Formal Methods Review

2020-01-02 23:18

Richard L. Ford
richardlford@gmail.com

Abstract

This is a review of current formal methods theory, tools, projects and people.

Contents

0. Introduction

This document briefly summarizes current formal method groups, proof systems, projects and researchers. For each it gives pointers for further information. It is a work-in-progress and is necessarily incomplete.

1. Formal Methods Research Groups

1.0. DeepSpec Project

- Home Page :: https://deepspec.org

1.1. Galois Inc

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

1.2. Yale FLINT Group

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

”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 secrurity, 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?"

1.3. 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"."

1.4. University of Washington Programming Languages and Software Engineering

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

2. Formal Methods Proof Systems

2.0. 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.

2.1. 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.

2.2. 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."

2.3. 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."

2.4. 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."

2.5. Dafny

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

- Tutorial :: https://rise4fun.com/dafny

2.6. 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

2.7. Why3

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

2.8. Frama-C

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

2.9. F*

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

2.10. LEAN

- Papers:
  - A Metaprogramming Framework for Formal Verification

3. Formal Methods Projects

3.0. DeepSpec Projects

3.0.0. 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.

3.0.1. 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 libraries 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.

3.0.2. 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.

3.0.3. 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.

3.0.4. 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.

3.0.5. 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.

3.0.6. 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.

3.0.7. 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.”

3.0.7.0. GNU Libmicrohttpd

- Home Page :: https://www.gnu.org/software/libmicrohttpd/
- Source :: https://gnunet.org/git/libmicrohttpd.git

3.0.8. Verdi

- Home Page :: http://verdi.uwplse.org/
- Source :: https://github.com/uwplse/verdi
- Example :: https://github.com/uwplse/verdi-raft

Verification of distributed systems.

3.0.9. 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.

3.0.9.0. 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 detectection instrumentation"

3.0.10. 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.

3.0.11. DeepSpecDb

- Home Page :: ?
- 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

3.0.12. 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 https://www.cs.purdue.edu/homes/bendy/Fiat/fiat-snapl.pdf

"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."

3.0.13. 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.

3.0.14. Bedrock2

- Source ::

3.0.15. 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.”

3.0.16. Template-Coq

- Home Page :: https://template-coq.github.io/template-coq/
- Source :: https://github.com/Template-Coq/template-coq
- Papeers :: 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.

3.0.17. 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

3.0.18. Galois Voting System

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

3.1. Everest Projects

3.1.0. Everest

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

3.1.1. Quackyducky

- Source :: https://github.com/project-everest/quackyducky

QuackyDucky is 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.

3.2. Other Projects

3.2.0. 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.”

3.2.1. 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

3.2.2. Compositional CompCert

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

Compcert supporting separate compilation. Last modified in 2015.

3.2.3. 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.”

3.2.4. Bedrock

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

3.2.5. FSCQ

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

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

3.2.6. Ur/Web

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

4. Formal Methods Researchers

Alphabetically by last name, then first.

4.0. Andrew W Appel

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

4.1. Adam Chlipala

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

4.2. Robert Harper

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

4.3. 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.

4.4. Zhong Shao

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

5. 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

6. References

References

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

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

dewale, n.d.] 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

, et al., 2017] 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

, et al., 2017] 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

hrendt, n.d.] 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/

chmitt, 2007] 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

rsberg, 2018] 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

Rompf, 2016] 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/

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

bareau, n.d.] 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

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

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

al., 2017] 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

pel, 2011] 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

pel, 2012] 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

pel, 2015] 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

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

ppel, n.d.] 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

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

lot, 2017] 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

al., 2014] 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

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

ite, 1971] Bate, R. R., Mueller, D. D., & White, J. E. (1971). *Fundamentals of astrodynamics*. New York: Dover Publications.

Mark, 2017] 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

chmitt, 2006] 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

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

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

'Hearn, 2006] 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.

Appel, 2014] 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,

[Bertot, n.d.] Bertot, Y. (n.d.). Yves Bertot. Retrieved January 31, 2019, from http://www-sop.inria.fr/members/Yves.Bertot/index.html

stéran, 2004] 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

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

matov, 2017] 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

Petiot, 2018] 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.

zer, 2018] 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

tot, 2017] 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

tot, 2018] 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`

ond, 2012] 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

ond, 2013] 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`

ond, 2016] 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

eau, 2017] 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

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

al., 2018] 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`

t al., n.d.] 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.

an, 2016] 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.

rookes, 2007] 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

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

'Hearn, n.d.] 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

: Yang, 2011] 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), 1–66. https://doi.org/10.1145/2049697.2049700

Appel, 2018] 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

astéran, n.d.] Castéran, P. (n.d.). Pierre Castéran's Home page. Retrieved January 31, 2019, from http://www.labri.fr/perso/casteran/index.html

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

éraud, 2010b] 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

uéraud, 2011] 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

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

ılipala, 2013] 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/

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

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

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

t al., n.d.] 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

nd, 2017] 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

olz, 2012] 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.

out, 2018] 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

ala, 2016] 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.

