Report from Dagstuhl Seminar 22492

Formal Methods and Distributed Computing: Stronger Together

Hagit Attiya*[1], Constantin Enea*[2], Sergio Rajsbaum*[3], and Ana Sokolova*[4]

1 **Technion – Haifa, IL.** `hagit@cs.technion.ac.il`
2 **Ecole Polytechnique – Palaiseau, FR & CNRS – Palaiseau, FR.** `cenea@lix.polytechnique.fr`
3 **National Autonomous University of Mexico, MX, on leave at IRIF, Paris, FR.** `rajsbaum@im.unam.mx`
4 **University of Salzburg, AT.** `anas@cs.uni-salzburg.at`

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 22492 "Formal Methods and Distributed Computing: Stronger Together", held in December 2022.

Seminar December 4–9, 2022 – http://www.dagstuhl.de/22492
2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Verification by model checking; Software and its engineering → Formal methods
Keywords and phrases automated verification and reasoning, concurrent data structures and transactions, distributed algorithms, large-scale replication
Digital Object Identifier 10.4230/DagRep.12.12.27

1 Executive Summary

Hagit Attiya (Technion – Haifa, IL)
Constantin Enea (Ecole Polytechnique – Palaiseau, FR & CNRS – Palaiseau, FR)
Sergio Rajsbaum (National Autonomous University of Mexico, MX, on leave at IRIF, Paris, FR)
Ana Sokolova (University of Salzburg, AT)

License Creative Commons BY 4.0 International license
© Hagit Attiya, Constantin Enea, Sergio Rajsbaum, and Ana Sokolova

Distributed applications represent nowadays a significant part of our everyday life. To mention just a few examples, our personal data are stored on remote distributed servers, data management relies on remote applications reachable via smartphones or tablets, data-intensive computations are performed on computer clusters, etc. Since distributed applications are increasingly deployed at large scale, they have to be reliable and robust, satisfying stringent correctness criteria. This is the point where a strong interaction of formal methods and of distributed computing becomes a necessity.

The goal of this Dagstuhl Seminar was to achieve a synergy by bringing together researchers working on applying formal methods for concurrent programs and distributed systems, and researchers from distributed computing. Both communities have a deep understanding of distributed computation, but from two different perspectives. Historically, these communities

* Editor / Organizer

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 4.0 International license

have common roots, but since more than two decades they evolved independently. The resulting gap slows down progress in both fields, and limits the applicability of the results obtained in each field, as each one develops its own techniques separately. The seminar addressed several topics that bridge the two research fields, and that have high potential to stimulate the development of the other area:

Concurrent data structures and transactions: Modern multicore architectures enable large performance boosts by executing a number of threads in parallel, which however, poses considerable challenges in maintaining correctness of shared data structures and thread synchronization. These challenges have been addressed using various paradigms like lock-free programming or transactional memory. However, turning these concepts into efficient programming support remains a big challenge, and formal methods may offer new ideas in this direction.

Formal approaches to large-scale replication: Current computing systems are increasingly large-scale distributed systems, for example, distributed databases, distributed ledgers (Blockchains) and key-value stores. At the heart of these systems are fundamental trade-offs between data consistency, availability, and the ability to tolerate failures. A formal approach to studying these issues will provide a common ground for the design, verification, analysis, implementation and use of these systems.

Distributed algorithms for verification: Reasoning about concurrent/distributed software is notoriously difficult due to the inherent non-determinism in its semantics. The different processes in a concurrent program can interleave in many different ways which leads to an enormous number of possible executions. Algorithmic methods are necessary to mitigate the difficulty of reasoning about this huge space of executions, and scalable distributed algorithms may be the answer for the future. These methods can manifest in various forms, e.g., automated testing, deductive verification, model checking, and have led to important results in many timely contexts. Performing verification in a distributed fashion is a particularly promising new direction of research.

The impact of all the areas above on a rigorous development of distributed applications was enhanced by fostering direct interactions between researchers from (automated) formal methods and from distributed computing.

2 Table of Contents

Executive Summary
Hagit Attiya, Constantin Enea, Sergio Rajsbaum, and Ana Sokolova 27

Experience Reports

Seminar Overview
Jennifer Welch 31

Panel discussions

Panel: Applications in Industry
Giuliano Losa 42

Talk Abstracts

(Approximately) Preserving Hyper-Properties without Strong Linearizability
Hagit Attiya 42

Verification of Liveness Properties on Weakly Consistent Platforms (TSO as an example)
Parosh Aziz Abdulla 43

A CEGAR approach to parameterized verification of distributed algorithms
Nathalie Bertrand 44

Highly-available access control in distributed systems
Annette Bieniusa 44

Verification of Distributed Systems
Ahmed Bouajjani 44

Asynchronous Distributed Runtime Verification and Enforcement of Linearizability
Armando Castaneda 45

Synchronizer: a recipe for building correct algorithms under partial synchrony
Gregory Chockler 45

Hyperproperties in Synthesis and Verification
Bernd Finkbeiner 46

Wait-Free Speedup Theorem
Pierre Fraigniaud 46

Correctness Conditions for Cross-chain Transactions
Maurice Herlihy 47

Testing Blockchain Consensus Algorithms
Burcu Kulahcioglu Ozkan 47

War of Consistency Violations and Self-Stabilization
Sandeep Kulkarni 47

Correctness in Distributed Computing
Petr Kuznetsov 48

Abstraction for Crash-Resilient Objects
Ori Lahav 48

Random Testing with Theoretical Guarantees
Rupak Majumdar 48

Tool Support for TLA+: TLC, Apalache, and TLAPS
Stephan Merz 49

On Graph Connectivity in Distributed Algorithms
Yoram Moses 49

Truthful information Dissemination in Asynchronous Networks
Rotem Oshman 49

Extending Intel-x86 Consistency and Persistency
Azalea Raad 50

Formal Methods for Distributed Systems at Amazon Simple Storage Service (S3)
Serdar Tasiran 50

Implementing Shared Objects in the Presence of Continual Churn
Jennifer L. Welch 51

Reasoning Principles for Verifying Concurrent Search Structures
Thomas Wies 51

Predictable Building Blocks for Randomized Shared Memory Algorithms
Philipp Woelfel 52

Participants 53

3 Experience Reports

3.1 Seminar Overview

Jennifer Welch (Texas A&M University - College Station, US)

License Creative Commons BY 4.0 International license
© Jennifer Welch

The seminar brought together researchers in distributed computing and in formal verification. It was a successful combination, with the right amount of overlap for appreciating the novelty and understanding the main points of the other community. Participants took part in lively conversations in the Q&A of the presentations, at the coffee breaks, meals, and after dinner. There was an enjoyable excursion to the bustling Christmas market in the nearby picturesque town of St. Wendel.

Monday, Dec. 5

The morning was devoted to two tutorials, to help the participants from the two areas get an overview of the other area.

The first tutorial, by **Victor Vafeiadis**, was entitled "Verification of Concurrent Data Structures." In addition to objects with sequential semantics, he also considered synchronization constructs, which are often bottlenecks in programs and thus are heavily optimized in practice. Victor started by pointing out there there several axes along which to consider the verification problem, in order to determine if the outcome is worth the effort. These include

- At what level of abstraction, between binary code and pseudocode, is the program to be verified?
- What properties are to be verified? These could include memory safety, incomplete functional correctness, linearizability, and liveness. Different properties may need different methods.
- What assumptions should be made about the memory model, memory management, and fairness. It is important that the memory model choice be correct with respect to the level at which the program is being verified.
- How thorough will the verification be? For machine-checked proofs, will all the metatheory be mechanized, or will some properties be axiomatized? Will pragmatic simplifying assumptions be made? Can quantities such as buffer sizes and number of threads be bounded?
- How much human expertise in the approach and tools will be needed? How much information will the user need to supply (e.g., specification, proof outine, invariants)?
- How much human and machine effort (time) will be needed?

