Abstract

The Dagstuhl Seminar “Formal Methods and Fault-Tolerant Distributed Computing: Forging an Alliance” took place May 22-25, 2018. Its goal was to strengthen the interaction between researchers from formal methods and from distributed computing, and help the two communities to better identify common research challenges.

1 Executive Summary

Anca Muscholl (University of Bordeaux, FR)
Javier Esparza (TU München, DE)
Pierre Fraigniaud (University Paris-Diderot and CNRS, FR)
Sergio Rajsbaum (National Autonomous University of Mexico, MX)

The original motivation of this workshop has to do with the evolution of research in Computer Science. The first ACM conference on Principles of Distributed Computing (PODC) was held in 1982. The proceedings of its first editions included papers on distributed algorithms\(^1\), formal methods for distributed systems\(^2\), or a combination of the two. However, in 1990 the area of formal methods for distributed computing branched out, and started its own conference, the International Conference on Concurrency Theory (CONCUR), now in its 27th edition. PODC and CONCUR have become the premier conferences in their respective fields, and, after over 20 years of almost independent evolution, feel the need to close a gap

---

\(^1\) Algorithms designed to run on computer hardware constructed from interconnected processors.

\(^2\) Mathematically based techniques for the specification, development and verification of software and hardware systems.

that slows down progress, limits the applicability of the results, and causes repetitions and inconsistencies.

Our seminar aimed at achieving synergy by bringing together the two research areas, both with deep understanding of distributed computation, but different perspectives. We had two longer tutorials, one about concurrent data structures by Ph. Woelfel and one about verification of concurrent programs by A. Bouajjani. In addition, we had several survey talks, on correctness in concurrent programming (H. Attiya), distributed runtime verification (B. Bonakdarpour), distributed property testing (K. Censor-Hillel), distributed synthesis (B. Finkbeiner), and parametrized verification (I. Konnov).

The scientific programme was quite dense, given that we had only 4 days and almost all participants proposed to give a talk. Exchanges were very lively, and the discussion that we had with all participants showed that this kind of workshop is a great opportunity to compare our approaches and find new research directions, inspired by the perspectives of the other community. We warmly thank Marie Fortin for the editorial work on this report and the Dagstuhl staff for the excellent conditions provided for our seminar.
# Table of Contents

## Executive Summary

*Anca Muscholl, Javier Esparza, Pierre Fraigniaud, and Sergio Rajsbaum*  

## Overview of Talks

- **On Verifying Robustness of Concurrent Systems**  
  *Ahmed Bouajjani*  
  
- **Visual/interactive design of fault-tolerant distributed algorithms**  
  *Paul C. Attie*  
  
- **Formal Analysis of Population Protocols**  
  *Michael Blondin*  
  
- **Synthesis of Distributed Systems from Logical Specifications**  
  *Benedikt Bollig and Marie Fortin*  
  
- **Automated Fine-Tuning of Probabilistic Self-Stabilizing Algorithms**  
  *Borzoo Bonakdarpour*  
  
- **Tutorial: Distributed Runtime Verification**  
  *Borzoo Bonakdarpour*  
  
- **Distributed Property Testing**  
  *Keren Censor-Hillel*  
  
- **Parameterized Verification of Topology-sensitive Distributed Protocols**  
  *Giorgio Delzanno*  
  
- **Communication-closed asynchronous protocols**  
  *Cezara Dragoi*  
  
- **Verification of a Fault-Tolerant Cache-Coherency Protocol**  
  *Jo Ebergen*  
  
- **Distributed Monitoring of Controlled Events**  
  *Yuval Emek*  
  
- **Specifying and Verifying Concurrent Objects**  
  *Constantin Enea*  
  
- **Distributed Synthesis**  
  *Bernd Finkbeiner*  
  
- **Faithful Delay Models in Circuits and Distributed Systems**  
  *Matthias Függer*  
  
- **Indistinguishability: Friend and Foe in Concurrent Programming**  
  *Hagit Attiya*  
  
- **Cutoff Results for Parameterized Verification and Synthesis**  
  *Swen Jacobs*  
  
- **What my computer can find about your distributed algorithm**  
  *Igor Konnov*  
  
- **Synthesizing Thresholds for Fault-Tolerant Distributed Algorithms**  
  *Marijana Lazic*
Breaking and (Partly) Fixing Pastry
Stephan Merz .......................................................... 72

