Report from Dagstuhl Seminar 23412

Formal Methods for Correct Persistent Programming

Ori Lahav*[1], Azalea Raad*[2], Joseph Tassarotti*[3], Viktor Vafeiadis*[4], and Anton Podkopaev†[5]

1 **Tel Aviv University, IL.** orilahav@tau.ac.il
2 **Imperial College London, GB.** azalea.raad@imperial.ac.uk
3 **New York University, US.** jt4767@nyu.edu
4 **MPI-SWS – Kaiserslautern, DE.** viktor@mpi-sws.org
5 **JetBrains – Amsterdam, NL.** anton@podkopaev.net

Abstract

Recently developed non-volatile memory (NVM) devices provide persistency guarantees along with byte-addressable accesses and performance characteristics that are much closer to volatile random-access memory (RAM). However, writing programs that correctly use these devices is challenging, and bugs related to their use can cause permanent data loss in applications.

This Dagstuhl Seminar brought together experts in a range of areas related to concurrency and persistent memory to explore and develop formal methods for ensuring the correctness of applications that use persistent memory. Talks and discussions at the seminar highlighted challenges related to correctness criteria for concurrent objects using persistent memory, liveness properties of persistent objects, and how changes in NVM and related technologies should shape the development of formal methods for NVM.

Seminar October 8–11, 2023 – https://www.dagstuhl.de/23412
2012 ACM Subject Classification Hardware → Non-volatile memory; Theory of computation → Program semantics; Theory of computation → Program verification
Keywords and phrases concurrency, formal methods, non-volatile-memory, persistency, verification
Digital Object Identifier 10.4230/DagRep.13.10.50

1 Executive Summary

Ori Lahav
Azalea Raad
Joseph Tassarotti
Viktor Vafeiadis

Many systems and applications need to store data in a durable way. Historically, durable storage devices had considerably higher latency than volatile random-access memory (RAM) and provided interfaces with larger, coarser access granularity. To achieve acceptable performance, applications requiring durable storage were structured to account for these characteristics. However, in recent years, novel storage systems, such as *non-volatile memory* (NVM), have emerged that provide durability along with performance and access granularity much closer to RAM. This provides the opportunity for applications to achieve durability with much lower latencies.

* Editor / Organizer
† Editorial Assistant / Collector

However, taking full advantage of that opportunity involves restructuring applications to make proper use of these new devices. Doing so requires care because bugs in these parts of applications can cause permanent data loss. Moreover, non-volatile memory interacts in subtle ways with caches and other parts of modern memory hierarchies, making it challenging for programmers to write correct code.

One promising approach to address this challenge is to develop formal methods techniques to certify the absence of bugs in programs using non-volatile memory. This Dagstuhl Seminar brought together experts in non-volatile memory, relaxed memory, concurrency, and formal methods to explore the application of formal methods to programming with persistent memory. Since this subfield involves deep theoretical work, but is also very dependent on technological developments, the participants of the seminar were from a spectrum of backgrounds ranging from theory of verification to hardware specification, design, and testing.

The seminar included a series of talks and discussions, some of which were unplanned additions prompted by topics or misunderstandings identified during earlier parts of the seminar. The addition of these unplanned talks proved beneficial, adding a dynamic element to the event. We decided to forgo smaller break-out sessions based on the feeling that much of the seminar's value was in discussions that spanned from theoretical to practical, drawing on the full range of participants' expertise.

