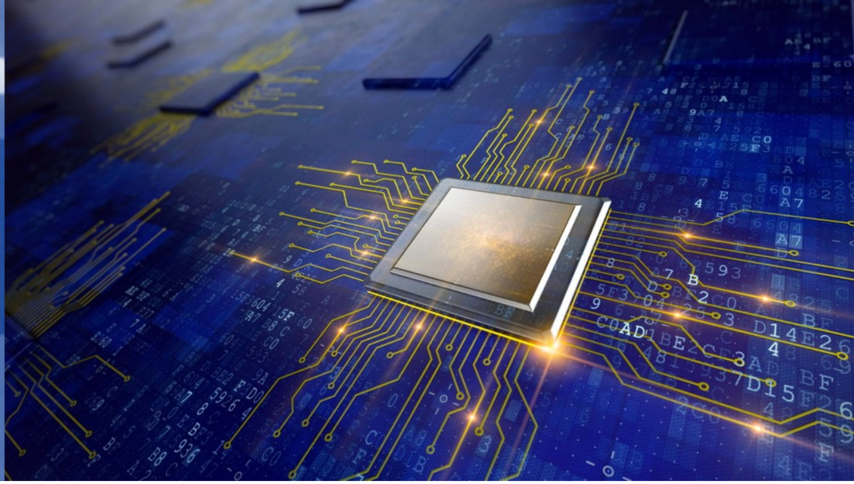


Research Institute in Verified Trustworthy Software Systems

UK's second research institute in cyber-security

Annual Report 2020/2021

EPSRC
Engineering and Physical Sciences
Research Council

National Cyber
Security Centre
a part of GCHQ

Imperial College
London

FOREWORD

Philippa Gardner, Director of VeTSS

The Research Institute in Verified Trustworthy Software Systems (VeTSS) is the UK's second Academic Research Institute in cyber security, funded by the Engineering and Physical Sciences Research Council (EPSRC) for five years, from April 2017. The purpose of VeTSS is to bring together and support world-class UK academics, industrialists and government employees, unified by a common interest in software analysis, testing and verification. VeTSS stands at the forefront of research developments in fundamental theories and industrial-strength tools, targeting real-world applications. It succeeds the previous three-year Research Institute in Automated Program Analysis and Verification, funded by EPSRC and GCHQ.

The National Cyber Security Centre (NCSC) anticipates giving approximately £2.5 million to VeTSS over five years to support academic research projects in software analysis, testing and verification. This annual report provides a description of the projects funded over the first four years, from 2017 to 2020. It demonstrates the deep connection between the VeTSS academic research, the standard bodies and industry. For example, Batty's two VeTSS projects on understanding the concurrent behaviour of C++ have directly influenced the ISO C++ standard, and Livshits's and Donaldson's VeTSS project relates to an academic start-up of Donaldson that was bought by Google in 2018. This report also describes how VeTSS funding has led directly to a total of more than £10M of further funding from EPSRC, the EU, government bodies, and industry. For example, Yoshida's work on her VeTSS project played a crucial part in her obtaining a £1.46M UKRI Established Career Fellowship, 2020-2025. Furthermore, the interaction between NCSC and VeTSS has led to an additional invited call by NCSC on "Verified High Assurance Software" in 2019.

We have held a number of events since the start of VeTSS, including our main annual workshop "Formal Methods and Tools for Security" at Microsoft Cambridge in September 2017 and 2018. This workshop was renamed into "Verified Software" and held at the Isaac Newton Institute in Cambridge in September, 2019 in preparation of the 2020 Newton Institute programme on "Verified Software" (delayed until 2022), organised by de Moura (Microsoft Redmond), Farzan (Toronto), Hoare (Microsoft), Gardner (Imperial), Larsen (Aalborg), Leroy (Inria Paris), McMillan (Microsoft Redmond), O'Hearn (Facebook and UCL), Sewell (Cambridge), Shankar (SRI, California, lead organiser) and Vardi (Rice). This meeting will bring international academics and industrialists to the UK for six weeks, laying the groundwork for the next generation of verification grand challenges. In anticipation of that meeting, two virtual workshops on "Verified Software: from Theory to Practice" will be held by the Newton Institute in the first half of 2021.

I hope that you will find this annual report of interest.

Professor Philippa Gardner
Director of VeTSS

VeTSS 2017/2018

Mechanising the Metatheory of SQL with Nulls

James Cheney
University of Edinburgh

Automated Testing for Web Browsers

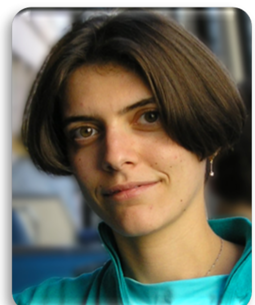
Benjamin Livshits
Imperial College London

PrideMM Web Interface

Mark Batty
University of Kent

Verifying Efficient Libraries in CakeML

Scott Owens
University of Kent



Supervectorizer

Greta Yorsh
Queen Mary Univ. of London

EASTEND: Efficient Automatic Security Testing for Dynamic Languages

Johannes Kinder
Royal Holloway Univ. of London

Automated Reasoning with Fine-Grained Concurrent Collections

Ilya Sergey
University College London

Mechanised Assume-Guarantee Reasoning for Control Law Diagrams via Circus

Jim Woodcock
University of York

MECHANISING THE METATHEORY OF SQL WITH NULLS

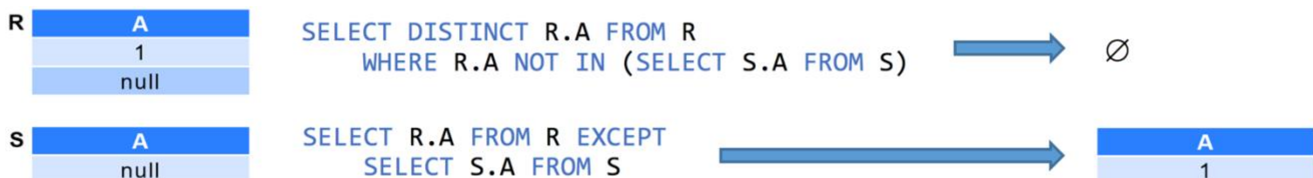
JAMES CHENEY WILMER RICCIOTTI

- SQL is the standard query language used by the multi-billion-dollar relational database industry
- SQL semantics is notoriously subtle, written in natural language, and inconsistent across implementations
- Previous attempts to verify SQL transformations have ignored widely-used features, such as null values
- We present the first mechanised semantics that models these features, making it possible to formally verify that real query optimisers are correct for real-world databases.

The Structured Query Language, SQL, is by far the most common language used by relational databases, which are the basis of a multi-billion-dollar industry. The SQL standard is described by a large and comprehensive definition (ISO/IEC 9075:2016), based on natural language rather than a formal specification; due to the lack of an agreed-on formal semantics, commercial SQL implementations interpret the standard in different ways, so that the same query can yield different results on the same input data depending on the SQL system it is run on.

SQL systems first run a *query optimiser* which applies a set of rewrite rules to obtain an equivalent query that can be processed more efficiently. However, due to the lack of a well-understood formal semantics, it is very difficult to validate the soundness of such rewrite rules, and incorrect implementations are known in the literature. Bugs in query optimisers could lead to corruption or errors in critical data.

Among SQL's features, its ability to deal with incomplete information, in the form of *null values*, accounts for a great deal of semantic complexity. To express uncertainty, logical predicates on tuples containing null values employ three truth values: *true*, *false*, and *unknown*. As a consequence, queries equivalent in the absence of null values can produce different results when applied to tables with incomplete data, as illustrated in the diagram below.



Although when this project was carried out there were some previous formalizations of SQL or relational query languages, all of them ignored null values, so they “proved” query equivalences that are unsound in the presence of these features. Our project built on a recent (on-paper) formal semantics for SQL with nulls by Guagliardo and Libkin, providing the validation of key metatheoretic properties in the Coq proof assistant. We view this as a first step towards a future in which query optimizers are certified. Our development, which can be publicly accessed at its GitHub repository (https://github.com/wricciot/nullSQL), provided us with a reliable reference which has guided us in our further work on querying databases in functional programming languages, published in leading conferences on programming languages and formal methods.

PUBLICATIONS. [1] Strongly Normalizing Higher-Order Relational Queries. Wilmer Ricciotti and James Cheney. FSCD 2020. [2] Query Lifting: Language-integrated query for heterogeneous nested collections. Wilmer Ricciotti and James Cheney. ESOP 2021.

RELATED GRANTS. Dr James Cheney, ERC Consolidator Grant: “Skye: Bridging theory and practice for scientific data curation”, 2016-2021, £1.75M.

Impact Statement. “Database queries and query languages are widely used in industry, yet their implementations and optimisation rules are error-prone due to complications, such as the semantics of nulls. This can easily lead to subtle bugs in relational database engines or incorrect queries, and work on formalising the semantics of existing query languages, including the real-world semantics of nulls, is very important and likely to have a tangible impact on making systems more reliable. For example, optimisation rules proposed in Kim’s seminal work on query un-nesting contained the famous count bug, which led to incorrect query results in the presence of null values and could have been prevented if formal verification techniques were used.”
– Matthias Brantner, Oracle –

Automated Testing for Web Browsers

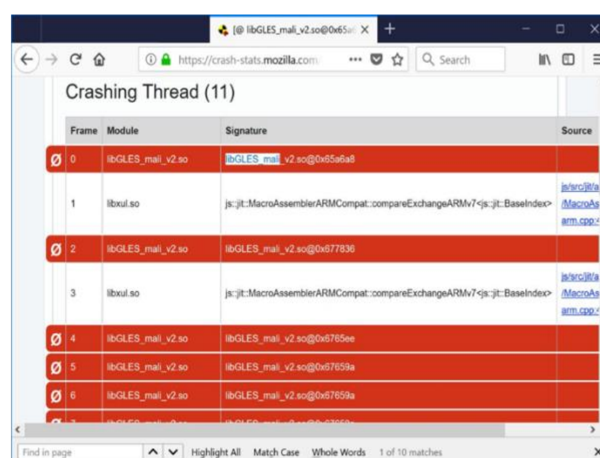
Benjamin Livshits

Alastair Donaldson

- Web browsers are among the most critical infrastructure on which society depends
- Testing web browsers to find semantic defects is fundamentally challenging
- We have employed mutation-based structural fuzzing to help address this problem, focussing on testing WebGL implementations inside major web browsers

The research work undertaken on this project at Imperial College London led to the development of an automated approach to finding defects in web browsers using mutation-based structural fuzz testing. The investigators decided to focus on testing components of web browsers related to high-performance graphics processing via the WebGL API, because the interaction between web browsers and graphics processing units has become a prominent attack surface in recent years. Two complementary approaches were explored: applying semantics-preserving transformations to WebGL pages to detect rendering problems, where a semantics-preserving change (which, by definition, should have no impact) leads to a change in what is rendered, and applying semantics-changing mutations to a well-formed page in order to test the browser's robustness to adversarial inputs. This led to the discovery and reporting of a number of issues in the Firefox and Chrome browsers, triggered by underlying defects in GPU drivers from a range of vendors. The associated tool in which the techniques are implemented will be open sourced in due course.

The funding from VeTTS was incredibly useful in allowing us to explore this emerging area. The work undertaken will form the basis for future publications, and has put us in a good position to apply for follow-on projects – a research grant from the Google Chrome University Research Program has already been secured (more details below). The work is strongly related to a line of work Donaldson has been pursuing for several years on *metamorphic testing* for graphics compilers, which led to the GraphicsFuzz start-up company (www.graphicsfuzz.com) that was acquired by Google and has since been open-sourced (https://github.com/google/graphicsfuzz). The VeTTS work on fuzzing WebGL has been integrated into GraphicsFuzz, which is actively being used to find defects in the Chrome web browser, including the vulnerabilities linked to below, which have all now been fixed (link).



A crash in Firefox caused by a driver bug discovered by our techniques

Publications. [1] Putting Randomized Compiler Testing into Production (Experience Report). Alastair F. Donaldson, Hugues Evrard, Paul Thomson, ECOOP 2020.

Related grants. Dr A. Donaldson, EPSRC Fellowship “Reliable Many-Core Programming”, 10/2016-09/2021, £1M. Dr A. Donaldson (Co-I), with C. Cadar (PI), EPSRC Grant “Automatically Detecting and Surviving Exploitable Compiler Bugs”, 01/2018-12/2020, £672K. Dr A. Donaldson, Google Chrome University Research Program project “Automatic Detection of Rendering-Related Security Vulnerabilities in Web Browsers”, 01/2018-04/2019, £130K.

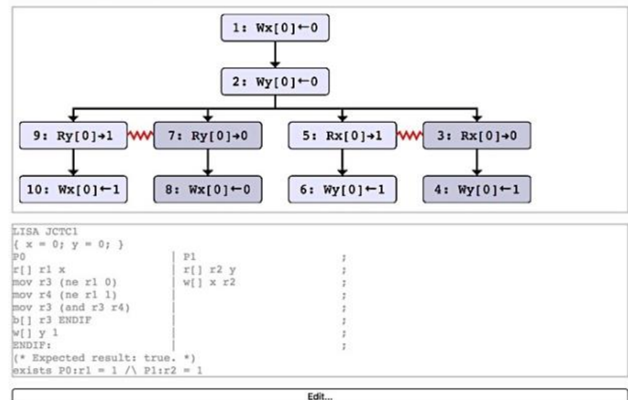
Impact Statement. “From a technical standpoint, the GraphicsFuzz work to which this VeTTS project is closely related has been highly successful in developing basic technologies for improving the security and reliability of billions of deployed mobile devices. From a broader point of view, this work has gotten widespread visibility and, of course, was seen by Google as being so valuable that they bought it.”
– John Regehr, Professor, University of Utah –

PrideMM Web Interface

Mark Batty **Radu Grigore**

- Prose specifications of relaxed memory behaviour are imprecise and lead to bugs in language specifications, processors, compilers and vendor-endorsed programming idioms
- Mechanised formal models have been used in academia to unambiguously specify and verify relaxed memory behaviour
- PrideMM is a Solver for Relaxed Memory Models, which improves on state-of-the-art descriptions of the concurrency behaviour of programming languages
- PrideMM provides a platform for comparison, testing, and refinement of relaxed memory models

Modern computer systems have *relaxed memory*: they exhibit highly unintuitive memory behaviour as a result of aggressive processor and compiler optimisations. At the same time, these systems are specified with relatively imprecise prose specifications, leading to bugs in language specifications, deployed processors, compilers and vendor-endorsed programming idioms. A push from academia has, in place of prose, introduced mechanised formal models that unambiguously specify relaxed memory behaviour, together with proofs and simulation tools that allow the validation of key design goals.



