. Wang, P. Cousot Verifying Numerical Programs via Iterative Abstract Testing. In Proc. of International Static Analysis Symposium (SAS 2019), pp. 247–267, vol. 11822 of LNCS. Springer, 2019.

[5] C. Urban, S. Ueltschi, and P. Müller Abstract Interpretation of CTL Properties. In Proc. of SAS'18, 402–422, Freiburg, Germany. Springer.

[6] X. Rival. Understanding the origin of alarms in Astrée. In Proc. of SAS'05, 303–319. Springer.

[7] 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.