Several recurring themes arose in the talks and following discussions:

- **Correctness Criteria and Specifications for Persistent Objects**: A number of correctness criteria have been proposed for concurrent objects that persistently store data. Many of these definitions are adaptations of the classical notion of linearizability. However, we discussed ways in which these existing definitions can have surprising consequences when objects are implemented in the setting of weak memory. A related topic was the appropriate guarantees that transactional interfaces should provide, as certain strong guarantees may prevent efficient implementations.
- **Liveness**: Our discussions revealed a lack of consensus on assumptions that can be made from existing architectures concerning liveness properties, underscoring the need for further research in this area.
- **Future of NVM and Related Technologies**: NVM remains an emerging technology, and manufacturers continue to announce large changes in plans for future product lines. We discussed the ramifications of these changes and how techniques for the semantics and verification of certain forms of NVM might apply to other persistency models. Moreover, we identified the need for generic verification methods, which would lend themselves more easily to ongoing changes in the exact semantics of the underlying memory system. Indeed, several talks suggested modular approaches for verification, that, to some extent, take the memory model as an input. Related technologies, including Remote Direct Memory Access (RDMA) and Compute Express Link (CXL), were discussed, focusing on the appropriate abstractions and semantics of interfaces for these devices, and challenges with testing these devices.

We believe that these issues will be an important focus in research on formal methods for persistent memory in the future.

2 Table of Contents

3 Overview of Talks

3.1 Semantics of Remote Direct Memory Access

Guillaume Ambal (Imperial College London, GB)

Joint work of Guillaume Ambal, Brijesh Dongol, Haggai Eran, Vasileios Klimis, Ori Lahav, Azalea Raad

Remote direct memory access (RDMA) is a modern technology enabling networked machines to exchange information without involving the operating system of either side, and thus significantly speeding up data transfer in computer clusters. While RDMA is extensively used in practice and studied in various research papers, a formal underlying model specifying the allowed behaviours of concurrent RDMA programs running in modern multicore architectures is still missing. This paper aims to close this gap and provide semantic foundations of RDMA on x86-TSO machines. We propose three equivalent formal models, two operational models in different levels of abstraction and one declarative model, and prove that the three characterisations are equivalent. To gain confidence in the proposed semantics, the more concrete operational model has been reviewed by NVIDIA experts, a major vendor of RDMA systems, and we have empirically validated the declarative formalisation on various subtle litmus tests by extensive testing. We believe that this work is a necessary initial step for formally addressing RDMA-based systems by proposing language-level models, verifying their mapping to hardware, and developing reasoning techniques for concurrent RDMA programs.

3.2 Checking Liveness Properties under Weak Consistency (TSO as an Example)

Parosh Aziz Abdulla (Uppsala University, SE)

Joint work of Parosh Aziz Abdulla, Mohamed Faouzi Atig, Adwait Godbole, Shankaranarayanan Krishna, Mihir Vahanwala
Main reference Parosh Aziz Abdulla, Mohamed Faouzi Atig, Adwait Godbole, Shankaranarayanan Krishna, Mihir Vahanwala: "Overcoming Memory Weakness with Unified Fairness – Systematic Verification of Liveness in Weak Memory Models", in Proc. of the Computer Aided Verification – 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 13964, pp. 184–205, Springer, 2023.
URL https://doi.org//10.1007/978-3-031-37706-8_10

We consider the verification of liveness properties for concurrent programs running on weak memory models. To that end, we identify notions of fairness that preclude demonic non-determinism, are motivated by practical observations, and are amenable to algorithmic techniques. We provide both logical and stochastic definitions of our fairness notions, and prove that they are equivalent in the context of liveness verification. In particular, we show that our fairness allows us to reduce the liveness problem (repeated control state reachability) to the problem of simple control state reachability. We show that this is a general phenomenon by developing a uniform framework which serves as the formal foundation of our fairness definition, and can be instantiated to a wide landscape of memory models. These models include SC, TSO, PSO, (Strong/Weak) Release-Acquire, Strong Coherence, FIFO-consistency, and RMO.

