Report from Dagstuhl Seminar 16471

Concurrency with Weak Memory Models: Semantics, Languages, Compilation, Verification, Static Analysis, and Synthesis

Edited by

Jade Alglave[1], Patrick Cousot[2], and Caterina Urban[3]

1 **University College London, GB,** j.alglave@ucl.ac.uk
2 **New York University, US,** pcousot@cims.nyu.edu
3 **ETH Zürich, CH,** caterina.urban@inf.ethz.ch

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 16471 "Concurrency with Weak Memory Models: Semantics, Languages, Compilation, Verification, Static Analysis, and Synthesis". The aim of the seminar was to bring together people from various horizons, including theoreticians and verification practitioners as well as hardware vendors, in order to set up a long-term research program to design formal methods and develop tools ensuring the correctness of concurrent programs on modern multi-processor architectures.

Seminar November 20–25, 2016 – http://www.dagstuhl.de/16471
1998 ACM Subject Classification D.2.4 Software/Program Verification, D.3.4 Processors, F.3.2 Semantics of Programming Languages
Keywords and phrases Compilation, Computer Memory, Concurrency, Memory Barrier, Memory Ordering, Micro-Architecture, Multiprocessor, Out-of-Order Execution, Parallelism, Program Synthesis, Programming Language, Semantics, Static Analysis, Verification, Weak Memory Model
Digital Object Identifier 10.4230/DagRep.6.11.108

1 Executive Summary

Jade Alglave
Patrick Cousot

In the last decade, research on weak memory has focussed on modeling accurately and precisely existing systems such as hardware chips. These laudable efforts have led to definitions of models such as IBM Power, Intel x86, Nvidia GPUs and others.

Now that we have faithful models, and know how to write others if need be, we can focus on how to use these models for verification, for example to assess the correctness of concurrent programs.

The goal of our seminar was to discuss how to get there. To do so, we gathered people from various horizons: hardware vendors, theoreticians, verification practitioners and hackers. We asked them what issues they are facing, and what tools they would need to help them tackle said issues.

The first day was dedicated to theory. We had overviews of classic semanticists tools such as event structures, message sequence charts, and pomsets. The remaining days were

Concurrency with Weak Memory Models: Semantics, Languages, Compilation, Verification, Static Analysis, and Synthesis, *Dagstuhl Reports*, Vol. 6, Issue 11, pp. 108–128
Editors: Jade Alglave, Patrick Cousot, and Caterina Urban
DAGSTUHL REPORTS Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

mostly dedicated to models and verification practices, whether from a user point of view, or a designer point of view. We chose to close the days early, so that our guests would have ample time to come back to an interesting point they had heard during one of the talks, or engage in deep discussions. The feedback we got was quite positive, in that the seminar help spark discussions with, for example, a PhD student in concurrency theory, and a verification practitioner from ARM.

2 Table of Contents

Executive Summary

Overview of Talks

3 Overview of Talks

3.1 Robustness against Consistency Models with Atomic Visibility

Giovanni Tito Bernardi (University Paris-Diderot, FR)

Joint work of Giovanni Bernardi, Alexey Gotsman
Main reference G. Bernardi, A. Gotsman, "Robustness against Consistency Models with Atomic Visibility", in Proc. of the 27th Int'l Conf. on Concurrency Theory (CONCUR 2016), LIPIcs, Vol. 59, pp. 7:1–7:15, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.
URL http://dx.doi.org/10.4230/LIPIcs.CONCUR.2016.7

To achieve scalability, modern Internet services often rely on distributed databases with consistency models for transactions weaker than serializability. At present, application programmers often lack techniques to ensure that the weakness of these consistency models does not violate application correctness. In this talk I will present criteria to check whether applications that rely on a database providing only weak consistency are robust, i.e., behave as if they used a database providing serializability, and I will focus on a consistency model called Parallel Snapshot Isolation. The results I will outline handle systematically and uniformly several recently proposed weak consistency models, as well as a mechanism for strengthening consistency in parts of an application.

3.2 Transactions on Mergeable Objects in Shared-Memory

Annette Bieniusa (TU Kaiserslautern, DE)

Joint work of Deepthi Devaki Akkoorath, Annette Bieniusa
Main reference D. D. Akkoorath, A. Bieniusa, "Transactions on Mergeable Objects", in Proc. of the 13th Asian Symp. on Programming Languages and Systems (APLAS 2015), LNCS, Vol. 9458, pp. 427–444, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-319-26529-2_23

Under high contention, serializability for transactions results in frequent aborts. This limits possible parallelism and results in performance degradation. In this talk, we introduce a new transactional semantics, Mergeable Transactions, which allows concurrent transactions on the same objects to execute in parallel. Instead of aborting and re-executing, the conflicting updates on shared objects are merged using type specific semantics. We show that mergeable transactions outperform serializable transactions under high contention workloads.

