Honest Linearizability*

ARTHUR OLIVEIRA VALE, Yale University, USA

Background. In the concurrency community, linearizability [10] is the standard of correctness for concurrent objects. While originally it could only specify non-blocking objects consistent with sequential objects, over time it was extended into more [13] and more general criteria [3], arriving at a criterion that is complete for distributed tasks [6] and blocking and non-blocking concurrent objects [14, 15]. In addition, and using insights from concurrent game semantics [5, 11], Oliveira Vale et al. [14] show that their generalization of linearizability, *compositional linearizability*, is a compositional criterion for the specification of concurrent libraries implemented as collections of parallel programs [15].

However, in their seminal paper Alpern and Schneider [2], show that every property of a concurrent object is the intersection of a safety and a liveness property. Linearizability is always a safety property and therefore is not enough to fully specify concurrent objects. This is why concurrent data structure implementations [4, 12] are typically shown correct by showing that they are linearizable to an appropriate linearized specification (safety), and then *separately* (i.e. directly on the un-linearized executions) shown to satisfy a progress property [9] (liveness).

As an example, consider a sequential counter specified as the set of traces it generates:

$$\nu_{\mathsf{Counter}} := \{\epsilon, \mathsf{get}, \mathsf{get} \cdot 0, \mathsf{inc}, \mathsf{inc} \cdot \mathsf{ok}, \mathsf{inc} \cdot \mathsf{ok} \cdot \mathsf{get}, \mathsf{inc} \cdot \mathsf{ok} \cdot \mathsf{get} \cdot 1, \ldots\}$$

The typical safety property one would state about the counter is functional correctness: that in every finite play $s \cdot \mathsf{get}$ if Proponent responds to get then get receives the number $\#\mathsf{inc}(s)$ of inc invocations in s as a response. For a concurrent counter ν'_{Counter}, if one sees a call to get there is no guarantee of what the return might be, as any number of threads may call inc after the get call, which might take effect before get determines its return value. In lieu of a functional correctness, the usual consistency condition for a concurrent counter is that ν'_{Counter} should be linearizable w.r.t. to ν_{Counter}, written $\nu'_{\mathsf{Counter}} \rightsquigarrow \nu_{\mathsf{Counter}}$.

Meanwhile, the typical liveness property for a sequential object is termination: every call to the counter eventually receives a response. In the concurrent setting, one replaces this for progress properties, such as lock-freedom, wait-freedom, deadlock-freedom, etc. Typically, one has to verify these properties directly on ν'_{Counter}. This has a different flavor from functional correctness, where we only check that $\nu'_{\mathsf{Counter}} \rightsquigarrow \nu_{\mathsf{Counter}}$ and then that ν_{Counter} is functionally correct. The issue is that linearizability does not reflect progress properties, in the sense that it could be that the linearized specification ν_{Counter} satisfies the desired progress property, but ν'_{Counter} does not.

Honest Linearizability. In this talk, we describe an extension to the framework of Oliveira Vale et al. [15] that allows linearizability to play the role of both a safety property and a liveness property. As a result, we show that if an object is linearizable $\nu_{\mathsf{Counter}'} \overset{\mathsf{h}}{\rightsquigarrow} \nu_{\mathsf{Counter}}$ and then if ν_{Counter} satisfies one of the traditional progress property then so does the concurrent implementation $\nu_{\mathsf{Counter}'}$.

One of the core ideas to achieve this is the notion of a winning strategy from game semantics models of logics [1]. Historically, they appear as a solution to the problem that total strategies do not compose because of the possibility of infinite stuttering. To solve this, games are equipped with a notion of "winning plays" which helps model terminating strategies. Then, while total strategies do not compose, total winning strategies do.

*Joint work with Zhongye Wang and Zhong Shao

Author's address: Arthur Oliveira Vale, arthur.oliveiravale@yale.edu, Yale University, New Haven, CT, USA.

Since we want to state arbitrary liveness properties, as opposed to just termination, we equip not the games but the strategies with winning plays, which we call a *win condition*. A win condition ω over A is simply any subset of plays (finite or infinite) over the game A. A winning strategy is then a pair $\langle \nu \vDash \omega \rangle$ of a strategy $\nu : A$ over A such that its set of P-maximal plays are all contained in ω, $\text{P-max}(\nu) \subseteq \omega$. From the point of view of object specifications, we view ν as its safety property and ω as its liveness property. The set of behaviors that it produces are then $\nu \cap \omega$, per the Alpern-Schneider decomposition. With appropriate assumptions, every object may instead be fully characterized by ω, which is helpful in practice. By using the computational interpretation of linearizability from Oliveira Vale et al. [15], which shows a correspondence between linearizability and the copycat strategy, we can derive an appropriate linearizability criterion for winning strategies which we call *honest linearizability* and write $\langle \nu' \vDash \omega' \rangle \overset{\text{h}}{\rightsquigarrow} \langle \nu \vDash \omega \rangle$.

This creates a challenge, however. We model concurrent strategies as non-deterministic strategies built out of non-alternating plays. To show that winning strategies compose, we still would like to assume totality, which plays a technical role in showing several important lemmas about P-maximal plays of strategy compositions. Concurrent objects, however, typically satisfy much weaker termination properties, such as minimal progress (if there is a pending invocation, then some pending invocation, potentially by a different thread, gets a response). This means that we may not assume termination, so we must find an alternate route to obtain that total strategies compose. Our idea, inspired by models of non-deterministic programming languages [7, 8], is to make silent divergence observable by marking non-terminating invocations q with a special divergence token $q \cdot \circlearrowleft$. Composition is then appropriately modified to generate divergences in the case of infinite stuttering. Total strategies (in the sense that every question q has a response, be it an answer or a divergence token) are seen to compose in this more precise model.

