LIST OF PUBLICATIONS

Jan 2026 (to appear)

The Ghosts of Empires: Extracting Modularity from Interleaving-Based Proofs (Conference Paper)

53rd ACM SIGPLAN Symposium on Principles of Programming Languages

★ *Selected publication.*

Implementation bugs threaten the soundness of algorithmic software verifiers. Generating correctness certificates for correct programs allows for efficient independent validation of verification results, and thus helps to reveal such bugs. Automatic generation of small, compact correctness proofs for concurrent programs is challenging, as the correctness arguments may depend on the particular interleaving, which can lead to exponential explosion. We present an approach that converts an interleaving-based correctness proof, as generated by many algorithmic verifiers, into a thread-modular correctness proof in the style of Owicki and Gries. We automatically synthesize *ghost variables* that capture the relevant interleaving information, and abstract away irrelevant details. Our evaluation shows that the approach is efficient in practice and generates compact proofs, compared to a baseline.

Authors: Frank Schüssele[a], Matthias Zumkeller[a], Miriam Lagunes-Rochin[a], Dominik Klumpp[a,b]

[a] University of Freiburg, Germany [b] LIX - CNRS - École Polytechnique, France

Nov 2025

Effective AGM Belief Contraction: A Journey beyond the Finitary Realm (Conference Paper)

22nd International Conference on Principles of Knowledge Representation and Reasoning

★ *Selected publication.*

Despite significant efforts towards extending the AGM paradigm of belief change beyond finitary logics, the computational aspects of AGM have remained almost untouched. We investigate the computability of AGM contraction on non-finitary logics, and show an intriguing negative result: there are infinitely many uncomputable AGM contraction functions in such logics. Drastically, we also show that the current de facto standard strategies to control computability, which rely on restricting the space of epistemic states, fail: uncomputability remains in all non-finitary cases. Motivated by this disruptive result, we propose new approaches to controlling computability beyond the finitary realm. Using Linear Temporal Logic (LTL) as a case study, we identify an infinite class of fully-rational AGM contraction functions that are computable by design. We use Büchi automata to construct such functions, and to represent and reason about LTL beliefs.

Authors: Dominik Klumpp[a], Jandson S. Ribeiro[b]

[a] University of Freiburg, Germany *and* LIX - CNRS - École Polytechnique, France

[b] Cardiff University, United Kingdom

Jul 2025

Counterexample-Guided Commutativity (Conference Paper)

37th International Conference on Computer-Aided Verification

★ *Selected publication.*

We consider the use of commutativity-based reduction for the algorithmic verification of concurrent programs. In existing work, the commutativity relation used for the reduction is mostly fixed statically. In this paper, we propose a demand-driven approach to compute the commutativity relation. The approach can be viewed as the direct analogue of the CEGAR approach which uses counterexamples to guide the incremental refinement of the abstraction. Instead of eliminating a counterexample by proving it infeasible and refining the abstraction, we can eliminate a counterexample by proving it redundant and expanding the commutativity relation. When we prove a counterexample redundant, we use the proof for a generalization step which allows us to eliminate not just a single counterexample, but a whole infinite set. We present a general scheme where we integrate the new approach with the CEGAR approach. We have implemented an instantiation of the general scheme. An experimental evaluation shows an increase in the number of successfully verified programs by 15 % on a challenging benchmark set.

Authors: Marcel Ebbinghaus, Dominik Klumpp, Andreas Podelski – University of Freiburg, Germany

Jan 2025

Correctness Witnesses for Concurrent Programs: Bridging the Semantic Divide with Ghosts (Conference Paper)

26th International Conference on Verification, Model Checking, and Abstract Interpretation

Static analyzers are typically complex tools and thus prone to contain bugs themselves. To increase the trust in the verdict of such tools, witnesses encode key reasoning steps underlying the verdict in an exchangeable format, enabling independent validation of the reasoning by other tools. For the correctness of concurrent programs, no agreed-upon witness format exists – in no small part due to the divide between the semantics considered by analyzers, ranging from interleaving to thread-modular approaches, making it challenging to exchange information. We propose a format that leverages the well-known notion of ghosts to embed the claims a tool makes about a program into a modified program with ghosts, such that the validity of a witness can be decided by analyzing this program. Thus, the validity of witnesses with respect to the interleaving and the thread-modular semantics coincides. Further, thread-modular invariants computed by an abstract interpreter can naturally be expressed in the new format using ghost statements. We evaluate the approach by generating such ghost witnesses for a subset of concurrent programs from the SV-COMP benchmark suite, and pass them to a model checker. It can confirm 75 % of these witnesses – indicating that ghost witnesses can bridge the semantic divide between interleaving and thread-modular approaches.