3.3 New Lace is a Program Logic for Weak Memory (Probably)

Richard Bornat (Middlesex University – London, GB)

Joint work of Richard Bornat, Jade Alglave, Matthew J. Parkinson
Main reference R. Bornat, J. Alglave, M. J. Parkinson, "New Lace and Arsenic: Adventures in Weak Memory with a Program Logic", arXiv:1512.01416v2 [cs.LO], 2015.
URL http://arxiv.org/abs/1512.01416v2

It is possible to reason about weak-memory executions of litmus tests (but not yet about synchronized assignment) using a version of rely/guarantee. Constraints between commands and/or control expressions control order of elaboration (local execution) and propagation of writes. The logic is driven by the temporal modality 'since' and some specialized modalities based on it.

The logic has only a weak grasp of causality (treated by auxiliary variables, as usual in Owicki-Gries logics). It has some surprising rules dealing with stability: five or six different kinds of stability. It has a proof-checker (Arsenic, available on GitHub); the proof-checker is needed for even smallish proofs, which shows that proofs in the logic are far too complicated.

3.4 A Denotational Framework for Weak Memory Conucrrency

Stephen Brookes (Carnegie Mellon University – Pittsburgh, US)

We present a denotational semantic framework for compositional reasoning about shared-memory parallel programs, assuming a form of weak memory. Traditional denotational models of shared-memory programs are trace-based, assume sequential consistency, and use global states and interleaving, rendering them poorly suited for expressing weak memory behavior. Instead we abandon sequential consistency and embrace "true" concurrency: a program denotes a set of pomsets (partially ordered multi-sets) of actions. Rather than global states we use footprints, built from "local" states. This framework is intended to offer an alternative to execution graphs, widely used in operational/axiomatic formalizations of weak memory. An execution graph represents the behavior of a complete program, all threads known in advance, running without external interference. The axiomatic method requires construction of various auxiliary relational edges (happens-before, reads-from, etc) constrained to satisfy a battery of axioms. These graph-based methods are inherently non-compositional, relying on knowledge of the entire program structure and assuming that execution takes place with no interference. A denotational semantics is by its very nature compositional, allowing us to take account of interference in a natural manner; and we can derive analogues of the relevant auxiliary relations automatically from the structure of pomset executions.

3.5 Weak Memory using Event Structures

Simon Castellan (ENS – Lyon, FR)

In this talk, I will introduce a methodology to model weak memory using event structures. The model is compositional and neatly separate thread & storage semantics. Moreover it comes from recent game semantics advances using causal models. The game semantics aspects allows to define the model by simply defining a few key strategies with higher-order type. In this talk, we show how to interpret the TSO model using event structures in a compositional way.

3.6 Analysing Snapshot Isolation

Andrea Cerone (Imperial College London, GB)

Joint work of Andrea Cerone, Alexey Gotsman
Main reference A. Cerone, A. Gotsman, "Analysing Snapshot Isolation", in Proc. of the 2016 ACM Symp. on Principles of Distributed Computing (PODC'16), pp. 55–64, ACM, 2016.
URL http://dx.doi.org/10.1145/2933057.2933096

Snapshot isolation (SI) is a widely used consistency model for transaction processing, implemented by most major databases and some of transactional memory systems. Unfortunately, its classical definition is given in a low-level operational way, by an idealized concurrency-control algorithm, and this complicates reasoning about the behavior of applications running under SI. We give an alternative specification to SI that characterizes it in terms of transactional dependency graphs of Adya et al., generalizing serialization graphs. Unlike previous work, our characterization does not require adding additional information to dependency graphs about start and commit points of transactions.

3.7 Game Semantics based on Event Structures

Pierre Clairambault (ENS – Lyon, FR)

Joint work of Simon Castellan, Pierre Clairambault, Silvain Rideau, Glynn Winskel
Main reference S. Castellan, P. Clairambault, S. Rideau, G. Winskel, "Games and Strategies as Event structures", arXiv:1604.04390v3 [math.LO], 2016.
URL https://arxiv.org/abs/1604.04390v3

Games are common objects in theoretical computer science, used in particular to model open systems: indeed, the dynamics of an open system can be regarded as a game between two players, one playing for the system and the other for the environment. In program semantics, the same idea yields a general methodology ("Game Semantics") for giving syntax-free representations of the execution in a compositional manner, for various high-level programming languages with complex control structure. In this talk, I will give an introduction to recent work on game semantics based on event structures, with a focus on the semantics of shared state concurrency. This served as the basis and inspiration for Simon Castellan's event structure semantics for weak memory, presented in a separate talk.

3.8 Proof of Mutual-Exclusion and Non-Starvation of a Program: PostgreSQL

Patrick Cousot (New York University, US) and Jade Alglave (University College London, GB)

Joint work of Jade Alglave, Patrick Cousot
Main reference J. Alglave, P. Cousot, "Ogre and Pythia: an Invariance Proof Method for Weak Consistency Models", in Proc. of the 44th ACM SIGPLAN Symp. on Principles of Programming Languages (POPL'17), pp. 3–18, ACM, 2017.
URL http://dx.doi.org/10.1145/3009837.3009883

Proof of mutual-exclusion and non-starvation of a program: PostgreSQL, Jade Alglave and Patrick Cousot.

Using the parallel program invariance proof method of Alglave and Cousot (POPL 2017), we prove the mutual exclusion property of the PostgreSQL program. The weakest memory model necessary and sufficient for this mutual exclusion property to hold is extracted from the proof by calculational design.