3.3 Correctly Combining Concurrent and Persistent Transactional Memory

Brijesh Dongol (University of Surrey – Guildford, GB)

License Creative Commons BY 4.0 International license
© Brijesh Dongol
Joint work of Brijesh Dongol, Piotr Balcer, Ori Lahav, Azalea Raad, John Wickerson
Main reference Azalea Raad, Ori Lahav, John Wickerson, Piotr Balcer, Brijesh Dongol: "Intel PMDK Transactions: Specification, Validation and Concurrency (Extended Version)", CoRR, Vol. abs/2312.13828, 2023.
URL https://doi.org//10.48550/ARXIV.2312.13828

Software Transactional Memory (STM) is an extensively studied paradigm that provides an easy-to-use mechanism for thread safety and concurrency control. With the recent advent of byte-addressable persisent memory, a natural question to ask is whether STM systems can be adapted to support recoverability via failure atomicity. In this article, we answer this question by showing how STM can be easily integrated with Intel's Persistent Memory Development Kit (PMDK) transactional library (which we refer to as txPMDK) to obtain STM systems that are both concurrent and persistent. We demonstrate this approach using known STM systems, TML and NOrec, which when combined with txPMDK result in persistent STM systems, referred to as PMDK-TML and PMDK-NORec, respectively. However, it turns out that existing correctness criteria are insufficient for specifying the behaviour of txPMDK and our concurrent extensions. We therefore develop a new correctness criterion, dynamic durable opacity, that extends the previously defined notion of durable opacity with dynamic memory allocation. We provide a model of txPMDK that has been validated for accuracy with Intel developers, then show that this model satisfies dynamic durable opacity. Moreover, dynamic durable opacity supports concurrent transactions, thus we also use it to show correctness of both PMDK-TML and PMDK-NORec.

3.4 Utilizing Coherence for Persistence

Michal Friedman (ETH Zürich, CH)

License Creative Commons BY 4.0 International license
© Michal Friedman
Joint work of Richard Braun, Abishek Ramdas, Michal Friedman, Gustavo Alonso
Main reference Richard Braun, Abishek Ramdas, Michal Friedman, Gustavo Alonso: "PLayer: Expanding Coherence Protocol Stack with a Persistence Layer", in Proc. of the 1st Workshop on Disruptive Memory Systems, DIMES 2023, Koblenz, Germany, 23 October 2023, pp. 8–15, ACM, 2023.
URL https://doi.org//10.1145/3609308.3625270

Mechanisms to explicitly manage data persistence for non-volatile main memories are fundamental for the correctness and performance of modern systems. So far, however, most solutions are primarily based on software techniques. In this talk, I will describe a persistence layer on hardware, to support correct handling of persistent lock-free data structures. By exploiting cache-coherence messages, persistence can be transparently managed by the hardware, with minimal user intervention. We have experimented with a partial design on a Soft-CPU running on an FPGA to explore the idea and plan to further extend it into a real hardware implementation.

3.5 Some compositional semantics for shared memory: sequential consistency and release/acquire

Ohad Kammar (University of Edinburgh, GB)

Joint work of Yotam Dvir, Ohad Kammar, Ori Lahav
Main reference Yotam Dvir, Ohad Kammar, Ori Lahav: "A Denotational Approach to Release-Acquire Concurrency". ESOP 2024.

I motivated and presented recent results, joint with Dvir and Lahav, in the compositional/denotational semantics for shared state concurrency that is structurally similar to standard, but general, denotational semantics for sequential programs.

The desire for a uniform treatment of programming languages as standard features (let-binding, function abstraction, pattern matching) augmented with domain-specific features (mutable state, backtracking search, etc.) is as old as the discipline itself. I traced a specific thread starting with Landin's pragmatics and axiomatics, later realised by Plotkin's structural-operational semantics and, in the sequential case, extended to denotational semantics by Moggi, and later refined by Plotkin and Power. With this perspective, I outlined our recent Brookes-trace account for sequential consistent shared state using universal algebra and monads and its limitations. I also outlined our ongoing account for the non-relaxed atomic fragment of the release-acquire weak memory model and invited participants to follow-up informally.