das, 2016] 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

as, 2017a] 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

as, 2017b] 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

Gu, 2016] Costanzo, D., Shao, Z., & Gu, R. (2016). End-to-end Verification of Information-flow Security for C and Assembly Programs. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 648–664). New York, NY, USA: ACM. https://doi.org/10.1145/2908080.2908100

, & Gu, n.d.] 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

[Crary, 2017] 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

Takeda, 2014] 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

aliszyk, n.d.] 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

Daniel, 2017] 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

lahaye, 2000] 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

nlipala, 2015] 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

Chlipala, n.d.] Delaware, B., Suriyakarn, S., Pit-Claudel, C., Ye, Q., & Chlipala, A. (n.d.). Narcissus: Correct-By-Construction Derivation of Decoders and Encoders from Binary Formats, 14.

nlipala, 2018] 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

l et al., 2017] 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/

ijkstra, 1975] 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 ang, 2006] 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

pel, 2009] 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

ura, 2017] 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

al., 2017] 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.

ski, 1994] 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

ski, 1999] 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

ond, 2017] 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

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

wler, n.d.] Fowler, M. (n.d.). Deriving Kepler's Laws from the Inverse-Square Law. Retrieved February 1, 2019, from `http://galileo.phys.virginia.edu/classes/152.mf1i.spring02/KeplersLaws.htm`

zer, 2015] 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

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

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

nboubi, 2010] 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

Dreyer, 2011] 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

Sergey, 2019] 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

 et al., 2015] 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

 et al., 2016] 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

 et al., 2018] 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

kinson, 2013] 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

Plotkin, 1993] 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

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

son, 2013] 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

son, 2012] 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

oşu, 2015] 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

al., 2015] 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

t al., n.d.] 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.

ran, 2015] 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

ing, 1990] 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

pel, 2010] 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

an, 2013] 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.

ţcu, 2015] 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

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

. Anna, 2017] 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

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

[Inria, n.d.] Inria. (n.d.). Inria - Inventors for the digital world.Inria. Retrieved January 31, 2019, from https://www.inria.fr/en

'Hearn, 2011] 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

Jacobs, 2019] 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

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

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

iessens, 2015] 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

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

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

Dreyer, 2017] 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

g et al., 2018] Jung, R., Krebbers, R., Jourdan, J.-H., Bizjak, A., Birkedal, L., & Dreyer, D. (2018). 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

Ziliani, n.d.] 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 al., 2018] 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

nd, 2015] 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

al., n.d.] 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

al., 2017] 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

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

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

ika, 2018] 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

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

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

êlo, 2012] 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.

zel, 2015] 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.

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

mpson, 2001] 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

nchard, 1969] Lancaster, E. R., & Blanchard, R. C. (1969). A unified form of lambert's theorem. *NASA Technical Note*, *{TN} D-5368*, 18.

Lucio, 2015] 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

laudel, 2016] 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, 2016a] 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, 2016b] 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/

Moskal, 2013] 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/

Smans, 2016] 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/

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

istholz, 2016] 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/

nov, 2016] 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/

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

uzey, n.d.] 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

Luo, n.d.] 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/

cha, 2018] Malecha, G. (2018). *Reflection library for Coq. Contribute to gmalecha/template-coq development by creating an account on GitHub*. Retrieved from https://github.com/gmalecha/template-coq

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

nik, 2013] 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.

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

al., 2016] 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

ckey, n.d.] Minsky, Y., Madhavapeddy, A., & Hickey, J. (n.d.). Real World OCaml. Retrieved February 1, 2019, from http://dev.realworldocaml.org/

nov, 2017] 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 nniaux, 2005] 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

velekos, 2016] 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

et al., 2017a] 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

et al., 2017b] 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

’Hearn, 2015] 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

’Hearn, 2019] O’Hearn, P. (2019). Separation logic. *Communications of the ACM*, *62*(2), 86–95. https://doi.org/10.1145/3211968

Yang, 2001] 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.

’Hearn, 2018] 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

’Hearn, n.d.] O’Hearn, P. W. (n.d.). Peter W O’hearn - acm profile. Retrieved from `https://dl.acm.org/author_page.cfm?id=81332519314&coll=DL&dl=ACM&trk=0`

[Pakin, n.d.] Pakin, S. (n.d.). The Comprehensive LaTeX Symbol List, 358.

ronkov, 2000] Parigot, M., & Voronkov, A. (2000). *Logic for Programming and Automated Reasoning: 7th International Conference, LPAR 2000 Reunion Island, France, November 6-10, 2000 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg.

ummers, n.d.] Parkinson, M. J., & Summers, A. J. (n.d.). The Relationship between Separation Logic and Implicit Dynamic Frames. *LNCS*, *6602*, 439–458.

