Formal Methods Review

2022-03-16 10:14

Richard L. Ford
richardlford@gmail.com

Abstract

This is a review of my research into formal methods theory, tools, projects and people, but also related topics in software engineering and mathematics. Some of this is summarized in the body of the document. But I have also been collecting and reviewing research papers and books on related topics. The references section of this document includes the related documents from my Zotero data-base. Some of these documents I've studied in detail, others I've only read the abstract and decided they were worth keeping for future reference.

Contents

1. Introduction

This document was written in the course of a research effort to apply formal methods to the task of independent verification and validation of software and hardware. Therefore its focus is on formal methods groups, proof systems, projects and researchers, but it also includes some more general topics in software engineering and mathematics. For each it gives pointers for further information. The bibliography at the end includes some papers or topics that are not mentioned in the body of the document. It is a work-in-progress and is necessarily incomplete. Apologies to the many formal methods researchers and organizations that have not been included.

2. Formal Methods Research Groups

In this section I will mention some of the research groups with which I'm most familiar.

2.1. DeepSpec Project

Home Page
https://deepspec.org

2.2. Galois Inc

Home Page
https://galois.com/
Sources
https://github.com/GaloisInc

2.3. Yale FLINT Group

Home Page
http://flint.cs.yale.edu/flint/
Publications
http://flint.cs.yale.edu/flint/publications

"The FLINT group at Yale aims to develop a novel and practical programming infrastructure for constructing large-scale certified systems software. By combining recent new advances in programming languages, formal semantics, certified operating systems, program verification, proof assistants and automation, language-based security, and certifying compilers, we hope to attack the following research questions:

- what system software structures can offer the best support for extensibility, security, and resilience?
- what program logics and semantic models can best capture these abstractions?
- what are the right programming languages and environments for developing such certified system software?
- how to build new automation facilities to make certified software really scale?"

2.4. Everest Expedition

Home Page
https://project-everest.github.io/
Source
https://github.com/project-everest

"We are a team of researchers and engineers from several organizations, including Microsoft Research, Carnegie Mellon University, INRIA, and the MSR-INRIA joint center.

Everest is a recursive acronym: It stands for the "Everest VERified End-to-end Secure Transport"."

2.5. University of Washington Programming Languages and Software Engineering

Home Page
http://uwplse.org/
Source
https://github.com/uwplse

3. Formal Methods Proof Systems

3.1. Coq

Home Page
https://coq.inria.fr/
Source
https://github.com/coq/coq.git

Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. Typical applications include the certification of properties of programming languages (e.g. the CompCert compiler certification project, or the Bedrock verified low-level programming library), the formalization of mathematics (e.g. the full formalization of the Feit-Thompson theorem or homotopy type theory) and teaching.

3.2. KeYmaeraX

Home Page
http://www.ls.cs.cmu.edu/KeYmaeraX/
Source
https://github.com/LS-Lab/KeYmaeraX-release

KeYmaeraX is a system for specifying and verifying Hybrid Cyber-Physical Systems.

3.3. Isabelle

Home Page
https://isabelle.in.tum.de/
Source
hg clone https://isabelle.in.tum.de/repos/isabelle

"Isabelle is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols."

3.4. HOL4

Home Page
https://hol-theorem-prover.org/
Source
https://github.com/HOL-Theorem-Prover

"The HOL interactive theorem prover is a proof assistant for higher-order logic: a programming environment in which theorems can be proved and proof tools implemented. Built-in decision procedures and theorem provers can automatically establish many simple theorems (users may have to prove the hard theorems themselves!) An oracle mechanism gives access to external programs such as SMT and BDD engines. HOL is particularly suitable as a platform for implementing combinations of deduction, execution and property checking."

3.5. HOL-Light

Home Page
https://www.cl.cam.ac.uk/~jrh13/hol-light/
Source
https://github.com/jrh13/hol-light/

"HOL Light is a computer program to help users prove interesting mathematical theorems completely formally in higher order logic. It sets a very exacting standard of correctness, but provides a number of automated tools and pre-proved mathematical theorems (e.g. about arithmetic, basic set theory and real analysis) to save the user work. It is also fully programmable, so users can extend it with new theorems and inference rules without compromising its soundness."

3.6. Dafny

Home Page
Dafny
Source
https://github.com/Microsoft/dafny
Tutorial
https://rise4fun.com/dafny

3.7. Boogie

Home Page
Boogie
Source
https://github.com/boogie-org/boogie
Documentation
https://boogie-docs.readthedocs.io/en/latest/
Online trial
https://rise4fun.com/Boogie

3.8. Why3

Home Page
http://why3.lri.fr/

3.9. Frama-C

Home Page
https://frama-c.com/

3.10. F*

Home Page
https://www.fstar-lang.org/
Source
http://github.com/FStarLang/FStar
Papers:
F* Tutorial

3.11. LEAN

"Lean is a functional programming language that makes it easy to write correct and maintainable code. You can also use Lean as an interactive theorem prover. Lean programming primarily involves defining types and functions. This allows your focus to remain on the problem domain and manipulating its data, rather than the details of programming."

Lean 4 is the current version and has reached its 3rd milestone. The previous version, Lean 3, has been used to construct an extensive mathematical library. It will be ported to Lean 4.

Home and related Pages

https://leanprover.github.io/
https://github.com/leanprover/lean4
https://leanprover-community.github.io/index.html

Papers

A Metaprogramming Framework for Formal Verification

Beyond Notations: Hygienic Macro Expansion for Theorem Proving Languages. (Ullrich and Moura, 2020)

Counting Immutable Beans: Reference Counting Optimized for Purely Functional Programming. (Ullrich and Moura, 2019)

4. Formal Methods Projects

4.1. DeepSpec Projects

4.1.1. CompCert

Home Page

http://compcert.inria.fr

Source

https://github.com/AbsInt/CompCert.git

The CompCert C verified compiler is a compiler for a large subset of the C programming language that generates code for the PowerPC, ARM, x86 and RISC-V processors.

The distinguishing feature of CompCert is that it has been formally verified using the Coq proof assistant: the generated assembly code is formally guaranteed to behave as prescribed by the semantics of the source C code.

4.1.2. Verified Software Toolchain

Home Page

http://vst.cs.princeton.edu/

Source

https://github.com/PrincetonUniversity/VST.git

The software toolchain includes static analyzers to check assertions about your program; optimizing compilers to translate your program to machine language; operating systems and li-

braries to supply context for your program. The Verified Software Toolchain project assures with machine-checked proofs that the assertions claimed at the top of the toolchain really hold in the machine-language program, running in the operating-system context.

In some application domains it is not enough to build reliable software systems, one wants proved-correct software. This is the case for safety-critical systems (where software bugs can cause injury or death) and for security-critical applications (where an attacker is deliberately searching for, and exploiting, software bugs). Since proofs are large and complex, the proof-checking must be mechanized. Machine-checked proofs of real software systems are difficult, but now should be possible, given the recent advances in the theory and engineering of mechanized proof systems applied to software verification. But there are several challenges:

- Real software systems are usually built from components in different programming languages.
- Some parts of the program need full correctness proofs, which must be constructed with great effort; other parts need only safety proofs, which can be constructed automatically.
- One reasons about correctness at the source-code level, but one runs a machine-code program translated by a compiler; the compiler must be proved correct.
- These proofs about different properties, with respect to different programming languages, must be integrated together end-to-end in a way that is also proved correct and machine-checked.

We address these challenges by defining Verifiable C, a program logic for the C programming language. Verifiable C is proved sound with respect to the operational semantics of CompCert C; in turn, the CompCert verified optimizing C compiler is proved correct with respect to the assembly-language semantics of the PowerPC, ARM, and x86 processors.

4.1.3. CertiKOS - Certified Kit Operating System

Home Page
http://flint.cs.yale.edu/certikos/
Source
https://github.com/npe9/certikos.git

Developed by the FLINT group.

4.1.4. VeriML

Home Page
http://flint.cs.yale.edu/shao/papers/veriml.html
Source
http://flint.cs.yale.edu/flint/publications/veriml-0.1.tar.gz
Paper
http://flint.cs.yale.edu/flint/publications/verimltr.pdf

Developed by the FLINT group.

Modern proof assistants such as Coq and Isabelle provide high degrees of expressiveness and assurance because they support formal reasoning in higher-order logic and supply explicit machine-checkable proof objects. Unfortunately, large scale proof development in these proof assistants is still an extremely difficult and time-consuming task. One major weakness of these proof assistants is the lack of a single language where users can develop complex tactics and decision procedures using a rich programming model and in a typeful manner. This limits the scalability of the proof development process, as users avoid developing domain-specific tactics and decision procedures.

In this paper, we present VeriML—a novel language design that couples a type-safe effectful computational language with first-class support for manipulating logical terms such as propositions and proofs. The main idea behind our design is to integrate a rich logical framework—similar to the one supported by Coq—inside a computational language inspired by ML. The language design is such that the added features are orthogonal to the rest of the computational language, and also do not require significant additions to the logic language, so soundness is guaranteed. We have built a prototype implementation of VeriML including both its type-checker and an interpreter. We demonstrate the effectiveness of our design by showing a number of type-safe tactics and decision procedures written in VeriML.

4.1.5. Certifying Low-Level Programs with Hardware Interrupts and Preemptive Threads

Home Page
http://flint.cs.yale.edu/shao/papers/aimjar.html
Source
http://flint.cs.yale.edu/flint/publications/aim.coq.tar.gz
Local repo
e/certhwint

Developed by the FLINT group.

4.1.6. Kami

Home Page
http://plv.csail.mit.edu/kami/
Source
https://github.com/mit-plv/kami

Kami is a library that turns Coq into an IDE for digital hardware development, based on a clean-slate reimplementation of a core of the Bluespec language. We span the gap from mathematical specifications to hardware circuit descriptions (RTL netlists). We support specifying, implementing, verifying, and compiling hardware, reasoning at a high level about particular hardware components but in the end deriving first-principles Coq theorems about circuits. No part of Kami need be trusted beside the formalization of low-level (Verilog-style) circuit descriptions; all other aspects have end-to-end correctness proofs checked by Coq. Hardware designs are broken into separately verified modules, reasoned about with a novel take on labeled transition systems. Furthermore, Coq provides a natural and expressive platform for metaprogramming, or building verified circuit generators, as for a memory caching system autogenerated for a particular shape of cache hierarchy, or a CPU generated given a number of concurrent cores as input.

4.1.7. Haskell Core Spec

Home Page
https://deepspec.org/entry/Project/Haskell+CoreSpec
Source
https://github.com/sweirich/corespec.git

The Haskell CoreSpec Project aims to develop formal specifications for a high-level, industrially-relevant functional programming language. In particular, this project targets the core language of the Glasgow Haskell Compiler, the primary compiler for the Haskell programming language. GHC has long been used as both an industrial strength compiler and a platform for language research. The compiler itself is open source, and has primarily been developed and is currently maintained by researchers at Microsoft Research, Cambridge. The CoreSpec project will develop a formal Coq specification of the GHC Core language, including the syntax, type system, and semantics, and connect that specification to other components of the DeepSpec project.

4.1.8. Deep Spec Server

Home Page

https://deepspec.org/entry/Project/DeepSpec+Web+Server

Source

Not available but see Libmicrohttpd below.

"For a final demo, unifying many of the Expedition threads, we aim to build a verified web server."

Status: "A team at Penn has begun exploring the design space and building a first-draft prototype (for now, running on Linux) of a web server library loosely based on the popular libmicrohttpd. The goal of this short-term effort is to understand the integration issues that will be involved in putting together a fully functional server from components under development within DeepSpec. In particular, we want to understand what demands it will place on CertiKOS in terms of OS features (IPC, network support, shared-memory processes, interoperation between native clients and Linux VMs), what verification challenges it raises for VST, what integration challenges it poses for using VST and CertiKOS together."

4.1.8.1. GNU Libmicrohttpd

Home Page

https://www.gnu.org/software/libmicrohttpd/

Source

https://gnunet.org/git/libmicrohttpd.git

4.1.9. Verdi

Home Page

http://verdi.uwplse.org/

Source

https://github.com/uwplse/verdi

Example

https://github.com/uwplse/verdi-raft

Verification of distributed systems.

4.1.10. Vellvm

Home Page

http://www.cis.upenn.edu/~stevez/vellvm/

Source

https://github.com/vellvm/vellvm

Old Source

https://github.com/vellvm/vellvm-legacy

"The Vellvm project is building a (verified LLVM), a framework for reasoning about programs expressed in LLVM's intermediate representation and transformations that operate on it. Vellvm provides a mechanized formal semantics of LLVM's intermediate representation, its type system, and properties of its SSA form. The framework is built using the Coq interactive theorem prover. It includes multiple operational semantics and proves relations among them to facilitate different reasoning styles and proof techniques.

4.1.10.1. Latest Results During the first year of DeepSpec we

1. worked on developing a new modular semantics for Vellvm, factoring out the memory model
2. made progress on connecting LLVM-IR like SSA semantics with higher-level structural operational semantics
3. applied low-level language verification techniques to the problem of race detection instrumentation"

4.1.11. Deep Spec Crypto

Home Page
https://deepspec.org/entry/Project/Cryptography
Source
https://github.com/mit-plv/fiat-crypto
Papers
http://adam.chlipala.net/papers/FiatCryptoSP19/FiatCryptoSP19.pdf

<http://www.cs.princeton.edu/~appel/papers/verified-hmac-drbg.pdf>

"We are pursuing end-to-end proofs of cryptographic functionality, via verification of C code at Princeton and synthesis of assembly code at MIT. We are considering both cryptographic primitives (e.g. pseudorandom number generation with VST at Princeton and elliptic curve operations with fiat-crypto at MIT) and protocols (verified with the Foundational Cryptography Framework and connected to results about C and assembly programs)."

Latest Results: Fiat Cryptography is now used in Google's BoringSSL library for elliptic-curve arithmetic. As a result, Chrome HTTPS connections now run our Coq-generated code. Our upcoming S&P 2019 paper goes into more detail.

VST verification has recently focused on the primitives HMAC-DGBG and HKDF – both clients of HMAC/SHA256, AES, and parts of the TweetNaCl library. In addition to verifying (families of) primitives, we hope to soon turn to integration in larger contexts like verified TLS libraries.

4.1.12. DeepSpecDb

Source
https://github.com/PrincetonUniversity/DeepSpecDB
Papers
VST Verification of B+Trees with Cursors
Implementing a high-performance key-value store using a trie of B+-Trees with cursors
Project Report on DeepSpecDB
The Theory and Verification of B+Tree Cursor Relations

4.1.13. Fiat

Home Page
http://plv.csail.mit.edu/fiat/
Source
https://github.com/mit-plv/fiat.git
Papers
The End of History? Using a Proof Assistant to Replace Language Design with Library Design

"Fiat is a library for the Coq proof assistant for synthesizing efficient correct-by-construction programs from declarative specifications. Programming by Fiat starts with a high-level description of a program, which can be written using libraries of specification languages for describing common programming tasks like querying a relational database. These specifications are then iteratively refined into efficient implementations via automated tactics. Each derivation in Fiat produces a formal proof trail certifying that the synthesized program meets the original specification. Code synthesized by Fiat can be extracted to an equivalent OCaml program that can be compiled and run as normal."

4.1.14. Narcissus

Home Page
https://www.cs.purdue.edu/homes/bendy/Narcissus/

Source

https://github.com/mit-plv/fiat/tree/master/src/Narcissus

https://github.com/bendy/fiat-asn.1

Papers

NARCISSUS: Deriving Correct-By-Construction Decoders and Encoders from Binary Formats

https://www.cs.purdue.edu/homes/bendy/Narcissus/narcissus.pdf

Video

Narcissus is part of the fiat project to derive Correct-By-Construction Decoders and Encoders from Binary Formats.

4.1.15. Bedrock2

Source

https://github.com/mit-plv/bedrock2

"The Bedrock2 repository contains ongoing work on a low-level systems programming language. One piece of the puzzle is a verified compiler targeting RISC-V. The source language itself is also equipped with a simple program logic for proving correctness of the source programs. It is not ready yet, at least for most uses. This project has similar goals as bedrock, but uses a different design. No code is shared between bedrock and bedrock2. The source language is a "C-like" language called ExprImp."

4.1.16. CertiCoq

Home Page

https://www.cs.princeton.edu/~appel/certicoq/

Source

https://github.com/PrincetonUniversity/certicoq

Paper

http://www.cs.princeton.edu/~appel/papers/certicoq-coqpl.pdf

"The CertiCoq project aims to build a proven-correct compiler for dependently-typed, functional languages, such as Gallinathe core language of the Coq proof assistant. A proved-correct compiler consists of a high-level functional specification, machine-verified proofs of important properties, such as safety and correctness, and a mechanism to transport those proofs to the generated machine code. The project exposes both engineering challenges and foundational questions about compilers for dependently-typed languages."

4.1.17. Template-Coq

Home Page
https://template-coq.github.io/template-coq/
Source
https://github.com/Template-Coq/template-coq
Papers
https://popl18.sigplan.org/event/coqpl-2018-typed-template-coq

Template Coq is a quoting library for Coq. It takes `Coq` terms and constructs a representation of their syntax tree as a `Coq` inductive data type. The representation is based on the kernel's term representation.

This is used as the first stage of CertiCoq.

4.1.18. QuickChick

Home Page
https://deepspec.org/entry/Project/QuickChick
Source
https://github.com/QuickChick
Book
QuickChick: Property-Based Testing in Coq

- Randomized property-based testing plugin for Coq; a clone of Haskell QuickCheck
- Includes a foundational verification framework for testing code
- Includes a mechanism for automatically deriving generators for inductive relations

4.1.19. Galois Voting System

Home Page
https://galois.com/blog/2009/03/trustworthy-voting-systems/

4.2. Everest Projects

Project Everest is the combination of the following projects.

- F*, a verification language for effectful programs
- miTLS, reference implementation of the TLS protocol in F*
- KreMLin, a compiler from a subset of F* to C

- HACL*, a verified library of cryptographic primitives written in F*
- Vale, a domain-specific language for verified cryptographic primitives in assembly
- EverCrypt, a verified crypto provider that combines HACL* and Vale via an agile, multi-platform, self-configuring cryptographic API.
- EverParse, a library and tool to automatically generate verified parsers and serializers for binary data formats
- Quackyducky, a small tool to translate informal specification of message formats found in RFC (in particular for TLS 1.3) into formal F# specifications, which are in turn transformed into efficient and verified parser implementations.

4.3. Other Projects

4.3.1. CakeML

Home page
https://cakeml.org/
Source
https://github.com/CakeML/cakeml

"CakeML is a functional programming language and an ecosystem of proofs and tools built around the language. The ecosystem includes a proven-correct compiler that can bootstrap itself."

4.3.2. VCC - A verifier for Concurrent C

Home Page
https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/
Source
https://github.com/Microsoft/vcc.git

4.3.3. Compositional CompCert

Source
https://github.com/PrincetonUniversity/compcomp

Compcert supporting separate compilation. Last modified in 2015.

4.3.4. GaloisInc Projects

Home Page
https://galois.com/
Source
https://github.com/GaloisInc

"Galois develops technology to guarantee the trustworthiness of systems where failure is unacceptable.

We apply cutting edge computer science and mathematics to advance the state of the art in software and hardware trustworthiness."

4.3.5. Bedrock

Home Page
http://plv.csail.mit.edu/bedrock/

4.3.6. FSCQ

Home Page
http://css.csail.mit.edu/fscq/

A file system verified in Coq using a separation logic for reasoning about crash safety

4.3.7. Ur/Web

Home Page
http://plv.csail.mit.edu/ur/

5. Formal Methods Researchers

Alphabetically by last name, then first.

5.1. Andrew W Appel

Home Page
http://www.cs.princeton.edu/~appel/index.html

5.2. Adam Chlipala

Home Page
http://adam.chlipala.net/

5.3. Robert Harper

Home Page
http://www.cs.cmu.edu/~rwh/

5.4. Benjamin Pierce

Home Page
http://www.cis.upenn.edu/~bcpierce/
LinkedIn
https://github.com/bcpierce00

Professor Department of Computer and Information Science University of Pennsylvania. Author of Software Foundations.

5.5. Gigore Rosu

Home Page
https://cs.illinois.edu/about/people/faculty/grosu

He is the inventor and a proponent of the K framework and Matching Logic (Rosu, 2017b, 2017a; X. Chen et al., 2020; X. Chen and Roşu, 2020; Shangbei Wang, 2021; Tuˇsil et al., 2021; Z. Lin et al., 2021; Bereczky et al., 2022) and also a founder of the Runtime Verification, Inc.

5.6. Zhong Shao

Home Page
http://cs-www.cs.yale.edu/homes/shao/

6. Statistics

Here are some statistics for projects using Coq. The "Types" column is the number of inductive types defined. The "Defs" column is the number of "Definitions". Some projects are broken up

by component.

Project	#Coq Files	SLOC	Proofs	Types	Defs	Notes
certikos						Kit Operating System
- compcert	205	206270	5359	465	2945	Modified compcert
- compcertx	50	8645	326	25	60	Compcert for sep compilation
- liblayers	55	22122	725	43	189	
- mcertikos	449	207281	5591	324	1757	
cfml	177	65587	2769	131	892	Tool for proving OCaml programs in Separation Logic
ch2o	116	49351	4472	153	424	A formalization of the C11 standard in Coq
compcert 3.3	231	215450	6728	593	4601	Formally Verified C Compiler
compcert 3.4	225	177117	6729	525	3031	Formally Verified C Compiler
coq	1984	247663	12131	1097	5666	Coq Proof Assistant Library
Coq-dL	84	83871	2849	50	894	Formalization of KeYmaeraX in Coq
coquelicot	28	41615	1751	6	324	User friendly Calculus in Coq
corespec	41	35694	1351	33	215	Formalization of Haskell Core in Coq
Corn	348	156363	6895	33	2118	Coq Constructive Repository at Nijmegen (Reals)
DeepSpecDB	55	32788	531	30	1151	DeepSpec Data Base

Project	#Coq Files	SLOC	Proofs	Types	Defs	Notes
certikos						Kit Operating System
- compcert	205	206270	5359	465	2945	Modified compcert
- compcertx	50	8645	326	25	60	Compcert for sep compilation
- liblayers	55	22122	725	43	189	
- mcertikos	449	207281	5591	324	1757	
dsss17 -total	490	302318	10580	1061	5162	DeepSpec Summer School 2017
- auto	6	3495	148	16	23	Proof Automation - Chlipala
- CAL	378	245501	8762	834	4321	Certifying software with crashes (Cert Abstr layers)
- compiler	6	3813	116	22	48	Compiler for Imp (Xaxier)
- Metalib	18	7015	307	7	97	Support for Stlc
- qc	9	5073	26	24	84	QuickChick
- SF	34	20875	656	107	316	Software Foundations
- Stlc	12	8942	392	27	62	Lang Spec and Variable binding
- vminus	27	7604	173	24	211	Vellvm: Verifying the LLVM
dsss18 - total	743	258178	8617	614	4399	DeepSpec Summeer School 2018
- charIO	18	3704	84	23	167	
- dw	88	17300	199	45	572	
- kami	75	47358	1996	97	624	
- lf	38	17147	480	78	194	SF - Logical Foundations
- plf	48	33305	589	140	249	SF - Programming Languages Foundations
- qc	10	6767	21	28	80	Quick Chick
- vc	15	9172	4915	174	1685	Verifiable C (Proofs using VST)
- vfa	30	8680	205	28	168	SF - Verified Functional Algorithms
fiat	647	197824	5623	76	4075	Deductive Synthesis of Abstract Data Types in a Proof Assistant
- Narcissus	72	32310	847	12	605	Subset of Fiat for interface generation

Project	#Coq Files	SLOC	Proofs	Types	Defs	Notes
certikos						Kit Operating System
- compcert	205	206270	5359	465	2945	Modified compcert
- compcertx	50	8645	326	25	60	Compcert for sep compilation
- liblayers	55	22122	725	43	189	
- mcertikos	449	207281	5591	324	1757	
flocq	40	67543	1225	21	317	Formalization of floating point
kami	101	53910	1937	87	976	Framework to Support Implementing and Verifying Bluespec-style Hardware Components
math-comp	92	111079	11379	38	3509	Mathematical Components Library
qc	9	6239	21	28	79	SF - Quick Chick 1.0
template-coq	72	11541	180	64	472	Quoting library for Coq (frontend for CertiCoq)
tlc	58	40300	2496	89	552	General purpose alternate to Coq's Standard Library
vellvm	55	24006	1085	122	635	Verifying LLVM
verified-ifc	58	31527	849	123	395	A Verified Information-Flow Architecture
vst	508	314515	11812	481	7882	Verified Software Toolchain
why2	98	40045	260	67	1787	Why2 verification tool
why3 1.0	189	44304	968	365	1030	Why3 verification tool

References

Abadi, M. (1998). Protection in programming-language translations. *DECSRC*, 16.

Abadi, M. (1999). Secrecy by typing in security protocols. *Journal of the ACM*, *46*(5), 749–786. https://doi.org/10.1145/324133.324266

Abadi, M., Fournet, C., & Gonthier, G. (2002). Secure Implementation of Channel Abstractions. *Information and Computation*, *174*(1), 37–83. https://doi.org/10.1006/inco.2002.3086

Abate, C., Amorim, A. A. de, Blanco, R., Evans, A. N., Fachini, G., Hritcu, C., … Tolmach, A. (2019). When Good Components Go Bad: Formally Secure Compilation Despite Dynamic Compromise. *arXiv:1802.00588 [cs]*. Retrieved from arXiv:1802.00588

Abate, C., Blanco, R., Ciobâcă, T., Durier, A., Garg, D., Hrit, C., … Thibault, J. (2021). An Extended Account of Trace-Relating Compiler Correctness and Secure Compilation. *TOPLAS*, *To Appear*, 48. Retrieved from https://people.mpi-sws.org/~dg/papers/toplas21-diff.pdf

Abate, C., Blanco, R., Garg, D., Hritcu, C., Patrignani, M., & Thibault, J. (2019). Journey Beyond Full Abstraction: Exploring Robust Property Preservation for Secure Compilation. *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, 256–25615. https://doi.org/10.1109/CSF.2019.00025

Absint. (n.d.). CompCert - Publications. Retrieved January 31, 2019, from http://compcert.inria.fr/publi.html

Abyaneh, A. S., & Kirsch, C. M. (n.d.). ASE: A Value Set Decision Procedure for Symbolic Execution, 12.

ACSL by Example.GitHub. (n.d.). Retrieved January 10, 2020, from https://github.com/fraunhoferfokus/acsl-by-example

Adamek, J., Herrlich, H., Strecker, G. E., & Schubert, C. (2004). Abstract and Concrete Categories - The Joy of Cats, 524. Retrieved from http://katmat.math.uni-bremen.de/acc/acc.pdf

Adams, M. (2015). The Common HOL Platform. *Electronic Proceedings in Theoretical Computer Science*, *186*, 42–56. https://doi.org/10.4204/EPTCS.186.6

Adewale, O. (n.d.). Implementing a high-performance key-value store using a trie of B+-Trees with cursors | Computer Science Department at Princeton University. Retrieved February 1, 2019, from https://www.cs.princeton.edu/research/techreps/TR-004-18

Agten, P., Strackx, R., Jacobs, B., & Piessens, F. (2012). Secure Compilation to Modern Processors. In *2012 IEEE 25th Computer Security Foundations Symposium* (pp. 171–185). Cambridge, MA, USA: IEEE. https://doi.org/10.1109/CSF.2012.12

Aguirre, A., & Birkedal, L. (2017). Step-Indexed Logical Relations for Nondeterministic and Probabilistic Choice, 27.

Ahman, D., Fournet, C., Hriţcu, C., Maillard, K., Rastogi, A., & Swamy, N. (2017). Recalling a Witness: Foundations and Applications of Monotonic State. *Proc. ACM Program. Lang.*, *2*, 65:1–65:30. https://doi.org/10.1145/3158153

Ahman, D., Hriţcu, C., Maillard, K., Martínez, G., Plotkin, G., Protzenko, J., … Swamy, N. (2017). Dijkstra Monads for Free. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (pp. 515–529). New York, NY, USA: ACM. https://doi.org/10.1145/3009837.3009878

Ahrendt, W. (n.d.). Deductive Software Verification – The KeY BookFrom Theory to Practice – The KeY Project. Retrieved January 31, 2019, from https://www.key-project.org/thebook2/

Ahrendt, W., Beckert, B., Hähnle, R., Rümmer, P., & Schmitt, P. H. (2007). Verifying Object-Oriented Programs with KeY: A Tutorial. In F. S. de Boer, M. M. Bonsangue, S. Graf, & W.-P. de Roever (Eds.), *Formal Methods for Components and Objects* (Vol. 4709, pp. 70–101). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-74792-5_4

Alam, I. (n.d.). Tailoring Taint Analysis for Database Applications in the K Framework. Retrieved December 7, 2021, from https://www.scitepress.org/Papers/2021/106186/106186.pdf

A Learning Environment for Theorem Proving with the Coq proof assistant: princeton-vl/CoqGym. (2019). Princeton Vision & Learning Lab. Retrieved from https://github.com/princeton-vl/CoqGym

Algebraic Specification of Stack Effects for Forth Programs.ResearchGate. (n.d.). Retrieved May 26, 2019, from https://www.researchgate.net/publication/269399251_Algebraic_Specification_of_Stack_Effects_for_Forth_Programs

Algehed, M., Bernardy, J.-P., & Hritcu, C. (n.d.). Dynamic IFC Theorems for Free! (pp. 1–14). IEEE Computer Society. https://doi.org/10.1109/CSF51468.2021.00005

Alloy - software modeling. (n.d.). Retrieved January 15, 2020, from http://alloytools.org/

Almeida, J. B., Barbosa, M., Barthe, G., Gregoire, B., Koutsos, A., Laporte, V., … Strub, P.-Y. (2020). The Last Mile: High-Assurance and High-Speed Cryptographic Implementations. In *2020 IEEE Symposium on Security and Privacy (SP)* (pp. 965–982). San Francisco, CA, USA: IEEE. https://doi.org/10.1109/SP40000.2020.00028

Alomari, H., & Stephan, M. (2021). *Clone Detection through srcClone: A Program Slicing Based Approach*.