The invariance proof method allows us to reason on any set of executions as defined by a set of read-from relations (each read-from relation uniquely determining a single execution trace, if any). Thanks to this property, we can use the inductive invariant to prove non-starvation. We prove that any execution that starves is impossible, either because this proof method cannot satisfy the verification conditions and so (by soundness of the proof method) is not a possible execution of the program, or by disallowing this execution thanks to labeled fences (which does not change the invariance proof and which effect is defined in cat), or thanks to properties of hardware architectures (such as no read of a future write beyond a cut) not expressible in the current version of cat.

3.9 Modeling and Analysis of Remote Memory Access Programming

Andrei Marian Dan (ETH Zürich, CH)

Joint work of Andrei Marian Dan, Patrick Lam, Torsten Hoefler, Martin Vechev
Main reference A. M. Dan, P. Lam, T. Hoefler, M. Vechev, "Modeling and Aanalysis of Remote Memory Access Programming", in Proc. of the 2016 ACM SIGPLAN Int'l Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'16), pp. 129-144, ACM, 2016.
URL http://dx.doi.org/10.1145/2983990.2984033

Recent advances in networking hardware have led to a new generation of Remote Memory Access (RMA) networks in which processors from different machines can communicate directly, bypassing the operating system and allowing higher performance. Researchers and practitioners have proposed libraries and programming models for RMA to enable the development of applications running on these networks, However, the memory models implied by these RMA libraries and languages are often loosely specified, poorly understood, and differ depending on the underlying network architecture and other factors. Hence, it is difficult to precisely reason about the semantics of RMA programs or how changes in the network architecture affect them. We address this problem with the following contributions: (i) a coreRMA language which serves as a common foundation, formalizing the essential characteristics of RMA programming; (ii) complete axiomatic semantics for that language; (iii) integration of our semantics with an existing constraint solver, enabling us to exhaustively generate core- RMA programs (litmus tests) up to a specified bound and check whether the tests satisfy their specification; and (iv) extensive validation of our semantics on real-world RMA systems. We generated and ran 7,441 litmus tests using each of the low-level RMA network APIs: DMAPP, VPI Verbs, and Portals 4. Our results confirmed that our model success- fully captures behaviors exhibited by these networks. More- over, we found RMA programs that behave inconsistently with existing documentation, confirmed by network experts. Our work provides an important step towards understand- ing existing RMA networks, thus influencing the design of future RMA interfaces and hardware.

3.10 Formalising the ARM Memory Model ... Again

Will Deacon (ARM Ltd. – Cambridge, GB)

Recent work within the ARM architecture group has led to the development of a formalization of the ARMv8 weakly consistent memory model using herd and 'cat'. Whilst this model can act as an invaluable tool when considering concurrent applications in userspace, its interactions with the system architecture are unclear and appear to be inexpressible with the current litmus test methodology. This talk will introduce the new model and highlight some of the challenges faced when integrating it with the broader architecture.

3.11 A Plea for Industrial-Strength Formal Methods for Concurrent Software

David Delmas (Airbus S.A.S. – Toulouse, FR)

Joint work of V. Brégeon, E. Cavailles, D. Delmas, V. Jégu, A. Miné, S. Sauvant, B. Triquet

Verification activities are liable for more than half of the overall effort in the development of critical avionics software. Therefore, (semi-) automatic formal verification techniques are increasingly used to improve industrial efficiency, while preserving safety. As required by the DO-333 formal method technical supplement to the DO-178C standard for avionics software development, such techniques must be sound, and associate tools have to undergo stringent qualification processes.

However, while most existing techniques and tools focus on sequential or synchronous software, an increasing share of embedded systems is being developed in asynchronous software, to save on cost, weight, and resources. AstréeA, an extension of the Astrée run-time error analyzer, is one of the very few sound static analyzers which may be used for such asynchronous software.

So far, asynchronous avionics software was mostly running on single-core platforms with real-time scheduling. Nonetheless, multi-core avionics architectures are currently being considered. There is interest for wait-free/lock-free message passing algorithms, to support ARINC 653 sampling and queuing schemes. At implementation level, there is a need for a sustainable way to ensure correctness via a minimal set of fences, without impairing hard real-time performance targets. At verification level, there is a need for sound techniques guaranteeing functional correctness of source and compiled programs. An issue is that the CompCert compiler is only certified for sequential executions. Finally, there is a need for a sound approach to timing analysis with complex multi-core processors.

3.12 Embedding Transactions in Weak Memory Models

Stephan Diestelhorst (ARM Ltd. – Cambridge, GB)

Joint work of Stephan Diestelhorst, Matt Horsnell, Grigorios Magklis

Hardware Transactional Memory has been proposed as a higher-level memory primitive that can improve performance of and simplify parallel programming. The baseline premise of HTM is simple: transactions behave as if they are executing in isolation, despite them executing concurrently.

Recent implementations and architectures, however, need to embed this core principle into a memory model for non-transactional accesses.

Especially with a weak non-tx memory model such as ARM, embedding the strong TM semantics leaves ample room for "impedance matching" of the different strengths. Together with the combinatorial explosion of options to add transactions to well-known litmus tests, a mechanized model for experimenting with semantics is prudent.

