Master's Internship Proposal (6 months)

Reinforcement Learning for Formal Verification of Liveness in Distributed Systems

Keywords: reinforcement learning; formal verification; model checking; distributed systems; concurrent software; testing

Internship supervisors

Inria – CNRS – Université de Rennes: **Ocan Sankur (researcher)** ocan.sankur@cnrs.fr
Mitsubishi Electric R&D Centre Europe**: Florian Faissole (researcher)**
f.faissole@fr.merce.mee.com

Location

Inria - IRISA, Université de Rennes, campus de Beaulieu 35042 Rennes, France

Overall context

Inria – IRISA is a joint academic research center in computer science of Université de Rennes, Inria, and CNRS, located in Rennes, France. The center offers an international environment; about half of the PhD students and some of the faculty are international.

Mitsubishi Electric R&D Centre Europe is the European R&D centre from the Corporate R&D organisation of Mitsubishi Electric. Situated at the heart of Europe's leading R&D community, Mitsubishi Electric R&D Centre Europe includes two entities, one located in Rennes, France and another located in Livingston, UK. We conduct R&D into next generation communication and information systems, power electronic systems and environment and energy systems.

We are interested in developing formal verification techniques for verifying the correctness of distributed algorithms and concurrent software, focusing, in this work, on liveness properties such as absence of livelocks.

Internship subject

A livelock in a concurrent software is a situation where two threads are continuously executed, but each thread requires a resource owned by the other thread, and they enter a loop where no progress is made by any of the threads. One example is two polite persons trying to eat soup with a shared spoon: they both request to eat soup, but the person holding the spoon is too polite and passes it to the other person before eating the soup. The other person has the same behavior, so they end up passing the spoon to each other and no one ever actually eats the soup.

Livelocks are notoriously difficult to detect because it is difficult to distinguish them from a trace where the threads just need to wait for a long time before making progress (say, because they are waiting for some computations to terminate). Theoretically, the difficulty can be explained by the fact that absence of livelocks is a liveness property (a proof of violation is an infinite execution), and not a safety property (for which a proof of violation could be finite).

Reinforcement Learning (RL) is set of techniques which aims at learning an optimal policy to control a system by interacting with it [6]. Lately RL has been used in test generation where the tester is seen as a policy, and the goal is to find the best tester which achieves maximal coverage, or some other criterion such as reaching a specific location in the code. This has been used in symbolic/concolic execution [5, 4], and fuzzing [1]. Deep reinforcement learning and similar techniques based on neural networks have been used to prove the termination of programs but also to establish their satisfaction of temporal properties [3].

Detailed objectives

We are interested in developing algorithms for automatically proving the absence of livelocks or detecting livelocks bugs in distributed protocols using reinforcement learning algorithms. We suggest developing deep RL algorithms to analyze maximal wait times in distributed protocols. The wait time is the number of steps a process is executed before it gains access to a resource. There are no livelocks if the wait times are always finite. The work consists in modeling this problem as an RL problem, choosing the right rewards and RL algorithms, and making sure it scales to real implementations of distributed algorithms.

Because the overall goal is formal verification, the computed neural policy must be formally verified at the end as in [3]. This can be achieved using SMT solvers or specific abstract interpretation techniques for neural networks.

Here the RL agent chooses at each step the schedule, that is, which process to execute, whether there are packet losses etc. and observes the next global state of the system. It receives a reward of 1 at each step a process waiting to access a resource is executed but without accessing that resource. Thus, the distributed protocol can be seen as a game which the RL agent must learn how to play to exhibit the worst-case behavior.

The model and RL algorithms can be chosen either to attempt to prove the absence of livelocks and compute bounds on wait times, or to detect livelock bugs. The precise direction to be taken and the weight given to RL versus formal verification in this work can be chosen according to the student's background and preferences.

The work also includes an extensive bibliographic study, the development of the above algorithms, implementation and experiments. Furthermore, an internship report will also be written by the intern as part of the training course in which this internship is included.

There is a possibility to extend this internship into a PhD thesis.

Prerequisites

- excellent theoretical background in computer science or related fields with a (ongoing) master's degree,
- good programming skills,
- knowledge (course work, internship, or master's thesis) in at least one of the two fields: reinforcement learning, and formal verification or logic, and a strong motivation to learn about the other field.

Period: anytime in 2025-2026 (possibility of flexibility, depending on schools' internships periods)

Contact : Florian Faissole (f.faissole@fr.merce.mee.com) + Ocan Sankur (ocan.sankur@cnrs.fr)

Please provide us an application letter, your CV, and transcripts of all courses taken during your master's.

References

[1] Konstantin Bottinger, Patrice Godefroid, and Rishabh Singh. Deep reinforcement fuzzing. In 2018 IEEE Security and Privacy Workshops (SPW), pages 116–122. IEEE, 2018.

[2] Malay K. Ganai. Dynamic livelock analysis of multi-threaded programs. In Shaz Qadeer and Serdar Tasiran, editors, Runtime Verification, pages 3–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[3] Mirco Giacobbe, Daniel Kroening, Abhinandan Pal, and Michael Tautschnig. Neural model checking. arXiv preprint arXiv:2410.23790, 2024.

[4] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. Pyse: Automatic worstcase test generation by reinforcement learning. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), pages 136–147, 2019.

[5] Ciprian Paduraru, Miruna Paduraru, and Alin Stefanescu. Optimizing decision making in concolic execution using reinforcement learning. In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 52–61, 2020.

[6] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning: An introduction. MIT press Cambridge, 1998.