Alpern, B., & Schneider, F. B. (1985). Defining liveness. *Information Processing Letters*, *21*(4), 181–185. https://doi.org/10.1016/0020-0190(85)90056-0

Al-Sibahi, A. S., Jensen, T., Møgelberg, R. E., & Wąsowski, A. (2020). Galois Connections for Recursive Types. In A. Di Pierro, P. Malacaria, & R. Nagarajan (Eds.), *From Lambda Calculus to Cybersecurity Through Program Analysis* (Vol. 12065, pp. 105–131). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-41103-9_4

Altamirano, C. (n.d.). Formal Verification of an Implementation of the Roughtime Server, 46.

Altenkirch, T., Capriotti, P., Dijkstra, G., Kraus, N., & Forsberg, F. N. (2018). Quotient inductive-inductive types. *arXiv:1612.02346 [cs]*, *10803*, 293–310. https://doi.org/10.1007/978-3-319-89366-2_16

Altenkirch, T., McBride, C., & McKinna, J. (n.d.). Why Dependent Types Matter, 21.

Ambal, G., Lenglet, S., & Schmitt, A. (n.d.). Certified Abstract Machines for Skeletal Semantics, 14.

Amin, N., Leino, R., & Rompf, T. (2016). Computing with an SMT Solver, *8570*. Retrieved from https://www.microsoft.com/en-us/research/publication/computing-smt-solver/

Ammarguellat, Z. (1992). A control-flow normalization algorithm and its complexity. *IEEE Transactions on Software Engineering*, *18*(3), 237–251. https://doi.org/10.1109/32.126773

Amorim, A. A. de, Collins, N., DeHon, A., Demange, D., Hritcu, C., Pichardie, D., … Tolmach, A. (2013). *A Verified Information-Flow Architecture (Long version)*.

Anand, A., Boulier, S., Cohen, C., Sozeau, M., & Tabareau, N. (n.d.). Towards Certified Meta-Programming with Typed Template-Coq | SpringerLink. Retrieved February 1, 2019, from https://link.springer.com/chapter/10.1007%2F978-3-319-94821-8_2

Anand, A., & Knepper, R. (2015). ROSCoq: Robots Powered by Constructive Reals. In C. Urban & X. Zhang (Eds.), *Interactive Theorem Proving* (Vol. 9236, pp. 34–50). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-22102-1_3

Anand, A., Tabareau, S. B. N., & Sozeau, M. (n.d.). Typed Template Coq, 2.

Andersen, L. O. (1994). *Program Analysis and Specialization for the C Programming Language* (phdthesis). University of Copenhagen.

Anderson, C. J., Foster, N., Guha, A., Jeannin, J.-B., Kozen, D., Schlesinger, C., & Walker, D. (2014). NetKAT: semantic foundations for networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 113–126). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2535838.2535862

Anderson, S., Hood, J., & Jones, E. (2020). A Comparison of Fuzzing Dynamic Analysis and Static Code Analysis, 13.

Andrew, A. (2008). *Oracle Semantics Aquinas Hobor*.

Andronick, J., Klein, G., & Lewis, C. (2018). Formal Model of a Multi-Core Kernel-based System, 33.

Annenkov, D., Milo, M., Nielsen, J. B., & Spitters, B. (2021). Extracting smart contracts tested and verified in Coq. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 105–121). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3437992.3439934

Appel, A., & Leroy, X. (2021). Efficient Extensional Binary Tries. *arXiv:2110.05063 [cs]*. Retrieved from arXiv:2110.05063

Appel Andrew W., Beringer Lennart, Chlipala Adam, Pierce Benjamin C., Shao Zhong, Weirich Stephanie, & Zdancewic Steve. (2017). Position paper: the science of deep specification. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20160331. https://doi.org/10.1098/rsta.2016.0331

Appel, A. W. (2011). VeriSmall: Verified Smallfoot Shape Analysis. In J.-P. Jouannaud & Z. Shao (Eds.), *Certified Programs and Proofs* (Vol. 7086, pp. 231–246). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-25379-9_18

Appel, A. W. (2012). Verified Software Toolchain. In *Proceedings of the 4th International Conference on NASA Formal Methods* (pp. 2–2). Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/978-3-642-28891-3_2

Appel, A. W. (2015). Verification of a Cryptographic Primitive: SHA-256. *ACM Trans. Program. Lang. Syst.*, *37*(2), 7:1–7:31. https://doi.org/10.1145/2701415

Appel, A. W. (2019). *DeepSpecDB - github*. PrincetonUniversity. Retrieved from `https://github.com/PrincetonUniversity/DeepSpecDB`

Appel, A. W. (2020). C-language floating-point proofs layered with VST and Flocq, 16.

Appel, A. W. (2022). Coq's vibrant ecosystem for verification engineering (invited talk). In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 2–11). Philadelphia PA USA: ACM. https://doi.org/10.1145/3497775.3503951

Appel, A. W. (n.d.). CertiCoq: A verified compiler for Coq - POPL 2017. Retrieved February 1, 2019, from https://popl17.sigplan.org/event/main-certicoq-a-verified-compiler-for-coq

Appel, A. W., Dockins, R., Hobor, A., Beringer, L., Dodds, J., Stewart, G., … Leroy, X. (2014a). *Program Logics for Certified Compilers* (1 edition). Cambridge University Press.

Appel, A. W., Dockins, R., Hobor, A., Beringer, L., Dodds, J., Stewart, G., … Leroy, X. (2014b). *Verifiabble C, Version 2.2*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781107256552

Apt, K. R., & Olderog, E.-R. (2021). Assessing the Success and Impact of Hoare's Logic. In *Theories of Programming: The Life and Works of Tony Hoare* (1st ed., Vol. 39, pp. 41–76). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3477355.3477359

Arasu, A., Chandramouli, B., Gehrke, J., Ghosh, E., Kossmann, D., Protzenko, J., … Swamy, N. (n.d.). FastVer: Making Data Integrity a Commodity, 13.

Arias, E. J. G., Pin, B., & Jouvelot, P. (2017). jsCoq: Towards Hybrid Theorem Proving Interfaces. *Electronic Proceedings in Theoretical Computer Science*, *239*, 15–27. https://doi.org/10.4204/EPTCS.239.2

Armstrong, A., Bauereiss, T., Campbell, B., Reid, A., Gray, K. E., Norton, R. M., … Sewell, P. (2019). ISA Semantics for ARMv8-a, RISC-v, and CHERI-MIPS. *Proc. ACM Program. Lang.*, *3*, 71:1–71:31. https://doi.org/10.1145/3290384

Arusoaie, A., & Lucanu, D. (2021). Proof-Carrying Parameters in Certified Symbolic Execution: The Case Study of Antiunification. *arXiv:2110.11700 [cs]*. Retrieved from arXiv:2110.11700

Arya, J., Kumar, A., Singh, A., Mishra, T., & Chong, P. (2021). *BLOCKCHAIN: BASICS, APPLICATIONS, CHALLENGES AND OPPORTUNITIES*. https://doi.org/10.13140/RG.2.2.33899.16160

Askarov, A., Hunt, S., Sabelfeld, A., & Sands, D. (2008). Termination-Insensitive Noninterference Leaks More Than Just a Bit. In S. Jajodia & J. Lopez (Eds.), *Computer Security - ESORICS 2008* (Vol. 5283, pp. 333–348). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-88313-5_22

Assaf, A., Burel, G., Cauderlier, R., Dowek, G., Dubois, C., Gilbert, F., … Saillard, R. (n.d.). Dedukti: a Logical Framework based on the Π-Calculus Modulo Theory, 36.

Astrauskas, V., Müller, P., Poli, F., & Summers, A. J. (2019). Leveraging rust types for modular specification and verification. *Proceedings of the ACM on Programming Languages*, *3*, 147:1–147:30. https://doi.org/10.1145/3360573

Athaiya, S., Komondoor, R., & Kumar, K. N. (2021). Data Flow Analysis of Asynchronous Systems using Infinite Abstract Domains. *Programming Languages and Systems*, *12648*, 30–58. https://doi.org/10.1007/978-3-030-72019-3_2

Aydemir, B., Charguéraud, A., Pierce, B. C., Pollack, R., & Weirich, S. (n.d.). Engineering Formal Metatheory, 13.

Azevedo de Amorim, A., Collins, N., DeHon, A., Demange, D., Hriţcu, C., Pichardie, D., … Tolmach, A. (2014). A Verified Information-flow Architecture. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 165–178). New York, NY, USA: ACM. https://doi.org/10.1145/2535838.2535839

Baccelli, E., Gundogan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., … Wahlisch, M. (2018). RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal*, *5*(6), 4428–4440. https://doi.org/10.1109/JIOT.2018.2815038

Bacelar Almeida, J. C., Barbosa, M., Barthe, G., Pacheco, H., Pereira, V., & Portela, B. (2021). A Formal Treatment of the Role of Verified Compilers in Secure Computation. *Journal of Logical and Algebraic Methods in Programming*, 100736. https://doi.org/10.1016/j.jlamp.2021.100736

Backes, J., Varming, C., Whalen, M., Bolignano, P., Cook, B., Gacek, A., … Tanash, R. (2019). One-Click Formal Methods. *IEEE Software*, *36*(6), 61–65. https://doi.org/10.1109/MS.2019.2930609

Bae, Y., Kim, Y., Askar, A., Lim, J., & Kim, T. (2021). Rudra: Finding Memory Safety Bugs in Rust at the Ecosystem Scale. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (pp. 84–99). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3477132.3483570

Bahr, P., & Hutton, G. (n.d.). Monadic Compiler Calculation, 27.

Bakhirkin, A., & Monniaux, D. (2017). Combining Forward and Backward Abstract Interpretation of Horn Clauses. In F. Ranzato (Ed.), *24th International Static Analysis Symposium (SAS)*. New York City, United States: Springer. Retrieved from https://hal.archives-ouvertes.fr/hal-01551447

Balabonski, T., Lanco, A., & Melquiond, G. (n.d.). A strong call-by-need calculus, 22.

Balajorshari, N. E., & McCamant, S. (n.d.). Better Program Analysis for Security via Data Flow Tracking and Symbolic Execution, 92.

Ballabriga, C., Forget, J., Gonnord, L., Lipari, G., & Ruiz, J. (2019). Static Analysis Of Binary Code With Memory Indirections Using Polyhedra. In *VMCAI'19 - International Conference on Verification, Model Checking, and Abstract Interpretation* (Vol. 11388, pp. 114–135). Cascais, Portugal: Springer. https://doi.org/10.1007/978-3-030-11245-5_6

Ball, T., Bjørner, N., Gember, A., Itzhaky, S., Karbyshev, A., Sagiv, M., … Valadarsky, A. (2014). VeriCon: towards verifying controller programs in software-defined networks. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 282–293). Edinburgh United Kingdom: ACM. https://doi.org/10.1145/2594291.2594317

Bansal, K., Loos, S. M., Rabe, M. N., Szegedy, C., & Wilcox, S. (2019). HOList: An Environment for Machine Learning of Higher-Order Theorem Proving (extended version). *arXiv:1904.03241 [cs]*. Retrieved from arXiv:1904.03241

Bao, J., Gaboardi, M., Hsu, J., & Tassarotti, J. (2021). A Separation Logic for Negative Dependence. *arXiv:2111.14917 [cs]*. https://doi.org/10.1145/3498719

Bao, Y., Leavens, G. T., & Ernst, G. (2018). Unifying separation logic and region logic to allow interoperability. *Formal Aspects of Computing*, *30*(3), 381–441. https://doi.org/10.1007/s00165-018-0455-5

Bao, Y., Sundararajah, K., Malik, R., Ye, Q., Wagner, C., Wang, F., … Kulkarni, M. (2020). HACCLE: An Ecosystem for Building Secure Multi-Party Computations. *arXiv:2009.01489 [cs]*. Retrieved from arXiv:2009.01489

Bao, Y., Wei, G., Bračevac, O., Jiang, Y., He, Q., & Rompf, T. (2021). Reachability types: tracking aliasing and separation in higher-order functional programs. *Proceedings of the ACM on Programming Languages*, *5*, 139:1–139:32. https://doi.org/10.1145/3485516

Barbar, M., & Sui, Y. (n.d.-a). Compacting Points-To Sets through Object Clustering, *5*, 27.

Barbar, M., & Sui, Y. (n.d.-b). Hash Consed Points-To Sets, 24.

Barbar, M., Sui, Y., & Chen, S. (2020). Flow-Sensitive Type-Based Heap Cloning. *34th European Conference on Object-Oriented Programming*, 26.

Barbar, M., Sui, Y., & Chen, S. (2021). Object Versioning for Flow-Sensitive Pointer Analysis. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (pp. 222–235). Seoul, Korea (South): IEEE. https://doi.org/10.1109/CGO51591.2021.9370334

Bardin, S., Bjørner, N., & Cadar, C. (2019). Bringing CP, SAT and SMT together: Next Challenges in Constraint Solving (Dagstuhl Seminar 19062). *Dagstuhl Reports*, *9*(2), 27–47. https://doi.org/10.4230/DagRep.9.2.27

Barras, B. (2010). Sets in Coq, Coq in Sets. *Journal of Formalized Reasoning*, *3*(1), 29–48. https://doi.org/10.6092/issn.1972-5787/1695

Barriere, A., & Appel, A. (n.d.). VST Verification of B+Trees with Cursors, 19.

Bartell, S. (2021). *Optimizing Whole Programs for Code Size* (phdthesis). University of Illinois Urbana–Champaign, Urbana, Illinois.

Bartha, S., Cheney, J., & Belle, V. (2021). One Down, 699 to Go: or, synthesising compositional desugarings. *arXiv:2109.06114 [cs]*. Retrieved from arXiv:2109.06114

Barthe, G., Blazy, S., Hutin, R., & Pichardie, D. (2021). Secure Compilation of Constant-Resource Programs. In *IEEE 34th Computer Security Foundations Symposium (CSF)*. Dubrovnik, Croatia. Retrieved from https://hal.archives-ouvertes.fr/hal-03221440

Barthe, G., Gregoire, B., & Laporte, V. (2018). Secure Compilation of Side-Channel Countermeasures: The Case of Cryptographic "Constant-Time." In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)* (pp. 328–343). Oxford: IEEE. https://doi.org/10.1109/CSF.2018.00031

Barthe, G., Gregoire, B., Laporte, V., & Priya, S. (2021). *Structured Leakage and Applications to Cryptographic Constant-Time and Cost* (No. 650). Retrieved from http://eprint.iacr.org/2021/650

Basu, S. (2018). Languages for Path-Based Network Programming. https://doi.org/10.7298/X4057D4D

Batty Mark. (2017). Compositional relaxed concurrency. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150406. https://doi.org/10.1098/rsta.2015.0406

Batz, K., Fesefeldt, I., Jansen, M., Katoen, J.-P., Keßler, F., Matheja, C., & Noll, T. (2022). Foundations for Entailment Checking in Quantitative Separation Logic (extended version). *arXiv:2201.11464 [cs]*. Retrieved from arXiv:2201.11464

Baudin, P., Bobot, F., Bühler, D., Correnson, L., Kirchner, F., Kosmatov, N., … Williams, N. (2021). The dogged pursuit of bug-free C programs: the Frama-C software analysis platform. *Communications of the ACM*, *64*(8), 56–68. https://doi.org/10.1145/3470569

Bauer, A., & Petković, A. (2021). An extensible equality checking algorithm for dependent type theories. *arXiv:2103.07397 [cs, Math]*. Retrieved from arXiv:2103.07397

Beckert, B., Hähnle, R., & Schmitt, P. H. (Eds.). (2006). *Verification of Object-Oriented Software. The KeY Approach* (Vol. 4334). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69061-0

Beckett, R., Greenberg, M., & Walker, D. (2016). Temporal NetKAT. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 386–401). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2908080.2908108

Bedford, A. (2017). Coqatoo: Generating Natural Language Versions of Coq Proofs. *arXiv:1712.03894 [cs]*. Retrieved from arXiv:1712.03894

Bedford, A. (n.d.). Coqatoo: Generating Natural Language Versions of Coq Proofs - Slides, 16.

Beeson, M., Narboux, J., & Wiedijk, F. (2019). Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, *85*(2), 213–257. https://doi.org/10.1007/s10472-018-9606-x

Beg, A., & Butterfield, A. (2020). *Working Document for state of the art - formality meets autonomy/robotics*. https://doi.org/10.13140/RG.2.2.30437.22249

Bégay, P.-L., Crégut, P., & Monin, J.-F. (2021). Developing and Certifying Datalog Optimizations in Coq/MathComp, 15.

Bengtson, J., Bhargavan, K., Fournet, C., Gordon, A. D., & Maffeis, S. (2011). Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems*, *33*(2), 1–45. https://doi.org/10.1145/1890028.1890031

Benzaken, V., Cohen-Boulakia, S., Contejean, É., Keller, C., & Zucchini, R. (2021). A Coq Formalization of Data Provenance, 18.

Benzmüller, C. (2022). A Simplified Variant of G\"odel's Ontological Argument. *arXiv:2202.06264 [cs, Math]*. Retrieved from arXiv:2202.06264

Berdine, J., Calcagno, C., & O'Hearn, P. W. (2006). Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In F. S. de Boer, M. M. Bonsangue, S. Graf, & W.-P. de Roever (Eds.), *Formal Methods for Components and Objects* (pp. 115–137). Springer Berlin Heidelberg.

Bereczky, P., Chen, X., Horpácsi, D., Mizsei, T. B., Peña, L., & Tusil, J. (2022). Mechanizing Matching Logic in Coq. *arXiv:2201.05716 [cs]*. Retrieved from arXiv:2201.05716

Beringer, L., Stewart, G., Dockins, R., & Appel, A. W. (2014). Verified Compilation for Shared-Memory C. In Z. Shao (Ed.), *Programming Languages and Systems* (Vol. 8410, pp. 107–127). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-54833-8_7,

Bernhard, L., Rodler, M., Holz, T., & Davi, L. (2022). xTag: Mitigating Use-After-Free Vulnerabilities via Software-Based Pointer Tagging on Intel x86-64. *arXiv:2203.04117 [cs]*. Retrieved from arXiv:2203.04117

Bertino, E., Bliss, D., Lopresti, D., Peterson, L., & Schulzrinne, H. (2021). Computing Research Challenges in Next Generation Wireless Networking. *arXiv:2101.01279 [cs]*. Retrieved from arXiv:2101.01279

Bertoni, G., Daemen, J., & Peeters, M. (n.d.). *Sponge Functions*. STMicroelectronics, Radbound University. Retrieved from https://keccak.team/files/SpongeFunctions.pdf

Bertot, Y., & Castéran, P. (2004). *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Berlin ; New York: Springer. Retrieved from http://www.labri.fr/perso/casteran/CoqArt/index.html

Beyer, D., Haltermann, J., Lemberger, T., & Wehrheim, H. (2022). Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR, 13.

Bhargavan, K., Bichhawat, A., Do, Q. H., Hosseyni, P., Küsters, R., Schmitz, G., & Würtele, T. (2021). A Tutorial-Style Introduction to DY*. In D. Dougherty, J. Meseguer, S. A. Mödersheim, & P. Rowe (Eds.), *Protocols, Strands, and Logic* (Vol. 13066, pp. 77–97). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-91631-2_4

Bhargavan, K., Bichhawat, A., Do, Q. H., Hosseyni, P., Küsters, R., Schmitz, G., & Würtele, T. (n.d.-a). DY* : A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code, 20.

Bhargavan, K., Bichhawat, A., Do, Q., Hosseyni, P., Küsters, R., Schmitz, G., & Würtele, T. (n.d.-b). DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code, 21.

Bhargavan, K., Cheval, V., & Wood, C. (n.d.). Handshake Privacy for TLS 1.3 - Technical report, 53.

Bhat, S., & Grosser, T. (2022). Lambda the Ultimate SSA: Optimizing Functional Programs in SSA. Retrieved from https://arxiv.org/abs/2201.07272v1

Bidmeshki, M.-M., & Makris, Y. (2015). VeriCoq: A Verilog-to-Coq converter for proof-carrying hardware automation. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 29–32). Lisbon, Portugal: IEEE. https://doi.org/10.1109/ISCAS.2015.7168562

Bílý, A., Matheja, C., & Müller, P. (2021). Flexible Refinement Proofs in Separation Logic. *arXiv:2110.13559 [cs]*. Retrieved from arXiv:2110.13559

Birkedal, L. (n.d.). A Taste of Categorical Logic — Tutorial Notes, 41.

Birkedal, L., & Bizjak, A. (n.d.-a). Iris Tutorial. Retrieved February 1, 2019, from https://iris-project.org/tutorial-material.html

Birkedal, L., & Bizjak, A. (n.d.-b). Lecture Notes on Iris: Higher-Order Concurrent Separation Logic, 138.

Birkedal, L., Bizjak, A., Clouston, R., Grathwohl, H. B., Spitters, B., & Vezzosi, A. (2017). Guarded Cubical Type Theory. *arXiv:1611.09263 [cs, Math]*. Retrieved from arXiv:1611.09263

Birkedal, L., Dinsdale-Young, T., Guéneau, A., Jaber, G., Svendsen, K., & Tzevelekos, N. (2021). Theorems for Free from Separation Logic Specifications, *1*(1), 28.

Bishop, S., Fairbairn, M., Mehnert, H., Norrish, M., Ridge, T., Sewell, P., … Wansbrough, K. (2018). Engineering with Logic: Rigorous Test-Oracle Specification and Validation for TCP/IP and the Sockets API. *J. ACM*, *66*(1), 1:1–1:77. https://doi.org/10.1145/3243650

Bizjak, A., Gratzer, D., Krebbers, R., & Birkedal, L. (2019). Iron: managing obligations in higher-order concurrent separation logic. *Proceedings of the ACM on Programming Languages*, *3*, 1–30. https://doi.org/10.1145/3290378

Bjørner, D. (2017). Manifest domains: analysis and description. *Formal Aspects of Computing*, *29*(2), 175–225. https://doi.org/10.1007/s00165-016-0385-z

Bjørner, D., & Havelund, K. (2014). 40 Years of Formal Methods: Some Obstacles and Some Possibilities? In C. Jones, P. Pihlajasaari, & J. Sun (Eds.), *FM 2014: Formal Methods* (Vol. 8442, pp. 42–61). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-06410-9_4

Bjorner, N., Foster, N., Godfrey, P. B., & Zave, P. (2015). Formal Foundations for Networking (Dagstuhl Seminar 15071). *Dagstuhl Reports*, *5*(2), 44–63. https://doi.org/10.4230/DagRep.5.2.44

Bjørner, N., Moura, L. de, Nachmanson, lev, & Wintersteiger, C. (n.d.). Programming Z3. Retrieved from http://theory.stanford.edu/~nikolaj/programmingz3.html

Blanchard, A., Loulergue, F., & Kosmatov, N. (2017). From Concurrent Programs to Simulating Sequential Programs: Correctness of a Transformation. *Electronic Proceedings in Theoretical Computer Science*, *253*, 109–123. https://doi.org/10.4204/EPTCS.253.9

Blanchard, A., Loulergue, F., & Kosmatov, N. (2019). Towards Full Proof Automation in Frama-C Using Auto-active Verification. In J. M. Badger & K. Y. Rozier (Eds.), *NASA Formal Methods* (Vol. 11460, pp. 88–105). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-20652-9_6

Blatter, L., Kosmatov, N., Le Gall, P., Prevosto, V., & Petiot, G. (2018). Static and Dynamic Verification of Relational Properties on Self-composed C Code. In C. Dubois & B. Wolff (Eds.), *Tests and Proofs* (pp. 44–62). Springer International Publishing.

Blom, S., Darabi, S., Huisman, M., & Safari, M. (2021). Correct program parallelisations. *International Journal on Software Tools for Technology Transfer*. https://doi.org/10.1007/s10009-020-00601-z

Blouin, A., & Jezequel, J.-M. (2021). Interacto: A Modern User Interaction Processing Model. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2021.3083321

Bodden, E. (2012). Inter-procedural data-flow analysis with IFDS/IDE and Soot. In *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis - SOAP '12* (pp. 3–8). Beijing, China: ACM Press. https://doi.org/10.1145/2259051.2259052

Bodden, E. (2018). The secret sauce in efficient and precise static analysis: the beauty of distributive, summary-based static analyses (and how to master them). In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops* (pp. 85–93). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3236454.3236500

Bodenmüller, S., Schellhorn, G., Bitterlich, M., & Reif, W. (2021). Flashix: Modular Verification of a Concurrent and Crash-Safe Flash File System. In A. Raschke, E. Riccobene, & K.-D. Schewe (Eds.), *Logic, Computation and Rigorous Methods* (Vol. 12750, pp. 239–265). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-76020-5_14

Bodin, M., Charguéraud, A., Filaretti, D., Gardner, P., Maffeis, S., Naudziuniene, D., … Smith, G. (2014). A trusted mechanised JavaScript specification. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 87–100). San Diego California USA: ACM. https://doi.org/10.1145/2535838.2535876

Boer. (n.d.). Formal analysis of the Java Collections framework, 108.

Bohrer, B., Tan, Y. K., Mitsch, S., Myreen, M. O., & Platzer, A. (2018). VeriPhy: verified controller executables from verified cyber-physical system models. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2018* (pp. 617–630). Philadelphia, PA, USA: ACM Press. https://doi.org/10.1145/3192366.3192406

Boldo, S., Faissole, F., & Chapoutot, A. (2017). Round-off Error Analysis of Explicit One-Step Numerical Integration Methods. In *24th IEEE Symposium on Computer Arithmetic*. London, United Kingdom. https://doi.org/10.1109/ARITH.2017.22

Boldo, S., Faissole, F., & Chapoutot, A. (2018). *Round-off error and exceptional behavior analysis of explicit Runge-Kutta methods*. Retrieved from https://hal.archives-ouvertes.fr/hal-01883843

Boldo, S., Lelay, C., & Melquiond, G. (2012). Improving Real Analysis in Coq: A User-Friendly Approach to Integrals and Derivatives. In C. Hawblitzel & D. Miller (Eds.), *Certified Programs and Proofs* (Vol. 7679, pp. 289–304). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35308-6_22

Boldo, S., Lelay, C., & Melquiond, G. (2013). Coquelicot: A User-Friendly Library of Real Analysis for Coq. Retrieved from https://hal.inria.fr/hal-00860648/document

Boldo, S., Lelay, C., & Melquiond, G. (2016). Formalization of Real Analysis: A Survey of Proof Assistants and Libraries. *Mathematical Structures in Computer Science*, *26*(7), 1196–1233. https://doi.org/10.1017/S0960129514000437

Bonifacio, R., Krüger, S., Narasimhan, K., Bodden, E., & Mezini, M. (2021). Dealing with Variability in API Misuse Specification. *arXiv:2105.04950 [cs]*. Retrieved from arXiv:2105.04950

Bora, U., Vaishay, S., Joshi, S., & Upadrasta, R. (2021). OpenMP aware MHP Analysis for Improved Static Data-Race Detection. *arXiv:2111.04259 [cs]*. Retrieved from arXiv:2111.04259

Bordg, A., Lachnitt, H., & He, Y. (2020). Certified Quantum Computation in Isabelle/HOL. *Journal of Automated Reasoning*, 1–19. https://doi.org/10.1007/s10817-020-09584-7

Bornebusch, F. (n.d.). COQ meets Clash: PROPOSING A HARDWARE DESIGN SYNTHESIS FLOW THAT COMBINES PROOF ASSISTANTS WITH FUNCTIONAL HARDWARE DESCRIPTION LANGUAGES, 186.

Bornholt, J., Joshi, R., Astrauskas, V., Cully, B., Kragl, B., Markle, S., … Tasiran, S. (2021). Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3, 15.

Bosamiya, J., Lim, W. S., & Parno, B. (n.d.). Provably-Safe Multilingual Software Sandboxing using WebAssembly, 18.

Bosshart, P., Daly, D., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., … Walker, D. (2014). Programming Protocol-Independent Packet Processors. *arXiv:1312.1719 [cs]*. Retrieved from arXiv:1312.1719

Boston, B., Breese, S., Dodds, J., Dodds, M., Huffman, B., Petcher, A., & Stefanescu, A. (n.d.). Verified Cryptographic Code for Everybody, 23.

Boudol, G. (2009). Secure Information Flow as a Safety Property. In P. Degano, J. Guttman, & F. Martinelli (Eds.), *Formal Aspects in Security and Trust* (Vol. 5491, pp. 20–34). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01465-9_2

Boulier, S., Pédrot, P.-M., & Tabareau, N. (2017). The next 700 syntactical models of type theory (pp. 182–194). https://doi.org/10.1145/3018610.3018620

Boulmé, S. (n.d.). *Formally Verified Defensive Programming* (phdthesis).

Bourdoncle, F. (1993). Efficient chaotic iteration strategies with widenings (pp. 128–141). Springer-Verlag.

Bourgeat, T., Clester, I., Erbsen, A., Gruetter, S., Wright, A., & Chlipala, A. (2021). A Multipurpose Formal RISC-V Specification. *arXiv:2104.00762 [cs]*. Retrieved from arXiv:2104.00762

Bowman, J. (n.d.). *J1: a small Forth CPU Core for FPGAs*.

Brady, E. (2013). Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, *23*(5), 552–593. https://doi.org/10.1017/S095679681300018X

Brahmi, A., Delmas, D., Essoussi, M. H., Randimbivololona, F., Atki, A., & Marie, T. (2018). Formalise to automate: deployment of a safe and cost-efficient process for avionics software. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*. Toulouse, France. Retrieved from https://hal.archives-ouvertes.fr/hal-01708332

Brahmi, A., Delmas, D., Essoussi, M. H., Randimbivololona, F., Informatics, C., Nauzere, L., … Marie, T. (n.d.). Formalise to automate: deployment of a safe and cost-efficient process for avionics software -Extended, 17.

Braibant, T., & Chlipala, A. (2013). Formal Verification of Hardware Synthesis. *arXiv:1301.4779 [cs]*, *8044*, 213–228. https://doi.org/10.1007/978-3-642-39799-8_14

Breitner, J., Spector-Zabusky, A., Li, Y., Rizkallah, C., Wiegley, J., & Weirich, S. (2018). Ready, Set, Verify! Applying Hs-to-coq to Real-world Haskell Code (Experience Report). *Proc. ACM Program. Lang.*, *2*, 89:1–89:16. https://doi.org/10.1145/3236784

Bridges, D. S., & Vîţă, L. S. (2011). *Apartness and Uniformity*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-22415-7

Brockschmidt, M., Cook, B., Ishtiaq, S., Khlaaf, H., & Piterman, N. (2016). T2: Temporal Property Verification. In M. Chechik & J.-F. Raskin (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 387–393). Springer Berlin Heidelberg.