Indistinguishability, Duality, and Coordination
Yoram Moses .......................................................... 73

Interactive Distributed Proofs
Rotem Oshman .......................................................... 73

Proof-Labeling Schemes: Broadcast, Unicast and In Between
Mor Perry ................................................................. 73

Pretend Synchrony- some distributed computing approaches
Sergio Rajsbaum ....................................................... 74

Biased Clocks: A way to Improve Effectiveness of Run Time Monitoring of Distributed Systems
Sandeep S. Kulkarni ..................................................... 75

Playing with scheduling policies
Arnaud Sangnier .......................................................... 75

Linearizability via Order-extension Results
Ana Sokolova .............................................................. 76

Model checking of incomplete systems
Paola Spoletini ........................................................... 76

Distributed Encoding of the Integers
Corentin Travers .......................................................... 77

Towards verification of distributed algorithms in the Heard-of model
Igor Walukiewicz .......................................................... 78

(Strong) Linearizability – A Tutorial
Philipp Woelfel ............................................................ 78

Participants ................................................................. 79
3 Overview of Talks

3.1 On Verifying Robustness of Concurrent Systems

Ahmed Bouajjani (University Paris-Diderot, FR)

Concurrent systems are in general used by their clients under strong assumptions on their visible behaviors. This allows a modular design approach: at the level of the client, these assumptions allow to reason in an abstract way about the behaviors of the invoked systems.

For instance, the users of a shared memory may assume that the implementation of the memory is sequentially consistent, which means that it behaves according to the standard interleaving model where write/read operations are considered to be atomic, and immediately visible to all parallel users. In another context, the users of web services may consider that their requests are handled atomically in a serial way, and in yet another context, the designers of protocols and distributed algorithms may consider that interactions between components are happening in a synchronous way, etc.

However, for performance reasons, the implementations of concurrent systems tend to parallelize operations and to use various optimizations in order to increase the throughput of the system. This leads in general to relaxations in the semantics guaranteed by these implementations w.r.t. to strong consistency models. In this talk, we will address the issue of checking that a given program of the client is robust against this kind of relaxations, i.e., the observable behaviors of the client are the same under both the strong and relaxed consistency models. Robustness corresponds to a correctness criterion that ensures the preservation by the considered relaxations of all properties that can be proved assuming the strong consistency models.

We show that robustness can be checked efficiently in several cases by linear reductions to state reachability problems. These cases include robustness against the weak memory model TSO, and also checking robustness against concurrency and asynchrony in event-driven programs and message passing programs where we compare the behaviors of a same program under two different semantics, one being the asynchronous one, and the other one being a stronger semantics that is synchronous in some sense (that will be defined).

3.2 Visual/interactive design of fault-tolerant distributed algorithms

Paul C. Attie (American University of Beirut, LB)

I advocate the design and verification of fault-tolerant distributed algorithms via the direct manipulation of the state-transition relation. To deal with state explosion (in the finite state case), and with combinatoric explosion and infinite states (in the general case), I propose the following:
1. Pairwise composition: analyze the interaction of two processes at a time, to verify safety and liveness properties of process-pairs.
2. Small subsystems: analyze the postcondition of an action (in a small subsystem containing the action) to verify deadlock freedom.
3. Fault actions: model faults as actions which perturb the global state. Synthesize the needed recovery transitions.
5. Refine atomicity: use knowledge acquired by a process to replace test & set by atomic read/write.
6. Abstraction: equivalence relation on states specifies abstraction. Manipulate abstraction and then concretize.
7. Finitely representable infinite-state structure: a node labeled by a recursive predicate represents a set of states, a transition labeled by a guarded command represents an action.

I have implemented some of these methods, and am currently implementing the remainder, in the Eshmun tool, available at eshmuntool.blogspot.com. The combination of these methods enables rapid semantic feedback and interaction for the distributed algorithm designer. The use of methods to combat complexity is not only for computational reasons, but also for visualization reasons: it helps the designer visualize the behavior of the algorithm.

### 3.3 Formal Analysis of Population Protocols

**Michael Blondin (TU München, DE)**

License: Creative Commons BY 3.0 Unported license

© Michael Blondin

Joint work of Michael Blondin, Javier Esparza, Stefan Jaax, Antonín Kučera


URL: [http://dx.doi.org/10.1145/3209108.3209110](http://dx.doi.org/10.1145/3209108.3209110)

Population protocols are a model of distributed computation by anonymous mobile agents with little computational power. Such protocols allow for modeling systems such as networks of passively mobile sensors and chemical reaction networks. Agents of a population protocol interact by meeting at random. In well-designed protocols, for every initial configuration of agents and every computation starting from this configuration, all agents eventually agree on a consensus value.