Authors: Julian Erhard[a,e], Manuel Bentele[b,f], Matthias Heizmann[d], Dominik Klumpp[b], Simmo Saan[c], Frank Schüssele[b], Michael Schwarz[a], Helmut Seidl[a], Sarah Tilscher[a,e], Vesal Vojdani[c]
[a] Technical University of Munich, Germany [b] University of Freiburg, Germany
[c] University of Tartu, Estonia [d] University of Stuttgart, Germany
[e] LMU München, Germany [f] Hahn-Schickard, Villingen-Schwenningen, Germany

Nov 2024

Walking the Tightrope between Expressiveness and Uncomputability: AGM Contraction beyond the Finitary Realm (Workshop Paper)

22nd International Workshop on Nonmonotonic Reasoning

Although there has been significant interest in extending the AGM paradigm of belief change beyond finitary logics, the computational aspects of AGM have remained almost untouched. We investigate the computability of AGM contraction on non-finitary logics, and show an intriguing negative result: there are infinitely many uncomputable AGM contraction functions in such logics. Drastically, even if we restrict the theories used to represent epistemic states, in all non-trivial cases, the uncomputability remains. On the positive side, we use Büchi automata to construct computable AGM contraction functions on Linear Temporal Logic (LTL).

Authors: Dominik Klumpp[a], Jandson S. Ribeiro[b]
[a] University of Freiburg, Germany [b] Cardiff University, United Kingdom

Apr 2024

Ultimate Automizer and the Abstraction of Bitwise Operations (Competition Contribution)

30th International Conference on Tools and Algorithms for the Construction and Analysis of Systems

The verification of Ultimate Automizer works on an SMT-LIB-based model of a C program. If we choose an SMT-LIB theory of (mathematical) integers, the translation is not precise, because we overapproximate bitwise operations. In this paper we present a translation for bitwise operations that improves the precision of this overapproximation.

Authors: Frank Schüssele, Manuel Bentele, Daniel Dietsch, Matthias Heizmann, Xinyu Jiang, Dominik Klumpp, Andreas Podelski – University of Freiburg, Germany

Jan 2024

Commutativity Simplifies Proofs of Parameterized Programs (Conference Paper)

51st ACM SIGPLAN Symposium on Principles of Programming Languages

★ *Selected publication.*

Commutativity has proven to be a powerful tool in reasoning about concurrent programs. Recent work has shown that a commutativity-based reduction of a program may admit simpler proofs than the program itself. The framework of lexicographical program reductions was introduced to formalize a broad class of reductions which accommodate sequential (thread-local) reasoning as well as synchronous programs. Approaches based on this framework, however, were fundamentally limited to program models with a fixed/bounded number of threads. In this paper, we show that it is possible to define an effective parametric family of program reductions that can be used to find simple proofs for parameterized programs, i.e., for programs with an unbounded number of threads. We show that reductions are indeed useful for the simplification of proofs of parameterized programs, in a sense that can be made precise: A reduction of

a parameterized program may admit a proof which uses fewer or less sophisticated ghost variables. The reduction may therefore be within reach of an automated verification technique, even when the original parameterized program is not. As our first technical contribution, we introduce a notion of reductions for parameterized programs such that the reduction R of a parameterized program P is again a parameterized program (the thread template of R is obtained by source-to-source transformation of the thread template of P). Consequently, existing techniques for the verification of parameterized programs can be directly applied to R instead of P. Our second technical contribution is that we define an appropriate family of pairwise preference orders which can be effectively used as a parameter to produce different lexicographical reductions. To determine whether this theoretical foundation amounts to a usable solution in practice, we have implemented the approach, based on a recently proposed framework for parameterized program verification. The results of our preliminary experiments on a representative set of examples are encouraging.

Authors: Azadeh Farzan[a], Dominik Klumpp[b], Andreas Podelski[b]
[a] University of Toronto, Canada [b] University of Freiburg, Germany

Jan 2024 **Petrification: Software Model Checking for Programs with Dynamic Thread Management (Conference Paper)**

25th International Conference on Verification, Model Checking, and Abstract Interpretation

We address the verification problem for concurrent program that dynamically create (fork) new threads or destroy (join) existing threads. We present a reduction to the verification problem for concurrent programs with a fixed number of threads. More precisely, we present petrification, a transformation from programs with dynamic thread management to an existing, Petri net-based formalism for programs with a fixed number of threads. Our approach is implemented in a software model checking tool for C programs that use the pthreads API.