Several different verification approaches were outlined. These were simulation of automata and program logic, both of which require a high level of expertise and take weeks; static analysis on a simplified implementation, requiring little-to-medium level of expertise and either takes minutes or fails; model checking of real code which requires little human expertise or time but requires an unpredictable amount of machine time which can be large; and randomized testing, which works on real code, requires little human expertise or effort and a tunable amount of machine effort, but gives no guarantees. State-of-the-art industrial practice is to use a combination of techniques. Barriers to adoption include the human cost and lack of expertise for interactive proofs, and the amount of time and lack of a "progress bar" for model checking.

The rest of the tutorial focused on his research on making model checking practical. To make it faster, one can employ state-space reduction, better algorithms, and reliance on user hints and common code patterns. To estimate the time that will be taken, one can estimate the size of the state space with randomized testing, and then use a "predictable" fast modeling checking algorithm whose time is proportional to the size of the state space. One such approach is his algorithm, TruST, "Truly Stateless C". In order to avoid redundant exploration of the state space and to limit the amount of memory needed, TruST explores alternate executions eagerly, represents executions with graphs, and uses maximal extensions. He went into more detail about the algorithm with a nice small running example.

Pierre Fraignaud gave a two-part tutorial on Distributed Certification.

The first part of the presentation assumed a fixed static network in which one would like to provide some kind of fault-tolerance, such as detecting illegal states, checking correctness of the output of a subroutine, or preventing the launch of a protocol if its prerequisites (such as having a spanning tree) are not satisfied. This problem is formalized as a graph in which each node has as input an id and a label, and we would like to know if the graph satisfies a predicate. For the distributed certification version of the problem, imagine there is a *prover*, a centralized, computationally unlimited, non-trusted orace that assigns certificates to nodes, and a *verifier*, a one-round, synchronous, failure-free distributed algorithm in which nodes exchange certificates with their neighbors to check whether the certificates form a proof that the system satisfies the predicate. Correctness properties are completeness (if the predicate is satisfied, then the verifier accepts at all nodes) and soundness (if the predicate is not satisfied, then the verifier rejects at at least one node). Similar notions in the literature include proof labeling schemes (PLS), locally checkable proofs, and nondeterministic local decisions. The main complexity measure of interest is the size of the certificates. Fortunately, every Turing-decidable predicate has a PLS using certificates of size $O(n^2 + kn)$ bits, where k is the size of the labels. Unfortunately, there exist predicates that require $\tilde{\Omega}(n^2)$ bit certifications, including the properties of not being 3-colorable and having a nontrivial automorphism. A useful primitive that can be certified with $O(\log n)$-bit certificates is a spanning tree: let the certificate be the pair (root id, distance to root). A number of other problems can be certified using $O(\text{polylog}(n))$-bit certificates, including minimum spanning tree, approximate optimization problems, and planar graph. He then presented some more recent results that can be viewed as meta-theorems.

The second part of the tutorial considered analogous ideas in the asynchronous wait-free model of computation. One theorem is that binary consensus is not *checkable* in this model, which can be proved much more easily than showing it is not *computable*. In fact, consensus is checkable with three values (accept, reject, inconclusive). Extensions to k-set agremeent and distributed runtime monitoring were discussed. A general result is that every input-output decidable predicate in an n-process system can be certified with certificates of size at most $\lceil Ack^{-1}(n) + 1 \rceil$ bits using 3-valued decisions; in contrast, $O(1)$-bit certificates do not suffice.

In summary, for the synchronous failure-free model, the theory of distributed certification is mature and evolving rapidly. In contrast, in the asynchronous crash-prone model, there are only partial results that need consolidating. Discussion with the audience during the Q&A indicated there might be some issue with the model in the latter case.

In the afternoon, the "regular" (30-minute) talks began.

Bernd Finkbeiner spoke about hyperproperties in synthesis and verification. A *hyperproperty* is a set of traces; examples include non-interference, secrecy, differential privacy, symmetry, fault detectability over a channel, robustness, partial observation, and dominance. In spite of much work on hyperproperties, there is a gap between general specification logics

and practical verification and synthesis algorithms. Bernd presented some research aimed at closing this gap. For example, HyperLT, which is linear-time temporal logic (LTL) plus prenex quantification over traces, is expensive to model-check because of the quantifiers. If we restrict attention to some specific graphs on the Kripke structure, then model-checking can be faster. To model-check HyperLTL, one can reduce the problem to emptiness of Büchi word automata. A recent approach is to apply ideas from game theory; for example, it can be shown that if the existential player has a winning strategy, then the system satisfies the hyperproperty.

The next talk was by **Parosh Aziz Abdula** on checking liveness properties under weak consistency, using TSO as an example. The definition of sequential consistency (SC) is that writes are immediately visible in the memory and reads are from memory; SC is simple and intuitive but expensive to implement. A weaker, and more efficiently-implementable, condition is total-store order (TSO), in which writes are non-atomic while reads are local or from memory. The definition of TSO uses the notion of a store buffer at each process, and data in the store buffer is nondeterministically transferred to memory. There are many existing works on verifying safety properties for TSO but not much on liveness properties. Parosh presented a way to take techniques for proving properties about SC, especially liveness, and extend them to TSO. Assuming some probabilistic fairness of the underlying system, such as that messages eventually leave the store buffers and each message updates memory with the same probability, one can prove that updates occur frequently and buffers tend to be small. Now we can use tools for SC and reduce to reachability analysis.

The rest of the afternoon was devoted to the industry panel, described later.

Tuesday, Dec. 6

The tutorials continued in the morning.

The first tutorial was by **Ahmed Bouajjani** on verification of distributed systems. He considered the situation where clients interact with an application layer, which runs on top of a data storage system, which in turn runs on top of a communication network. The tutorial focused on problems that arise when verifying such systems, especially related to consistency conditions, including isolation levels for transactions. He then presented (1) a formal framework for specifying the conditions, (2) ways to verify an application running under a weak consistency model, and (3) ways to verify that a storage system provides a certain consistency level.

To specify a condition, we define the expected observable behaviors when interacting with memory. The values returned by reads depend on the current set of "visible" actions by each process and the order in which actions are seen by each process. In *strong consistency*, updates are visible without delay and in the same order. If the memory is replicated, a weaker consistency holds, since participants can see different sets of updates and in different orders. Causal consistency, sequential consistency, and snapshot isolation were given as examples.

Challenges in verifying reachability under weak consistency include complex program semantics and reordering of operations, which requires unbounded memory to track dependencies. In many situations, the problem is undecidable; when it is decidable, it can be expensive. "Robustness" is checking whether the semantics of a program is the same under two different consistency models. He surveyed a line of work in this direction, with different papers considering different consistency conditions and different data structures.

In conclusion, decidability and complexity are still open in many cases. We need efficient static analysis algorithms and testing methods. We also need languages for describing different consistency models. And finally, we need to be able to reason systematically about tradeoffs between consistency and performance. During the Q&A session, someone raised the question about analyzing systems that have combinations of different consistency levels. Apparently, there is no formal work in this area.

The last tutorial, by **Petr Kuznetsov**, was on correctness in distributed computing. He focused on correctness conditions that relate the behavior of a concurrent system to that of a sequential system, especially for implementing shared objects in the presence of crash failures. One way of classifying properties is into "safety" (informally, nothing bad ever happens) and "liveness" (informally, eventually something good happens). He compared and contrasted how Lynch formally defined these properties and how Alpern and Schneider did, and their approaches to proving that every property is the intersection of a safety property and a liveness property. He also went over a few techniques for proving safety and liveness properties, including the interesting fact by Padon et al. that liveness can sometimes be reduced to safety. Next he defined the safety properties "linearizability" [1] and "sequential consistency" and discussed some of their pros and cons. A variety of progress properties (from a table in Herlihy and Shavit) were presented. Then he introduced hyperproperties, which are sets of traces.. *There are no formal definitions of safety and liveness for these; such definitions might need a topology on sets of traces.* Then he talked about a particular hyperproperty, "strong linearizability", which is a strengthening of linearizability that preserves probability distributions of the enclosing program.