PrideMM screenshot. One specifies a test, model, and outcome and PrideMM works out whether the outcome is allowed or not. “True” indicates the outcome is allowed, and the graph indicates the underlying mathematical structure justifying this outcome.

This project concerns PrideMM: a solver that allows one to run tests over state-of-the-art descriptions of the concurrency behaviour of programming languages. Previous relaxed memory simulators were based on ad-hoc backends or SAT solvers. Additional computational complexity arises in cutting-edge language models that must consider multiple paths of control flow, so the

simulator backend embodies a problem outside of the scope of SAT. The problem is, however, within the scope of rapidly improving QBF solvers, atop which PrideMM is built.

The Web Interface to PrideMM, available at https://www.cs.kent.ac.uk/projects/prideweb/, is an essential outcome of this project. It allows one to run large batteries of automatically generated tests, and compare its runtime to those of the existing state of the art. The goal of PrideMM is to facilitate discussion with the specifiers of industrial concurrency models, promoting the latest academic solutions to open problems faced by industry.

PUBLICATIONS. [1] M. Batty et al. "PrideMM: A Solver for Relaxed Memory Models", draft paper detailing representations of key memory models, a proof-of-concept backend, and a specification language that marries expressiveness and ease of solving. [2] M. Janota, R. Grigore, V. Manquinho. "On the Quest for an Acyclic Graph", draft paper on finding acyclic graphs under a set of constraints, a general problem central to PrideMM.

RELATED GRANTS. Dr Mark Batty, EPSRC Grant: "Compositional, dependency-aware C++ concurrency", PI, £98,786, 04/2018-03/2020. Dr Mark Batty, EPSRC Grant "Verifiably Correct Transactional Memory", Co-I, £82,904, 07/2018-06/2021. PrideMM is the starting point for tools envisaged by these two grants.

IMPACT STATEMENT. "I believe that a well-reasoned memory model is the most important feature of any parallel programming platform, and that Mark Batty's work has contributed to building confidence in these models more than anyone else's."
– Olivier Giroux, Distinguished Architect at NVIDIA, Chair of Concurrency & Parallelism for ISO C++ –

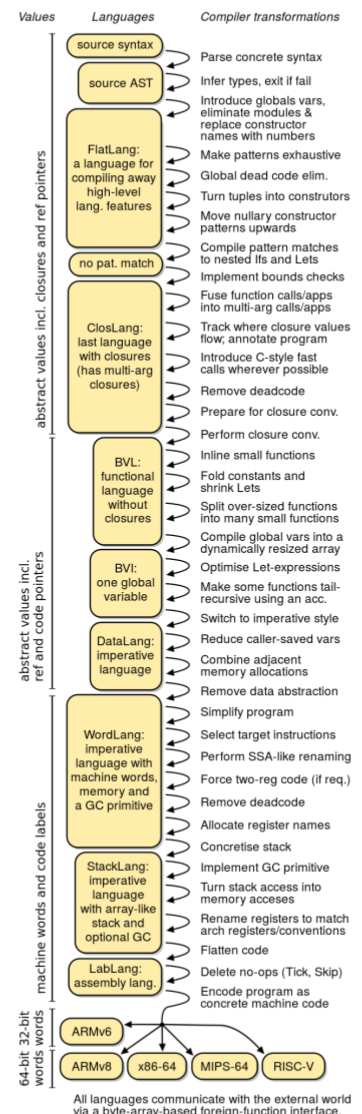
VERIFYING EFFICIENT LIBRARIES IN CAKEML

SCOTT OWENS

- CakeML is a functional programming language and an ecosystem of associated proofs and tools, including a formally verified compiler to various processor architectures
- CakeML currently lacks support for verifying libraries that use unsafe features, for example, array accesses w/o bounds checks
- The RustBelt project uses the Iris framework to reason about unsafe features of Mozilla's Rust language
- This exploratory project investigated the feasibility of using RustBelt's Iris to verify CakeML programs: we established that it is not possible to use Iris as-is, and that it one must develop an Iris-like logic for CakeML

CakeML is a dialect of the ML family of programming languages and was originally designed to play a central role in trustworthy software systems. The CakeML project is an ongoing collaboration between S. Owens (Kent, UK), M. Myreen (Chalmers, Sweden), and J. Pohjola and M. Norrish (Data61, Australia). The project's main accomplishment is the first fully verified compiler for a practical, functional programming language.

The RustBelt project aims to put the safety of Mozilla's Rust programming language on a firm semantic foundation. Rust's standard libraries make widespread internal use of *unsafe* blocks, which enable them to opt out of the type system when necessary. The hope is that such unsafe code is properly



CakeML Infrastructure

encapsulated, preserving language-level safety guarantees from Rust's type system. However, subtle significant bugs with such code have already been discovered by RustBelt.

This project explored the way in which fundamental mathematical insights from RustBelt could be incorporated into CakeML's suite of verification tools, setting the foundation for follow-up projects with greater scope for more advanced unsafe features, such as C's *malloc* and *free*, or passing CakeML data to C functions. Such features are important, as they bring end-to-end verification to performance-critical areas, such as uni-kernel operating systems, or distributed systems where even (non-end-to-end) verified systems are known to be buggy.

We have established, while Iris technology can, in principle, solve the problems observed in CakeML, we would need to re-design the logical foundations of Iris to accommodate the CakeML proof ecosystem. In particular, the HOL4 theorem prover of CakeML has foundational differences from RustBelt's Coq theorem prover. This is the subject of our subsequent VeTSS project.

RELATED GRANTS. Dr Scott Owens, EPSRC Grant: "Trustworthy Refactoring", 09/2016-03/2020, £728,766.

PUBLICATIONS. H. Férée, J. Å. Pohjola, R. Kumar, S. Owens, M. O. Myreen, and S. Ho "Program Verification in the Presence of I/O Semantics, verified library routines, and verified applications", VSTTE 2018.

IMPACT STATEMENT. "At Rockwell Collins, we use CakeML in projects to build avionics components with formally proven behavioural guarantees: these components have to exhibit high performance. In some cases, this can be achieved by algorithmic transformations already justifiable in CakeML. Beyond that, a great deal more performance can be obtained by unsafe (formally verified) compilation steps, and we are eager to take advantage of such advances when they become available."
– Konrad Slind, Senior Industrial Logician, Rockwell Collins –

SUPERVECTORIZER

GRETA YORSH

- Optimising compilers for Single-Instruction-Multiple-Data (SIMD) architectures rely on sophisticated program analyses and transformations
- Correctness hard to prove due to interaction between optimisation passes and SIMD semantics/costs
- Supervectorizer: integration of *unbounded superoptimization* with *auto-vectorisation* enables software to take full advantage of SIMD capabilities of existing and new microprocessor designs
- Potential for fundamental advances in SMT solvers and industrial-strength SIMD optimizing compilers

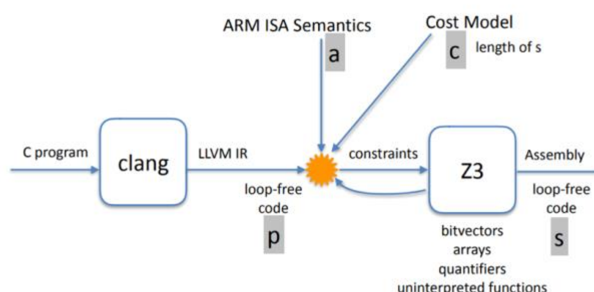
Optimising compilers for Single Instruction Multiple Data (SIMD) architectures rely on sophisticated program analyses and transformations. In particular, auto-vectorisation is designed to automatically identify and exploit data-level parallelism. To deliver expected performance improvements, compiler writers resort to changing optimisation passes, heuristics, and cost models. This process is highly challenging even for the few experts who possess the required range of skills, and any errors introduced affect the entire software stack, likely compromising its reliability and security.

Ensuring correctness of these compiler optimisations is hard due to implicit interactions between optimisation passes and abstruse details of SIMD instructions semantics and costs. It results in missed optimisation opportunities and subtle bugs, such as miscompiled code, which might remain undiscovered for a long time and manifest themselves in obscure ways across abstraction layers of a software stack.

This project aimed at enabling software to take full advantage of SIMD capabilities of microprocessor designs, without modifying the compiler. In particular, we integrate unbounded superoptimization with auto-vectorisation. This approach reduces the engineering effort needed to tune a production compiler for new SIMD architectures and improves compiler reliability without compromising the performance of generated code. We believe that this approach will lead to fundamental advances in SMT solvers and industrial-strength optimising compilers targeting SIMD architectures.

The work done in this project has had the following impact:

- Initial results were presented, by invitation, at Intel's Compiler, Architecture and Tools Conference (CATC).
- Postdoctoral research assistant, Julian Nagele, who joined in January 2018, has been working on a robust prototype implementation and experiments with SIMD instructions. Julian is engaged with the LLVM community and obtained valuable early-stage feedback from developers at EuroLLVM 2018.
- The work on this project has led directly to the award to Dr Yorsh of ERC Starting Grant. Initial results obtained under VeTSS funding demonstrated feasibility of the proposed ERC plan and the work under ERC will build on the infrastructure and experimental results obtained under VeTSS funding.
- The quantitative trading firm Jane Street expressed interest in incorporating techniques developed under this grant into the compiler for OCaml.
- Amazon invited Dr Yorsh to join as Amazon Scholar to work with Amazon Video on tools for improving correctness and performance of their code.



Structure of the preliminary prototype

RELATED GRANTS. Dr Greta Yorsh, ERC Starting Grant, £1.25M, 2018-2022.

EASTEND: EFFICIENT AUTOMATIC SECURITY TESTING FOR DYNAMIC LANGUAGES

ROYAL HOLLOWAY UNIVERSITY OF LONDON

JOHANNES KINDER

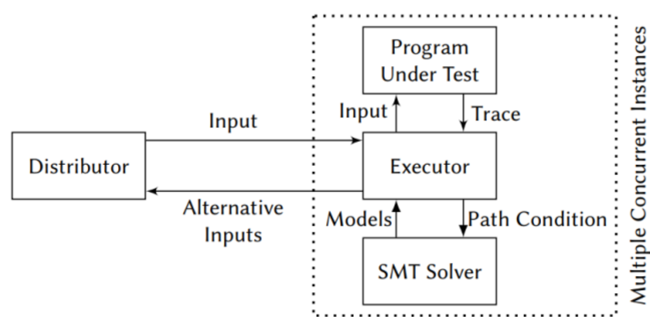
- Dynamic languages like JavaScript and Python are immensely popular
- Dynamic types and non-standard semantics make security bugs difficult to spot
- EASTEND focused on automated security testing for dynamic languages, in particular JavaScript.
- EASTEND improves the applicability of dynamic symbolic execution for JavaScript code and develops a flexible specification and testing methodology for security properties

EASTEND is based on the hypothesis that inherently dynamic languages are best served by a dynamic approach to verification that points to errors in the code without restricting the freedom of the developer. It uses test generation via dynamic symbolic execution (DSE) to systematically cover paths through programs and check security properties along those paths. The two main research objectives of EASTEND were: improving the applicability of dynamic symbolic execution (DSE) for real-world JavaScript code (RO1); and developing a flexible specification and testing methodology for security properties that goes beyond simple assertion checking (RO2).

Regular expressions (REs) limit applicability of DSE to testing code security in practical client- and server-side web applications, as modern solvers cannot reason about real-world REs as used by developers. We developed an encoding of complex REs and with a refinement scheme that soundly translates REs into the subset supported by state-of-the-art solvers. We implemented our approach in our DSE engine for JavaScript, ExpoSE [1], and evaluated it on 1,131 Node.js packages, demonstrating that the encoding is effective and can increase line coverage by up to 30%, meaning that more parts of the program can be reached, increasing the analysis surface for detecting bugs/vulnerabilities, e.g., using the specification and testing methods developed as part of RO2.

We have developed a methodology for specification-based testing of cryptographic applications based on type-like tags attached to runtime values that we call "Security Annotations" (SAs) [2]. We have developed explicit SAs for the widely-used JavaScript library Crypto.JS, which implements common cryptographic algorithms and primitives. These will allow developers using Crypto.JS to automatically inject our annotations into their testing environment at runtime without any expert knowledge required. By using DSE with ExpoSE on a program using an appropriately annotated API, developers will be able automatically detect cryptographic bugs without additional annotation requirements. The code and data that resulted from the project can be found here.

PUBLICATIONS. [1] B. Loring, D. Mitchell, J. Kinder. "ExpoSE: Practical Symbolic Execution of Standalone JavaScript". In Proc. Int. Symp. on Model Checking of Software (SPIN), pp. 196–199, ACM, 2017. [2] D. Mitchell, L. T. van Binsbergen, B. Loring, and J. Kinder. "Checking Cryptographic API Usage with Composable Annotations". In ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM), 2018. [3] D. Mitchell, J. Kinder. A Formal Model for Checking Cryptographic API Usage in JavaScript. ESORICS (1) 2019: 341-360. [3] B. Loring, D. Mitchell, and J. Kinder. Sound Regular Expression Semantics for Dynamic Symbolic Execution of JavaScript. In Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI), pp. 425–438, ACM, 2019.



Parallel testing architecture of ExpoSE

IMPACT STATEMENT. "We have started using ExpoSE as a key component of a research project on privacy-preserving proxy servers. To the best of my knowledge, it is the only existing tool for dynamic symbolic execution of modern real-world JavaScript code."
– Prof. James Mickens, Harvard University –

AUTOMATED REASONING WITH FINE-GRAINED CONCURRENT COLLECTIONS

ILYA SERGEY **NIKOS GOROGIANNIS**

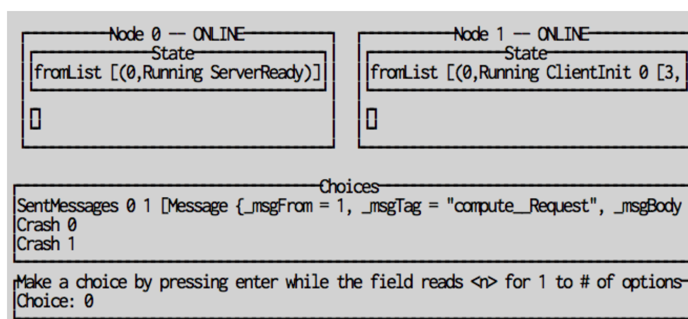
- A domain-specific language (DSL) for concurrent implementations of distributed protocols.
- Prototype DSL implementations of consensus protocols: Two-Phase Commit, Paxos, Multi-Paxos.
- An extension of Disel, a higher-order separation logic for distributed systems to handle concurrent per-node implementations of distributed protocols.

