# Research Statement

**Chris Poskitt**
School of Computing and Information Systems
Singapore Management University
Tel: (65) 6828-1376; Email: cposkitt@smu.edu.sg
*21st December 2023*

## Research Overview

My research broadly addresses the problem of **engineering correct and secure software/systems**, ensuring they can be trusted by users and deployed in contexts where reliability is paramount. Working towards this goal is rewarding for two reasons: (1) software has become ubiquitous in everyday life, and steps towards eradicating faults have a potentially larger impact now than ever before; and (2) systems are increasingly heterogeneous and complex, making it a stimulating technical challenge to find the right abstractions, tools, and analyses to support engineers in practice.

I am particularly interested in addressing this problem for **cyber-physical systems** (e.g. critical infrastructure, autonomous vehicles) that are becoming more important in our daily lives, but the complexity of which makes it difficult to directly apply traditional techniques for ensuring software quality. With my collaborators, we have co-developed approaches that adapt these techniques for such systems, whether using mutation testing to inspire the construction of anomaly detectors for critical infrastructure, domain-specific languages to model traffic laws for autonomous vehicle testing, or fuzzing to generate security benchmarks.

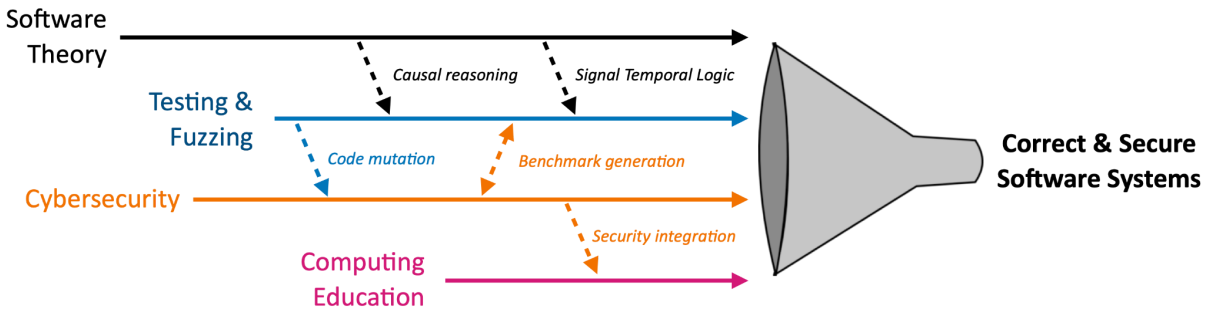
Furthermore, I am interested in how this domain knowledge can be **translated to the classroom** to help graduate cutting-edge software engineers into the workforce.

Some key overarching questions my research addresses include:

- How can traditional approaches for software quality be generalised to complex cyber-physical systems?
- How do we build defence mechanisms for critical infrastructure that effectively detect/prevent attacks?
- How do we intelligently fuzz cyber-physical systems to uncover rare (but feasible) scenarios that violate safety properties?
- How can undergraduate curricula better prepare students to engineer correct and secure code?

My research tends to **straddle both theory and practice**: I am motivated to work in new and emerging application domains (e.g. autonomous vehicles, critical infrastructure), but our solution approaches are underpinned by rigorous ideas from software theory. For example, we developed a fuzzing technique for testing autonomous driving systems such as Apollo: our approach is practical, in that it actually finds concrete ways in which certain traffic laws are violated, but it is also theoretically rigorous in that the fuzzing engine is built on top of a Signal Temporal Logic semantics of those traffic laws.

These ‘interplays’ are not just limited to software theory and testing – the predominant approach I take to solving research problems is to look for **creative and effective interplays** between the main subfields I work in:




For example, using:

- 'Mutation testing' to inspire a new way to learn anomaly detectors for cyber-physical systems (CPS);
- Fuzzers to generate benchmarks for testing CPS defence mechanisms;
- Causal reasoning to help fuzzers distinguish between classes of 'equivalent' tests;
- Security scanners to raise insecurity awareness among undergraduate students.

The techniques and domains I work with are diverse, but I find that this diversity leads to exciting new ideas that all chip away at the grand challenge of making our systems safer and more secure.

## Research Highlights

In the following, I summarise some of the key results from the projects I have worked on.