These informal discussions covered: (a) Brookes's seminal work on trace semantics for sequentially consistent shared-state and some of Lahav's more recent results about its abstraction; and (b) a technical description of our release-acquire denotational semantics.

3.6 Programming Persistency Should Be Easy – but is it?

Jeehoon Kang (KAIST – Daejeon, KR)

Joint work of Kyeongmin Cho, Seungmin Jeon, Azalea Raad, Sung-Hwan Lee, Jeehoon Kang
Main reference Kyeongmin Cho, Seungmin Jeon, Azalea Raad, Jeehoon Kang: "Memento: A Framework for Detectable Recoverability in Persistent Memory", Proc. ACM Program. Lang., Vol. 7(PLDI), pp. 292–317, 2023.
URL https://doi.org//10.1145/3591232

Programming persistency poses two major challenges:

- Non-determinism: Persist instructions may be reordered. Consequently, an earlier write might be discarded while a later one is preserved in the event of a crash. Though this challenge has been somewhat mitigated by recent hardware changes, including (e)ADR and GPF.
- Recovery: In the event of a crash, it is crucial to recover pre-crash contexts to complete their execution.

Efficient and easy-to-implement crash recovery is still a promising area of research in programming persistency. Although prior works like NVTraverse (PLDI 2020) and Mirror (PLDI 2021) offer automatic translation of concurrent programs into persistent ones equipped with recovery code, their applicability is confined to simpler program forms. Memento (PLDI 2023) is applicable to a broader range of programs but misses some optimization opportunities related to DRAM caches and transactions.

What is the next generation technique for efficient and easy-to-implement crash recovery?

3.7 Challenges in Empirically Testing Memory Persistency Models

Vasileios Klimis (Queen Mary University of London, GB)

Joint work of Vasileios Klimis, Azalea Raad, Viktor Vafeiadis, John Wickerson, Alastair F. Donaldson

Memory persistency models provide a foundation for persistent programming by specifying which (and when) writes to non-volatile memory (NVM) become persistent. Memory persistency models for the Intel-x86 and Arm architectures have been formalised, but not empirically validated against real machines. Traditional validation methods used for memory *consistency* models do not straightforwardly apply because a test program cannot directly observe when its data has become persistent: it cannot distinguish between reading data from a volatile cache and from NVM. We investigate addressing this challenge using a commercial off-the-shelf device that intercepts data on the memory bus and logs all writes in the order they reach the memory. Using this technique we conducted a litmus-testing campaign aimed at empirically validating the persistency guarantees of Intel-x86 and Arm machines. We observed writes propagating to memory out of order, and took steps to build confidence that these observations were not merely artefacts of our testing setup. However, despite gaining high confidence in the trustworthiness of our observation method, our conclusions remain largely negative. We found that the Intel-x86 architecture is not amenable to our approach, and on consulting Intel engineers discovered that there are currently no reliable methods of validating their persistency guarantees. For Arm, we found that even a machine recommended to us by a persistency expert at Arm did not match the formal Arm persistency model, due to a loophole in the specification. Nevertheless, our investigation and results provide confidence that if Intel were to produce machines with more transparent persistency behaviour, or if Arm machines with proper persistency support were to become available, our approach would be valuable for empirically validating them against their specifications.

3.8 Automating Weak Memory Model Metatheory and Verification

Michalis Kokologiannakis (MPI-SWS – Kaiserslautern, DE)

Joint work of Michalis Kokologiannakis, Ori Lahav, Viktor Vafeiadis
Main reference Michalis Kokologiannakis, Ori Lahav, Viktor Vafeiadis: "Kater: Automating Weak Memory Model Metatheory and Consistency Checking", Proc. ACM Program. Lang., Vol. 7(POPL), pp. 544–572, 2023.
URL https://doi.org//10.1145/3571212

Weak memory consistency models capture the outcomes of concurrent programs that appear in practice and yet cannot be explained by thread interleavings. Such outcomes pose two major challenges to formal methods. First, establishing that a memory model satisfies its intended properties (e.g., supports a certain compilation scheme) is extremely error-prone: most proposed language models were initially broken and required multiple iterations to achieve soundness. Second, weak memory models make verification of concurrent programs much harder, as a result of which there are no scalable verification techniques beyond a few that target very simple models.

