

Formal Verification of Distributed Algorithms

Edited by

Bernadette Charron-Bost¹, Stephan Merz², Andrey Rybalchenko³,
and Josef Widder⁴

1 École polytechnique, Palaiseau, FR, charron@lix.polytechnique.fr

2 INRIA, Nancy, FR, stephan.merz@loria.fr

3 TU München, DE, rybal@in.tum.de

4 TU Wien, AT, widder@forsyte.tuwien.ac.at

Abstract

The Dagstuhl Seminar 13141 “Formal Verification of Distributed Algorithms” brought together researchers from the areas of distributed algorithms, model checking, and semi-automated proofs with the goal to establish a common base for approaching the many open problems in verification of distributed algorithms. In order to tighten the gap between the involved communities, who have been quite separated in the past, the program contained tutorials on the basics of the concerned fields. In addition to technical talks, we also had several discussion sessions, whose goal was to identify the most pressing research challenges. This report describes the program and the outcomes of the seminar.

Seminar 1.–5. April, 2013 – www.dagstuhl.de/13141

1998 ACM Subject Classification C.2 Computer-Communication Networks, D.3.1 Formal Definitions and Theory, D.2.4 Software/Program Verification, F.3 Logics and Meanings of Programs

Keywords and phrases Distributed algorithms, semi-automated proofs, model checking

Digital Object Identifier 10.4230/DagRep.3.4.1

Edited in cooperation with Thomas Nowak


1 Executive Summary

Bernadette Charron-Bost

Stephan Merz

Andrey Rybalchenko

Josef Widder

License  Creative Commons BY 3.0 Unported license
© Bernadette Charron-Bost, Stephan Merz, Andrey Rybalchenko, and Josef Widder

While today’s society depends heavily on the correct functioning of distributed computing systems, the current approach to designing and implementing them is still error prone. This is because there is a methodological gap between the theory of distributed computing and the practice of designing and verifying the correctness of reliable distributed systems. We believe that there are two major reasons for this gap: On the one hand, distributed computing models are traditionally represented mainly in natural language, and algorithms are described in pseudo code. The classical approach to distributed algorithms is thus informal, and it is not always clear under which circumstances a given distributed algorithm actually is correct. On the other hand, distributed algorithms are designed to overcome non-determinism due to issues that are not within the control of the distributed algorithm, including the system’s



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Formal Verification of Distributed Algorithms, *Dagstuhl Reports*, Vol. 3, Issue 4, pp. 1–16

Editors: Bernadette Charron-Bost, Stephan Merz, Andrey Rybalchenko, and Josef Widder



DAGSTUHL
REPORTS

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

timing behavior or faults of some components. Such issues lead to a huge executions space which is the major obstacle in applying verification tools to distributed algorithms.

The rationale behind our Dagstuhl seminar was that closing the methodological gap requires collaboration from researchers from distributed algorithms and formal verification. In order to spur the interaction between the communities, the program contained the following overview talks on the related subjects:

Distributed algorithms: Eric Ruppert (York University)

Semi-automated proofs: John Rushby (SRI)

Parameterized model checking: Muralidhar Talupur (Intel)

In addition to the tutorials, we organized several open discussion rounds. The seminar participants identified modeling issues as a central question, which confirmed one of our motivations for the seminar, namely, the lack of a universal model for distributed algorithms. Hence, one of the discussion rounds was exclusively devoted to this topic. Unlike sequential programs, whose semantics is well understood and closely follows the program text, the executions of distributed algorithms are to a large extent determined by the environment, including issues such as the distribution of processes, timing behavior, inter-process communication, and component faults. Models of distributed algorithms and systems embody different assumptions about how the environment behaves. These hypotheses are often left implicit but are of crucial importance for assessing the correctness of distributed algorithms. The discussions during the seminar raised the awareness of these issues among the researchers, and showed that research in this area is a necessary first step towards a structured approach to formal verification of distributed algorithms. In addition to modeling, we discussed issues such as benchmarks, implementation of distributed algorithms, or application areas of distributed algorithms.

To round-off the technical program, we had several short presentations by participants who presented their past and current work in the intersection of formal methods and distributed algorithms, and a joint session with the other seminar going on concurrently at Dagstuhl on Correct and Efficient Accelerator Programming. The topics of the talks spanned large parts of the concerned areas, for instance, there were talks on model checking techniques such as partial order reductions or abstractions, and their applications to distributed algorithms; several talks focus on proof assistants, and how they can be used to verify distributed algorithms; some talks considered concurrent systems, and some focused on transactional memory. The atmosphere during these sessions was very constructive, and the short talks were always followed by elaborate and insightful discussions.