Authors: Matthias Heizmann, Dominik Klumpp, Lars Nitzke, Frank Schüssele – University of Freiburg, Germany

Apr 2023 **Ultimate Taipan and Race Detection in Ultimate (Competition Contribution)**

29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems

Ultimate Taipan integrates trace abstraction with algebraic program analysis on path programs. Taipan supports data race checking in concurrent programs through a reduction to reachability checking. Though the subsequent verification is not tuned for data race checking, the results are encouraging.

Authors: Daniel Dietsch, Matthias Heizmann, Dominik Klumpp, Frank Schüssele, Andreas Podelski – University of Freiburg, Germany

Apr 2023 **Ultimate Automizer and the CommuHash Normal Form (Competition Contribution)**

29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems

The verification approach of Ultimate Automizer utilizes SMT formulas. This paper presents techniques to keep the size of the formulas small. We focus especially on a normal form, called *CommuHash normal form*, that was easy to implement and had a significant impact on the runtime of our tool.

Authors: Matthias Heizmann, Max Barth, Daniel Dietsch, Leonard Fichtner, Jochen Hoenicke, Dominik Klumpp, Mehdi Naouar, Tanja Schindler, Frank Schüssele, Andreas Podelski – University of Freiburg, Germany

Jan 2023 **Stratified Commutativity in Verification Algorithms for Concurrent Programs (Conference Paper)**

50th ACM SIGPLAN Symposium on Principles of Programming Languages

★ *Selected publication.*

The importance of exploiting *commutativity relations* in verification algorithms for concurrent programs is well-known. They can help simplify the proof and improve the time and space efficiency. This paper studies commutativity relations as a first-class object in the setting of verification algorithms for concurrent programs. A first contribution is a general framework for *abstract commutativity relations*. We introduce a general soundness condition for commutativity relations, and present a method to automatically derive sound abstract commutativity relations from a given proof. The method can be used in a verification algorithm based on abstraction refinement to compute a new commutativity relation in each iteration of the abstraction refinement loop. A second result is a general proof rule that allows one to combine multiple commutativity relations, with incomparable power, in a *stratified* way that preserves soundness and allows one to profit from the full power of the combined relations. We present an algorithm

for the stratified proof rule that performs an optimal combination (in a sense made formal), enabling usage of stratified commutativity in algorithmic verification. We empirically evaluate the impact of abstract commutativity and stratified combination of commutativity relations on verification algorithms for concurrent programs.

Authors: Azadeh Farzan[a], Dominik Klumpp[b], Andreas Podelski[b]
[a] University of Toronto, Canada [b] University of Freiburg, Germany

Jun 2022 **Sound Sequentialization for Concurrent Program Verification (Conference Paper)**

43rd ACM SIGPLAN Conference on Programming Language Design and Implementation

★ *Selected publication.*

We present a systematic investigation and experimental evaluation of a large space of algorithms for the verification of concurrent programs. The algorithms are based on sequentialization. In the analysis of concurrent programs, the general idea of sequentialization is to select a subset of interleavings, represent this subset as a sequential program, and apply a generic analysis for sequential programs. For the purpose of verification, the sequentialization has to be sound (meaning that the proof for the sequential program entails the correctness of the concurrent program). We use the concept of a preference order to define which interleavings the sequentialization is to select ("the most preferred ones"). A verification algorithm based on sound sequentialization that is parametrized in a preference order allows us to directly evaluate the impact of the selection of the subset of interleavings on the performance of the algorithm. Our experiments indicate the practical potential of sound sequentialization for concurrent program verification.

Authors: Azadeh Farzan[a], Dominik Klumpp[b], Andreas Podelski[b]
[a] University of Toronto, Canada [b] University of Freiburg, Germany

Apr 2022 **Ultimate GemCutter and the Axes of Generalization (Competition Contribution)**

28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems

Ultimate GemCutter verifies concurrent programs using the CEGAR paradigm, by generalizing from spurious counterexample traces to larger sets of correct traces. We integrate classical CEGAR generalization with orthogonal generalization across interleavings. Thereby, we are able to prove correctness of programs otherwise out-of-reach for interpolation-based verification. The competition results show significant advantages over other concurrency approaches in the Ultimate family.