In my talk, I will show our work in progress of using the CAT language for formalizing HTM, and also informally present some of the challenges associated with the interplay of transactional and non-transactional accesses.

3.13 A Discrete Model of Concurrent Program Execution

Charles Anthony Richard Hoare (Microsoft Research UK – Cambridge, GB)

Joint work of Jade Alglave, Charles Anthony Richard Hoare, Peter O'Hearn, Stephan van Staden, Viktor Vafeiadis, Ian Wehrman, John Wickerson

A two-dimensional discrete (non-metric) geometry is proposed for recording the trace of execution of a concurrent program that has been expressed in an object-oriented high-level programming language. From this is derived an algebraic semantics for the programming language, and a logic for reasoning about correctness of its semantics, and an operational semantics for implementing it.

3.14 (A New Methodology for) Inductive Verification of Message-Passing Programs

Bernhard Kragl (IST Austria – Klosterneuburg, AT)

Joint work of Bernhard Kragl, Shaz Qadeer

Designing and implementing distributed systems is a hard and challenging problem. A major obstacle is to manage the complexity of the sheer number of behaviors of a system due to, e.g., nondeterministic scheduling and unreliable message delivery. This complexity transfers to the amount of annotations required in correctness proofs and significantly hinders the adoption of verification technology.

In this talk we present a methodology developed atop the CIVL verification system [CAV'15] (originally designed for shared-memory programs) to simplify the construction of correctness proofs for message-passing programs. The central theme is to establish conditions that allow message handlers to be inlined at message sends and thus enable sequential reasoning, which further eliminates complicated case distinctions in the necessary invariants. For example, in our proof of a two-phase commit protocol we do not need to state complex conditions on the history of the system or the current network state (i.e., messages in delivery).

3.15 A Promising Semantics for Relaxed-Memory Concurrency

Ori Lahav (MPI-SWS – Kaiserslautern, DE)

Joint work of Derek Dreyer, Chung-Kil Hur, Jeehoon Kang, Ori Lahav, Viktor Vafeiadis
Main reference J. Kang, C.-K. Hur, O. Lahav, V. Vafeiadis, D. Dreyer, "A Promising Semantics for Relaxed-Memory Concurrency", in Proc. of the 44th ACM SIGPLAN Symp. on Principles of Programming Languages (POPL'17), pp. 175–189, ACM, 2017.
URL http://dx.doi.org/10.1145/3009837.3009850

Despite many years of research, it has proven very difficult to develop a memory model for concurrent programming languages that adequately balances the conflicting desiderata of programmers, compilers, and hardware. In this talk, we present the first relaxed memory model that (1) accounts for a broad spectrum of features from the C++11 concurrency model, (2) is implementable, in the sense that it provably validates many standard compiler optimizations and reorderings, as well as standard compilation schemes to x86-TSO and Power, (3) justifies simple invariant-based reasoning, thus demonstrating the absence of bad "out-of-thin-air" behaviors, (4) supports "DRF" guarantees, ensuring that programmers who use sufficient synchronization need not understand the full complexities of relaxed-memory semantics, and (5) defines the semantics of racy programs without relying on undefined behaviors, which is a prerequisite for applicability to type-safe languages like Java.

The key novel idea behind our model is the notion of promises: a thread may promise to execute a write in the future, thus enabling other threads to read from that write out of order. Crucially, to prevent out-of-thin-air behaviors, a promise step requires a thread-local certification that it will be possible to execute the promised write even in the absence of the promise. To establish confidence in our model, we have formalized most of our key results in Coq.

3.16 Automatic Synthesis of Comprehensive Litmus Test Suites

Daniel Lustig (NVIDIA Corp. – Santa Clara, US)

Joint work of Olivier Giroux, Daniel Lustig, Alexandros Papakonstantinou, Andrew Wright
Main reference D. Lustig, A. Wright, A. Papakonstantinou, O. Giroux, "Automatic Synthesis of Comprehensive Memory Model Litmus Test Suites", Architectural Support for Programming Languages and Operating Systems (ASPLOS'17), 2017.

Litmus tests are the basic units of testing weak memory models. Most memory model analysis infrastructures and testing suites make heavy use of litmus tests as the basic units of testing and understanding. The success of such techniques requires that the suites of litmus tests be comprehensive: that they cover every obvious and obscure corner of the memory model and/or of its implementation. However, most litmus test suites today are generated through some combination of manual effort and randomization, and this leaves them prone to human error and incompleteness.

We present a methodology for synthesizing comprehensive litmus test suites directly from the memory model specification. By construction, these suites contain all tests satisfying a minimality criterion: that no synchronization mechanism in the test can be weakened without causing new behaviors to become observable. We formalize this notion using the Alloy modeling language, and we apply it to a number of existing and newly-proposed memory models. Our results show not only that this synthesis technique can automatically reproduce all manually-generated tests from existing suites, but also that it discovers new tests that are not as well studied.

