VeTSS

Research Institute in Verified Trustworthy Software Systems

UK's second research institute in cyber-security

Annual Report 2017/2018

EPSRC
Engineering and Physical Sciences
Research Council

National Cyber
Security Centre
a part of GCHQ

Imperial College
London

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

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) has given VeTSS £2.5 million over five years to support academic research projects in software analysis, testing and verification. This annual report provides a description of the projects funded from April 2017 to March 2018. In effect, most projects were funded for eight months from August 2017, due to the start date of VeTSS. This report demonstrates the deep connection between the VeTSS academic research and industry. For example, Livshits' and Donaldson's VeTSS project is related to an academic start-up of Donaldson (Imperial) that has recently been bought by Google. This report also describes how VeTSS funding has led directly to further funding by EPSRC, the EU and industry. For example, Yorsh's initial work on her VeTSS project led directly to her ERC Starting Grant for £1.5 million. There has already been some published work on the VeTSS projects but, in most cases, the work will be published in the coming year. This timeline for publication is normal, and we will update the report as the publications emerge.

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. We have held meetings at NCSC and MoD, arranging for Gernot Heiser (Data61, Australia) to speak at the NCSC about the verified microkernel seL4 and John Launchbury (Galois, USA) to speak at the MoD about the DARPA HACMS project on verified autonomous vehicles. We have been part of a successful bid for the Cambridge Newton Institute programme on "Verified Software" in 2020, organised by de Moura (Microsoft Redmond), Farzan (Toronto), Hoare (formerly Oxford and 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.

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

Professor Philippa Gardner
Director of VeTSS

MECHANISING THE METATHEORY OF SQL WITH NULLS

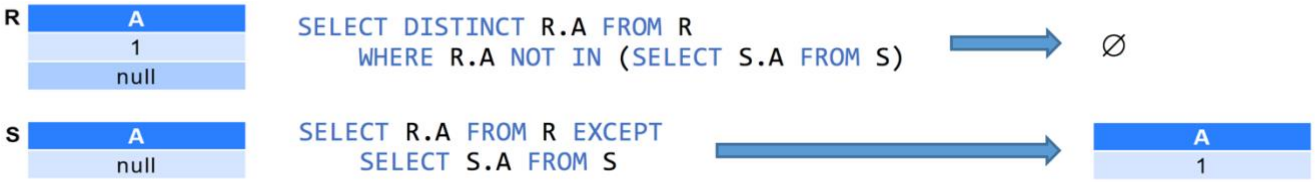
JAMES CHENEY WILMER RICCIOTTI

- SQL is the standard query language used by the multi-billion-dollar relational database industry
- SQL semantics is notoriously subtle: it is written in natural language and is 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, given the same input data, the same query can yield different results 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.



R

A
1
null

S

A
null

```
SELECT DISTINCT R.A FROM R
    WHERE R.A NOT IN (SELECT S.A FROM S)
```

∅

```
SELECT R.A FROM R EXCEPT
    SELECT S.A FROM S
```

A
1

Although there are some previous formalisations of SQL or relational query languages, all of them ignore null values, so they “prove” query equivalences that are unsound in the presence of these features. Our project builds on a recent (on-paper) formal semantics for SQL with nulls by Guagliardo and Libkin, providing the validation of key meta-theoretic properties in the Coq proof assistant. We view this as a first step towards a future in which query optimisers are *certified*. Our development can be publicly accessed at its GitHub repository (https://github.com/wricciot/nullSQL).

PUBLICATIONS. An article is to be submitted to a leading conference on verification.