One way of implementing wait-free linearizable objects is to rely on consensus, which can provide a total order on concurrent, or contending, operations. But this can be expensive or even impossible. An alternate approach to resolve contending operations is with the notion of a lattice, where overlapping operations can return any join of previous and overlapping operations. This definition is suitable for CRDTs (conflict-free replicated data types). Then he talked about using quorum systems.

In conclusion, the philosopher's stone of distributed computing would be a universal machine, allowing any sequential program to be run in a distributed environment with all the good properties. But because of challenges in dealing with real systems, there are numerous lower bounds and impossibility results. Maybe the way around the challenges is to consider weaker properties or friendlier models. One issue raised during the Q&A, is that sequential consistency is not very useful for some data structures that are inherently concurrent, for example, a producer-consumer queue.

The regular 30-minute talkes continued in the afternoon.

Jennifer Welch's talk was on implementing shared objects in the presence of continual churn. She presented a model for crash-prone dynamic systems that allows a limited number of processes to enter and leave during time intervals of a fixed length. The model was inspired by mobile ad hoc networks and may have relevance for permissionless blockchains. Algorithms for implementing a linearizable register and a non-linearizable store-collect object were presented, which make progress even while churn is ongoing. There is a lower bound on the crash resilience, showing that the larger the rate of churn, the smaller must be the fraction of faulty processes.

[1] A later clarification was given: Lynch proved in 1996 that linearizability is a safety property for finite nondeterminism, otherwise, as shown by Guerraoui and Ruppert in 2014, it is not.

Ori Lahav spoke about abstraction for crash-resilient objects. There has been recent interest in *persistent objects*, concurrent objects that can recover from crashes, which are implemented in non-volatile memory (NVM). A natural question is how to define correctness as some variation of linearizability, especially in a way that can be used in verification. His approach is to focus on refinement with respect to another implementation. The new aspect is to include special constructs in the language that allow for intuitive specifications. He focused on the simplest model, persistent sequential consistency and explained how it can be mapped to x86-TSO by adding appropriate fence instructions.

Burcu Kulahcioglu Ozkan's talk was on testing blockchain consensus algorithms, with a focus on consortium/voting-based Byzantine fault-tolerant (BFT) consensus algorithms, inspired by PBFT. The correctness of the consensus algorithm is crucial for the correctness of the overall blockchain. However, many bugs have been found in BFT algorithms, in part because there is a lack of practical testing tools. For instance, software model checking is infeasible due to the possibility of Byzantine faults. Random testing is the current practice for concurrency and network faults. Burcu's talk showed how to apply this idea to blockchain consensus algorithms. First, she explained the challenges to be overcome to make this idea practical. For enumerating executions, she proposed using abstractions of time with protocol rounds. For network partition faults, a small set of random partitions can provide full coverage with high probability. Byzantine faults are modeled by structure-aware and small-scope mutations to protocol messages. This provides equivocation, amnesia, double-voting, losing internal state, and incorrect forward progress. Using this approach, she found a new bug in the Ripple XRP Ledger, namely a violation of termination.

Annette Bieniusa spoke about highly available access control in distributed systems. Assuming the system has POSIX access control policies that specify the types of authorization and the types of access allowed, we need to avoid conflicts without causing internal data leakage and corruption. Her recent work proved the CIA theorem (Confidentiality, Integrity, and Accessibility), analogous to the CAP theorem, showing that one cannot achieve all three properties if there are partitions. In response, she considered weakening the security properties to have a causal order, instead of total order, on access control policies and data. Her work formalized the approach in the modeling system Repliss and tested for counter-examples, which found some corner cases. Her ongoing work is considering decentralized systems with multiple administrators using the idea of forking into parallel universes. She concluded her talk with a discussion of lessons learned.

Gregory Chockler's presentation was about a synchronizer primitive for building correct algorithms under partial synchrony, in the presence of Byzantine failures. He reviewed the motivation for partial synchrony as well as the challenges for designing bug-free consensus algorithms for blockchains. He proposed an abstraction for leader-driven consensus, based in a sequence of views, that cleanly separates safety and liveness properties. He described a case study for proving that PBFT is live using this abstraction. The structure of this liveness proof can be reused for proofs of other protocols, as it establishes properties similar to failure detectors (for crash failures). They are completeness (if a correct process never executes a received command, then it eventually advances to a new view) and eventual accuracy (eventually, in a view of a correct leader and with large enough timeouts, no correct process will advance to a new view). He then explained how to implement the synchronizer using ideas from Bracha's broadcast algorithm, modified to work in bounded space.

Alexey Gotsman presented his work on getting strong consistency and availability under network partitions. He first pointed out that the CAP theorem does not preclude availability in the majority side of a partition. His work looked more deeply into the kind

of communication failures that can occur and considers partially synchronous systems with some unidirectional channels that can fail by dropping messages whenever they choose. New results are that consensus (state machine replication) is solvable if and only if at most a minority of processes crash and there is a majority of correct processes that are strongly connected via correct channels. Then he presented a specification and implementation of a synchronizer for this model (cf. immediately previous talk by Gregory Chockler). His last topic was finding the minimal amount of connectivity needed to guarantee availability anywhere at all. For this, he first showed that places where the system is available must be connected, and that to be available more than half the processes must be connected.

Wednesday, Dec. 7

Rupak Majumdar's talk was on random testing with theoretical guarantees. He first pointed out that despite the existence of formal approaches, practitioners usually test their code and it's effective. To explain why, he gave a metaphor using ninjas at a banquet for testing distributed systems where events are partially ordered and we need a schedule for tests. It turns out that many bugs in production software involve just a few events, say d, and thus finding them only requires a d-hitting family of schedules. This concept is related to the "order dimension" of a poset. When the system is running, we can learn an "upgrowing" poset in an on-line fashion. His random testing algorithm PCTCP maintains an on-line chain partition, assigns a random priority to each chain, and executes enabled events from the highest priority chain. At certain random points in the execution, the priority of a chain is decreased. Follow-on work, the Trace Aware HitMC algorithm, exploits knowledge of the algorithm using the notion of communication-closed rounds. A general open question in this area is providing a theoretical explanation for situations when randomized testing behaves well.

Yoram Moses spoke on graph connectivity in distributed algorithms. A new distributed problem called *card consensus*, which cannot be solved under certain circumstances, was used as a running example. A general theorem was presented showing that specifying constraints on actions induces requirements on the knowledge that the processes must have. Using this theorem, he provided an *epistemic* analysis of card consensus, and also discussed the impact of requiring events to occur simultaneously. In general, knowledge can be modeled as a graph, and is the dual of *indistinguishability*, a concept with applications to numerous classical results in distributed computing. Epistemic reasoning about card consensus is the one-dimensional analog of *topological* reasoning for the set consensus and renaming problems. Both epistemic and topological reasoning are useful in analyzing distributed computing. Challenges for future research include finding a clean variant of common knowledge that captures the consensus problem in asynchronous systems, finding topological models that can handle in a natural way timing-related constraints (such as actions that must occur close together), and adapting topological models to handle loss of information with global state.

Serdar Tasiran's presentation was on formal methods for distributed systems at Amazon Simple Storage Services (S3). He is in the automated reasoning group, which has about 20 people, while the whole S3 team has about 1000 developers. Their mandate is to integrate formal methods into software processes across S3. They use a range of methods, starting with lightweight to more powerful. Research is needed on methodology and tooling to enable this transition. Testing is widespread and integrates well into the software processes. Different tools need to be connected and assurance needs to be quantified. They are adapting a "model-first" approach, in which models and code are in the same repositories, and increasingly the models are being written before the code on new projects. An important problem is to

ensure that executions are consistent with models, so for instance, while you are watching Netflix, an automaton is being run! S3's new ShardStore storage node pays special attention to crash consistency, has a complex implementation, and is deployed weekly. Model checking and protocol-level deductive proofs are done in TLA+ style, but using Dafny. However, they are not mechanically connected.