In this talk, I present solutions to both of these problems. First, I show that the relevant metatheory of weak memory models can be effectively decided and present Kater, a tool that can answer metatheoretic queries in a matter of seconds. Second, I present GenMC, the first scalable stateless model checker that is parametric in the choice of the memory model. To enhance the usability of GenMC, I demonstrate how Kater can be used to automate the porting of new memory models into GenMC, as well as how the state-space size of concurrent programs can be estimated.

3.9 Abstraction for Crash-Resilient Objects

Ori Lahav (Tel Aviv University, IL)

Joint work of Artem Khyzha, Ori Lahav
Main reference Artem Khyzha, Ori Lahav: "Abstraction for Crash-Resilient Objects", in Proc. of the Programming Languages and Systems – 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Lecture Notes in Computer Science, Vol. 13240, pp. 262–289, Springer, 2022.
URL https://doi.org//10.1007/978-3-030-99336-8_10

We study abstraction for crash-resilient concurrent objects using non-volatile memory (NVM). We develop a library-correctness criterion that is sound for ensuring contextual refinement in this setting, thus allowing clients to reason about library behaviors in terms of their abstract specifications, and library developers to verify their implementations against the specifications abstracting away from particular client programs. As a semantic foundation we employ a recent NVM model, called Persistent Sequential Consistency, and extend its language and operational semantics with useful specification constructs. The proposed correctness criterion accounts for NVM-related interactions between client and library code due to explicit persist instructions, and for calling policies enforced by libraries. We illustrate our approach on two implementations and specifications of simple persistent objects with different prototypical durability guarantees. Our results provide the first approach to formal compositional reasoning under NVM.

3.10 Fairness for load buffering memory models

Anton Podkopaev (JetBrains – Amsterdam, NL)

Joint work of Anton Podkopaev, Ori Lahav
Main reference Ori Lahav, Egor Namakonov, Jonas Oberhauser, Anton Podkopaev, Viktor Vafeiadis: "Making weak memory models fair", Proc. ACM Program. Lang., Vol. 5(OOPSLA), pp. 1–27, 2021.
URL https://doi.org//10.1145/3485475

Liveness properties, such as termination, of even the simplest shared-memory concurrent programs under sequential consistency typically require some fairness assumptions about the scheduler. Under weak memory models, we observe that the standard notions of thread fairness are insufficient, and an additional fairness property, which we call memory fairness, is needed.

In previous work (Lahav et al., 2021), we proposed a uniform definition for memory fairness that can be integrated into any declarative memory model enforcing acyclicity of the union of the program order (po) and the reads-from (rf) relations. For the well-known models, SC, x86-TSO, RA, and StrongCOH, that have equivalent operational and declarative presentations, we showed that our declarative memory fairness condition is equivalent to an intuitive model-specific operational notion of memory fairness, which requires the memory system to fairly execute its internal propagation steps.

Now, we raise a question on how memory models allowing poUrf-cycles, i.e., allowing load buffering, should be restricted to provide liveness guarantees. We showed that for Armv8 and Power memory models the memory fairness condition is enough to preserve our compilation correctness result (Podkopaev et al., 2019) for the Promising semantics (Kang et al., 2017), if the set of locations accessed by a program is finite (for both Armv8 and Power compilation targets) as well as a number of threads spawned by the program (needed only for Power). We also show that the compilation result is preserved for a restricted version of the Promising semantics, which we call Promising_fair. The restriction guarantees that any promise made by a thread is eventually fulfilled.

