Parameterised Linearisability

Andrea Cerone[1], Alexey Gotsman[1], and Hongseok Yang[2]

[1] IMDEA Software Institute
[2] University of Oxford

Abstract Many concurrent libraries are parameterised, meaning that they implement generic algorithms that take another library as a parameter. In such cases, the standard way of stating the correctness of concurrent libraries via linearisability is inapplicable. We generalise linearisability to parameterised libraries and investigate subtle trade-offs between the assumptions that such libraries can make about their environment and the conditions that linearisability has to impose on different types of interactions with it. We prove that the resulting parameterised linearisability is closed under instantiating parameter libraries and composing several non-interacting libraries, and furthermore implies observational refinement. These results allow modularising the reasoning about concurrent programs using parameterised libraries and confirm the appropriateness of the proposed definitions. We illustrate the applicability of our results by proving the correctness of a parameterised library implementing flat combining.

1 Introduction

Concurrent libraries encapsulate high-performance concurrent algorithms and data structures behind a well-defined interface, providing a set of methods for clients to call. Many such libraries [5,6,12] are *parameterised*, meaning that they implement generic algorithms that take another library as a parameter and use it to implement more complex functionality (we give a concrete example in §2). Reasoning about the correctness of parameterised libraries is challenging, as it requires considering all possible libraries that they can take as parameters.

Correctness of concurrent libraries is usually stated using *linearisability* [7], which fixes a certain correspondence between the *concrete* library implementation and a (possibly simpler) *abstract* library, whose behaviour the concrete one is supposed to simulate. For example, a high-performance concurrent stack that allows multiple push and pop operations to access the data structure at the same time may be specified by an abstract library where each operation takes effect atomically. However, linearisability considers only *ground* libraries, where all of the library implementation is given, and is thus inapplicable to parameterised ones. In this paper we propose a notion of *parameterised linearisability* (§3 and §4) that lifts this limitation. The key idea is to take into account not only interactions of a library with its client, but also with its parameter library, with the two types of interactions being subject to different conditions.

A challenge we have to deal with while generalising linearisability in this way is that parameterised libraries are often correct only under some assumptions about the context in which they are used. Thus, a parameterised library may assume that the library it takes as a parameter is *encapsulated*, meaning that clients cannot call its methods directly. A parameterised library may also accept as a parameter only libraries satisfying certain properties. For this reason, we actually present three notions of parameterised

linearisability, appropriate for different situations: a general one, which does not make any assumptions about the client or the parameter library, a notion appropriate for the case when the parameter library is encapsulated, and *up-to linearisability*, which allows making assumptions about the parameter library. These notions differ in subtle ways: we find that there is a trade-off between the assumptions that parameterised libraries make about their environment and the conditions that a notion of linearisability has to impose on different types of interactions with it.

We prove that the proposed notions of parameterised linearisability are *contextual* (§5), i.e., closed under parameter instantiation. This includes the case when the parameter library is itself parameterised. On the other hand, when the parameter is an ordinary ground library, this result allows us to derive the classical linearisability of the instantiated library from our notion for the parameterised one. We also prove that parameterised linearisability is *compositional* (§5): if several non-interacting libraries are linearisable, so is their composition. Finally, we show that parameterised linearisability implies *observational refinement* (§6): the behaviours of any complete program using a concrete parameterised library can be reproduced if the program uses a corresponding abstract one instead. All these results allow modularising the reasoning about concurrent programs using parameterised libraries: contextuality and compositionality break the reasoning about complex parameterised libraries into that about individual libraries from which they are constructed; observational refinement then lifts this to complete programs, including clients. The properties of parameterised linearisability we establish also serve to confirm the appropriateness of the proposed definitions.

We illustrate the applicability of our results by proving the up-to linearisability of flat combining [5] (§4), a generic algorithm for converting hard-to-parallelise sequential data structures into concurrent ones.

Due to space constraints, we defer the proofs of most theorems to §B.