In this talk, I will give an overview of recent advances on the formal analysis of population protocols. In particular, I will discuss the problem of automatically determining whether a protocol is correct, and the problem of computing an asymptotic bound on the expected time a protocol needs to reach consensus.
3.4 Synthesis of Distributed Systems from Logical Specifications

Benedikt Bollig (ENS – Cachan, FR) and Marie Fortin (ENS – Cachan, FR)

We are concerned with formally modeling and specifying distributed systems, with the aim of ensuring their correctness. As a system model, we consider communicating finite-state machines (CFMs), in which finite-state processes exchange messages through unbounded FIFO channels. On the specification side, we focus on the first-order logic of message sequence charts (MSCs). MSCs, also known as space-time diagrams, arise naturally as executions of CFMs and feature Lamport’s happened-before relation. First-order logic captures many interesting properties of distributed systems, and it subsumes various temporal logics. This presentation consists of two parts:

Part I: Logics over Message Sequence Charts (M. Fortin). In the first part, we study the expressive power of first-order logic, establish connections with temporal logics and propositional dynamic logic, and present a normal-form construction. As a corollary, we establish that first-order logic has the three-variable property.

Part II: From Logic to Communicating Finite-State Machines (B. Bollig). In the second part, we address the synthesis problem: Relying on the normal-form construction of Part I and a (nondeterministic) gossip protocol, we show that every first-order specification can be transformed into a CFM. The latter can then be considered as a system model that is correct by construction. Moreover, the translation is useful in the automata-theoretic approach to model checking distributed systems.

3.5 Automated Fine-Tuning of Probabilistic Self-Stabilizing Algorithms

Borzoo Bonakdarpour (McMaster University – Hamilton, CA)

Although randomized algorithms have widely been used in distributed computing as a means to tackle impossibility results, it is currently unclear what type of randomization leads to the best performance in such algorithms. In this talk, I propose automated techniques to find the probability distribution that achieves minimum average recovery time for an input randomized distributed self-stabilizing protocol without changing the behavior of the algorithm. Our first technique is based on solving symbolic linear algebraic equations in order to identify fastest state reachability in parametric discrete-time Markov chains. The second approach applies parameter synthesis techniques from probabilistic model checking to compute the rational function describing the average recovery time and then uses dedicated solvers to find the optimal parameter valuation. The third approach computes over- and under-approximations of the result for a given parameter region and iteratively refines the regions with minimal recovery time up to the desired precision. The latter approach finds sub-optimal solutions with negligible errors, but it is significantly more scalable in orders of magnitude as compared to the other approaches.
3.6 Tutorial: Distributed Runtime Verification

Borzoo Bonakdarpour (McMaster University – Hamilton, CA)

License Creative Commons BY 3.0 Unported license © Borzoo Bonakdarpour

This tutorial surveys the most prominent works on distributed runtime verification.

3.7 Distributed Property Testing

Keren Censor-Hillel (Technion – Haifa, IL)

License Creative Commons BY 3.0 Unported license © Keren Censor-Hillel

This survey talk will overview the recent achievements in the area of distributed property testing.

Background will be given on the computational model, the related distributed decision tasks, and the relaxations that allow overcoming expensive computations in settings of limited bandwidth, within a small number of local queries.

3.8 Parameterized Verification of Topology-sensitive Distributed Protocols

Giorgio Delzanno (University of Genova, IT)

Joint work of Giorgio Delzanno, Sylvain Conchon, Angelo Ferrando


We show that Cubicle, an SMT-based infinite-state model checker, can be applied as a verification engine for GLog, a logic-based specification language for topology-sensitive distributed protocols with asynchronous communication. Existential coverability queries in GLog can be translated into verification judgements in Cubicle by encoding relational updates as unbounded array transitions. We apply the resulting framework to automatically verify a distributed version of the Dining Philosopher mutual exclusion protocol formulated for an arbitrary number of nodes and communication buffers.

