PhD projects in the Department of Informatics, AY 25-26 — Systems (software engineering, programming)

The PhD projects listed below will be considered for 2025/26 studentships available in the Department of Informatics to start on 1 October 2025 or later during the 2025/26 academic year.

Please note that this list is not exhaustive and potential applicants can alternatively identify and contact appropriate supervisors outlining their background and research interests or proposing their own project ideas.

Each project is designated for a single student, meaning it can only be assigned to one successful applicant. Some projects come with allocated studentships, while others are eligible for "unallocated" studentships. Applicants who apply for projects with allocated studentships and are selected will be offered a full studentship. In the project list, these are marked as "studentship allocated." Applicants chosen for other projects will compete for the unallocated studentships.

We welcome applications from students who have secured, or are applying for, or plan to apply for other funding (within other studentships internal to the university or external schemes) and from self-funded students. See also this list of funding opportunities available at King's for post-graduate research in Computer Science.

PhD projects

- Debugging and runtime visualisation in a frame-based system (studentship allocated)
- Nominal Specification and Verification Environments (studentship allocated)
- Reliable source-level debugging of optimised code by stateful approaches
- Visual live programming in scientific computing and similar domains
- Brzozowski Derivatives for Fast Regular Expression Matching
- Leveraging Language Models for Contextual Vulnerability Identification
- Validation and Testing of GPUs
- Software sustainability analysis and improvement
- Verified Complexity Theory: Probabilistic Computation and Verified Post-Quantum Cryptography
- Software Verification and Nominal Dependent Type Theory
- Privacy in the Internet of Things

Debugging and runtime visualisation in a frame-based system

Supervisor: Prof Michael Kolling / Dr Neil Brown

Areas: Human-centred computing (human-computer interaction), Computing Education, Systems (software engineering, programming)

Project Description

Frame-based editing is a novel program manipulation paradigm that combines advantages from both text-based and block-based editors. It has been implemented in the Stride language [1], and in the online Strype system [2]. The visualisation possibilities embedded in frame-based editors provide opportunities for improved debugging and runtime visualisation functionality, beyond what is available in typical text-based or block-based systems. The goal of this project is to design and implement novel debugging/visualisation functionality in a frame-based system. Candidates for this project must have exceptional programming skills in multiple languages, a deep understanding of object-orientation, interest in HCI and in programming education.

References

[1] https://stride-lang.net [2] https://strype.org

Nominal Specification and Verification Environments

Supervisor: Maribel Fernandez

Areas: Foundations of computing (algorithms, computational complexity), Systems (software engineering, programming)

Project Description

Software verification techniques have been successfully used to prove correctness of low-level programs, but verification of high-level programming languages is challenging: it requires reasoning about name binding (e.g., visible/hidden channel names, scoping rules defining local and global variable names, parameter passing and substitution of values for variables). A standard approach to deal with name binding in verification tasks is to replace names with numerical codes (de Bruijn indices). While this avoids some of the difficulties of reasoning about names in programs, conducting a formalisation in de Bruijn style is labour-intensive and imposes a significant overhead to comprehending and reusing proofs. Nominal techniques offer an alternative, user-friendly approach, which does not require to replace names with codes. We aim to apply novel nominal techniques to simplify the handling of names and binders in programming languages and verification tasks (e.g., verification of blockchain languages). For this, we aim to develop a core nominal calculus and use it as a basis to enrich with nominal features two successful verification frameworks: Maude and K.

Reliable source-level debugging of optimised code by stateful approaches

Supervisor: Stephen Kell

Areas: Systems (software engineering, programming)

Project Description

Debugging optimised code, as generated by ahead-of-time compilers such as GCC and LLVM, has long been poorly supported by existing tools and infrastructure. A recurring semantic bottleneck is that there is no comprehensive way to retain state in the debugger -- for example, when a local variable is eliminated by the compiler and cannot be reconstructed computationally from other program state. Instead of such a comprehensive approach, there are several obscure and highly partial mechanisms which do this "by the back door" but are poorly specified and rarely well-implemented. These include "location views" and "entry values". There are also other debugging tools, such as rr or UndoDB, which maintain large amounts of additional state, at a cost of sometimes large slowdown. This thesis project will investigate how to evolve our current execution infrastructure in commodity debuggers, operating systems and language implementations to provide reliable source-level debugging using stateful approaches. For example, it may devise an overhaul to DWARF debugging information that provides a general and elegant approach to debugger-side state. It may develop efficient hybrids that bridge the gap between powerful but state-heavy recording techniques and the lightweight convenience of attaching the debugger only when needed. And it is likely to explore, to some extent, the usability issues inherent in more stateful tools that add a fresh layer of illusion over the code that is actually running. It builds on our EPSRC-funded prior work and DARPA-funded ongoing work, so will not be starting from scratch.