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

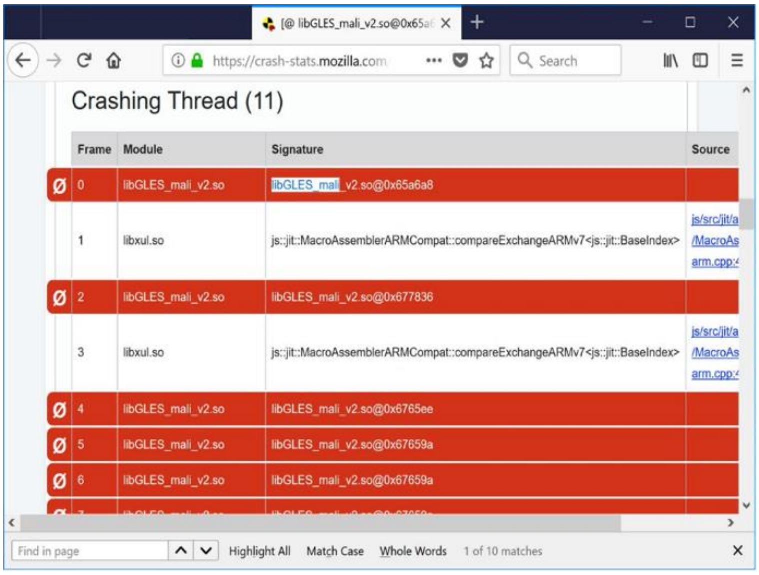
The research work undertaken on this project at Imperial College London led to the development of an automated approach to finding defects in web browsers using mutation-based structural fuzz testing. The investigators decided to focus on testing components of web browsers related to high-performance 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. We have not yet published work on the results, but the work undertaken so far 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, with more details available 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 recently acquired by Google and has since been open-sourced (https://github.com/google/graphicsfuzz). Open-sourcing of the VeTTS project and potential integration with the GraphicsFuzz code base will further the impact potential of the project.

PUBLICATIONS. Several articles are in preparation to be submitted to leading conferences in the field.

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 Vulnera-bilities 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 –



Crashing Thread (11)

	Frame	Module	Signature	Source
Ø	0	libGLES_mali_v2.so	libGLES_mali_v2.so@0x65a6a8	
	1	libxul.so	js::jit::MacroAssemblerARMCompat::compareExchangeARMv7<js::jit::BaseIndex>	js/src/jit/a /MacroAs arm.cpp:4
Ø	2	libGLES_mali_v2.so	libGLES_mali_v2.so@0x677836	
	3	libxul.so	js::jit::MacroAssemblerARMCompat::compareExchangeARMv7<js::jit::BaseIndex>	js/src/jit/a /MacroAs arm.cpp:4
Ø	4	libGLES_mali_v2.so	libGLES_mali_v2.so@0x6765ee	
Ø	5	libGLES_mali_v2.so	libGLES_mali_v2.so@0x67659a	
Ø	6	libGLES_mali_v2.so	libGLES_mali_v2.so@0x67659a	

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

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 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.

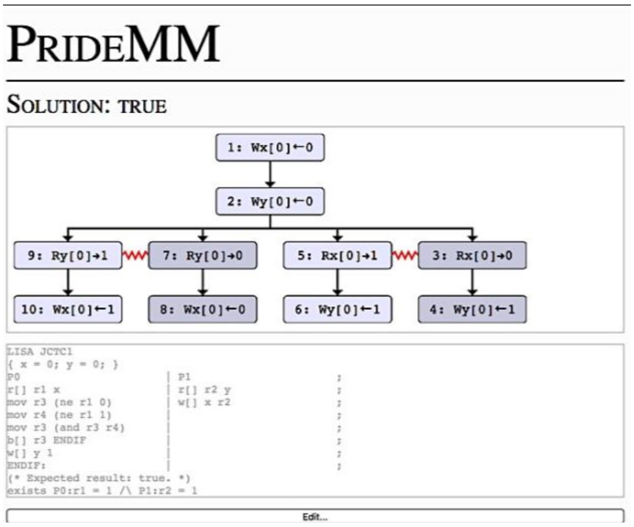
This project concerns PrideMM: a solver that allows one to run tests over state-of-the-art descriptions of the concurrency behaviour of programming languages. Previous relaxed memory simulators were based on ad-hoc backends or SAT solvers. Additional computational complexity arises in cutting-edge language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

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++ –*



PrideMM

Solution: true

```
LISA JCTC1
{ x = 0; y = 0; }
P0                    | P1                ;
r[] r1 x              | r[] r2 y          ;
mov r3 (ne r1 0)      | w[] x r2          ;
mov r4 (ne r1 1)      |                   ;
mov r3 (and r3 r4)    |                   ;
b[] r3 ENDIF          |                   ;
w[] y 1               |                   ;
ENDIF:                |                   ;
(* Expected result: true. *)
exists P0:r1 = 1 /\ P1:r2 = 1
```

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.