3.9 Communication-closed asynchronous protocols

Cezara Dragoi (ENS – Paris, FR)

Joint work of Cezara Dragoi, Josef Widder

Communication closed round-based models are a particular type of synchronous models that simplify the verification of fault-tolerant distributed systems. We present a sound method to check that an asynchronous protocol is communication closed. The verification conditions...
implied by this method can be automatically discarded using of the self SMT-solvers or static analysers. Provided that an asynchronous protocol is communication close we define a code-to-code translation into the Heard-Of computational model, which is a communication closed round-based model.

3.10 Verification of a Fault-Tolerant Cache-Coherency Protocol

Jo Ebergen (Oracle Labs – Redwood Shores, US)

This short presentation tells some of the lessons we learned while verifying a fault-tolerant cache-coherency protocol.

References

3.11 Distributed Monitoring of Controlled Events

Yuval Emek (Technion – Haifa, IL)

Joint work of Yuval Emek, Amos Korman, Shimon Bitton, Shay Kutten

Monitoring is a fundamental task in many distributed systems. In its most basic form, monitoring is concerned with counting the number of events and detecting when this number reaches some threshold. A good monitoring protocol should run in the background without consuming too many network resources. The challenge in this regard is that the events to be counted may occur in different locations and at unpredicted times.

In this talk, we focus on the task of monitoring controlled events, namely, events that actually take place (or commit) only after they receive a permit from the monitoring protocol. We will discuss scenarios involving this kind of events and explore the connections between the task of monitoring them and the classic distributed controller problem including some recent advances in the study of this problem.

The talk will be self contained.

References
Modern software developments kits simplify the programming of concurrent applications by providing shared state abstractions which encapsulate low-level accesses into higher-level abstract data types (ADTs). Programming such abstractions is however error prone. To minimize synchronization overhead between concurrent ADT invocations, implementors avoid blocking operations like lock acquisition, allowing methods to execute concurrently. However, concurrency risks unintended inter-operation interference, and risks conformance to well-established correctness criteria like linearizability. We present several results concerning the theoretical limits of verifying such concurrent ADTs and testing-based methods for discovering violations in practical implementations.

Distributed synthesis automates the construction of distributed systems. Instead of programming an implementation, the developer writes a formal specification of the desired system properties, for example in a temporal logic. The check whether the specified properties are realizable and the construction of the actual implementation is taken care of by the synthesis algorithm. In this talk, I give an overview on decidability results and algorithms for the two prominent models for distributed synthesis, the Pnueli/Rosner model and the Causal Memory model. The talk concludes with an outlook on the synthesis problem for HyperLTL, a temporal logic for hyperproperties. HyperLTL makes it possible to synthesize distributed systems that additionally satisfy conditions such as symmetric responses, secrecy, and fault tolerance.

It is well known that the communication delay model assumed for a distributed system has large impact on the solvability of problems within it. The same is true for signal propagation...
delay models in circuits. In the talk we discuss solvability issues for several circuit delay models, and draw the relation to distributed computing models and verification of such systems.

References
2. Matthias Függer, Robert Najvirt, Thomas Nowak, and Ulrich Schmid. Towards binary circuit models that faithfully capture physical solvability. DATE’15

3.15 Indistinguishability: Friend and Foe in Concurrent Programming

Hagit Attiya (Technion – Haifa, IL)

License © Creative Commons BY 3.0 Unported license
© Hagit Attiya
Joint work of Hagit Attiya, Ramalingam, Noam Rinetzky, Rachid Guerraoui, Danny Hendler, Peter Kuznetsov, Maged Michael, Martin Vechev, Sandeep Hans, Alexey Gotsman

Uncertainty about the global state is a major obstacle for achieving synchronization in concurrent systems. Formally, uncertainty is captured by showing that a process cannot distinguish two different global states. Indistinguishability arguments play a key role in many lower bounds for concurrent data structures, one of them, on the need for memory barriers, will be presented in this talk. Surprisingly, however, indistinguishability can also help in the verification of concurrent data structures, as demonstrated by a reduction theorem we will describe, or in understanding their specification, as we will show in the context of transactional memory.

(Overview talk.)