Stephan Merz talked about tool support for TLA+. TLA+ is a specification language that has become a set of tools: TLC, Apalache, and TLAPS. Help is also provided by PlusCal and IDES. He used a distributed termination detection algorithm by Safra as a running example. First, he showed how to use TLA to specify the problem. Then one can start expressing correctness properties, including safety and liveness. TLC is an explicit state model checker that works for finite state descriptions, so the number of nodes and maximum number of pending messages need to be fixed. Apalache is a symbolic model checker based on SMT which checks properties of finite prefixes of an execution. Until recently, it only checked safety properties. It relies on constraint solving, not state enumeration. Again, the number of nodes needs to be fixed but it can handle infinitely many messages. The time grows exponentially with the length of the prefix and the number of nodes. However, if you have an inductive invariant, then you only need to check traces of size 0 or 1! TLAPS is a proof assistant; the proof effort here is independent of the size of the instance, although the user has to guide the verification. It uses automatic back-end provers to discharge proof obligations. All the tools share the same input language.

Giuliano Losa's presentation was on formal verification of a classic distributed algorithm using inductive invariants. His example algorithm is a termination detection algorithm of Kumar. He showed a new proof with a simple inductive invariant. He showed how to apply TLA+, Apalache, and Isabelle/HOL. Interestingly, the informal proof of correctness presented is actually wrong! The error is that the algorithm doesn't make sure that all processes are visited. After that problem in the code is fixed, another invariant is proposed. He played around with the tools to help him come up with the invariant, which is fairly simple. Then he did an automatic proof in Isabelle. One lesson from the talk is the advantage of human-machine collaboration to find invariants.

Thursday, Dec. 8

Philipp Woelfel talked about predictable building blocks for randomized shared memory algorithms. He started with a brief history of such algorithms, including the definition of the strong adaptive adversary. Replacing atomic (instantaneous) objects with linearizable objects gives this adversary more power in randomized algorithms. He defined *strong linearizability* (SL), which is sufficient, and necessary, for overcoming this problem. It also has the nice property of being composable. There are general SL constructions using locks or using universal constructions with, for example, compare-and-swap objects. He then summarized more efficient constructions and various positive and negative results. He has work in progress on constructing a strongly linearizable LL/SC object from CAS objects; the motivation is that CAS is available in hardware, but LL/SC is not although it is very useful in designing algorithms. The problem is that a CAS operation may succeed even though an ABA violation occurs, while LL/SC fixes this. One can easily get a strongly linearizable LL/SC from CAS with unbounded tags. The challenge is to bound the tags. He sketched his ideas for bounding the tags. An open question is to find additional efficient strongly linearizable implementations from strong synchronization primitives. Another is how to deal with the oblivious adversary. During the Q&A, the issue of finding complexity lower bounds for SL was raised as well.

Maurice Herlihy's presentation was on correctness conditions for cross-chain transactions. A distributed ledger is an abstraction that can be used for financial transactions and other applications where everyone agrees on the content and is tamper-proof. In the future, there will be many different chains and we want inter-operability. In the past, the classical adversary was considered for solving consensus. But the modern adversary corresponds to real people/governments and makes more sophisticated attacks and brings more complicated problems. Thus we need to rethink correctness. The classical ACID properties for transactions don't work for blockchain models. Instead of a fixed fraction of participants that can behave maliciously while the rest follow the protocol, now we have the notion of "deviating" parties with no bound on the number of deviators. He went through the ACID properties and spoke in more detail about how they need to be adapted for blockchains. (A) Atomicity is impossible; instead, we could use the conditions of liveness (if all parties conform, then all transfers happen) and safety (if some parties deviate, then the conforming parties are "no worse off"). Cross-chain commerce is a cooperative game, where a protocol is a strategy for that game. Parties can form coalitions and the result is the payoff. This area needs formalization! (C) Consistency says that application-specific constraints are respected. This could be restated by saying that conforming to the protocol should be a strong Nash equilibrium in the game. (I) Isolation states that no transaction sees another's intermediate steps. But we don't even want this condition in multiparty swaps, as we may want to use escrow accounts and similar mechanisms. (D) Durability says theat committed transactions survive crashes (a legacy of 20th century technology). In the blockchain world, we want to avoid "censorship" by government, corporations, hackers, etc. In summary, adversarial commerce is here to stay, regardless of specific blockchains, and we need to rethink notions of correctness.

Thomas Wies's talk was on reasoning principles for verifying concurrent search data structures, such as sets and maps. Modularity in proofs used to help us. But now it is the wrong level of abstraction. The challenge is that concurrency and memory safety proofs are intertwined. He gave examples of the problem with a B-link tree algorithm and a hash table algorithm. The proposed solution is to separate the two by finding a common link-technique proof hidden in the specific proofs, abstract it as a template, verify it once, and then use it in different data structure implementations. There are four issues to be dealt with. First, the template must be data-structure-independent. This is done using the concept of edge sets. Second, local operations might have global effects. This is dealt with using keyset resource algebra. Third, global properties must be rendered local. This is done using the inset concept and ideas of flows. Finally, in some implementations (those that are not strongly linearizable), the linearization points depend on the future, or said another way, the linearization points are determined in hindsight. The ability to do this has been added into a separation logic in his current work, supported by a tool called plankton that works well for proof automation. In summary, some modular techniques for reasoning about concurrent search data structures were presented which allow better proof automation.

Azalea Raad spoke about extending Intel-x86 consistency and persistency by formalizing the semantics of Intel-x86 memory types and non-temporal stores. Non-temporal stores write directly to memory, bypassing the cache which avoids cache pollution; they are heavily used in application-level code. One can declare the "type" (cache-ability) of memory with several different options, with the default being write-back (WB), which is used in system-level code. The extended Intel-x86 consistency semantics provide more options, as she explained with some small code examples. It turns out that the behavior of actual programs, especially

those that use a mixture of different kinds of accesses, is not consistent with what is written in the manual. The actual, validated, behavior is summarized in an extensive table. The next issue to be considered is persistence. When using non-volatile memory, execution can lag persistence. This behavior has been included in the consistency semantics. She tried to test for post-crash behavior by directly monitoring the memory bus using an expensive piece of hardware, but the results were inconclusive.

Armando Castañeda's presentation covered asynchronous distributed run-time verification and enforcement of linearizability. Efficient linearizable implementations are often complicated and bug-prone, as they use fine-grained synchronization. Formal verification of linearizability is sometimes undecidable or NP-hard. He presented a complementary dynamic approach which is to monitor running executions, ideally in a wait-free manner in order be less intrusive. The key differentiator from most previous work on runtime verification is the emphasis on being wait-free, that is, crash-tolerant in asynchronous systems. He first described his way of modeling the problem, as an interaction between a verifier and an implementation, both of which are concurrent programs. If the (simulated) execution of the implementation satisfies the property of interest (e.g., linearizability), then no verifier process reports an error, otherwise some verifier process reports an error and provides a witness. At first glance, there is a simple proof that linearizability is impossible to verify at run-time for some object. However, looking more closely, one can define a "stretched" version of the original implementation through which the linearizability of the original implementation can be tested indirectly. By-products of this work are simple methodologies to derive fault-tolerant distributed monitors and self-enforced linearizable implementations.

Nathalie Bertrand spoke about a CEGAR approach to parameterized verification of distributed algorithms, where CEGAR stands for counter-example-guided abstraction refinement. Ben-Or's randomized consensus algorithm for Byzantine failures was used as a running example throughout the talk, with randomization replaced by nondeterminism. It is described using three parameters, the number of processes n, the maximum number of faulty processes t, and the actual number of faulty processes f. She presented formal semantics for the algorithm and pointed out that there are two dimensions of infinity. Tools to overcome this difficulty include message abstraction and counting abstraction. Layered threshold automata (LTA) are used for counting abstraction. To get around the problem that parameterized model check of LTAs is undecidable, even just for safety properties, she used predicate abstraction via a guard automaton and CEGAR. The approach has been implemented in a tool PyLTA written in Python and that has been benchmarked with some promising results as well as some inconclusive cases. Future work includes formalizing model extraction from pseudocode and extending the approach to randomized algorithms in order to show, for example, almost-sure temrination of Ben-Or's algorithm.

