Higher-Order Linearisability

Andrzej S. Murawski[1] and Nikos Tzevelekos[2]

1 **University of Warwick**
2 **Queen Mary University of London**

Abstract

Linearisability is a central notion for verifying concurrent libraries: a library is proven correct if its operational history can be rearranged into a sequential one that satisfies a given specification. Until now, linearisability has been examined for libraries in which method arguments and method results were of ground type, including libraries parameterised with such methods. In this paper we extend linearisability to the general higher-order setting: methods can be passed as arguments and returned as values. A library may also depend on abstract methods of any order.

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1. Introduction

Software libraries provide implementations of routines, often of specialised nature, to facilitate code reuse and modularity. To support the latter, they should follow specifications that describe the range of acceptable behaviours for correct and safe deployment. Adherence to specifications can be formalised using the classic notion of contextual approximation (refinement), which scrutinises the behaviour of code in any possible context. Unfortunately, the quantification makes it difficult to prove contextual approximations directly, which motivates research into sound techniques for establishing it.

In the concurrent setting, a notion that has been particularly influential is that of *linearisability* [5]. Linearisability requires that, for each history generated by a library, one should be able to find another history from the specification (a *linearisation*), which matches the former up to certain rearrangements of events. In the original formulation by Herlihy and Wing [5], these permutations were not allowed to disturb the order between library returns and client calls. Moreover, linearisations were required to be *sequential* traces, that is, sequences of method calls immediately followed by their returns.

In this paper we shall work with *open higher-order* libraries, which provide implementations of *public* methods and may themselves depend on *abstract* ones, to be supplied by parameter libraries. The classic notion of linearisability only applies to closed libraries (without abstract methods). Additionally, both method arguments and results had to be of *ground* type. The closedness limitation was recently lifted in [1], which distinguished between public and abstract methods, and in [6], where the methods are called implemented and callable respectively in the context of generalised weak memory. However, the allowable methods were still restricted to first-order functions of type $\mathsf{int} \to \mathsf{int}$. In this paper, we present linearisability for general higher-order concurrent libraries, where methods can be of arbitrary higher-order types. In doing so, we also propose a corresponding notion of sequential history for higher-order library interactions.

We examine libraries L that can interact with their environments by means of public and abstract methods: a library L with abstract methods of types $\Theta = \theta_1, \cdots, \theta_n$ and public methods $\Theta' = \theta'_1, \cdots, \theta'_{n'}$ is written as $L : \Theta \to \Theta'$. We shall work with arbitrary higher-order types generated from the ground types unit and int. Types in Θ, Θ' must always be function types, i.e. their order is at least 1.

A library L may be used in computations by placing it in a context that will keep on calling its public methods (via a client K) as well as providing implementations for the abstract ones (via a parameter library L'). The setting is depicted in Figure 1. Note that, as the library L interacts with K and L', they exchange functions between each other. Consequently, in addition to K making calls to public methods of L and L making calls to its abstract methods, K and L' may also issue calls to

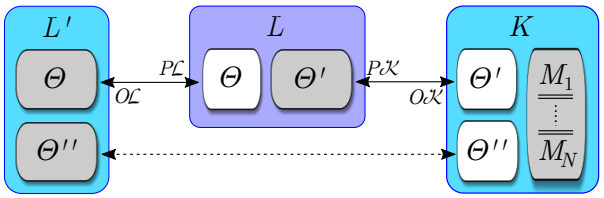

Figure 1 A library $L : \Theta \to \Theta'$ in environment comprising a parameter library $L' : \varnothing \to \Theta, \Theta''$ and a client K of the form $\Theta', \Theta'' \vdash M_1 \| \cdots \| M_N$.

functions that were passed to them as arguments during higher-order interactions. Analogously, L may call functions that were communicated to it via library calls.

Our framework is operational in flavour and draws upon concurrent [8, 2] and operational game semantics [7, 9, 3]. We shall model library use as a game between two participants: *Player (P)*, corresponding to the library L, and *Opponent (O)*, representing the environment (L', K) in which the library was deployed. Each call will be of the form $\mathsf{call}\, m(v)$ with the corresponding return of the shape $\mathsf{ret}\, m(v)$, where v is a value. As we work in a higher-order framework, v may contain functions, which can participate in subsequent calls and returns. Histories will be sequences of *moves*, which are calls and returns paired with thread identifiers. A history is sequential just if every move produced by O is immediately followed by a move by P in the same thread. In other words, the library immediately responds to each call or return delivered by the environment. In contrast to classic linearisability, the move by O and its response by P need not be a call/return pair, as the higher-order setting provides more possibilities (in particular, the P response may well be a call). Accordingly, linearisable higher-order histories can be seen as sequences of atomic segments (linearisation points), starting at environment moves and ending with corresponding library moves.

In the spirit of [1], we are going to consider two scenarios: one in which K and L' share an explicit communication channel (the general case) as well as a situation in which they can only communicate through the library (the encapsulated case). Further, we also handle the case in which extra closure assumptions can be made about the parameter library (the relational case), which can be useful for dealing with a variety of assumptions on the use of parameter libraries that may arise in practice. In each case, we present a candidate definition of linearisability and illustrate it with tailored examples. The suitability of each kind of linearisability is demonstrated by showing that it implies the relevant form of contextual approximation (refinement). We also examine compositionality of the proposed concepts. One of our examples will discuss the implementation of the flat-combining approach [4, 1], adapted to higher-order types.

References

1 A. Cerone, A. Gotsman, and H. Yang. Parameterised linearisability. In *Proceedings of ICALP'14*, volume 8573 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2014.

2 D. R. Ghica and A. S. Murawski. Angelic semantics of fine-grained concurrency. In *Proceedings of FOSSACS*, volume 2987 of *Lecture Notes in Computer Science*, pages 211–225. Springer-Verlag, 2004.

3 D. R. Ghica and N. Tzevelekos. A system-level game semantics. *Electr. Notes Theor. Comput. Sci.*, 286:191–211, 2012.

4 D. Hendler, I. Incze, N. Shavit, and M. Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In *Proceedings of SPAA 2010*, pages 355–364, 2010.

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

6 Radha Jagadeesan, Gustavo Petri, Corin Pitcher, and James Riely. Quarantining weakness - compositional reasoning under relaxed memory models (extended abstract). In *ESOP 2013*, pages 492–511, 2013. URL: http://dx.doi.org/10.1007/978-3-642-37036-6_27, doi:10.1007/978-3-642-37036-6_27.

7 A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. *Theor. Comput. Sci.*, 338(1-3):17–63, 2005.

8 J. Laird. A game semantics of Idealized CSP. In *Proceedings of MFPS'01*, pages 1–26. Elsevier, 2001. ENTCS, Vol. 45.

9 J. Laird. A fully abstract trace semantics for general references. In *Proceedings of ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007.