2 Parameterised Libraries

We consider ***parameterised libraries*** (or simply libraries) L, which provide some ***public methods*** to their ***clients***. The latter are multi-threaded programs that can call the methods in parallel. In §4 and §6 we introduce a particular syntax for libraries and clients; for now it suffices to treat them abstractly. Our libraries are called parameterised because we allow their method implementations to call ***abstract methods***, whose implementation is left unspecified. Abstract methods are meant to be implemented by another library provided by L's client, which we call the ***parameter library*** of L.

We identify methods by names from a set $\mathcal{M}$, ranged over by m, and threads by identifiers from a set $\mathcal{T}$, ranged over by t. For the sake of simplicity, we assume that methods take a single integer as a parameter and always return an integer. We annotate libraries with types as in $L : M \to M'$, where $M, M' \subseteq \mathcal{M}$ give the sets of abstract and public methods of L, respectively. If $M = \emptyset$ we call L a ***ground library***. The sets M and M' do not have to be disjoint: methods in $M \cap M'$ may be called by L's clients, but their implementation is inherited from the one given by the parameter library.

Example: Flat Combining. Flat combining [5] is a recent synchronisation paradigm, which can be viewed [13] as a parameterised library $\mathsf{FC} : \{m_i\}_{i=1}^n \to \{\mathsf{do\_}m_i\}_{i=1}^n$ for a given set of methods $\{m_i\}_{i=1}^n$. In Figure 1 we show a pseudocode of its implementation, which simplifies the original one in ways orthogonal to our goals. FC takes a library,

whose methods m_i are meant to be executed sequentially, and efficiently turns it into a library with methods do_m_i that can be called concurrently.

As usual, this is achieved by means of mutual exclusion, implemented using a lock, but in a way that is more sophisticated than just acquiring it before calling a method m_i. A thread executing do_m_i first publishes the operation it would like to execute and its parameter in its entry of the requests array. It then spins, trying to acquire the global lock. Having acquired a lock, the thread becomes a *combiner*: it performs the operations requested by all threads, stored in requests, by calling methods m_i of the parameter library and writing the values returned into the retval field of the corresponding entries in requests. Each spinning thread periodically checks this field and stops if some other thread has performed the operation it requested (for simplicity, we assume that nil is a special value that is never returned by any method). This algorithm benefits from cache locality when the combiner executes several operations in sequence, and thus yields good performance even for hard-to-parallelise data structures, such as stacks and queues.

```
LOCK lock;
struct{op,param,retval} requests[NThread];

do_m_i(int z):
  requests[mytid()].op = i;
  requests[mytid()].param = z;
  requests[mytid()].retval = nil;
  do:
    if (lock.tryacquire()):
      for (t = 0; t < NThread; t++):
        if (requests[t].retval == nil):
          int j = requests[t].op;
          int w = requests[t].param;
          requests[t].retval = m_j(w);
      lock.release();
  while (requests[mytid()].retval == nil);
  return requests[mytid()].retval;
```

Figure 1. Flat combining: implementation FC.

In this paper, we develop a framework for specifying and verifying parameterised concurrent libraries. For flat combining, our framework suggests using an ***abstract*** library $\mathsf{FC}^\sharp : \{m_i\}_{i=1}^n \to \{\mathsf{do}\_m_i\}_{i=1}^n$ in Figure 2 as a specification for the ***concrete*** library in Figure 1. $\mathsf{FC}^\sharp$ specifies the expected behaviour of flat combining by using the naive mutual exclusion. Showing that the implementation satisfies this specification in our framework amounts to proving that it is related to $\mathsf{FC}^\sharp$ by *parameterised linearisability*, which we present next.

```
LOCK lock;
do_m_i(int z):
  lock.acquire();
  int retval = m_i(z);
  lock.release();
  return retval;
```

Figure 2. Flat combining: specification $\mathsf{FC}^\sharp$.