Visual live programming in scientific computing and similar domains

Supervisor: Stephen Kell

Areas: Systems (software engineering, programming), Human-centred computing (human-computer interaction)

Project Description

Currently, working interactively with data means either using a pre-built application offering a fixed interface, which is visual and interactive but offers limited programmability, or using custom workflows built by programming/scripting or command-line wizardry, which are flexible but technically demanding and far less visual and interactive. Computational notebooks like Jupyter are in some senses a third way, being somewhat visual, and have proven approachable by those seeking to learn programming 'on the job'. However, they currently suffer many usability and reproducibility issues, and still present a 'walled garden' environment with poor integration into the surrounding system. This PhD is about ways to combine the interactivity of applications and the flexibility of command lines, possibly by designing a notebook system that works differently than Jupyter et al. We observe that crude operating system (OS) interfaces are the bottleneck to interoperable, visual programming, since they lack a rich data model on which to build visualisation as a system-wide service; this is what leads to smaller-scope walled-garden approaches. Recent work adding run-time type information to native code has shown that such limitations can be overcome without defining an entirely new platform. This project will pursue similar approaches encompassing file data and graphical user interface elements. The objective is to demonstrate a graphical workspace that is highly compatible and interoperable, dealing in files of existing formats, but can support working visually and programmatically via a palette of small, composable, user-tailorable graphical tools. Target audiences include computational scientists, data scientists, digital artists and the like. The project requires systems programming skills and an interest in human-- computer interaction topics.

Brzozowski Derivatives for Fast Regular Expression Matching

Supervisor: Christian Urban

Areas: Foundations of computing (algorithms, computational complexity), Systems (software engineering, programming)

Project Description

If you want to connect a computer directly to the Internet, it must be immediately hardened against outside attacks. The current technology for this is to use regular expressions in order to automatically scan all incoming network traffic for signs when a computer is under attack and if found, to take appropriate counter-actions. Unfortunately, algorithms for regular expression matching are sometimes slow and inefficient. My research is about making breakthroughs in this area. The results are also applicable to DNA searches, security, compilers and algorithms. In addition I am interested in supervising topics in programming languages, formal methods, functional programming, Rust and theorem provers. Skills in these areas are in high demand in both industry and academia: my former PhD students work now at ARM and Imperial College London.

Leveraging Language Models for Contextual Vulnerability Identification

Supervisor: Maher Salem

Areas: Artificial Intelligence (symbolic AI, logic, etc.), Machine learning / Deep learning, Cybersecurity, Systems (software engineering, programming)

Project Description

As software systems grow increasingly complex, the need for effective vulnerability detection methods becomes paramount. Traditional static analysis tools often struggle to identify context-specific vulnerabilities due to their reliance on predefined patterns and rules. This research proposes leveraging advanced language models, such as transformers, to enhance the identification of vulnerabilities in software code by understanding its context. The central idea of this topic is to explore how large language models (LLMs) can be trained to analyze code not merely as isolated snippets but as part of a larger context. By fine-tuning LLMs on extensive datasets that include both vulnerable and secure code, the model can learn to recognize subtle patterns and interactions that indicate potential vulnerabilities. This approach aims to move beyond conventional methods by incorporating an understanding of how different code components interact with each other, thereby improving detection accuracy. The research will involve several key phases. First, a comprehensive dataset will be curated, containing various programming languages and a range of vulnerability types, such as SQL injection, cross-site scripting, and buffer overflows. This dataset will serve as the foundation for training the language models. Next, the study will focus on developing a framework that integrates the LLMs into an existing vulnerability detection pipeline, allowing for real-time analysis and feedback during the software development lifecycle. Furthermore, the research will explore the effectiveness of different model architectures and training techniques, including transfer learning and few-shot learning, to optimize performance. By evaluating the models against established benchmarks and real-world codebases, the study aims to quantify improvements in vulnerability detection rates compared to traditional static analysis tools. Another important aspect of this research is the interpretability of the model's predictions. It is crucial for developers to understand why a particular piece of code was flagged as potentially vulnerable. Therefore, the study will investigate methods to enhance the transparency of LLMs, providing explanations that can guide developers in addressing identified vulnerabilities. Ultimately, this research seeks to contribute to the field of cybersecurity by providing a novel approach to vulnerability detection that leverages the capabilities of modern AI. By harnessing the contextual understanding of language models, the goal is to create more robust and intelligent tools that can significantly enhance software security, helping developers proactively identify and mitigate vulnerabilities before they can be exploited.