2 Table of Contents

Executive Summary

Bernadette Charron-Bost, Stephan Merz, Andrey Rybalchenko, and Josef Widder . . . 1

Overview of Talks

Partial-Order Reductions: Landscape & Practice <i>Péter Bokor</i>	5
Automated Repair of Distributed Systems: Beyond Verification <i>Borzoo Bonakdarpour</i>	5
Formal Proofs in Coq of Local Computation Systems <i>Pierre Castéran</i>	5
Semantics of Eventually Consistent Systems <i>Alexey Gotsman</i>	6
A Fault-tolerant Communication Mechanism for Cooperative Robots <i>Serge Haddad</i>	6
Scaling Up Interactive Verification <i>Gerwin Klein</i>	7
Parameterized Model Checking of Fault-Tolerant Broadcasting Algorithms <i>Igor Konnov</i>	7
Verification of a Quasi Certification Protocol over a DHT <i>Fabrice Kordon</i>	7
Finding Non-terminating Executions in Distributed Asynchronous Programs <i>Akash Lal</i>	8
A Framework for Formally Verifying Software Transactional Memory (and Other Concurrent Algorithms) <i>Victor Luchangco</i>	8
Verification of Fault-Tolerant Distributed Algorithms in the Heard-Of Model <i>Stephan Merz</i>	9
Verifying Consensus . . . Using Process Calculi, State Machines, and Proof Checkers <i>Uwe Nestmann</i>	10
Tutorial on Distributed Algorithms <i>Eric Ruppert</i>	10
Getting the Best out of General-Purpose Tools: Theorem Provers and Infinite-Bounded Model Checker <i>John Rushby</i>	11
An Epistemic Perspective on Consistency of Concurrent Computations <i>Andrey Rybalchenko</i>	11
Unidirectional Channel Systems Can Be Tested <i>Philippe Schnoebelen</i>	11
Formal Verification of Distributed Algorithms at TTTech <i>Wilfried Steiner</i>	12

4 13141 – Formal Verification of Distributed Algorithms

Tutorial on Parameterized Model Checking <i>Murali Talupur</i>	12
Correctness without Serializability: Verifying Transactional Programs under Snapshot Isolation <i>Serdar Tasiran</i>	13
(Dis)Proof Automation: What We Can Do, What We Could Do and What Is Needed? <i>Christoph Weidenbach</i>	13
Efficient Checking of Link-Reversal-Based Concurrent Systems <i>Josef Widder</i>	14
Panel Discussions	
Session 1: What are the problems?	14
Session 2: Modeling	15
Session 3: Follow-up	15
Participants	16

3 Overview of Talks

3.1 Partial-Order Reductions: Landscape & Practice

Péter Bokor (ALTEN Engineering, Berlin, DE)

License © Creative Commons BY 3.0 Unported license
© Péter Bokor

Joint work of Bokor, Péter; Kinder, Johannes; Serafini, Marco; Suri, Neeraj

Main reference P. Bokor, J. Kinder, M. Serafini, N. Suri, “Supporting domain-specific state space reductions through local partial-order reduction,” in Proc. of the 26th IEEE/ACM Int’l Conf. on Automated Software Engineering (ASE’11), pp. 113–122. IEEE Press, New York City, 2011.

URL <http://dx.doi.org/10.1109/ASE.2011.6100044>

This talk is about mainstream partial-order reductions (such as DPOR), their extensions (LPOR), new approaches (our current ongoing work Ostrich), and applications (to fault-tolerant distributed protocols).

3.2 Automated Repair of Distributed Systems: Beyond Verification

Borzoo Bonakdarpour (University of Waterloo, CA)

License © Creative Commons BY 3.0 Unported license
© Borzoo Bonakdarpour

Although verification of concurrent and distributed applications has recently made considerable attention in the recent years, correct construction of such applications still remains a challenge. This is simply due to the inherently complex structure of concurrent applications caused by non-determinism and occurrence of faults (in a distributed setting). To deal with the subtleties of developing concurrent applications, my position is to focus on formal methods that automatically build such applications that are correct-by-construction. In this talk, I briefly describe the efforts made for achieving correctness by construction for concurrent/distributed applications in the area of automated repair of concurrent models.

3.3 Formal Proofs in Coq of Local Computation Systems

Pierre Castéran (University of Bordeaux, FR)

License © Creative Commons BY 3.0 Unported license
© Pierre Castéran

Joint work of Castéran, Pierre; Filou, Vincent; Fontaine, Allyx

Main reference P. Castéran, V. Filou, “Tasks, types and tactics for local computation systems,” *Studia Informatica Universalis* 9(1):39–86, 2011.