3.17 C11 Compiler Mappings: Exploration, Verification, and Counterexamples

Yatin Manerkar (Princeton University, US)

Joint work of Daniel Lustig. Yatin Manerkar, Margaret Martonosi, Michael Pellauer, Caroline Trippel
Main reference Y. A. Manerkar, C. Trippel, D. Lustig, M. Pellauer, M. Martonosi, "Counterexamples and Proof Loophole for the C/C++ to POWER and ARMv7 Trailing-Sync Compiler Mappings", arXiv:1611.01507v2 [cs.PL], 2016
URL https://arxiv.org/abs/1611.01507v2

C and C++ atomic operations get mapped down to individual instructions or combinations of instructions by compilers, depending on the ordering guarantees and synchronization instructions provided by the underlying architecture. These compiler mappings must uphold the ordering guarantees provided by C/C++ atomics or the compiled program will not behave according to the C/C++ memory model. In this talk I discuss a counterexample we discovered to the well-known trailing-sync compiler mappings for the Power and ARMv7 architectures that were previously thought to be proven correct. I also discuss the loophole in the proof of the mappings that allowed the incorrect mappings to be proven correct, as well as a few optimization-related bugs that I discovered in the IBM XL C++ compiler's implementation of C++ atomics.

3.18 Taming CAT

Luc Maranget (INRIA – Paris, FR)

Joint work of Jade Alglave, Luc Maranget
URL http://diy.inria.fr

In this demo-talk, a Lamport style model of Sequential Consistency is created live in CAT. CAT is the Domain Specific Language used by the memory model simulator herd7 to describe and execute shared memory models.

See http://diy.inria.fr for software and documentation.

3.19 Linux-Kernel Memory Ordering: Help Arrives At Last!

Paul McKenney (IBM – Beaverton, US)

Joint work of Jade Alglave, Luc Maraget, Paul McKenney, Andrea Parri, Alan Stern

It has been said that Documentation/memory-barriers.txt can be used to frighten small children [1], and perhaps this is true. However, it is woefully inefficient. After all, there are a very large number of children in this world, and it would take a huge amount of time and effort to read it to all of them.

This situation clearly calls out for automation, which has been developed over the past two years. An automated tool takes short fragments of C code as input, along with an assertion, and carries out the axiomatic equivalent of a full state-space search to determine whether the assertion always, sometimes, or never triggers. This talk will describe this tool and give a short demonstration of its capabilities.

To the best of our knowledge, this is the first realistic Linux-kernel memory model, and the first memory model of any kind incorporating a realistic model of RCU.

References

1 Mel Gorman. [PATCH 11/18] mm: fix TLB flush race between migration, and change_protection_range. Linux Kernel Mailing List, Hillsboro, OR, USA, 2013

3.20 Portability Analysis for Axiomatic Memory Models

Roland Meyer (TU Braunschweig, DE)

Joint work of Florian Furbach, Keijo Heljanko, Roland Meyer, Hernán Ponce de León

We present PORTHOS, the first tool that discovers porting bugs in performance-critical code. PORTHOS takes as input a program, the memory model of the source architecture for which the program has been developed, and the memory model of the targeted architecture. If the code is not portable, PORTHOS finds a porting bug in the form of an unexpected execution – an execution that is consistent with the target but inconsistent with the source memory model. Technically, PORTHOS implements a bounded model checking method that reduces portability analysis to the satisfiability modulo theories (SMT) problem with integer difference logic. There are two problems in the reduction that are unique to portability and that we present novel and efficient solutions for. First, the formulation of portability contains a quantifier alternation (consistent + inconsistent). We encode inconsistency as an existential query. Second, the memory models may contain recursive definitions. We compute the corresponding least fixed points efficiently in SMT. Interestingly, we are able to prove that our execution-based notion of portability is the most liberal one that admits an efficient algorithmic analysis: for state-based portability, a polynomial SAT encoding cannot exist. Experimentally, we applied PORTHOS to a number of case studies. It is able to check portability of non-trivial programs between interesting architectures. Notably, we present the first algorithmic analysis of portability from TSO to Power.

3.21 Hazard Pointers: C++ Memory Ordering Issues

Maged M. Michael (Facebook – New York, US)

In this talk I review the hazard pointers method with focus on memory ordering issues for the main access patterns under the C++ memory consistency model. This is in the context of an ongoing effort at the C++ standard committee to add hazard pointers to the standard library.

One of the challenges in determining correct memory ordering for a hazard pointers library implementation is that main access patterns include user code that may use weak memory order specifiers. This makes using default sequentially consistent memory accesses insufficient.

I use the herd memory model simulator to find sufficient memory ordering options for preventing incorrect execution patterns. The conclusions from the experience are that support is needed for read-modify-write C++ atomic operations such as compare_exchange, fetch_add, and exchange; and that it would be very useful for complex access patterns if memory model simulation tools can generate a list of memory ordering options for a litmus test without requiring the user to specify memory ordering in the litmus test.