3 Histories and Parameterised Linearisability

Histories. Informally, for a concrete library (such as the one in Figure 1) to be correct with respect to an abstract one (such as the one in Figure 2), the two should interact with their environment—the client and the parameter library—in similar ways. In this paper, we assume that different libraries and their clients access disjoint portions of memory, and thus interactions between them are limited to passing parameters and return values at method calls and returns. This is a standard assumption [7], which we believe can be relaxed using existing techniques [4]; see §7 for discussion. We record interactions of a parameterised library $L : M \to M'$ with its environment using *histories* (Definition 1 below), which are certain sequences of ***actions*** of the form

$$\mathsf{Act} ::= (t, \mathsf{call?}\, m'(z)) \mid (t, \mathsf{ret!}\, m'(z)) \mid (t, \mathsf{call!}\, m(z)) \mid (t, \mathsf{ret?}\, m(z)),$$

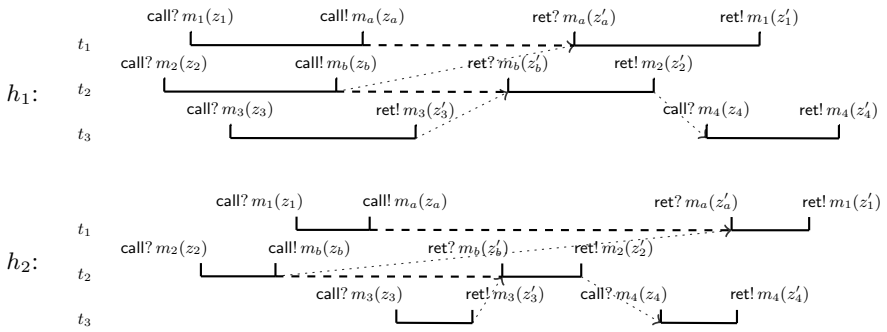


Figure 4. Illustration of histories and parameterised linearisability. A solid line represents a thread executing the code of the parameterised library, and a dashed one, the parameter library.

where $t \in \mathcal{T}$ is the thread performing the action, $m' \in M'$ or $m \in M$ is the method involved, and $z \in \mathbb{Z}$ is the method parameter or a return value.

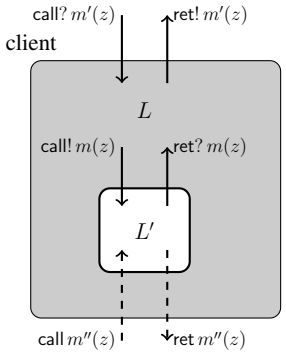


Figure 3. Interactions of a library L with its client and parameter library L'.

We illustrate the meaning of the actions in Figure 3: call? and ret! describe the client invoking public methods m' of the parameterised library L, and call! and ret? the library L invoking implementations of abstract methods m provided by a parameter library L'. We denote the sets of actions corresponding to interactions with these two entities by ClAct and AbsAct, respectively. In the spirit of the opponent-proponent distinction in game semantics [8,10], we annotate actions by ! or ? depending on whether the action was initiated by L or by an external entity, and we denote the corresponding sets of actions by Act! and Act?. We also use sets ActCall?, ActRet!, ActCall! and ActRet? with the expected meaning. Clients can also call methods $m'' \in M \cap M'$ directly, as represented by the dashed lines in the figure. Since such interactions do not involve the library L, we do not include them into Act. Histories are finite sequences of actions with invocations of abstract methods properly nested inside those of public ones.

DEFINITION 1 (Histories) *A **history** $h : M \to M'$ is a finite sequence of actions such that for every t, the projection of h to t's actions is a prefix of a sequence generated by the grammar* SHist *below, where $m \in M$ and $m' \in M'$:*

$$\begin{array}{lll} \mathsf{SHist} & ::= \varepsilon \mid (t, \mathsf{call?}\, m'(z))\, \mathsf{IntSHist}\, (t, \mathsf{ret!}\, m'(z')) \mid \mathsf{SHist}\ \mathsf{SHist} \\ \mathsf{IntSHist} & ::= \varepsilon \mid (t, \mathsf{call!}\, m(z))\, (t, \mathsf{ret?}\, m(z')) \mid \mathsf{IntSHist}\ \mathsf{IntSHist} \end{array}$$

We denote the set of histories by Hist. See Figure 4 for examples. In this paper, we focus on safety properties of libraries and thus let histories be finite. This assumption is also taken by the classical notion of linearisability [7] and can be relaxed as described in [3] (§7). For a history h and $A \subseteq \mathsf{Act}$, we let $h|_A$ be the projection of h onto actions in A and we denote the i-th action in h by $h(i)$.