3.11 DARTAGNAN: One tool for all models

Hernán Ponce de León (Huawei Technologies – München, DE)

Main reference Thomas Haas, Roland Meyer, Hernán Ponce de León: "CAAT: consistency as a theory", Proc. ACM Program. Lang., Vol. 6(OOPSLA2), pp. 114–144, 2022.
URL https://doi.org//10.1145/3563292

The notion of consistency exists in several communities within computer science: programming languages, CPUs, databases, GPUs, non-volatile memory, etc. Despite the different application domains, the foundations are not that different. CAT is a domain specific language that allows to formalize different notions of consistency. Having a unified DSL facilitates the building of verification technology that works across all the application domains.

In this talk, I present how to encode program correctness with respect to a given CAT model using SMT based Bounded Model Checking. I show how to achieve scalability (i.e., we can verify real code coming from the Linux kernel) based on two key ideas. The first contribution (OOPSLA'22) is based in interpreting consistency as an SMT theory. This allows to simplify the part of the encoding that needs to be handled by the SAT solver and moves the hardness of encoding consistency into the theory solver where we can use domain specific knowledge to improve solving time. The second contribution (OOPSLA'23) is a static analysis of the consistency model which allows to propagate information in two directions: bottom-up from base relations to derived relations, and top-down from consistency axioms over derived relations to base relations.

3.12 Towards a formal specification of the Intel Architecture

Alastair Reid (Intel – London, GB)

Formal specifications of CPU architectures are useful for formally verifying hardware and software; for verifying compilers; for discovering compiler peepholes; and for analyzing programs for security issues. We are creating a formal specification of the Intel Architecture (aka "x86") with the intention that the specification is complete (e.g., able to boot an OS or run SGX code); correct (e.g., validated using the same tests that processors are tested with); readable (e.g., suitable for use in the Intel Software Developer's Manual); available (e.g., on GitHub and licensed under a suitably permissive license); usable (tools are available and can be used as the basis of other tools); and used by our customers, 3rd party developers, the open source community, academic researchers, etc. (we will help users understand what the spec can do and how to use and adapt the tools to enable a diverse set of uses).

3.13 System and Failure Models Matter

Michael Scott (University of Rochester, US)

Those of us here today are intensely interested in formal models of concurrency and persistence. In addition to posing challenging problems from an intellectual perspective, these models need to capture key aspects of real-world systems if they are to have an impact on practice. In this talk I briefly survey (my opinions regarding) the space of interesting models. In particular, we can consider

- system models, which capture the hardware and software architecture on which our algorithms run;
- persistency models, which capture instruction-level ordering and the reads-see-writes relationship in the presence of crashes; and
- failure models, which consider what exactly may fail, what resumes, and how.

Among other things, I suggest that:

- Independent thread failures (esp. with threads that are recovered and continue) have few if any real-world analogues, and should be pursued with caution.
- Real-world systems are likely to have much more NVM than DRAM, so work that mirrors all persistent data in DRAM should be pursued with caution.
- Hardware designers have considerable motivation to develop persistent caches, so work that assumes that cached data will always be transient should be pursued with caution.
- The world is still looking for an NVM killer app.

3.14 Transactional Semantics with Zombies

Michael Scott (University of Rochester, US)

Main reference Michael Scot: "Transactional Semantics with Zombies" Invited presentation, 6th Workshop on the Theory of Transactional Memory (WTTM), Paris, France, July 2014.

Different formal models of transactional memory are required at different levels of the system stack. This paper focuses on the run-time level, where the semantics of individual operations (start, read, write, try-commit) govern the interactions between the compiler and the TM system. For sandboxing TM systems, which allow a doomed transaction (a "zombie") to continue for some time beyond an inconsistent read, run-time–level semantics cannot be captured by opacity as currently defined: we need a formal model of zombie execution.