ed, n.d.-a] 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

ed, n.d.-b] 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

son, 2000] Paulson, L. C. (2000). The Foundation of a Generic Theorem Prover. *arXiv:cs/9301105*. Retrieved from arXiv:cs/9301105

ett, 2015] 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

nd, 2015] 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

nd, 2018] Petiot, G., Kosmatov, N., Botella, B., Giorgetti, A., & Julliand, J. (2018). How testing helps to diagnose proof failures. *Form Asp Comp*, *30*(6), 629–657. https://doi.org/10.1007/s00165-018-0456-4

udel, n.d.] Pit-Claudel, C. (n.d.). Clément Pit-Claudel. Retrieved January 31, 2019, from http://pit-claudel.fr/clement/

pala, n.d.] 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

zer, 2008] 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

zer, 2015] Platzer, A. (2015). Differential Game Logic. *ACM Trans. Comput. Logic*, *17*(1), 1:1–1:51. https://doi.org/10.1145/2817824

zer, 2017] 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

er, 2018a] 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, 2018] Platzer, A. (2018). *Logical Foundations of Cyber-Physical Systems*. Springer International Publishing. Retrieved from https://www.springer.com/gp/book/9783319635873

atzer, 2018b] 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, n.d.] Platzer, A. (n.d.). KeYmaera X: Documentation. Retrieved January 31, 2019, from http://www.ls.cs.cmu.edu/KeYmaeraX/documentation.html

Sergey, 2019] 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

Gianas, n.d.] 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

o et al., 2017] Protzenko, J., Zinzindohoué, J.-K., Rastogi, A., Ramananandro, T., Wang, P., Zanella-Béguelin, S., … Swamy, N. (2017). Verified Low-level Programming Embedded in F*. *Proc. ACM Program. Lang.*, *1*, 17:1–17:29. https://doi.org/10.1145/3110261

Qureshi, n.d.] 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

& Dias, 2006] 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

erman, 2017] 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

Knight, 2009] 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

ulman, 2013] Shulman, M. (2013, March 12). The HoTT Book.Homotopy Type Theory. Retrieved February 1, 2019, from https://homotopytypetheory.org/book/

Sozeau, 2010] 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, 2019] Sozeau, M. (2019). *MetaCoq - Metaprogramming in Coq (Was template-coq)*. MetaCoq. Retrieved from https://github.com/MetaCoq/metacoq au, n.d.-a] Sozeau, M. (n.d.-a). Typed Template Coq - POPL 2018. Retrieved February 1, 2019, from https://popl18.sigplan.org/event/coqpl-2018-typed-template-coq

u, n.d.-b] Sozeau, M. (n.d.-b). Typed Template Coq - Slides, 11.

ich, 2018] 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

pel, 2012] 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

amy, n.d.] 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/

nits, 2013] 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/

al., 2016] Swamy, N., Hriţcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., … Zanella-Béguelin, S. (2016). Dependent Types and Multi-monadic Effects in F*. 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

ne, 2019a] 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

ne, 2019b] 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

ken, 2015] 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

lsky, n.d.] Voevodsky, V. (n.d.). Homotopy Type Theory: Univalent Foundations of Mathematics, 490.

zel, 2018] Wenzel, M. (2018). The Isabelle/Isar Reference Manual. Retrieved from https://core.ac.uk/display/22830292 oderick, 2017] 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

Viedijk, 2008] Wiedijk, F. (2008). Formal Proof—Getting Started, *55*(11), 7.

ipedia, 2017] Wikipedia. (2017). Category:Formal methods people. In *Wikipedia*. Retrieved from https://en.wikipedia.org/w/index.php?title=Category:Formal_methods_people&oldid=812800009

blitzel, 2011] 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

g et al., 2018] 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

[n.d.-a] (n.d.-a). ACM Classification Codes. Retrieved February 1, 2019, from https://cran.r-project.org/web/classifications/ACM.html

[n.d.-b] (n.d.-b). Coq for PL conference series - CoqPL 2019. Retrieved January 31, 2019, from https://popl18.sigplan.org/series/CoqPL

[n.d.-c] (n.d.-c). CoqPL 2018 The Fourth International Workshop on Coq for Programming Languages - POPL 2018. Retrieved January 31, 2019, from https://popl18.sigplan.org/track/CoqPL-2018

[n.d.-d] (n.d.-d). CoqPL 2019 The Fifth International Workshop on Coq for Programming Languages - POPL 2019. Retrieved January 31, 2019, from https://popl19.sigplan.org/track/CoqPL-2019#program

[n.d.-e] (n.d.-e). Frama-C. Retrieved February 1, 2019, from https://frama-c.com/

[n.d.-f] (n.d.-f). LaTeX - Wikibooks, open books for an open world. Retrieved February 1, 2019, from https://en.wikibooks.org/wiki/LaTeX

[n.d.-g] (n.d.-g). MSC2010 database. Retrieved February 1, 2019, from https://mathscinet.ams.org/msc/msc2010.html [n.d.-h] (n.d.-h). Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. Retrieved February 1, 2019, from https://royalsocietypublishing.org/journal/rsta

[n.d.-i] (n.d.-i). POPL conference series - POPL 2020. Retrieved January 31, 2019, from https://popl18.sigplan.org/series/POPL

[n.d.-j] (n.d.-j). Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. Retrieved February 1, 2019, from https://royalsocietypublishing.org/journal/rspa