As per the original proposal, the funding has been used to host Kristoffer Just Andersen as a visiting student at the CS department of UCL, where he has worked under our supervision on the applications of techniques for

logic-based reasoning about concurrency to the verification of distributed systems with internal multi-threaded parallelism. The project thus naturally evolved from the initially proposed research, elaborating and extending it for the distributed setting. The artefacts produced to date include the runnable prototype (in Haskell) as well as a (partially) mechanised logical development for the verification of multithreaded distributed programs. During Andersen's stay at UCL, Sergey and Andersen developed a domain-specific language for specifying, implementing, randomised testing and visual debugging of distributed protocols.

We have developed *Distributed Protocol Combinators* (DPC), a declarative programming framework that aims to bridge the gap between specifications and runnable implementations of distributed systems, as well as facilitate their modelling, testing, and execution. DPC builds on the ideas from the state-of-the art logics for compositional systems verification. DPC contributes with a novel family of program-level primitives, which allows construction of larger distributed systems from smaller components, streamlining the usage of the most common asynchronous message-passing communication patterns, and providing machinery for testing and user-friendly dynamic verification of systems. The approach has been implemented in a form of a reusable Haskell library, as well as a tool for visual debugging of asynchronous systems.

Declarative programming over distributed protocols is possible and could lead to new insights, such as better understanding on how to structure systems implementations. Even though there are several known limitations to the design of DPC due to the chosen linguistic foundations (i.e., Haskell), we consider our approach beneficial and illuminating for the purposes of prototyping, exploration, and teaching distributed system design. In future, we will explore the opportunities, opened by DPC, for randomised protocol testing and lightweight verification with refinement types.



Visual debugging of asynchronous systems using DPC

PUBLICATIONS. [1] K. J. A. Andersen, I. Sergey, "Distributed Protocol Combinators", PADL'19. [2] N. Polikarpova, I. Sergey. "Structuring the Synthesis of Heap-Manipulating Programs", POPL'19. [3] K. J. A. Andersen, I. Sergey, "Protocol Combinators for Modelling, Testing, and Execution of Distributed Systems", J. Funct. Program. 2021. [4] Distributed Protocol Combinators, Kristoffer Just Arndal Andersen, and Ilya Sergey, PADL 2019.

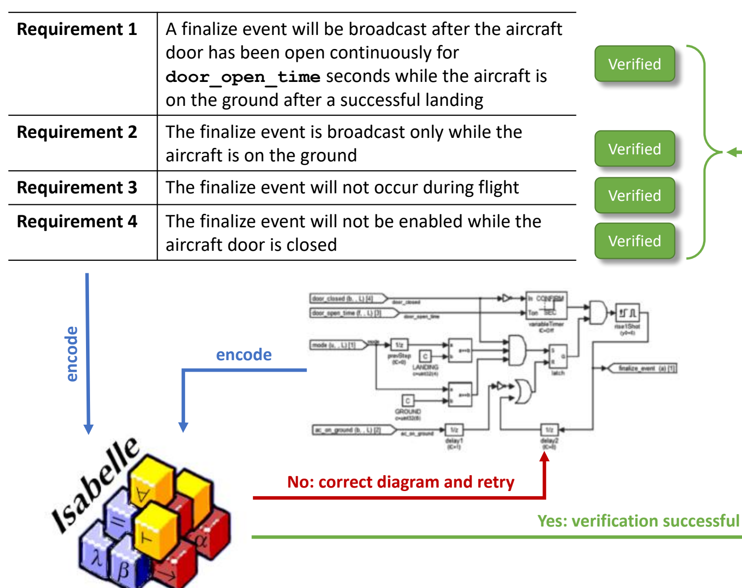
RELATED GRANTS. Dr Ilya Sergey, EPSRC Grant "Program Logics for Compositional Specification and Verification of Distributed Systems", 01/2017-11/2018, £101,009. Dr. Ilya Sergey, Google Faculty Research Award, "Distributed System Optimisations as Network Semantics Transformations", 2018.

MECHANISED ASSUME-GUARANTEE REASONING FOR CONTROL LAW DIAGRAMS VIA CIRCUS

JIM WOODCOCK **SIMON FOSTER**

- Theoretical reasoning framework for discrete-time part of control-law block diagrams (such as Simulink), based on mathematical semantics of diagrams and capable of dealing with large state spaces
- Contract-based compositional reasoning using refinement for verification of large systems
- Supports reasoning about diagrams with algebraic loops, ignored by most other verification approaches
- Verification of a subsystem of an industrial aircraft cabin-pressure control application

Control-law diagrams are used in industry to model complex engineering systems, such as the many components of modern aircrafts. These systems must be built to the very highest standards possible, and their control laws must be verified to ensure that they behave as required. Our project proposes a general methodology based on mathematical descriptions of diagrams. It is expressive enough both to capture the full range of behaviours required and to be used with other engineering techniques and their own diagrams and notations. Our techniques scale up to tackle verification of large-scale systems. In this VeTSS-funded project, we developed a theoretical reasoning framework for discrete-time blocks of control-law diagrams. As well as giving a mathematical meaning to Simulink (an industry-standard diagrammatic notation for depicting control laws), our framework links to Modelica (another industry standard notation) for multi-model descriptions. Our verification technique relies on computer programs that automatically follow human patterns of reasoning.



Workflow: from textual representation to formal verification

We used our framework to verify the control laws for a subsystem used in aircrafts that controls the cabin pressure after landing. Specifically, the cabin-pressure system must keep working until the aircraft has made a successful landing and the cabin doors have been open for a minimum amount of time. The subsystem is made by Honeywell and we worked with colleagues at D-RisQ. Our technique revealed a vulnerable block that should be improved. The outcomes of this project include a theory to reason about block diagrams using mathematical contracts, mechanisation of the theory in the Isabelle theorem prover, as well as the verification of the cabin-pressure control subsystem. A technical report is available online at http://eprints.whiterose.ac.uk/129640/.

PUBLICATIONS. K. Ye, S. Foster, J. Woodcock. "Compositional Assume-Guarantee Reasoning of Control-Law Diagrams using UTP", From Astrophysics to Unconventional Computation, pp. 215-254, Springer International Publishing, 2020.

RELATED GRANTS. Dr Simon Foster, EPSRC UKRI Innovation Fellowship: "CyPhyAssure: Com-positional Safety Assurance for Cyber-Physical Systems", £562,549, 06/2018–05/2021, with project partners ClearSy and D-RisQ.

IMPACT STATEMENT. "Simulink is a language highly applied by industry in the development of safety-critical embedded, real-time, and cyber-physical systems, where the establishment of accessible verification support can have substantial impact. This VeTSS project has made a crucial step forward in this area by provision of theorem proving technology in Isabelle/UTP, validated by its application to a real-world aircraft cabin-pressure control application from our company."*– Colin O'Halloran, CEO, D-RisQ –*

VeTSS 2018/2019

A Foundation for Testing and Verifying C++ Transactions

John Wickerson
Imperial College London

Session-type-based Verification Framework for Message-passing in Go

Nobuko Yoshida
Imperial College London

Specification and Verification of C++ Data-structure Libraries

Mark Batty
University of Kent

Trustworthy Software for Nuclear Arms Control

Andy King
University of Kent

Building Verified Applications in CakeML

Scott Owens
University of Kent

Operating System Components as Verified Libraries

Tom Ridge
University of Leicester

Formal Verification of Quantum Security Protocols using Coq

Raja Nagarajan
Middlesex University London

Generating Exploitable Crashes

Daniel Kroening
Oxford University

Supervectorizer (Phase II)

Greta Yorsh
Queen Mary University of London

Automated Black-box Verification of Networking systems

Alexandra Silva
University College London

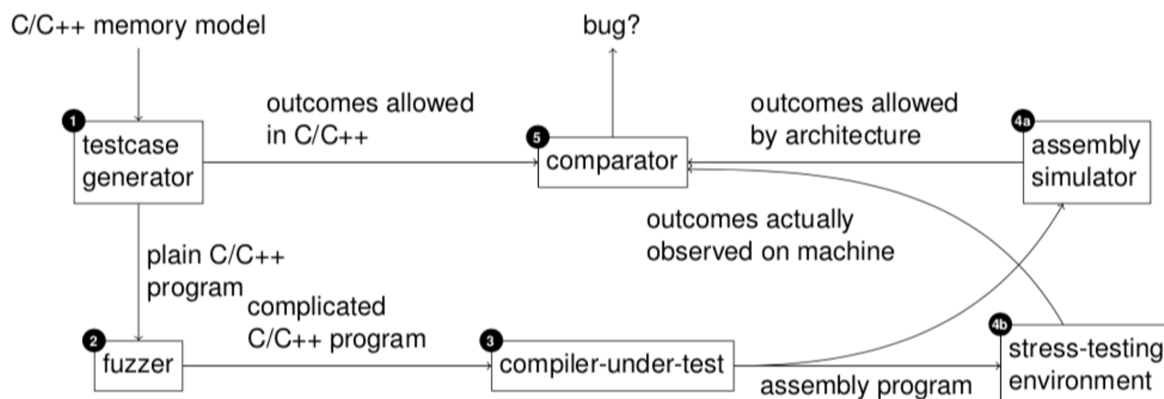
A Foundation for Testing and Verifying C++ Transactions

John Wickerson

- Concurrent programming is hard, but transactional memory promises to make it simpler.
- Transactional memory lets programmers make a group of instructions execute 'instantaneously' by marking them as a 'transaction'.
- However, transactional memory is notoriously hard to specify and implement correctly in compilers.
- The aim of this project is to build a framework for testing whether or not compilers are implementing this form of concurrency correctly.

The scope of this project has broadened considerably compared to the original workplan. The original focus was on testing the compilation of transactional C/C++ concurrency. However, we soon realised that the proposed approach could be applied to test the compilation of any C/C++ concurrency, in a way that has never been tried before, but which has the potential to be a hugely valuable technique for improving compiler reliability. This generalisation has the potential to greatly amplify the impact of our work.

We have developed a prototype tool that uses a combination of techniques (automatic testcase generation, semantics-preserving code mutation, and exhaustive simulation) to search for bugs in the way mainstream compilers translate concurrent C/C++ (including atomic operations). The tool is open-sourced and available under a permissive licence (https://c4-project.github.io/). It has generated and run hundreds of thousands of tests on several mainstream compilers targeting a range of architectures. Our efforts have revealed concurrency-related bugs in historic versions of the GCC compiler, as well as two non-concurrency-related bugs in recent versions of GCC and the IBM XL compiler. The diagram below illustrates the structure of the tool.



Publications. Our prototype tool has been presented at a scientific meeting at the UCL Computer Science department (November 2018) and at the S-REPLS programming languages workshop (February 2019). An article about its design is currently (February 2021) under submission at a leading conference in the field.

Related grants. This project is also being partially funded by the EPSRC Programme Grant "IRIS: Interface Reasoning for Interacting Systems" (£6.1M, 2018–2023).

Impact Statement. "The programmability of concurrent systems, especially under weak-memory models, is an important challenge for Arm. This is an active area of interest to Arm, and we are delighted to see work that looks at a fuller formalisation of C++ transactional memory."
– Nathan Chong, formerly Principal Researcher at Arm, now Principal Engineer at Amazon Web Services –

SESSION-TYPE BASED VERIFICATION FRAMEWORK FOR MESSAGE-PASSING IN GO

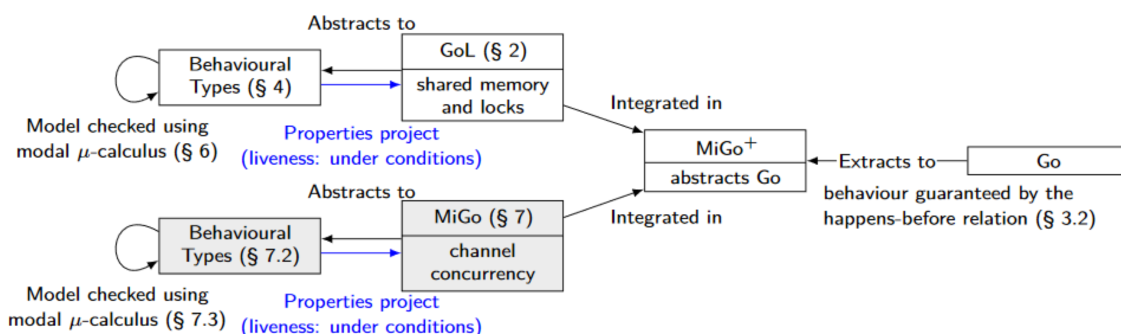
NOBUKO YOSHIDA

- Concurrent programs introduce a specific class of bugs relating to memory safety and deadlocks.
- Detection of concurrency bugs in Go and other concurrent languages often relies on runtime analysing with fuzzing techniques, making it unreliable and allowing bugs to make it to the production software.
- Static detection frameworks allow to reduce the fraction of those bugs that make it to production stages, by consistently analysing a session types model of the source code against properties based on behavioural and memory models of the target language.
- Coupling both techniques allows for an increase in robustness for the development chain of trusted software, ensuring a lesser number of critical bugs end up in production.

Go is a concurrent programming language developed in recent years by Google. It is used by various projects, including cloud service providers like Twitter and Dropbox, open source projects including Docker, Kubernetes and CoreOS. This increasing popularity is reflected on Go's position in several programming language's rankings, including IEEE Spectrum Top Programming Languages (from place 20 in 2014 to 9 in 2017), StackOverflow's most loved and most wanted languages (5th and 3rd place resp., in 2018), and Github's top growing languages (where it ranked 7th in 2018). Go's most appreciated features are notably its concurrency features, including channel-based message-passing and lightweight thread creation features. These features, however, make programmers struggle with bugs such as communication deadlocks, message mismatches or memory safety issues.