Authors: Dominik Klumpp[a], Daniel Dietsch[a], Matthias Heizmann[a], Frank Schüssele[a], Marcel Ebbinghaus[a], Azadeh Farzan[b], Andreas Podelski[a]
[a] University of Freiburg, Germany [b] University of Toronto, Canada

Apr 2021 **𝕂 and KIV: Towards Deductive Verification for Arbitrary Programming Languages (Conference Paper)**

25th International Workshop on Algebraic Development Techniques

Deductive program verification is a powerful tool to gain confidence in the correctness of software. However, its application to real programs faces a major hurdle: Each programming language requires development of dedicated verification tool support. In this work, we aim to advance deductive software verification to arbitrary programming languages. We developed a tool that derives algebraic specifications for the deductive proof assistant KIV from the syntax and operational semantics of a programming language specified in the 𝕂 semantic framework. We adapt and implement the generic One-Path Reachability calculus and provide instant tool support for deductive proofs. Through a sophisticated automation approach, we drastically reduce the manual proof steps.

Authors: Dominik Klumpp[a], Philip Lenzen[b]
[a] University of Freiburg, Germany [b] Augsburg University, Germany

Jan 2021 **Verification of Concurrent Programs Using Petri Net Unfoldings (Conference Paper)**

22nd International Conference on Verification, Model Checking, and Abstract Interpretation

Given a verification problem for a concurrent program (with a fixed number of threads) over infinite data domains, we can construct a model checking problem for an abstraction of the concurrent program through a Petri net (a problem which can be solved using McMillan's unfoldings technique). We present a method of abstraction refinement which translates Floyd/Hoare-style proofs for sample traces into additional synchronization constraints for the Petri net.

Authors: Daniel Dietsch, Matthias Heizmann, Dominik Klumpp, Mehdi Naouar, Andreas Podelski, Claus Schätzle – University of Freiburg, Germany

Dec 2018

Automated Control Flow Reconstruction from Assembler Programs (Master Thesis)

Master Thesis for the Elite Graduate Program Software Engineering, University of Augsburg

Standard software analyses typically require the control flow graph (CFG) of the analysed software. This poses no problem for analyses based on the source code. Sometimes however, an analysis must be based on the compiled binary, e.g. if the source code is unavailable or lacks the necessary low-level information. Control flow analysis of binary code is a non-trivial task due to the presence of *indirect branches* that transfer control to a dynamically computed location. A formal notion of control flow graphs for binary programs is presented, and quality requirements for sensible CFGs are proposed. The automated verification technique *trace abstraction refinement* is adapted to construct provably sound and precise CFGs. The approach is evaluated empirically on a standard set of benchmarks.

Advisor: Wolfgang Reif[a] **Supervisors:** Franck Cassez[b], Gerhard Schellhorn[a]
[a] University of Augsburg, Germany [b] Macquarie University, Sydney, Australia

Nov 2018

Measuring and Evaluating the Performance of Self-Organization Mechanisms within Collective Adaptive Systems (Conference Paper)

8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation

By restructuring and reconfiguring itself at run-time, through self-organization (SO) mechanisms, a collective adaptive system is able to fulfill its requirements under uncertain, ever-changing environmental conditions. These unpredictable conditions pose a challenge for the design of high-performing SO mechanisms. In this paper, we present a performance metric for SO mechanisms, and a framework that enables design-time evaluation of this metric. We evaluate the metric within the framework for a smart energy management system, and a self-organizing production cell.

Authors: Benedikt Eberhardinger, Hella Ponsar, Dominik Klumpp, Wolfgang Reif – University of Augsburg, Germany

Nov 2017

Test Case Selection Strategy for Self-Organization Mechanisms (Chapter)

Test, Analyse und Verifikation von Software – gestern, heute, morgen

A major challenge in testing self-organization mechanisms is the large state space resulting from the systems' autonomy and their unpredictable environments. This challenges the test case generation and selection, since exhaustive testing is largely impossible. This paper presents an approach for test case generation and selection tailored for self-organization mechanisms, efficiently reducing the number of test cases compared to a random selection strategy without losing adequacy. The approach is empirically evaluated using a self-organizing production cell scenario.

Authors: Benedikt Eberhardinger, Hella Seebach, Dominik Klumpp, Wolfgang Reif – University of Augsburg, Germany

Sep 2016

Optimising Runtime Safety Analysis Efficiency for Self-Organising Systems (Conference Paper)

2016 IEEE 1st International Workshops on Foundations and Applications of Self Systems (FAS*W)*

Self-organising resource-flow systems typically have a high tolerance for component faults: The system can replace failed components by other components of the same type. However, this redundancy is eventually exhausted. An analysis of these self-organisation limits is essential to assess the system's safety. This paper presents several techniques that significantly reduce analysis time in order to facilitate safety analyses at runtime. An evaluation of these techniques, using a case study producing personalised medicine, demonstrates the performance benefits.

Authors: Dominik Klumpp, Axel Habermaier, Benedikt Eberhardinger, Hella Seebach – University of Augsburg, Germany