PhD Position

Machine Learning for Improving Formal Verification of Code

Keywords: Machine Learning, Code Representation Learning, Graph Neural Networks, Formal Methods, Abstract Interpretation, Deductive Verification

Programming Languages: *Python*, *OCaml*, *C*

Tools: *TensorFlow*, *PyTorch*, *Frama-C*, *Eva*, *WP*

Context: CEA LIST, Software Safety and Security Lab

CEA LIST's offices are located at the heart of Université Paris-Saclay, the largest European cluster of public and private research. Within CEA LIST, the Software Safety and Security Lab has an ambitious goal: help designers, developers and validation experts ship high-confidence systems and software.

Systems in our surroundings are getting more and more complex, and we have built a reputation for efficiently using formal reasoning to demonstrate their trustworthiness through the design of methods and tools to ensure that real-world systems can comply with the highest safety and security standards. In doing so, we get to interact with the most creative people in academia and the industry, worldwide.

Our organizational structure is simple: those who pioneer new concepts are the ones who get to implement them. We are a fifty-person team, and your work will have a direct and visible impact on the state of formal verification.

Work Description

Formal verification consists in providing mathematical guarantees on the conformity of a program's behavior with respect to certain property specifications. In particular, static program verification aims at determining the properties of a program for all possible concrete inputs, without executing the program itself. A notable example is static verification of the absence of errors at runtime.

Our team develops *Frama-C* [3] (`http://frama-c.com`), a code analysis platform for C programs which provides several analyzers as plug-ins. *Frama-C* allows the user to annotate a *C* program with formal specifications, written in the *ACSL* specification language [4], and then ensure it satisfies its formal specification by relying on several static verification techniques including abstract interpretation, provided by the plug-in *Eva*, and the weakest preconditions calculus, provided by the plug-in *WP*.

Both plug-ins provide highly parametrizable techniques that may be efficiently combined, but their activation may be costly in terms of resources such as time of computation and memory footprint. It requires a thorough knowledge of *Frama-C* and its analysis techniques to choose wisely which to use in which cases. Moreover, many of these techniques are more or less based on heuristics which are usually manually conceived. These heuristics are often suboptimal, fragile, and require considerable technical knowledge and effort to be devised. As such, they do not generalize or scale well to different code bases, and need to be updated over time.

Machine learning has been recently used to perform several tasks on code [1, 7, 5], and seems particularly adapted to help in overcoming the aforementioned obstacles. On the one hand, it allows to treat the difficulty no longer as a problem of understanding the analysis tool but as a problem of understanding the forms of code adapted to the different techniques. On the other hand, the automatic learning of strategies can be repeated as the requirements and the tool evolve. These improvements can be achieved at a much finer level than a human can accomplish in a reasonable amount of time.

The ultimate goal of the PhD is to integrate machine learning approaches to the *Frama-C* static analyzers in order to improve their usability and scalability. The PhD will start by studying which heuristics already in place in *Eva* or *WP* could be automatically learned. Later, the PhD will investigate the best representations and learning algorithms for treating code, with a particular focus on maintaining as much as possible the semantic elements. A special interest will be devoted to graph embeddings [8] and graph neural networks [2, 6] which seem the best approaches for the task.

The PhD student will design and implement a machine learning pipeline for code verification strategies. Furthermore, the PhD student will integrate such strategies into *Frama-C* and evaluate their performances with respect to those already in place today.

Application

Knowledge in the following fields is required:

- Machine learning or deep learning
- *Python* programming

Knowledge in the following fields is welcome:

- *OCaml* programming (or another functional programming language)
- Deep learning frameworks (*TensorFlow* or *PyTorch*)
- Program verification (abstract interpretation, deductive verification, *etc.*)
- *C* programming language
- Formal semantics of programming languages

Availability: September 2022. Since the administrative delays for recruitment at CEA are 2 to 3 months minimum, please contact us as soon as possible.

Contacts:

- Michele Alberti (`michele.alberti@cea.fr`)
- Valentin Perrelle (`valentin.perrelle@cea.fr`)

Please join a detailed CV, and a motivation letter.

References

[1] Miltiadis Allamanis, Earl T. Barr, Premkumar T. Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Comput. Surv.*, 51(4):81:1–81:37, 2018.

[2] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[3] P. Baudin, F. Bobot, D. Bühler, L. Correnson, F. Kirchner, N. Kosmatov, A. Maroneze, V. Perrelle, V. Prevosto, J. Signoles, and N. Williams. The Dogged Pursuit of Bug-Free C Programs: The Frama-C Software Analysis Platform. *Communications of the ACM*, 2021.

[4] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. *ACSL: ANSI/ISO C Specification Language.*

[5] Victor Chibotaru, Benjamin Bichsel, Veselin Raychev, and Martin Vechev. Scalable taint specification inference with big code. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 760–774, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Chris Cummins, Zacharias Fisches, Tal Ben-Nun, Torsten Hoefler, Michael O'Boyle, and Hugh Leather. ProGraML: A Graph-based Program Representation for Data Flow Analysis and Compiler Optimizations. In *Thirty-eighth International Conference on Machine Learning (ICML)*, 2021.

[7] Veselin Raychev, Martin Vechev, and Andreas Krause. Predicting program properties from "big code". In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 111–124, New York, NY, USA, 2015. Association for Computing Machinery.

[8] Yulei Sui, Xiao Cheng, Guanqin Zhang, and Haoyu Wang. Flow2vec: Value-flow-based precise code embedding. *Proc. ACM Program. Lang.*, 4(OOPSLA), nov 2020.