This project builds on the foundations laid by works on detection of channel-based concurrency issues, and brings them further by proving the theoretical base of these works and extending it greatly. Our recent work tackles both channel-based concurrency issues (including deadlocks and safe channel usage) and shared memory-based issued, especially revolving on the correct usage of mutual exclusion locks and data race detection. We use Go's official memory model detailed in the documentation of the language, extracting from it a happens-before relation that is used to define how a data race can be statically detected.

We also formally prove the equivalence between properties of our abstracted types and properties of the source language, defining precisely what conditions programs need to meet so they can be correctly analysed by our framework. This framework is then implemented in a tool, Godel 2, the workflow of which is described in the figure on the right. It uses the mCRL2 model checker and the KiTTEL termination checker to verify the properties we extract from the code against the model behavioural-types we infer from program source code.



PUBLICATIONS. [1] J. Gabet and N. Yoshida, Static Race Detection and Mutex Safety and Liveness for Go Programs, ECOOP 2020. [2] R. Griesemer, R. Hu, W. Kokke, J. Lange, I.L. Taylor, B. Toninho, P. Wadler and N. Yoshida, Featherweight Go, OOPSLA 2020. [3] D. Castro, R. Hu, S. Jongmans, N. Ng and N. Yoshida, Distributed Programming Using Role Parametric Session Types in Go, POPL 2019.

RELATED GRANTS. Nobuko Yoshida, PI, EPSRC Established Career Fellowship: "POST: Protocols, Observabilities and Session Types", 04/2020-03/2025, £1.46M; JSPS Invitation Fellowship for Research in Japan, £7,000, 07-08/2019.

IMPACT. The approach used in this project can be extended to a toolchain, Scribble, a protocol description language based on Multiparty Session Types, that is used to design and verify protocols and their implementations. Scribble is used by teams and projects in companies such as Red Hat and Estafet to generate deadlock-free microservices in Go (http://estafet.com/scribble/).

"I want to thank you and your team for all the type theory work on Go so far — it really helped clarify our understanding to a massive degree. So thanks!"
– R. Griesemer, Google USA, to P. Wadler about the work on Featherweight Go. –

SPECIFICATION AND VERIFICATION OF C++ DATA-STRUCTURE LIBRARIES

MARK BATTY

- Concurrency in the C++ language is ill-specified in the current International Standards Organisation (ISO) definition: it allows values to be conjured out of thin air (OOTA).
- Working towards verification of C++ code, we fixed this problem with a proposed change to the standard called Modular Relaxed Dependencies (MRD).
- We presented our proposal to the ISO, and they voted unanimously to pursue it.
- We developed a refinement relation that can be used to verify C++ code.
- Further flaws recently uncovered in the standard must be rectified before full verification of data structure libraries is possible.

C AND C++ SEMANTICS: AN ONGOING ACADEMIC/INDUSTRIAL EFFORT. The memory behaviour of modern systems is extremely subtle. Processor vendors avoid the cost of fully hiding micro-architectural details, such as buffering and speculation, by permitting unintuitive program executions. Compiler optimisations alter accesses to memory to similar effect. The end result is a system with relaxed memory behaviour: behaviour that deviates from sequential consistency (SC), where concurrent memory accesses are simply interleaved.

Relaxed memory breaks intuitions about system behaviour leading to bugs in language specifications, deployed processors, compilers, and vendor endorsed programming idioms – it is clear that current engineering practice is severely lacking.

We have an ongoing academic/industrial partnership with the International Standards Organisation (ISO) that has exposed fundamental flaws in the way we specify programming languages. We have shown that the state-of-the-art definitions of C and C++ concurrency are broken -- a problem that stems from a tension between performance and the strength of ordering guarantees provided to programmers. Compilers optimise away some syntactic dependencies, but these programming patterns are an idiomatic way to provide ordering in machine code, and if they are left in place, they serve as a cheap source of ordering at the language level. The language definition must specify which dependencies are left in place: too many, and useful optimisations will be forbidden, as in Java Hotspot; too few, and the semantics of the language permits bizarre behaviour, as in C++. It is provably impossible to strike this balance in C++ by making only minor changes to the concurrency design, so a different approach was necessary. To reason about code, we must fix these problems.

As part of this VeTSS project, we developed Modular Relaxed Dependencies, a model for C++ concurrency which is recognised by the ISO as the best solution to the pernicious OOTA problem. We went on to develop a

refinement relation for this model that allows one to verify the correctness of C++ code. Most data structures use pointers, and as part of this work, we started to examine further newly-recognised flaws in the specification of pointers in C++.

PUBLICATIONS. M. Batty, S. Cooksey, S. Owens, A. Paradis, M. Paviotti, and D. Wright. Modular relaxed dependencies: A New Approach to the Out-of-Thin-Air Problem. ESOP 2020.

IMPACT STATEMENT. The ISO unanimously passed the following motion endorsing our semantics: "*OOTA is a major problem for C++, modular relaxed dependencies is the best path forward we have seen, and we wish to continue to pursue this direction.*"

TRUSTWORTHY SOFTWARE FOR NUCLEAR ARMS CONTROL

ANDY KING

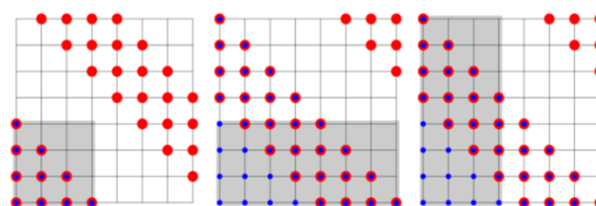
- We have developed new analysis techniques that are able to correctly handle machine arithmetic for integers of various width.
- This is crucial for reasoning about how paths can be taken, and cannot be taken, through binary programs.
- Our approach allows the model checker itself to be verified by using a classic bit-vector solver, which does not need to support interpolation.

The project focuses on the problem of how to automatically compare two AVR binaries. Using one given binary as the reference semantics, the problem is to determine whether the other binary has the same behaviour (even if it syntactically different), or whether it has been tampered with. We use interpolation-based model-checking to search for a path in one binary which does not exist in the other.

We have developed a new interpolation method for bit-vector formulae by leveraging on existing interpolation techniques for linear integer arithmetic, and integrated this method into the Impact interpolation algorithm, demonstrating how it improves on interpolation techniques which do not reason about the wrap-around nature of machine arithmetic. Interpolation is used to relax a sequence of symbolic formulae which represent a path through a program to give a more general sequence that describes, not just one path, but many.

We have also shown how interpolant methods can be extended beyond systems of linear constraints, and also how to take an interpolant which is a system of linear inequalities and always encode it as a compact bit-vector formula. We have improved an existing interpolation algorithm for bit-vectors by adapting computer algebra techniques to work over modulo arithmetic rather than the field of real numbers. Paradoxically, modulo arithmetic makes it easier, not harder, to automatically reason about the behaviour of the program.

Our analysis work is built on atop a tool-chain for decompiling AVR binaries which is, in turn, build on the QEMU toolkit for emulating various architectures. While undertaking our project, we have made contributions to the AVR support for QEMU, both making it both more robust and extending its functionality.



Integer solutions to the inequality $x + y - 4 \leq 3$. In linear integer arithmetic (LIA), these are the same as the solutions to $x + y \leq 7$, but its modulo solutions (in this case, mod 8) are easier to represent as bit-vector formulae. This gives way to converting LIA interpolants into bit-vector interpolants.

This research is of key importance to AWE's work on treaty veri-fication in an arms control context. It will pro-vide tools and techniques to verify the authenticity of monitoring equipment that could be deployed for future arms control treaties.

PUBLICATIONS. [1] Mind the Gap: Bit-vector Interpolation recast over Linear Integer Arithmetic, T. Okudono and A. King, International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'20); (2) Polynomial Analysis of Modulo Arithmetic, T. Seed, C. Coppins, A. King and N. Evans (under review); [3] Improved AVR support for QEMU, S. Harris, E. Robbins (signed off by M. Rolnik), https://lists.sr.ht/~philmd/qemu/, 2019.

RELATED GRANTS. Atomic Weapons Establishment. Follow-on funding, Dec 2019–Apr 2020, £20K.

IMPACT STATEMENT. "As a professional reverse engineer and binary analysis tool developer with 16 years of experience, I believe that the future of binary analysis lies in the further adoption and advancement of techniques in mathematical program analysis. In my estimation, the research group at the University of Kent is one of the only remaining groups – certainly the most prolific one – that is regularly attempting to tackle the theoretical issues plaguing the scalability of binary analysis. Not only are they "attempting" to tackle these issues, they have published a number of deep and fascinating papers, which offered truly novel contributions to binary analysis."
– Rolf Rolles, Möbius Strip Reverse Engineering –

BUILDING VERIFIED APPLICATIONS IN CAKEML

OLAF CHITIL

SCOTT OWENS

- CakeML is a functional programming language and ecosystem of associated proofs and tools, including an end-to-end, formally verified compiler targeting several processor architectures.
- CakeML lacks support for verifiably safe composition of user code with libraries that use "unsafe" features such as array accesses without bounds checks.
- To integrate "unsafe" features we are building a semantic type system by applying the technique of step-indexed Kripke logical relations to a CakeML-style lambda calculus in the simply-typed logic of HOL4.
- This is a three-year project supporting a PhD student, the results of which will be applied to CakeML itself.

CakeML is an ML-like programming language, intended to play a central role in trustworthy software systems. The CakeML project is an ongoing collaboration between researchers at the University of Kent (UK), Chalmers University of Technology (Sweden), and Data61 (Australia). The key contribution of the project is the first end-to-end verified, optimising compiler for a practical, functional programming language. However, CakeML lacks support for verifiably safe composition of user code with libraries that use "unsafe" features (such as array accesses without bounds checks).

We aim to solve this problem by producing a semantic type soundness result for CakeML, taking inspiration from the RustBelt project (Dreyer et al.). The latter aims to develop rigorous formal foundations for the Rust programming language, leveraging the Iris theorem-prover to verify that "unsafe" code (which opts out of Rust's type system) from standard libraries is safely encapsulated, and does not violate the safety guarantees intended by the type system. The RustBelt team have already discovered and fixed subtle, significant bugs. Our previous VeTSS project found that the mathematical insights behind the Iris logic are not straightforwardly ported to the logic of HOL4 for use in CakeML.

The PhD student (H. Kanabar) built a System F-like calculus in HOL4 in the style of CakeML, and established a semantic type soundness result via a logical relation, identifying several application areas for technology once applied to CakeML. The introduction of general references to the logical relation is the key problem in HOL4; the student is continuing to explore techniques and is engaging with experienced researchers in the field.

The PhD student is also pursuing other avenues to strengthen the guarantees offered by the CakeML project. During an internship at Arm, he investigated the Sail ecosystem (Sewell et al.) and its applications in formal reasoning about the semantics of the Arm instruction set architecture. He has made progress in verifying the existing Arm ISA model used in CakeML proofs against the official machine-checked model released by Arm, automatically extracted to HOL4 via Sail. He is also contributing to PureCake, a pure language in the style of Haskell (Peyton Jones et al.) , which will compile to CakeML to permit end-to-end verification.

RELATED GRANTS. Dr Scott Owens, EPSRC Grant: "Trustworthy Refactoring", 09/2016-03/2020, £728,766.

PUBLICATIONS. [1] H. Férée, J. Å. Pohjola, R. Kumar, S. Owens, M. O. Myreen, and S. Ho "Program Verification in the Presence of I/O Semantics, verified library routines, and verified applications", VSTTE 2018. [2] O. Abrahamsson, S. Ho, H. Kanabar, R. Kumar, M. O. Myreen, M. Norrish, and Y. K. Tan, "Proof-Producing Synthesis of CakeML from Monadic HOL Functions", JAR 2020.

IMPACT STATEMENT. CakeML is an important strategic research project for Data61. The 'strategic' tag means that we support it as a future-looking endeavour that does not necessarily have a short-term payoff, nor necessarily receives external funding. Instead, we pursue such projects because of our belief that they have high potential for future impact. That said, some of our work on CakeML does receive external funding in the form of money from a multi-million dollar DARPA-funded research projects. This project sees CakeML being integrated into high-assurance systems development, in collaboration with staff at Collins Aerospace. CakeML's ongoing development makes it a stronger and stronger component (better performing and more featureful) in existing and future systems of this sort. As we work with research collaborators around the world and continue to produce CakeML-related publications in the academic press, we also continue to demonstrate our commitment to the CakeML project as a vehicle for pure research."
– Michael Norrish, Principal Researcher, Data61, Australia –

OPERATING SYSTEMS COMPONENTS

TOM RIDGE

- Core operating system functionality has been verified in projects such as L4.verified, which targeted the seL4 microkernel.
- However, two other major components should also be verified: the network stack and the file system.
- A few verified file systems already exist, but their performance is slow compared to traditional file systems such as ext4 and ZFS. Moreover, they also lack important modern features, such as file system snapshots.
- We aim to develop a verified file system "ImpFS", constructed from small, well-defined components. ImpFS should match or exceed existing file systems, both in terms of performance and features. It will be available both as a desktop file system and as a library of components suitable for use in other software, and even in library-based unikernels such as MirageOS.

In previous work, we developed SibylFS, a formal model of POSIX and real-world file systems. The formal specification was usable directly as a test oracle, to check conformance of existing file systems. We now aim to actually implement a verified file system.

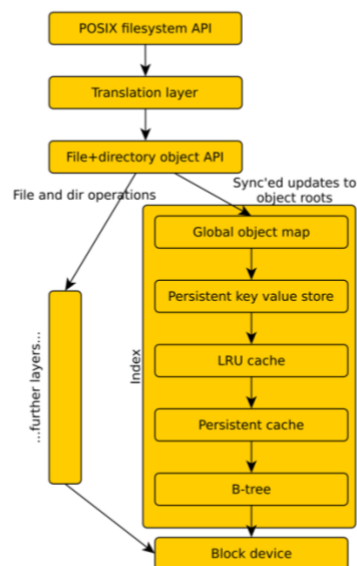
Building a file system is hard. Building a high-performance file system with advanced features is extremely difficult: the BTRFS file system has been in development by a team of engineers for around 10 years and is still

not considered stable enough for production use. Thus, we should expect that building a verified high-performance file system will be extremely challenging.

The most important components that we have developed so far are:

- A high-performance novel B-tree-like data-structure with both Copy-on-Write and Mutate-in-place semantics. This is formalised in Isabelle/HOL and extracted to OCaml for execution.
- A persistent cache, offering transactional-log-like functionality.
- A persistent key-value store.