3.15 Specifying and Verifying Persistent Libraries

Léo Stefanesco (MPI-SWS – Kaiserslautern, DE)

Joint work of Léo Stefanesco, Azalea Raad, Viktor Vafeiadis
Main reference Léo Stefanesco, Azalea Raad, Viktor Vafeiadis: "Specifying and Verifying Persistent Libraries", CoRR, Vol. abs/2306.01614, 2023.
URL https://doi.org//10.48550/ARXIV.2306.01614

We present a general framework for specifying and verifying persistent libraries, that is, libraries of data structures that provide some persistency guarantees upon a failure of the machine they are executing on. Our framework enables modular reasoning about the correctness of individual libraries (horizontal and vertical compositionality) and is general enough to encompass all existing persistent library specifications ranging from hardware architectural specifications to correctness conditions such as durable linearizability. As case studies, we specify the FliT and Mirror libraries, verify their implementations over Px86, and use them to build higher-level durably linearizable libraries, all within our framework. We also specify and verify a persistent transaction library that highlights some of the technical challenges which are specific to persistent memory compared to weak memory and how they are handled by our framework.

3.16 A Type System for Intermittent Computing

Milijana Surbatovich (Carnegie Mellon University – Pittsburgh, US)

Joint work of Milijana Surbatovich, Naomi Spargo, Limin Jia, Brandon Lucia
Main reference Milijana Surbatovich, Naomi Spargo, Limin Jia, Brandon Lucia: "A Type System for Safe Intermittent Computing", Proc. ACM Program. Lang., Vol. 7(PLDI), pp. 736–760, 2023.
URL https://doi.org//10.1145/3591250

Batteryless, energy harvesting devices (EHDs) enable computing in environments that are too remote or inaccessible to support battery maintenance, benefiting application domains like disaster monitoring, health wearables, and smart civil and agricultural infrastructure. Instead of relying on a battery, these devices harvest all energy they need from their surroundings. Because the target application domains have high assurance requirements, computation on EHDs should be correct; unfortunately, harvested energy is typically too weak to power a device continuously, resulting in frequent power failures that break software and systems designed to run on continuous power. The field of intermittent computing seeks to overcome the correctness and programmability challenges introduced by these power failures but has historically relied on ad-hoc correctness reasoning that provides no guarantees.

This talk motivates the need for formal methods research for designing correct intermittent systems, highlighting the importance of modularity and abstraction in both formalism and system design. It then presents Curricle, an information-flow type system for reasoning about safe intermittent execution that gives programmers more control and provides natural layering between the application and runtime system levels of the stack. The talk concludes by discussing open problems in the intermittent computing field.

3.17 Separation Logic for Concurrent, Crash-Safe Systems

Joseph Tassarotti (New York University, US)

Joint work of Tej Chajed, Joseph Tassarotti, Mark Theng, Ralf Jung, M. Frans Kaashoek, Nickolai Zeldovich
Main reference Tej Chajed, Joseph Tassarotti, Mark Theng, Ralf Jung, M. Frans Kaashoek, Nickolai Zeldovich: "GoJournal: a verified, concurrent, crash-safe journaling system", in Proc. of the 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021, pp. 423–439, USENIX Association, 2021.
URL https://www.usenix.org/conference/osdi21/presentation/chajed

Storage systems, such as databases and file systems, often have concurrent implementations. These systems are expected to be crash-safe, meaning that they should be able to recover from failures caused by power loss. However, achieving crash-safety is difficult because programmers must consider many potential interleavings of threads as well as the possibility of interruption from crashes at any point. Perennial is a separation logic framework for formally verifying crash safety of concurrent systems. This talk describes the core reasoning principles of Perennial and our experience using Perennial to verify GoTxn, a concurrent transaction system.