Brookes, S. (2007). A semantics for concurrent separation logic. *Theoretical Computer Science*, *375*(1), 227–270. https://doi.org/10.1016/j.tcs.2006.12.034

Brookes, S., & O'Hearn, P. W. (2016). Concurrent Separation Logic. *ACM SIGLOG News*, *3*(3), 47–65. https://doi.org/10.1145/2984450.2984457

Brotherston, J., Gorogiannis, N., & Kanovich, M. (2016). Biabduction (and Related Problems) in Array Separation Logic. *arXiv:1607.01993 [cs, Math]*. Retrieved from arXiv:1607.01993

Brown, C. (n.d.). Semi-Automatic Ladderisation: Improving Code Security through Rewriting and Dependent Types. Retrieved December 7, 2021, from https://research-repository.st-andrews.ac.uk/bitstream/handle/10023/24384/Brown_2021_Semi_automatic_ladderisation_PEPM2022_AAM.pdf?sequence=1&isAllowed=y

Bruening, D., Zhao, Q., & Amarasinghe, S. (n.d.). Transparent Dynamic Instrumentation, 11.

Bugariu, A., Ter-Gabrielyan, A., & Müller, P. (2021). Identifying Overly Restrictive Matching Patterns in SMT-based Program Verifiers. *arXiv:2105.04385 [cs]*. Retrieved from arXiv:2105.04385

Bugliesi, M., & Giunti, M. (2007). Secure implementations of typed channel abstractions. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '07* (p. 251). Nice, France: ACM Press. https://doi.org/10.1145/1190216.1190253

Burlò, C. B., Francalanza, A., & Scalas, A. (2021). On the Monitorability of Session Types, in Theory and Practice. *arXiv:2105.06291 [cs]*. Retrieved from arXiv:2105.06291

Busi, M., & Galletta, L. (2019). A Brief Tour of Formally Secure Compilation. In *Proceedings of the Third Italian Conference on Cyber Security* (p. 13). Pisa, Italy: CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-2315/paper03.pdf

Busse, F., Nowack, M., & Cadar, C. (2020). Running symbolic execution forever. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 63–74). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3395363.3397360

Cadar, C., Dunbar, D., & Engler, D. (2008). KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation* (pp. 209–224). USA: USENIX Association.

Cai, Y., Yao, P., & Zhang, C. (2021). Canary: Practical Static Detection of Inter-thread Value-Flow Bugs, 15.

Cai, Y., Yun, H., Wang, J., Qiao, L., & Palsberg, J. (2021). Sound and efficient concurrency bug prediction. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 255–267). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3468264.3468549

Calcagno, C., Distefano, D., Dubreil, J., & O'Hearn, P. (n.d.). Moving Fast with Software Verification.Facebook Research. Retrieved February 1, 2019, from https://research.fb.com/publications/moving-fast-with-software-verification

Calcagno, C., Distefano, D., O'Hearn, P. W., & Yang, H. (2011). Compositional Shape Analysis by Means of Bi-Abduction. *Journal of the ACM*, *58*(6), 26:1–26:66. https://doi.org/10.1145/2049697.2049700

Campbell, E. H., Ramamurthy, V., Hallahan, W. T., Srikumar, P., Cascone, C., Liu, J., … Foster, N. (n.d.). Avenir: Managing Data Plane Diversity with Control Plane Synthesis, 21.

Canini, M., Venzano, D., Peresˇımi, P., Kostic, D., & Rexford, J. (2012). A NICE Way to Test OpenFlow Applications, 14.

Cao, Q., Beringer, L., Gruetter, S., Dodds, J., & Appel, A. W. (2018). VST-Floyd: A Separation Logic Tool to Verify Correctness of C Programs. *J. Autom. Reason.*, *61*(1), 367–422. https://doi.org/10.1007/s10817-018-9457-5

Carbonneaux, Q., Zilberstein, N., Klee, C., O'Hearn, P. W., & Nardelli, F. Z. (2021). Applying Formal Verification to Microkernel IPC at Meta, 14.

Carnegie Mellon University research repository - Browse. (n.d.). Retrieved February 1, 2021, from https://kilthub.cmu.edu/etd

Carneiro, M. (2019a). Specifying verified x86 software from scratch. *arXiv:1907.01283 [cs]*. Retrieved from arXiv:1907.01283

Carneiro, M. (2019b). The Type Theory of Lean. Retrieved from https://github.com/digama0/lean-type-theory/releases/download/v1.0/main.pdf

Carneiro, M. (2020a). *Metamath Zero*. Retrieved from https://github.com/digama0/mm0

Carneiro, M. (2020b). Metamath Zero: Designing a Theorem Prover Prover. In C. Benzmüller & B. Miller (Eds.), *Intelligent Computer Mathematics* (pp. 71–88). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-53518-6_5

Casado, M., Foster, N., & Guha, A. (2014). Abstractions for software-defined networks. *Communications of the ACM*, *57*(10), 86–95. https://doi.org/10.1145/2661061.2661063

Casinghino, C. (2014). *Combining Proofs and Programs* (phdthesis). University of Pennsylvania, Philadelphia, PA, USA. Retrieved from https://www.seas.upenn.edu/~sweirich/papers/casinghino-thesis.pdf

Casinghino, C., Sjöberg, V., & Weirich, S. (2014). Combining Proofs and Programs in a Dependently Typed Language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 33–45). New York, NY, USA: ACM. https://doi.org/10.1145/2535838.2535883

Cassez, F. (2021). Verification of the Incremental Merkle Tree Algorithm with Dafny. *arXiv:2105.06009 [cs]*. Retrieved from arXiv:2105.06009

Cassez, F., Fuller, J., & Asgaonkar, A. (2021). Formal Verification of the Ethereum 2.0 Beacon Chain. *arXiv:2110.12909 [cs]*. Retrieved from arXiv:2110.12909

Castéran, P., Damour, J., Palmskog, K., Pit-Claudel, C., & Zimmermann, T. (2021). *Hydras & Co.: Formalized mathematics in Coq for inspiration and entertainment*. Retrieved from https://hal.archives-ouvertes.fr/hal-03404668

Cauligi, S., Disselkoen, C., Moghimi, D., Barthe, G., & Stefan, D. (2021). SoK: Practical Foundations for Spectre Defenses. *arXiv:2105.05801 [cs]*. Retrieved from arXiv:2105.05801

Cauligi, S. R. (n.d.). *Foundations for Speculative Side Channels* (phdthesis). UC San Diego. Retrieved from https://escholarship.org/uc/item/64n9f44x

Cavallo, E. (2021). Higher Inductive Types and Internal Parametricity for Cubical Type Theory, 322.

Cecchetti, E., Yao, S., Ni, H., & Myers, A. C. (n.d.). Compositional Security for Reentrant Applications (Technical Report), 56.

Celik, A., Palmskog, K., Parovic, M., Jesus Gallego Arias, E., & Gligoric, M. (2019). Mutation Analysis for Coq. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 539–551). San Diego, CA, USA: IEEE. https://doi.org/10.1109/ASE.2019.00057

CerCo - Certified Complexity. (n.d.). Retrieved January 13, 2020, from http://cerco.cs.unibo.it/

Chailloux, E., Manoury, P., & Pagano, B. (2000). *Developing Applications with Objective Caml*. Paris: O'Reilly.

Chajed, T. (n.d.). Record Updates in Coq, 2.

Chajed, T., Tassarotti, J., Kaashoek, M. F., Theng, M., Jung, R., & Zeldovich, N. (n.d.). GoJournal: a verified, concurrent, crash-safe journaling system, 17.

Chajed, T., Tassarotti, J., Kaashoek, M. F., & Zeldovich, N. (2019). Verifying concurrent, crash-safe systems with Perennial. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (pp. 243–258). Huntsville Ontario Canada: ACM. https://doi.org/10.1145/3341301.3359632

Chajed, T., Tassarotti, J., Kaashoek, M. F., & Zeldovich, N. (n.d.). Verifying concurrent Go code in Coq with Goose, 3.

Chaliasos, S., Sotiropoulos, T., Drosos, G.-P., Mitropoulos, C., Mitropoulos, D., & Spinellis, D. (2021a). Well-typed programs can go wrong: a study of typing-related bugs in JVM compilers. *Proceedings of the ACM on Programming Languages*, *5*, 123:1–123:30. https://doi.org/10.1145/3485500

Chaliasos, S., Sotiropoulos, T., Drosos, G.-P., Mitropoulos, C., Mitropoulos, D., & Spinellis, D. (2021b). Well-typed programs can go wrong: a study of typing-related bugs in JVM compilers. *Proceedings of the ACM on Programming Languages*, *5*, 1–30. https://doi.org/10.1145/3485500

Chaliasos, S., Sotiropoulos, T., Drosos, G.-P., Mitropoulos, C., Mitropoulos, D., & Spinellis, D. (2021c). Well-typed programs can go wrong: a study of typing-related bugs in JVM compilers. *Proceedings of the ACM on Programming Languages*, *5*, 1–30. https://doi.org/10.1145/3485500

Chang, B.-Y. E., Drăgoi, C., Manevich, R., Rinetzky, N., & Rival, X. (2020). Shape Analysis. *Foundations and Trends® in Programming Languages*, *6*(1), 1–158. https://doi.org/10.1561/2500000037

Chapman, R. (n.d.). The Fumble Programmer. Retrieved from https://proteancode.com/wp-content/uploads/2018/02/the_fumble_programmer.pdf

Charguéraud, A. (2010a). *Characteristic Formulae for Mechanized Program Verification* (phdthesis). UNIVERSITÉ PARIS.DIDEROT, Paris, France.

Charguéraud, A. (2010b). Program Verification Through Characteristic Formulae. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming* (pp. 321–332). New York, NY, USA: ACM. https://doi.org/10.1145/1863543.1863590

Charguéraud, A. (2011). Characteristic Formulae for the Verification of Imperative Programs. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming* (pp. 418–430). New York, NY, USA: ACM. https://doi.org/10.1145/2034773.2034828

Charguéraud, A., Filliâtre, J.-C., Lourenço, C., & Pereira, M. (2019). GOSPEL—Providing OCaml with a Formal Specification Language. In M. H. ter Beek, A. McIver, & J. N. Oliveira (Eds.), *Formal Methods – The Next 30 Years* (Vol. 11800, pp. 484–501). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-30942-8_29

Chen, J. (2021). Homotopy Type Theory in Isabelle. *arXiv:2002.09282 [cs]*. Retrieved from arXiv:2002.09282

Chen, T., Heo, K., & Raghothaman, M. (2021). Boosting static analysis accuracy with instrumented test executions. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1154–1165). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3468264.3468626

Chen, X., Lucanu, D., & Roşu, G. (2020). Initial Algebra Semantics in Matching Logic. Retrieved from https://www.ideals.illinois.edu/handle/2142/107781

Chen, X., & Roşu, G. (2020). A general approach to define binders using matching logic. *Proceedings of the ACM on Programming Languages*, *4*, 1–32. https://doi.org/10.1145/3408970

Chen, Y. (n.d.). Project Report on DeepSpecDB, 35.

Chen, Y., Li, P., Xu, J., Guo, S., Zhou, R., Zhang, Y., … Lu. (2019). SAVIOR: Towards Bug-Driven Hybrid Testing. *arXiv:1906.07327 [cs]*. Retrieved from arXiv:1906.07327

Chen, Y., Liu, J., Feng, Y., & Bodik, R. (2022). Tree Traversal Synthesis Using Domain-SpecificSymbolic Compilation, 13.

Chen, Y., Yang, C., Zhang, X., Xiong, Y., Tang, H., Wang, X., & Zhang, L. (n.d.). Accelerating Program Analyses in Datalog by Merging Library Facts, 24.

Chhak, C. H. R., Tolmach, A., & Anderson, S. (2020). Towards Formally Verified Compilation of Tag-Based Policy Enforcement. *arXiv:2012.10313 [cs]*. Retrieved from arXiv:2012.10313

Chihani, Z. (n.d.). Certification of First-order proofs in classical and intuitionistic logics, 167.

Chin, J., & Pearce, D. J. (2021). Finding Bugs with Specification-Based Testing is Easy!, 35.

Chiplunkar, S., Pit-Claudel, C., & Chlipala, A. (n.d.). Automated Synthesis of Verified Firewalls, 3.

Chiricescu, S., DeHon, A., Demange, D., Iyer, S., Kliger, A., Morrisett, G., … Wittenberg, D. (2013). SAFE: A clean-slate architecture for secure systems. In *2013 IEEE International Conference on Technologies for Homeland Security (HST)* (pp. 570–576). Waltham, MA, USA: IEEE. https://doi.org/10.1109/THS.2013.6699066

Chlipala, A. (2013). *Certified programming with dependent types: a pragmatic introduction to the Coq proof assistant*. Cambridge, MA: The MIT Press. Retrieved from http://adam.chlipala.net/cpdt/

Chlipala, A. (2019). *Formal Reasoning About Programs - Github*. Retrieved from https://github.com/achlipala/frap

Chlipala, A. (n.d.-a). An Introduction to Programming and Proving with Dependent Types in Coq. *Journal of Formalized Reasoning*, *3*(2), 93.

Chlipala, A. (n.d.-b). Certified Programming with Dependent Types, 369.

Chlipala, A. (n.d.-c). Skipping the Binder Bureaucracy with Mixed Embeddings in a Semantics Course (Functional Pearl), *5*, 28.

Chlipala, A., Delaware, B., Duchovni, S., Gross, J., Pit-Claudel, C., Suriyakarn, S., … ye, K. (n.d.). THE END OF HISTORY? USING A PROOF ASSISTANT TO REPLACE LANGUAGE DESIGN WITH LIBRARY DESIGN. Retrieved February 1, 2019, from https://snapl.org/2017/abstracts/Chlipala.html

Choi, J., Vijayaraghavan, M., Sherman, B., Chlipala, A., & Arvind. (2017). Kami: A Platform for High-level Parametric Hardware Specification and Its Modular Verification. *Proc. ACM Program. Lang.*, *1*, 24:1–24:30. https://doi.org/10.1145/3110268

Chong, N., Cook, B., Eidelman, J., Kallas, K., Khazem, K., Monteiro, F. R., … Tuttle, M. R. (2021). Code-level model checking in the software development workflow at Amazon Web Services. *Software: Practice and Experience*, *n/a*. https://doi.org/10.1002/spe.2949

Choudhury, P. (n.d.). Towards a Formalization of Nominal Sets in Coq, 3.

Christakis, M., Müller, P., & Wüstholz, V. (2012). Collaborative Verification and Testing with Explicit Assumptions. In D. Giannakopoulou & D. Méry (Eds.), *FM 2012: Formal Methods* (pp. 132–146). Springer Berlin Heidelberg.

Christensen, M. A. (2021). *Programming Language Techniques for Improving ISA and HDL Design* (phdthesis). UC Santa Barbara, Santa Barbara, CA. Retrieved from https://escholarship.org/content/qt7sz5r3vd/qt7sz5r3vd.pdf

Cito, J., Dillig, I., Murali, V., & Chandra, S. (2021). Counterfactual Explanations for Models of Code. *arXiv:2111.05711 [cs]*. Retrieved from arXiv:2111.05711

Clark, P. L. (2019). The Instructor's Guide to Real Induction. *Mathematics Magazine*, *92*(2), 136–150. https://doi.org/10.1080/0025570X.2019.1549902

Clarkson, M. R., & Schneider, F. B. (2008). Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium* (pp. 51–65). https://doi.org/10.1109/CSF.2008.7

Cofer, D., Miller, S. P., & Collins, R. (n.d.). Formal Methods Case Studies for DO-333, 203.

Cohen, C. (2013). Pragmatic Quotient Types in Coq. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.), *Interactive Theorem Proving* (Vol. 7998, pp. 213–228). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39634-2_17

Collins, R. (n.d.). SECURE MATHEMATICALLY- ASSURED COMPOSITION OF CONTROL MODELS, 134.

Conchon, S., Coquereau, A., Iguernlala, M., & Mebsout, A. (2018). Alt-Ergo 2.2. In *SMT Workshop: International Workshop on Satisfiability Modulo Theories*. Oxford, United Kingdom. Retrieved from https://hal.inria.fr/hal-01960203

Conchon, S., & Iguernlala, M. (2016). Increasing Proofs Automation Rate of Atelier-B Thanks to Alt-Ergo. In T. Lecomte, R. Pinger, & A. Romanovsky (Eds.), *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification* (pp. 243–253). Springer International Publishing.

Consortium, P. (2020, June 11). P4_16 Language Specification. Retrieved January 5, 2021, from https://p4.org/p4-spec/docs/P4-16-working-spec.html

Consortium, P. (2021). P4 Language and Related Specifications. Retrieved January 13, 2021, from https://p4.org/specs/

Cooper, S. B., & Soskova, M. I. (Eds.). (2017). *The Incomputable*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-43669-2

Coq Coq correct! verification of type checking and erasure for Coq, in Coq. (n.d.). https://doi.org/10.1145/3371076

CoqEAL. (2020). CoqEAL. Retrieved from https://github.com/CoqEAL/CoqEAL

Coquelicot.Coquelicot. (n.d.). Retrieved November 3, 2019, from http://coquelicot.saclay.inria.fr/html/Coquelicot.Coquelicot.html

Costan, V., Lebedev, I., & Devadas, S. (2016). Sanctum: Minimal Hardware Extensions for Strong Software Isolation (pp. 857–874). Retrieved from https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

Costan, V., Lebedev, I., & Devadas, S. (2017a). Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture. *Foundations and Trends® in Electronic Design Automation*, *11*(1), 1–248. https://doi.org/10.1561/1000000051

Costan, V., Lebedev, I., & Devadas, S. (2017b). Secure Processors Part II: Intel SGX Security Analysis and MIT Sanctum Architecture. *Foundations and Trends® in Electronic Design Automation*, *11*(3), 249–361. https://doi.org/10.1561/1000000052

Costanzo, D., Shao, Z., & Gu, R. (2016). End-to-end verification of information-flow security for C and assembly programs. *ACM SIGPLAN Notices*, *51*(6), 648–664. https://doi.org/10.1145/2980983.2908100

Costanzo, D., Shao, Z., & Gu, R. (n.d.). End-to-End Verification of Information-Flow Security for C and Assembly Programs - Tech Report, 21. Retrieved from http://flint.cs.yale.edu/certikos/publications/security-tr.pdf

Costea, A., Tiwari, A., Chianasta, S., R, K., Roychoudhury, A., & Sergey, I. (2021). HIP-PODROME: Data Race Repair using Static Analysis Summaries. *arXiv:2108.02490 [cs]*. Retrieved from arXiv:2108.02490

Cousineau, D., & Dowek, G. (2007). Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. In S. R. Della Rocca (Ed.), *Typed Lambda Calculi and Applications* (Vol. 4583, pp. 102–117). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73228-0_9

Cousot, P., Cousot, R., Fähndrich, M., & Logozzo, F. (2013). Automatic Inference of Necessary Preconditions. In R. Giacobazzi, J. Berdine, & I. Mastroeni (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 128–148). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-35873-9_10

Cousot, P., Giacobazzi, R., & Ranzato, F. (2019). A2I: Abstract2 Interpretation. *Proc. ACM Program. Lang.*, *3*, 42:1–42:31. https://doi.org/10.1145/3290355

Covington, G. A., Naous, J., Erickson, D., & Mckeown, N. (n.d.). *Implementing an OpenFlow Switch on the NetFPGA platform.*

Coward, S., Paulson, L., Drane, T., & Morini, E. (n.d.). Formal Verification of Transcendental Fixed and Floating Point Algorithms using an Automatic Theorem Prover, 20.

Cowley, J. (2014). Job Analysis Results for Malicious-Code Reverse Engineers: A Case Study, 114.

Crain, T., Natoli, C., & Gramoli, V. (2021). *Red Belly: A Secure, Fair and Scalable Open Blockchain*. https://doi.org/10.1109/SP40001.2021.00087

Crary, K. (2017). Modules, Abstraction, and Parametric Polymorphism. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (pp. 100–113). New York, NY, USA: ACM. https://doi.org/10.1145/3009837.3009892

Crick, T., Hall, B. A., Ishtiaq, S., & Takeda, K. (2014). "Share and Enjoy": Publishing Useful and Usable Scientific Models. *arXiv:1409.0367 [cs]*. Retrieved from arXiv:1409.0367

Csoma, A., Sonkoly, B., Csikor, L., Németh, F., Gulyas, A., Tavernier, W., & Sahhaf, S. (2014). ESCAPE: extensible service chain prototyping environment using mininet, click, NETCONF and POX. In *Proceedings of the 2014 ACM conference on SIGCOMM* (pp. 125–126). Chicago Illinois USA: ACM. https://doi.org/10.1145/2619239.2631448

Curry, C., & Le, Q. L. (2020). Bi-Abduction for Shapes with Ordered Data. *arXiv:2006.10439 [cs]*. Retrieved from arXiv:2006.10439

Czajka, L., & Kaliszyk, C. (n.d.). CoqHammer: Strong Automation for Program Verification - CoqPL 2018. Retrieved January 31, 2019, from https://popl18.sigplan.org/event/coqpl-2018-coqhammer-strong-automation-for-program-verification

Dagand, P.-E., Berthou, G., Demange, D., & Risset, T. (2022). *A Formal Model of Interrupt-based Checkpointing with Peripherals* (Technical Report) (pp. 1–36). IRIF ; IRISA ; INSA RENNES. Retrieved from https://hal.archives-ouvertes.fr/hal-03557760

Daggitt, M. L., Kokke, W., Atkey, R., Arnaboldi, L., & Komendantskya, E. (2022). Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers. *arXiv:2202.05207 [cs]*. Retrieved from arXiv:2202.05207

Dang, H.-H., Jourdan, J.-H., Kaiser, J.-O., & Dreyer, D. (n.d.). RustBelt Meets Relaxed Memory, *4*, 29.

Danvy, O., & Filinski, A. (1990). Abstracting control. In *Proceedings of the 1990 ACM conference on LISP and functional programming* (pp. 151–160). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/91556.91622

Darais, D., & Van Horn, D. (2016). Constructive Galois Connections: Taming the Galois Connection Framework for Mechanized Metatheory. *arXiv:1511.06965 [cs]*. Retrieved from arXiv:1511.06965

Darvas, D., Majzik, I., & Blanco Viñuela, E. (2016). Formal Verification of Safety PLC Based Control Software. In *Proceedings of the 12th International Conference on Integrated Formal Methods - Volume 9681* (pp. 508–522). Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/978-3-319-33693-0_32

David Cristina, & Kroening Daniel. (2017). Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150403. https://doi.org/10.1098/rsta.2015.0403

Davie, B. (2020, December 14). Why 5G Matters.Systems Approach. Retrieved January 20, 2021, from http://www.systemsapproach.org/1/archives/12-2020

Dawes, J. H. (n.d.). Specifying Properties over Inter-Procedural, Source Code Level Behaviour of Programs, 20.

Debnath, J., Chau, S. Y., & Chowdhury, O. (2021). On Re-engineering the X.509 PKI with Executable Specification for Better Implementation Guarantees. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1388–1404). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3460120.3484793

Deducteam/Dedukti. (2019). Deducteam. Retrieved from https://github.com/Deducteam/Dedukti

Deducteam/Holide. (2019). Deducteam. Retrieved from https://github.com/Deducteam/Holide

Dedukti - a Logical Framework. (n.d.). Retrieved January 10, 2020, from https://deducteam.github.io/

Delahaye, D. (2000). A Tactic Language for the System Coq. In M. Parigot & A. Voronkov (Eds.), *Logic for Programming and Automated Reasoning* (Vol. 1955, pp. 85–95). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-44404-1_7

Delaware, B., Pit-Claudel, C., Gross, J., & Chlipala, A. (2015). Fiat: Deductive Synthesis of Abstract Data Types in a Proof Assistant. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 689–700). New York, NY, USA: ACM. https://doi.org/10.1145/2676726.2677006

Delaware, B., Suriyakarn, S., Pit–Claudel, C., Ye, Q., & Chlipala, A. (2018). Narcissus: Deriving Correct-By-Construction Decoders and Encoders from Binary Formats. Retrieved from https://arxiv.org/abs/1803.04870v2

Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Protzenko, J., Rastogi, A., Swamy, N., … Zinzindohoue, J. K. (2017). Implementing and Proving the TLS 1.3 Record Layer. Retrieved from https://www.microsoft.com/en-us/research/publication/implementing-proving-tls-1-3-record-layer/

Delignat-Lavaud, A., Fournet, C., Parno, B., Protzenko, J., Ramananandro, T., Bosamiya, J., … Zhou, Y. (n.d.). A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer, 17.

DeMeo, W. (2021a). The Agda Universal Algebra Library, Part 1: Foundation. *arXiv:2103.05581 [cs, Math]*. Retrieved from arXiv:2103.05581

DeMeo, W. (2021b). The Agda Universal Algebra Library, Part 2: Structure. *arXiv:2103.09092 [cs, Math]*. Retrieved from arXiv:2103.09092

De Millo, R. A., Lipton, R. J., & Perlis, A. J. (1979). Social Processes and Proofs of Theorems and Programs. *Commun. ACM*, *22*(5), 271–280. https://doi.org/10.1145/359104.359106

Derakhshan, F., Balzer, S., & Jia, L. (n.d.). Session Logical Relations for Noninterference, 31.

Dévai, G. (2009). Embedding a Proof System in Haskell. In *Central European Functional Programming School* (pp. 354–371). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17685-2_10

Devkota, S., Legendre, M., Kunen, A., Aschwanden, P., & Isaacs, K. E. (2021). CFGConf: Supporting high level requirements for visualizing Control Flow Graphs. *arXiv:2108.03047 [cs]*. Retrieved from arXiv:2108.03047

Dickerson, R., Ye, Q., & Delaware, B. (2020). RHLE: Modular Deductive Verification of Relational $\forall\exists$ Properties. *arXiv:2002.02904 [cs]*. Retrieved from arXiv:2002.02904

Di Cosmo, R., & Miller, D. (2019). Linear Logic. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2019). Metaphysics Research Lab, Stanford University. Retrieved from https://plato.stanford.edu/archives/sum2019/entries/logic-linear/

Dietsch, D., Heizmann, M., Klumpp, D., Naouar, M., Podelski, A., & Schätzle, C. (2021). Verification of Concurrent Programs Using Petri Net Unfoldings. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 174–195). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_9

Dietz, W., Li, P., Regehr, J., & Adve, V. (n.d.). Understanding Integer Overflow in C/C++, 11.

Dijkstra, E. W. (1975). Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Commun. ACM*, *18*(8), 453–457. https://doi.org/10.1145/360933.360975

Dijkstra, E. W., DeMillo, R. A., Lipton, R. J., & Perlis, A. J. (1978). On a Political Pamphlet from the Middle Ages. *SIGSOFT Softw. Eng. Notes*, *3*(2), 14–16. https://doi.org/10.1145/1005888.1005890

Din, C. C., Hähnle, R., Henrio, L., Johnsen, E. B., Pun, V. K. I., & Tarifa, S. L. T. (2022). LAGC Semantics of Concurrent Programming Languages. *arXiv:2202.12195 [cs]*. Retrieved from arXiv:2202.12195

Ding, Y., Suneja, S., Zheng, Y., Laredo, J., Morari, A., Kaiser, G., & Ray, B. (2021). VELVET: a noVel Ensemble Learning approach to automatically locate VulnErable sTatements. *arXiv:2112.10893 [cs]*. Retrieved from arXiv:2112.10893

Di, P., & Sui, Y. (2016). Accelerating Dynamic Data Race Detection Using Static Thread Interference Analysis. In *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores* (pp. 30–39). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2883404.2883405

Disselkoen, C., Cauligi, S., Tullsen, D., & Stefan, D. (n.d.). Finding and Eliminating Timing Side-Channels in Crypto Code with Pitchfork, 8.

Distefano, D., Fähndrich, M., Logozzo, F., & O'Hearn, P. W. (2019). Scaling static analyses at Facebook. *Communications of the ACM*, *62*(8), 62–70. https://doi.org/10.1145/3338112

Distefano, D., O'Hearn, P. W., & Yang, H. (2006). A Local Shape Analysis Based on Separation Logic. In H. Hermanns & J. Palsberg (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems* (Vol. 3920, pp. 287–302). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/11691372_19

Djoudi, A., Hána, M., & Kosmatov, N. (2021). Formal Verification of a JavaCard Virtual Machine with Frama-C. In M. Huisman, C. Păsăreanu, & N. Zhan (Eds.), *Formal Methods* (Vol. 13047, pp. 427–444). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-90870-6_23

Dockins, R., Hobor, A., & Appel, A. W. (2009). A Fresh Look at Separation Algebras and Share Accounting. In Z. Hu (Ed.), *Programming Languages and Systems* (Vol. 5904, pp. 161–177). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-10672-9_13

Doenges, R., Arashloo, M. T., Bautista, S., Chang, A., Ni, N., Parkinson, S., … Foster, N. (2020). Petr4: Formal Foundations for P4 Data Planes. *arXiv:2011.05948 [cs]*. Retrieved from arXiv:2011.05948

Downey, R. G., & Hirschfeldt, D. R. (2010). *Algorithmic Randomness and Complexity*. New York, NY: Springer New York. https://doi.org/10.1007/978-0-387-68441-3

Dral, J. (2022). *Verified Compiler Optimisations* (Masters). Utrecht University.

Dreyer, D., & O'Hearn, P. W. (n.d.). Concurrent Incorrectness Separation Logic. Retrieved December 7, 2021, from https://research.fb.com/wp-content/uploads/2021/11/Concurrent-Incorrectness-Separation-Logic.pdf

Dross, C., Furia, C. A., Huisman, M., Monahan, R., & Müller, P. (2020). VerifyThis 2019: A Program Verification Competition (Extended Report). *arXiv:2008.13610 [cs]*. Retrieved from arXiv:2008.13610

D'Silva, V., Payer, M., & Song, D. (2015). The Correctness-Security Gap in Compiler Optimization. In *2015 IEEE Security and Privacy Workshops* (pp. 73–87). https://doi.org/10.1109/SPW.2015.33

Dubut, J., & Yamada, A. (2022). Fixed Points Theorems for Non-Transitive Relations. *Logical Methods in Computer Science*, *Volume 18, Issue 1*, 6809. https://doi.org/10.46298/lmcs-18(1:30)2022

Dumitrescu, D., Stoenescu, R., Negreanu, L., & Raiciu, C. (2020). bf4: towards bug-free P4 programs. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (pp. 571–585). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3387514.3405888

Easterbrook, S., & Callahan, J. (1997). Formal methods for V & V of partial specifications: an experience report. In *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering* (pp. 160–168). https://doi.org/10.1109/ISRE.1997.566865

Easterbrook, S., & Callahan, J. (n.d.). Formal Methods for V&V of partial specifications: An experience report, 9.