These components are freely available online from http://www.tom-ridge.com/filesystems.html. All the components are implemented in a purely-functional style, which is a pre-requisite for easy verification. The performance of the components is extremely good. For example, the key-value store matches the performance of the well-regarded LMDB key-value store.



The Design of ImpFS

At OCaml'2020 we demonstrated a working file system. We are now in the process of tuning the implementation. We are confident that the file system will out-perform existing file systems in many areas. We are now working with industry collaborators and the MirageOS project to push this work into production. More importantly, the components will serve as the basis for many further interesting file system designs.

The VeTSS funding has been critical for securing research time for the author. Most of the work has involved high-level design and low-level component implementation. In addition, a lot of effort has been put into performance engineering of the components. This work is lengthy and would not have been possible without the financial support provided by VeTSS.

PUBLICATIONS. [1] A B-tree library for OCaml. T. Ridge. ICFP 2017, OCaml’17 workshop; [2] Towards verified file systems. A. Giugliano, 2018. PhD thesis; [3] A Key-Value store for OCaml. T. Ridge. ICFP 2019, OCaml’19 workshop. [4] The ImpFS filesystem. T. Ridge. IFCP2020, OCaml'20 workshop.

IMPACT STATEMENT. “I have been working, with a PhD student, on a verified copy-on-write filesystem. In the course of our work, we have relied heavily on Dr. Ridge’s ImpFS work. The availability of an existing verified copy-on-write B-Tree implementation has been of great help. We continue to use it heavily as a reference point for not only our implementation, but also for an example of formally specifying this crucial class of data structures.”
– Colin S. Gordon, Drexel University, Pennsylvania –

FORMAL VERIFICATION OF QUANTUM SECURITY PROTOCOLS USING COQ

RAJAGOPAL NAGARAJAN

WITH

JAAP BOENDER

RICHARD BORNAT

FLORIAN KAMMÜLER

- Quantum computing and communication systems are increasingly becoming practical and are likely to revolutionise modern technology
- Formal methods have been extremely valuable in ensuring correctness as well as security of classical systems and are widely used in industry
- The aim of this project has been to use the proof assistant Coq to verify quantum communication and cryptographic protocols
- Qtpi, our implementation of the quantum process calculus CQP, allows for rapid prototyping

This project uses the proof assistant Coq to verify quantum communication and cryptographic protocols. In earlier work, we formalised qubits and quantum operations in Coq. In this project, we implemented a Quantum IO monad in Coq for the specification of the protocols. In addition to quantum gates and measurement, the monad gives us a lightweight process calculus which supports sequencing of operations and keeping of state. We have proved this monad has the necessary properties. The process simulation function that gives the QIO monad its semantics has also been written. We have been proving properties of simple quantum protocols.

```
Theorem teleportation:
forall phi: qubit 1, forall q: qubit 3,
forall pr: IR, forall qp: List.In (pr, q)‘(Alice phi),
 exists z: qubit 2,
   ‘Bob(q, Alice_out phi q (Alice_case phi q pr qp)) {=} ‘(z {o} phi).
```

As an example, we show the formalisation of the main theorem in Coq that proves that the quantum teleportation protocol actually transmits Alice’s qubit ϕ to Bob. We do not show the formalisation of Alice and Bob here, but the aim is to show that the combination of Alice’s and Bob’s functions results in a triple of qubits whose last element is the same as ϕ. The theorem states that, for each of the four possible outcomes for q which is an instance of ϕ, a suitable z exists.

In preparation for the Coq verification, teleportation and many other protocols were specified and analysed using Microsoft's Q# and our own symbolic evaluator, Qtpi, which has been developed within the timeframe of this project. Qtpi is an implementation of the quantum process calculus CQP. It is more suited to modelling distributed computation than Q#. It also uses symbolic rather than numeric quantum calculation. Programs are checked statically, before they run, to ensure that they obey real-world restrictions on the use of qubits (e.g. no cloning, no sharing). Qtpi should be of independent interest as a quantum programming language implementation and is available from GitHub (https://github.com/mdxtoc/qtpi).

IMPACT. “Quantum Technologies are set to play a big role in the development of technology and modern society. Novel work done by Prof. Nagarajan and his collaborators on quantum programming and formal verification, such as in his VeTTS project, is likely to make a strong impact in making quantum systems safe and secure. *– Bob Coecke, Chief Scientist at Cambridge Quantum Computing, formerly Professor of Quantum Foundations, Logics and Structures, Head of the Quantum Group, Department of Computer Science, University of Oxford." –*

PUBLICATIONS. [1] R. Bornat, J.Boender, F. Kammueller, G. Poly and R. Nagarajan, “Describing and Simulating Concurrent Quantum Systems”, Tool Demonstration Paper, TACAS 2020. An extended version of this paper will appear in Samson Abramsky on Logic and Structure in Computer Science and Beyond, Palmigiano and Sadrzadeh (eds.), Outstanding Contributions to Logic Series, Springer, 2021.

TOWARDS OPTIMISED TAINT ANALYSIS

DANIEL KROENING   JOHN GALEA

- Generic Taint Analysis is a flexible technique that enables the enforcement of different taint policies via the same underlying taint tracking system.
- However, generic taint analysis incurs severe performance overheads.
- We introduce the Taint Rabbit, an optimized generic taint engine capable of analysing x86 binary applications. Its enhanced performance is based on optimizations that we have investigated and relate to fast path generation and vectorization. Overall, the work done acts as a foundation for further research on security vulnerabilities.
- Results show that with our optimizations, the Taint Rabbit performs faster than existing generic trackers.

Dynamic taint analysis is a pivotal technique in software security that enables the tracking of interesting/suspicious data as it flows during execution. With the essential funding provided by VETSS, we have researched approaches for optimizing an expensive but generic variant of the analysis. Crucially, the analysis supports various user-defined policies via the same underlying taint tracking system. The research is carried out in an effort towards our long-term goal of automatically detecting and analysing software vulnerabilities and reasoning over their exploitation.

The main performance bottleneck of taint analysis stems from the execution of taint propagation code that is intensively instrumented into the target application at instruction granularity. Unlike specialised bitwise tainting, a generic taint tracker cannot be optimised for a specific taint policy. Instead, it must perform elaborate propagation in order to be versatile. We adopt two strategies to address the performance issue. First, we aggressively elide the execution of propagation routines whenever possible, by generating fast paths that result in basic blocks being instrumented based on frequent taint contexts identified at runtime. Second, we directly optimize the code that is responsible for actually conducting taint propagation, leveraging vectorization so that all taint information pertaining to source operands of a given instruction are processed simultaneously.

Our research has led to the development of the Taint Rabbit, a novel generic taint tracker that uses our proposed techniques. We evaluated our approach on a number of real-world applications including Apache, PHP, and bzip2, as well as on CPU-intensive benchmarks such as SpecCPU 2017. Results indicate that the Taint Rabbit is the fastest generic taint engine amongst those we assessed. Furthermore, to demonstrate the flexibility of the Taint Rabbit, we also developed several taint-based applications using our versatile system despite their dependence on different taint propagation policies. In particular, we considered Use-After-Free debugging, control-flow hijack detection, and vulnerability discovery through fuzzing. Overall, VETSS has given us the opportunity to engage in imperative research which resulted in generic taint tracking to scale better for binaries than the current state-of-the-art. The Taint Rabbit serves as a vital stepping-stone to automatically analyse and understand security-critical software vulnerabilities.

PUBLICATIONS. [1] John Galea and Daniel Kroening. 2020. The Taint Rabbit: Optimizing Generic Taint Analysis with Dynamic Fast Path Generation. ASIACCS '20.

IMPACT. The Taint Rabbit and all tools built upon it will be made open-source upon publication. We have also made several contributions to DynamoRIO, the open-source DBI system that the Taint Rabbit uses. In this regard: "John has had significant impact on the open-source DynamoRIO project: he has contributed numerous fixes and features to the code base; he has joined the small set of core developers who voluntarily help maintain the continuous integration testing and other infrastructure; he has influenced design decisions for new features by other developers; and he has helped to build the community around this project."
– Derek Bruening, Software Engineer, Google –

SUPERVECTORIZER (PHASE II)

GRETA YORSH

- Optimising compilers for Single-Instruction-Multiple-Data (SIMD) architectures rely on sophisticated program analyses and transformations
- Correctness hard to prove due to interaction between optimisation passes and SIMD semantics/costs
- Supervectorizer: integration of *unbounded superoptimization* with *auto-vectorisation* enables software to take full advantage of SIMD capabilities of existing and new microprocessor designs
- Potential for fundamental advances in SMT solvers and industrial-strength SIMD optimising compilers

The original aim of the project was to apply unbounded superoptimization to the problem of generating SIMD code. This approach results in faster code than what can be generated using traditional compiler optimization technique called vectorization. With this approach, the problem is encoded in first-order constraints and solved using an SMT solver which has to be extended with special heuristics.

As a prerequisite, the PI started the theoretical development of a framework for symbolic cost models of modern compute architecture, targeting ARM8 as the first experimental platform. The RA supported by this grant, Julian Nagele, worked on the improvement of the initial prototype to make it more reconfigurable and extensible, and evaluation on small, but important benchmarks.

Unfortunately, the PI had substantial health problems, and could not continue to be involved in the work. Without the support of the PI, the RA was not able to carry out the work on SIMD code generation, which required expert knowledge of vectorization, ARM architecture, and SMT solvers.

However, the RA realised the potential of applying this technology to smart contracts. He took a leading role on this research, identifying the direction of blockchain, and brought the work to completion. The PI provided high-level guidance, but was not actively involved in this work. The RA independently established a collaboration at UCL with a group interested in start contracts verification, which motivated this work on optimization. The paper has been accepted for publication at LOPSTR'19. The reviewers recognised the importance of the application domain of smart contracts, novelty of superoptimization in this context, the extensive specification and benchmarking produced by the authors. The prototype and benchmarks are available as open-source (https://github.com/juliannagele/ebso). The RA and the PhD student are preparing a journal version of the paper. Recently, there has also been interest in this prototype from industry (https://www.embecosm.com).

Upon completion of this project, the PI has gone on to work at the global proprietary trading firm Jane Street, whereas the RA has gone on to work at Bank of America.

PUBLICATIONS. Blockchain Superoptimizer. Julian Nagele, Maria A. Schett. LOPSTR 2019.

RELATED GRANTS. Dr Greta Yorsh, ERC Starting Grant, £1.25M, 2018-2022.

AUTOMATED BLACK-BOX VERIFICATION OF NETWORKING SYSTEMS

ALEXANDRA SILVA

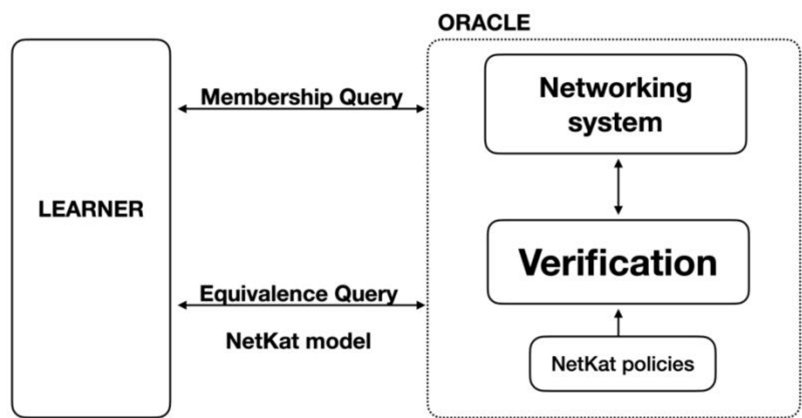
MATTEO SAMMARTINO

- Modern technology relies on complex and safety-critical networking systems
- Automated verification techniques can detect unintended network behaviour and security vulnerabilities
- Our approach incrementally learns a state-based model from a network by observing its behaviour
- Long-term objectives include achieving scalability of learning and verification of network programs

As complex networking systems, such as Software Defined Networks, Cloud Computing and the Internet of Things, become more and more popular, automated verification tools capable of assessing the security and reliability of such systems are in high demand.

Classical approaches to verification tasks require the existence of a model of the system of interest, able to express all its relevant behaviour. Unfortunately, in reality such a model rarely exists, as networking systems may be extremely heterogeneous and parts may lack a formal specification, or the manual construction of a model is simply unfeasible.

Our approach aims to address this issue by automatically inferring a model of a network in a black-box fashion, by observing its behaviour. The goal is to provide a modular framework for learning and verifying networking systems, based on the NetKat language, which provides a compositional programming abstraction of networking behaviour. The envisioned framework is shown in the diagram: the learner interacts with the system via queries, aiming to gather observations (membership query) and to check the correctness of the model (equivalence query); the oracle includes a verification component, which compares the current model against NetKat policies, allowing one to check for specific properties incrementally, as the model is being learnt.



The main objectives for this project are taming the complexity of networks and achieving scalability of learning and verification. One of the key challenges is the fact that NetKAT models are non-deterministic, hence they lack canonical representatives; this hinders convergence of learning. To tackle this issue, we have investigated a general framework that allows deriving canonical representatives for a wide class of non-deterministic systems. One journal paper [2] is being finalised; another paper to be submitted to a top conference is in preparation.

PUBLICATIONS. [1] Learning weighted automata over principal ideal domains. Gerco van Heerdt, Clemens Kupke, Jurriaan Rot and Alexandra Silva. FOSSACS 2020. [2] S. Zetzsche, A. Silva, and M. Sammartino, "Bases for algebras over a monad", arXiv preprint arXiv:2010.10223, 2020.

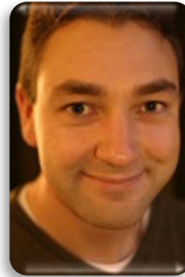
RELATED GRANTS. EPSRC Standard Grant "Verification of Hardware Concurrency via Model Learning (CLeVer)" (EP/S028641/1), £693K.