URL http://studia.complexica.net/index.php?option=com_content&view=article&id=186%3Atasks-types-and-tactics-for-local-computation-systems-pp-39-86-

We present a library written for the Coq proof assistant, for reasoning about a quite abstract model of distributed computing based on graph rewriting: Local Computations Systems. A first development allowed us to prove some facts about the expressive power of several subclasses of such systems, e.g., impossibility results and certified transformations. Directions for future evolutions will be also discussed, in particular reasoning on dynamic graphs and self-stabilizing systems.

3.4 Semantics of Eventually Consistent Systems

Alexey Gotsman (IMDEA Software, Madrid, ES)

License  Creative Commons BY 3.0 Unported license
© Alexey Gotsman

Joint work of Burckhardt, Sebastian; Gotsman, Alexey; Yang, Hongseok

Main reference S. Burckhardt, A. Gotsman, H. Yang, “Understanding Eventual Consistency,” Technical Report MSR-TR-2013-39, Microsoft Research, 2013.

URL <http://software.imdea.org/~gotsman/papers/distrmm.pdf>

Modern geo-replicated databases underlying large-scale Internet services guarantee immediate availability and tolerate network partitions at the expense of providing only weak forms of consistency, commonly dubbed eventual consistency. At the moment there is a lot of confusion about the semantics of eventual consistency, as different systems implement it with different sets of features and in subtly different forms, stated either informally or using disparate and low-level formalisms.

We address this problem by proposing a framework for formal and declarative specification of the semantics of eventually consistent systems using axioms. Our framework is fully customisable: by varying the set of axioms, we can rigorously define the semantics of systems that combine any subset of typical guarantees or features, including conflict resolution policies, session guarantees, causality guarantees, multiple consistency levels and transactions. We prove that our specifications are validated by an example abstract implementation, based on algorithms used in real-world systems. These results demonstrate that our framework provides system architects with a tool for exploring the design space, and lays the foundation for formal reasoning about eventually consistent systems.

This is joint work with Sebastian Burckhardt (MSR Redmond) and Hongseok Yang (Oxford).

3.5 A Fault-tolerant Communication Mechanism for Cooperative Robots

Serge Haddad (ENS Cachan, FR)

License  Creative Commons BY 3.0 Unported license
© Serge Haddad

Joint work of El Haddad, Joyce; Haddad, Serge

Main reference J. El Haddad, S. Haddad, “A fault-tolerant communication mechanism for cooperative robots,” International Journal of Production Research, 42(14):2793–2808, 2004.

URL <http://dx.doi.org/10.1080/00207540410001705185>

Operations in unpredictable environments require coordinating teams of robots. This coordination implies peer-to-peer communication between the team’s robots, resource allocation, and coordination. We address the problem of autonomous robots which alternate between execution of individual tasks and peer-to-peer communication. Each robot keeps in its permanent memory a set of locations where it can meet some of the other robots. The proposed protocol is constructed by two layered modules (sub-algorithms: a self-stabilizing scheduling and a construction of a minimum-hop path forest). The first self-stabilizing algorithm solves the management of visits to these locations ensuring that, after the stabilizing phase, every visit to a location will lead to a communication. We model the untimed behaviour of a robot by a Petri net and the timed behaviour by an (infinite) Discrete Time Markov Chain. Theoretical results in this area are then combined in order to establish the proof of the algorithm. The second self-stabilizing algorithm computes the minimum-hop path between a specific robot’s location and the locations of all the other robots of the system in order to implement routing.

3.6 Scaling Up Interactive Verification

Gerwin Klein (NICTA & UNSW, Sydney, AU)

License © Creative Commons BY 3.0 Unported license
© Gerwin Klein

This talk gives a brief overview of the formal verification of the seL4 microkernel. I will cover the proof of functional correctness, later high-level security properties, the extension of the proof to the binary level, and the effect of maintenance on the verification. After this, the idea is to open the floor to a more free-form discussion on the experience of large-scale software verification and the applicability of our experience to the distributed algorithms section.