Ebner, G., Ullrich, S., Roesch, J., Avigad, J., & Moura, L. de. (2017). A Metaprogramming Framework for Formal Verification. *Proc. ACM Program. Lang.*, *1*, 34:1–34:29. https://doi.org/10.1145/3110278

Echenim, M., & Peltier, N. (2022). A Proof Procedure For Separation Logic With Inductive Definitions and Theory Reasoning. *arXiv:2201.13227 [cs]*. Retrieved from arXiv:2201.13227

Efficient Fixpoint Computation for Abstract Interpretation - University of California, Davis. (n.d.). Retrieved November 16, 2021, from https://video.ucdavis.edu/media/Efficient+Fixpoint+Computation+for+Abstract+Interpretation/1_1trufm5a

Egolf, D., Lasser, S., & Fisher, K. (n.d.). Verbatim: A Verified Lexer Generator, 9.

Eichholz, M. (n.d.). Dependently-Typed Data Plane Programming. Retrieved December 7, 2021, from https://ericthewry.github.io/pdfs/pi4.pdf

Eichholz, M., Campbell, E. H., Krebs, M., Foster, N., & Mezini, M. (2022). Dependently-typed data plane programming. *Proceedings of the ACM on Programming Languages*, *6*, 40:1–40:28. https://doi.org/10.1145/3498701

Eisenberg, R. A. (2016). *DEPENDENT TYPES IN HASKELL: THEORY AND PRACTICE* (phdthesis). Pennsylvania, Philadelphia, PA, USA.

Eisenberg, R. A., Duboc, G., Weirich, S., & Lee, D. (2021). An existential crisis resolved: type inference for first-class existential types. *Proceedings of the ACM on Programming Languages*, *5*, 64:1–64:29. https://doi.org/10.1145/3473569

Ekici, B., Mebsout, A., Tinelli, C., Keller, C., Katz, G., Reynolds, A., & Barrett, C. (2017). SMTCoq: A Plug-In for Integrating SMT Solvers into Coq. In R. Majumdar & V. Kunčak (Eds.), *Computer Aided Verification* (pp. 126–133). Springer International Publishing.

Eklind, R. (2015). Compositional Decompilation using LLVM IR, 100.

El-Beheiry, L., Reis, G., & Karkour, A. (2021). SMLtoCoq: Automated Generation of Coq Specifications and Proof Obligations from SML Programs with Contracts. *Electronic Proceedings in Theoretical Computer Science*, *337*, 71–87. https://doi.org/10.4204/EPTCS.337.6

El-Korashy, A., Blanco, R., Thibault, J., Durier, A., Garg, D., & Hritcu, C. (2021). SecurePtrs: Proving Secure Compilation with Data-Flow Back-Translation and Turn-Taking Simulation. *arXiv:2110.01439 [cs]*. Retrieved from arXiv:2110.01439

Emre, M., Schroeder, R., Dewey, K., & Hardekopf, B. (2021). Translating C to safer Rust. *Proceedings of the ACM on Programming Languages*, *5*, 121:1–121:29. https://doi.org/10.1145/3485498

Epstein, R. L., & Carnielli, W. A. (1989). *Computability: Computable Functions Logic and the Foundations of Math* (1 edition). Pacific Grove, Calif: Chapman and Hall/CRC.

Erickson, D. (2013). The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 13–18). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2491185.2491189

Ernst, G. (2022). Loop Verification with Invariants and Contracts. In B. Finkbeiner & T. Wies (Eds.), *Verification, Model Checking, and Abstract Interpretation* (Vol. 13182, pp. 69–92). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-94583-1_4

Ernst, G., Blau, J., & Murray, T. (n.d.). Deductive Verification via the Debug Adapter Protocol, 8.

Erosa, A. M., & Hendren, L. J. (1994). Taming control flow: a structured approach to eliminating goto statements. In *Proceedings of 1994 IEEE International Conference on Computer Languages (ICCL'94)* (pp. 229–240). https://doi.org/10.1109/ICCL.1994.288377

Event-B and the Rodin Platform. (n.d.). Retrieved January 10, 2020, from http://www.event-b.org/

EverestTeam. (2021). Project Everest. Retrieved January 21, 2021, from https://project-everest.github.io/

Fahmideh, M., Grundy, J., Ahmed, A., Shen, J., Yan, J., Mougouei, D., … Abedin, B. (2021). Software Engineering for Blockchain Based Software Systems: Foundations, Survey, and Future Directions. *arXiv E-Prints*, *2105*, arXiv:2105.01881. Retrieved from http://adsabs.harvard.edu/abs/2021arXiv210501881F

Farrell, M., Luckcuck, M., & Fisher, M. (2018). Robotics and Integrated Formal Methods: Necessity meets Opportunity. *arXiv:1805.11996 [cs]*, *11023*, 161–171. https://doi.org/10.1007/978-3-319-98938-9_10

Fasse, J. (n.d.). Code Transformations to Increase Prepass Scheduling Opportunities in CompCert, 53.

Fava, D. S. (2020). Finding and Fixing a Mismatch Between the Go Memory Model and Data-Race Detector: A Story on Applied Formal Methods. In F. de Boer & A. Cerone (Eds.), *Software Engineering and Formal Methods* (Vol. 12310, pp. 24–40). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-58768-0_2

Feamster, N., Rexford, J., & Zegura, E. (2013). The Road to SDN: An Intellectual History of Programmable Networks, 13.

Fei, S., Yan, Z., Ding, W., & Xie, H. (2021). Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Computing Surveys*, *54*(6), 126:1–126:36. https://doi.org/10.1145/3456631

Feldman, Y. M. Y., Sagiv, M., Shoham, S., & Wilcox, J. R. (2021). Property-Directed Reachability as Abstract Interpretation in the Monotone Theory. *arXiv:2111.00324 [cs]*. Retrieved from arXiv:2111.00324

Feldman, Y. M. Y., Wilcox, J. R., Shoham, S., & Sagiv, M. (2019). Inferring Inductive Invariants from Phase Structures. *arXiv:1905.07739 [cs]*. Retrieved from arXiv:1905.07739

Ferles, K. (2020). *Practical Formal Methods for Software Analysis and Development* (phdthesis). University of Texas at Austin, Austin, Texas.

Ferles, K., Stephens, J., & Dillig, I. (2021). Verifying correct usage of context-free API protocols. *Proceedings of the ACM on Programming Languages*, *5*, 17:1–17:30. https://doi.org/10.1145/3434298

Ferrando, A. (n.d.). Incrementally Predictive Runtime Verification, 15.

Ferrando, A., Dennis, L. A., Cardoso, R. C., Fisher, M., Ancona, D., & Mascardi, V. (2021). Toward a Holistic Approach to Verification and Validation of Autonomous Cognitive Systems. *ACM Transactions on Software Engineering and Methodology*, *30*(4), 43:1–43:43. https://doi.org/10.1145/3447246

Filinski, A. (1994). Representing Monads. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 446–457). New York, NY, USA: ACM. https://doi.org/10.1145/174675.178047

Filinski, A. (1999). Representing Layered Monads. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 175–188). New York, NY, USA: ACM. https://doi.org/10.1145/292540.292557

Filliâtre, J.-C., Gondelman, L., Lourenço, C., Paskevich, A., & Pereira, M. (n.d.). A Toolchain to Produce Verified OCaml Libraries, 13.

First, E. (2022). Diversity-Driven Automated Formal Verification, 13.

Fisher Kathleen, Launchbury John, & Richards Raymond. (2017). The HACMS program: using formal methods to eliminate exploitable bugs. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150401. https://doi.org/10.1098/rsta.2015.0401

Fleury, M. (2019). Optimizing a Verified SAT Solver. In J. M. Badger & K. Y. Rozier (Eds.), *NASA Formal Methods* (pp. 148–165). Springer International Publishing.

FM folks - richardlford@gmail.com - Gmail. (n.d.). Retrieved October 14, 2019, from https://mail.google.com/mail/u/0/#inbox/FMfcgxwDrlVnZmDccTxHFBnzPRMfbmpn?projector=1&messagePartId=0.1

Fogarty, S., Pasalic, E., Siek, J., & Taha, W. (2007). Concoqtion: Indexed Types Now! In *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation* (pp. 112–121). New York, NY, USA: ACM. https://doi.org/10.1145/1244381.1244400

Ford, R. L., Simon, R. T., Bevier, W. R., & Smith, L. M. (1997). The specification-based testing of a trusted kernel: MK++. In *First IEEE International Conference on Formal Engineering Methods* (pp. 151–160). https://doi.org/10.1109/ICFEM.1997.630422

Formalization of the Interaction Tree Datatype in Coq: DeepSpec/InteractionTrees. (2019). DeepSpec. Retrieved from https://github.com/DeepSpec/InteractionTrees

Formal Versus Agile: Survival of the Fittest.ResearchGate. (n.d.). Retrieved January 14, 2020, from https://www.researchgate.net/publication/224587383_Formal_Versus_Agile_Survival_of_the_Fittest

Fornaia, A., Scafiti, S., & Tramontana, E. (2019). *JSCAN: Designing an Easy to use LLVM-Based Static Analysis Framework*. https://doi.org/10.1109/WETICE.2019.00058

Forster, Y., Jahn, F., & Smolka, G. (n.d.). A Constructive and Synthetic Theory of Reducibility: Myhill's Isomorphism Theorem and Post's Problem for Many-one and Truth-table Reducibility in Coq (Full Version), 26.

Forster, Y., Kunze, F., & Lauermann, N. (n.d.). Synthetic Kolmogorov Complexity in Coq, 20.

Foster, N., Freedman, M. J., Harrison, R., Rexford, J., Meola, M. L., & Walker, D. (2010). Frenetic: a high-level language for OpenFlow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow* (pp. 1–6). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1921151.1921160

Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M. J., Katta, N. P., … Harrison, R. (2013). Languages for software-defined networks. *IEEE Communications Magazine*, *51*(2), 128–134. https://doi.org/10.1109/MCOM.2013.6461197

Foster, N., Kozen, D., Mamouras, K., Reitblatt, M., & Silva, A. (2016). Probabilistic NetKAT. In P. Thiemann (Ed.), *Programming Languages and Systems* (pp. 282–309). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-662-49498-1_12

Foster, N., Kozen, D., Milano, M., Silva, A., & Thompson, L. (2015). A Coalgebraic Decision Procedure for NetKAT. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 343–355). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2676726.2677011

Foster, N., McKeown, N., Rexford, J., Parulkar, G., Peterson, L., & Sunay, O. (2020). Using deep programmability to put network owners in control. *ACM SIGCOMM Computer Communication Review*, *50*(4), 82–88. https://doi.org/10.1145/3431832.3431842

Foster, S., Hur, C.-K., & Woodcock, J. (2021). Formally Verified Simulations of State-Rich Processes using Interaction Trees in Isabelle/HOL. *arXiv:2105.05133 [cs]*. Retrieved from arXiv:2105.05133

Fournet, C., Hawblitzel, C., Parno, B., & Swamy, N. (n.d.). Deploying a Verified Secure Implementation of the HTTPS Ecosystem, 10.

Franceschino, L., Pichardie, D., & Talpin, J.-P. (2021). Verified Functional Programming of an Abstract Interpreter. *arXiv:2107.09472 [cs]*. Retrieved from arXiv:2107.09472

Friedman, G. (2016). An elementary illustrated introduction to simplicial sets. *arXiv:0809.4221 [math]*. Retrieved from arXiv:0809.4221

Fromherz, A., Rastogi, A., Swamy, N., Gibson, S., Martínez, G., Merigoux, D., & Ramananandro, T. (n.d.). Steel: Proof-oriented Programming in a Dependently Typed Concurrent Separation Logic, *1*(1), 27.

Frumin, D., Krebbers, R., & Birkedal, L. (2021). ReLoC Reloaded: A Mechanized Relational Logic for Fine-Grained Concurrency and Logical Atomicity. *Logical Methods in Computer Science*, *Volume 17, Issue 3*, 6598. https://doi.org/10.46298/lmcs-17(3:9)2021

Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., & Platzer, A. (2015). KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In A. P. Felty & A. Middeldorp (Eds.), *Automated Deduction - CADE-25* (Vol. 9195, pp. 527–538). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-21401-6_36

Furia, C. A., Nordio, M., Polikarpova, N., & Tschannen, J. (2017). AutoProof: auto-active functional verification of object-oriented programs. *International Journal on Software Tools for Technology Transfer*, *19*(6), 697–716. https://doi.org/10.1007/s10009-016-0419-0

Fu, W., Krause, F., & Thiemann, P. (2021). Label dependent lambda calculus and gradual typing. *Proceedings of the ACM on Programming Languages*, *5*, 108:1–108:29. https://doi.org/10.1145/3485485

Gabbay, M., Jakobsson, A., & Sojakova, K. (2021). Money grows on (proof-)trees: the formal FA1.2 ledger standard. *arXiv:2109.09451 [cs]*. Retrieved from arXiv:2109.09451

Galois, Inc. Tech Talk: JaVerT: a JavaScript Verification Toolchain (Dr. Philippa Gardner). (n.d.). Retrieved from https://www.youtube.com/watch?v=uNVAmCYL1Jo

Garavel, H. (n.d.). Formal Methos for Safe and Secure Computers Systems. Retrieved September 24, 2021, from https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.pdf?__blob=publicationFile&v=1

GARCÍA, C. P. (n.d.). *Side-Channel Analysis and Cryptography Engineering - Getting OpenSSL Closer to Constant-Time* (phdthesis). Retrieved from https://trepo.tuni.fi/bitstream/handle/10024/137100/978-952-03-2289-2.pdf?sequence=2

Garillot, F., Gonthier, G., Mahboubi, A., & Rideau, L. (2009). Packaging Mathematical Structures. In S. Berghofer, T. Nipkow, C. Urban, & M. Wenzel (Eds.), *Theorem Proving in Higher Order Logics* (pp. 327–342). Springer Berlin Heidelberg.

Gasser, M. (1988). *Building a secure computer system*. New York: Van Nostrand Reinhold Co.

Gauhar, A., Rashid, A., Hasan, O., Bispo, J., & Cardoso, J. M. P. (2021). Formal verification of Matrix based MATLAB models using interactive theorem proving. *PeerJ Computer Science*, *7*, e440. https://doi.org/10.7717/peerj-cs.440

Genkin, D., Nissan, N., Schuster, R., & Tromer, E. (n.d.). Lend Me Your Ear: Passive Remote Physical Side Channels on PCs, 18.

Georges, A. L., Guéneau, A., Strydonck, T. V., Timany, A., Trieu, A., Devriese, D., & Birkedal, L. (n.d.). Cerise: Program Verification on a Capability Machine in the Presence of Untrusted Code, *1*(1), 55.

GHC User's Guide — Glasgow Haskell Compiler 8.6.5 User's Guide. (n.d.). Retrieved May 26, 2019, from https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/index.html

Gheorghiu, A., Docherty, S., & Pym, D. (2021). Provability in BI's Sequent Calculus is Decidable. *arXiv:2103.02343 [cs, Math]*. Retrieved from arXiv:2103.02343

Giallorenzo, S., Montesi, F., Peressotti, M., Richter, D., Salvaneschi, G., & Weisenburger, P. (2021). Multiparty Languages: The Choreographic and Multitier Cases, 28.

Gilbert, G., Cockx, J., Sozeau, M., & Tabareau, N. (2019). Definitional Proof-irrelevance Without K. *Proc. ACM Program. Lang.*, *3*, 3:1–3:28. https://doi.org/10.1145/3290316

Girard, J.-Y. (1995). Linear Logic: its syntax and semantics. In J.-Y. Girard, Y. Lafont, & L. Regnier (Eds.), *Advances in Linear Logic* (pp. 1–42). Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511629150.002

Giuffrida, C. (2014). *Safe and automatic live update* (phdthesis).

Giuffrida, C., Kuijsten, A., Tanenbaum, A. S., Giuffrida, C., Kuijsten, A., Tanenbaum, A. S., … Tanenbaum, A. S. (2013). Safe and automatic live update for operating systems. *ACM SIGARCH Computer Architecture News*, *41*(1), 279–292. https://doi.org/10.1145/2451116.2451147

Gleirscher, M., & Marmsoler, D. (2020). Formal methods in dependable systems engineering: a survey of professionals from Europe and North America. *Empirical Software Engineering*, *25*(6), 4473–4546. https://doi.org/10.1007/s10664-020-09836-5

Goguen, J. A., & Meseguer, J. (1984). Unwinding and Inference Control. In *1984 IEEE Symposium on Security and Privacy* (pp. 75–75). Oakland, CA, USA: IEEE. https://doi.org/10.1109/SP.1984.10019

Gong, S., Altınbüken, D., Fonseca, P., & Maniatis, P. (n.d.). Snowboard: Finding Kernel Concurrency Bugs through Systematic Inter-thread Communication Analysis, 18.

Gonthier, G. (2008). Formal Proof—The Four- Color Theorem, *55*(11), 12.

Gonthier, G., & Mahboubi, A. (2010). An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning*, *3*(2), 95–152. https://doi.org/10.6092/issn.1972-5787/1979

Gonthier, G., Mahboubi, A., & Tassi, E. (2015). *A Small Scale Reflection Extension for the Coq system* (report). Inria Saclay Ile de France. Retrieved from `https://hal.inria.fr/inria-00258384/document`

Gonthier, G., Ziliani, B., Nanevski, A., & Dreyer, D. (2011). How to Make Ad Hoc Proof Automation Less Ad Hoc. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming* (pp. 163–175). New York, NY, USA: ACM. https://doi.org/10.1145/2034773.2034798

Gonthier, G., Ziliani, B., Nanevski, A., & Dreyer, D. (2013). How to make ad hoc proof automation less ad hoc. *Journal of Functional Programming*, *23*(4), 357–401. https://doi.org/10.1017/S0956796813000051

Gorjiara, H., Xu, G. H., & Demsky, B. (2022). Yashme: Detecting Persistency Races, 16.

Gorogiannis, N., O'Hearn, P. W., & Sergey, I. (2019). A true positives theorem for a static race detector. *Proceedings of the ACM on Programming Languages*, *3*, 1–29. https://doi.org/10.1145/3290370

Goudsmid, O., Grumberg, O., & Sheinvald, S. (2021). Compositional Model Checking for Multi-properties. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 55–80). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_4

Grannan, Z., Vazou, N., Darulova, E., & Summers, A. J. (n.d.). REST: Integrating Term Rewriting with Program Verification. Retrieved February 22, 2022, from https://arxiv.org/pdf/2202.05872.pdf

Gratzer, D., & Birkedal, L. (n.d.). A stratified approach to Löb induction, 23.

Gratzer, D., Kavvos, G. A., Nuyts, A., & Birkedal, L. (2021). Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, *Volume 17, Issue 3*, 7571. https://doi.org/10.46298/lmcs-17(3:11)2021

Griesemer, R., Hu, R., Kokke, W., Lange, J., Taylor, I. L., Toninho, B., … Yoshida, N. (2020). Featherweight go. *Proceedings of the ACM on Programming Languages*, *4*, 1–29. https://doi.org/10.1145/3428217

Griffin, M., & Dongol, B. (2021). Verifying Secure Speculation in Isabelle/HOL. In M. Huisman, C. Păsăreanu, & N. Zhan (Eds.), *Formal Methods* (pp. 43–60). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-90870-6_3

Groote, J. F., & Larsen, K. G. (Eds.). (2021). *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II* (Vol. 12652). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-72013-1

Gross, J., Chlipala, A., & Spivak, D. I. (2014). Experience Implementing a Performant Category-Theory Library in Coq. *arXiv:1401.7694 [cs, Math]*. Retrieved from arXiv:1401.7694

Gross, J. S. (2021). Performance Engineering of Proof-Based Software Systems at Scale, 258.

Group, P. or. A. W. (2020, January 6). Inband Network Telemetry (INT) Dataplane Specification.GitHub. Retrieved January 21, 2021, from https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf

Guarnieri, M., & Patrignani, M. (2020). Contract-Aware Secure Compilation. *arXiv:2012.14205 [cs]*. Retrieved from arXiv:2012.14205

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, *38*(3), 105–110. https://doi.org/10.1145/1384609.1384625

Guéneau, A. (n.d.). Procrastination, 7.

Guéneau, A., Jourdan, J.-H., Charguéraud, A., & Pottier, F. (n.d.). Formal Proof and Analysis of an Incremental Cycle Detection Algorithm, 23.

Guéneau, A., Myreen, M. O., Kumar, R., & Norrish, M. (2017). Verified Characteristic Formulae for CakeML. In H. Yang (Ed.), *Programming Languages and Systems* (Vol. 10201, pp. 584–610). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54434-1_22

Guo, Y., Zhou, J., Yao, P., Shi, Q., & Zhang, C. (2022). Precise Divide-By-Zero Detection with Affirmative Evidence, 12.

Gu, R., Koenig, J., Ramananandro, T., Shao, Z., Wu, X. (Newman), Weng, S.-C., … Guo, Y. (2015). Deep Specifications and Certified Abstraction Layers. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 595–608). New York, NY, USA: ACM. https://doi.org/10.1145/2676726.2676975

Gu, R., Shao, Z., Chen, H., Wu, X., Kim, J., Sjöberg, V., & Costanzo, D. (2016). CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (pp. 653–669). Berkeley, CA, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=3026877.3026928

Gu, R., Shao, Z., Kim, J., Wu, X. (Newman), Koenig, J., Sjöberg, V., … Ramananandro, T. (2018). Certified Concurrent Abstraction Layers. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 646–661). New York, NY, USA: ACM. https://doi.org/10.1145/3192366.3192381

Guttag, J. V., Horning, J. J., Garland, S. J., Jones, K. D., Modet, A., & Wing, J. M. (1993). *Larch: Languages and Tools for Formal Specification*. New York, NY: Springer New York. https://doi.org/10.1007/978-1-4612-2704-5

Haase, C., Ishtiaq, S., Ouaknine, J., & Parkinson, M. J. (2013). SeLoger: A Tool for Graph-Based Reasoning in Separation Logic. In N. Sharygina & H. Veith (Eds.), *Computer Aided Verification* (Vol. 8044, pp. 790–795). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39799-8_55

Haase, D., Grädel, E., & Wilke, R. (2021). Separation logic and logics with team semantics. *Annals of Pure and Applied Logic*, 103063. https://doi.org/10.1016/j.apal.2021.103063

Habib, A. (2021). *Learning to Find Bugs in Programs and their Documentation* (phdthesis). Technische Universität, Darmstadt. https://doi.org/10.26083/tuprints-00017377

Hance, T., Heule, M., Martins, R., & Parno, B. (n.d.). Finding Invariants of Distributed Systems: It's a Small (Enough) World After All, 17.

Handbook Of Floating Point Arithmetic Download eBook for Free. (n.d.). Retrieved May 26, 2019, from http://ebook4scaricare.com/gratis/handbook-of-floating-point-arithmetic/

Han, H., Wesie, A., & Pak, B. (2021). Precise and Scalable Detection of Use-after-Compacting-Garbage-Collection Bugs. *Proceeding of 30th USENIX Security Symposium*, 17.

Harper, R., Honsell, F., & Plotkin, G. (1993). A Framework for Defining Logics. *J. ACM*, *40*(1), 143–184. https://doi.org/10.1145/138027.138060

Harrison, J. (2008). Formal Proof—Theory and Practice, *55*(11), 12.

Harrison, J. (2013). The HOL Light Theory of Euclidean Space. *Journal of Automated Reasoning*, *50*(2), 173–190. https://doi.org/10.1007/s10817-012-9250-9

Harrison, L. (1997). Can abstract interpretation become a mainstream compiler technology? In P. Van Hentenryck (Ed.), *Static Analysis* (pp. 395–395). Springer Berlin Heidelberg.

Haslbeck, M. P. L., & Lammich, P. (n.d.-a). For a Few Dollars More: Verified Fine-Grained Algorithm Analysis Down to LLVM. *J. ACM*, *37*(4), 35.

Haslbeck, M. P. L., & Lammich, P. (n.d.-b). Verified Fine-Grained Algorithm Analysis Down to LLVM, 28.

Hatcliff, J., Leavens, G. T., Leino, K. R. M., Müller, P., & Parkinson, M. (2012). Behavioral Interface Specification Languages. *ACM Comput. Surv.*, *44*(3), 16:1–16:58. https://doi.org/10.1145/2187671.2187678

Hathhorn, C., Ellison, C., & Roşu, G. (2015). Defining the Undefinedness of C. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 336–345). New York, NY, USA: ACM. https://doi.org/10.1145/2737924.2737979

Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., … Menth, M. (2021). A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. *arXiv:2101.10632 [cs]*. Retrieved from arXiv:2101.10632

Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J. R., Parno, B., Roberts, M. L., … Zill, B. (2015). IronFleet: Proving Practical Distributed Systems Correct. In *Proceedings of the 25th Symposium on Operating Systems Principles* (pp. 1–17). New York, NY, USA: ACM. https://doi.org/10.1145/2815400.2815428

Hawblitzel, C., Howell, J., Lorch, J. R., Narayan, A., Parno, B., Zhang, D., & Zill, B. (n.d.). Ironclad Apps: End-to-End Security via Automated Full-System Verification, 18.

Hawblitzel, C., Petrank, E., Qadeer, S., & Tasiran, S. (2015). Automated and Modular Refinement Reasoning for Concurrent Programs. In *Computer Aided Verification* (pp. 449–465). Springer, Cham. https://doi.org/10.1007/978-3-319-21668-3_26

Hazel, a live functional programming environment featuring typed holes. (n.d.). Retrieved August 30, 2019, from http://hazel.org/

Hazel, a live functional programming environment with typed holes: hazelgrove/hazel. (2019). hazelgrove. Retrieved from https://github.com/hazelgrove/hazel

Hedin, D., & Sabelfeld, A. (n.d.). A Perspective on Information-Flow Control, 29.

Hellerstein, J. M., & Alvaro, P. (2020). Keeping CALM: when distributed consistency is easy. *Communications of the ACM*, *63*(9), 72–81. https://doi.org/10.1145/3369736

He, P., Westbrook, E., Carmer, B., Phifer, C., Robert, V., Smeltzer, K., … Zdancewic, S. (2021). A type system for extracting functional specifications from memory-safe imperative programs. *Proceedings of the ACM on Programming Languages*, *5*, 1–29. https://doi.org/10.1145/3485512

Herklotz, Y. (2021). *Vericert*. Retrieved from https://github.com/ymherklotz/vericert

Herklotz, Y., Pollard, J., Ramanathan, N., & Wickerson, J. (2017). Formal Verification of High-Level Synthesis, 14.

Herlihy, M. P., & Wing, J. M. (1990). Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.*, *12*(3), 463–492. https://doi.org/10.1145/78969.78972

Hickman, T., Laursen, C. P., & Foster, S. (2021). Certifying Differential Equation Solutions from Computer Algebra Systems in Isabelle/HOL. *arXiv:2102.02679 [cs, Math]*. Retrieved from arXiv:2102.02679

Hiet, G. (2021). *Security at the Hardware/Software Interface* (Habilitation à diriger des recherches). Université de Rennes 1. Retrieved from https://hal.archives-ouvertes.fr/tel-03511334

Hinton, G. (2020). The Next Generation of Neural Networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (p. 1). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3397271.3402425

Hinton, G. E., Krizhevsky, A., & Wang, S. D. (2011). Transforming Auto-Encoders. In T. Honkela, W. Duch, M. Girolami, & S. Kaski (Eds.), *Artificial Neural Networks and Machine Learning – ICANN 2011* (pp. 44–51). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-21735-7_6

Hoang, D., Moy, Y., Wallenburg, A., & Chapman, R. (2015). SPARK 2014 and GNATprove. *International Journal on Software Tools for Technology Transfer*, *17*(6), 695–707. https://doi.org/10.1007/s10009-014-0322-5

Hoare, T. (2003). The Verifying Compiler: A Grand Challenge for Computing Research. *J. ACM*, *50*(1), 63–69. https://doi.org/10.1145/602382.602403

Hobor, A., Dockins, R., & Appel, A. W. (2010). A Theory of Indirection via Approximation. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 171–184). New York, NY, USA: ACM. https://doi.org/10.1145/1706299.1706322

Hoder, K., Bjørner, N., & Moura, L. de. (2011). Z– An Efficient Engine for Fixed Points with Constraints. In G. Gopalakrishnan & S. Qadeer (Eds.), *Computer Aided Verification* (pp. 457–462). Springer Berlin Heidelberg.

Hofmann, M., Leutgeb, L., Moser, G., Obwaller, D., & Zuleger, F. (2021). Type-Based Analysis of Logarithmic Amortised Complexity. *arXiv:2101.12029 [cs]*. Retrieved from arXiv:2101.12029

Hogan, M., Landau-Feibish, S., Tahmasbi Arashloo, M., Rexford, J., Walker, D., & Harrison, R. (2020). Elastic Switch Programming with P4All. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (pp. 168–174). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3422604.3425933

Hölzl, J., Immler, F., & Huffman, B. (2013). Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.), *Interactive Theorem Proving* (pp. 279–294). Springer Berlin Heidelberg.

Honoré, W., Kim, J., Shin, J.-Y., & Shao, Z. (n.d.). Much ADO about Failures: A Fault-Aware Model for Compositional Verification of Strongly Consistent Distributed Systems, *5*, 42.

Hood, J. (2016). Trusted, Third-Party Authenticated, Quantum Key Distribution. Retrieved from https://etd.auburn.edu//handle/10415/5373

Hriţcu, C. (2015). Micro-Policies: Formally Verified, Tag-Based Security Monitors. In *Proceedings of the 10th ACM Workshop on Programming Languages and Analysis for Security - PLAS'15* (pp. 1–1). Prague, Czech Republic: ACM Press. https://doi.org/10.1145/2786558.2786560

Hriţcu, C. (n.d.). The Quest for Formally Secure Compartmentalizing Compilation, 96.