IMPACT STATEMENT. "The completeness, fidelity, and trustworthiness of models is an important challenge for Arm. Arm is highly interested in the development of techniques that offer the potential to make the design of these models more automatic - both tools that provide a design aid for human designers, and tools that automate the modelling process altogether."
– Dominic Mulligan, Staff Research Engineer, Arm Research –

VeTSS 2019/2020

Higher-Order Program Invariants and Higher-Order Constrained Horn Clauses

Steven Ramsay
University of Bristol



Verifying performance impacts of micro-architecture vulnerability mitigations

David Aspinall
University of Edinburgh

Mechanising the Theory of Build Systems

James McKinna
University of Edinburgh

Reliable High-Level Synthesis

John Wickerson
Imperial College London

Fluid Session Types: End-To-End Verification of Communication Protocols

Nobuko Yoshida
Imperial College London

Generalised static checking for type and bounds errors

Stephen Kell
University of Kent

Formal Verification of Information Flow Security for Relational Databases

Andrei Popescu
Middlesex University London

Persistent Safety and Security

Brijesh Dongol
University of Surrey

Higher-order Program Invariants and Higher-order Constrained Horn Clauses

Steven Ramsay Luke Ong

- Higher-order programming is gaining traction, especially in financial and scientific industries.
- There is no consensus around a mathematical foundation for higher-order program verification.
- We propose higher-order constrained Horn clauses as a unifying logical paradigm.
- We look to exploit logical techniques to provide practical program verification technologies.
- We presented a highly efficient pattern-match safety analysis based on logical resolution and verified a number of widely-used, open source Haskell libraries.

First-order program verification benefits enormously from a shared lexicon of notions of program invariant, such as inductive invariants and procedure summaries. In contrast, there are no generally accepted notions of invariant in the higher-order verification literature. The goal of this studentship is to initiate the development of a unifying theory of higher-order program invariants. A theory which can express the common, logical underpinnings of the subject and yet supports effective and reusable automated reasoning tools.

In the last year, we have developed a new compositional reasoning technique for formally verifying that a given functional program is free from pattern-match safety exceptions, based on a restricted form of logical resolution. Such exceptions can arise whenever a program does not handle all the cases that can arise in practice, such as data arriving in an unexpected format or when a defective component interacts using a wider variety of actions than are advertised in its protocol.

```
[1 of 1] Compiling Main             ( test/PaperExamples.hs, /home/stersay/projects/int
ensional-constraints/dist-newstyle/build/x86_64-linux/ghc-8.8.3/intensional-datatys-0.2
.0.0/t/test/build/test/test-tmp/Main.o )

test/PaperExamples.hs:(73,1)-(76,56): warning: [Intensional]
  Could not verify that 'Not' from test/PaperExamples.hs:60:21-34
    cannot reach the incomplete match at test/PaperExamples.hs:(32,1)-(35,39)
```

The technique is designed to be lightweight and usable by the average programmer. Our prototype tool is fully automatic, but we have avoided the "push a button and hope for the best" user-experience that can hinder adoption by giving strong guarantees on predictability:

- compositionality ensures the running time of our analysis scales linearly in the size of the analysed program and, in practice, large Haskell packages can be verified in a few hundred milliseconds.
- the power of the analysis is characterised by an intuitive extension of the Haskell type system, so programmers are readily able to understand why a component may not be verifiable.

Publications. E. Jones and S. Ramsay. 2021. Intensional datatype refinement with application to scalable verification of pattern-match safety. POPL 21.

Related grants. Higher-order Constrained Horn Clauses: A New Approach to Verifying Higher-order Programs. (EPSRC EP/T006595/1).

Impact. Our tool is packaged as a GHC plugin and available on the Hackage package repository at https://hackage.haskell.org/package/intensional-datatys. It has been used to verify that a number of widely-used, open-source Haskell libraries (such as aeson, containers, and time) are free from pattern-match safety exceptions.

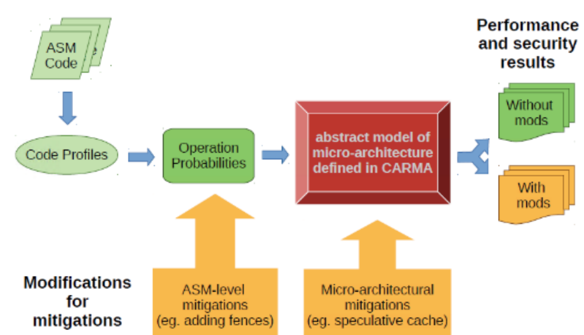
WHERE SOFTWARE MEETS HARDWARE: VERIFYING PERFORMANCE IMPACTS OF MICRO-ARCHITECTURE VULNERABILITY MITIGATIONS

DAVID ASPINALL VASHTI GALPIN

- Transient execution vulnerabilities (such as Spectre) take place at micro-architecture level in out-of-order processors with speculation.
- Attacks leveraging these vulnerabilities can undermine the security of all system components built on top of a susceptible micro-architecture.
- Mitigations for these vulnerabilities can be expensive in terms of loss of computing cycles due to a reduction in speculation.
- Formal modelling of micro-architecture behaviour together with performance analysis is needed
- We present a cycle-level model of a generic out-of-order processor with speculation in CARMA, a quantitative modelling language with Markov chain semantics.

We have developed a CARMA model of micro-architecture that includes pipeline details and hardware speculation. We have developed a very generic ISA (instruction set architecture) that consists of instructions that are similar to those found in RISC processors (such as ARM) and to the micro-ops that are obtained from the ISA in x86 systems. This enables us to work with essentially a RISC ISA and to abstract from the more complex ISA of x86. This model takes as input an abstract stream of instructions which indicates the dependence of an instruction on the results of previous instructions. This allows for input to be created from profiles of actual code rather than working with the code directly, and supports abstraction of program characteristics hence allowing for performance analysis over a family of programs rather than specific executions. To ensure that attacks and mitigations can be modelled, the ROB (reorder buffer) is an explicit component of the model. This gives sufficient micro-architectural details that Spectre can be modelled as well as mitigations based on memory fences and timing experiments can be conducted to understand performance. The model serves as a proof-of-concept for the proposed approach.

PUBLICATIONS. The model developed and the software tool to experiment with it can be found at https://homepages.inf.ed.ac.uk/vgalpin1/.



RELATED GRANTS. V Nagarajan, S Ainsworth, TC Grosser, D Aspinall, EPSRC Grant EP/V038699/1, Dijkstra's Pipe: Timing-Secure Processors by Design, £535,239.

IMPACT. "One class of vulnerabilities that we hope the Morello board will mitigate against is side-channel attacks, whilst minimising any impact on performance. In order for this to be fully assessed, we need to be able to reason about and model both security properties and performance at the micro-architecture level. The simulator that this proposal hopes to develop will be a valuable asset in this regard. We also welcome the longer term aim of developing a compositional approach to micro-architecture security, as deriving the properties of a full system from individual components will become increasingly important as systems and interconnects get increasingly complex."
– Matt Rivers-Latham, Senior Director of Operations, Arm Ltd. –

MECHANISING THE THEORY OF BUILD SYSTEMS

JAMES MCKINNA PERDITA STEVENS

- Build systems form part of the critical infrastructure of modern software development
- Unlike for compilers, there has been little formal modelling or verification of build systems
- This is a pilot project to explore the development of formal models of some existing systems (make, pluto) in an interactive theorem proving system
- The aim is to develop new conceptual, and formal foundations in this area

Build systems form part of the critical infrastructure of modern software development, but unlike compilers they have not been so much the focus of formal modelling or verification. Many users might be familiar with the Unix workhouse tool 'make', but notwithstanding superficial advances, progress beyond it has been slow. This 9-month pilot project aims to develop formal models of some existing systems (make, pluto) in an interactive theorem proving system, with a view to providing more secure foundations for future work in this area. The ultimate aim is to develop new conceptual, and formal foundations in this area, and use them to increase confidence in contemporary software engineering practices.

The project focussed on the pluto build system, with the aim of formalising of the basic algorithm as a collection of mutually inductive definitions, following McKinna et al. (2009) “Programming Reachability Algorithms in Coq”, which is unusual in being an approach to imperative program verification in a type-theoretic setting. An initial formalisation of the basic pluto algorithms and data-structures in Agda was created: this revealed many subtleties resulting from the impedance (mis)match between the set-theoretic/object-oriented model underlying pluto, and its realisation in constructive type theory. Further work was done on trying to match the object-oriented model underlying pluto with the functional decomposition identified in the Mokhov et al. “A la carte“ paper, with difficulties arising with respect to identifying a distinct “scheduler” component or scheduling policy in the pluto model.

Little progress has been made on the general abstract proof search model of build systems, and its relation to extensions of the ‘compiler forest’ model. In retrospect, the proposal was too ambitious in aiming to tackle this problem at this stage.

PUBLICATIONS. P. Stevens, “Connecting Software Build with Maintaining Consistency between Models: Towards Sound, Optimal, and Flexible Building from Megamodels”, Software and System Modelling (accepted).

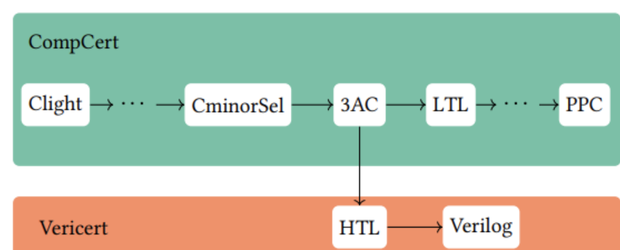
Reliable High-Level Synthesis

John Wickerson
Yann Herklotz

- The aim of this project is to make high-level synthesis (HLS) tools more reliable.
- Using fuzz-testing in the style of Csmith, we have demonstrated that existing HLS tools are not as reliable as previously thought.
- In response, we are developing a new HLS tool whose correctness is formally proven in Coq.

High-level synthesis (HLS) is the process of automatically translating a software program (written in, e.g., C or C++) into an equivalent hardware design (written in, e.g., Verilog) that can be implemented on a programmable chip (e.g. an FPGA). HLS is attractive because it promises to let software engineers reap the huge performance and energy-efficiency improvements that a custom hardware implementation can bring. This, together with the increasing power and availability of FPGAs (e.g. in AWS clouds) explains why HLS is rapidly growing in popularity. Conventional compilers have recently made great improvements in reliability thanks to rigorous testing (e.g. with Csmith and other fuzz-testers) and formal verification using a proof assistant (e.g. CompCert). Yet HLS has not received this attention. The aim of this project is to address that. We will build automatic fuzz-testers that will assess and improve the reliability of existing HLS tools, and we will design and implement a new, formally verified HLS tool that is bug-free by construction, thus setting a new standard for reliability in HLS.

On the testing front, we have fuzz-tested existing HLS tools such as Xilinx Vivado HLS and the Intel HLS Compiler using randomly generated C programs from Csmith, suitably restricted to the language fragment supported by those tools. This effort, led by MSc student Zewei Du and PhD student Yann Herklotz, showed that out of 6700 randomly generated test-cases, 1178 of them failed in at least one of the four tools that we tested. After reducing the test-cases to find their minimal forms, we discovered at least 8 unique bugs, 5 of which have been confirmed by the vendors, and 1 of which has now been fixed. This work emphasises the reliability shortcomings of current HLS tools, and thus motivates the second part of the project.



On the verification front, Herklotz has built a prototype HLS tool, called Vericert, fully verified in Coq. It is implemented by extending CompCert with a new hardware-oriented intermediate language and a new Verilog backend. Initial benchmarking suggests that Vericert generates hardware designs that are within an order of magnitude, in terms of performance and area-efficiency, of those generated by an existing state-of-the-art HLS tool, called LegUp. We intend to close this gap in the near future by implementing various optimisations in Vericert, such as operator scheduling and pipelined arithmetic.

Publications. [1] Z. Du, Y. Herklotz, N. Ramanathan, and J. Wickerson. Fuzzing High-Level Synthesis Tools. ACM/SIGDA 2021. (Poster) [2] A paper about the design and implementation of Vericert is under review for a top-tier programming languages venue.

Impact. "Dr Wickerson's interests mesh with [...] Xilinx's view that robust HLS tools are a technology critical to our future success. The proposal [...] to add support for Verilog as a CompCert target would be a significant contribution to the research community. Its principal value lies in the definition of the formal chain of correctness transformations unique to high-level synthesis compilers targeting hardware. Success in defining these transformations and proofs for even a small subset of C would provide a locus for research activity."
– Dr Samuel Bayliss, Principal Engineer, Xilinx Research Labs, San Jose. –

FLUID SESSION TYPES: END-TO-END VERIFICATION OF COMMUNICATION PROTOCOLS

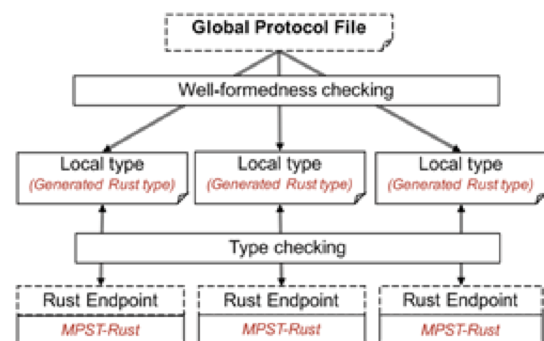
NOBUKO YOSHIDA

RUMYANA NEYKOVA

NICOLAS LAGAILLARDIE

- The first library, MultiCrusty, for multiparty programming in Rust. MultiCrusty leverages Scribble toolchain and Rusts type system for deadlock detection
- Formalisation of our implementation and proof of correctness, including a type system for affine channels with deadlock-freedom guarantees
- Performance evaluation shows MultiCrusty scales better than existing stats-of-the-art implementations

The advantage of message-passing concurrency is well-understood: it allows cheap horizontal scalability at a time when technology providers have to adapt and scale their tools and applications to various devices and platforms. Multiparty session types (MPST) is a type-based discipline that ensures that concurrent and distributed systems are safe by design. It guarantees that message-passing processes following a predefined communication protocol, are free from communication errors and deadlocks. Rust is a particularly appealing language for the practical embedding of session types. Its affine type system allows for static typing of linear resources -- an essential requirement for the safety of session type systems. Despite the interest in the Rust community for verification techniques handling multiple communicating processes, the existing Rust implementations are limited to binary (two-party) session types.



We have designed and implemented multiparty session types in Rust, visualised on the right. Our design follows a state-of-the-art encoding of multiparty into binary session types. We generate local types in Rust, utilising the Scribble toolchain. Our library for MPST programming in Rust, MultiCrusty, is implemented as a thin wrapper over an existing binary session types library. Differently from other MPST implementations that check linear usage of channels at runtime, we rely on the Rust affine type system to type-check MPST programs. Ours is the first formalisation of a multiparty session type system for affine processes. In addition, as we generate the local types from a readable global specification, errors caused by an affine (and not linear) usage of channels, a well-known limitation of the previous libraries, are easily avoided. Our library is available here.