3.22 Static Analysis by Abstract Interpretation of Numeric Properties of Programs under Weak Memory Models

Antoine Miné (CNRS and University Pierre & Marie Curie – Paris, FR)

Joint work of Antoine Miné, Thibault Suzanne
Main reference T. Suzanne, A. Miné, “From Array Domains to Abstract Interpretation Under Store-Buffer-Based Memory Models”, in Proc. of the 23rd Int’l Static Analysis Symp. (SAS’16), LNCS, Vol. 9837, pp. 469–488, Springer, 2016.
URL http://dx.doi.org/10.1007/978-3-662-53413-7_23

In this talk, we discuss the verification of concurrent programs running under weak memory models by sound and automatic static analysis based on abstract interpretation. We first recall the principles of abstract interpretation and the well-known result that abstracting thread interference in a flow-insensitive way makes the analysis robust against reordering of independent reads and writes. Then, we focus on the TSO and PSO memory models, and propose more precise abstractions tailored for these models. Starting from an operational semantics, we leverage existing numeric abstract domains as well as array abstractions to model the store buffers in a sound way. Finally, we present an application to fence removal on small code examples.

3.23 Musketeer in Dagstuhl: Automated Fencing in Software?

Vincent Nimal (Microsoft Research UK – Cambridge, GB)

Joint work of Jade Alglave, Daniel Kroening, Vincent Nimal, Daniel Poetzl
Main reference J. Alglave, D. Kroening, V. Nimal, D. Poetzl, "Don't Sit on the Fence", in Proc. of the 26th Int'l Conf. on Computer Aided Verification (CAV'14), LNCS, Vol. 8559, pp. 508–524, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08867-9_33

```
                |
               / \                +
              / + \____           /\
             / ||| \   \\         ||
             |_____|____\\       /__\
____   ______|     |  , ||  _____|  |____
  /\ \/      | ||| |____||_/|  , |__| ,  |\
 _____\======| === |      /_|____________|_\
 /\ /\ |     ||   ||______| || /\     /\ || |
 || || | |+| ||||||| |+|  | || || /=\ || || |
       |     ||___||      | || || | | || || |
 || || | |+| || | || |+|  | ||    | |    || |
_______|_____||_|_||______|_||___|===|___||_|

‘Power, ARM!‘                       ‘SC!‘
        \                            /
         ~__    .                   __~
          |)  /                    (|
         /|__/                 _____\_/
        | |                          \
         /|                          /\
        / |                         |  \

Musketeer in Dagstuhl: automated fencing in software?
-----------------------------------------------------
```

Modern architectures rely on memory fences to prevent undesired weakenings of memory consistency. As the fences' semantics may be subtle, the automation of their placement is highly desirable, e.g., in the context of legacy code. But precise methods restoring consistency do not scale to deployed systems code. We choose to trade some precision for scalability: we present a technique suitable for larger code bases. This method is implemented in the tool musketeer, that we experimented on more than 350 executables of packages found in a Debian Linux distribution, e.g. memcached (about 10000 LoC).

This talk recalls some results of our CAV 2014 paper, with updated results and insights from our TOPLAS paper. It then discusses some difficulties inherent to the fence insertion problem preventing good compositionality, which also apply to other approaches.

3.24 Verifying a Concurrent Garbage Collector

Gustavo Petri (University Paris-Diderot, FR) and Delphine Demange (IRISA – Rennes, FR)

Joint work of Delphine Demange, Suresh Jagannathan, Gustavo Petri, David Pichardie, Jan Vitek, Yannick Zakowski

We consider the problem of mechanically verifying a state-of-the-art, on-the-fly concurrent garbage collector. To facilitate this task, we present a compiler intermediate representation (IR) that subsumes a concurrent programming language and a proof methodology. Our IR provides strong type guarantees, abstract concurrent data structures, and intrinsic support for threads, roots management and object inspection via high-level iterators. Our IR is also accompanied with a rely-guarantee program logic which we use to prove the functional correctness of programs. In the implementation of our collector, data races are omnipresent. To argue about the correctness of our garbage collector under the TSO memory model, we plan to exploit the fact that the "safe publication idiom" under TSO provides a semantics equivalent to that under the SC memory model.

3.25 Mixed-Size Concurrency: ARM, POWER, C/C++11, and SC

Susmit Sarkar (University of St. Andrews, GB)

Joint work of Mark Batty, Shaked Flur, Kathryn E. Gray, Luc Maranget, Kyndylan Nienhuis, Christopher Pulte, Susmit Sarkar, Peter Sewell, Ali Sezgin
Main reference S. Flur, S. Sarkar, C. Pulte, K. Nienhuis, L. Maranget, K. E. Gray, A. Sezgin, M. Batty, P. Sewell, "Mixed-Size Concurrency: ARM, POWER, C/C++11, and SC", in Proc. of the 44th ACM SIGPLAN Symp. on Principles of Programming Languages (POPL'17), pp. 429–442, ACM, 2017.
URL http://dx.doi.org/10.1145/3009837.3009839

Previous work on the semantics of relaxed shared-memory concurrency has only considered the case in which each load reads the data of exactly one store. In practice, however, multiprocessors support mixed-size accesses, and these are used by programs in C/C++, and particularly systems software.

I will describe recent work on modeling mixed-size behavior of POWER and ARM architectures and implementations, showing new aspects of memory consistency models that arise in this setting. In particular, the abstract notion of coherence becomes more subtle, and adding a barrier between each instruction does not restore Sequential Consistency.

This work was published at POPL'17.