Hsieh, C., & Mitra, S. (2019a). Dione: A Protocol Verification System Built with Dafny for I/O Automata. In W. Ahrendt & S. L. Tapia Tarifa (Eds.), *Integrated Formal Methods* (Vol. 11918, pp. 227–245). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-34968-4_13

Hsieh, C., & Mitra, S. (2019b). Dione: A Protocol Verification System Built with Dafny for I/O Automata. In W. Ahrendt & S. L. Tapia Tarifa (Eds.), *Integrated Formal Methods* (pp. 227–245). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-34968-4_13

Huang, K., Huang, Y., Payer, M., Qian, Z., Sampson, J., Tan, G., & Jaeger, T. (n.d.). The Taming of the Stack: Isolating Stack Data from Memory Errors. *Network and Distributed Systems Security (NDSS) Symposium 2022*, 17.

Hublet, F. (2021). The Databank Model, 212 p. https://doi.org/10.3929/ETHZ-B-000477329

Hughes, J. (1989). Why Functional Programming Matters. *The Computer Journal*, *32*(2), 98–107. https://doi.org/10.1093/comjnl/32.2.98

Huihui, C., & Hongwei, Z. (n.d.). Optimizing demand-driven null dereference verification via merging branches. *Expert Systems*, *n/a*, e12707. https://doi.org/10.1111/exsy.12707

Hu, J. Z. S., & Carette, J. (2021). Formalizing category theory in Agda. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 327–342). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3437992.3439922

Hunt Warren A., Kaufmann Matt, Moore J Strother, & Slobodova Anna. (2017). Industrial hardware and software verification with ACL2. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150399. https://doi.org/10.1098/rsta.2015.0399

Hur, C.-K., Neis, G., Dreyer, D., & Vafeiadis, V. (2013). The power of parameterization in coinductive proof. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 193–206). Rome, Italy: Association for Computing Machinery. https://doi.org/10.1145/2429069.2429093

Hurd, J. (2011). The OpenTheory Standard Theory Library. In M. Bobaru, K. Havelund, G. J. Holzmann, & R. Joshi (Eds.), *NASA Formal Methods* (Vol. 6617, pp. 177–191). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-20398-5_14

Huyghebaert, S., Keuchel, S., & Devriese, D. (2021). Semi-automatic verification of ISA security guarantees in the form of universal contracts. *SILM Workshop 2021*, 6. Retrieved from https://silm-workshop-2021.inria.fr/wp-content/uploads/2021/09/ISAVerif.pdf

IEEE. (n.d.). *IEEE Standard for Universal Verification Methodology Language Reference Manual*. https://doi.org/10.1109/IEEESTD.2017.7932212

IEEE Computer Society. (2014). *Guide to the software engineering body of knowledge*. (P. Bourque & R. E. Fairley, Eds.).

IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic. (n.d.), 70.

IEEE Std 1800™-2012 (Revision of IEEE Std 1800-2009) IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language. (n.d.), 1315.

IEEE Xplore Full-Text PDF: (n.d.). Retrieved July 16, 2021, from https://ieeexplore-ieee-org.rsic.army.mil:3443/stamp/stamp.jsp?tp=&arnumber=9470541&tag=1

Immler, F. (2018). A Verified ODE Solver and the Lorenz Attractor. *Journal of Automated Reasoning*, *61*(1), 73–111. https://doi.org/10.1007/s10817-017-9448-y

Infer#: Interprocedural Memory Safety Analysis For C#..NET Blog. (2020, December 8). Retrieved January 20, 2021, from https://devblogs.microsoft.com/dotnet/infer-interprocedural-memory-safety-analysis-for-c/

infersharp. (2021). Microsoft. Retrieved from https://github.com/microsoft/infersharp

Introduction to Domain Theory. (n.d.). Retrieved January 13, 2020, from http://www.cs.nott.ac.uk/~pszgmh/domains.html

Ioannidis, E., Kaashoek, F., & Zeldovich, N. (2019). Extracting and Optimizing Formally Verified Code for Systems Programming. In J. M. Badger & K. Y. Rozier (Eds.), *NASA Formal Methods* (Vol. 11460, pp. 228–236). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-20652-9_15

Iosif, R., Serban, C., Reynolds, A., & Sighireanu, M. (n.d.). Encoding Separation Logic in SMT-LIB v2.5, 8.

Iris / stdpp.GitLab. (n.d.). Retrieved January 10, 2020, from https://gitlab.mpi-sws.org/iris/stdpp

Iron: Managing Obligations in Higher-Order Concurrent Separation Logic (POPL 2019). (n.d.). Retrieved January 14, 2020, from https://iris-project.org/iron/

Ishimwe, D., Nguyen, K., & Nguyen, T. (n.d.). Dynaplex: Analyzing Program Complexity using Dynamically Inferred Recurrence Relations, *5*, 23.

Ishtiaq, S., & O'Hearn, P. W. (2011). BI As an Assertion Language for Mutable Data Structures. *SIGPLAN Not.*, *46*(4), 84–96. https://doi.org/10.1145/1988042.1988050

Islam, R., Tian, R., Batten, L., & Versteeg, S. (2010). Classification of Malware Based on String and Function Feature Selection. In *2010 Second Cybercrime and Trustworthy Computing Workshop* (pp. 9–17). https://doi.org/10.1109/CTC.2010.11

Jacobs, B. (2017). *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9781316823187

Jacobs, B. (2019). *verifast/verifast: Research prototype tool for modular formal verification of C and Java programs*. verifast. Retrieved from https://github.com/verifast/verifast

Jacobs, B., & Piessens, F. (2008). *The VeriFast program verifier*.

Jacobs, B., Smans, J., & Piessens, F. (2017). The VeriFast Program Verifier: A Tutorial, 102.

Jacobs, B., Vogels, F., & Piessens, F. (2015). Featherweight VeriFast. *Logical Methods in Computer Science*, *11*(3). https://doi.org/10.2168/LMCS-11(3:19)2015

Jacobs, J., Balzer, S., & Krebbers, R. (n.d.). Connectivity Graphs: A Method for Proving Deadlock Freedom Based on Separation Logic, *6*, 33.

Jain, R., Purandare, R., & Sharma, S. (2021). BiRD: Race Detection in Software Binaries under Relaxed Memory Models. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.1145/3498538

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., … Vahdat, A. (n.d.). B4: Experience with a Globally-Deployed Software Defined WAN, 12.

Jaiswal, S., Khedker, U. P., & Mycroft, A. (2021). A Unified Model for Context-Sensitive Program Analyses: The Blind Men and the Elephant. *ACM Computing Surveys*, *54*(6), 114:1–114:37. https://doi.org/10.1145/3456563

Jang, J., Gélineau, S., Monnier, S., & Pientka, B. (2021). Moebius: Metaprogramming using Contextual Types – The stage where System F can pattern match on itself (Long Version). *arXiv:2111.08099 [cs]*. Retrieved from arXiv:2111.08099

Janson, C., & Struck, P. (2022). *Sponge-based Authenticated Encryption: Security against Quantum Attackers* (No. 139). Retrieved from http://eprint.iacr.org/2022/139

Jeannet, B., & Miné, A. (n.d.). APRON numerical abstract domain library. Retrieved February 1, 2019, from http://apron.cri.ensmp.fr/library/

Jeong, D. R., Kim, K., Shivakumar, B., Lee, B., & Shin, I. (2019). Razzer: Finding Kernel Race Bugs through Fuzzing. In *2019 IEEE Symposium on Security and Privacy (SP)* (pp. 754–768). San Francisco, CA, USA: IEEE. https://doi.org/10.1109/SP.2019.00017

Joe Leslie-Hurd. (2015, July 19). The Slowest Software Development Methodology in the World.The Robot Mathematician. Retrieved January 10, 2020, from https://gilith.wordpress.com/2015/07/19/the-slowest-software-development-methodology-in-the-world/

Jones, C. (2013). Function Points As a Universal Software Metric. *SIGSOFT Softw. Eng. Notes*, *38*(4), 1–27. https://doi.org/10.1145/2492248.2492268

Jordan, E. (n.d.-a). The Ultimate Guide to Open RAN: Open RAN Intelligent Controller (RIC) - Part 1. Retrieved January 14, 2021, from https://www.thefastmode.com/expert-opinion/18213-the-ultimate-guide-to-open-ran-open-ran-intelligent-controller-ric-part-1

Jordan, E. (n.d.-b). The Ultimate Guide to Open RAN: Open RAN Intelligent Controller (RIC) - Part 2: Implementations. Retrieved January 14, 2021, from https://www.thefastmode.com/expert-opinion/18274-the-ultimate-guide-to-open-ran-open-ran-intelligent-controller-ric-part-2-implementations

Jourdan, J.-H. (2016). *Verasco: a Formally Verified C Static Analyzer* (phdthesis). L'université Paris Diderot (Paris 7) Sorbonne Paris Cité, Paris, France. Retrieved from https://jhjourdan.mketjh.fr/thesis_jhjourdan.pdf

Juglaret, Y., Hritcu, C., Amorim, A. A. de, Eng, B., & Pierce, B. C. (2017). Beyond Good and Evil: Formalizing the Security Guarantees of Compartmentalizing Compilation. *arXiv:1602.04503 [cs]*. Retrieved from arXiv:1602.04503

Juglaret, Y., Hritcu, C., Amorim, A. A. de, Pierce, B. C., Spector-Zabusky, A., & Tolmach, A. (2015). Towards a Fully Abstract Compiler Using Micro-Policies: Secure Compilation for Mutually Distrustful Components. *arXiv:1510.00697 [cs]*. Retrieved from arXiv:1510.00697

Jung, R. (n.d.). Iris Project. Retrieved February 1, 2019, from https://iris-project.org/

Jung, R., Jourdan, J.-H., Krebbers, R., & Dreyer, D. (2017). RustBelt: securing the foundations of the rust programming language. *Proceedings of the ACM on Programming Languages*, *2*, 1–34. https://doi.org/10.1145/3158154

Jung, R., Jourdan, J.-H., Krebbers, R., & Dreyer, D. (2021). Safe systems programming in Rust. *Communications of the ACM*, *64*(4), 144–152. https://doi.org/10.1145/3418295

Jung, R., Krebbers, R., Jourdan, J.-H., Bizjak, A., Birkedal, L., & Dreyer, D. (2018a). Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, *28*. https://doi.org/10.1017/S0956796818000151

Jung, R., Krebbers, R., Jourdan, J.-H., Bizjak, A., Birkedal, L., & Dreyer, D. (2018b). Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, *28*. https://doi.org/10.1017/S0956796818000151

Kachapova, F. (2021). Formalizing relations in type theory. *arXiv:2102.08595 [cs, Math]*. Retrieved from arXiv:2102.08595

Kaiser, J.-O., & Ziliani, B. (n.d.). A "destruct" Tactic for Mtac2 - POPL 2018. Retrieved February 1, 2019, from https://popl18.sigplan.org/event/coqpl-2018-a-destruct-tactic-for-mtac2

Kamburjan, E., & Wasser, N. (2021). Deductive Verification of Programs with Underspecified Semantics by Model Extraction. *arXiv:2110.01964 [cs]*. Retrieved from arXiv:2110.01964

Kanabar, H., Fox, A. C. J., & Myreen, M. O. (n.d.). Taming an Authoritative Armv8 ISA Specification: L3 Validation and CakeML Compiler Verification, 21.

Kang, J., Kim, Y., Song, Y., Lee, J., Park, S., Shin, M. D., … Yi, K. (2018). Crellvm: Verified Credible Compilation for LLVM. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 631–645). New York, NY, USA: ACM. https://doi.org/10.1145/3192366.3192377

Kang, M., Kang, E.-Y., Hwang, D.-Y., Kim, B.-J., Nam, K.-H., Shin, M.-K., & Choi, J.-Y. (2013). *Formal Modeling and Verification of SDN-OpenFlow*. https://doi.org/10.1109/ICST.2013.69

Kästner, D., & Pohland, J. (2015). Program Analysis on Evolving Software. In M. Roy (Ed.), *CARS 2015 - Critical Automotive applications: Robustness & Safety*. Paris, France. Retrieved from https://hal.archives-ouvertes.fr/hal-01192985

Kästner, D., Wilhelm, S., Nenova, S., Miné, A., Rival, X., Mauborgne, L., … Cousot, R. (n.d.). Astree: Proving the Absence of Runtime Errors, 9. Retrieved from https://www.di.ens.fr/~rival/papers/erts10.pdf

Kaur, S., Singh, J., & Ghumman, N. (2014). *Network Programmability Using POX Controller*. https://doi.org/10.13140/RG.2.1.1950.6961

Kell, S., Mulligan, D. P., & Sewell, P. (2016). The Missing Link: Explaining ELF Static Linking, Semantically. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 607–623). New York, NY, USA: ACM. https://doi.org/10.1145/2983990.2983996

Kennedy, A. (2010). Types for Units-of-Measure: Theory and Practice. In Z. Horváth, R. Plasmeijer, & V. Zsók (Eds.), *Central European Functional Programming School: Third Summer School, CEFP 2009, Budapest, Hungary, May 21-23, 2009 and Komárno, Slovakia, May 25-30, 2009, Revised Selected Lectures* (pp. 268–305). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-17685-2_8

Khan, W., Hou, Z., Sanan, D., Nebhen, J., Liu, Y., & Tiu, A. (2022). An Executable Formal Model of the VHDL in Isabelle/HOL. *arXiv:2202.04192 [cs]*. Retrieved from arXiv:2202.04192

Khayam, A., Noizet, L., & Schmitt, A. (2021). JSkel: Towards a Formalization of JavaScript's Semantics, 22.

Kiaei, P., Breunesse, C.-B., Ahmadi, M., Schaumont, P., & Woudenberg, J. van. (2020). Rewrite to Reinforce: Rewriting the Binary to Apply Countermeasures against Fault Injection. *arXiv:2011.14067 [cs]*. Retrieved from arXiv:2011.14067

Kim, H., Chen, X., Brassil, J., & Rexford, J. (2021). Experience-Driven Research on Programmable Networks. *SIGCOMM Computer Communications Review*, 7.

Kim, H., Reich, J., Gupta, A., Shahbaz, M., Feamster, N., & Clark, R. (n.d.). Kinetic: Verifiable Dynamic Network Control, 15.

Kim, S. K., Venet, A. J., & Thakur, A. V. (2020a). Deterministic Parallel Fixpoint Computation. *Proceedings of the ACM on Programming Languages*, *4*, 1–33. https://doi.org/10.1145/3371082

Kim, S. K., Venet, A. J., & Thakur, A. V. (2020b). Memory-Efficient Fixpoint Computation. *arXiv:2009.05865 [cs]*. Retrieved from arXiv:2009.05865

Kimura, D., & Tatsuta, M. (2021). Decidability for Entailments of Symbolic Heaps with Arrays. *Logical Methods in Computer Science*, *17*(2), 33. https://doi.org/DOI:10.23638/LMCS-17(2:15)2021

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882 [cs]*. Retrieved from arXiv:1408.5882

Kirschenbaum, J., Adcock, B., Bronish, D., Smith, H., Harton, H., Sitaraman, M., & Weide, B. W. (2009). Verifying Component-Based Software: Deep Mathematics or Simple Bookkeeping? In S. H. Edwards & G. Kulczycki (Eds.), *Formal Foundations of Reuse and Domain Engineering* (pp. 31–40). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-04211-9_4

Kj, J., Madsen, F. P., & Battle, N. (n.d.). The Specification Language Server Protocol: A Proposal for Standardised LSP Extensions. *F-IDE2021*, 16.

Klein, G., Andronick, J., Fernandez, M., Kuz, I., Murray, T., & Heiser, G. (2018). Formally verified software in the real world. *Communications of the ACM*, *61*(10), 68–77. https://doi.org/10.1145/3230627

Klein Gerwin, Andronick June, Keller Gabriele, Matichuk Daniel, Murray Toby, & O'Connor Liam. (2017). Provably trustworthy systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150404. https://doi.org/10.1098/rsta.2015.0404

Knaggs, P. J. (1993). Practical and theoretical aspects of FORTH software development.

Knüppel, A., Schaer, L., & Schaefer, I. (2021). How much Specification is Enough? Mutation Analysis for Software Contracts. In *2021 IEEE/ACM 9th International Conference on Formal Methods in Software Engineering (FormaliSE)* (pp. 42–53). https://doi.org/10.1109/FormaliSE52586.2021.00011

Koch, M. (n.d.). *Mechanizing Second-Order Logic in Coq* (Bachelor's Thesis). Saarland University, 8/23/2021.

Koenig, J., & Leino, R. (2016). Programming Language Features for Refinement. Retrieved from https://www.microsoft.com/en-us/research/publication/programming-language-features-refinement/

Koenig, J., & Shao, Z. (2021). CompCertO: Compiling Certified Open C Components, 15.

Koh, N., Li, Y., Li, Y., Xia, L., Beringer, L., Honoré, W., … Zdancewic, S. (2019). From C to Interaction Trees: Specifying, Verifying, and Testing a Networked Server. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 234–248). New York, NY, USA: ACM. https://doi.org/10.1145/3293880.3294106

Komel, A. P. (2021). META-ANALYSIS OF TYPE THEORIES WITH AN APPLICATION TO THE DESIGN OF FORMAL PROOFS, 194.

Konečný, M., Park, S., & Thies, H. (2022). Extracting efficient exact real number computation from proofs in constructive type theory. *arXiv:2202.00891 [cs]*. Retrieved from arXiv:2202.00891

Korczynski, D. (2019, September 17). Comparison of the LLVM IR generated by three binary-to-llvm translators. Retrieved February 17, 2021, from https://adalogics.com/blog/binary-to-llvm-comparison

Korencik, L., & University, M. (2019). *Decompiling Binaries into LLVM IR Using McSema and Dyninst* (Diploma). Masaryk University. Retrieved from https://is.muni.cz/th/pxe1j/thesis.pdf

Krebbers, R. (n.d.). Diaframe: Automated Verification of Fine-Grained Concurrent Programs in Iris. Retrieved December 7, 2021, from https://robbertkrebbers.nl/research/articles/diaframe.pdf

Krebbers, R., Jourdan, J.-H., Jung, R., Tassarotti, J., Kaiser, J.-O., Timany, A., … Dreyer, D. (2018). MoSeL: a general, extensible modal framework for interactive proofs in separation logic. *Proceedings of the ACM on Programming Languages*, *2*, 1–30. https://doi.org/10.1145/3236772

Krebbers, R., & Spitters, B. (2011a). Computer Certified Efficient Exact Reals in Coq. In J. H. Davenport, W. M. Farmer, J. Urban, & F. Rabe (Eds.), *Intelligent Computer Mathematics* (Vol. 6824, pp. 90–106). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-22673-1_7

Krebbers, R., & Spitters, B. (2011b). Type classes for efficient exact real arithmetic in Coq. *arXiv:1106.3448 [cs, Math]*. https://doi.org/10.2168/LMCS-9(1:01)2013

Krishnan, R., & Lalithambika, V. R. (2018). Modelling and validating 1553B protocol using the SPIN model checker. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)* (pp. 472–475). Bengaluru: IEEE. https://doi.org/10.1109/COMSNETS.2018.8328247

Kruger, S., Nadi, S., Reif, M., Ali, K., Mezini, M., Bodden, E., … Kamath, R. (2017). CogniCrypt: Supporting developers in using cryptography. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 931–936). Urbana, IL: IEEE. https://doi.org/10.1109/ASE.2017.8115707

Krüger, S., Späth, J., Ali, K., Bodden, E., & Mezini, M. (2019). CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2019.2948910

K. Rustan M. Leino. (n.d.-a). Dafny Source Code.

Kubota, K. (2016). Foundations of Mathematics. https://doi.org/10.4444/100.111

Kubota, K. (n.d.). Foundations of Mathematics – Owl of Minerva Press. Retrieved February 1, 2019, from http://owlofminerva.net/foundations-of-mathematics/

Kulik, T., Dongol, B., Larsen, P. G., Macedo, H. D., Schneider, S., Tran-Jørgensen, P. W. V., & Woodcock, J. (2021). A Survey of Practical Formal Methods for Security. *arXiv:2109.01362 [cs]*. Retrieved from arXiv:2109.01362

Kumar, S., Agrawal, A., & Biswas, S. (2021). Efficient Data Race Detection of Async-Finish Programs Using Vector Clocks. *arXiv:2112.04352 [cs]*. Retrieved from arXiv:2112.04352

Labrosse, J. (1992a). A Portable Real-Time Kernel in C, 5/92. *Embedded Systems Programming*, 12.

Labrosse, J. (1992b). Implementing a Real-Time Kernel, 6/92, 6.

Lahav, O., & Boker, U. (n.d.). What's Decidable about Causally Consistent Shared Memory?, *0*(0), 54.

Lahiri, S. K., Hawblitzel, C., Kawaguchi, M., & Rebêlo, H. (2012). SYMDIFF: A Language-Agnostic Semantic Diff Tool for Imperative Programs. In P. Madhusudan & S. A. Seshia (Eds.), *Computer Aided Verification* (pp. 712–717). Springer Berlin Heidelberg.

Lahiri, S. K., Sinha, R., & Hawblitzel, C. (2015). Automatic Rootcausing for Program Equivalence Failures in Binaries. In D. Kroening & C. S. Păsăreanu (Eds.), *Computer Aided Verification* (pp. 362–379). Springer International Publishing.

Lambdapi, a proof assistant based on the Π-calculus modulo rewriting. (2020). Deducteam. Retrieved from https://github.com/Deducteam/lambdapi

Lamport, L. (n.d.). Specifying Systems. Retrieved February 1, 2019, from https://lamport.azurewebsites.net/tla/book.html

Lamport, L., & Distributed Computing \& Education Column by Juraj Hromkovic, S. S. (2018). If You're Not Writing a Program, Don't Use a Programming Language. *Bulletin of EATCS*, *2*(125). Retrieved from http://eatcs.org/beatcs/index.php/beatcs/article/view

Lamport, L., & Schneider, F. B. (2005, December 27). Pretending Atomicity, Digital Systems Research Center: Report 44. Retrieved December 30, 2019, from https://web.archive.org/web/20051227134748/http://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/abstracts/src-rr-044.html

Lampson, B. (2001). The ABCD's of Paxos. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing* (p. 13 –). New York, NY, USA: ACM. https://doi.org/10.1145/383962.383969

Lanzinger, F., Weigl, A., Ulbrich, M., & Dietl, W. (2021). Scalability and precision by combining expressive type systems and deductive verification. *Proceedings of the ACM on Programming Languages*, *5*, 143:1–143:29. https://doi.org/10.1145/3485520

Lasser, S., Casinghino, C., Fisher, K., & Roux, C. (2021). CoStar: a verified ALL(*) parser. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 420–434). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3453483.3454053

Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., … Zinenko, O. (2020). MLIR: A Compiler Infrastructure for the End of Moore's Law. *arXiv:2002.11054 [cs]*. Retrieved from arXiv:2002.11054

Lean Forward: Usable Computer-Checked Proofs and Computations. (n.d.). Retrieved February 8, 2019, from https://lean-forward.github.io/

Lee, Y. (2021). *instruction2vec*. Retrieved from https://github.com/firmcode/instruction2vec

Lee, Y., Kwon, H., Choi, S.-H., Lim, S.-H., Baek, S. H., & Park, K.-W. (2019). Instruction2vec: Efficient Preprocessor of Assembly Code to Detect Software Weakness with CNN. *Applied Sciences*, *9*(19), 4086. https://doi.org/10.3390/app9194086

Leger, M., Darling, M., Jones, S., Matzen, L., Stracuzzi, D., Wilson, A., … Williams, J. (2021). *Exploring Explicit Uncertainty for Binary Analysis (EUBA).* (No. {SAND}2021-14600, 1832314, 701941) (pp. SAND2021–SAND14600, 1832314, 701941). https://doi.org/10.2172/1832314

Lehmann, N., Kunkel, R., Brown, J., Yang, J., Vazou, N., Polikarpova, N., … Jhala, R. (2021). STORM: Refinement Types for Secure Web Applications. *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, 19.

Leijen, D., Zorn, B., & Moura, L. de. (2019). Mimalloc: Free List Sharding in Action. In A. W. Lin (Ed.), *Programming Languages and Systems* (pp. 244–265). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-34175-6_13

Leino, K. R. M. (2009). Dynamic-frame specifications in Dafny. JML seminar, Dagstuhl, Germany.

Leino, K. R. M. (n.d.-a). Dafny Quick Reference.

Leino, K. R. M. (n.d.-b). Main Microsoft Research Dafny Web page.

Leino, K. R. M. (n.d.-c). Try Dafny In Your Browser.

Leino, K. R. M., & Lucio, P. (2015). An Assertional Proof of the Stability and Correctness of Natural Mergesort. *ACM Trans. Comput. Logic*, *17*(1), 6:1–6:22. https://doi.org/10.1145/2814571

Leino, K. R. M., & Moskal, M. (2014). Co-induction Simply: Automatic Co-inductive Proofs in a Program Verifier. Manuscript KRML 230.

Leino, K. R. M., & Pit-Claudel, C. (2016). Trigger Selection Strategies to Stabilize Program Verifiers. In S. Chaudhuri & A. Farzan (Eds.), *Computer Aided Verification* (Vol. 9779, pp. 361–381). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-41528-4_20

Leino, K. R. M., & Polikarpova, N. (2013). Verified Calculations. Manuscript KRML 231.

Leino, R. (2016a). Compiling Hilbert's epsilon Operator. *LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning*, *35*. Retrieved from https://www.microsoft.com/en-us/research/publication/compiling-hilberts-%cf%b5-operator/

Leino, R. (2016b). Well-Founded Functions and Extreme Predicates in Dafny: A Tutorial, *40*. Retrieved from https://www.microsoft.com/en-us/research/publication/well-founded-functions-extreme-predicates-dafny-tutorial/

Leino, R. (2021). *Dafny*. Dafny. Retrieved from https://github.com/dafny-lang/dafny

Leino, R., & Moskal, M. (2013). Co-Induction Simply: Automatic Co-Inductive Proofs in a Program Verifier. Retrieved from https://www.microsoft.com/en-us/research/publication/co-induction-simply-automatic-co-inductive-proofs-in-a-program-verifier/

Leino, R., Müller, P., & Smans, J. (2016). Verification of Concurrent Programs with Chalice. Retrieved from https://www.microsoft.com/en-us/research/publication/verification-concurrent-programs-chalice/

Leino, R., & Polikarpova, N. (2016). Verified Calculations. Retrieved from https://www.microsoft.com/en-us/research/publication/verified-calculations/

Leino, R., & Wüstholz, V. (2016). Fine-grained Caching of Verification Results, *9206*. Retrieved from https://www.microsoft.com/en-us/research/publication/fine-grained-caching-verification-results/

Leino, R., & Yessenov, K. (2016). Stepwise Refinement of Heap-Manipulating Code in Chalice. Retrieved from https://www.microsoft.com/en-us/research/publication/stepwise-refinement-heap-manipulating-code-chalice/

Leinster, T. (2003). Higher Operads, Higher Categories. *arXiv:math/0305049*. Retrieved from arXiv:math/0305049

Lem semantic definition language. (2019). REMS. Retrieved from https://github.com/rems-project/lem

Lepigre, R., Sammler, M., Memarian, K., Krebbers, R., Dreyer, D., & Sewell, P. (n.d.). VIP: Verifying Real-World C Idioms with Integer-Pointer Casts, *6*, 32.

Le, Q. L. (2021). Compositional Satisfiability Solving in Separation Logic. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 578–602). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_26

Le, Q. L., Raad, A., Villard, J., Berdine, J., Dreyer, D., & O'Hearn, P. W. (n.d.-a). Finding Real Bugs in Big Programs Appendix, *1*(1), 6.

Le, Q. L., Raad, A., Villard, J., Berdine, J., Dreyer, D., & O'Hearn, P. W. (n.d.-b). Finding Real Bugs in Big Programs with Incorrectness Logic, *1*(1), 31.

Leroy, X. (n.d.). OCaml Home Page. Retrieved February 1, 2019, from https://ocaml.org/

Lescuyer, S. (2015). *ProvenCore: Towards a Verified Isolation Micro-Kernel*. Retrieved from https://zenodo.org/record/47990#.XOrmF-tKi24

Lescuyer, S. (n.d.). ProvenCore: Towards a Verified Isolation Micro-Kernel, 69.

Le Temps des Cerises: Efficient Temporal Stack Safety on Capability Machines using Directed Capabilities. (n.d.), *1*, 30.

Letouzey, P. (n.d.). Certified functional programming : Program extraction within Coq proof assistant.ResearchGate. Retrieved February 1, 2019, from https://www.researchgate.net/publication/280790704_Certified_functional_programming_Program_extraction_within_Coq_proof_assistant

Liew, D., Schemmel, D., Cadar, C., Donaldson, A. F., Zahl, R., & Wehrle, K. (2017). Floating-point symbolic execution: A case study in N-version programming. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 601–612). Urbana, IL: IEEE. https://doi.org/10.1109/ASE.2017.8115670

Li, J. M., & Appel, A. W. (n.d.). Deriving Efficient Program Transformations from Rewrite Rules, *5*, 29.

Li, J., Miller, S., Zhuo, D., Chen, A., Howell, J., & Anderson, T. (2021). An Incremental Path Towards a Safer OS Kernel, 8.

Li, L. (2020). *A VERIFICATION FRAMEWORK SUITABLE FOR PROVING LARGE LANGUAGE TRANSLATIONS* (phdthesis). University of Illinois at Urbana-Champaign.

Li, L., Allix, K., Li, D., Bartel, A., Bissyande, T. F., & Klein, J. (2015). Potential Component Leaks in Android Apps: An Investigation into a New Feature Set for Malware Detection. In *2015 IEEE International Conference on Software Quality, Reliability and Security* (pp. 195–200). Vancouver, BC, Canada: IEEE. https://doi.org/10.1109/QRS.2015.36

Li, L., Liu, Y., Postol, D. L., Lampropoulos, L., Van Horn, D., & Hicks, M. (2022). A Formal Model of Checked C. *arXiv:2201.13394 [cs]*. Retrieved from arXiv:2201.13394

Lin, Y.-Y., & Tzevelekos, N. (2020). Symbolic Execution Game Semantics. *arXiv:2002.09115 [cs]*. Retrieved from arXiv:2002.09115

Lin, Z., Chen, X., & Roşu, G. (2021). An Interactive Theorem Prover for Matching Logic with Proof Object Generation, 12.