PUBLICATIONS. [1] F. Zhou, F. Ferreira, R. Hu, R. Neykova, N. Yoshida. Statically Verified Refinements for Multiparty Protocols. OOPSLA'20. [2] N. Lagaillardie, R. Neykova, N. Yoshida. Implementing Multiparty Session Types in Rust. Coordination'20.

RELATED GRANTS. Nobuko Yoshida, PI, EPSRC Standard Grant: "Session Types for Reliable Distributed Systems", 10/2020-09/2024, £697,651; W Vanderbauwhede, PI (Nobuko Yoshida, Co-I) EPSRC Standard Grant: "AppControl: Enforcing Application Behaviour through Type-Based Constraints", 09/2020-06/2024, £1.4M.

IMPACT. "One core challenge at Actyx is to give average programmers and automation engineers software tools for successfully digitising this world's factories. Achieving this in a safe and modular way requires a behavioural typing discipline for static protocol verification such as the one Nicolas, Rumyana, and Nobuko develop and refine in close collaboration with us."
– Dr. Roland Kuhn, CTO & co-founder of Actyx AG –

GENERALISED STATIC CHECKING FOR TYPE AND BOUNDS ERRORS

STEPHEN KELL

- Much critical code continues to be written in C and other lower-level languages that include unsafe operations. These can lead to security vulnerabilities.
- Existing approaches to gain assurance tend to be inflexible, targeting very specific scenarios.
- This project seeks ways to increase flexibility, while retaining the accessibility and `explainability' of dynamic tools.
- We built a prototype tool built atop Frama-C and KLEE which can slice C code down to smaller programmes which are in some cases loop-free and therefore symbolically execute to termination.

Critical code continues to be written in C and other lower-level languages that include unsafe operations, such as unchecked array accesses and pointer casts, leading to security vulnerabilities. Many approaches exist for gaining assurance about these, but each is inflexible and target specific scenarios.

Static type-checking can be viewed as a program analysis that is baked into the host language design. Conversely, dynamic checking can be made `partially static' by integrating it into a symbolic execution engine. This project explored this continuum in search of a *configurable* program analysis of which traditional static type-checking is just one configuration. We framed the problem as seeking methods that would allow a relatively dynamic approach to become terminating, hence `static', given the right program abstraction. Our insight is that program slicing offers such an abstraction; it retains or discards code based on a slicing criterion, which in our case is "Does this code affect whether the type assertions will pass"? For syntactically type-checkable programs, only a loop-free program should remain, roughly encoding "Might this raise a type error?". Syntactic type assignments are loop-invariant by definition, and indeed this is how syntax-directed type-checking scales; in effect, we recover this fact during slicing, rather than enforcing it by construction in the language design.

We built a tool, Slice & Run, which encodes type-based properties as assertions which can be interpreted by the KLEE symbolic execution engine. Before feeding the program to KLEE, we apply a slicing-based abstraction whose goal is to recover a smaller version of the program whose exploration by KLEE terminates /in those cases that would also be statically type-checkable/, but is still (non-terminatingly) useful in those that would not. The tool is built using Frama-C and takes unmodified C code as input.

Besides considerable basic tool-building effort, the main challenge addressed in the 13 months of the project has been to accommodate patterns of polymorphic code. This requires careful design of the *invariant protocol* i.e. the rules for inserting assertions. This exercise mirrors the design of the language-level type-checking rules, but is separated from the base language, so can likely be applied largely independently of the source language. This contrasts with conventional type-checking innovations, whose benefits are conferred only on new source code written in new languages.

Symbolic execution is itself a technique offering a configurable depth of analysis, profiting from much complementary work. Meanwhile, it is likely that even programs that do not slice perfectly would benefit opportunistically from slicing, yielding greater coverage per unit time.

Were the project able to continue, we would be exploring the benefits and practicalities around these approaches.

PUBLICATIONS: J. Adam, S. Kell. Type checking beyond type checkers with Slice & Run. TAPAS workshop at SPLASH 2020.

FORMAL VERIFICATION OF INFORMATION FLOW SECURITY FOR RELATIONAL DATABASES

The University Of Sheffield.

UNIVERSITY OF TWENTE.

ANDREI POPESCU

PETER LAMMICH

- Our society relies increasingly on systems that handle sensitive information by storing it in databases and offering selective access to it.
- Expressive policies and mechanisms are needed to prevent information leaks while not inhibiting legitimate flows.
- In this project, we formalise a framework for reasoning about the information flow security of web-based database-backed systems.
- Our objective is the creation of concepts and tools for specifying and formally proving fine-grained policies, addressing confidentiality needs of real-world systems.

In this project, we design and implement a framework for reasoning about the information flow security of user interactions with relational databases–where the possible interactions are prescribed by input/output reactive programs that query and update the data.

An important characteristic of this framework is that its policy language is extremely flexible, allowing to express not only the absence of certain flows, often captured by variants of non-interference, but also controlled release of information, also known as declassification, in a very fine-grained manner. For example, in an employee database one could be allowed to query the employee salary field, but only collectively, and nothing should be made available beyond the average salary. As another example, a medical insurance agent should be able to query a medical database for aspects of a patient's treatment, but only in a manner that does not reveal whether the patient suffers from a terminal illness. Another characteristic of the framework is the support for both fully automatic reasoning when possible and partly interactive reasoning when necessary, using a compositionality infrastructure developed in the proof assistant Isabelle/HOL.

We validate the framework on two case studies of secure-by-design systems: CoCon, a conference management system, and CoSMed, a social media platform. These systems have been previously verified for confidentiality in an ad hoc manner. Using our framework, we further automate and uniformise their verification code base.

PUBLICATIONS. [1] A. Popescu, P. Lammich, P. Hou. CoCon: A Conference Management System with Formally Verified Document Confidentiality. Journal of Automated Reasoning 2021. [2] A. Popescu, T. Bauereiss, P. Lammich. Bounded-Deducibility Security. ITP 2021 (to appear). [3] A paper, reporting on the formalisation of a sizable fragment of the SQL standard in the Isabelle/HOL proof assistant, is under preparation.

IMPACT. The guest editorial in the special issue of the Journal of Computer Security (volume 25, Issue 4-5) dedicated to verified information-flow security notes: "The past few years have seen the seeds of information flow security sown in the preceding three decades bear practical fruit. A number of real-world systems with formally verified guarantees of information flow security now exist." and goes on to cite nine such systems, two of which are the verified web applications CoCon and CoSMed.

Persistent Safety and Security

Brijesh Dongol **Francois Dupressoir** **John Derrick**

- Computing architectures have recently shifted towards new persistent or non-volatile memory technologies (NVRAM), whose state is preserved even in case of a system crash or loss of power.
- NVRAM closes the latency gap between different forms of storage and has the potential to vastly improve system speed and stability in systems ranging from personal devices to large data clusters.
- Rebooting from persistent memory can leave the system in an unsafe/insecure state, as persistent memory writes are controlled by the system and the programmer, introducing several research challenges.
- Information must be persisted in the correct order (in case of a system crash), and moreover, any secrets stored in persistent storage must not be available to unauthorised parties.
- Our focus is the rigorous development of concurrent programs in systems that use persistent memory.

Our main aim is to *develop correctness conditions, programming abstractions and verification methods that enable developers to build safe and secure persistent memory systems.* To support scalability, we focus on verified libraries implementing concurrent objects, which form basic software components developers can re-use. To future-proof our results, we conduct our research in the context of cutting-edge techniques on weak memory semantics and associated verification methodologies (including mechanisation in the Isabelle/HOL theorem prover), software libraries (including concurrent objects and transactional memory), and security verification techniques. We work across the hardware/software interface and consider the impact of low-level read/write primitives on high-level concurrency abstractions. In particular, the work is being conducted in the context of weak memory models for both programming languages (C/C++11) and hardware (Intel-TSO, ARM).

To better understand the atomicity requirements and associated safety properties for NVRAM-based systems, we have developed verification methods for concurrent objects (e.g., stacks, queues). For software transactional memory, we have developed a new notion of atomicity (called durable opacity) and the associated verification techniques. We are developing high-level logics for reasoning about systems that combine persistency with relaxed accesses from weak memory, forming the first step towards program verification. We are also investigating the interaction between concurrency abstractions and hyper-properties, including within our recent VETSS small grant. Our current works are mechanised within the Isabelle/HOL and KIV theorem provers.

Publications. [1] J. Derrick, S. Doherty, B. Dongol, H. Wehrheim, G. Schellhorn: Verifying Correctness of Persistent Concurrent Data Structures: A Sound and Complete Method. Formal Aspects of Computing (2021). [2] E. Bila, J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, H. Wehrheim: Modularising Verification of Durable Opacity. (under review). [3] E. Bila, S. Doherty, B. Dongol, J. Derrick, G. Schellhorn, H. Wehrheim: Defining and Verifying Durable Opacity: Correctness for Persistent Software Transactional Memory. FORTE 2020, best paper award.

Related grants. VeTSS small grant on Transactional Memory Security (£30k). EPSRC Cross Research Institute Grant on Verifiably Correct Swarm Attestation (£514k): Brijesh Dongol (PI), Santanu Dash (co-I), Helen Treharne (co-I) and Liqun Chen (co-I), with project partners Arm, Thales, NTU, and SRI.

Impact. Software correctness and security are two fundamental properties that need to be addressed as systems transition to using Non-Volatile Memory (NVM). While program logics, design patterns, and libraries exist for traditional weak-consistent memory, these techniques have to be revised and adapted in the context of NVM. Arm believes that progress on methodologies to guarantee correctness and security of NVM programs is of paramount importance to realize the benefits of NVM.
– Gustavo Petri, Staff Research Engineer, Arm Research –

VeTSS 2020/2021

Aion: Verification of Critical Components' Timely Behaviour in Probabilistic Environments

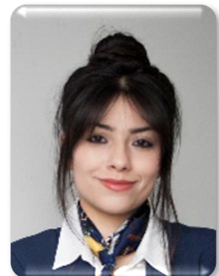
Vincent Rahli
University of Birmingham

Mechanising Concurrent WebAssembly

Neel Krishnaswami
University of Cambridge

CONVENER: Continuous Verification of Neural Networks

Ekaterina Komendantskaya
Heriot Watt University



Validating the Foundations of Verified Persistent Programming

Azalea Raad
Imperial College London

Fixing the Thin-Air Problem: ISO Dissemination

Mark Batty
University of Kent

Quantitative Algebraic Reasoning for Hybrid Programs: reasoning precisely about imprecisions

Alexandra Silva
University College London

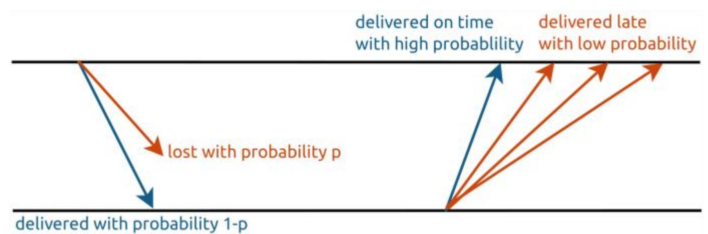
AION: VERIFICATION OF CRITICAL COMPONENTS' TIMELY BEHAVIOR IN PROBABILISTIC ENVIRONMENTS

VINCENT RAHLI DAVID PARKER

- Distributed critical information infrastructures need to behave correctly and in a timely fashion, while tolerating faults and attacks.
- We use the Coq proof assistant to prove the correctness of such systems, in particular their timeliness.
- We are developing models and proof methods to verify timing properties of such systems.
- We are using these tools to verify correctness of real-time Byzantine fault-tolerant broadcast protocols.

Our society strongly depends on distributed critical information infrastructures such as electrical grids, autonomous vehicles, blockchain applications, IoT critical infrastructures, etc.. Such systems need to behave correctly, even in the face of faults and attacks. For time-critical systems, one correctness criteria is that operations are performed within a certain time bound. For example, in the context of a SCADA system, a desirable property is that actuators receive commands reliably and punctually.

One state-of-the-art technique to ensure that services in general are secure against faults and attacks is to use Byzantine Fault Tolerant State Machine Replication (BFT-SMR) that replicates a service and masks faults behind the behaviour of correct replicas. Some protocols have been proposed that can provide timing guarantees in synchronous environments (where message transmission delays are bounded by a known bound), while others work in asynchronous environments (where message transmissions are unbounded) but do not provide timing guarantees. We have developed in [1] a collection of protocols that provide timing guarantees in probabilistic synchronous networks, where messages have a low probability of being delivered late. We have informally proved that these protocols satisfy timeliness properties, in addition to standard BFT properties.




```
Lemma ex_2bcats :
  (* [lbound] is the maximum number of steps that we allow ourselves *)
  exists (lbound : nat),
    (* The property below holds for all all bounds between [lbound] and [maxTime],
       where [maxTime] is the parameter that bounds the maximum number of steps
       allowed, necessary for finite probabilities *)
    forall (n : nat),
      (maxTime > n)%nat ->
      (n > lbound)%nat ->
      (* The probability of 2 broadcasts being delivered after [n] steps of computation
         is greater than the probability of those message not getting lost *)
      ((1 - LostProb)^2
       <= Pr (steps2dist n two_bcasts_intransit) (finset.set1 true))%R.
```

Many formal theory tools have been developed to verify the correctness of BFT-SMR protocols. However, none support the kind of protocols developed in [1]. Therefore, as part of this project we are developing within the Coq theorem prover, models and verification techniques to guarantee the correctness of real-time BFT-SMR protocols, and in particular, to prove timeliness properties in probabilistic synchronous environments. Towards this goal we have developed a probabilistic model of distributed computations, where messages are assigned a probability of getting lost or delayed, and where a faction of the nodes can behave arbitrarily (i.e., are Byzantine). We are now using this model to verify properties of distributed systems, some of which only hold with high probability. In particular, we are now using it to verify properties of the real-time Byzantine fault tolerant protocols developed in [1], such as timeliness.