Parameterised Linearisability. We would like the notion of correctness of a concrete library $L : M \to M'$ with respect to an abstract one $L^\sharp : M \to M'$ to imply *observational refinement*. Informally, this property means that $L^\sharp$ can be used to replace L in any program (consisting of a client, the library and an instantiation of the parameter library) while keeping its observable behaviours reproducible; a formal definition is given in §6. While this notion is intuitive, establishing it between two libraries directly is challenging because of the quantification over all possible programs they can be used by. We therefore set out to find a correctness criterion that compares the concrete and abstract libraries in isolation and thus avoids this quantification. For ground libraries, ***linearisability*** [7] formulates such a criterion by matching a history h_1 of L with a history h_2 of $L^\sharp$ that yields the same client-observable behaviour. The following definition generalises it to parameterised libraries.

DEFINITION 2 (Parameterised linearisability: general case) *A history $h_1 : M \to M'$ is **linearised** by another one $h_2 : M \to M'$, written $h_1 \sqsubseteq h_2$, if there exists a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that*

$$\forall i.\, h_1(i) = h_2(\pi(i)) \wedge (\forall j.\, i < j \wedge ((\exists t.\, h_1(i) = (t, -) \wedge h_2(j) = (t, -)) \vee (h_1(i) \in \mathsf{Act!} \wedge h_1(j) \in \mathsf{Act?})) \implies \pi(i) < \pi(j)).$$

For sets of histories H_1, H_2 we let $H_1 \sqsubseteq H_2 \iff \forall h_1 \in H_1.\, \exists h_2 \in H_2.\, h_1 \sqsubseteq h_2$.

In §4 we show how to generate all histories of a library in a particular language and define linearisability on libraries by the $\sqsubseteq$ relation on their sets of histories. For now we explain the above abstract definition. According to it, a history h_1 is linearised by a history h_2 when the latter is a permutation of the former preserving the order of actions within threads and the precedence relation between the actions initiated by the library and those initiated by its environment. As we explain below, we have $h_1 \sqsubseteq h_2$ for the histories h_1, h_2 in Figure 4. Hence, parameterised linearisability is able to match a history of a concurrent library with a simpler one where every contiguous block of library execution (e.g., the one between $(t_1, \mathsf{call?}\, m_1(z_1))$ and $(t_1, \mathsf{call!}\, m_a(z_a))$) is executed without interleaving with other such blocks. On the other hand, $h_2 \not\sqsubseteq h_1$, since $(t_1, \mathsf{call!}\, m_a(z_a))$ precedes $(t_3, \mathsf{call?}\, m_3(z_3))$ in h_2, but not in h_1.

When $h_1, h_2 : \emptyset \to M'$, i.e., these are histories of a ground library and thus contain only call? and ret! actions, Definition 2 coincides with a variant of the classical linearisability [7], which requires preserving the order between ret! and call? actions. For example, Definition 2 requires preserving the order between $(t_2, \mathsf{ret!}\, m_2(z_2'))$ and $(t_3, \mathsf{call?}\, m_4(z_4))$ in h_1 from Figure 4 (shown by a diagonal arrow). This requirement is needed for linearisability to imply observational refinement: informally, during the interval of time between $(t_2, \mathsf{ret!}\, m_2(z_2'))$ and $(t_3, \mathsf{call?}\, m_4(z_4))$ in an execution of a program producing h_1, both threads t_2 and t_3 execute pieces of client code, which can communicate via the client memory. To preserve the behaviour of the client when replacing the concrete library in the program by an abstract one in observational refinement, this communication must not be affected, and, for this, the abstract library has to admit a history in which the order between the above actions is preserved.