### *Defending Cyber-Physical Systems*

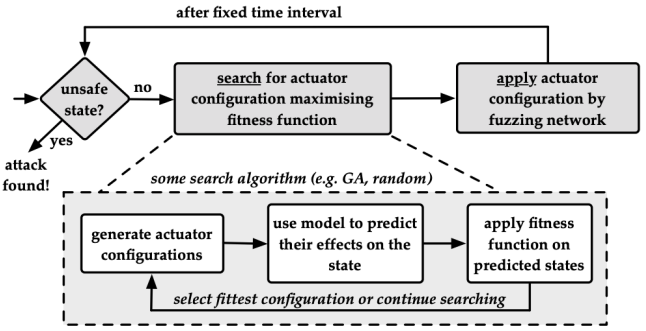
Cyber-Physical Systems (CPSs) are characterised by a deep integration of software and physical processes. These complex systems are commonly used to automate aspects of public infrastructure (e.g. water treatment, smart grids) and thus have become prime targets for cyberattackers. Identifying whether a CPS has been compromised based on its behaviour, however, is very challenging, given the tight integration of algorithmic control in the "cyber" part with continuous behaviour in the "physical" part. While the software components (e.g. PLCs) are often simple when viewed in isolation, this simplicity betrays the typical complexity of a CPS when taken as a whole. To address this, we have been developing a number of defence mechanisms that attempt to intelligently overcome this complexity.

In our S&P'18 paper [3], taking inspiration from 'mutation testing', we systematically mutated PLC code to generate traces of abnormal physical behaviour, using this to learn **precise anomaly detectors** based on machine learning models. In our ESEC/FSE'21 paper [2], we focused on the integrity of PLCs themselves, developing an approach based on privacy-preserving black box models that could be used to **attest the input/output behaviour** of their programs. Our approach was general, and resulted in near-100% accuracy at detecting effective code modification attacks. Finally, in our IEEE TDSC paper [1], we explored the limitations of building defence mechanisms from data. In a detailed case study

on the SWaT water treatment testbed[1], we found that anomaly detectors trained solely on historical data were not enough, and could be exploited by adversaries with knowledge of the plant design (e.g. backup pumps). We advocate for a hybrid approach in which anomaly detectors are constructed using a combination of data as well 'invariants' extracted from the plant's design documents.

1. *Mitigating Adversarial Attacks on Data-Driven Invariant Checkers for Cyber-Physical Systems*, R.R. Maiti, C.H. Yoong, V.R. Palleti, A. Silva, and C.M. Poskitt. **IEEE TDSC'23**
2. *Code Integrity Attestation for PLCs using Black Box Neural Network Predictions*, Y. Chen, C.M. Poskitt, and J. Sun. **ESEC/FSE'21**
3. *Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System*, Y. Chen, C.M. Poskitt, and J. Sun. **S&P'18**

### *Testing Critical Infrastructure*

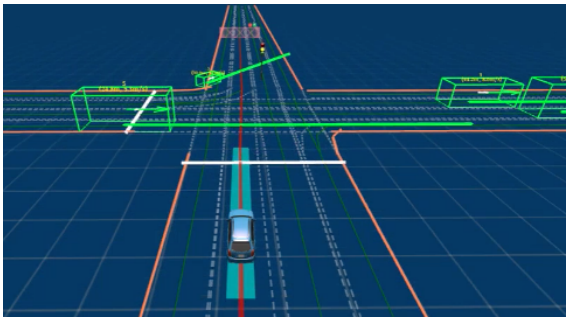



A problem we repeatedly encountered while developing defence mechanisms for critical infrastructure was a **lack of benchmarks** to evaluate them against. This spurred a new line of work in which we tried to apply ideas from black-box fuzzing to systematically generate test suites of network attacks. In our ASE'19 paper [6], we designed an approach that combined metaheuristic search with predictive machine learning models (learnt by observing sensor readings in the system's logs) to find actuator manipulations that pushed the system towards **anomalous behaviours**. In our ISSTA'20 paper [5], we adapted the techniques to work at the network packet-level – flipping bits in the payloads – and used an 'active learning' approach to predict the effects so as to minimise the amount of training data needed. Our work was able to automatically derive tests for SWaT covering 27 different unsafe states, including six that were not covered by an existing expert-crafted benchmark. Finally, in our ICSE'23 paper [4] we developed algorithms that help these fuzzers ensure that they uncover causally different tests, thus ensuring test suite diversity.