Rotem Oshman's presentation was on truthful information dissemination in asynchronous networks. She was inspired to choose this topic by Maurice Herlihy's presentation. Suppose the network is composed of *rational* players who try to accomplish a goal but have their own incentives. The algorithm should be "incentive-compatible", meaning that following the algorithm is an equilibrium, with no coalitions: if all players but one follow the algorithm, then that player has no incentive to deviate. Consider the problem of getting all players to output every player's input, where the network graph is known, called the information dissemination problem. Much prior work has focused on "fair" solutions (if many solutions are possible, then choose one uniformly at random). Her work focuses on "good" solutions. First she explained how to set up the information dissemination problem as a Bayesian game

and mentioned that her work shows that the graph must be (at least) 2-connected. The main part of the talk was a description of an algorithm for solving the problem in a ring, in which players 0 and 1 learn each other's inputs and commit to a random string, then players learn the other inputs using the random string, and finally all the inputs are revealed and any cheaters are caught. She also outlined the methodology for proving the correctness and mentioned that the algorithm has optimal bit complexity. Future work includes handling coalitions and Byzantine players, as well as relaxing some of the technical assumptions.

Pierre Fraignaud presented a wait-free speedup theorem, which can be used for proving lower bounds. Suppose we could show that there exists a function F such that for every nonnegative t and every problem Π, Π has complexity t if and only if $F(\Pi)$ has complexity $t-1$. Such a theorem would imply that Π has complexity t if and only if $F^{(t)}(\Pi)$ has complexity 0. Then if we can show that the transformed problem $F^{(t)}(\Pi)$ cannot be solved with complexity 0, that would imply that the original problem Π cannot be solved with complexity t. The inspiration includes Linial's round lower bound for 3-coloring in a ring as well as more recent results for the anonymous LOCAL model and for maximal matchings and maximal independent set. The general questions are which models admit speedup theorems and which problems admit speedup theorems for a specific model. Using diagrams to provide intuition, he presented a result for the iterated immediate snapshot model, in which asynchronous crash-prone processes communicate through levels of shared registers that provide operations to write and obtain atomic snapshots. The generic transformation F is instantiated with a specific function called the *closure*, which has a slightly larger set of outputs and requires solving a local task in one round. The (weak) speedup theorem obtained is that if Π is solvable in t rounds, then the closure of Π is solvable in $t-1$ rounds. One application is that the closure of the consensus problem is just the consensus problem, and thus the easily-shown fact that consensus is not solvable in 0 rounds implies that consensus is not solvable at all. Another application is that the closure of ϵ-agreement (approximate agreement where decisions must be within ϵ) is (2ϵ)-agreement, which implies a lower bound of $\lceil \log_2 \frac{1}{\epsilon} \rceil$ for ϵ-agreement. The theorem extends to the use of test-and-set and binary consensus objects. However, the closure of set-agreement is trivial, as it can be solved in 0 rounds (because the theorem is not in both directions). Open questions include identifying which tasks have non-trivial closures, is there an if-and-only-if speedup theorem for asynchronous wait-free computing, and which models allow for the design of useful speedup theorems (e.g., what about t-resilient models).

Sandeep Kulkarni's talk was on using informal methods for distributed computing, in particular, the "war" of consistency violations and self-stabilization. A classic model of distributed computing lets each node see its neighbors' states to decide on its next action, assuming interleaving semantics. Local mutual exclusion provides concurrency in a large system with many processes and small neighborhoods, but it has significant overhead. He want to explore what happens if we don't use local mutual exclusion, and instead allow the resulting consistency violations to occur. In particular, a node can see neighbor states that are stale to different degrees. He then related this behavior to self-stabilization, a form of fault-tolerance that requires a program to eventually reach, and remain in, a set of "legal" states, no matter what state it begins in. The intuition is that if the rate of consistency violations is very high, then the program may not converge; if the rate is low, then the program will (probabilistically) converge, perhaps requiring more steps but without the need for local mutual exclusion or other synchronization mechanisms. He described several case studies (Dijkstra's 3-state token ring, graph coloring, and maximal matching) and made several observations. One is that the benefit of a program transition

has exponential distribution. Another is that the cost of a consistency violation transition has exponential distribution. Prior work related to consistency violations includes that on eventually consistent shared memory and that on lattice linearity. In conclusion, systems that tolerate consistency violations have the potential to benefit from the currency; although arbitrary programs may not tolerate such violations, self-stabilizing programs automatically do. Finally the performance of stabilizing programs can be predicted analytically in the presence of consistency violations.

Hagit Attiya spoke about approximately preserving hyper-properties without strong linearizability. When programming with (atomic abstract) objects that are implemented from other objects, a popular consistency condition for the implementation is linearizability, as it preserves trace properties of the program. However, it does not preserve hyper-properties, which include probability distributions. She gave an example of a toy program that terminates with probability at least one-half when it uses atomic registers but that can be made to never terminate if the registers are implemented with the well-known ABD algorithm. Strong linearizability (SL), in which linearization points are prefix-preserved, does preserve probability distributions. She then showed how SL is an example of *strong refinement*, which preserves hyper-properties and is equivalent to the existence of a forward simulation. Unfortunately, numerous objects do not have SL implementations. She then presented a solution to this problem for object implementations that are "tail strongly linearizable" and have "effect-free preambles"; informally this means that there is a prefix of the implementation that does not impact concurrent operations and after which its linearization point is chosen in a prefix-preserving way. The preamble is repeated some number of times and then one of the iterations is chosen at random to be used in the tail. This technique can be applied to several well-known object implementations. She presented a formula showing that the probability of a bad outcome with the transformed object approaches that with atomic objects as the number of preamble iterations increases. Regarding the extra cost incurred, she gave an example of applying the transformation to a specific atomic snapshot implementation for n processes resulting in a wait-free SL implementation taking $O(n^3)$ steps per process, and compared it to a previously known non-wait-free SL implementation that takes $O(n^3)$ steps per process. During the Q&A, the question was asked whether there exist implementations or even objects that cannot be improved in this way.

Friday, Dec. 9

A wrap-up session was held in the morning, during which suggestions for improvement, in case there is a follow-up seminar, were solicited from attendees. The main recommendations were to include more junior researchers, especially PhD students, to have some tools tutorials (e.g., on TLA+), and to put out a call to the distributed computing community requesting open challenges for verification.

4 Panel discussions

4.1 Panel: Applications in Industry

Giuliano Losa (Stellar Development Foundation – San Francisco, US)

License Creative Commons BY 4.0 International license
© Giuliano Losa

The panelists started by each giving a 10 minute talk giving an overview of the use of formal methods at their respective employers. A lively discussion on areas of improvements and missing tools ensued, with emphasis on what academics can do to help.

The panelists were Serdar Tasiran (Principal Scientist, Amazon S3), Akash Lal (Senior Principal Researcher, Microsoft Research), Manuel Bravo (Informal Systems), and Giuliano Losa (Stellar Development Foundation). Cezara Dragoi (Principal Applied Scientist, Amazon) and Bernhard Kragl (Applied Scientist, Amazon) also joined the conversation virtually.

The audience learned that formal methods have made their way in the software development process in many teams at Amazon, Microsoft, and Informal Systems, while the Stellar Development Foundation is investing is more targeted use-cases such as the development of distributed algorithms and smart contracts. Languages and tools such as TLA+, P, Coyote, CBMC, and Ivy are helping software engineers at those companies develop better systems. Moreover, there is heavy investment in new tooling.

The panel identified compositional verification of large systems as an important area in which tooling, best practices, and even scientific experiments identifying the important problems are missing. Academics from the audience highlighted the difficulty of working on large-scale code bases (or even having access to them) in an academic setting where engineering resources are often scarce.

Finally, the panel discussed training software engineers in formal methods. It transpired that typical undergraduate software-engineering classes do not address formal methods, and that the material currently available to train engineers on the job may be insufficient or not accessible to most software engineers because they do not have the prerequisite knowledge.