When $h_1, h_2 : M \to M'$ correspond to a non-ground parameterised library, i.e., $M \neq \emptyset$, a similar situation arises with communication between the methods of the parameter library executing in different threads. For this reason, our generalisation of linearisability requires preserving the order between call! and ret? actions, such as

$(t_2, \mathsf{call!}\, m_b(z_b))$ and $(t_1, \mathsf{ret?}\, m_a(z'_a))$ in Figure 4; this requirement is dual to the one considered in classical linearisability. It is not enough, however. Definition 2 also requires preserving the order between call! and call?, as well as ret! and ret? actions, e.g., $(t_3, \mathsf{ret!}\, m_3(z'_3))$ and $(t_2, \mathsf{ret?}\, m_b(z'_b))$ in Figure 4. In the case when $M \cap M' \neq \emptyset$, this is also required to validate observational refinement. For example, during the interval of time between $(t_3, \mathsf{ret!}\, m_3(z'_3)$ and $(t_2, \mathsf{ret?}\, m_b(z'_b))$ in an execution producing h_1, the client code in thread t_3 can call a method $m'_b \in M \cap M'$ of the parameter library (cf. the dashed arrows in Figure 3). The code of the method m'_b executed by t_3 can then communicate with that of the method m_b executed by t_2, and to preserve this communication, we need to preserve the order between $(t_3, \mathsf{ret!}\, m_3(z'_3))$ and $(t_2, \mathsf{ret?}\, m_b(z'_b))$.

In §5 and §6 we prove that the above notion of linearisability indeed validates observational refinement. If the library $L : M \to M'$ producing the histories h_1, h_2 in Definition 2 is such that $M \cap M' = \emptyset$, then the client cannot directly call methods of its parameter library, and, as we show, parameterised linearisability can be weakened without invalidating observational refinement.

DEFINITION 3 (Parameterised linearisability: encapsulated case) *For $h_1, h_2 : M \to M'$ with $M \cap M' = \emptyset$ we let $h_1 \sqsubseteq_{\mathsf{e}} h_2$ if there exists a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that*

$$\forall i.\, h_1(i) = h_2(\pi(i)) \wedge (\forall j.\, i < j \wedge ((\exists t.\, h_1(i) = (t, -) \wedge h_2(j) = (t, -)) \vee (h_1(i), h_1(j)) \in (\mathsf{ActRet!} \times \mathsf{ActCall?}) \cup (\mathsf{ActCall!} \times \mathsf{ActRet?})) \implies \pi(i) < \pi(j)).$$

Since this definition does not take into account the order between $(t_1, \mathsf{call!}\, m_a(z_a))$ and $(t_3, \mathsf{call?}\, m_3(z_3))$ in h_2 from Figure 4, we have $h_2 \sqsubseteq_{\mathsf{e}} h_1$ even though $h_2 \not\sqsubseteq h_1$.

Definitions 2 and 3 do not make any assumptions about the implementation of the parameter library. However, sometimes the correctness of a parameterised library can only be established under certain assumptions about the behaviour of its parameter. In particular, this is the case for the flat combining library from §2. In its implementation FC from Figure 1, a request by a thread t to execute a method m_i of the parameter library can be fulfilled by another thread t' who happens to act as a combiner; in contrast, the specification $\mathsf{FC}^\sharp$ in Figure 2 pretends that m_i is executed in the requesting thread. Thus, FC and $\mathsf{FC}^\sharp$ will behave differently if we supply as their parameter a library whose methods depend on the identifiers of executing threads (e.g., with m_i implemented as "return mytid()"). As a consequence, FC does not simulate $\mathsf{FC}^\sharp$. On the other hand, this will be the case if we restrict ourselves to parameter libraries whose behaviour is independent of thread identifiers. The following version of parameterised linearisability allows us to use such assumptions, formulated as closure properties on histories of interactions between a parameterised library and its parameter. Given a history h, let $\overline{h}$ be the history obtained by swapping ! and ? actions in h.

DEFINITION 4 (Up-to linearisability) *For $h_1, h_2 : M \to M'$ such that $M \cap M' = \emptyset$ and a binary relation $\mathcal{R}$ on histories of type $\emptyset \to M$, we say that h_1 **is linearised by** h_2 **up to** $\mathcal{R}$, written $h_1 \sqsubseteq_{\mathcal{R}} h_2$, if $(h_1|_{\mathsf{ClAct}}) \sqsubseteq (h_2|_{\mathsf{ClAct}})$ and $\overline{(h_1|_{\mathsf{AbsAct}})}\ \mathcal{R}\ \overline{(h_2|_{\mathsf{AbsAct}})}$.*

For flat combining, a suitable relation $\mathcal{R}_{\mathsf{t}}$ relates two histories if one can be obtained from the other by replacing thread identifiers of some pairs of a call and a corresponding (if any) return action. There are other useful choices of $\mathcal{R}$, such as equivalence up to commuting abstract method invocations [6].