3.7 Parameterized Model Checking of Fault-Tolerant Broadcasting Algorithms

Igor Konnov (TU Wien, AT)

License © Creative Commons BY 3.0 Unported license
© Igor Konnov
Joint work of John, Annu; Konnov, Igor; Schmid, Ulrich; Veith, Helmut; Widder, Josef
Main reference A. John, I. Konnov, U. Schmid, H. Veith, J. Widder, “Counter Attack on Byzantine Generals: Parameterized Model Checking of Fault-tolerant Distributed Algorithms,” arXiv:1210.3846v2 [cs.LO], 2013.
URL <http://arxiv.org/abs/1210.3846v2>

We introduce an automated parameterized verification method for fault-tolerant distributed algorithms (FTDA). FTDAs are parameterized by both the number of processes and the assumed maximum number of Byzantine faulty processes. At the center of our technique is a parametric interval abstraction (PIA) where the interval boundaries are arithmetic expressions over parameters. Using PIA for both data abstraction and a new form of counter abstraction, we reduce the parameterized problem to finite-state model checking. We demonstrate the practical feasibility of our method by verifying several variants of the well-known distributed algorithm by Srikanth and Toueg. Our semi-decision procedures are complemented and motivated by an undecidability proof for FTDA verification which holds even in the absence of interprocess communication. To the best of our knowledge, this is the first paper to achieve parameterized automated verification of Byzantine FTDA.

3.8 Verification of a Quasi Certification Protocol over a DHT

Fabrice Kordon (UPMC, Lab. LIP6, Paris, FR)

License © Creative Commons BY 3.0 Unported license
© Fabrice Kordon
Joint work of Kordon, Fabrice; Bonnaire, Xavier; Cortés, Rudyar; Marin, Olivier

Building a certification authority that is both decentralized and fully reliable is impossible. However, the limitation thus imposed on scalability is unacceptable for many types of information systems, such as e-government services. We propose a solution to build an highly reliable certification authority, based on a distributed hash table and a dedicated protocol ensuring a very low probability of arbitrary failure. Thus, in practice, false positives should never occur. This talk briefly presents the protocol and shows its verification in two steps: (1) a formal model to assess that the protocol behaves as expected in an “ideal world” where

communications are reliable, and, (2) a probabilistic analysis to evaluate the probability of failure of the certification.

3.9 Finding Non-terminating Executions in Distributed Asynchronous Programs

Akash Lal (Microsoft Research India, Bangalore, IN)

License © Creative Commons BY 3.0 Unported license
© Akash Lal

Joint work of Emmi, Michael; Lal, Akash

Main reference M. Emmi, A. Lal, “Finding non-terminating executions in distributed asynchronous programs,” in Proc. of the 19th Int’l Symp. on Static Analysis (SAS’12), LNCS, Vol. 7460, pp. 439–455, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-33125-1_29

Programming distributed and reactive asynchronous systems is complex due to the lack of synchronization between concurrently executing tasks, and arbitrary delay of message-based communication. As even simple programming mistakes have the capability to introduce divergent behavior, a key liveness property is *eventual quiescence*: for any finite number of external stimuli (e.g., client-generated events), only a finite number of internal messages are ever created.

In this work we propose a practical three-step reduction-based approach for detecting divergent executions in asynchronous programs. As a first step, we give a code-to-code translation reducing divergence of an asynchronous program P to completed state-reachability, i.e., reachability to a given state with no pending synchronous tasks, of a polynomially-sized asynchronous program P' . In the second step, we give a code-to-code translation under-approximating completed state-reachability of P' by state-reachability of a polynomially-sized recursive sequential program $P''(K)$, for the given analysis parameter K . Following Emmi et al.’s delay-bounding approach, $P''(K)$ encodes a subset of P' , and thus of P , by limiting scheduling nondeterminism. As K is increased, more possibly divergent behaviors of P are considered, and in the limit as K approaches infinity, our reduction is complete for programs with finite data domains. As the final step we give the resulting state-reachability query to an of-the-shelf SMT-based sequential program verification tool.

We demonstrate the feasibility of our approach by implementing a prototype analysis tool called ALIVE, which detects divergent executions in several hand-coded variations of textbook distributed algorithms.

3.10 A Framework for Formally Verifying Software Transactional Memory (and Other Concurrent Algorithms)

Victor Luchangco (Oracle Corporation, Burlington, US)

License © Creative Commons BY 3.0 Unported license
© Victor Luchangco

Joint work of Lesani, Mohsen; Luchangco, Victor; Moir, Mark

Main reference M. Lesani, V. Luchangco, M. Moir, “A framework for formally verifying software transactional memory algorithms,” in Proc. of the 23rd Int’l Conf. on Concurrency Theory (CONCUR’12), LNCS, Vol. 7454, pp. 516–530. Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-32940-1_36

We present a framework for verifying transactional memory (TM) algorithms. Specifications and algorithms are specified using I/O automata, enabling hierarchical proofs that the

algorithms implement the specifications. We have used this framework to develop what we believe is the first fully formal machine-checked verification of a practical TM algorithm: the NOrec algorithm of Dalessandro, Spear and Scott. Our framework is available for others to use and extend. New proofs can leverage existing ones, eliminating significant work and complexity.