3.18 Persistent Scripting

Haris Volos (University of Cyprus – Nicosia, CY)

Joint work of Zi Fan Tan, Jianan Li, Terence Kelly, Arnold Robbins, Haris Volos

Persistent scripting brings the benefits of persistent memory programming to high-level interpreted languages. More importantly, it brings the convenience and programmer productivity of scripting to persistent memory programming. We have integrated a novel generic persistent memory allocator into a popular scripting language interpreter, which now exposes a simple and intuitive persistence interface: A flag notifies the interpreter that a script's variables reside in a persistent heap in a specified file. The interpreter begins script execution with all variables in the persistent heap ready for immediate use. New variables defined by the running script are allocated on the persistent heap and are thus available to subsequent executions. Scripts themselves are unmodified and persistent heaps may be shared freely between unrelated scripts.

3.19 Verifying the persistency library FliT

Heike Wehrheim (Universität Oldenburg, DE)

Joint work of Stefan Bodenmüller, John Derrick, Brijesh Dongol, Gerhard Schellhorn, Heike Wehrheim
Main reference Stefan Bodenmüller, John Derrick, Brijesh Dongol, Gerhard Schellhorn, Heike Wehrheim: "A Fully Verified Persistency Library", in Proc. of the Verification, Model Checking, and Abstract Interpretation – 25th International Conference, VMCAI 2024, London, United Kingdom, January 15-16, 2024, Proceedings, Part II, Lecture Notes in Computer Science, Vol. 14500, pp. 26–47, Springer, 2024.
URL https://doi.org//10.1007/978-3-031-50521-8_2

Non-volatile memory (NVM) technologies offer DRAM-like speeds with the added benefit of failure resilience. However, developing concurrent programs for NVM can be challenging since programmers must consider both inter-thread synchronisation and durability aspects at the same time. To alleviate this, libraries such as FliT have been developed to manage transformations to durability, allowing a linearizable concurrent object to be converted into a durably linearizable one with minimal programmer effort. However, a formal proof of correctness for FliT is missing, and standard proof techniques for durable linearizability are challenging to apply since FliT itself is not durably linearizable.

In this talk, I report on our work on showing correctness of transformations to durability. First, we develop an abstract persistency library (called PLib) that operationally characterises transformations to durability and we prove its correctness. Second, we show correctness of the library FliT by proving that FliT refines PLib under the realistic Px86 memory model, i.e., the persistent version of TSO memory model implemented by Intel architectures. The proof of refinement between FliT and PLib has been mechanised within the theorem prover KIV. Taken together, these proofs guarantee that FliT is also sound wrt transformations to durability.

Participants

- Guillaume Ambal
Imperial College London, GB
- Parosh Aziz Abdulla
Uppsala University, SE
- Mark Batty
University of Kent – Canterbury, GB
- Michael D. Bond
Ohio State University – Columbus, US
- Ahmed Bouajjani
Université Paris Cité, FR
- Paulo Emílio de Vilhena
Imperial College London, GB
- Brijesh Dongol
University of Surrey – Guildford, GB
- Michal Friedman
ETH Zürich, CH
- Ohad Kammar
University of Edinburgh, GB
- Jeehoon Kang
KAIST – Daejeon, KR
- Vasileios Klimis
Queen Mary University of London, GB
- Michalis Kokologiannakis
MPI-SWS – Kaiserslautern, DE
- Ori Lahav
Tel Aviv University, IL
- Anton Podkopaev
JetBrains – Amsterdam, NL
- Hernán Ponce de León
Huawei Technologies – München, DE
- Azalea Raad
Imperial College London, GB
- Alastair Reid
Intel – London, GB
- Michael Scott
University of Rochester, US
- Léo Stefanesco
MPI-SWS – Kaiserslautern, DE
- Milijana Surbatovich
Carnegie Mellon University – Pittsburgh, US
- Joseph Tassarotti
New York University, US
- Viktor Vafeiadis
MPI-SWS – Kaiserslautern, DE
- Haris Volos
University of Cyprus – Nicosia, CY
- Heike Wehrheim
Universität Oldenburg, DE