Checking Correctness of Concurrent Objects: Tractable Reductions to Reachability

Ahmed Bouajjani[1], **Michael Emmi**[2], **Constantin Enea**[3], **and Jad Hamza**[3]

1 **Université Paris Diderot and Institut Universitaire de France, Paris, France**
`abou@liafa.univ-paris-diderot.fr`

2 **IMDEA Software Institute, Madrid, Spain**
`michael.emmi@imdea.org`

3 **Université Paris Diderot, Paris, France**
`{cenea,jad.hamza}@liafa.univ-paris-diderot.fr`

Abstract

Efficient implementations of concurrent objects such as semaphores, locks, and atomic collections including stacks and queues are vital to modern computer systems. Programming them is however error prone. To minimize synchronization overhead between concurrent object-method invocations, implementors avoid blocking operations like lock acquisition, allowing methods to execute concurrently. However, concurrency risks unintended inter-operation interference. Their correctness is captured by *observational refinement* which ensures conformance to atomic reference implementations. Formally, given two libraries L_1 and L_2 implementing the methods of some concurrent object, we say L_1 *refines* L_2 if and only if every computation of every program using L_1 would also be possible were L_2 used instead.

Linearizability [11], being an equivalent property [8, 5], is the predominant proof technique for establishing observational refinement: one shows that each concurrent execution has a linearization which is a valid sequential execution according to a specification, given by an abstract data type or atomic reference implementation.

However, checking linearizability is intrinsically hard. Indeed, even in the case where method implementations are finite-state and object specifications are also finite-state, and when a fixed number of threads (invoking methods in parallel) is considered, the linearizability problem is EXPSPACE-complete [9], and it becomes undecidable when the number of threads is unbounded [3]. These results show in particular that there is a complexity/decidability gap between the problem of checking linearizability and the problem of checking reachability (i.e., the dual of checking safety/invariance properties), the latter being, PSPACE-complete and EXPSPACE-complete in the above considered cases, respectively.

We address here the issue of investigating cases where tractable reductions of the observational refinement/linearizability problem to the reachability problem, or dually to invariant checking, are possible. Our aim is (1) to develop algorithmic approaches that avoid a systematic exploration of all possible linearizations of all computations, (2) to exploit existing techniques and tools for efficient invariant checking to check observational refinement, and (3) to establish decidability and complexity results for significant classes of concurrent objects and data structures.

We present two approaches that we have proposed recently. The first approach [5] introduces a parameterized approximation schema for detecting observational refinement violations. This approach exploits a fundamental property of shared-memory library executions: their histories are *interval orders*, a special case of partial orders which admit *canonical representations* in which each operation o is mapped to a positive-integer-bounded interval $I(o)$. Interval orders are equipped with a natural notion of *length*, which corresponds to the smallest integer constant for which an interval mapping exists. Then, we define a notion of bounded-interval-length analysis, and demonstrate its efficiency, in terms of complexity, coverage, and scalability, for detecting observational refinement bugs.

35th IARCS Annual Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015).
Editors: Prahladh Harsha and G. Ramalingam; pp. 2–4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The second approach [4] focuses on a specific class of abstract data types, including common concurrent objects and data structures such as stacks and queues. We show that for this class of objects, the linearizability problem is actually as hard as the control-state reachability problem. Indeed, we prove that in this case, the existence of linearizability violations (i.e., finite computations that are not linearizable), can be captured *completely* by a finite number of finite-state automata, even when an unbounded number of parallel operations is allowed (assuming that libraries are data-independent).

Related work. Several semi-automated approaches for proving linearizability, and thus observational refinement, have relied on annotating operation bodies with *linearization points* [2, 12, 13, 15, 16], to reduce the otherwise-exponential space of possible history linearizations to one single linearization. These methods often rely on programmer annotation, and do not admit conclusive evidence of a violation in the case of a failed proof.

Existing automated methods for proving linearizability of an atomic object implementation are also based on reductions to safety verification [1, 10, 15]. Abdulla et al. [1] is and Vafeiadis [15] consider implementations where operation's *linearization points* are fixed to particular source-code locations. Such approaches are incomplete since not all implementations have fixed linearization points (see for instance [7]). Aspect-oriented proofs [10] reduce linearizability to the verification of four simpler safety properties. However, this approach has only been applied to queues, and has not produced a fully automated and complete proof technique. Dodds et al. [7] prove linearizability of stack implementations with an automated proof assistant. Their approach does not lead to full automation however, e.g., by reduction to safety verification.

Automated approaches for detecting linearizability violations such as Line-Up [6] enumerate the exponentially-many possible history linearizations. This exponential cost effectively limits such approaches to executions with few operations. Colt [14]'s approach mitigates this cost with programmer-annotated linearization points, as in the previously-mentioned approaches, and ultimately suffers from the same problem: a failed proof only indicates incorrect annotation.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.2.5 Testing and Debugging

Keywords and phrases Concurrent objects, Linearizability, Verification, Bug detection

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2015.2

Category Invited Talk

References

1 Parosh Aziz Abdulla, Frédéric Haziza, Lukás Holík, Bengt Jonsson, and Ahmed Rezine. An integrated specification and verification technique for highly concurrent data structures. In *TACAS*, pages 324–338, 2013.

2 Daphna Amit, Noam Rinetzky, Thomas W. Reps, Mooly Sagiv, and Eran Yahav. Comparison under abstraction for verifying linearizability. In *CAV'07*, volume 4590 of *LNCS*, pages 477–490, 2007.

3 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. Verifying concurrent programs against sequential specifications. In *ESOP'13*. Springer, 2013.

4 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. On reducing linearizability to state reachability. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2015.

5 Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Jad Hamza. Tractable refinement checking for concurrent objects. In *POPL'15*. ACM, 2015.

6 Sebastian Burckhardt, Chris Dern, Madanlal Musuvathi, and Roy Tan. Line-Up: a complete and automatic linearizability checker. In *PLDI'10*, pages 330–340. ACM, 2010.

7 Mike Dodds, Andreas Haas, and Christoph M. Kirsch. A scalable, correct time-stamped stack. In *POPL'15*. ACM, 2015.

8 Ivana Filipovic, Peter W. O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theor. Comput. Sci.*, 411(51-52):4379–4398, 2010.

9 Jad Hamza. On the complexity of linearizability. In *3rd Intern. Conf. on Networked Systems, NETYS'15, Agadir, Morocco*, volume 9466 of *Lecture Notes in Computer Science*. Springer, 2015.

10 Thomas A. Henzinger, Ali Sezgin, and Viktor Vafeiadis. Aspect-oriented linearizability proofs. In *CONCUR*, pages 242–256, 2013.

11 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

12 Yang Liu, Wei Chen, Yanhong A. Liu, and Jun Sun. Model checking linearizability via refinement. In *FM'09*, volume 5850 of *LNCS*, pages 321–337, 2009.

13 Peter W. O'Hearn, Noam Rinetzky, Martin T. Vechev, Eran Yahav, and Greta Yorsh. Verifying linearizability with hindsight. In *PODC'10*, pages 85–94. ACM, 2010.

14 Ohad Shacham, Nathan Grasso Bronson, Alex Aiken, Mooly Sagiv, Martin T. Vechev, and Eran Yahav. Testing atomicity of composed concurrent operations. In *OOPSLA'11*, pages 51–64. ACM, 2011.

15 Viktor Vafeiadis. Automatically proving linearizability. In *CAV'10*, volume 6174 of *LNCS*, pages 450–464, 2010.

16 Shao Jie Zhang. Scalable automatic linearizability checking. In *ICSE'11*, pages 1185–1187. ACM, 2011.