3.16 Cutoff Results for Parameterized Verification and Synthesis

Swen Jacobs (Universität des Saarlandes, DE)

License © Creative Commons BY 3.0 Unported license
© Swen Jacobs
Joint work of Simon Außerlechner, Swen Jacobs, Ayrat Khalimov, Mouhammad Sakr
URL http://dx.doi.org/10.1007/978-3-319-73721-8_12

In this talk, I highlight some of the principles and challenges of the cutoff-based approach to the verification and synthesis of systems of parametric size. I give an overview of some of our recent results that tackle these challenges, specifically in the framework of guarded protocols. Finally, I talk about our ongoing work on extensions of these techniques.
References


3.17 What my computer can find about your distributed algorithm

Igor Konnov (INRIA Nancy – Grand Est, FR)

License © Creative Commons BY 3.0 Unported license

Joint work of Igor Konnov, Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Marijana Lazic, Sasha Rubin, Helmut Veith, Josef Widder


URL http://dx.doi.org/10.2200/S00658ED1V01Y201508DCT013

Parameterized model checking is an active research field that addresses automated verification of distributed or concurrent systems, for all numbers of participating processes. The system models that are studied in this field are inspired by those from distributed computing. In this talk, I summarize the prominent techniques for parameterized model checking. Starting with the first undecidability results. Continuing with techniques such as cut-off proofs and abstraction. Finishing with our recent results on verification of threshold-guarded distributed algorithms.

Based on joint work with Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Marijana Lazic, Sasha Rubin, Helmut Veith, and Josef Widder.

References

3.18 Synthesizing Thresholds for Fault-Tolerant Distributed Algorithms

Marijana Lazic (TU Wien, AT)

License © Creative Commons BY 3.0 Unported license
© Marijana Lazic
Joint work of Marijana Lazic, Igor Konnov, Josef Widder, Roderick Bloem
URL http://dx.doi.org/10.4230/LIPIcs.OPODIS.2017.32

We focus on threshold-based distributed algorithms, where a process has to wait until the number of messages it receives reaches a certain threshold, in order to perform an action. Examples of such distributed algorithms include fault-tolerant broadcast, non-blocking atomic commitment, and consensus. I present an automated method for synthesizing these thresholds, given a sketch of a distributed algorithm and specifications. In this way we synthesize distributed algorithms that are correct for every number $n$ of processes and every number $t$ of faults, provided some resilience condition holds, e.g. $n > 3t$.

3.19 Breaking and (Partly) Fixing Pastry

Stephan Merz (INRIA Nancy – Grand Est, FR)

License © Creative Commons BY 3.0 Unported license
© Stephan Merz
Joint work of Stephan Merz, Noran Azmy, Tianxiang Lu, Christoph Weidenbach
URL http://dx.doi.org/10.1016/j.scico.2017.08.003

Pastry [1] is a well-known algorithm for maintaining a distributed hash table over a peer-to-peer overlay network. A key correctness requirement is that the algorithm must ensure a sufficiently consistent view among the participating nodes of which nodes are live members of the network, in the absence of centralized control. In particular, this is necessary for requests to be routed to the intended destination. This property represents an interesting target for formal verification.

We analyzed formal models of Pastry using the TLA$^+$ model checker and identified problems in the different published versions of the algorithm that can lead to unrepairable loss of connectivity among the nodes in the Pastry ring, even in the absence of spontaneous node departures. Identifying the root cause of the problem, we suggest a variant of the algorithm and formally prove, using the TLA$^+$ proof system, that it ensures that requests are routed correctly, assuming that nodes do not fail [2]. We do not know to what extent our Pastry variant is robust to spontaneous node departures.

References
3.20 Indistinguishability, Duality, and Coordination

Yoram Moses (Technion – Haifa, IL)

Indistinguishability is a fundamental notion in distributed systems. It serves as the central tool in impossibility proofs and lower bounds. Indeed, indistinguishability can be used to determine when actions are disallowed. Its dual, which corresponds to the knowledge that a process has, plays the opposite role, and determines when actions are allowed. This talk will discuss the relation between knowledge and action in distributed systems, and present several theorems that apply across all models of distributed computation. The connections drawn also relate a semantic approach, which can be viewed in terms a modal logic, and algorithmic issues.

References

3.21 Interactive Distributed Proofs

Rotem Oshman (Tel Aviv University, IL)

Interactive proof systems allow a resource-bounded verifier to decide an intractable language (or compute a hard function) by communicating with a powerful but untrusted prover. Such systems guarantee that the prover can only convince the verifier of true statements. In the context of centralized computation, a celebrated result shows that interactive proofs are extremely powerful, allowing polynomial-time verifiers to decide any language in PSPACE.