So far we have defined our notions of linearisability abstractly, on sets of histories. We next introduce a language for parameterised libraries and show how to generate sets

of histories of a library in this language. This lets us lift the notion of linearisability to libraries and prove that FC in Figure 1 is indeed linearised up to $\mathcal{R}_{\mathsf{t}}$ by $\mathsf{FC}^\sharp$ in Figure 2.

4 Lifting Linearisability to Libraries

Library Syntax. We use the following language to define libraries:

$$L \ ::= \ \langle \mathsf{public} : B; \mathsf{private} : B \rangle \qquad B \ ::= \ \varepsilon \ \mid \ (m \Leftarrow C); B \ \mid \ (\mathsf{abstract}\ m); B$$
$$C \ ::= \ c \ \mid \ m() \ \mid \ C; C \ \mid \ \mathsf{if}(E)\ \mathsf{then}\ C\ \mathsf{else}\ C \ \mid \ \mathsf{while}(E)\ C$$

A parameterised library L is a collection of methods, some implemented by commands C and others declared as abstract, meant to be implemented by a parameter library. Methods can be public or private, with only the former made available to clients. In §5 and §6 we extend the language to complete programs, consisting of a multithreaded client using a parameterised library with its parameter instantiated. In particular, we introduce private methods here to define parameter library instantiation in §5.

In commands, c ranges over ***primitive commands*** from a set PComm, and E over expressions, whose set we leave unspecified. The command $m()$ invokes the method m; it does not mention its parameter or return value, since, as we explain below, these are passed via dedicated thread-local memory locations. We consider only well-formed libraries where a method is declared at most once and every method called is declared. We identify libraries up to the order of method declarations and α-renaming of private non-abstract methods. For a library $L = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}} \rangle$ we have $L : \mathsf{Abs}(L) \to \mathsf{Pub}(L)$, where $\mathsf{Pub}(L)$ is the set of methods declared in B_{pub}, and $\mathsf{Abs}(L)$ of those declared as abstract in B_{pub} or B_{pvt}.

Linearisability on Libraries and the Semantics Idea. We now show how to generate the set of histories $[\![L]\!] \in 2^{\mathsf{Hist}}$ of a library L. Then we let a library L_1 be ***linearised by*** a library L_2, written $L_1 \sqsubseteq L_2$, if $[\![L_1]\!] \sqsubseteq [\![L_2]\!]$; similarly for $\sqsubseteq_{\mathsf{e}}$ and $\sqsubseteq_{\mathcal{R}}$.

We actually generate all library *traces*, which, unlike histories, also record its internal actions. Let us extend the set of actions Act with elements of the forms (t, c) for $c \in \mathsf{PComm}$, $(t, \mathsf{call}\, m(z))$ and $(t, \mathsf{ret}\, m(z))$, leading to a set TrAct. The latter two kinds of actions correspond to calls and returns between methods implemented inside the library. A ***trace*** τ is a finite sequence of elements in TrAct; we let $\mathsf{Traces} = \mathsf{TrAct}^*$.

The denotation $[\![L]\!]$ of a library $L : M \to M'$ includes the histories extracted from traces that L produces in any possible environment, i.e., assuming that client threads perform any sequences of calls to methods in M' with arbitrary parameter values and that abstract methods in M return arbitrary values. The definition of $[\![L]\!]$ follows the intuitive semantics of our programming language. An impatient reader can skip it on first reading and jump directly to Theorem 1 at the end of this section.

Heaps and Primitive Command Semantics. Let Locs be the set of memory locations. As we noted in §3, we impose a standard restriction that different libraries and their clients access different sets of memory locations, except the ones used for method parameter passing. Formally, we assume that each library L is associated with a set of its locations $\mathsf{Locs}_L \subseteq \mathsf{Locs}$. The state of L is thus given by a ***heap*** $\sigma \in \mathsf{Locs}_L \to \mathbb{Z}$. We assume a special subset of locations $\{\mathsf{arg}_t\}_{t \in \mathcal{T}}$ belonging to every Locs_L, which we use to pass parameters and return values for method invocations in thread t.