3.11 Verification of Fault-Tolerant Distributed Algorithms in the Heard-Of Model

Stephan Merz (LORIA, Nancy, FR)

License © Creative Commons BY 3.0 Unported license
© Stephan Merz

Joint work of Débrat, Henri; Merz, Stephan

Main reference H. Debrat, S. Merz, “Verifying fault-tolerant distributed algorithms in the Heard-Of model,”
Archive of Formal Proofs, 2012.

URL http://afp.sf.net/entries/Heard_Of.shtml


Distributed algorithms are quite subtle, both in the way they function and in the hypotheses assumed for their correctness. Moreover, many different computational models exist in the literature, but comparisons between algorithms expressed in different models is difficult. Formal verification can help ascertain the correctness of a given algorithm w.r.t. well-specified hypotheses. We present work on the formal verification of fault-tolerant distributed algorithms in the Heard-Of model introduced by Charron-Bost and Schiper [1, 2]. In this model, algorithms execute in communication-closed rounds and are subject to hypotheses expressed in terms of Heard-Of sets, i.e., the sets of processes from which messages are received in a given round. We formally prove a reduction theorem that justifies verifying these algorithms in a coarse-grained (synchronous) model of execution. In this way, entire system rounds become the unit of atomicity, and verification becomes much simpler than when interleavings of individual process actions are considered. We have verified six different Consensus algorithms that differ with respect to the presence of a coordinator, the types and numbers of faults they tolerate (both benign and Byzantine failures are considered), and the degree of synchrony that is required for correctness. Both the reduction proof and the verification of the various algorithms are carried out in the proof assistant Isabelle/HOL [3], and they are available online [4].

References

- 1 Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing* 22(1):49-71, 2009.
- 2 Martin Biely, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, André Schiper, and Josef Widder. Tolerating corrupted communication. *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing (PODC’07)*, pp. 244–253. ACM, New York City, 2007.
- 3 Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*. LNCS 2283, Springer, 2002.
- 4 Henri Debrat and Stephan Merz. Verifying fault-tolerant distributed algorithms in the Heard-Of model. *Archive of Formal Proofs*, http://afp.sf.net/entries/Heard_Of.shtml, 2012.

3.12 Verifying Consensus . . . Using Process Calculi, State Machines, and Proof Checkers

Uwe Nestmann (TU Berlin, DE)

License  Creative Commons BY 3.0 Unported license
© Uwe Nestmann

We focus on the gap between pseudo code, as often used for the description of distributed algorithms, and correctness proofs, as usually written in math-enhanced natural language. Trying to bridge the gap, we discuss the use of process calculi and state machines, as well as their connection. We also briefly report on the mechanisation of state-machine-based correctness proofs within the proof assistant Isabelle.

References

- 1 R. Fuzzati, M. Merro and U. Nestmann. Distributed Consensus, Revisited. *Acta Inf.*, 44(6):377–425, 2007.
- 2 M. Kühnrich and U. Nestmann. On Process-Algebraic Proof Methods for Fault Tolerant Distributed Systems. In D. Lee, A. Lopes and A. Poetzsch-Heffter, eds, *FMOODS/FORTE*, volume 5522 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2009.
- 3 P. Küfner, U. Nestmann and C. Rickmann. Formal Verification of Distributed Algorithms – From Pseudo Code to Checked Proofs. In J. C. M. Baeten, T. Ball and F. S. de Boer, eds, *IFIP TCS*, volume 7604 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2012.
- 4 U. Nestmann and R. Fuzzati. Unreliable Failure Detectors via Operational Semantics In V. A. Saraswat, ed, *ASIAN*, volume 2896 of *Lecture Notes in Computer Science*, pages 54–71. Springer, 2003.
- 5 U. Nestmann, R. Fuzzati and M. Merro. Modeling Consensus in a Process Calculus. In R. M. Amadio and D. Lugiez, eds, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2003.

3.13 Tutorial on Distributed Algorithms

Eric Ruppert (York University, Toronto, CA)

License  Creative Commons BY 3.0 Unported license
© Eric Ruppert

I gave some background information on the way distributed algorithm designers model distributed systems and define correctness properties. I briefly described some of the challenges faced in designing distributed algorithms and some techniques used to overcome them, with examples of algorithms that use the techniques. These techniques include quorums, repeated reads to obtain consistent views, timestamps, helping, using CAS to synchronize, pointer swinging and locks.