VERIFYING EFFICIENT LIBRARIES IN CAKEML

University of Kent

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

SCOTT OWENS

DEREK DREYER

- CakeML is a functional programming language and an ecosystem of associated proofs and tools, including a formally verified compiler to various processor architectures
- CakeML lacks support for verifying libraries that use unsafe features, e.g., array accesses w/o bounds checks
- The RustBelt project (Dreyer) 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 is necessary to develop an Iris-like logic for CakeML

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

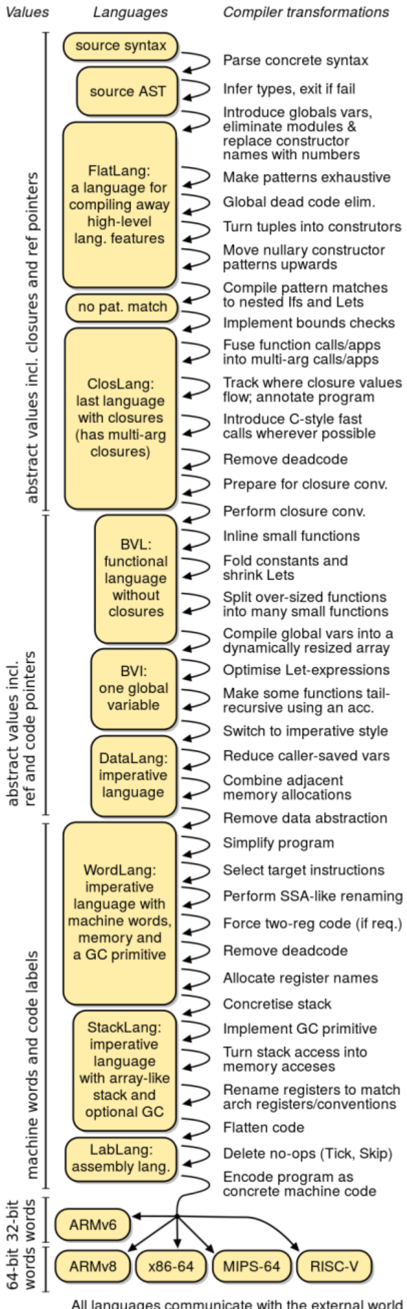
The RustBelt project aims to put the safety of Mozilla's Rust programming language on a firm semantic foundation. Rust's standard libraries make widespread internal use of *unsafe* blocks, which enable them to opt out of the type system when necessary. The hope is that such unsafe code is properly encapsulated, 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 that the Iris technology can, in principle, solve the problems observed in CakeML. However, subsequent work on developing an initial prototype demonstrated that we cannot directly apply the existing Iris work to CakeML, as hoped, and that we need to re-design its logical foundations 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.