Lin, Z., Chen, X., Trinh, M.-T., Wang, J., & Roşu, G. (n.d.-a). Making Formal Verification Trustworthy via Proof Generation, 15.

Lin, Z., Chen, X., Trinh, M.-T., Wang, J., & Roşu, G. (n.d.-b). Trustworthy Program Verification via Proof Generation, 43.

Lin, Z., Kasampalis, T., & Adve, V. (n.d.). A Translation Validation Algorithm for LLVM Register Allocators, 11.

Li, P. (n.d.). TOWARDS PRACTICAL INFORMATION FLOW ANALYSIS, 226.

Li, P., & Zhang, D. (2021). Towards a General-Purpose Dynamic Information Flow Policy. *arXiv:2109.08096 [cs]*. Retrieved from arXiv:2109.08096

Li, S., Qiao, L., & Yang, M. (2021). Memory State Verification Based on Inductive and Deductive Reasoning. *IEEE Transactions on Reliability*, 1–14. https://doi.org/10.1109/TR.2021.3074709

Li, S.-W., Li, X., Gu, R., Nieh, J., & Hui, J. Z. (n.d.). A Secure and Formally Verified Linux KVM Hypervisor, 18.

Li, T., Bai, J.-J., Sui, Y., & Hu, S.-M. (2022). Path-sensitive and alias-aware typestate analysis for detecting OS bugs. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 859–872). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3503222.3507770

Liu, B., Liu, P., Li, Y., Tsai, C.-C., Da Silva, D., & Huang, J. (2021). When threads meet events: efficient and precise static race detection with origins. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 725–739). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3453483.3454073

Liu, C., Chen, Y., & Lu, L. (n.d.). KUBO: Precise and Scalable Detection of User-triggerable Undefined Behavior Bugs in OS Kernel, 15.

Liu, C., Zhang, X., Chai, K. K., Loo, J., & Chen, Y. (n.d.). A survey on blockchain-enabled smart grids: Advances, applications and challenges. *IET Smart Cities*, *n/a*. https://doi.org/10.1049/smc2.12010

Liu, J., Corbett-Davies, J., Ferraiuolo, A., Ivanov, A., Luo, M., Suh, G. E., … Campbell, M. (2018). Secure Autonomous Cyber-Physical Systems Through Verifiable Information Flow Control. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy* (pp. 48–59). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3264888.3264889

Liu, J., Hallahan, W., Schlesinger, C., Sharif, M., Lee, J., Soulé, R., … Foster, N. (2018). p4v: practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (pp. 490–503). Budapest Hungary: ACM. https://doi.org/10.1145/3230543.3230582

Liu, M., Rieg, L., Shao, Z., Gu, R., Costanzo, D., Kim, J.-E., & Yoon, M.-K. (2019). Virtual timeline: a formal abstraction for verifying preemptive schedulers with temporal isolation. *Proceedings of the ACM on Programming Languages*, *4*, 1–31. https://doi.org/10.1145/3371088

Li, Y., & Weirich, S. (n.d.). Program Adverbs: Structures for Embedding Effectful Programs, 31.

Li, Y., Xia, L., & Weirich, S. (2021). Reasoning about the garden of forking paths. *arXiv:2103.07543 [cs]*. Retrieved from arXiv:2103.07543

Li, Y., Zhang, Q., & Reps, T. (2020). Fast graph simplification for interleaved Dyck-reachability. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 780–793). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3385412.3386021

Li, Y., Zhang, Q., & Reps, T. (2021). On the complexity of bidirected interleaved Dyck-reachability. *Proceedings of the ACM on Programming Languages*, *5*, 59:1–59:28. https://doi.org/10.1145/3434340

Li, Z., Tang, J., Zou, D., Chen, Q., Xu, S., Zhang, C., … Jin, H. (2021). Towards Making Deep Learning-based Vulnerability Detectors Robust. *arXiv:2108.00669 [cs]*. Retrieved from arXiv:2108.00669

Li, Z., Wang, J., Sun, M., & Lui, J. C. S. (n.d.). MirChecker: Detecting Bugs in Rust Programs via Static Analysis, 14.

Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic. (n.d.). Retrieved September 8, 2021, from `http://plv.mpi-sws.org/ISL/`

Lombardi, H., & Quitté, C. (2015). Commutative algebra: Constructive methods. Finite projective modules. *arXiv:1605.04832 [math]*, *20*. https://doi.org/10.1007/978-94-017-9944-7

Longley, J., & Normann, D. (2015). *Higher-Order Computability*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-47992-6

Loring, B. W. (n.d.). Practical Dynamic Symbolic Execution for JavaScript, 222.

Losa, G., & Dodds, M. (2020). On the Formal Verification of the Stellar Consensus Protocol, 9.

Lubin, J., & Chasins, S. E. (2021). How statically-typed functional programmers write code. *Proceedings of the ACM on Programming Languages*, *5*, 155:1–155:30. https://doi.org/10.1145/3485532

Lu, J., He, D., & Xue, J. (2021). Eagle: CFL-Reachability-Based Precision-Preserving Acceleration of Object-Sensitive Pointer Analysis with Partial Context Sensitivity. *ACM Transactions on Software Engineering and Methodology*, *30*(4), 46:1–46:46. https://doi.org/10.1145/3450492

Luo, W., & Demsky, B. (2021). C11Tester: A Race Detector for C/C++ Atomics Technical Report. *arXiv:2102.07901 [cs]*. Retrieved from arXiv:2102.07901

Luo, Z. (n.d.). An Extended Calculus of Constructions. Retrieved February 1, 2019, from http://www.lfcs.inf.ed.ac.uk/reports/90/ECS-LFCS-90-118/

Magrino, T., Liu, J., Foster, N., Gehrke, J., & Myers, A. C. (2019). Efficient, Consistent Distributed Computation with Predictive Treaties. In *Proceedings of the Fourteenth EuroSys Conference 2019* (pp. 1–16). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3302424.3303987

Mahboubi, A. (n.d.). Machine-checked computer-aided mathematics, 111.

Maillard \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Kenji, Ahman \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Danel, Atkey \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Robert, Martínez \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Guido, Hritcu \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Catalin, Rivas \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Exequiel, & Tanter \textless}/span \textgreater}, {\textless}span itemprop=“author” itemtype=“http://schema org/Person”{\textgreater}Éric. (2019). Dijkstra Monads for All. In *24th ACM SIGPLAN International Conference on Functional Programming (ICFP)*. Retrieved from https://arxiv.org/abs/1903.01237

Maillard, K., Margulies, N., Sozeau, M., Tabareau, N., & Tanter, É. (n.d.). The Multiverse: Logical Modularity for Proof Assistants, 28.

Majumder, K., & Bondhugula, U. (2021). HIR: An MLIR-based Intermediate Representation for Hardware Accelerator Description. *arXiv:2103.00194 [cs]*. Retrieved from arXiv:2103.00194

Makarov, E., & Spitters, B. (2013). The Picard Algorithm for Ordinary Differential Equations in Coq. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.), *Interactive Theorem Proving* (Vol. 7998, pp. 463–468). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39634-2_34

Malecha, G., Anand, A., & Stewart, G. (2020). Towards an Axiomatic Basis for C++. *The Coq Workshop 2020*, 3.

Malecha, G., Ricketts, D., Alvarez, M. M., & Lerner, S. (2016). Towards foundational verification of cyber-physical systems. In *2016 Science of Security for Cyber-Physical Systems Workshop (SOSCYPS)* (pp. 1–5). https://doi.org/10.1109/SOSCYPS.2016.7580000

Manna, Z., & Pnueli, A. (1995). *Temporal Verification of Reactive Systems: Safety*. New York: Springer-Verlag. Retrieved from https://www.springer.com/gp/book/9780387944593

Mansky, W. (n.d.). Verifying Concurrent Programs with VST, 15.

Maršík, J., Amblard, M., & Groote, P. de. (2021). Introducing , a -calculus for Effectful Computation. *Theoretical Computer Science*. https://doi.org/10.1016/j.tcs.2021.02.038

Martin-Dorel, É., & Melquiond, G. (2016). Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq. *Journal of Automated Reasoning*, *57*(3), 187–217. https://doi.org/10.1007/s10817-015-9350-4

Martínez, G., Ahman, D., Dumitrescu, V., Giannarakis, N., Hawblitzel, C., Hritcu, C., … Swamy, N. (2018). Meta-Fstar: Proof Automation with SMT, Tactics, and Metaprograms. *arXiv:1803.06547 [cs]*. Retrieved from arXiv:1803.06547

Martin, K., Hoffman, B., & Cedilnik, A. (2013). *Mastering CMake: a cross-platform build system ; covers installing and running CMake ; details converting existing build processes to CMake ; create powerful cross-platform build scripts* (6. ed). Clifton Park, NY: Kitware.

Marty, J.-J., Franceschino, L., Talpin, J.-P., & Vazou, N. (2020). LIO*: Low Level Information Flow Control in F*. *arXiv:2004.12885 [cs]*. Retrieved from arXiv:2004.12885

Masuda, M., & Kameyama, Y. (n.d.). Unified Program Generation and Verification: A Case Study on Number-Theoretic Transform, 19.

Matita - Interactive Theorem Prover. (n.d.). Retrieved January 13, 2020, from http://matita.cs.unibo.it/

Matsushita, Y., Tsukada, T., & Kobayashi, N. (2020). RustHorn: CHC-based Verification for Rust Programs (full version). *arXiv:2002.09002 [cs]*. Retrieved from arXiv:2002.09002

Ma, X., Yan, J., Wang, W., Yan, J., Zhang, J., & Qiu, Z. (n.d.). Detecting Memory-Related Bugs by Tracking Heap Memory Management of C++ Smart Pointers, 12.

Mazuera-Rozo, A., Escobar-Velásquez, C., Espitia-Acero, J., Vega-Guzmán, D., Trubiani, C., Linares-Vásquez, M., & Bavota, G. (2022). Taxonomy of security weaknesses in Java and Kotlin Android apps. *Journal of Systems and Software*, 111233. https://doi.org/10.1016/j.jss.2022.111233

Mcbride, C., & Paterson, R. (2008). Applicative programming with effects. *Journal of Functional Programming*, *18*(1). https://doi.org/10.1017/S0956796807006326

McCauley, J., Harchol, Y., Panda, A., Raghavan, B., & Shenker, S. (2019). Enabling a permanent revolution in internet architecture. In *Proceedings of the ACM Special Interest Group on Data Communication* (pp. 1–14). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3341302.3342075

McKinna, J. (2006). Why Dependent Types Matter. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 1–1). New York, NY, USA: ACM. https://doi.org/10.1145/1111037.1111038

Medina-Martínez, D., Bárcenas, E., Molero-Castillo, G., Velázquez-Mena, A., & Aldeco-Pérez, R. (2021). Database Management System Verification with Separation Logics. *Programming and Computer Software*, *47*(8), 654–672. https://doi.org/10.1134/S036176882108017X

Melquiond, G. (n.d.). Why3. Retrieved February 1, 2019, from http://why3.lri.fr/

Memarian, K., Gomes, V. B. F., Davis, B., Kell, S., Richardson, A., Watson, R. N. M., & Sewell, P. (2019). Exploring C Semantics and Pointer Provenance. *Proc. ACM Program. Lang.*, *3*, 67:1–67:32. https://doi.org/10.1145/3290380

Menon, A., Murugan, S., Rebeiro, C., Gala, N., & Veezhinathan, K. (2017). Shakti-T: A RISC-V Processor with Light Weight Security Extensions. In *Proceedings of the Hardware and Architectural Support for Security and Privacy* (pp. 2:1–2:8). New York, NY, USA: ACM. https://doi.org/10.1145/3092627.3092629

Merigoux, D., & Fromherz, A. (2020, January). Steel: scaling up memory reasoning for F* (ADSL 2020) - POPL 2020. Retrieved January 22, 2021, from https://popl20.sigplan.org/details/adsl-2020-papers/8/Steel-scaling-up-memory-reasoning-for-F-

Merigoux, D., Kiefer, F., & Bhargavan, K. (n.d.). Hacspec: succinct, executable, verifiable specifications for high-assurance cryptography embedded in Rust, 17.

Meseguer, J. (n.d.). Symbolic Computation in Maude: Some Tapas, 26.

Messadi, I., Neumann, S., Weichbrodt, N., Almstedt, L., Mahhouk, M., & Kapitza, R. (2021). Precursor: a fast, client-centric and trusted key-value store using RDMA and Intel SGX. In *Proceedings of the 22nd International Middleware Conference* (pp. 1–13). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3464298.3476129

Metis Theorem Prover - Gilith. (n.d.). Retrieved January 10, 2020, from http://www.gilith.com/metis/

Meyer, B., Arkadova, A., Kogtenkov, A., & Naumchev, A. (2021). The concept of class invariant in object-oriented programming. *arXiv:2109.06557 [cs]*. Retrieved from arXiv:2109.06557

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. Retrieved from arXiv:1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546 [cs, Stat]*. Retrieved from arXiv:1310.4546

Mikolov, T., Yih, S. W., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. Retrieved from https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations/

Milner, R. (1973). *Models of LCF* (No. {STAN}-{CS}-73-332, Memo {AIM}-186) (p. 19). Stanford University: Stanford University. Retrieved from http://i.stanford.edu/pub/cstr/reports/cs/tr/73/332/CS-TR-73-332.pdf

Miné, A. (2007). The Octagon Abstract Domain. *arXiv:cs/0703084*. Retrieved from arXiv:cs/0703084

Miné, A., Mastroeni, I., Møller, A., Chailloux, E., Logozzo, F., Müller, P., … Verona, U. di. (n.d.). Static Type and Value Analysis by Abstract Interpretation of Python Programs with Native C Libraries, 271.

Miné, A., Mauborgne, L., Rival, X., Feret, J., Cousot, P., Kästner, D., … Ferdinand, C. (2016). Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*. Toulouse, France. Retrieved from https://hal.archives-ouvertes.fr/hal-01271552

Minsky, Y., Madhavapeddy, A., & Hickey, J. (n.d.). Real World OCaml. Retrieved February 1, 2019, from http://dev.realworldocaml.org/

Mirliaz, S., & Pichardie, D. (2021). *A Flow-Insensitive-Complete Program Representation*. Retrieved from https://hal.archives-ouvertes.fr/hal-03384612

Mirman, M., Gehr, T., & Vechev, M. (n.d.). Differentiable Abstract Interpretation for Provably Robust Neural Networks, 13.

Mirman, M., Singh, G., & Vechev, M. (2020). A Provable Defense for Deep Residual Networks. *arXiv:1903.12519 [cs, Stat]*. Retrieved from arXiv:1903.12519

Mitchell, S. (2015, December 16). What is OpenFlow and why should you care? Retrieved January 21, 2021, from https://itbrief.co.nz/story/what-openflow-and-why-should-you-care

Mitsch, S. (n.d.). Implicit and Explicit Proof Management in KeYmaera X, 15.

Mitsch, S., & Platzer, A. (2016). ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, *49*(1), 33–74. https://doi.org/10.1007/s10703-016-0241-z

Mi, Z., Zhuang, H., Zang, B., & Chen, H. (2021). General and Fast Inter-Process Communication via Bypassing Privileged Software. *IEEE Transactions on Computers*, 1–1. https://doi.org/10.1109/TC.2021.3130751

Mokhov, A. (2017). Algebraic Graphs with Class (Functional Pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell* (pp. 2–13). New York, NY, USA: ACM. https://doi.org/10.1145/3122955.3122956

Molina, F., Ponzio, P., Aguirre, N., & Frias, M. (2021). EvoSpex: An Evolutionary Algorithm for Learning Postconditions. *arXiv:2102.13569 [cs]*. Retrieved from arXiv:2102.13569

Møller, A., & Schwartzbach, M. I. (n.d.). *Static Program Analysis*.

Moll, S. (n.d.). *Decompilation of LLVM IR* (Bachelor's Thesis). Saarland University. Retrieved from https://compilers.cs.uni-saarland.de/publications/theses/moll_bsc.pdf

Monniaux, D. (2005). The parallel implementation of the Astr\'{e}e static analyzer. *arXiv:cs/0701191*, *3780*, 86–96. https://doi.org/10.1007/11575467_7

Monniaux, D., & Six, C. (2021). *Simple, Light, Yet Formally Verified, Global Common Subexpression Elimination and Loop-Invariant Code Motion*. Retrieved from https://hal.archives-ouvertes.fr/hal-03212087

Montagu, B., Pierce, B. C., & Pollack, R. (2013). A Theory of Information-Flow Labels. In *2013 IEEE 26th Computer Security Foundations Symposium* (pp. 3–17). https://doi.org/10.1109/CSF.2013.8

Monteiro, F. R., Gadelha, M. R., & Cordeiro, L. C. (2021). Model Checking C++ Programs. *arXiv:2107.01093 [cs]*. Retrieved from arXiv:2107.01093

Morrisett, G., Shi, E., Sojakova, K., Fan, X., & Gancher, J. (2021). *IPDL: A Simple Framework for Formally Verifying Distributed Cryptographic Protocols* (No. 147). Retrieved from http://eprint.iacr.org/2021/147

Morshtein, S., & Ettinger, R. (n.d.). Verifying Time Complexity of Binary Search using Dafny, 16.

Moscato, M. M., Titolo, L., Feliú, M. A., & Muñoz, C. A. (2019). Provably Correct Floating-Point Implementation of a Point-in-Polygon Algorithm. In M. H. ter Beek, A. McIver, & J. N. Oliveira (Eds.), *Formal Methods – The Next 30 Years* (pp. 21–37). Springer International Publishing.

Moskal, M. (2009). Programming with triggers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories* (pp. 20–29). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1670412.1670416

Mosses, P. D. (2021). Fundamental Constructs in Programming Languages. *arXiv:2107.10545 [cs]*. Retrieved from arXiv:2107.10545

Mössenböck, H., Löberbauer, M., & Wöß, A. (2013). The Compiler Generator Coco/R. Open source from University of Linz.

M. P. L., & Group, } V. (2021). *Fiat-Crypto*. Programming Languages and Verification Group at MIT CSAIL. Retrieved from https://github.com/mit-plv/fiat-crypto

Mullen, E., Pernsteiner, S., Wilcox, J. R., Tatlock, Z., & Grossman, D. (2018a). Œuf: Minimizing the Coq Extraction TCB. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 172–185). New York, NY, USA: ACM. https://doi.org/10.1145/3167089

Mullen, E., Pernsteiner, S., Wilcox, J. R., Tatlock, Z., & Grossman, D. (2018b). Œuf: Minimizing the Coq Extraction TCB. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 172–185). New York, NY, USA: ACM. https://doi.org/10.1145/3167089

Müller, P., & Shankar, N. (2021). The First Fifteen Years of the Verified Software Project. In *Theories of Programming: The Life and Works of Tony Hoare* (1st ed., Vol. 39, pp. 93–124). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3477355.3477362

Müller, P., Wolf, F. A., & Schwerhoff, M. (2020). *Concise Outlines for a Complex Logic: A Proof Outline Checker for TaDA* (Working Paper) (p. 2010.07080). Cornell University. https://doi.org/10.3929/ethz-b-000456825

Mulligan, D. P., Owens, S., Gray, K. E., Ridge, T., & Sewell, P. (2014). Lem: Reusable Engineering of Real-world Semantics. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming* (pp. 175–188). New York, NY, USA: ACM. https://doi.org/10.1145/2628136.2628143

Murawski, A. S., & Tzevelekos, N. (2016). An Invitation to Game Semantics. *ACM SIGLOG News*, *3*(2), 56–67. https://doi.org/10.1145/2948896.2948902

Murphy, L., Viger, T., Sandro, A. D., Shahin, R., & Chechik, M. (2021). Validating Safety Arguments with Lean. In R. Calinescu & C. S. Păsăreanu (Eds.), *Software Engineering and Formal Methods* (pp. 23–43). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-92124-8_2

Murray, T., Yan, P., & Ernst, G. (2021). Incremental Vulnerability Detection via Back-Propagating Symbolic Execution of Insecurity Separation Logic. *arXiv:2107.05225 [cs]*. Retrieved from arXiv:2107.05225

Myreen, M. O. (2021a). A minimalistic verified bootstrapped compiler (proof pearl). In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 32–45). Virtual Denmark: ACM. https://doi.org/10.1145/3437992.3439915

Myreen, M. O. (2021b). A minimalistic verified bootstrapped compiler (proof pearl). In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 32–45). Virtual Denmark: ACM. https://doi.org/10.1145/3437992.3439915

Myreen, M. O. (2021c). The CakeML Project's Quest for Ever Stronger Correctness Theorems, 10.

Naeem, N. A., Lhoták, O., & Rodriguez, J. (2010). Practical Extensions to the IFDS Algorithm. In R. Gupta (Ed.), *Compiler Construction* (Vol. 6011, pp. 124–144). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-11970-5_8

Namjoshi, K. S., & Tabajara, L. M. (2019). Witnessing Secure Compilation. *arXiv:1911.05866 [cs]*. Retrieved from arXiv:1911.05866

Namjoshi, K. S., & Xue, A. (2021). A Self-certifying Compilation Framework for WebAssembly. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 127–148). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_7

Nelson, L., Sigurbjarnarson, H., Zhang, K., Johnson, D., Bornholt, J., Torlak, E., & Wang, X. (2017a). Hyperkernel: Push-Button Verification of an OS Kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles* (pp. 252–269). New York, NY, USA: ACM. https://doi.org/10.1145/3132747.3132748

Nelson, L., Sigurbjarnarson, H., Zhang, K., Johnson, D., Bornholt, J., Torlak, E., & Wang, X. (2017b). Hyperkernel: Push-Button Verification of an OS Kernel - Slides. In *Proceedings of the 26th Symposium on Operating Systems Principles - SOSP '17* (pp. 252–269). Shanghai, China: ACM Press. https://doi.org/10.1145/3132747.3132748

Neumann, P. G. (n.d.). The RISKS Digest. *The RISKS Digest*. Retrieved from `http://catless.ncl.ac.uk/risks/`

New, M. S., Licata, D. R., & Ahmed, A. (2021). Gradual type theory. *Journal of Functional Programming*, *31*, e21. https://doi.org/10.1017/S0956796821000125

Nguyen, V.-A., Nguyen, D. Q., Nguyen, V., Le, T., Tran, Q. H., & Phung, D. (2021). ReGVD: Revisiting Graph Neural Networks for Vulnerability Detection. *arXiv:2110.07317 [cs]*. Retrieved from arXiv:2110.07317

Nienhuis, K., Memarian, K., & Sewell, P. (2016). An Operational Semantics for C/C++11 Concurrency. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 111–128). New York, NY, USA: ACM. https://doi.org/10.1145/2983990.2983997

Nie, P., Palmskog, K., Li, J. J., & Gligoric, M. (2021). Roosterize: Suggesting Lemma Names for Coq Verification Projects Using Deep Learning. *arXiv:2103.01346 [cs]*. Retrieved from arXiv:2103.01346

Nipkow, T., Blanchette, J., Eberl, M., Gómez-Londoño, A., Lammich, P., Sternagel, C., … Zhan, B. (n.d.). Functional Algorithms, Verified!, 276.

Oak, A., Ahmadian, A. M., Balliu, M., & Salvaneschi, G. (n.d.). Language Support for Secure Software Development with Enclaves, 16.

O'Callahan, R., Jones, C., Froyd, N., Huey, K., Noll, A., & Partush, N. (2017). Engineering Record And Replay For Deployability: Extended Technical Report. *arXiv:1705.05937 [cs]*. Retrieved from arXiv:1705.05937

O'Connor, B., Tseng, Y., Pudelko, M., Cascone, C., Endurthi, A., Wang, Y., … Vahdat, A. (2019). Using P4 on Fixed-Pipeline and Programmable Stratum Switches. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (pp. 1–2). https://doi.org/10.1109/ANCS.2019.8901885

O'Connor, L., Chen, Z., Rizkallah, C., Jackson, V., Amani, S., Klein, G., … Keller, G. (2021). Cogent: uniqueness types and certifying compilation. *Journal of Functional Programming*, *31*, e25. https://doi.org/10.1017/S095679682100023X

O'Connor, R., & Spitters, B. (2010). A computer-verified monadic functional implementation of the integral. *Theoretical Computer Science*, *411*(37), 3386–3402. https://doi.org/10.1016/j.tcs.2010.05.031

O'Hearn, P. (2015). From Categorical Logic to Facebook Engineering. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (pp. 17–20). Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/LICS.2015.11

O'Hearn, P. (2019). Separation logic. *Communications of the ACM*, *62*(2), 86–95. https://doi.org/10.1145/3211968

O'Hearn, P., Reynolds, J., & Yang, H. (2001). Local Reasoning about Programs that Alter Data Structures. In L. Fribourg (Ed.), *Computer Science Logic* (pp. 1–19). Springer Berlin Heidelberg.

O'Hearn, P. W. (2018). Continuous Reasoning: Scaling the impact of formal methods. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18* (pp. 13–25). Oxford, United Kingdom: ACM Press. https://doi.org/10.1145/3209108.3209109

O'Hearn, P. W. (2019). Incorrectness logic. *Proceedings of the ACM on Programming Languages*, *4*, 10:1–10:32. https://doi.org/10.1145/3371078

ONF. (2021). SDN Technical Specifications.Open Networking Foundation. Retrieved January 21, 2021, from https://opennetworking.org/software-defined-standards/specifications/

Ongaro, D., & Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm, 18.

Open Compute Project.Open Compute Project. (n.d.). Retrieved January 20, 2021, from https://www.opencompute.org

OpenTheory Project - Gilith. (n.d.). Retrieved January 10, 2020, from http://www.gilith.com/opentheory/

Open vSwitch. (n.d.). Retrieved January 14, 2021, from https://www.openvswitch.org/

Open vSwitch 2.15.90 documentation. (n.d.). Retrieved January 20, 2021, from https://docs.openvswitch.org/en/latest/

O-RAN ALLIANCE.O-RAN ALLIANCE. (n.d.). Retrieved January 14, 2021, from https://www.o-ran.org

Ozdemir, A., Brown, F., & Wahby, R. S. (2020). *Unifying Compilers for SNARKs, SMT, and More* (No. 1586). Retrieved from http://eprint.iacr.org/2020/1586

Padon, O., McMillan, K. L., Panda, A., Sagiv, M., & Shoham, S. (2016). Ivy: Safety Verification by Interactive Generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 614–630). New York, NY, USA: ACM. https://doi.org/10.1145/2908080.2908118

Pakin, S. (n.d.). The Comprehensive LaTeX Symbol List, 358.

Palit, T., Moon, J. F., Monrose, F., & Polychronakis, M. (n.d.). DynPTA: Combining Static and Dynamic Analysis for Practical Selective Data Protection, 19.

Panchenko, M., Auler, R., Nell, B., & Ottoni, G. (2019). BOLT: a practical binary optimizer for data centers and beyond. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization* (pp. 2–14). Washington, DC, USA: IEEE Press.

Paraskevopoulou, Z., & Grover, A. (2021). Compiling with continuations, correctly. *Proceedings of the ACM on Programming Languages*, *5*, 114:1–114:29. https://doi.org/10.1145/3485491

Paraskevopoulou, Z., Li, J. M., & Appel, A. W. (n.d.). Compositional Optimizations for CertiCoq, *5*, 30.

Parkinson, M. J., & Summers, A. J. (2012). THE RELATIONSHIP BETWEEN SEPARATION LOGIC AND IMPLICIT DYNAMIC FRAMES. *Logic Methods in Computer Science*, *802*(3), 54. https://doi.org/10.2168/LMCS-8(3:1)2012

Parkinson, M. J., & Summers, A. J. (n.d.). The Relationship between Separation Logic and Implicit Dynamic Frames. *LNCS*, *6602*, 439–458.

Parr, T., Harwell, S., & Fisher, K. (n.d.). Adaptive LL(*) Parsing: The Power of Dynamic Analysis, 19.

Passmore, G., Cruanes, S., Ignatovich, D., Aitken, D., Bray, M., Kagan, E., … Mometto, N. (2020). The Imandra Automated Reasoning System (System Description). In N. Peltier & V. Sofronie-Stokkermans (Eds.), *Automated Reasoning* (pp. 464–471). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-51054-1_30

Patel, N., Krishna, S., Shasha, D., & Wies, T. (2021). Verifying Concurrent Multicopy Search Structures. *arXiv:2109.05631 [cs]*. Retrieved from arXiv:2109.05631

Patrignani, M., Ahmed, A., & Clarke, D. (2019). Formal Approaches to Secure Compilation: A Survey of Fully Abstract Compilation and Related Work. *ACM Computing Surveys*, *51*(6), 125:1–125:36. https://doi.org/10.1145/3280984

Patrignani, M., & Garg, D. (2021). Robustly Safe Compilation, an Efficient Form of Secure Compilation. *ACM Transactions on Programming Languages and Systems*, *43*(1), 1–41. https://doi.org/10.1145/3436809

Patterson, D., & Ahmed, A. (n.d.-a). On Compositional Compiler Correctness and Fully Abstract Compilation, 3. Retrieved from https://popl18.sigplan.org/event/prisc-2018-on-compositional-compiler-correctness-and-fully-abstract-compilation

Patterson, D., & Ahmed, A. (n.d.-b). On Compositional Compiler Correctness and Fully Abstract Compilation - POPL 2018. Retrieved February 1, 2019, from https://popl18.sigplan.org/event/prisc-2018-on-compositional-compiler-correctness-and-fully-abstract-compilation

Paulson, L. (2021). *The Inductive Approach to Verifying Cryptographic Protocols*.

Paulson, L. C. (2000). The Foundation of a Generic Theorem Prover. *arXiv:cs/9301105*. Retrieved from arXiv:cs/9301105

Peng, D., Zheng, S., Li, Y., Ke, G., He, D., & Liu, T.-Y. (2021). How could Neural Networks understand Programs? *arXiv:2105.04297 [cs]*. Retrieved from arXiv:2105.04297

Perceus: Garbage Free Reference Counting with ReuseMicrosoft Technical Report, MSR-TR-2020-42, Jan 11, 2021, v3. (2020), 41.

Pereira, F. M. Q., & Berlin, D. (2009). Wave Propagation and Deep Propagation for Pointer Analysis. In *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization* (pp. 126–135). USA: IEEE Computer Society. https://doi.org/10.1109/CGO.2009.9

Perez-Lopez, Á. R. (n.d.). Puppetmaster: a certified hardware architecture for task parallelism, 49.