In this work we initiate the study of distributed interactive proofs: a network of nodes interacts with a single untrusted prover, who sees the entire network graph, to decide whether the graph satisfies some property. We focus on the communication cost of the protocol — the number of bits the nodes must exchange with the prover and each other. Our model can also be viewed as a generalization of the various models of “distributed NP” (proof labeling schemes, etc.) which received significant attention recently: while these models only allow the prover to present each network node with a string of advice, our model allows for back-and-forth interaction. We prove both upper and lower bounds for the new model. We show that for some problems, interaction can exponentially decrease the communication cost compared to a non-interactive prover, but on the other hand, some problems retain non-trivial cost even with interaction.
3.22  Proof-Labeling Schemes: Broadcast, Unicast and In Between

Mor Perry (Tel Aviv University, IL)

We study the effect of limiting the number of different messages a node can transmit simultaneously on the verification complexity of proof-labeling schemes (PLS). In a PLS, each node is given a label, and the goal is to verify, by exchanging messages over each link in each direction, that a certain global predicate is satisfied by the system configuration. We consider a single parameter \( r \) that bounds the number of distinct messages that can be sent concurrently by any node: in the case \( r = 1 \), each node may only send the same message to all its neighbors (the broadcast model), in the case \( r \) is at least \( \Delta \), where \( \Delta \) is the largest node degree in the system, each neighbor may be sent a distinct message (the unicast model), and in general, for \( r \) between 1 and \( \Delta \), each of the \( r \) messages is destined to a subset of the neighbors.

We show that message compression linear in \( r \) is possible for verifying fundamental problems such as the agreement between edge endpoints on the edge state. Some problems, including verification of maximal matching, exhibit a large gap in complexity between \( r = 1 \) and \( r > 1 \). For some other important predicates, the verification complexity is insensitive to \( r \), e.g., the question whether a subset of edges constitutes a spanning-tree. We also consider the congested clique model. We show that the crossing technique for proving lower bounds on the verification complexity can be applied in the case of congested clique only if \( r = 1 \).

Together with a new upper bound, this allows us to determine the verification complexity of MST in the broadcast clique.

3.23  Pretend Synchrony- some distributed computing approaches

Sergio Rajasbaum (National Autonomous University of Mexico, MX)

Pretend Synchrony is the title of a recent talk at VDS in Essaouira, Morocco 2018 by Ranjit Jhala where a restricted computational model is shown to be sufficient to verify correctness assertions for several distributed problems. In addition to Ranjit, others discussed related approaches at VDS, including Josef Widder, Cezara Dragoi, Bernhard Kragl and Ahmed Bouajjani, sometimes emphasizing the importance of the classic Communication-Closed Layers paradigm of Elrad and Frances. Motivated by these works, I will describe some of the research (not as recent, some dating back to 1998) which we have done on pretending synchrony from the distributed computing perspective, in the hope that this topics serves as a good point for exchanging ideas between the verification and distributed computing communities. I will discuss work on layering analysis for consensus, generalizations to other problems using topology [1], and iterated models together with recursive distributed algorithms [3, 4].
3.24 Biased Clocks: A way to Improve Effectiveness of Run Time Monitoring of Distributed Systems

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

Runtime Monitoring of distributed systems requires $O(n)$ sized timestamps given that events in a system cannot be partitioned into a total order. $O(n)$ sized timestamps severely limit the ability to utilize them in practice. $O(1)$ sized timestamps such as logical clocks or hybrid logical clocks can be used for runtime monitoring. However, they miss several instances where the property of interest is violated but the violation is not detected. We propose a new type of clocks, biased clocks, that improve the effectiveness of clocks in monitoring. Biased clocks treat local events on a process differently than messages. In particular, by adding a bias to the timestamp received in a message, we show that it substantially improves the ability to detect violations of desired system properties.

3.25 Playing with scheduling policies

Arnaud Sangnier (University Paris-Diderot, FR)

In order to develop distributed algorithms, assumptions are made on their execution context: will the entities behave synchronously or in an asynchronous way, will the entities execution be scheduled in a round-robin way or will its order be completely non-deterministic, will the entities crash, will they be dependent one from each other or should they be able to run the algorithm independently, etc.