3.26 Reachability for Dynamic Parametric Processes

Helmut Seidl (TU München, DE)

Joint work of Anca Muscholl, Helmut Seidl, Igor Walukiewicz
Main reference A. Muscholl, H. Seidl, I. Walukiewicz, "Reachability for Dynamic Parametric Processes", in Proc. of the 18th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'17), LNCS, Vol. 10145, pp. 424-441, Springer, 2017.
URL http://dx.doi.org/10.1007/978-3-319-52234-0_23

In a dynamic parametric process every subprocess may spawn arbitrarily many, identical child processes, that may communicate either over global variables, or over local variables that are shared with their parent.

We show that reachability for dynamic parametric processes is decidable under mild assumptions. These assumptions are, e.g., met if individual processes are realized by pushdown systems, or even higher-order pushdown systems. We also discuss in how far these methods can also deal with weak memory models.

3.27 Data Consistency Check of Very Large Execution Traces

Suzanne Shoaraee (ARM France SAS – Sophia-Antipolis, FR)

Verifying ARM CPU implementations could be challenging especially when it deals with the memory system. In this talk, I present one of our current challenges: be able to check the data consistency of very large execution traces containing millions of memory accesses.

I introduce ARM specific requirements (use of temporal information, barriers parameters ...) and the difficulty to adapt existing formalizations of the memory model. A limited but scalable checker has been developed and is presented as it is already of interest to our verification teams.

At last the remaining challenges to be solved such as the handling of single, multiple-copy atomicity, atomic instructions or barriers are discussed.

3.28 From Architecture to Implementation

Daryl Stewart (ARM Ltd. – Cambridge, GB)

An architecture specification represents a contract between hardware and software which defines the permitted behaviors of a system. For reasoning about software this contract should be weak so that programmers code defensively against undesirable outcomes. For hardware it should be strong, so that implementations exhibit no more behaviors than permitted (or expected.) We propose that a no-man's land between the two communities is safer than attempting perfection.

When verifying hardware during bottom-up development we seek a framework for ensuring that the local behavior of subunits is correct with respect to the specified global behaviors. I will show some of the local behaviors of hardware which give rise to the surprising behaviors of weak memory systems, along with some verification properties which can be applied to them in order to highlight the semantic gap between architecture and implementation.

3.29 TriCheck: Memory Model Verification at the Trisection of Software, Hardware, and ISA

Caroline J. Trippel (Princeton University, US)

Joint work of Daniel Lustig, Yatin A. Manerkar, Margaret Martonosi, Michael Pellauer, Caroline J. Trippel
Main reference C. Trippel, Y. A. Manerkar, D. Lustig, M. Pellauer, M. Martonosi, "TriCheck: Memory Model Verification at the Trisection of Software, Hardware, and ISA", to appear in Proc. of the 22nd Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS'17), 2017; pre-print available from author's webpage.
URL http://mrmgroup.cs.princeton.edu/papers/ctrippel_ASPLOS17.pdf

The ISA is a multi-part specification of hardware behavior as seen by software. One significant, yet often under-appreciated aspect of this specification is the memory consistency model which governs inter-module interactions in a shared memory system. We make a case for full-stack memory model design and verification and provide a toolflow – TriCheck – to support it. We apply TriCheck to the open source RISC-V ISA, focusing on the goal of accurate, efficient, and legal compilations from C11/C++11. In doing so, we uncover under-specifications and potential inefficiencies in the current RISC-V ISA documentation and identify possible solutions for each. We also identify two counter-examples to previously "proven-correct" compiler mappings from C11/C++11 to POWER and ARMv7.

3.30 Explaining Relaxed Memory Models with Program Transformations

Viktor Vafeiadis (MPI-SWS – Kaiserslautern, DE)

Joint work of Ori Lahav, Viktor Vafeiadis
Main reference O. Lahav, V. Vafeiadis, "Explaining Relaxed Memory Models with Program Transformations", in Proc. of the 21st Int'l Symp. on Formal Methods (FM'16), LNCS, Vol. 9995, pp. 479–495, Springer, 2016.
URL http://dx.doi.org/10.1007/978-3-319-48989-6_29

Weak memory models determine the behavior of concurrent programs. While they are often understood in terms of reorderings that the hardware or the compiler may perform, their formal definitions are typically given in a very different style – either axiomatic or operational. In the talk, we investigate to what extent weak behaviors of existing memory models can be fully explained in terms of reorderings and other program transformations. We prove that TSO is equivalent to a set of two local transformations over sequential consistency, but that non-multi-copy-atomic models (such as C11, Power and ARM) cannot be explained in terms of local transformations over sequential consistency. We then show that transformations over a basic non-multi-copy-atomic model account for the relaxed behaviors of (a large fragment of) Power, but that ARM's relaxed behaviors cannot be explained in a similar way. Our positive results may be used to simplify correctness of compilation proofs from the promising semantics of Kang et al. [1] to TSO or Power. More details can be found in [2].

References

1 J. Kang, C.-K. Hur, O. Lahav, V. Vafeiadis, D. Dreyer. A promising semantics for relaxed-memory concurrency. POPL 2017, pp. 175–189, ACM, 2017.