References

DOI: 10.1145/3460318.3464820
DOI: 10.1145/3404835.3462831
DOI: 10.1145/3397670
DOI: 10.1109/TSE.2020.2986860

Validation and Testing of GPUs

Supervisor: Hector Menendez, Maribel Fernandez and Karine Even-Mendoza

Areas: Foundations of computing (algorithms, computational complexity), Systems (software engineering, programming)

Project Description

Considering the rise of graphics and machine learning libraries, Graphics Processing Units (GPUs) are becoming increasingly relevant, especially for solutions that aim to train large machine learning models. GPUs can significantly reduce the training and inference time of these models by employing multiple graphics kernels. Although these systems present a significant promise for advancing AI and other dependent systems, their reliance on floating point operations limits how they can be validated and tested. The literature shows various contributions to the process of testing GPUs, with the most notable being the work of Alastair Donaldson. He initially developed the GPUVerifier (Betts et al., 2012) and later focused on the problems of concurrency (Alglave et al, 2015) and automatic testing of system rendering (Donaldson et al., 2017). Considering current rendering technologies that focus on universal platforms, such as browsers, through WebGPU technologies (Bernhard et al. 2024), it is also important to evaluate the quality of these specific APIs to ensure that GPU resources are properly utilized and optimized. The main goal of this PhD is to define different verification and testing strategies to validate new GPU technologies. To achieve this, the PhD will start by investigating the work of Alastair Donaldson and others on GPU evaluation, including the GPUVerifier, WebGPU evaluation system, GraphicsFuzz, and other strategies adapted for GPU validation. It may also be beneficial to explore how the testing and validation methods apply to similar accelerators, such as Tensor Processing Units (TPUs) and Neural Processing Units (NPUs). These hardware accelerators are becoming increasingly common in AI. Investigating how these methods translate to TPUs and NPUs could provide additional opportunities for optimizing and ensuring the robustness of AI systems. The thesis will be divided into: Part 1: Literature Review and Identification of Gaps in the Literature: Identify gaps related to the problem of sandboxing current browser APIs and improving the security of systems against potential adversarial attacks. Part 2: Verification: Design different verification strategies based on formal verification methods. These methods will work within a determined context to bound the system's complexity and manage a deeper analysis for validation. Part 3: Testing and Fuzzing: Redesign testing methods to identify specific use cases that expose vulnerabilities in GPU-based systems. These methods focus on the specific technological context, particularly concurrency and floating-point. This PhD will contribute to the development of robust verification and testing strategies, ensuring the efficient and secure use of GPUs in modern computing applications.

References

Alglave, J., Batty, M., Donaldson, A. F., Gopalakrishnan, G., Ketema, J., Poetzl, D., ... & Wickerson, J. (2015). GPU concurrency: Weak behaviours and programming assumptions. ACM SIGARCH Computer Architecture News, 43(1), 577-591.

Bernhard, L., Schiller, N., Schloegel, M., Bars, N., & Holz, T. (2024). DarthShader: Fuzzing WebGPU Shader Translators & Compilers. arXiv preprint arXiv:2409.01824.

Betts, A., Chong, N., Donaldson, A., Qadeer, S., & Thomson, P. (2012, October). GPUVerify: a verifier for GPU kernels. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications (pp. 113-132).

Donaldson, A. F., Evrard, H., Lascu, A., & Thomson, P. (2017). Automated testing of graphics shader compilers. Proceedings of the ACM on Programming Languages, 1(OOPSLA), 1-29.