We assume that the execution of primitive commands and the evaluation of expressions are atomic. The semantics of a primitive command $c \in \mathsf{PComm}$ used by a

Traces of commands $(\!|C|\!)_t : (\mathcal{M} \times \mathcal{T} \to 2^{\mathsf{Traces}}) \to 2^{\mathsf{Traces}}$

$$(\!|c|\!)_t\eta = \{(t, c)\} \qquad (\!|C_1; C_2|\!)_t\eta = \{\tau_1\tau_2 \mid \tau_1 \in (\!|C_1|\!)_t\eta \wedge \tau_2 \in (\!|C_2|\!)_t\eta\}$$

$$(\!|\mathsf{if}(E)\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2|\!)_t\eta = (t, \mathsf{assume}(E))\,((\!|C_1|\!)_t\eta) \cup (t, \mathsf{assume}(!E))((\!|C_2|\!)_t\eta)$$

$$(\!|\mathsf{while}(E)\ C|\!)_t\eta = ((t, \mathsf{assume}(E))([\![C]\!]\eta))^*(t, \mathsf{assume}(!E))$$

$$(\!|m()|\!)_t\eta = \begin{cases} \{(t, \mathsf{call!}\, m(z))\,\tau\,(t, \mathsf{ret?}\, m(z')) \mid \tau \in \eta(m,t) \wedge z, z' \in \mathbb{Z}\}, & \text{if } m \in M \\ \{(t, \mathsf{call}\, m(z))\,\tau\,(t, \mathsf{ret}\, m(z')) \mid \tau \in \eta(m,t) \wedge z, z' \in \mathbb{Z}\}, & \text{otherwise} \end{cases}$$

Traces of library bodies

$$\mathcal{F} : (\mathcal{M} \times \mathcal{T} \to 2^{\mathsf{Traces}}) \to (\mathcal{M} \times \mathcal{T} \to 2^{\mathsf{Traces}})$$

$$(\mathcal{F}(\eta))(m,t) = \begin{cases} \eta(m,t) \cup ((\!|C|\!)_t\eta), & \text{if } (m \Leftarrow C) \text{ appears in } L \\ \{\varepsilon\}, & \text{if } m \in M \\ \emptyset, & \text{otherwise} \end{cases}$$

$$(\!|B|\!) : \mathcal{M} \times \mathcal{T} \to 2^{\mathsf{Traces}}$$

$$(\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!) = \mathsf{lfp}(\mathcal{F})$$

Traces of libraries $(\!|L : M \to M'|\!) : 2^{\mathsf{Traces}}$

$$(\!|L|\!) = \mathsf{prefix}\left(\bigcup\nolimits_{k>0} \Big\|_{t=1}^{k} \left(\bigcup\nolimits_{\substack{z,z' \in \mathbb{Z} \\ m \in M' \setminus M}} (t, \mathsf{call?}\, m(z))\,((\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)(m,t))\,(t, \mathsf{ret!}\, m(z'))\right)^*\right)$$

Figure 5. Possible traces of a library $L = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}} \rangle : M \to M'$. Here $\big\|_{t=1}^{k} T_t$ denotes the set of all interleavings of traces from the sets $T_1, \ldots, T_k$.

$$\sigma \rightsquigarrow^{L}_{\mathsf{call}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma, \sigma(\mathsf{arg}_t) = z \qquad \sigma \rightsquigarrow^{L}_{\mathsf{ret}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma, \sigma(\mathsf{arg}_t) = z$$

$$\sigma \rightsquigarrow^{L}_{\mathsf{call?}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z] \qquad \sigma \rightsquigarrow^{L}_{\mathsf{ret!}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma, \sigma(\mathsf{arg}_t) = z$$

$$\sigma \rightsquigarrow^{L}_{\mathsf{call!}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma, \sigma(\mathsf{arg}_t) = z \qquad \sigma \rightsquigarrow^{L}_{\mathsf{ret?}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z]$$