3.14 Getting the Best out of General-Purpose Tools: Theorem Provers and Infinite-Bounded Model Checker

John Rushby (SRI, Menlo Park, US)

License © Creative Commons BY 3.0 Unported license
© John Rushby

In traditional “by hand” formal verification of distributed algorithms it is often beneficial to work with a model of computation specialized to the issue of primary concern (e.g., timed automata). But the best developed, most powerful mechanized verification tools tend to be general-purpose (or specialized to a different model than the one you want). I describe and demonstrate some techniques for getting the best out of general-purpose tools through adjustments to (the representation of) models that better exploit the underlying automation. I cover representation of nondeterminism in a theorem prover (illustrated using Byzantine Agreement in PVS) and timed systems in an infinite bounded model checker (illustrated using Biphase Mark in SAL). I also briefly describe computational reflection, and some prospects and hopes for the future.

3.15 An Epistemic Perspective on Consistency of Concurrent Computations

Andrey Rybalchenko (TU München, DE)

License © Creative Commons BY 3.0 Unported license
© Andrey Rybalchenko

Joint work of von Gleissenthal, Klaus; Rybalchenko, Andrey

Main reference K. v. Gleissenthal, A. Rybalchenko, “An Epistemic Perspective on Consistency of Concurrent Computations,” arXiv:1305.2295v1 [cs.LO], 2013.

URL <http://arxiv.org/abs/1305.2295>

Consistency properties of concurrent computations, e.g., sequential consistency, linearizability, or eventual consistency, are essential for devising correct concurrent algorithms. In this paper, we present a logical formalization of such consistency properties that is based on a standard logic of knowledge. Our formalization provides a declarative perspective on what is imposed by consistency requirements and provides some interesting unifying insight on differently looking properties.

3.16 Unidirectional Channel Systems Can Be Tested

Philippe Schnoebelen (ENS Cachan, FR)

License © Creative Commons BY 3.0 Unported license
© Philippe Schnoebelen

Joint work of Jančár, Petr; Karandikar, Prateek; Schnoebelen, Philippe

Main reference P. Jančár, P. Karandikar, P. Schnoebelen, “Unidirectional channel systems can be tested,” in Proc. of the 7th IFIP TC1/WG2.2 Int’l Conf. on Theoretical Computer Science (TCS’12), LNCS, Vol. 7604, pp. 149–163, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-33475-7_11

“Unidirectional channel systems” (Chambart & Schnoebelen, CONCUR 2008) are systems where one-way communication from a sender to a receiver goes via one reliable and one unreliable (unbounded fifo) channel. Equipping these systems with the possibility of testing

regular properties on the contents of channels makes verification undecidable. Decidability is preserved when only emptiness and nonemptiness tests are considered: the proof relies on a series of reductions eventually allowing us to take advantage of recent results on Post's Embedding Problem.

3.17 Formal Verification of Distributed Algorithms at TTTech


Wilfried Steiner (TTTech Computertechnik, Vienna, AT)

License  Creative Commons BY 3.0 Unported license
© Wilfried Steiner

System design for safety-critical systems and mixed-criticality systems, such as aerospace or space applications, is inherently complex and demands a level of quality assurance often only to be met by the use of formal methods. This is due to the tightly interwoven requirements of fault tolerance, the ability to sustain partial failures of the system, and real-time control. One key element of a safety-critical system is its communication infrastructure, which more and more determines the overall system architecture. With its central role, the correct design of the communication infrastructure, and in particular the distributed algorithms that the infrastructure implements, is a crucial pre-requisite for mission success. In this talk we discuss how formal methods have been used during the design of the TTEthernet communication infrastructure and their general use at TTTech.

3.18 Tutorial on Parameterized Model Checking

Murali Talupur (Intel SCL, Hillsboro, US)

License  Creative Commons BY 3.0 Unported license
© Murali Talupur

With the move towards multi-core processors and SoCs (systems-on-chip) parameterized verification of distributed protocols has taken on a new urgency. Protocols like cache coherence protocols, bus lock protocols form the bedrock on which these processors are built and verifying them is a challenging task. In this talk I will describe a highly scalable and automated method, called the CMP+ Flows method, for formally and parametrically verifying protocols. As the name indicates the method has two components. The first component, the CMP method, is a compositional reasoning technique that uses abstraction to reduce an unbounded parameterized verification problem to a finite problem that can then be model checked. The abstraction operation is completely automatic but the user has to supply lemmas (or candidate invariants) to progressively refine the abstraction. Though the CMP method imposes less manual burden than pure theorem proving, supplying lemmas is still a non-trivial task, especially for large industrial protocols. The second component of our method addresses this gap by showing how to derive invariants automatically from informal design artifacts called Flows. Flows are essentially partial orders on system events, such as sending and receiving of messages, that architects typically use to conceive the protocols. These are readily available in design documents and as we show they yield powerful invariants. The combined CMP+ Flows method is extremely scalable while imposing minimal burden on the user. Using this method we have verified multiple industrial strength cache coherence protocols and other co-ordination protocols. To our knowledge no other method has been used successfully to verify protocols of such sizes.