5 Talk Abstracts

5.1 (Approximately) Preserving Hyper-Properties without Strong Linearizability

Hagit Attiya (Technion – Haifa, IL)

License Creative Commons BY 4.0 International license
© Hagit Attiya
Joint work of Hagit Attiya, Constantin Enea, and Jennifer L. Welch

Atomic shared objects, whose operations take place instantaneously, are a powerful abstraction for designing complex concurrent programs. Since they are not always available, they are typically substituted with software implementations. A prominent condition relating these implementations to their atomic specifications is linearizability, which preserves safety properties of the programs using them. However linearizability does not preserve hyper-properties, which include probabilistic guarantees of randomized programs: an adversary can greatly amplify the probability of a bad outcome, such as nontermination, by manipulating the order of events inside the implementations of the operations. This unwelcome behavior

prevents modular reasoning, which is the key benefit provided by the use of linearizable object implementations. A more restrictive property, strong linearizability, does preserve hyper-properties but it is impossible to achieve in many situations.

This paper suggests a novel approach to blunting the adversary's additional power that works even in cases where strong linearizability is not achievable. We show that a wide class of linearizable implementations, including well-known ones for registers and snapshots, can be modified to approach the probabilistic guarantees of randomized programs when using atomic objects. The technical approach is to transform the algorithm of each operation of an existing linearizable implementation by repeating a carefully chosen prefix of the operation several times and then randomly choosing which repetition to use subsequently. We prove that the probability of a bad outcome decreases with the number of repetitions, approaching the probability attained when using atomic objects. The class of implementations to which our transformation applies includes the ABD implementation of a shared register using message-passing, the Afek et al. implementation of an atomic snapshot using single-writer registers, the Vitanyi and Awerbuch implementation of a multi-writer register using single-writer registers, and the Israeli and Li implementation of a multi-reader register using single-reader registers, all of which are widely used in asynchronous crash-prone systems.

5.2 Verification of Liveness Properties on Weakly Consistent Platforms (TSO as an example)

Parosh Aziz Abdulla (Uppsala University, SE)

License Creative Commons BY 4.0 International license
© Parosh Aziz Abdulla

We present Probabilistic Total Store Ordering (PTSO) – a probabilistic extension of the classical TSO semantics. For a given (finite-state) program, the operational semantics of PTSO induces an infinite-state Markov chain. We resolve the inherent non-determinism due to process schedulings and memory updates according to given probability distributions. We provide comprehensive results showing the decidability of several properties for PTSO. (i) Almost-Sure (Repeated) Reachability: whether a run, starting from a given initial configuration, almost surely visits (resp. almost surely repeatedly visits) a given set of target configurations. (ii) Almost-Never (Repeated) Reachability: whether a run from the initial configuration almost never visits (resp. almost never repeatedly visits) the target. (iii) Approximate Quantitative (Repeated) Reachability: to approximate, up to an arbitrary degree of precision, the measure of runs that start from the initial configuration and (repeatedly) visit the target. (iv) Expected Average Cost: To approximate the expected average run cost from the initial configuration to the target up to an arbitrary degree of precision.

We derive our results through a nontrivial combination of results from the classical theory of (infinite-state) Markov chains, the theories of decisive and eager Markov chains, specific techniques from combinatorics, as well as, decidability and complexity results for the classical (non-probabilistic) TSO semantics. As far as we know, this is the first work considering probabilistic verification of programs running on weak memory models.

5.3 A CEGAR approach to parameterized verification of distributed algorithms

Nathalie Bertrand (INRIA – Rennes, FR)

License Creative Commons BY 4.0 International license
© Nathalie Bertrand
Joint work of Nathalie Bertrand, Ocan Sankur, Bastien Thomas, Josef Widder

Distributed algorithms are central to many domains such as scientific computing, telecommunications and the blockchain. Even when they aim at performing simple tasks, their behaviour is hard to analyze, due to the presence of faults (crashes, message losses, etc.) and to the asynchrony between the processes. Parameterized verification techniques have been developed to prove correctness of distributed algorithms independently of actual setup, i.e. the number of processes and the potential failures.

In this talk, we present a CEGAR approach to checking safety and liveness properties for fault tolerant distributed algorithms that use threshold conditions, typically on the number of received messages of a given type.

5.4 Highly-available access control in distributed systems

Annette Bieniusa (TU Kaiserslautern, DE)

License Creative Commons BY 4.0 International license
© Annette Bieniusa

Highly available distributed systems rely on replication for partition- and fault-tolerance. This results in weaker consistency guarantees for shared data and introduces challenges for the correctness of the application under (temporary) data inconsistencies. In particular regarding application security, it is difficult to determine which inconsistencies can be tolerated and which might lead to security breaches. In this talk, we will introduce state-of-the-art approach to enforcing dynamic access control policies in highly-available systems. As use case, we discuss the interplay of security and consistency in distributed file systems and provide an impossibility result that indicates that confidentiality, integrity and accessibility cannot be achieved together in a highly-available partition-tolerant setting. We further discuss a CRDT-based model, implementing the traditional POSIX access control policy, that guarantees confidentiality and integrity while precluding accessibility only in rare situations while reflecting the users' intention.

5.5 Verification of Distributed Systems

Ahmed Bouajjani (Université Paris Cité, FR)

License Creative Commons BY 4.0 International license
© Ahmed Bouajjani

The talk presents explains the problems to address for the verification of distributed systems, in particular problems related to consistency and isolation levels. We present formal framework for specifying consistency models and isolation levels (in the transactional case). Then, we describe existing approaches and results concerning (1) the verification of application running over weak consistency environments, and (2) the verification that system implementation conform to consistency/isolation levels.

5.6 Asynchronous Distributed Runtime Verification and Enforcement of Linearizability

Armando Castaneda (National Autonomous University of Mexico, MX)

License Creative Commons BY 4.0 International license
© Armando Castaneda

This work studies the problem of distributed runtime verification of linearizability for asynchronous concurrent implementations. It proposes an interactive model for distributed runtime verification and shows that it is impossible to runtime verify this correctness condition for some common sequential objects such as queues, stacks, sets, priority queues, counters and the consensus problem. The impossibility captures informal arguments used in the past that argue distributed runtime verification is impossible. Then, it argues that linearizability can be indirectly verified through a class of implementations. Namely, it shows that (1) linearizability of a class of concurrent implementations can be distributed runtime verified and (2) every implementation can be easily transformed to its counterpart that belongs to the class. From these algorithms, it is easy to derive distributed monitors, as well as concurrent implementations that self-enforce linearizability, namely, these implementations produces outputs that are guaranteed to be linearizable. The same results hold for extensions of linearizability such as set-linearizability and interval-linearizability.

5.7 Synchronizer: a recipe for building correct algorithms under partial synchrony

Gregory Chockler (University of Surrey – Guildford, GB)

License Creative Commons BY 4.0 International license
© Gregory Chockler

Joint work of Manuel Bravo, Gregory V. Chockler, Alexey Gotsman, Alejandro Pastoriza
Main reference Manuel Bravo, Gregory V. Chockler, Alexey Gotsman: "Liveness and Latency of Byzantine State-Machine Replication", in Proc. of the 36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA, LIPIcs, Vol. 246, pp. 12:1–12:19, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
URL https://doi.org/10.4230/LIPIcs.DISC.2022.12

Byzantine state-machine replication (SMR) ensures the consistency of replicated state in the presence of malicious replicas and lies at the heart of the modern blockchain technology. Byzantine SMR protocols often guarantee safety under all circumstances and liveness only under synchrony. However, guaranteeing liveness even under this assumption is nontrivial. So far we have lacked systematic ways of incorporating liveness mechanisms into Byzantine SMR protocols, which often led to subtle bugs. To close this gap, we introduce a modular framework to facilitate the design of provably live and efficient Byzantine SMR protocols. Our framework relies on a view abstraction generated by a special SMR synchronizer primitive to drive the agreement on command ordering. We present a simple formal specification of an SMR synchronizer and its bounded-space implementation under partial synchrony. We also apply our specification to prove liveness and analyze the latency of three Byzantine SMR protocols via a uniform methodology. In particular, one of these results yields what we believe is the first rigorous liveness proof for the algorithmic core of the seminal PBFT protocol.