As a matter of fact, some tasks may be achieved in some executions contexts and changing an hypothesis on these contexts may lead to impossibility results. For instance, consensus cannot be achieved with 2 processes running a wait-free algorithm on a shared memory system, but this task is feasible when considering obstruction-free algorithms. One difficulty is however to find a formal way to define executive contexts. In this talk, I will present a
recent approach which consists in using automata to represent some executive contexts for
shared memory systems and two-player games to detect the possibility or impossibility of
achieving consensus in such contexts.

3.26 Linearizability via Order-extension Results

Ana Sokolova (Universität Salzburg, AT)

License ☐ Creative Commons BY 3.0 Unported license
© Ana Sokolova
Joint work of Ana Sokolova, Harald Woracek

The semantics of concurrent data structures is usually given by a sequential specification
and a consistency condition. Linearizability is the most popular consistency condition due
to its simplicity and general applicability. Verifying linearizability is a difficult, in general
undecidable, problem.

In this talk, I will discuss the semantics of concurrent data structures and (1) give an
overview of work done on this topic by myself and a group of coauthors, as well as (2) present
recent order extension results (joint work with Harald Woracek) that lead to characterizations
of linearizability in terms of violations, a.k.a. aspects. The approach works for pools, queues,
and priority queues; finding other applications is ongoing work. In the case of pools and
queues we obtain already known characterizations, but the proof method is new, elegant,
and simple, and we expect that it will lead to deeper understanding of linearizability.

3.27 Model checking of incomplete systems

Paola Spoletini (Kennesaw State University – Marietta, US)

License ☐ Creative Commons BY 3.0 Unported license
© Paola Spoletini
Joint work of Paola Spoletini, Claudio Menghi, Carlo Ghezzi, Marsha Chechik, Anna Bernasconi, Lenore Zuck
Main reference Claudio Menghi, Paola Spoletini, Carlo Ghezzi: “Dealing with Incompleteness in Automata-Based
Model Checking”, in Proc. of the FM 2016: Formal Methods – 21st International Symposium,
Limassol, Cyprus, November 9-11, 2016, Proceedings, Lecture Notes in Computer Science,
URL http://dx.doi.org/10.1007/978-3-319-48989-6_32

Incomplete models [3] describe the behavior of systems where some components or func-
tionalities are still unspecified. These models can be used in different scenarios; examples
are (1) analysis the trade-offs among alternative solutions for the unspecified parts, (2)
development of component-based and distributed systems. Classic model checking assumes
that a complete model of the system is available and does not support the verification of
incomplete models. This is an obstacle to early detection of design errors since in early
phases of the system design models are often incomplete.

In this talk, I present a novel automata-based model checking approach that supports
verification of incomplete models. I explore two complementary solutions for handling cases
in which the satisfaction of a given property depends on the yet unspecified parts of the
model.

The first solution enables the computation of constraints that must be satisfied by future
replacements of the unspecified components to guarantee the satisfaction of the given property.
The satisfaction of these constraints by the replacements of the unspecified components,
that can be checked in isolation, ensures the fulfilment of the property of interest [3]. This
approach can be complemented with a framework [1] that helps developers understanding
why a property of interest is satisfied or “possibly” satisfied (i.e., its satisfaction depends
on unknown parts) by enriching the model checker outcome with a proof of satisfaction or
“possibly” satisfaction in these cases.

While the presented approach was developed to deal with incomplete systems, it could
be also used to distribute the complexity of the verification of very large systems. This may
be obtained though an iterative decomposition of the system into smaller parts that are
capsulated into unspecified components.

The second solution is based on a framework that supports (1) incompleteness through a
formal specification of pre- and post-conditions and (2) independent development, reuse of
off-the-shelf components, synthesis and verification of sub-components [2].

References
1 Anna Bernasconi, Claudio Menghi, Paola Spoletini, Lenore D. Zuck, Carlo Ghezzi. From
Model Checking to a Temporal Proof for Partial Models. SEFM 2017:54–69
2 Claudio Menghi, Paola Spoletini, Marsha Chechik, Carlo Ghezzi. Supporting Verification-
Driven Incremental Distributed Design of Components. FASE 2018:169–188
3 Claudio Menghi, Paola Spoletini, Carlo Ghezzi. Dealing with Incompleteness in Automata-
Based Model Checking. FM 2016:531–550

3.28 Distributed Encoding of the Integers

Corentin Travers (University of Bordeaux, FR)

A distributed encoding of the integer is a distributed structure that encodes each positive
integer \( n \) with a word \( w \) of length \( n \) over some (non-necessarily finite) alphabet \( A \), such
that any for any \( n' < n \), any subword \( w' \) of \( w \) of length \( n' \) is not the distributed code of \( n' \).
Relying on well-quasi order theory, we show that the first \( N \) integers can be distributedly
encoded using words on an alphabet with letters on \( O(\log(\alpha(n))) \) bits, where \( \alpha \) is a function
growing at least as slowly as the inverse-Ackerman function.