Software sustainability analysis and improvement

Supervisor: Kevin Lano

Areas: Systems (software engineering, programming), Machine learning / Deep learning, Artificial Intelligence (symbolic AI, logic, etc.)

Project Description

The project would consider techniques for analysing software sustainability (in the sense of energy use and energy efficiency) using rule-based analysis and refactoring, or by the use of deep learning techniques such as LLMs to identify energy use flaws and potential refactorings. It would be particularly useful to consider analysis and refactorings at the specification or design levels of a software system, in order that programming-language independent advice and improvements can be made. There is the potential for industrial collaboration in this area.

References

(Lano et al., 2024a) K. Lano et al., "Software modelling for sustainable software engineering", STAF 2024.
(Lano et al., 2024b) K. Lano et al., "Design Patterns for Software Sustainability", PLoP 2024.

Verified Complexity Theory: Probabilistic Computation and Verified Post-Quantum Cryptography

Supervisor: Mohammad Abdulaziz

Areas: Foundations of computing (algorithms, computational complexity), Systems (software engineering, programming)

Project Description

In 1971, Cook [9] formulated the question P vs. NP for the first time. To this day, this question has not been solved, but could be argued to have been a steady motivator in the field of complexity theory. Many other complexity classes have been defined and studied, like PSPACE, EXP, their nondeterministic variants NPSPACE, NEXP and the polynomial hierarchy PH. The introduction of probabilistic computational models led to further classes like BPP, PP and IP. The relationship between these classes is still being investigated, but there have been some conditional and unconditional results, like PSPACE = NPSPACE [?] and IP = PSPACE [20]. In this project, I propose the use of Interactive Theorem Provers (ITPs) (aka proof assistants) to the area of computational complexity. ITPs are mechanised mathematical systems, i.e. systems which can be used to develop machine-checkable (aka formal) proofs. To prove a theorem in an ITP, the user provides high-level steps of a proof, and the ITP fills in the details, at the level of axioms, culminating in a formal proof. The fact that ITPs can use human expertise is a source of their strength in many applications, e.g. when the properties to prove are undecidable. ITPs have been used to formally prove results from a large number of areas, ranging from the correctness of realistic software systems [16, 14, 13] to formally prove results from more theoretical areas of computer science, especially efficient algorithms and combinatorial optimisation, e.g. algorithms for matching [3, 2], minimum-cost flows [1], maximum flows [15], the simplex algorithm to optimally solve linear programs [17], and geometric algorithms [19]. Furthermore, ITPs are currently attracting the attention of mathematicians and are being used to formally prove many important results in pure mathematics [8, 4, 10]. However, despite all these applications, ITPs have not been used to formally prove anything but the rather elementary results from complexity theory. Most relevant to this proposal, ITPs were recently used to formally prove the Cook-Levin-Theorem, which states that SAT is NP-complete, by Gaher and Kunze [11] in the ITP Coq and Blabach in the ITP Isabelle/HOL [6]. A notable complication when using ITPs to formally prove results from complexity theory is the difficulty of formal reasoning about computational models. Indeed, such reasoning requires the development of a reasoning infrastruc- ture, including formally proving many theorems and developing methods to automate proofs. For instance, Balbach used Turing machines as their underlying computation model, which Gaher and Kunze used Lambda calculus. In this project, you will develop a framework for formal reasoning about probablistic computation models, e.g. Turing machines with a source of random bits. One application area of such a reasoning framework is verifying randomised complexity theoretic reductions. A notable example is the NP-Hardness proof of the problem of Learning with Errors by Atjai [5], which is used to prove the security of CRYSTALS-KYBER [7], one of the few NIST approved post-quantum cryptographic algorithms. An outcome could thus be an implementation of CRYSTAL-KYBER that is formally verified to be secure. Methods that you will need to master include developing automation for the ITP Isabelle/HOL [18] and applying quantitative program logics [12, 21].