5.8 Hyperproperties in Synthesis and Verification

Bernd Finkbeiner (CISPA – Saarbrücken, DE)

License Creative Commons BY 4.0 International license
© Bernd Finkbeiner

Traditionally, most verification efforts have focused on the satisfaction of trace properties, such as that an assertion is satisfied at a particular program location or that the computation terminates eventually. Many policies from information-flow security, like observational determinism or noninterference, and many other system properties including promptness and knowledge can, however, not be expressed as trace properties, because these properties are hyperproperties, i.e., they relate multiple execution traces. In this talk, I will give an overview on logics for the specification of hyperproperties and on algorithms and tools for verification and synthesis.

5.9 Wait-Free Speedup Theorem

Pierre Fraigniaud (Université de Paris, FR & CNRS, Paris, FR)

License Creative Commons BY 4.0 International license
© Pierre Fraigniaud

Joint work of Pierre Fraigniaud, Ami Paz, Sergio Rajsbaum
Main reference Pierre Fraigniaud, Ami Paz, Sergio Rajsbaum: "A Speedup Theorem for Asynchronous Computation with Applications to Consensus and Approximate Agreement", in Proc. of the PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 – 29, 2022, pp. 460–470, ACM, 2022.
URL https://doi.org/10.1145/3519270.3538422

We study two fundamental problems of distributed computing, consensus and approximate agreement, through a novel approach for proving lower bounds and impossibility results, that we call the asynchronous speedup theorem. For a given n-process task Pi and a given computational model M, we define a new task, called the closure of Pi with respect to M. The asynchronous speedup theorem states that if a task Pi is solvable in $t > 0$ rounds in M, then its closure w.r.t. M is solvable in $t - 1$ rounds in M. We prove this theorem for iterated models, as long as the model allows solo executions. We illustrate the power of our asynchronous speedup theorem by providing a new proof of the wait-free impossibility of consensus using read/write registers, and a new proof of the wait-free impossibility of solving consensus using registers and test&set objects for $n > 2$. The proof is merely by showing that, in each case, the closure of consensus (w.r.t. the corresponding model) is consensus itself. Our main application is the study of the power of additional objects, namely test&set and binary consensus, for wait-free solving approximate agreement faster. By analyzing the closure of approximate agreement w.r.t. each of the two models, we show that while these objects are more powerful than read/write registers from the computability perspective, they are not more powerful as far as helping solving approximate agreement faster is concerned.

5.10 Correctness Conditions for Cross-chain Transactions

Maurice Herlihy (Brown University – Providence, US)

License Creative Commons BY 4.0 International license
© Maurice Herlihy
Joint work of Maurice Herlihy, Barbara Liskov, Liuba Shrira

Modern distributed data management systems face a new challenge: how can autonomous, mutually-distrusting parties cooperate safely and effectively? Addressing this challenge brings up questions familiar from classical distributed systems: how to combine multiple steps into a single atomic action, how to recover from failures, and how to synchronize concurrent access to data. Nevertheless, each of these issues requires rethinking when participants are autonomous and potentially adversarial.

5.11 Testing Blockchain Consensus Algorithms

Burcu Kulahcioglu Ozkan (TU Delft, NL)

License Creative Commons BY 4.0 International license
© Burcu Kulahcioglu Ozkan

Byzantine fault-tolerant algorithms promise agreement on a correct value, even if a subset of processes can deviate from the algorithm arbitrarily. While these algorithms provide strong guarantees in theory, in practice, protocol bugs and implementation mistakes may still cause them to go wrong. This talk introduces a simple yet effective method and our recent experience for automatically finding errors in implementations of Byzantine fault-tolerant algorithms through randomized testing. Our approach navigates the space of possible process faults by limiting process faults in an execution to a bounded number of round-based, structure-preserving, small-scope mutations to the protocol messages.

5.12 War of Consistency Violations and Self-Stabilization

Sandeep Kulkarni (Michigan State University – East Lansing, US)

License Creative Commons BY 4.0 International license
© Sandeep Kulkarni
Joint work of Sandeep Kulkarni, Duong Nguyen, Arya Gupta

Consider distributed programs that are designed in a strong memory model (e.g., one that allows a node to read its neighbor atomically). When such program is implemented without synchronization requirements such as local mutual exclusion, they can result in consistency violations (cvfs). Cvfs can cause the program to violate the specification. However, when combined with self-stabilization, we get a war between cvf and stabilization; the fore pushes the program away from the invariant whereas the latter tries to get closer.

We observe the following for various self-stabilizing programs (1) permitting cvfs provides a better performance, (2) benefit of program transition and cost of cvfs follow an exponential distribution, (3) Cvf cost distribution is independent of the number of processes, and (4) Cost of cvf can be computed by sampling the state space.

5.13 Correctness in Distributed Computing

Petr Kuznetsov (Telecom Paris, FR)

License Creative Commons BY 4.0 International license
© Petr Kuznetsov

We take a walk through the space of correctness criteria that relate concurrent behavior of a distributed system to a sequential specification. In particular, we focus on safety and liveness, linearizability, sequential consistency, progress conditions (from deadlock-freedom to obstruction-freedom). We then discuss the cost of achieving correctness and ways to achieve this kind of correctness, and then highlight several ways to cur the costs by relaxing correctness reqauirements.

5.14 Abstraction for Crash-Resilient Objects

Ori Lahav (Tel Aviv University, IL)

License Creative Commons BY 4.0 International license
© Ori Lahav
Joint work of Artem Khyzha, Ori Lahav

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.

5.15 Random Testing with Theoretical Guarantees

Rupak Majumdar (MPI-SWS – Kaiserslautern, DE)

License Creative Commons BY 4.0 International license
© Rupak Majumdar

We describe an algorithm called PCTCP for random testing of distributed systems. We show that the properties of the algorithm can be understood by looking at online dimension of the underlying partial order of events, which is related to online chain partitioning. We define d-hitting families as a way to organize behaviors. We show that the PCTCP algorithm gives a guarantee on the minimal probability of sampling behaviors from d-hitting families for a fixed d.

5.16 Tool Support for TLA+: TLC, Apalache, and TLAPS

Stephan Merz (INRIA Nancy – Grand Est, FR)

License Creative Commons BY 4.0 International license
© Stephan Merz
Joint work of Igor Konnov, Markus Kuppe, Stephan Merz
Main reference Igor Konnov, Markus Kuppe, Stephan Merz: "Specification and Verification with the TLA+ Trifecta: TLC, Apalache, and TLAPS", CoRR, Vol. abs/2211.07216, 2022.
URL https://doi.org/10.48550/arXiv.2211.07216

Using an algorithm due to Safra for distributed termination detection as a running example, we present the main tools for verifying specifications written in TLA+. Examining their complementary strengths and weaknesses, we suggest a workflow that supports different types of analysis and that can be adapted to the desired degree of confidence.

Our TLA+ specifications and proofs are available at `https://github.com/tlaplus/Examples/tree/ISoLA2022/specifications/ewd998`.

5.17 On Graph Connectivity in Distributed Algorithms

Yoram Moses (Technion – Haifa, IL)

License Creative Commons BY 4.0 International license
© Yoram Moses

Knowledge and Topological reasoning capture aspects of local and global information that are essential in a distributed setting. This talk will present a simple problem whose analysis uses graph connectivity in an essential way, and leverage this to discuss how the connection between common knowledge and connectivity has been exploited in the literature. Finally, we present some thoughts regarding the interaction between epistemic analysis and more topological analysis of distributed algorithms and consider future challenges.