3.19 Correctness without Serializability: Verifying Transactional Programs under Snapshot Isolation

Serdar Tasiran (Koc University, Istanbul, TR)

License © Creative Commons BY 3.0 Unported license
© Serdar Tasiran

We present a static verification approach for programs running under snapshot isolation (SI) and similar relaxed transactional semantics. In a common pattern in distributed and concurrent programs, transactions each read a large portion of shared data, perform local computation, and then modify a small portion of the shared data. Requiring conflict serializability in this scenario results in serial execution of transactions or worse, and performance suffers. To avoid such performance problems, relaxed conflict detection schemes such as snapshot isolation (SI) are used widely. Under SI, transactions are no longer guaranteed to be serializable, and the simplicity of reasoning sequentially within a transaction is lost. In this paper, we present an approach for statically verifying properties of transactional programs operating under SI. Differently from earlier work, we handle transactional programs even when they are designed not to be serializable.

In our approach, the user first verifies his program in the static verification tool VCC pretending that transactions run sequentially. This task requires the user to provide program annotations such as loop invariants and function pre- and post-conditions. We then apply a source-to-source transformation which augments the program with an encoding of the SI semantics. Verifying the resulting program with transformed user annotations and specifications is equivalent to verifying the original transactional program running under SI—a fact we prove formally. Our encoding preserves the modularity and scalability of VCC’s verification approach. We applied our method successfully to benchmark programs from the transactional memory literature. In each benchmark, we were able to verify the encoded program without manually providing any extra annotations beyond those required for verifying the program sequentially. The correctness argument of the sequential versions generalized to SI, and verification times were similar.

3.20 (Dis)Proof Automation: What We Can Do, What We Could Do and What Is Needed?

Christoph Weidenbach (MPI für Informatik, Saarbrücken, DE)

License © Creative Commons BY 3.0 Unported license
© Christoph Weidenbach

After an introduction to the underlying principles of designing automated reasoning systems, I discuss $\text{FOL}(T)$, the hierarchic combination of a theory T and first-order logic. In particular for the case where T is a language of arithmetic, e.g., linear rational arithmetic. I show that this language is expressive enough to represent timed, hybrid, and probabilistic automata. Superposition-based reasoning delivers a decision procedure for known decidable reasoning challenges and beyond. The language also strictly generalizes the SMT setting as it considers universally quantified variables in addition to constants.

3.21 Efficient Checking of Link-Reversal-Based Concurrent Systems

Josef Widder (TU Wien, AT)

License © Creative Commons BY 3.0 Unported license
© Josef Widder

Joint work of Függer, Matthias; Widder, Josef

Main reference M. Függer, J. Widder, “Efficient checking of link-reversal-based concurrent systems,” in Proc. of the 23rd Int’l Conf. on Concurrency Theory (CONCUR’12), LNCS, Vol. 7454, pp. 486–499, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-32940-1_34

Link reversal is an algorithmic method with various applications. Originally proposed by Gafni and Bertsekas in 1981 for routing in radio networks, it has been later applied also to solve concurrency related problems as mutual exclusion, resource allocation, and leader election. For resource allocation, conflicts can be represented by conflict graphs, and link reversal algorithms work on these graphs to resolve conflicts. In this talk I explain that executions of link reversal algorithms on large graphs are similar (a notion which I make precise) to executions on smaller graphs. This similarity then allows to verify linear time temporal properties of the large systems, by verifying a smaller one.

4 Panel Discussions

The tentative program included three time slots for discussions. It was intended to have several working groups in parallel where specific topics would be discussed, and then report on the outcomes in a joint session. However, just listing the topics for the working groups sparked a lively discussion, in which most of the participants participated actively. Because the format of seminar-wide discussions turned out to be fruitful, we decided to stick with seminar-wide, moderated sessions.

In the first session, participants were asked what they considered the major open questions in the area of formal verification of distributed algorithms, and what kind of information from the other community they would need to make progress in this area. We identified modeling as the most urgent point: While the formal methods community is used to have a precise description of the object they consider (programming languages, input languages of model checking tools, etc.), distributed algorithms are typically given in pseudo code only. This formalization gap appeared to be crucial, and it was decided to devote the second discussion session to this subject. The final discussion session was devoted to the next steps we could take to bring the concerned communities together. We list the major topics that were discussed:

4.1 Session 1: What are the problems?

- There exists a large variety of **modeling frameworks**, corresponding to different classes of systems. The DA community usually presents models and algorithms informally (using text and pseudo-code), whereas FM researchers and tools require formal semantics definitions. Is formalization just a tedious detail or is it a contribution in itself? Is there a classification of relevant computational models and their relationships?
- A collection of **benchmark problems** could give some guidance to the FM community. Different formal methods could tackle these problems at different levels of abstraction,

enabling their comparison over practically relevant case studies. These should include novel paradigms of the DA field and different application areas. A collection of verified algorithms, as well as results of **verification competitions** would help DA researchers evaluate the potential and the usability of FM methods and tools.