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 –


Values
Languages
Compiler transformations
source syntax
source AST
FlatLang: a language for compiling away high-level lang. features
no pat. match
ClosLang: last language with closures (has multi-arg closures)
BVL: functional language without closures
BVI: one global variable
DataLang: imperative language
WordLang: imperative language with machine words, memory and a GC primitive
StackLang: imperative language with array-like stack and optional GC
LabLang: assembly lang.
ARMv6
ARMv8
x86-64
MIPS-64
RISC-V
abstract values incl. closures and ref pointers
abstract values incl. ref and code pointers
machine words and code labels
32-bit words
64-bit words
Parse concrete syntax
Infer types, exit if fail
Introduce globals vars, eliminate modules & replace constructor names with numbers
Make patterns exhaustive
Global dead code elim.
Turn tuples into construtors
Move nullary constructor patterns upwards
Compile pattern matches to nested Ifs and Lets
Implement bounds checks
Fuse function calls/apps into multi-arg calls/apps
Track where closure values flow; annotate program
Introduce C-style fast calls wherever possible
Remove deadcode
Prepare for closure conv.
Perform closure conv.
Inline small functions
Fold constants and shrink Lets
Split over-sized functions into many small functions
Compile global vars into a dynamically resized array
Optimise Let-expressions
Make some functions tail-recursive using an acc.
Switch to imperative style
Reduce caller-saved vars
Combine adjacent memory allocations
Remove data abstraction
Simplify program
Select target instructions
Perform SSA-like renaming
Force two-reg code (if req.)
Remove deadcode
Allocate register names
Concretise stack
Implement GC primitive
Turn stack access into memory acceses
Rename registers to match arch registers/conventions
Flatten code
Delete no-ops (Tick, Skip)
Encode program as concrete machine code

All languages communicate with the external world via a byte-array-based foreign-function interface.

CakeML Infrastructure

SUPERVECTORIZER

Queen Mary University of London

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 superoptimisation* 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

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.

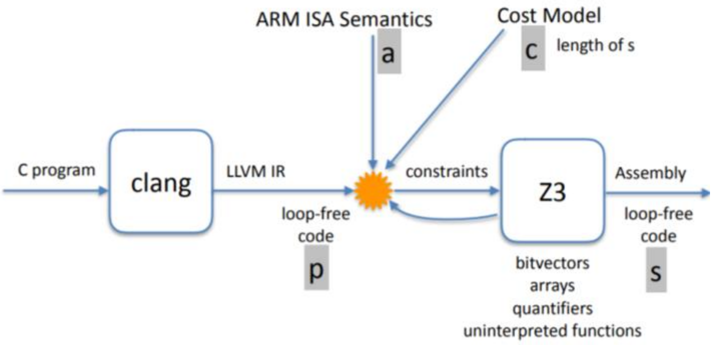
This project aimed at enabling software to take full advantage of SIMD capabilities of microprocessor designs, without modifying the compiler. In particular, we integrate unbounded superoptimisation with auto-vectorisation. This approach reduces the engineering effort needed to tune a production compiler 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.

PUBLICATIONS. An article on the symbolic cost model for SIMD instructions is in preparation.

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



Structure of the preliminary prototype

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).

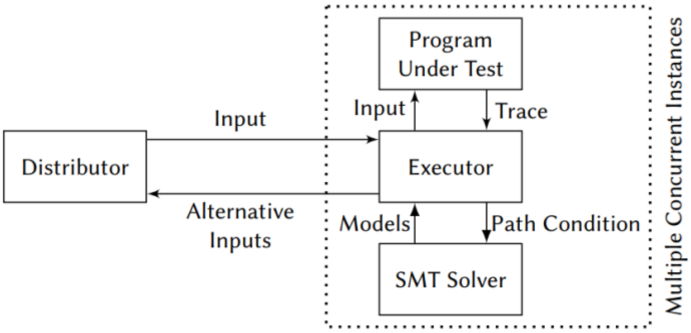
Regular expressions (REs) limit applicability of DSE to testing code security in practical client- and server-side web applications, as modern solvers cannot reason about real-world REs as they are used by developers. We developed an encoding of complex REs and with a refinement scheme that soundly translates 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%. The increased coverage demonstrates that more parts of the program can be reached, increasing the analysis surface for detecting bugs and 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.

Publications. [1] B. Loring, D. Mitchell, J. Kinder. “ExpoSE: Practical Symbolic Execution of Standalone JavaScript”. In Proc. Int. SPIN Symp. 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.

Related grants. “Cryptobugs – Detecting Incorrect Use of Cryptographic Routines”, GCHQ Studentship, 09/2015-02/2019. £120,449.