References

[1] Mohammad Abdulaziz and Thomas Ammer. A Formal Analysis of Capacity Scaling Algorithms for Minimum Cost Flows. In The 15th International Conference on Interactive Theorem Proving (ITP 2024).
[2] Mohammad Abdulaziz and Christoph Madlener. A Formal Analysis of RANKING. In The 14th Conference on Interactive Theorem Proving (ITP).
[3] Mohammad Abdulaziz, Kurt Mehlhorn, and Tobias Nipkow. Trustworthy graph algorithms (invited paper). In The 44th International Symposium on Mathematical Foundations of Computer Science (MFCS).
[4] Mohammad Abdulaziz and Lawrence C. Paulson. An Isabelle/HOL Formalisation of Green's Theorem.
[5] Miklos Ajtai. The shortest vector problem in L 2 is NP -hard for randomized reductions (extended abstract). In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC '98.
[6] Frank J. Balbach. The Cook-Levin theorem.
[7] Joppe W. Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehle. CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM.
[8] Davide Castelvecchi. Mathematicians welcome computer-assisted proof in 'grand unification' theory.
[9] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In 3rd Annual ACM Symposium on Theory of Computing (STOC).
[10] prefix=van useprefix=false family=Doorn, given=Floris, Patrick Massot, and Oliver Nash. Formalising the h- Principle and Sphere

Eversion. In Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023.
[11] Lennard Gaher and Fabian Kunze. Mechanising Complexity Theory: The Cook-Levin Theorem in Coq. In 12th International Conference on Interactive Theorem Proving (ITP).
[12] Maximilian Paul Louis Haslbeck. Verified Quantitative Analysis of Imperative Algorithms (Verifizierte Quanti- tative Analyse von Imperativen Algorithmen).
[13] Sudeep Kanav, Peter Lammich, and Andrei Popescu. A Conference Management System with Verified Document Confidentiality. In The 26th International Conference on Computer Aided Verification (CAV).
[14] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In 22nd ACM Symposium on Operating Systems Principles 2009 (SOSP).
[15] Peter Lammich and S. Reza Sefidgar. Formalizing Network Flow Algorithms: A Refinement Approach in Isabelle/HOL.
[16] Xavier Leroy. Formal verification of a realistic compiler.
[17] Filip Maric, Mirko Spasic, and Rene Thiemann. An Incremental Simplex Algorithm with Unsatisfiable Core Generation.
[18] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. Isabelle/HOL - A Proof Assistant for Higher-Order Logic.
[19] Martin Rau and Tobias Nipkow. Verification of Closest Pair of Points Algorithms. In Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II.
[20] Adi Shamir. IP = PSPACE.
[21] Joseph Tassarotti and Robert Harper. A separation logic for concurrent randomized programs.

Software Verification and Nominal Dependent Type Theory

Supervisor: Maribel Fernandez

Areas: Systems (software engineering, programming), Foundations of computing (algorithms, computational complexity)

Project Description

Dependent Type Theory is a mathematical tool to write formal specifications and prove the correctness of software implementations. The proof assistants used to certify the correctness of programs (such as Coq), are based on dependently typed higher-order abstract syntax. The goal of this project is to explore alternative foundations for proof assistants using nominal techniques. The nominal approach has roots in set theory and has been successfully used to specify programming languages. This project will focus on the combination of dependent types and nominal syntax and explore the connections between the nominal approach and the higher-order syntax approach used in current proof assistants.

References

Typed Nominal Rewriting, Elliot Fairweather and Maribel Fernandez, ACM Transactions on Computational Logic, vol.19, number 1, 2018.

Privacy in the Internet of Things

Supervisor: Maribel Fernandez

Areas: Cybersecurity, Systems (software engineering, programming)

Project Description

Data Collection policies are used to restrict the kind of data transmitted by devices in the Internet of Things (e.g., health trackers, smart electricity meters, etc.) according to the privacy preferences of the user. The goal of this project is to develop cloud/IoT architectures with integrated data collection and data sharing models, to allow users to specify their own policies and trade data for services. For this, new data collection and data sharing models will have to be developed, with appropriate user interfaces, policy languages, and policy enforcement mechanisms. An important aspect of the project is the development of policy recommendation systems that can suggest/create policies based on user profiles, making privacy an integral part of the system (according to the privacy-by-design" IoT paradigm).

References

A Privacy-Preserving Architecture and Data-Sharing Model for Cloud-IoT Applications, Maribel Fernandez; Jenjira Jaimunk; Bhavani Thuraisingham IEEE Transactions on Dependable and Secure Computing, vol. 20, no. 4, pp. 3495-3507, 1 July-Aug. 2023, doi: 10.1109/TDSC.2022.3204720.