- Within the FM community, an important line of research is about the **integration of different techniques** such as model checking, proof assistants, SMT solving, etc. Doing so requires consistent semantic models and raises the issue of trust in the results obtained in this way.
- Beyond the verification of distributed algorithms at a high level of abstraction, issues of **verified implementation** of distributed systems, as well as formal assurances on non-functional properties such as **performance or security** are of great interest. To what extent can formal verification play a role in **certification processes** that are used in safety-critical settings? Is it possible to obtain guarantees on systems that integrate verified and unverified components?

4.2 Session 2: Modeling

- The main purpose of a formal model is to clearly express the **semantics** of an algorithm or system. There is a tradeoff between models expressed in natural language and pseudo-code (concise and readable but potentially ambiguous) and formal semantics description (complete and precise but maybe cluttered with too much detail).
- Different models are geared towards different **purposes**. For example, there is a tradeoff between efficient checking vs. the ease of expressing distributed algorithms.
- For distributed algorithms that are intended to operate continually, specifying **initial states** can be problematic. For example, there is a subtle difference between the Consensus problem where all nodes start from scratch, and the repeated Consensus problem where nodes may start at vastly different times.
- The DA community should provide a catalogue of the most important **existing models** for distributed algorithms. This includes variations of shared memory vs. message passing models, round-based vs. asynchronous models, failure models etc. The **system model** should be distinguished from the **computational model**.

4.3 Session 3: Follow-up

- A **follow-up seminar** should be organized in a few years, presenting progress on research into the issues raised during the seminar. Besides another Dagstuhl seminar, a workshop gathering researchers from the DA and FM communities could be organized as a satellite of a major conference, such as FLOC 2014 in Vienna. It would also be useful to raise the awareness of the topics discussed in this seminar by inviting DA researchers to give talks at FM conferences, and vice versa.
- The topic of **models** was identified as the most important one, and it would be useful to work out a catalogue or classification, as mentioned above.
- Research about the application of FM methods and tools on interesting distributed algorithms would benefit from maintaining a list of verified algorithms, e.g. in the form of a **Wiki** page to which different people could contribute.

Participants

- Béatrice Bérard
UPMC, Lab. LIP6 – Paris, FR
- Péter Bokor
ALTEN Engineering – Berlin, DE
- Borzoo Bonakdarpour
University of Waterloo, CA
- Pierre Castéran
University of Bordeaux, FR
- Bernadette Charron-Bost
Ecole Polytechnique –
Palaiseau, FR
- Marie Duflot
LORIA & INRIA Nancy, FR
- Cormac Flanagan
University of California – Santa
Cruz, US
- Matthias Függer
TU Wien, AT
- Alexey Gotsman
IMDEA Software – Madrid, ES
- Serge Haddad
ENS – Cachan, FR
- Gerwin Klein
NICTA & UNSW – Sydney, AU
- Igor Komov
TU Wien, AT
- Fabrice Kordon
UPMC, Lab. LIP6 – Paris, FR
- Akash Lal
Microsoft Research India –
Bangalore, IN
- Victor Luchangco
Oracle Corporation –
Burlington, US
- Stephan Merz
LORIA – Nancy, FR
- Uwe Nestmann
TU Berlin, DE
- Thomas Nowak
Ecole Polytechnique –
Palaiseau, FR
- Eric Ruppert
York University – Toronto, CA
- John Rushby
SRI – Menlo Park, US
- Andrey Rybalchenko
TU München, DE
- André Schiper
EPFL – Lausanne, CH
- Klaus Schneider
TU Kaiserslautern, DE
- Philippe Schnoebelen
ENS – Cachan, FR
- Wilfried Steiner
TTTech Computertechnik –
Wien, AT
- Murali Talupur
Intel SCL – Hillsboro, US
- Serdar Tasiran
Koc University – Istanbul, TR
- Helmut Veith
TU Wien, AT
- Christoph Weidenbach
MPI für Informatik –
Saarbrücken, DE
- Jennifer L. Welch
Texas A&M University –
College Station, US
- Josef Widder
TU Wien, AT
- Karsten Wolf
Universität Rostock, DE