PUBLICATIONS. D. Kozhaya, J. Decouchant, V. Rahli, and P. Verissimo. PISTIS: An Event-Triggered Real-Time Byzantine-Resilient Protocol Suite. IEEE Transactions on Parallel and Distributed Systems, 2021

MECHANISING CONCURRENT WEBASSEMBLY

NEEL KRISHNASWAMI CONRAD WATT JEAN PICHON

- WebAssembly is the first new programming language to be supported on the Web in over 20 years.
- We have developed WasmCert-Coq, a new mechanisation of the WebAssembly language in Coq.
- We mechanise WebAssembly's linking and allocation phase for the first time.
- We investigate WebAssembly's proposed relaxed memory concurrency model.

WebAssembly is a new language supported by all major Web browsers, intended to be an efficient compilation target for low-level languages such as C++ and Rust, with the aim that the resulting compiled program can be executed in the browser with near-native performance. WebAssembly is unusual in that its standards body maintains a full, normative formal specification for the language, and all features must be fully formalised before they are accepted as extensions to the standard. WebAssembly was initially single-threaded, but an in-progress extension to the language introduces threads and concurrent memory operations, including atomic accesses. Concurrent access to memory in WebAssembly can give rise to "relaxed memory" behaviours, allowing program results that cannot be explained by considering only naive sequential interleavings of the operations of individual threads. The WebAssembly concurrency proposal includes a formal model of the permitted relaxed memory behaviour. This project aims to investigate areas of the WebAssembly semantics which interact with the proposed concurrent extension, and develop a Coq mechanisation of the language to complement the existing Isabelle mechanisation, which was based on an earlier draft of the WebAssembly semantics.

We have completed a Coq mechanisation of WebAssembly's core sequential semantics, and extended both the Coq and Isabelle mechanisations to model “instantiation” for the first time. Instantiation is a linking and allocation phase of the WebAssembly program lifecycle not described in the initially published semantics.

The WebAssembly concurrency proposal does not include a mechanism for thread creation. Instead, WebAssembly is embedded within a "host language" (such as JavaScript on the Web) which is responsible for the creation of threads running WebAssembly code. As an initial step towards mechanising this behaviour, we mechanised an abstract host language which is intended to capture the capabilities of JavaScript's WebAssembly API. We are currently investigating integration with the Iris framework (Jung et al, ICFP 2016), which facilitates the specification of higher-order concurrent program logics.

We discovered two deficiencies in WebAssembly's concurrency model. First, WebAssembly and the related JavaScript model were incorrectly specifying the atomic compare-exchange operation, making its synchronisation guarantees too strong in the case that the comparison fails. This issue was corrected in both specifications. More seriously, we discovered that the intended compilation scheme from concurrent C++11 to WebAssembly is formally unsound due to an issue with the WebAssembly model related to the notorious "out-of-thin-air" problem. Luckily, this issue is primarily theoretical, as no real system is expected to exhibit thin-air executions.

IMPACT. As interest grows around WebAssembly as a bytecode format for smart contracts, we're excited about WasmCert-Coq and hope to use it in future verification efforts!
– Vilhelm Sjöberg, Research Scientist at CertiK –

With WebAssembly, we strove to design a high-assurance platform, which includes a specification with a formal semantics. Mechanisation is vital both for building confidence in this specification and its proposed extensions and for formal reasoning about WebAssembly programs. I'm very pleased that this work has brought new parts of the WebAssembly semantics into Coq.
– Andreas Rossberg, WebAssembly Specification Editor –

CONVENER: CONTINUOUS VERIFICATION OF NEURAL NETWORKS

EKATERINA KOMENDANTSKAYA
DAVID ASPINALL

- Conflict between continuous methods (that enable data classification in multi-dimensional real space) and discrete methods (used by solvers and provers) poses a challenge for neural network (NN) verification
- This project aims to turn this from a disadvantage to a capability
- A continuum of models can serve as suitable classifiers for NNs and give reasonable prediction accuracy.
- Given the task of verifying a neural network, we are no longer required to think of the object as immutable, i.e. we are allowed to verify and deploy a different NN instead.

Most challenges encountered in neural network verification are due to the conflict between continuous and discrete methods. Conventionally, we assume that the object we verify is uniquely defined, often hand-written, and therefore needs to be verified as-is. Neural networks are different—often there is a continuum of models that can serve as suitable classifiers, and we usually do not have much preference for any of them, as long as they give reasonable prediction accuracy. Given the task of verifying a neural network, we are no longer required to think of the object as immutable, i.e., we are allowed to verify and deploy a different neural network instead.

This opens up new possibilities for verification and justifies several methods of NN transformation, including NN size reduction, piece-wise linearisation of activation functions either during or after training, and use of constraint-aware loss functions during training or interleave verification with adversarial training, which improves NN safety. Thus, verification becomes part of the object construction. We also assume that the training-verification cycle may repeat potentially indefinitely, especially if NNs are retrained using new data. We call this approach *continuous verification*.

However, to be truly successful, this methodology needs proper programming language support. Ideally, the programmer should only need to specify basic neural network parameters and the desired verification constraints, leaving the work of fine-tuning of the training-verification cycle to the integrated tools.

We cast a type-theoretic view on these problems, and have conducted our first successful experiments at building a verification infrastructure based around these ideas. For this, we initially used F* and Liquid Haskell, functional languages with refinement types. We have also explored different ways of integrating neural network verification into other mainstream languages (e.g., our Z3 verifier that works with TensorFlow models in Python and the Agda extension that allows us to perform neural network verification via refinement type

In the second half of the project, we explored the problem of NN verification as program synthesis. We performed an in-depth comparison of approaches to improving NN robustness, including their relationship, assumptions, interpretability and after-training verifiability. We also looked at constraint-driven training, a general approach designed to encode arbitrary constraints, and showed that not all of these definitions are directly encodable. Finally, we performed experiments to compare applicability and efficacy of the training methods at ensuring the network obeys these different definitions. These results highlight that even the encoding of such a simple piece of knowledge, such as robustness of an NN, training is fraught with difficult choices and pitfalls.

PUBLICATIONS. [1] W. Kokke, E. Komendantskaya, D. Kienitz, R. Atkey, and D. Aspinall. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS’20

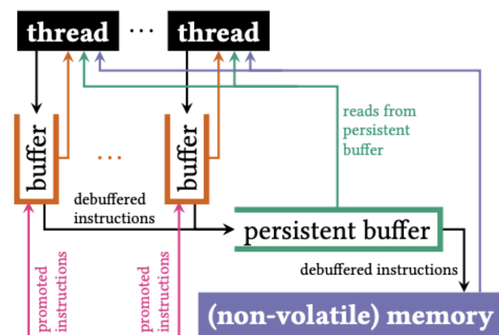
VALIDATING THE FOUNDATIONS OF VERIFIED PERSISTENT PROGRAMMING

Imperial College London

AZALEA RAAD JOHN WICKERSON

- Non-volatile memory (NVM) is fast becoming mainstream.
- Therefore, we need to be able to write programs that exploit it, correctly and efficiently.
- In order to verify that these programs are correct, we need formal models of how CPUs interact with NVM, able to answer questions like "is my data guaranteed to become persistent in the same order as it is written to memory?"
- This project will design and implement techniques for empirically validating these models against mainstream architectures such as Armv8 and Intel x86.

The emergence of non-volatile memory (NVM) is expected to revolutionise how software is written. NVM provides storage persistence across power failures, yet offers performance close to that of traditional (volatile) memory. As such, programs that require data persistence (e.g. databases) can achieve an orders-of-magnitude lower latency by storing their data on NVM rather than on standard disks. It is widely believed that NVM will supplant volatile memory entirely, allowing efficient access to persistent data.



However, writing *correct* persistent programs is a difficult task. A key challenge for the programmer is correctly accounting for the "persistency semantics" of the hardware, which describes the order in which writes performed by the CPU become persistent. Volatile caches between the CPU and the NVM can make persistency semantics quite counterintuitive, and motivates the need for formal verification.

Prior work has developed models of how persistency works in mainstream hardware such as Arm and Intel-x86. These models have been developed by studying specifications and interviewing designers, but have not yet been validated empirically on real-world hardware.

This project aims to address that shortcoming. Validating persistency models on hardware is challenging because in order to test whether data is truly persistent we need to power-cycle the test machine, yet doing so for every test-case would led to infeasibly low testing throughput. To get around this difficulty, we have obtained a specialised hardware device called a "DDRDetective", which is able to monitor the data traffic to and from the persistent memory while our test-cases are being executed. Work is expected to start imminently on using the DDRDetective to validate the persistency models for the Armv8 and Intel-x86 architectures.

RELATED GRANTS. This project is also being partially funded by the EPSRC Programme Grant "IRIS: Interface Reasoning for Interacting Systems" (£6.1M, 2018–2023).

IMPACT. “The correctness of programs that exploit non-volatile memory is an important challenge to Arm. The formal semantics for how persistent memory works will be key in addressing this challenge. Practical methods to run litmus test for persistent memory empirically on real chips would help validate microarchitectural implementations of Arm persistency models."
– Andrea Kells, Director, Research Ecosystem, Arm Ltd. –

FIXING THE THIN-AIR PROBLEM: ISO DISSEMINATION

MARK BATTY

SIMON COOKSEY

- Concurrency in the C++ language is ill-specified in the current International Standards Organisation (ISO) definition: it allows values to be conjured out of thin air (OOTA). Working towards verification of C++ code, we fixed this problem with a proposed change to the standard, called Modular Relaxed Dependencies (MRD).
- The ISO has acknowledged our approach as its best path forward to fix the problems of C and C++.
- COVID-19 interrupted ISO meetings quite severely. We pivoted in this project, reallocating travel resources that could not be spent to equipment for remote and future in-person demos, and hiring Abigail Pattenden to develop a web-based interface to our executable model, MRDer, for remote experimentation by committee members.
- Pattenden, Cooksey, Batty attended ISO meetings, presenting MRDer to the concurrency subgroup.
- Following this project, we secured a larger grant for studying C semantics.

The path taken in this project was radically different than proposed because of COVID-19. When ISO meetings were cancelled, we pivoted from planning in-person travel to the development of online interactive tools. We have previous experience of making changes to the ISO definitions of C and C++, and instrumental to our changes was the CPPMem tool. CPPMem allows one to explore the behaviour of small test programs online, under the C++ concurrency specification. ISO committee members use CPPMem to answer their own questions about corner cases of the concurrency specification.

We have exposed further problems in the concurrency specification of C++, problems that require more fundamental change. MRD is a possible solution to those problems, and ISO SG1 voted to pursue this direction. To present the finer details of MRD at a time with no in-person meeting, we developed an online variant of our new model, following the familiar style of the previous CPPMem tool (see above). The tool itself is available at https://www.cs.kent.ac.uk/projects/MRDer/.

RELATED GRANTS. Mark Batty, PI; S. Kell, Co-I. UKRI DSbD grant “CapC: Capability C semantics, tools and reasoning” (EP/V000470/1), 08/2020-07/2023, £485,168.

IMPACT. The ISO unanimously passed the following motion endorsing our semantics: “OOTA is a major problem for C++, modular relaxed dependencies is the best path forward we have seen, and we wish to continue to pursue this direction”

Quantitative Algebraic Reasoning for Hybrid Programs: reasoning precisely about imprecisions

Alexandra Silva

Fredrik Dahlqvist

Renato Neves

- It is usually insufficient to tell whether two programs are equivalent or not.
- In cyber-physical programming, particularly, it is important to tell “how close” are two programs of being equivalent to each other.
- Our approach uses algebraic reasoning mechanisms to achieve precisely that.
- Applications of our work include real-time and probabilistic programs.

Championed as a core ingredient of the twenty-first century technology, cyber-physical systems intertwine digital computation with continuous, physical processes and possess a wide range of application domains. They are found not only in small medical devices, such as pacemakers and insulin pumps, but also in networks of autonomous vehicles and district-wide electric grids. They are also used in the analysis of biological mechanisms such as disease propagation and personalised treatments against cancer. The presence of physical processes, however, hinders an effective use of classical programming notions and techniques. A most basic case is the classical notion of program equivalence, which in the presence of continuous behaviour becomes too strict, for it requires that two programs behave in exactly the same way for a continuous range of possible values.

Whilst there have been recent important developments in cyber-physical program semantics there are currently no algebraic methods for telling `how close' are two programs of being equivalent to each other. This is where our project aims to make progress. Specifically, we aim at providing useful notions of distance between cyber-physical programs and subsequently developing corresponding axiomatisations for reasoning about approximate program equivalence.

We are targeting two specific families of cyber-physical programs: real-time and probabilistic programs. In the first case it is useful to tell how close are the execution times of two programs, for instance in the setting where time constraints and synchronisation mechanisms are a main concern (e.g. cruise control systems and platoon vehicles). In the second case it is useful to tell how close are the probabilities of two programs producing a specific output, with applications in e.g., machine learning and noise-aware quantum programming.

Our current results include a complete, generic algebraic system for reasoning about approximate equivalence that is applicable to both real-time and probabilistic programs.

Publications. Two papers are in preparation, to be submitted to top conferences.

Impact. This project proposal is concretely motivated by R. Neves' participation in the scientific projects DaVinci and Klee, which have extensive industrial collaborations. The former concerns coordination of components in the cyber-physical domain and collaborates with the Belgian company Altreonic (specialised in the development of vehicular systems). The latter concerns synthesis and analysis of biological systems via hybrid programming techniques, and collaborates with Silicolife (a company specialised in industrial biotechnology). Notions of approximate equivalence are critical in vehicular systems (for instance to compare the speeds attained by two cruise controllers) and in biological systems due to the presence of noise in sensors and actuators.

VeTSS 2021/2022

Verified Program Synthesis for Refactoring Rust Programs

Meng Wang
University of Bristol

Type-driven data-science infrastructure for Idris2

Ohad Kammar
University of Edinburgh

Neural Network Verification: in Search of the Missing Spec

Ekaterina Komendantskaya
Heriot Watt University

Safe and Reliable Concurrent Robotics Programming with Choreographies

Nobuko Yoshida
Imperial College London

Symbolic Computation for Mainstream Verification

Budi Arief
University of Kent

Research Institute in
Verified Trustworthy Software Systems
UK's second research institute in cyber-security

Contact us

Petar Maksimović
Academic Program Manager

Teresa Carbajo García
Administrative Program Manager

Research Institute in Verified Trustworthy Software Systems
Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ
United Kingdom

Phone: +44 (0)20 759 43140

E-mail: VeTSS@imperial.ac.uk

EPSRC
Engineering and Physical Sciences Research Council

National Cyber Security Centre
a part of GCHQ

Imperial College London