5.18 Truthful information Dissemination in Asynchronous Networks

Rotem Oshman (Tel Aviv University, IL)

License Creative Commons BY 4.0 International license
© Rotem Oshman
Joint work of Lior Solodkin, Rotem Oshman
Main reference Lior Solodkin, Rotem Oshman: "Truthful Information Dissemination in General Asynchronous Networks", in Proc. of the 35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference), LIPIcs, Vol. 209, pp. 37:1–37:19, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
URL https://doi.org/10.4230/LIPIcs.DISC.2021.37

We give a protocol for information dissemination in asynchronous networks of rational players, where each player may have its own desires and preferences as to the outcome of the protocol, and players may deviate from the protocol if doing so achieves their goals. We show that under minimalistic assumptions, it is possible to solve the information dissemination problem in a truthful manner, such that no participant has an incentive to deviate from the protocol we design. Our protocol works in any asynchronous network, provided the network graph is at least 2-connected. We complement the protocol with two impossibility results, showing that 2-connectivity is necessary, and also that our protocol achieves optimal bit complexity. As an application, we show that truthful information dissemination can be used to implement

a certain class of communication equilibria, which are equilibria that are typically reached by interacting with a trusted third party. Recent work has shown that communication equilibria can be implemented in synchronous networks, or in asynchronous, complete networks; we show that in some useful cases, our protocol yields a lightweight mechanism for implementing communication equilibria in any 2-connected asynchronous network.

5.19 Extending Intel-x86 Consistency and Persistency

Azalea Raad (Imperial College London, GB)

License Creative Commons BY 4.0 International license
© Azalea Raad
Main reference Azalea Raad, Luc Maranget, Viktor Vafeiadis: "Extending Intel-x86 consistency and persistency: formalising the semantics of Intel-x86 memory types and non-temporal stores", Proc. ACM Program. Lang., Vol. 6(POPL), pp. 1–31, 2022.
URL https://doi.org/10.1145/3498683

Existing semantic formalisations of the Intel-x86 architecture cover only a small fragment of its available features that are relevant for the consistency semantics of multi-threaded programs as well as the persistency semantics of programs interfacing with non-volatile memory.

We extend these formalisations to cover: (1) non-temporal writes, which provide higher performance and are used to ensure that updates are flushed to memory; (2) reads and writes to other Intel-x86 memory types, namely uncacheable, write-combined, and write-through; as well as (3) the interaction between these features. We develop our formal model in both operational and declarative styles, and prove that the two characterisations are equivalent. We have empirically validated our formalisation of the consistency semantics of these additional features and their subtle interactions by extensive testing on different Intel-x86 implementations. Our work on validating the persistency semantics is ongoing.

5.20 Formal Methods for Distributed Systems at Amazon Simple Storage Service (S3)

Serdar Tasiran (Amazon Web Services – New York City, US)

License Creative Commons BY 4.0 International license
© Serdar Tasiran

Amazon's Simple Storage Service (S3) is increasingly adopting a "model first" approach, with formal models being first-class artifacts in the software development process. In this approach, we start by model checking or deductively proving that design models of distributed systems provide key properties such as durability or strong consistency. Then, during integration testing, gamma stages, or even in production, we monitor that the implementation code conforms to the design models. This not only detects/prevents implementation errors, but also forces models and implementations to remain in sync, ensuring that the investments in writing and analyzing models continue to pay off. In this talk, I will present several examples of the model-first approach in S3, and challenges of using formal methods at S3 scale.

5.21 Implementing Shared Objects in the Presence of Continual Churn

Jennifer L. Welch (Texas A&M University – College Station, US)

License Creative Commons BY 4.0 International license
© Jennifer L. Welch
Main reference Hagit Attiya, Sweta Kumari, Archit Somani, Jennifer L. Welch: "Store-collect in the presence of continuous churn with application to snapshots and lattice agreement", Inf. Comput., Vol. 285(Part), p. 104869, 2022.
URL https://doi.org/10.1016/j.ic.2022.104869

I will present a model for dynamic distributed systems that permits a bounded amount of ongoing churn as well as node crashes. In this model I will present an algorithm for simulating a linearizable shared read-write register and a non-linearizable shared store-collect object, as well as a lower bound on the crash-resilience that is possible. The talk is based on the papers.

References

1 Attiya, Chung, Ellen, Kumar, and Welch, "Emulating a Shared Register in a System that Never Stops Changing", IEEE Transactions on Parallel and Distributed Systems, Vol. 30, Issue 3, March 2019 (DOI: 10.1109/TPDS.2018.2867479)
2 Attiya, Kumar, Somani, and Welch, "Store-collect in the presence of continuous churn with application to snapshots and lattice agreement," Information and Computation, Vol. 285, Part B, May 2022 (doi.org/10.1016/j.ic.2022.104869)

5.22 Reasoning Principles for Verifying Concurrent Search Structures

Thomas Wies (New York University, US)

License Creative Commons BY 4.0 International license
© Thomas Wies
Joint work of Thomas Wies, Siddharth Krishna, Nisarg Patel, Dennis Shasha, Roland Meyer, Sebastian Wolff

Search structures support the fundamental data storage primitives on key-value pairs: insert a pair, delete by key, search by key, and update the value associated with a key. Concurrent search structures are parallel algorithms to speed access to search structures on multicore and distributed servers. These sophisticated algorithms perform fine-grained synchronization between threads, making them notoriously difficult to design correctly. In this talk, I will present a framework for obtaining correctness proofs for concurrent search structures that are modular, reusable, and amenable to automation. The framework takes advantage of recent advances in local reasoning techniques based on concurrent separation logic. I will provide an overview of these techniques and demonstrate there use for verifying realistic search structures such as concurrent B-link trees.

5.23 Predictable Building Blocks for Randomized Shared Memory Algorithms

Philipp Woelfel (University of Calgary, CA)

License Creative Commons BY 4.0 International license
© Philipp Woelfel

In this talk I will discuss why linearizability is not a sufficient condition for building blocks used in randomized shared memory algorithms. I will present strong linearizability (introduced by Golab, Higham and Woelfel, 2011), which can be used to preserve the power of the adaptive adversary when replacing atomic operations with implemented ones in randomized algorithms. I will then outline ideas for implementing a wait-free strongly linearizable load-linked/store conditional object from compare-and-swap.

Participants

- Hagit Attiya
Technion – Haifa, IL
- Parosh Aziz Abdulla
Uppsala University, SE
- Nathalie Bertrand
INRIA – Rennes, FR
- Raven Beutner
CISPA – Saarbrücken, DE
- Annette Bieniusa
TU Kaiserslautern, DE
- Ahmed Bouajjani
Université Paris Cité, FR
- Manuel Bravo
Informal Systems – Madrid, ES
- Armando Castaneda
National Autonomous University of Mexico, MX
- Gregory Chockler
University of Surrey – Guildford, GB
- Constantin Enea
Ecole Polytechnique – Palaiseau, FR & CNRS – Palaiseau, FR
- Bernd Finkbeiner
CISPA – Saarbrücken, DE
- Pierre Fraigniaud
Université de Paris, FR & CNRS, Paris, FR
- Alexey Gotsman
IMDEA Software Institute – Madrid, ES
- Maurice Herlihy
Brown University – Providence, US
- Burcu Kulahcioglu Ozkan
TU Delft, NL
- Sandeep Kulkarni
Michigan State University – East Lansing, US
- Petr Kuznetsov
Telecom Paris, FR
- Ori Lahav
Tel Aviv University, IL
- Giuliano Losa
Stellar Development Foundation – San Francisco, US
- Rupak Majumdar
MPI-SWS – Kaiserslautern, DE
- Stephan Merz
INRIA Nancy – Grand Est, FR
- Yoram Moses
Technion – Haifa, IL
- Rotem Oshman
Tel Aviv University, IL
- Azalea Raad
Imperial College London, GB
- Sergio Rajsbaum
National Autonomous University of Mexico, MX, on leave at IRIF, Paris, FR
- Ana Sokolova
University of Salzburg, AT
- Serdar Tasiran
Amazon Web Services – New York City, US
- Viktor Vafeiadis
MPI-SWS – Kaiserslautern, DE
- Jennifer L. Welch
Texas A&M University – College Station, US
- Thomas Wies
New York University, US
- Philipp Woelfel
University of Calgary, CA