Petcher, A., & Morrisett, G. (2015). The Foundational Cryptography Framework. In R. Focardi & A. Myers (Eds.), *Principles of Security and Trust* (pp. 53–72). Springer Berlin Heidelberg. Retrieved from http://www.cs.cornell.edu/~jgm/papers/FCF.pdf

Peterson, L., Anderson, T., Katti, S., McKeown, N., Parulkar, G., Rexford, J., … Vahdat, A. (2019). Democratizing the Network Edge. *ACM SIGCOMM Computer Communication Review*, *49*(2), 31–36. https://doi.org/10.1145/3336937.3336942

Peterson, L., & Davie, B. (2020). Computer Networks: A Systems Approach — Computer Networks: A Systems Approach Version 6.2-dev documentation. Retrieved December 31, 2020, from https://book.systemsapproach.org/

Peterson, L., Davie, B., Cascone, C., O'Connor, B., & Vachuska, T. (2020). Software-Defined Networks: A Systems Approach. Retrieved December 31, 2020, from https://sdn.systemsapproach.org/

Peterson, L., & Sunay, O. (2020). 5G Mobile Networks: A Systems Approach. Retrieved December 31, 2020, from https://5g.systemsapproach.org/

Petiot, G., Kosmatov, N., Botella, B., Giorgetti, A., & Julliand, J. (2015). Your Proof Fails? Testing Helps to Find the Reason. *arXiv:1508.01691 [cs]*. Retrieved from arXiv:1508.01691

Petiot, G., Kosmatov, N., Botella, B., Giorgetti, A., & Julliand, J. (2018). How testing helps to diagnose proof failures. *Formal Aspects of Computing*, *30*(6), 629–657. https://doi.org/10.1007/s00165-018-0456-4

Pichardie, D. (2008). Building Certified Static Analysers by Modular Construction of Well-founded Lattices. *Electronic Notes in Theoretical Computer Science*, *212*, 225–239. https://doi.org/10.1016/j.entcs.2008.04.064

Pickard, M., & Hutton, G. (2021). Calculating dependently-typed compilers (functional pearl). *Proceedings of the ACM on Programming Languages*, *5*, 1–27. https://doi.org/10.1145/3473587

Pimpalkhare, N., Mora, F., Polgreen, E., & Seshia, S. A. (n.d.). MedleySolver: Online SMT Algorithm Selection, 18.

Pit-Claudel, C. (n.d.). Relational compilation: functional-to-imperative code generation for performance-critical applications, 160.

Pit-Claudel, C., Wang, P., Delaware, B., Gross, J., & Chlipala, A. (n.d.). Extensible Extraction of Efficient Imperative Programs with Foreign Functions, Manually Managed Memory, and Proofs, 14. Retrieved from http://pit-claudel.fr/clement/papers/fiat-to-facade.pdf

Pizzuti, F., Steuwer, M., & Dubach, C. (2021). Generating high performance code for irregular data structures using dependent types. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing* (pp. 37–49). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3471873.3472977

Platzer, A. (2008). Differential Dynamic Logic for Hybrid Systems. *Journal of Automated Reasoning*, *41*(2), 143–189. https://doi.org/10.1007/s10817-008-9103-8

Platzer, A. (2015). Differential Game Logic. *ACM Trans. Comput. Logic*, *17*(1), 1:1–1:51. https://doi.org/10.1145/2817824

Platzer, A. (2017). A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *Journal of Automated Reasoning*, *59*(2), 219–265. https://doi.org/10.1007/s10817-016-9385-1

Platzer, A. (2018a). Differential Equations & Differential Invariants. In A. Platzer, *Logical Foundations of Cyber-Physical Systems* (pp. 287–322). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-63588-0_10

Platzer, A. (2018). *Logical Foundations of Cyber-Physical Systems*. Springer International Publishing. Retrieved from https://www.springer.com/gp/book/9783319635873

Platzer, A. (2018b). *Logical Foundations of Cyber-Physical Systems - Slides*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-63588-0

Platzer, A. (n.d.). KeYmaera X: Documentation. Retrieved January 31, 2019, from http://www.ls.cs.cmu.edu/KeYmaeraX/documentation.html

Plotkin, G. D. (1977). LCF considered as a programming language. *Theoretical Computer Science*, *5*(3), 223–255. https://doi.org/10.1016/0304-3975(77)90044-5

Poettering, B., Rösler, P., Schwenk, J., & Stebila, D. (2021). *SoK: Game-based Security Models for Group Key Exchange* (No. 305). Retrieved from https://eprint.iacr.org/2021/305

Poial, J. (n.d.). Forth and Formal Language Theory, 6.

Polikarpova, N. (2021). Synthesis of Safe Pointer-Manipulating Programs. In L. Cohen & C. Kaliszyk (Eds.), *12th International Conference on Interactive Theorem Proving (ITP 2021)* (Vol. 193, pp. 2:1–2:1). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ITP.2021.2

Polikarpova, N., & Sergey, I. (2019). Structuring the Synthesis of Heap-manipulating Programs. *Proc. ACM Program. Lang.*, *3*, 72:1–72:30. https://doi.org/10.1145/3290385

Polikarpova, N., Tschannen, J., & Furia, C. A. (2018). A fully verified container library. *Formal Aspects of Computing*, *30*(5), 495–523. https://doi.org/10.1007/s00165-017-0435-1

Porncharoenwase, S., Nelson, L., Wang, X., & Torlak, E. (2022). A formal foundation for symbolic evaluation with merging. *Proceedings of the ACM on Programming Languages*, *6*, 1–28. https://doi.org/10.1145/3498709

Potteiger, B., Dubey, A., Cai, F., Koutsoukos, X., & Zhang, Z. (2022). Moving target defense for the security and resilience of mixed time and event triggered cyber-physical systems. *Journal of Systems Architecture*, 102420. https://doi.org/10.1016/j.sysarc.2022.102420

Pottier, F., & REgis-Gianas, Y. (n.d.). Menhir Reference Manual (version 20181113). Retrieved February 1, 2019, from http://gallium.inria.fr/~fpottier/menhir/manual.html

Power, J. F., & Sinclair, D. (n.d.). *A Formal Model of Forth Control Words in the Pi-Calculus*.

Protzenko, J., Beurdouche, B., Merigoux, D., & Bhargavan, K. (2019). Formally Verified Cryptographic Web Applications in WebAssembly. In *2019 IEEE Symposium on Security and Privacy (SP)* (pp. 1256–1274). San Francisco, CA, USA: IEEE. https://doi.org/10.1109/SP.2019.00064

Protzenko, J., Zinzindohoué, J.-K., Rastogi, A., Ramananandro, T., Wang, P., Zanella-Béguelin, S., … Swamy, N. (2017). Verified Low-level Programming Embedded in Fstar. *Proc. ACM Program. Lang.*, *1*, 17:1–17:29. https://doi.org/10.1145/3110261

Provable Security - Amazon Web Services (AWS).Amazon Web Services, Inc. (n.d.). Retrieved January 14, 2021, from https://aws.amazon.com/security/provable-security/

Pujet, L., & Tabareau, N. (2022). Observational Equality: Now For Good, 29.

Pulte, C., Flur, S., Deacon, W., French, J., Sarkar, S., & Sewell, P. (2017). Simplifying ARM Concurrency: Multicopy-atomic Axiomatic and Operational Models for ARMv8. *Proc. ACM Program. Lang.*, *2*, 19:1–19:29. https://doi.org/10.1145/3158107

Punchihewa, H., & Wu, N. (2021). Safe mutation with algebraic effects. In *Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell* (pp. 122–135). Virtual Republic of Korea: ACM. https://doi.org/10.1145/3471874.3472988

Qian, Z., Kavvos, G. A., & Birkedal, L. (n.d.). Client-Server Sessions in Linear Logic, *1*(1), 28.

qsharp-language. (2021). Microsoft. Retrieved from https://github.com/microsoft/qsharp-language

Quines, C. (n.d.). Type Theory by Example. Retrieved September 24, 2021, from https://www.cjquines.com/files/typetheory.pdf

Qureshi, Z. H. (n.d.). Formal Modelling and Analysis of Mission-Critical Software in Military Avionics Systems. *11th Australian Workshop on Safety Related Programmable Systems (SCS'06)*, 11. Retrieved from http://crpit.com/confpapers/CRPITV69Qureshi.pdf

Qu, W., Gaboardi, M., & Garg, D. (2021). Relational cost analysis in a functional-imperative setting. *Journal of Functional Programming*, *31*, e27. https://doi.org/10.1017/S0956796821000071

Raad, A., Berdine, J., Dang, H.-H., Dreyer, D., O'Hearn, P., & Villard, J. (n.d.). Local Reasoning about the Presence of Bugs: Incorrectness Separation Logic, 41.

Rakotomalala, L., Roux, P., & Boyer, M. (2021). Verifying min-plus Computations with Coq (extended version with appendix). In *13th NASA Formal Methods Symposium (NFM 2021)*. virtual, United States. Retrieved from https://hal.archives-ouvertes.fr/hal-03176024

Ramsey, N., & Dias, J. (2006). An Applicative Control-Flow Graph Based on Huet's Zipper. *Electronic Notes in Theoretical Computer Science*, *148*(2), 105–126. https://doi.org/10.1016/j.entcs.2005.11.042

Rand, R. (n.d.). Formally Verified Quantum Programming, 222.

Rastogi, A. (n.d.). Programming and Proving with Indexed Effects, 28.

Rastogi, A., Martínez, G., Fromherz, A., Ramananandro, T., & Swamy, N. (n.d.). Layered Indexed Effects. *Unpublished*, *1*, 28.

Rastogi, A., Swamy, N., & Hicks, M. (2019). Wys*: A DSL for Verified Secure Multi-party Computations. In *Principles of Security and Trust (POST 2019)*. Retrieved from https://www.microsoft.com/en-us/research/publication/wys-a-dsl-for-verified-secure-multi-party-computations/

Recto, R., Algehed, M., & Myers, A. C. (n.d.). Secure Information Flow for Concurrent Programs with Expressive Synchronization, 28.

Redmond, P., Shen, G., & Kuper, L. (2021). Toward Hole-Driven Development with Liquid Haskell. *arXiv:2110.04461 [cs]*. Retrieved from arXiv:2110.04461

Reich, J., Monsanto, C., Foster, N., Rexford, J., & Walker, D. (2013). Modular SDN Programming with Pyretic, *38*(5), 8.

Reitblatt, M. (2017). Formal Reasoning in Software-defined Networks. https://doi.org/10.7298/X45M63PH

REMS: Rigorous Engineering of Mainsteam Systems, Papers. (n.d.). Retrieved December 27, 2019, from https://www.cl.cam.ac.uk/~pes20/rems/rems-all.html

Reps, T., Horwitz, S., & Sagiv, M. (1995). Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 49–61). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/199448.199462

Reps, T., Horwitz, S., & Sagiv, M. (n.d.). Precise Interprocedural Dataflow Analysis with Applications to Constant Propagation. https://doi.org/10.1016/0304-3975(96)00072-2

Reynolds, A., Iosif, R., Serban, C., & King, T. (2016). A Decision Procedure for Separation Logic in SMT. In C. Artho, A. Legay, & D. Peled (Eds.), *Automated Technology for Verification and Analysis* (Vol. 9938, pp. 244–261). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-46520-3_16

Riecke, C. (2016, July 13). Frenetic Programmers Guide.GitHub. Retrieved January 13, 2021, from https://github.com/frenetic-lang/manual

Riehl, E. (2017). *Category Theory in Context*. Dover Publications.

Riganelli, O., Fagadau, I. D., Micucci, D., & Mariani, L. (2022). Proactive Libraries: Enforcing Correct Behaviors in Android Apps. *arXiv:2202.11999 [cs]*. https://doi.org/10.1145/3510454.3516837

Ringer, T. (n.d.). *Proof Repair - Talia Ringer Thesis* (phdthesis).

Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., & Tatlock, Z. (2019). QED at Large: A Survey of Engineering of Formally Verified Software. *Foundations and Trends® in Programming Languages*, *5*(2), 102–281. https://doi.org/10.1561/2500000045

Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., & Tatlock, Z. (2019). *QED at Large: A Survey of Engineering of Formally Verified Software*. now. Retrieved from http://ieeexplore.ieee.org/document/8824174

Robles, V., Kosmatov, N., Prevosto, V., Rilling, L., & Paris-Saclay, U. (n.d.). Methodology for Specification and Verification of High-Level Requirements with MetAcsl, 14.

Rocha Pinto, P. da, Dinsdale-Young, T., & Gardner, P. (2014). TaDA: A Logic for Time and Data Abstraction. In R. Jones (Ed.), *ECOOP 2014 – Object-Oriented Programming* (pp. 207–231). Springer Berlin Heidelberg.

Ross, P. E. (2005a). The exterminators [software bugs]. *IEEE Spectrum*, *42*(9), 36–41. https://doi.org/10.1109/MSPEC.2005.1502527

Ross, P. E. (2005b). The exterminators [software bugs. *IEEE Spectrum*, *42*(9), 36–41. https://doi.org/10.1109/MSPEC.2005.1502527

Rosu, G. (2017a, July). Guarded Matching Logic is Decidable. Retrieved December 7, 2021, from https://www.ideals.illinois.edu/bitstream/handle/2142/112795/rodrigues-chen-rosu-2022-tr-decidability.pdf?sequence=2&isAllowed=y

Rosu, G. (2017b, December). Matching logic · Formal Systems Laboratory. Retrieved May 24, 2021, from https://fsl.cs.illinois.edu/publications/rosu-2017-lmcs.html

Roy, S., Hsu, J., & Albarghouthi, A. (2021). Learning Differentially Private Mechanisms. *arXiv:2101.00961 [cs]*. Retrieved from arXiv:2101.00961

Ruffy, F., Wang, T., & Sivaraman, A. (n.d.). Gauntlet: Finding Bugs in Compilers for Programmable Packet Processing, 17.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Russinovich, M., Costa, M., Fournet, C., Chisnall, D., Delignat-Lavaud, A., Clebsch, S., … Bhatia, V. (2021). Toward Confidential Cloud Computing: Extending hardware-enforced cryptographic protection to data while in use. *Queue*, *19*(1), Pages 20:49 – Pages 20:76. https://doi.org/10.1145/3454122.3456125

Saborido, J. B. (n.d.). Verification of linked data structures in Dafny, 68.

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic Routing Between Capsules. *arXiv:1710.09829 [cs]*. Retrieved from arXiv:1710.09829

Saillard, R. (2015). *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice* (phdthesis). Ecole Nationale Supérieure des Mines de Paris. Retrieved from https://pastel.archives-ouvertes.fr/tel-01299180

Sakaguchi, K. (2022). Reflexive tactics for algebra, revisited. *arXiv:2202.04330 [cs]*. Retrieved from arXiv:2202.04330

Salvia, R., Titolo, L., Feliú, M. A., Moscato, M. M., Muñoz, C. A., & Rakamarić, Z. (2019). A Mixed Real and Floating-Point Solver. In J. M. Badger & K. Y. Rozier (Eds.), *NASA Formal Methods* (pp. 363–370). Springer International Publishing.

Sammler, M., Lepigre, R., & Krebbers, R. (2021). RefinedC: Automating the Foundational Verification of C Code with Refined Ownership Types, 17.

Sanan, D., Zhao, Y., Lin, S.-W., & Yang, L. (2021). CSim^2: Compositional Top-down Verification of Concurrent Systems using Rely-Guarantee. *ACM Transactions on Programming Languages and Systems*, *43*(1), 2:1–2:46. https://doi.org/10.1145/3436808

Sanchez-Stern, A. (n.d.). *Hybrid-Neural Synthesis of Machine Checkable Software Correctness Proofs* (phdthesis). UC San Diego. Retrieved from https://escholarship.org/uc/item/5j10b5w8

Schaik, S. van, Kwong, A., Genkin, D., & Yarom, Y. (n.d.). SGAxe: How SGX Fails in Practice, 14.

Schaik, S. van, Minkin, M., Kwong, A., Genkin, D., & Yarom, Y. (2021). CacheOut: Leaking Data on Intel CPUs via Cache Evictions. In *2021 IEEE Symposium on Security and Privacy (SP)* (pp. 339–354). San Francisco, CA, USA: IEEE. https://doi.org/10.1109/SP40001.2021.00064

Scharager, M., Cordwell, K., Mitsch, S., & Platzer, A. (2021). Verified Quadratic Virtual Substitution for Real Arithmetic. *arXiv:2105.14183 [cs]*. Retrieved from arXiv:2105.14183

Schmid, G., & Kunčak, V. (2021). Proving and Disproving Programs with Shared Mutable Data. *arXiv:2103.07699 [cs]*. Retrieved from arXiv:2103.07699

Schnepf, N., Badonnel, R., Lahmadi, A., & Merz, S. (2017). Automated verification of security chains in software-defined networks with synaptic. In *2017 IEEE Conference on Network Softwarization (NetSoft)* (pp. 1–9). https://doi.org/10.1109/NETSOFT.2017.8004195

Schoolderman, M., Moerman, J., Smetsers, S., & Eekelen, M. van. (2020). Efficient Verification of Optimized Code: Correct High-speed Curve25519. *arXiv:2012.09919 [cs]*. Retrieved from arXiv:2012.09919

Schoolderman, M., Moerman, J., Smetsers, S., & Eekelen, M. van. (2021). *Efficient Verification of Optimized Code: Correct High-speed X25519* (No. 415). Retrieved from https://eprint.iacr.org/2021/415

Schubert, P. D., Hermann, B., & Bodden, E. (2019). PhASAR: An Inter-procedural Static Analysis Framework for C/C++. In T. Vojnar & L. Zhang (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems* (Vol. 11428, pp. 393–410). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-17465-1_22

Schubert, P. D., Leer, R., Hermann, B., & Bodden, E. (2019). Know your analysis: how instrumentation aids understanding static analysis. In *Proceedings of the 8th ACM SIGPLAN International Workshop on State Of the Art in Program Analysis* (pp. 8–13). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3315568.3329965

Schubert, P. D., Leer, R., Hermann, B., & Bodden, E. (n.d.). Into the Woods: Experiences from Building a Dataflow Analysis Framework for C/C++, 6.

Schubert, P. D., Sattler, F., Schiebel, F., Hermann, B., & Bodden, E. (n.d.). Modeling the Effects of Global Variables in Data-Flow Analysis for C/C++, 6.

Schulte, E., Dorn, J., Flores-Montoya, A., Ballman, A., & Johnson, T. (2020). GTIRB: Intermediate Representation for Binaries. *arXiv:1907.02859 [cs]*. Retrieved from arXiv:1907.02859

Schumi, R., & Sun, J. (2022). ExAIS: Executable AI Semantics. *arXiv:2202.09868 [cs]*. https://doi.org/10.1145/3510003.3510112

Schwarz, M., Saan, S., Seidl, H., Apinis, K., Erhard, J., & Vojdani, V. (2021). Improving Thread-Modular Abstract Interpretation. *arXiv:2108.07613 [cs]*. Retrieved from arXiv:2108.07613

Scott, D. (n.d.). Continuous lattices.ResearchGate. Retrieved January 12, 2020, from https://www.researchgate.net/publication/251394986_Continuous_lattices

Selsam, D., & Bjørner, N. (2019). Guiding High-Performance SAT Solvers with Unsat-Core Predictions. In M. Janota & I. Lynce (Eds.), *Theory and Applications of Satisfiability Testing – SAT 2019* (pp. 336–353). Springer International Publishing.

Selsam, D., Ullrich, S., & Moura, L. de. (2020). Tabled Typeclass Resolution. *arXiv:2001.04301 [cs]*. Retrieved from arXiv:2001.04301

Sewell, P. (2019). *The Ott tool for writing definitions of programming languages and calculi: ott-lang/ott*. ott-lang. Retrieved from https://github.com/ott-lang/ott

Sewell, P. (n.d.). REMS - Rigorous Engineering of Mainstream Systems. Retrieved February 28, 2019, from https://www.cl.cam.ac.uk/~pes20/rems/index.html

Shafiq, S., & Minhas, N. M. (2014). Integrating Formal Methods in XP—A Conceptual Solution. *Journal of Software Engineering and Applications*, *07*(4), 299–310. https://doi.org/10.4236/jsea.2014.74029

Shao, Z., & Koenig, J. (n.d.). https://jhc.sjtu.edu.cn/~yutingwang/files/popl22.pdf. Retrieved December 7, 2021, from https://jhc.sjtu.edu.cn/~yutingwang/files/popl22.pdf

Shen, G., & Kuper, L. (2021). Toward SMT-Based Refinement Types in Agda. *arXiv:2110.05771 [cs]*. Retrieved from arXiv:2110.05771

Sherman, B. (2017). *Making Discrete Decisions Based on Continuous Values* (Master of Science). MIT, Cambridge, MA. Retrieved from http://adam.chlipala.net/theses/sherman_sm.pdf

Shi, L., Li, Y., Loo, B. T., & Alur, R. (2021). Network Traffic Classification by Program Synthesis. *TACAS21*, 19.

Shin, J.-Y., Kim, J., Honoré, W., Vanzetto, H., Radhakrishnan, S., Balakrishnan, M., & Shao, Z. (2019). WormSpace: A Modular Foundation for Simple, Verifiable Distributed Systems. In *Proceedings of the ACM Symposium on Cloud Computing - SoCC '19* (pp. 299–311). Santa Cruz, CA, USA: ACM Press. https://doi.org/10.1145/3357223.3362739

Shi, Q., Yao, P., Wu, R., & Zhang, C. (2021). Path-Sensitive Sparse Analysis withoutPath Conditions, 14.

Shrobe, H., DeHon, A., & Knight, T. (2009). Trust-Management, Intrusion-Tolerance, Accountability, and Reconstitution Architecture (TIARA), 133. Retrieved from https://apps.dtic.mil/dtic/tr/fulltext/u2/a511350.pdf

Sidhpurwala, H. (2019, August 21). Security flaws caused by compiler optimizations. Retrieved May 28, 2021, from https://www.redhat.com/en/blog/security-flaws-caused-compiler-optimizations

Silver, L., & Zdancewic, S. (n.d.). Dijkstra Monads Forever: Termination-Sensitive Specifications for Interaction Trees, *5*, 28.

Singh, S., Chaturvedy, K., & Mishra, B. (2021). Multi-View Learning for Repackaged Malware Detection. In *The 16th International Conference on Availability, Reliability and Security* (pp. 1–9). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3465481.3470040

Šinkarovs, A., & Cockx, J. (2021a). Choosing is Losing: How to combine the benefits of shallow and deep embeddings through reflection. *arXiv:2105.10819 [cs]*. Retrieved from arXiv:2105.10819

Šinkarovs, A., & Cockx, J. (2021b). Extracting the Power of Dependent Types, 13.

Sison, R., & Murray, T. (2020). Verified Secure Compilation for Mixed-Sensitivity Concurrent Programs. *arXiv:2010.14032 [cs]*. Retrieved from arXiv:2010.14032

Sitaraman, M., Adcock, B., Avigad, J., Bronish, D., Bucci, P., Frazier, D., … Weide, B. W. (2011). Building a push-button RESOLVE verifier: Progress and challenges. *Formal Aspects of Computing*, *23*(5), 607–626. https://doi.org/10.1007/s00165-010-0154-3

Sivaramakrishnan, K., Dolan, S., White, L., Kelly, T., Jaffer, S., & Madhavapeddy, A. (2021). Retrofitting effect handlers onto OCaml. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (pp. 206–221). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3453483.3454039

Six, C. (2021). *Compilation optimisante et formellement prouvée pour un processeur VLIW* (Theses). Université Grenoble Alpes. Retrieved from https://hal.archives-ouvertes.fr/tel-03326923

Six, C., Boulmé, S., & Monniaux, D. (2020). Certified and efficient instruction scheduling: application to interlocked VLIW processors. *Proceedings of the ACM on Programming Languages*, *4*, 1–29. https://doi.org/10.1145/3428197

Sjösten, A., Hedin, D., & Sabelfeld, A. (2018). Information Flow Tracking for Side-Effectful Libraries. In C. Baier & L. Caires (Eds.), *Formal Techniques for Distributed Objects, Components, and Systems* (pp. 141–160). Springer International Publishing.

Skalka, C., Ring, J., Darias, D., Kwon, M., Gupta, S., Diller, K., … Foster, N. (2019). Proof-Carrying Network Code. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer*

and Communications Security (pp. 1115–1129). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3319535.3363214

SLEdge: Bounded Model Checking in Separation Logic (ADSL 2020) - POPL 2020. (n.d.). Retrieved January 22, 2021, from https://popl20.sigplan.org/details/adsl-2020-papers/1/SLEdge-Bounded-Model-Checking-in-Separation-Logic

Smolka, G. (n.d.). Modeling and Proving in Computational Type Theory Using the Coq Proof Assistant, 338.

Smolka, S., Foster, N., Hsu, J., Kappé, T., Kozen, D., & Silva, A. (2019). Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proceedings of the ACM on Programming Languages*, *4*, 61:1–61:28. https://doi.org/10.1145/3371129

Smolka, S. J. (2019). A (Co)algebraic Approach to Programming and Verifying Computer Networks. https://doi.org/10.7298/1dpd-c128

Smolka, S., Kumar, P., Foster, N., Kozen, D., & Silva, A. (2017). Cantor meets Scott: semantic foundations for probabilistic networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (pp. 557–571). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3009837.3009843

Smolka, S., Kumar, P., Kahn, D. M., Foster, N., Hsu, J., Kozen, D., & Silva, A. (2019). Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 190–203). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3314221.3314639

Soare, R. I. (2016). *Turing Computability*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31933-4

Sokar, G., Mocanu, D. C., & Pechenizkiy, M. (2021). Self-Attention Meta-Learner for Continual Learning. *arXiv:2101.12136 [cs]*. Retrieved from arXiv:2101.12136

Soni, H., Rifai, M., Kumar, P., Doenges, R., & Foster, N. (2020). Composing Dataplane Programs with P4. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (pp. 329–343). Virtual Event USA: ACM. https://doi.org/10.1145/3387514.3405872

Sozeau, M. (2010). Equations: A Dependent Pattern-Matching Compiler. In M. Kaufmann & L. C. Paulson (Eds.), *Interactive Theorem Proving* (pp. 419–434). Springer Berlin Heidelberg.

Sozeau, M. (2019). *MetaCoq - Metaprogramming in Coq (Was template-coq)*. MetaCoq. Retrieved from https://github.com/MetaCoq/metacoq

Sozeau, M. (2021). Touring the MetaCoq Project (Invited Paper). *Electronic Proceedings in Theoretical Computer Science*, *337*, 13–29. https://doi.org/10.4204/EPTCS.337.2

Sozeau, M. (n.d.-a). Subset coercions in Coq. *Springer-Verlag LNCS*, 237–252.

Sozeau, M. (n.d.-b). The MetaCoq Project, 39.

Sozeau, M. (n.d.-c). Typed Template Coq - POPL 2018. Retrieved February 1, 2019, from https://popl18.sigplan.org/event/coqpl-2018-typed-template-coq

Sozeau, M. (n.d.-d). Typed Template Coq - Slides, 11.

Sozeau, M., & Oury, N. (2008a). *First-class type classes*.

Sozeau, M., & Oury, N. (2008b). First-Class Type Classes - TPHOLs Talk. In O. A. Mohamed, C. Muñoz, & S. Tahar (Eds.), *Theorem Proving in Higher Order Logics* (Vol. 5170, pp. 278–293). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-71067-7_23

Spath, J. (2019). *Synchronized Pushdown Systems for Pointer and Data-Flow Analysis* (phdthesis). Paderborn University, Paderborn.

Späth, J., Ali, K., & Bodden, E. (2017). IDEal: efficient and precise alias-aware dataflow analysis. *Proceedings of the ACM on Programming Languages*, *1*, 99:1–99:27. https://doi.org/10.1145/3133923

Späth, J., Ali, K., & Bodden, E. (2019). Context-, flow-, and field-sensitive data-flow analysis using synchronized Pushdown systems. *Proceedings of the ACM on Programming Languages*, *3*, 48:1–48:29. https://doi.org/10.1145/3290361

Späth, J., Do, L. N. Q., Ali, K., & Bodden, E. (2016). Boomerang: Demand-Driven Flow- and Context-Sensitive Pointer Analysis for Java, 26.

Spector-Zabusky, A. (2021). *DON'T MIND THE FORMALIZATION GAP: THE DESIGN AND USAGE OF HS-TO-COQ* (phdthesis). University of Pennsylvania. Retrieved from https://www.cis.upenn.edu/~sweirich/papers/spector-zabusky-thesis.pdf

Spector-Zabusky, A., Breitner, J., Rizkallah, C., & Weirich, S. (2018). Total Haskell is Reasonable Coq. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 14–27). New York, NY, USA: ACM. https://doi.org/10.1145/3167092

Spitters, B. (n.d.). A Verified Pipeline from a Specification Language to Optimized, Safe Rust. Retrieved December 7, 2021, from https://cs.au.dk/~spitters/CoqPL22.pdf

Staats, W. (2021). Scanner Project V2. Colsa.

Stanford, C., Veanes, M., & Bj, N. (2021). Symbolic Boolean Derivatives for Efficiently Solving Extended Regular Expression Constraints, 16.

Stefano, L. D. (n.d.). Verification of Distributed Systems via Sequential Emulation, 42.

Steinberg, F., Théry, L., & Thies, H. (n.d.). Computable analysis and notions of continuity in Coq, *17*, 43.

Stewart, G. (n.d.). Verified Separate Compilation for C | Computer Science Department at Princeton University. Retrieved March 3, 2019, from https://www.cs.princeton.edu/research/techreps/TR-980-15

Stewart, G., Beringer, L., & Appel, A. W. (2012). Verified Heap Theorem Prover by Paramodulation. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming* (pp. 3–14). New York, NY, USA: ACM. https://doi.org/10.1145/2364527.2364531

Stoddart, B., Ritchie, C., & Dunne, S. (2012). Forth Semantics for Compiler Verification.