4. *Finding Causally Different Tests for an Industrial Control System,* Christopher M. Poskitt, Yuqi Chen, Jun Sun, and Yu Jiang. **ICSE'23**
5. *Active Fuzzing for Testing and Securing Cyber-Physical Systems*, Y. Chen, B. Xuan, C.M. Poskitt, J. Sun, and F. Zhang. **ISSTA'20**
6. *Learning-Guided Network Fuzzing for Testing Cyber-Physical System Defences*, Y. Chen, C.M. Poskitt, J. Sun, S. Adepu, and F. Zhang. **ASE'19**

[1] https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/

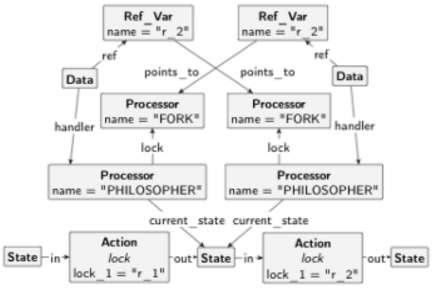
### *Fuzzing Autonomous Vehicles*



Autonomous Driving Systems (ADSs) such as Baidu Apollo must be comprehensively tested before they can be deployed on real roads. High-fidelity simulators exist for this purpose, and allow for them to be tested in scenarios that are difficult to recreate in the real world. Previous ADS testing work, however, has largely focused on weak test oracles, e.g. “did the car collide?” or “did the car reach the destination?”. On the road, however, the quality of the journey is just as important as getting to the destination in one piece (skipping every traffic light, for example, is unacceptable, even if it can be done without crashing). In our ASE’22 paper [7], we addressed this problem by developing a domain-specific language for **specifying traffic laws** based on Signal Temporal Logic (STL). We gave the underlying STL formalisation a quantitative semantics that our fuzzing engine then used to search for scenarios that came ‘closer’ to violating our traffic laws. As our case study, we formalised the traffic laws of China, and were able to cause Apollo to **violate 14 of them**.

7. *LawBreaker: An Approach for Specifying Traffic Laws and Fuzzing Autonomous Vehicles*, Y. Sun, C.M. Poskitt, J. Sun, Y. Chen, Z. Yang. **ASE'22**

### *Reasoning about Graph-like Structures*



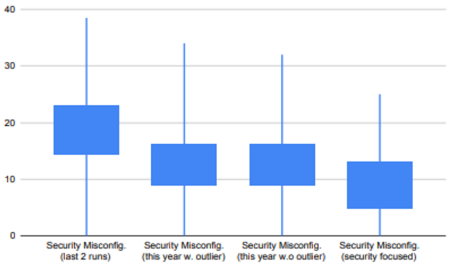

As part of my PhD, I studied techniques for proving the correctness of problems modelled as **graphs and algebraic graph transformations** (i.e., “graph programs”), and still maintain a thread of research in this community. For example, in recent work [8,9], I adapted Peter O’Hearn’s “incorrectness logic” to this setting. Unlike traditional program logics (which prove facts about *all* possible executions), an incorrectness logic proves the *presence* of certain executions, which could be particularly important for graph programs given the large amount of nondeterminism.

In some earlier work [10], I used a graph-based modelling tool (“GROOVE”) to model the semantics of SCOOP, an extension of the Eiffel object-oriented language that provided concurrency via an actor-like message-passing model. By abstracting the semantics to graphs, we were able to **uncover a discrepancy** between two versions of the SCOOP semantics that had implications for the behaviour of real programs.

8. *Monadic Second-Order Incorrectness Logic for GP 2*, Christopher M. Poskitt and Detlef Plump. **JLAMP'23**
9. *Incorrectness Logic for Graph Programs*, C.M. Poskitt. **ICGT'21**
10. *A Semantics Comparison Workbench for a Concurrent, Asynchronous, Distributed Programming Language,* C. Corrodi, A. Heußner, and C.M. Poskitt. **FAOC’18**

### *Software Engineering Education*

I occasionally publish in the computing education community to disseminate any new pedagogical interventions that we feel may be useful for other practitioners (e.g. [11], [12]). Most recently, our ITiCSE'22 paper [11] addressed our observation that many computing curricula provide limited exposure to topics in computer security, risking graduates with very little software security awareness. Our paper advocates a **"security integration" approach** to solve this, in which security is used as a unifying topic across non-security courses. We assessed one such example in a web development course, integrating the use of security scanners to make students aware of the risks of, e.g. blindly trusting user input. Upon scanning students' project code ourselves before and after this modest intervention, we observed notably fewer flaws.




*Impact of our intervention on reducing security misconfiguration errors*


11. *XSS for the Masses: Integrating Security in a Web Programming Course using a Security Scanner*, L.K. Shar, C.M. Poskitt, K.J. Shim, L.Y.L. Wong. **ITiCSE'22**
12. *Securing Bring-Your-Own-Device (BYOD) Programming Exams*, O. Kurniawan, N.T.S. Lee, and C.M. Poskitt. **SIGCSE'20**