Resource-Aware Conservative Static Analysis

PhD position, starting in Fall 2025

Raphaël Monat & Julien Forget
Informal enquiries are welcome by email (in English or French)

1 Context

One approach aiming at reducing the number of bugs is static program analysis through the framework of abstract interpretation [1]. Contrary to dynamic analyses such as fuzzing [7], the program is not executed but its source code is analyzed. Thanks to this approach, the analysis conservatively considers all possible execution paths of the program during the analysis, ensuring the absence of false negatives. In addition, the analyses are automatic: they do not require any user interaction to complete their task and they will be completed in a guaranteed finite time. These analyses can be seen as "push-button" as no expert knowledge is required to run them. This approach has been particularly successful to certify the absence of runtime errors in critical embedded C software. Astrée [2] has proved the absence of runtime errors in software of Airbus planes.

2 Goal

The daily use of conservative static analyzers by non-experts remains a challenge. First, these tools offer a wide range of configuration options which is difficult to choose from. Each option will have a different impact on the performance-precision tradeoff of the analysis, that will also vary depending on the considered program. Mansur et al. [5], Heo et al. [3] have looked into ways to automatically choose options to attain the highest precision when analyzing a program, given a resource envelope (CPU time, memory usage); but their approaches are however limited in terms of scalability. Second, most static analyzers cannot express their progress during an analysis, which results in an unfriendly black-box behavior. The overall goal of this thesis is to address these two usability barriers. We plan to explore the following research directions:

- *Estimating the experimental complexity of analyzing a given program.* In the static program analyses we consider, we hypothesize that the complexity of analyzing a program is mostly impacted by the number of programs loops and function calls, the maximum depth of these nested constructs. We will need to confirm this hypothesis, and then focus on finding measures of the complexity of a program's analysis. We will start by considering a simplified setting focusing on a toy imperative language. In a way, this complexity measure will be an *analysis of the program analysis* itself. If needed, we will consider additional, yet realistic, hypotheses on the convergence of widening used during loop analysis.

- *Estimation of remaining analysis time.* This estimation will be performed online (i.e, during the analysis), when the full configuration of the static analyzer is fixed. Current static analyzers are often guaranteed to terminate in finite time, but do not provide any estimate of the remaining analysis time. We plan to go beyond the work of Lee et al. [4] by developing a semantic, language-independent and domain-independent progress bar that will work with fully relational numerical abstract domains.

- *Offline choice of best configuration.* We plan to develop techniques finding the configurations yielding the most precise analyses of a given program. We will start by investigating a posteriori techniques where the analyzer suggests precision improvements to remove some of the alarms it found (e.g, by suggesting to enable specific options). We will then consider a priori techniques relying on pre-analyses to find the best configuration options to analyze a program. Combined with an estimation of the cost of analyzing the program in a given configuration, we will be able to find a configuration reaching the best precision while respecting a pre-specified resource envelope. We will leverage the partial order on the precision of analyzers to guide our exploration.

These approaches will be integrated within the Mopsa static analysis platform, a state-of-the-art static analyzer we are developping, and which won the “Software Systems” track of the academic Software Verification Competition in 2024 [6]. The Software Verification Competition (SV-Comp) is an ideal testbed for the considered research: it requires each program to be analyzed as precisely as possible within 15 minutes of analysis time.

3 Developed skills

The candidate will work in the overall field of formal methods and programming language theory. They will work on conservative static analyses, and in particular some rooted in the framework of abstract interpretation. We expect the successful candidate to be motivated to improve experimental research tools such as Mopsa, which is implemented in the OCaml functional programming language.

4 Logistics

The PhD student will be part of the SyCoMoRES team of Inria Lille & CRIStAL lab, which currently hosts 4 fellow PhD students and one postdoc. Lille is a city close to Brussels, Paris & London, easily reachable by train, with a large student population and a number of cultural places & events. The lab has a very active equality and parity commission, which raises awareness on this topic to all staff (with specific events for newcomers), and provides outreach activities for high-schoolers. One of the advisors (Raphaël Monat) is an active member of this commission.

PhD students are appointed for a duration of 3 years, with a monthly gross income of €2100 (fixed by national definition). We plan to organize weekly research meetings with the PhD student. In addition, the student will be able to attend monthly meetings with other Mopsa practitioners. This research project is part of ANR JCJC RAISIN. We will hold quarterly project meetings with Sophie Cerf (member of the project), who is a researcher at Inria with expertise in control theory for software systems.

References

[1] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL 1977*, 1977. doi:10.1145/512950.512973.

[2] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Combination of abstractions in the astrée static analyzer. In *ASIAN 2006*, 2006. doi:10.1007/978-3-540-77505-8_23.

[3] Kihong Heo, Hakjoo Oh, and Hongseok Yang. Resource-aware program analysis via online abstraction coarsening. In *ICSE 2019*, 2019. doi:10.1109/ICSE.2019.00027.

[4] Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi. A progress bar for static analyzers. In *SAS 2014*, 2014. doi:10.1007/978-3-319-10936-7_12.

[5] Muhammad Numair Mansur, Benjamin Mariano, Maria Christakis, Jorge A. Navas, and Valentin Wüstholz. Automatically tailoring abstract interpretation to custom usage scenarios. In *CAV 2021*, 2021. doi:10.1007/978-3-030-81688-9_36.

[6] Raphaël Monat, Marco Milanese, Francesco Parolini, Jérôme Boillot, Abdelraouf Ouadjaout, and Antoine Miné. MOPSA-C: Improved verification for C programs, simple validation of correctness witnesses (competition contribution). In *Proc. TACAS*, 2024. doi:10.1007/978-3-031-57256-2_26.

[7] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2024. URL `https://www.fuzzingbook.org/`. Retrieved 2024-07-01 16:50:18+02:00.