Stoenescu, R., Dumitrescu, D., Popovici, M., Negreanu, L., & Raiciu, C. (2018). Debugging P4 programs with vera. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (pp. 518–532). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3230543.3230548

Strange Loop. (n.d.). "Proof Theory Impressionism: Blurring the Curry-Howard Line" by Dan Pittman. Retrieved from https://www.youtube.com/watch?v=jrVPB-Ad5Gc&t=31s

Straub, J. (2020). The Use of Runtime Verification for Identifying and Responding to Cybersecurity Threats Posed to State Actors During Cyberwarfare. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 83–87). https://doi.org/10.1109/CSCI51800.2020.00021

Strydonck, T. V., Leuven, K., Georges, A. L., Gueneau, A., & Trieu, A. (n.d.). Proving full-system security properties under multiple attacker models on capability machines, 16.

Subramaniam, C. L. (2021). *From dependent type theory to higher algebraic structures* (phdthesis). Retrieved from arXiv:2110.02804

Suchy, B., Ghosh, S., Kersnar, D., Chai, S., Huang, Z., Nelson, A., … Dinda, P. (2022). CARAT CAKE: Replacing Paging via Compiler/Kernel Cooperation, 18.

Sui, Y., Cheng, X., Zhang, G., & Wang, H. (2020). Flow2Vec: value-flow-based precise code embedding. *Proceedings of the ACM on Programming Languages*, *4*, 1–27. https://doi.org/10.1145/3428301

Sui, Y., Di, P., & Xue, J. (2016). Sparse flow-sensitive pointer analysis for multithreaded programs. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (pp. 160–170). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2854038.2854043

Sui, Y., Fan, X., Zhou, H., & Xue, J. (2018). Loop-Oriented Pointer Analysis for Automatic SIMD Vectorization. *ACM Transactions on Embedded Computing Systems*, *17*(2), 56:1–56:31. https://doi.org/10.1145/3168364

Sui, Y., & Xue, J. (2016a). On-demand strong update analysis via value-flow refinement. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 460–473). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2950290.2950296

Sui, Y., & Xue, J. (2016b). SVF: interprocedural static value-flow analysis in LLVM. In *Proceedings of the 25th International Conference on Compiler Construction* (pp. 265–266). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2892208.2892235

Sui, Y., & Xue, J. (2020). Value-Flow-Based Demand-Driven Pointer Analysis for C and C++. *IEEE Transactions on Software Engineering*, *46*(8), 812–835. https://doi.org/10.1109/TSE.2018.2869336

Sui, Y., Ye, D., & Xue, J. (2014). Detecting Memory Leaks Statically with Full-Sparse Value-Flow Analysis. *IEEE Transactions on Software Engineering*, *40*(2), 107–122. https://doi.org/10.1109/TSE.2014.2302311

Su, J., Tian, C., & Duan, Z. (2021). Conditional interpolation: making concurrent program verification more effective. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 144–154). Athens Greece: ACM. https://doi.org/10.1145/3468264.3468602

Suneja, S., Zheng, Y., Zhuang, Y., Laredo, J. A., & Morari, A. (2021). Towards Reliable AI for Source Code Understanding. In *Proceedings of the ACM Symposium on Cloud Computing* (pp. 403–411). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3472883.3486995

Sun, X., Li, L., Bissyandé, T. F., Klein, J., Octeau, D., & Grundy, J. (2021). Taming Reflection: An Essential Step Toward Whole-program Analysis of Android Apps. *ACM Transactions on Software Engineering and Methodology*, *30*(3), 32:1–32:36. https://doi.org/10.1145/3440033

Svendsen, K., & Birkedal, L. (2014). Impredicative Concurrent Abstract Predicates. In Z. Shao (Ed.), *Programming Languages and Systems* (Vol. 8410, pp. 149–168). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-54833-8_9

Svendsen, K., Birkedal, L., & Parkinson, M. (2013). Modular Reasoning about Separation of Concurrent Data Structures. In M. Felleisen & P. Gardner (Eds.), *Programming Languages and Systems* (Vol. 7792, pp. 169–188). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37036-6_11

Swamy, N. (n.d.). Project Everest - Verified Secure Implementations of the HTTPS Ecosystem.Microsoft Research. Retrieved February 1, 2019, from https://www.microsoft.com/en-us/research/project/project-everest-verified-secure-implementations-https-ecosystem/

Swamy, N., Chen, J., & Livshits, B. (2013). Verifying Higher-order Programs with the Dijkstra Monad. Retrieved from https://www.microsoft.com/en-us/research/publication/verifying-higher-order-programs-with-the-dijkstra-monad/

Swamy, N., Hriţcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., … Zanella-Béguelin, S. (2016). Dependent Types and Multi-monadic Effects in Fstar. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 256–270). New York, NY, USA: ACM. https://doi.org/10.1145/2837614.2837655

Swamy, N., Rastogi, A., Fromherz, A., Merigoux, D., Ahman, D., & Martínez, G. (2020). SteelCore: an extensible concurrent separation logic for effectful dependently typed programs. *Proceedings of the ACM on Programming Languages*, *4*, 121:1–121:30. https://doi.org/10.1145/3409003

Swierstra, W. (2008). Data types à la carte. *Journal of Functional Programming*, *18*(4). https://doi.org/10.1017/S0956796808006758

Syme, D. (2019a). *Fsharp design: RFCs and docs related to the F# language design process,*. F# Software Foundation Repositories. Retrieved from https://github.com/fsharp/fslang-design

Syme, D. (2019b). *The Fsharp Compiler, Core Library & Tools (F# Software Foundation Repository): fsharp/fsharp*. F# Software Foundation Repositories. Retrieved from https://github.com/fsharp/fsharp

Tabareau, N., Tanter, É., & Sozeau, M. (2021). The Marriage of Univalence and Parametricity. *Journal of the ACM*, *68*(1), 5:1–5:44. https://doi.org/10.1145/3429979

Tanaka, A. (2021). Coq to C translation with partial evaluation. In *Proceedings of the 2021 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation* (pp. 14–31). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3441296.3441394

Tan, Y. K., Heule, M. J. H., & Myreen, M. O. (2021). cake_lpr: Verified Propagation Redundancy Checking in CakeML. In J. F. Groote & K. G. Larsen (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 223–241). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-72013-1_12

Tao, R., Yao, J., Li, X., Li, S.-W., Nieh, J., & Gu, R. (2021). Formal Verification of a Multiprocessor Hypervisor on Arm Relaxed Memory Hardware. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (pp. 866–881). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3477132.3483560

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, *5*(2), 285–309. Retrieved from https://www.projecteuclid.org/journals/pacific-journal-of-mathematics/volume-5/issue-2/A-lattice-theoretical-fixpoint-theorem-and-its-applications/pjm/1103044538.full

Thakur, A., Lal, A., Lim, J., & Reps, T. (2013). PostHat and All That: Automating Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, 20.

The ATS Programming Language. (n.d.). Retrieved January 10, 2020, from http://www.ats-lang.org/

The convergence of source code and binary vulnerability discovery - A case study | EURECOM. (n.d.). Retrieved November 26, 2021, from https://www.eurecom.fr/publication/6732

The Mezzo programming language. (n.d.). Retrieved August 5, 2019, from http://protz.github.io/mezzo/

Theng, M. (2022). *GoTxn: Verifying a Crash-Safe, Concurrent Transaction System* (phdthesis). MIT.

Theodoridis, T., Rigger, M., & Su, Z. (2022). Finding missed optimizations through the lens of dead code elimination. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 697–709). Lausanne Switzerland: ACM. https://doi.org/10.1145/3503222.3507764

Theorem Proving and the Real Numbers: Overview Proving and the Real Numbers: Overview and Challenges Lawrence C. Paulson Computer Laboratory, University of Cambridge, England lp15@cl.cam.ac.uk - [Download PDF].vdocuments.mx. (n.d.). Retrieved November 1, 2021, from https://vdocuments.mx/theorem-proving-and-the-real-numbers-overview-proving-and-the-real-numbers.html

Theorem Proving in Lean — Theorem Proving in Lean 3.4.0 documentation. (n.d.). Retrieved January 10, 2020, from https://leanprover.github.io/theorem_proving_in_lean/index.html

The ProofPower Web Pages. (n.d.). Retrieved January 10, 2020, from http://www.lemma-one.com/ProofPower/index/

Timany, A., Gregersen, S. O., Stefanesco, L., Gondelman, L., Nieto, A., & Birkedal, L. (2021). Trillium: Unifying Refinement and Higher-Order Distributed Separation Logic. *arXiv:2109.07863 [cs]*. Retrieved from arXiv:2109.07863

Timany, A., & Sozeau, M. (2018). Cumulative Inductive Types In Coq. In *FSCD*. https://doi.org/10.4230/LIPIcs.FSCD.2018.29

Tondwalkar, A., Kolosick, M., & Jhala, R. (2021). Refinements of Futures Past: Higher-Order Specification with Implicit Refinement Types (Extended Version). *arXiv:2105.01954 [cs]*. Retrieved from arXiv:2105.01954

Torfah, H., Junges, S., Fremont, D. J., & Seshia, S. A. (2021). Formal Analysis of AI-Based Autonomy: From Modeling to Runtime Assurance. In L. Feng & D. Fisman (Eds.), *Runtime Verification* (pp. 311–330). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-88494-9_19

Trinh, M.-T., Le, Q. L., David, C., & Chin, W.-N. (2013). Bi-Abduction with Pure Properties for Specification Inference. In C. Shan (Ed.), *Programming Languages and Systems* (Vol. 8301, pp. 107–123). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-03542-0_8

Tsampas, A. E.-K. S., Patrignani, M., Devriese, D., & Piessens, D. G. F. (2021). CapablePtrs: Securely Compiling Partial Programs Using the Pointers-as-Capabilities Principle (p. 16).

Tsampas, S., Nuyts, A., Devriese, D., & Piessens, F. (2020). A categorical approach to secure compilation. *arXiv:2004.03557 [cs]*. Retrieved from arXiv:2004.03557

Tschantz, M. C., & Wing, J. M. (2009). Formal Methods for Privacy. In A. Cavalcanti & D. R. Dams (Eds.), *FM 2009: Formal Methods* (Vol. 5850, pp. 1–15). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-05089-3_1

Tsuzaki, Y., & Okabe, Y. (2017). Reactive configuration updating for Intent-Based Networking. In *2017 International Conference on Information Networking (ICOIN)* (pp. 97–102). https://doi.org/10.1109/ICOIN.2017.7899484

Tu˘sil, J., Chen, X., & Rosu, G. (2021). Hyperproperties in Matching Logic, 70.

Tuch, H., Klein, G., & Norrish, M. (2007). Types, Bytes, and Separation Logic. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 97–108). New York, NY, USA: ACM. https://doi.org/10.1145/1190216.1190234

Tušil, J. (2017). An Executable Formal Semantics of C++, 87.

Ullrich, S., & Moura, L. de. (2019). Counting Immutable Beans: Reference Counting Optimized for Purely Functional Programming. *arXiv:1908.05647 [cs]*. Retrieved from arXiv:1908.05647

Ullrich, S., & Moura, L. de. (2020). Beyond Notations: Hygienic Macro Expansion for Theorem Proving Languages. *arXiv:2001.10490 [cs]*, *12167*, 167–182. https://doi.org/10.1007/978-3-030-51054-1_10

Urban, C., & Miné, A. (2021). A Review of Formal Methods applied to Machine Learning. *arXiv:2104.02466 [cs]*. Retrieved from arXiv:2104.02466

Utture, A. (2022). Fast and Precise Application Code Analysis using a Partial Library, 12.

Vacca, A., Di Sorbo, A., Visaggio, C. A., & Canfora, G. (2021). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. *Journal of Systems and Software*, *174*, 110891. https://doi.org/10.1016/j.jss.2020.110891

Vale, A. O. (n.d.). Layered and Object-Based Game Semantics, *6*, 49.

Vandenbroucke, A., & Schrijvers, T. (2019). PloNK: functional probabilistic NetKAT. *Proceedings of the ACM on Programming Languages*, *4*, 39:1–39:27. https://doi.org/10.1145/3371107

Van Renesse, R., & Altinbuken, D. (2015). Paxos Made Moderately Complex. *ACM Comput. Surv.*, *47*(3), 42:1–42:36. https://doi.org/10.1145/2673577

Vasilache, N., Zinenko, O., Bik, A. J. C., Ravishankar, M., Raoux, T., Belyaev, A., … Cohen, A. (2022). Composable and Modular Code Generation in MLIR: A Structured and Retargetable Approach to Tensor Compiler Construction. *arXiv:2202.03293 [cs]*. Retrieved from arXiv:2202.03293

Védrine, F., Jacquemin, M., Kosmatov, N., & Signoles, J. (2021). Runtime Abstract Interpretation for Numerical Accuracy and Robustness. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 243–266). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_12

Vilhena, P. E. de, & Pottier, F. (2021). A separation logic for effect handlers. *Proceedings of the ACM on Programming Languages*, *5*, 33:1–33:28. https://doi.org/10.1145/3434314

Vindum, S. F., Frumin, D., & Birkedal, L. (2022). Mechanized Verification of a Fine-Grained Concurrent Queue from Meta's Folly Library, 16.

Vintila, E. Q., Zieris, P., & Horsch, J. (2021). MESH: A Memory-Efficient Safe Heap for C/C++. In *The 16th International Conference on Availability, Reliability and Security* (pp. 1–10). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3465481.3465760

Vishwanathan, H., Shachnai, M., Narayana, S., & Nagarakatte, S. (2021). Semantics, Verification, and Efficient Implementations for Tristate Numbers. *arXiv:2105.05398 [cs]*. Retrieved from arXiv:2105.05398

Vishwanathan, H., Shachnai, M., Narayana, S., & Nagarakatte, S. (n.d.). Sound, Precise, and Fast Abstract Interpretation with Tristate Numbers, 12.

Voevodsky, V. (n.d.). Homotopy Type Theory: Univalent Foundations of Mathematics, 490.

Vu, S. T., Cohen, A., Grandmaison, A. D., Guillon, C., & Heydemann, K. (2021). Reconciling Optimization with Secure Compilation, *5*, 30.

Vu, S. T., Cohen, A., & Heydemann, K. (2021). Secure Optimization Through Opaque Observations. *Preprint PriSC Workshop (with POPL 2021)*, 43.

Vu, S. T., Heydemann, K., Grandmaison, A. de, & Cohen, A. (2020). Secure delivery of program properties through optimizing compilation. In *Proceedings of the 29th International Conference on Compiler Construction* (pp. 14–26). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3377555.3377897

Wang, H., Zhu, H., Xiao, L., & Fei, Y. (2018). Formalization and Verification of the OpenFlow Bundle Mechanism Using CSP. *International Journal of Software Engineering and Knowledge Engineering*, *28*(11), 1657–1677. https://doi.org/10.1142/S0218194018400223

Wang, J., Huang, Y., Wang, S., & Wang, Q. (2021). Find Bugs in Static Bug Finders. *arXiv:2109.02245 [cs]*. Retrieved from arXiv:2109.02245

Wang, J., Sung, C., Raghothaman, M., & Wang, C. (2021). Data-Driven Synthesis of Provably Sound Side Channel Analyses. *arXiv:2102.06753 [cs]*. Retrieved from arXiv:2102.06753

Wang, K., Wang, J., Poskitt, C. M., Chen, X., Sun, J., & Cheng, P. (2022). K-ST: A Formal Executable Semantics of PLC Structured Text Language. *arXiv:2202.04076 [cs]*. Retrieved from arXiv:2202.04076

Wang, S. (2021). Concurrent matching logic. *arXiv:2109.00319 [cs]*. Retrieved from arXiv:2109.00319

Wang, S., Wang, P., & Wu, D. (2015). Reassembleable Disassembling (pp. 627–642). Retrieved from https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wang-shuai

Wang, S., Wang, P., & Wu, D. (2016). UROBOROS: Instrumenting Stripped Binaries with Static Reassembling. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 236–247). Suita: IEEE. https://doi.org/10.1109/SANER.2016.106

Wang, Y., Li, Q., Chen, Z., Zhang, P., Zhang, G., & Shi, Z. (2021). BCI-CFI: A Context-Sensitive Control-Flow Integrity Method Based on Branch Correlation Integrity. *Information and Software Technology*, 106572. https://doi.org/10.1016/j.infsof.2021.106572

Wan, Z., Xia, X., Lo, D., Chen, J., Luo, X., & Yang, X. (2021). Smart Contract Security: a Practitioners' Perspective. *arXiv:2102.10963 [cs]*. Retrieved from arXiv:2102.10963

Watanabe, Y., College, Y.-N., Gopinathan, K., Pîrlea, G., Polikarpova, N., Sergey, I., & College, Y.-N. (n.d.). Certifying the Synthesis of Heap-Manipulating Programs, *5*, 29.

Watertor, R. (n.d.). Assessing the standard-compliance for multi-threading primitives in C compilers, 58.

Watt, C. (2021). *Mechanising and evolving the formal semantics of WebAssembly: the Web's new low-level language* (phdthesis). St Catharine's College, University of Cambridge, Cambridge, UK.

Watt, C., Rao, X., Pichon-Pharabod, J., Bodin, M., & Gardner, P. (n.d.). Two Mechanisations of WebAssembly 1.0, 19.

Weirich, S., Voizard, A., Amorim, P. H. A. de, & Eisenberg, R. A. (2017). A Specification for Dependent Types in Haskell. *Proc. ACM Program. Lang.*, *1*, 31:1–31:29. https://doi.org/10.1145/3110275

Weisstein, E. W. (n.d.). Mathworld Classroom [Text]. Retrieved January 12, 2020, from http://mathworld.wolfram.com/classroom/

Welch, D. (2017). Scaling Up Automated Verification: A Case Study and Formal-IDE for the Construction of High Integrity Software. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 785–786). Seattle, Washington, USA: Association for Computing Machinery. https://doi.org/10.1145/3017680.3022456

Wenzel, M. (2018). The Isabelle/Isar Reference Manual. Retrieved from https://core.ac.uk/display/22830292

White Neil, Matthews Stuart, & Chapman Roderick. (2017). Formal verification: will the seedling ever flower? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *375*(2104), 20150402. https://doi.org/10.1098/rsta.2015.0402

Wickerson, J., & Brunet, P. (2020). Pearl: Diagrams for Composing Compilers, 13.

Wiedijk, F. (2008). Formal Proof—Getting Started, *55*(11), 7.

Wilcox, J. R. (2021). *Compositional and Automated Verification of Distributed Systems* (phdthesis). University of Washington.

Willis, J., & Wu, N. (2021). Design patterns for parser combinators (functional pearl). In *Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell* (pp. 71–84). Virtual Republic of Korea: ACM. https://doi.org/10.1145/3471874.3472984

Wing, J. M. (1990). A Specifier's Introduction to Formal Methods. *IEEE Computer*.

Witten, I., Frank, E., Hall, M., & Pal, C. (n.d.). Data Mining: Practical Machine Learning Tools and Techniques. Retrieved March 5, 2021, from https://www.cs.waikato.ac.nz/ml/weka/book.html

Wolf, F. A., Arquint, L., Clochard, M., Oortwijn, W., Pereira, J. C., & Müller, P. (2021). Gobra: Modular Specification and Verification of Go Programs (extended version). *arXiv:2105.13840 [cs]*. Retrieved from arXiv:2105.13840

Wolf, F. A., Schwerhoff, M., & Müller, P. (2021). Concise Outlines for a Complex Logic: A Proof Outline Checker for TaDA. In M. Huisman, C. Păsăreanu, & N. Zhan (Eds.), *Formal Methods* (pp. 407–426). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-90870-6_22

Wolff, F. (n.d.). Modular Specification and Verification of Closures in Rust, 27.

Woos, D., Wilcox, J. R., Anton, S., Tatlock, Z., Ernst, M. D., & Anderson, T. (2016). Planning for change in a formal verification of the raft consensus protocol. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs* (pp. 154–165). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2854065.2854081

Writing Compiler Front-Ends for LLVM with Lua using Mewa.CodeProject. (2021, May 26). Retrieved June 1, 2021, from https://www.codeproject.com/Articles/5301384/Writing-Compiler-Front-Ends-for-LLVM-with-Lua-usin

Wu, Y., Lu, J., Zhang, Y., & Jin, S. (2021). Vulnerability Detection in C/C++ Source Code With Graph Representation Learning. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 1519–1524). https://doi.org/10.1109/CCWC51732.2021.9376145

Xia, L., Zakowski, Y., He, P., Hur, C.-K., Malecha, G., Pierce, B. C., & Zdancewic, S. (2019). Interaction trees: representing recursive and impure programs in Coq. *Proceedings of the ACM on Programming Languages*, *4*, 51:1–51:32. https://doi.org/10.1145/3371119

Xie, N., Brachthäuser, J. I., Hillerström, D., Schuster, P., & Leijen, D. (2020). Effect handlers, evidently. *Proceedings of the ACM on Programming Languages*, *4*, 99:1–99:29. https://doi.org/10.1145/3408981

Xie, N., & Leijen, D. (2020). Effect handlers in Haskell, evidently. In *Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell* (pp. 95–108). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3406088.3409022

Xi, H. (2017). Applied Type System: An Approach to Practical Programming with Theorem-Proving. *arXiv:1703.08683 [cs]*. Retrieved from arXiv:1703.08683

Xi, H. (n.d.). Introduction to Programming in ATS, 252.

Xu, F. F., Vasilescu, B., & Neubig, G. (2021). In-IDE Code Generation from Natural Language: Promise and Challenges. *arXiv:2101.11149 [cs]*. Retrieved from arXiv:2101.11149

Xu, X., Sui, Y., Yan, H., & Xue, J. (2019). VFix: value-flow-guided precise program repair for null pointer dereferences. In *Proceedings of the 41st International Conference on Software Engineering* (pp. 512–523). Montreal, Quebec, Canada: IEEE Press. https://doi.org/10.1109/ICSE.2019.00063

Yadav, R. (2020). Light-Weighted CNN for Text Classification. *arXiv:2004.07922 [cs, Stat]*. Retrieved from arXiv:2004.07922

Yang, J., & Hawblitzel, C. (2011). Safe to the last instruction: automated verification of a type-safe operating system. *Communications of the ACM*, *54*(12), 123. https://doi.org/10.1145/2043174.2043197

Yang, K., & Deng, J. (2019). Learning to Prove Theorems via Interacting with Proof Assistants. *arXiv:1905.09381 [cs, Stat]*. Retrieved from arXiv:1905.09381

Yang, M., Lie, D., & Papernot, N. (n.d.). Accelerating Symbolic Analysis for Android Apps, 6.

Yang, Z., Qiu, Z., Zhou, Y., Huang, Z., Bodeveix, J.-P., & Filali, M. (2021). C2AADL_Reverse: A model-driven reverse engineering approach to development and verification of safety-critical software. *Journal of Systems Architecture*, *118*, 102202. https://doi.org/10.1016/j.sysarc.2021.102202

Yan, P., & Murray, T. (n.d.). SecRSL: Security Separation Logic for C11 Release-Acquire Concurrency, *5*, 26.

Yao, P., Shi, Q., Huang, H., & Zhang, C. (2021). Program analysis via efficient symbolic abstraction. *Proceedings of the ACM on Programming Languages*, *5*, 118:1–118:32. https://doi.org/10.1145/3485495

Yao, P., Zhou, J., Xiao, X., Shi, Q., Wu, R., & Zhang, C. (2021). Efficient Path-Sensitive Data-Dependence Analysis. *arXiv:2109.07923 [cs]*. Retrieved from arXiv:2109.07923

Ye, D., Sui, Y., & Xue, J. (2014). Accelerating Dynamic Detection of Uses of Undefined Values with Static Value-Flow Analysis. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization* (pp. 154–164). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2581122.2544154

Ye, Q., & Delaware, B. (2019). A Verified Protocol Buffer Compiler. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 222–233). New York, NY, USA: ACM. https://doi.org/10.1145/3293880.3294105

Ye, W. (2016). Type-Directed Operational Semantics for Gradual Typing, 29.

Yodaiken, V. (2021). How ISO C became unusable for operating systems development. In *Proceedings of the 11th Workshop on Programming Languages and Operating Systems* (pp. 84–90). Virtual Event Germany: ACM. https://doi.org/10.1145/3477113.3487274

Yousefi-Azar, M., Varadharajan, V., Hamey, L., & Chen, S. (2021). Mutual Information and Feature Importance Gradient Boosting: Automatic byte n-gram feature reranking for Android malware detection. *Software: Practice and Experience*. https://doi.org/10.1002/spe.2971

Yuan, S., & Talpin, J.-P. (n.d.). Verified functional programming of an IoT operating system's bootloader, 17.

Yu, K., Wang, C., Cai, Y., Luo, X., & Yang, Z. (2021). Detecting concurrency vulnerabilities based on partial orders of memory and thread events. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 280–291). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3468264.3468572

Yurichev, D. (n.d.). SAT/SMT by Example, 555.

Zakowski, Y., Beck, C., Yoon, I., Zaichuk, I., Zaliva, V., & Zdancewic, S. (2021). Modular, compositional, and executable formal semantics for LLVM IR. *Proceedings of the ACM on Programming Languages*, *5*, 67:1–67:30. https://doi.org/10.1145/3473572

Zakowski, Y., He, P., Hur, C.-K., & Zdancewic, S. (2020). An equational theory for weak bisimulation via generalized parameterized coinduction. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (pp. 71–84). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3372885.3373813

Zaliva, V. (2021). *HELIX: From Math to Verified Code* (thesis). Carnegie Mellon University. https://doi.org/10.1184/R1/13636808.v1

Zaliva, V., & Franchetti, F. (2018). HELIX: a case study of a formal verification of high performance program generation. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional High-Performance Computing* (pp. 1–9). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3264738.3264739

Zave, P., & Rexford, J. (2019). The compositional architecture of the internet. *Communications of the ACM*, *62*(3), 78–87. https://doi.org/10.1145/3226588

Zave, P., Rexford, J., & Sonchack, J. (2020). The Remaining Improbable: Toward Verifiable Network Services. *arXiv:2009.12861 [cs]*. Retrieved from arXiv:2009.12861

Zeng, B. (2012). *Static Analysis on Binary Code* (Technical Report) (p. 19). Bethlehem, PA: Lehigh University.

Zhang, H., Chen, W., Hao, Y., Li, G., Zhai, Y., Zou, X., & Qian, Z. (n.d.). Statically Discovering High-Order Taint Style Vulnerabilities in OS Kernels, 14.

Zhang, H., Koh, N., Li, Y., Beringer, L., Pierce, B., Honoré, W., … Zdancewic, S. (2021). Verifying an HTTP Key-Value Server with Interaction Trees and VST, 19.

Zhang, H., Zhang, C., Azevedo de Amorim, A., Agarwal, Y., Fredrikson, M., & Jia, L. (2021). Netter: Probabilistic, Stateful Network Models. In F. Henglein, S. Shoham, & Y. Vizel (Eds.), *Verification, Model Checking, and Abstract Interpretation* (pp. 486–508). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-67067-2_22

Zhang, L., Blaauwbroek, L., Piotrowski, B., Kaliszyk, C., & Urban, J. (n.d.). Decision Trees for Tactic Prediction in Coq, 3.

Zhang, Q., Wang, J., Xu, G. H., & Kim, M. (2022). HeteroGen: Transpiling C to Heterogeneous HLS Code with Automated Test Generation and Program Repair, 13.

Zhang, T., Wiegley, J., Giannakopoulos, T., Eakman, G., Pit-Claudel, C., Lee, I., & Sokolsky, O. (2018). Correct-by-Construction Implementation of Runtime Monitors Using Stepwise Refinement. In X. Feng, M. Müller-Olm, & Z. Yang (Eds.), *Dependable Software Engineering. Theories, Tools, and Applications* (Vol. 10998, pp. 31–49). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-99933-3_3

Zhang, W., & Sun, Y. (n.d.). Compositional Programming, *1*(1), 60.

Zhang, Z., Feng, Y., Ernst, M. D., Porst, S., & Dillig, I. (2021). Checking Conformance of Applications against GUI Policies, 12.

Zhaohui, L., & Feng, X. (2021). *Verifying Contextual Refinement with Ownership Transfer(Extended Version)*.

Zhao, J., Nagarakatte, S., Martin, M. M. K., & Zdancewic, S. (2013). Formal verification of SSA-based optimizations for LLVM. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 175–186). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2491956.2462164

Zhu, W., Feng, Z., Zhang, Z., Ou, Z., Yang, M., & Zhang, C. (2021). iCallee: Recovering Call Graphs for Binaries. *arXiv:2111.01415 [cs]*. Retrieved from arXiv:2111.01415

Zou, C., Wang, X., Gao, Y., & Xue, J. (2022). Buddy Stacks: Protecting Return Addresses with Efficient Thread-Local Storage and Runtime Re-Randomization. *ACM Transactions on Software Engineering and Methodology*, *31*(2), 35e:1–35e:37. https://doi.org/10.1145/3494516

Zou, D., Wu, Y., Yang, S., Chauhan, A., Yang, W., Zhong, J., … Jin, H. (2021). IntDroid: Android Malware Detection Based on API Intimacy Analysis. *ACM Transactions on Software Engineering and Methodology*, *30*(3), 39:1–39:32. https://doi.org/10.1145/3442588

Zuleger, F., & Katelaan, J. (n.d.). Strong-Separation Logic (ADSL 2020) - POPL 2020. Retrieved January 22, 2021, from https://popl20.sigplan.org/details/adsl-2020-papers/9/Strong-Separation-Logic

Zuo, Z., Wang, K., Hussain, A., Sani, A. A., Zhang, Y., Lu, S., … Xu, G. H. (2021). Systemizing Interprocedural Static Analysis of Large-scale Systems Code with Graspan. *ACM Transactions on Computer Systems*, *38*(1), 4:1–4:39. https://doi.org/10.1145/3466820

Zuo, Z., Zhang, Y., Pan, Q., Lu, S., Li, Y., Wang, L., … Xu, G. H. (2021). Chianina: An Evolving Graph System for Flow- and Context-Sensitive Analyses of Million Lines of C Code, 16.

(2021, January 20). Fstar: A Higher-Order Effectful Language Designed for Program Verification. Retrieved January 20, 2021, from https://www.fstar-lang.org/

(n.d.-b). ACM Classification Codes. Retrieved February 1, 2019, from https://cran.r-project.org/web/classifications/ACM.html

(n.d.-c). Frama-C. Retrieved February 1, 2019, from https://frama-c.com/

(n.d.-d). LaTeX - Wikibooks, open books for an open world. Retrieved February 1, 2019, from https://en.wikibooks.org/wiki/LaTeX

(n.d.-e). MSC2010 database. Retrieved February 1, 2019, from https://mathscinet.ams.org/msc/msc2010.html