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 –



Parallel testing architecture of ExpoSE

Automated Reasoning with Fine-Grained Concurrent Collections

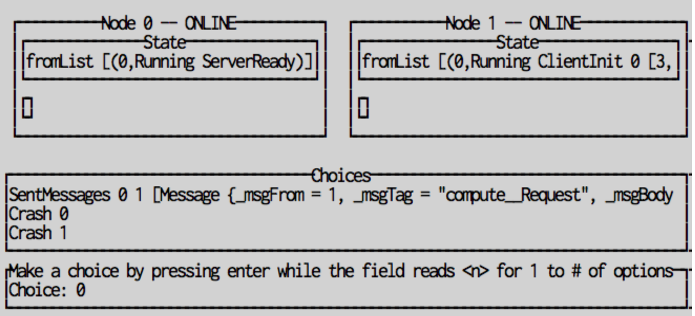
Ilya Sergey **Nikos Gorogiannis**

- 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, we believe, can 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 the future, we are going to explore the opportunities, opened by DPC, for randomised protocol testing and lightweight verification with refinement types.



```
Node 0 — ONLINE                          Node 1 — ONLINE
State                                    State
fromList [(0,Running ServerReady)]       fromList [(0,Running ClientInit 0 [3,

Choices
SentMessages 0 1 [Message {_msgFrom = 1, _msgTag = "compute__Request", _msgBody
Crash 0
Crash 1

Make a choice by pressing enter while the field reads <n> for 1 to # of options
Choice: 0
```

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.

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

UNIVERSITY of York

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
- Support for reasoning about diagrams with algebraic loops, ignored by most other verification approaches
- Verification of a subsystem of an industrial aircraft cabin-pressure control application

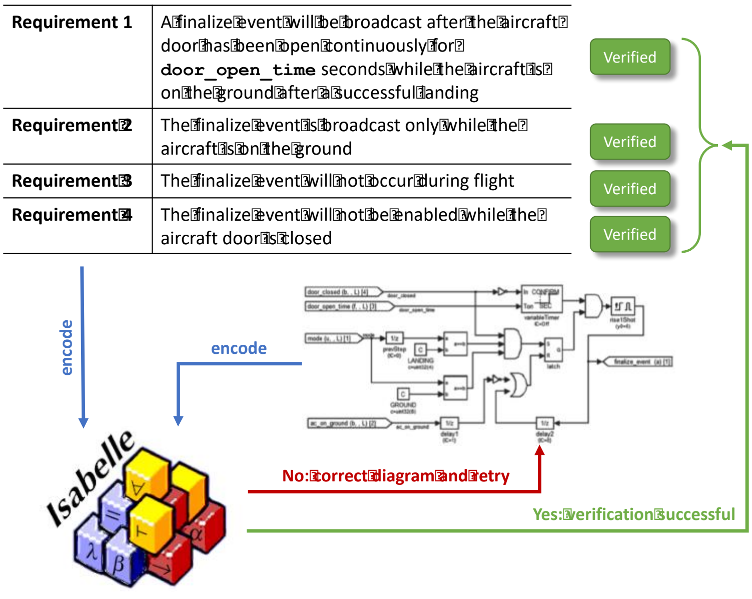
Control-law diagrams are used in industry to model complex engineering systems, such as the many components of modern aircrafts. These systems must be built to the very highest standards possible, and their control laws must be verified to ensure that they behave as required. Our project proposes a general 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.

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”, under submission.

Related grants. Dr Simon Foster, EPSRC UKRI Innovation Fellowship: “CyPhyAssure: Compositional 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 –



Requirement 1	A finalize event will be broadcast after the aircraft door has been open continuously for **door_open_time** seconds while the aircraft is on the ground after a successful landing	Verified
Requirement 2	The finalize event is broadcast only while the aircraft is on the ground	Verified
Requirement 3	The finalize event will not occur during flight	Verified
Requirement 4	The finalize event will not be enabled while the aircraft door is closed	Verified

Workflow: from textual representation to formal verification

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

VeTSS

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