The game semantics model we arrive at can state arbitrary liveness properties and therefore provides a very flexible model for verification, our intended application. Particularly interesting is that the model is compatible with both partial and total correctness. The reason for this is that we may always take $\omega = P_A$ (i.e. ω admits any play as winning). Then, the object $\langle \nu \vDash P_A \rangle$ represents a partially correct object ν. In particular, $\langle \nu' \vDash P_A \rangle \overset{\text{h}}{\rightsquigarrow} \langle \nu \vDash P_A \rangle$ is equivalent to $\nu' \rightsquigarrow \nu$. Meanwhile, if ω is taken to be some maximal progress property, then we obtain total correctness. For instance, let starvf be the starvation-freedom property (every invocation returns under fair scheduling). Then $\langle \nu \vDash \text{starvf} \rangle$ stands for total correctness under fair scheduling. The model, however, allows for properties that neither correspond to total or partial correctness. This happens in practice with minimal progress properties, but also when we tensor partially correct objects with totally correct objects.

From the point of view of linearizability, we retrieve the key results of compositional linearizability [15] for honest linearizability, including observational refinement and locality, which play a key role in deriving the compositional structure of linearizable implementations. Since honest linearizability subsumes linearizability, we obtain an improved and compatible semantics for verification of concurrent objects with the previous development in Oliveira Vale et al. [15]. Moreover, we show honest linearizability reflects all the usual progress properties. This means that it is enough to verify honest linearizability against a linearized specification satisfying the desired progress property to obtain that the implementations satisfy the same progress property. We have extended their program logic to reason about honest linearizability and shown several interesting concurrent object implementations are honestly linearizable.

I believe these results demonstrate a pleasant instance of how game semantics techniques help resolve longstanding problems in the compositional verification of complex systems, with interesting theoretical insights and practical applications.

REFERENCES

[1] Samson Abramsky and Radha Jagadeesan. 1992. Games and full Completeness for multiplicative Linear Logic. In *Foundations of Software Technology and Theoretical Computer Science*, Rudrapatna Shyamasundar (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 291–301.

[2] Bowen Alpern and Fred B. Schneider. 1985. Defining liveness. *Inform. Process. Lett.* 21, 4 (1985), 181–185. https://doi.org/10.1016/0020-0190(85)90056-0

[3] Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. 2015. Specifying Concurrent Problems: Beyond Linearizability and up to Tasks. In *Proceedings of the 29th International Symposium on Distributed Computing - Volume 9363 (DISC 2015)*. Springer-Verlag, Berlin, Heidelberg, 420âĂŞ435. https://doi.org/10.1007/978-3-662-48653-5_28

[4] Mike Dodds, Andreas Haas, and Christoph M. Kirsch. 2015. A Scalable, Correct Time-Stamped Stack. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 233âĂŞ246. https://doi.org/10.1145/2676726.2676963

[5] Dan R. Ghica and Andrzej S. Murawski. 2004. Angelic Semantics of Fine-Grained Concurrency. In *Foundations of Software Science and Computation Structures*, Igor Walukiewicz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 211–225.

[6] Éric Goubault, Jérémy Ledent, and Samuel Mimram. 2018. Concurrent Specifications Beyond Linearizability. In *22nd International Conference on Principles of Distributed Systems (OPODIS 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Jiannong Cao, Faith Ellen, Luis Rodrigues, and Bernardo Ferreira (Eds.), Vol. 125. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 28:1–28:16. https://doi.org/10.4230/LIPIcs.OPODIS.2018.28

[7] William John Gowers and James David Laird. 2018. A Fully Abstract Game Semantics for Countable Nondeterminism. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK (LIPIcs)*, Dan R. Ghica and Achim Jung (Eds.), Vol. 119. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24:1–24:18. https://doi.org/10.4230/LIPICS.CSL.2018.24

[8] R. Harmer and G. McCusker. 1999. A fully abstract game semantics for finite nondeterminism. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. 422–430. https://doi.org/10.1109/LICS.1999.782637

[9] Maurice Herlihy and Nir Shavit. 2011. On the Nature of Progress. In *Principles of Distributed Systems*, Antonio Fernàndez Anta, Giuseppe Lipari, and Matthieu Roy (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 313–328.

[10] Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (jul 1990), 463âĂŞ492. https://doi.org/10.1145/78969.78972

[11] James Laird. 2001. A Game Semantics of Idealized CSP. *Electronic Notes in Theoretical Computer Science* 45 (2001), 232–257. https://doi.org/10.1016/S1571-0661(04)80965-4 MFPS 2001,Seventeenth Conference on the Mathematical Foundations of Programming Semantics.

[12] Maged M. Michael and Michael L. Scott. 1996. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*. Association for Computing Machinery, New York, NY, USA, 267âĂŞ275. https://doi.org/10.1145/248052.248106

[13] Gil Neiger. 1994. Set-Linearizability. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '94)*. Association for Computing Machinery, New York, NY, USA, 396. https://doi.org/10.1145/197917.198176

[14] Arthur Oliveira Vale, Zhong Shao, and Yixuan Chen. 2023. A Compositional Theory of Linearizability. *Proc. ACM Program. Lang.* 7, POPL, Article 38 (jan 2023), 32 pages. https://doi.org/10.1145/3571231

[15] Arthur Oliveira Vale, Zhong Shao, and Yixuan Chen. 2024. A Compositional Theory of Linearizability. *J. ACM* 71, 2, Article 14 (apr 2024), 107 pages. https://doi.org/10.1145/3643668