2 O. Lahav, V. Vafeiadis. Explaining relaxed memory models with program transformations. FM 2016, LNCS, Vol. 9995, pp. 479–495, Springer, 2016.

3.31 Event Structures and Stable Families

Glynn Winskel (University of Cambridge, GB)

Main reference G. Winskel, "Event Structure Semantics of CCS and Related Languages", in Proc. of the 9th Colloquium on Automata, Languages and Programming (ICALP'82), LNCs, Vol. 140, pp. 561–576, Springer, 1982; pre-print available from author's webpage.
URL http://dx.doi.org/10.1007/BFb0012800
URL https://www.cl.cam.ac.uk/~gw104/eventStructures82.pdf

This talk revisits old work on Event Structures (1978) and Stable Families (1981) which are relevant or potentially relevant in the modeling of weak memory in hardware design. Some recent work, e.g. that of Alan Jeffrey or separately Simon Castellan, uses event structures in modeling weak memory. In particular, Castellan uses a product of event structures; his work also fits within concurrent games where the composition of strategies uses the pullback of event structures. Both product and pullback of event structures are difficult to define directly on event structures. Here stable families come to the rescue: a coreflection from a category of event structures to a category of stable families allows us to transport the constructions from the simpler construction of product and pullback in stable families.

3.32 Weak Memory Models: Balancing Definitional Simplicity and Implementation Flexibility

Sizhuo Zhang (MIT – Cambridge, US)

Joint work of Arvind, Muralidaran Vijayaraghavan, Sizhuo Zhang
Main reference S. Zhang, Arvind, M. Vijayaraghavan, "Taming Weak Memory Models", arXiv:1606.05416v1 [cs.PL], 2016
URL http://arxiv.org/abs/1606.05416

RISC-V, a newly developed open source ISA, has not finalized its memory model, and thus offers an opportunity to explore the design space of weak memory models. We propose two new weak memory models: WMM and WMM-S, which balance definitional simplicity and implementation flexibility differently. Both allow all instruction reorderings except overtaking of loads by a store. We show that this restriction has little impact on performance and it considerably simplifies operational definitions. It also rules out the out-of-thin-air problem that plagues many definitions. WMM is simple (it is similar to the Alpha memory model), but it disallows behaviors arising due to shared store buffers and shared write-through caches (which are seen in POWER processors). WMM-S, on the other hand, is more complex and allows these behaviors. We give the operational definitions of both models using

Instantaneous Instruction Execution (I2E), which has been used in the definitions of SC and TSO. We also show how both models can be implemented using conventional cache-coherent memory systems and out-of-order processors, and encompasses the behaviors of most known optimizations.

Participants

- Jade Alglave
  University College London, GB
- Giovanni Tito Bernardi
  University Paris-Diderot, FR
- Annette Bieniusa-Middelkoop
  TU Kaiserslautern, DE
- Richard Bornat
  Middlesex University –
  London, GB
- Stephen Brookes
  Carnegie Mellon University –
  Pittsburgh, US
- Simon Castellan
  ENS – Lyon, FR
- Andrea Cerone
  Imperial College London, GB
- Pierre Clairambault
  ENS – Lyon, FR
- Patrick Cousot
  New York University, US
- Andrei Marian Dan
  ETH Zürich, CH
- Will Deacon
  ARM Ltd. – Cambridge, GB
- David Delmas
  Airbus S.A.S. – Toulouse, FR
- Delphine Demange
  IRISA – Rennes, FR
- Stephan Diestelhorst
  ARM Ltd. – Cambridge, GB
- Charles Anthony Richard Hoare
  Microsoft Research UK –
  Cambridge, GB
- Vincent Jacques
  University College London, GB
- Bernhard Kragl
  IST Austria –
  Klosterneuburg, AT
- Ori Lahav
  MPI-SWS – Kaiserslautern, DE
- Daniel Lustig
  NVIDIA Corp. –
  Santa Clara, US
- Yatin Manerkar
  Princeton University, US
- Luc Maranget
  INRIA – Paris, FR
- Paul McKenney
  IBM – Beaverton, US
- Paul-Andre Mellies
  University Paris-Diderot, FR
- Roland Meyer
  TU Braunschweig, DE
- Maged M. Michael
  Facebook – New York, US
- Antoine Miné
  CNRS and University Pierre &
  Marie Curie – Paris, FR
- Vincent Nimal
  Microsoft Research UK –
  Cambridge, GB
- Andrea Parri
  INRIA – Paris, FR
- Gustavo Petri
  University Paris-Diderot, FR
- Susmit Sarkar
  University of St. Andrews, GB
- Helmut Seidl
  TU München, DE
- Suzanne Shoaraee
  ARM France SAS –
  Sophia-Antipolis, FR
- Daryl Stewart
  ARM Ltd. – Cambridge, GB
- Caroline J. Trippel
  Princeton University, US
- Caterina Urban
  ETH Zürich, CH
- Viktor Vafeiadis
  MPI-SWS – Kaiserslautern, DE
- Derek Williams
  IBM Research Lab. – Austin, US
- Glynn Winskel
  University of Cambridge, GB
- Sizhuo Zhang
  MIT – Cambridge, US