We then show that distributed encoding of the integers can be applied in failure prone
distributed systems to build failure detector outputting very few bits and to construct short
certificate for distributed decision.

References
1 Pierre Fraigniaud, Sergio Rajsbaum, Corentin Travers, Petr Kuznetsov, Thibault Rieutord.
3.29 Towards verification of distributed algorithms in the Heard-of model

Igor Walukiewicz (University of Bordeaux, FR)

We consider algorithms in the Heard-of model of distributed computation proposed by Charron-Bost and Schiper in 2009. We aim at verifying automatically if a given algorithm solves the consensus problem. In order to state the problem formally we need to fix what operations can algorithms perform. We propose to consider operations that are definable by existentially quantified linear inequalities. We call such algorithms tame. We show that even for tame algorithms the problem is undecidable. Then we present two decidable special cases. One when algorithms use only two values. The other is based on a short run property. We show that every run is equivalent to a short run if the algorithm has what we call stability property.

3.30 (Strong) Linearizability – A Tutorial

Philipp Woelfel (University of Calgary, CA)

This is a tutorial on linearizability, the gold standard of correctness conditions for shared memory algorithms. The first part of the talk will cover necessary definitions, examples, properties, and why linearizability is so important. The second part of the talk will show why linearizability is not enough for randomized algorithms, and will introduce a stronger correctness condition, strong linearizability, that resolves the issues with linearizability in certain randomized models.
Participants

- Paul C. Attie
  American University of Beirut, LB
- Hagit Attiya
  Technion – Haifa, IL
- A. R. Balasubramanian
  Chennai Mathematical Institute, IN
- Michael Blondin
  TU München, DE
- Benedikt Bollig
  ENS – Cachan, FR
- Borzoo Bonakdarpour
  McMaster University – Hamilton, CA
- Ahmed Bouajjani
  University Paris-Diderot, FR
- Dan Brownstein
  Ben Gurion University – Beer Sheva, IL
- Keren Censor-Hillel
  Technion – Haifa, IL
- Aiswarya Cyriac
  Chennai Mathematical Institute, IN
- Giorgio Delzanno
  University of Genova, IT
- Cezara Dragoi
  ENS – Paris, FR
- Jo Ebergen
  Oracle Labs – Redwood Shores, US
- Yuval Emek
  Technion – Haifa, IL
- Constantin Enea
  University Paris-Diderot, FR
- Javier Esparza
  TU München, DE
- Bernd Finkbeiner
  Universität des Saarlandes, DE
- Marie Fortin
  ENS – Cachan, FR
- Pierre Fraigniaud
  University Paris-Diderot and CNRS, FR
- Matthias Függer
  ENS – Cachan, FR
- Paul Gastin
  ENS – Cachan, FR
- Swen Jacobs
  Universität des Saarlandes, DE
- Igor Konnov
  INRIA Nancy – Grand Est, FR
- Sandeep Kulkarni
  Michigan State University – East Lansing, US
- Marijana Lazic
  TU Wien, AT
- Jérémy Ledent
  École Polytechnique – Palaiseau, FR
- Martin Leucker
  Universität Lübeck, DE
- Stephan Merz
  INRIA Nancy – Grand Est, FR
- Roland Meyer
  TU Braunschweig, DE
- Yoram Moses
  Technion – Haifa, IL
- Anca Muscholl
  University of Bordeaux, FR
- Rotem Oshman
  Tel Aviv University, IL
- Mor Perry
  Tel Aviv University, IL
- Sergio Rajsbaum
  National Autonomous University of Mexico, MX
- David A. Rosenblueth
  National Autonomous University of Mexico, MX
- Arnaud Sangnier
  University Paris-Diderot, FR
- Ana Sokolova
  Universität Salzburg, AT
- Paola Spoletini
  Kennesaw State University – Marietta, US
- Corentin Travers
  University of Bordeaux, FR
- Igor Walukiewicz
  University of Bordeaux, FR
- Philipp Woelfel
  University of Calgary, CA