Figure 6. Transformers for calls and returns to, from and inside a library L.

library L is defined by a family of transformers $\{\rightsquigarrow^{L}_{c,t}\}_{t \in \mathcal{T}}$, where $\rightsquigarrow^{L}_{c,t} \subseteq (\mathsf{Locs}_L \to \mathbb{Z}) \times (\mathsf{Locs}_L \to \mathbb{Z})$ describes how c affects the state of the library. The fact that the transformers are defined on locations from Locs_L formalises our assumption that L accesses only these locations. We assume that the transformers satisfy some standard properties [14], deferred to §A due to space constraints. To define the semantics of expressions, we assume that for each E the set PComm contains a special command $\mathsf{assume}(E)$, used only in defining the semantics, that allows the computation to proceed only if E is non-zero: $\sigma \rightsquigarrow^{L}_{\mathsf{assume}(E),t} \sigma'$ iff $\sigma' = \sigma$ and E is non-zero in σ.

Library Denotations. The set of traces of a library is generated in two stages. First, we generate a superset $(\!|L|\!) \subseteq 2^{\mathsf{Traces}}$ of traces produced by L, defined in Figure 5. If we think of commands as control-flow graphs, these traces contain interleavings of all possible paths through the control-flow graphs of L's methods, invoked in an arbitrary sequence. We then select those traces in $(\!|L|\!)$ that correspond to valid executions starting in a given heap using a predicate $[\![\tau]\!]_L : (\mathsf{Locs}_L \to \mathbb{Z}) \to \{\mathsf{true}, \mathsf{false}\}$. We define $[\![\cdot]\!]_L$ by generalising $\rightsquigarrow$ to calls and returns as shown in Figure 6 and letting

$$[\![\varepsilon]\!]_L\sigma = \mathsf{true}; \quad [\![(t,a)\,\tau]\!]_L\sigma = \text{if } (\exists\sigma'.\, \sigma \rightsquigarrow^{L}_{a,t} \sigma' \wedge [\![\tau]\!]_L\sigma' = \mathsf{true}) \text{ then true else false.}$$

Finally, we let the set of histories $[\![L]\!]$ of a library L consist of those obtained from traces representing its valid executions from a heap with all locations set to 0:

$$[\![L]\!] = \mathsf{history}(\{\tau \in (\!|L|\!) \mid [\![\tau]\!]_L(\lambda x \in \mathsf{Locs}_L.\, 0) = \mathsf{true}\}),$$

where history projects to actions in Act.

THEOREM 1 (Correctness of flat combining) *For the libraries* FC *in Figure 1 and* $\mathsf{FC}^\sharp$ *in Figure 2 and the relation* $\mathcal{R}_\mathsf{t}$ *from §3 we have* $\mathsf{FC} \sqsubseteq_{\mathcal{R}_\mathsf{t}} \mathsf{FC}^\sharp$.

PROOF SKETCH. Consider $h \in [\![\mathsf{FC}]\!]$. In such a history, any invocation of an abstract method $(t, \mathsf{call}!\, m_i(z_i))\,(t, \mathsf{ret}?\, m_i(z_i'))$ happens within the execution of the corresponding wrapper method $(t', \mathsf{call}?\, \mathsf{do\_}m_i(z_i))\,(t', \mathsf{ret}!\, \mathsf{do\_}m_i(z_i'))$ (or just $(t', \mathsf{call}?\, \mathsf{do\_}m_i(z_i))$ if the execution of the method is uncompleted in h), though not necessarily in the same thread. This correspondence is one-to-one, as different invocations of abstract methods correspond to different requests to perform them. Furthermore, abstract methods in h are executed sequentially. We then construct a history h' by replacing every abstract method call $(t, \mathsf{call}!\, m_i(z_i))\,(t, \mathsf{ret}?\, m_i(z_i'))$ in $h|_{\mathsf{AbsAct}}$ by

$$(t', \mathsf{call}?\, \mathsf{do\_}m_i(z_i))\,(t', \mathsf{call}!\, m_i(z_i))\,(t', \mathsf{ret}?\, m_i(z_i'))\,(t', \mathsf{ret}!\, \mathsf{do\_}m_i(z_i')),$$

where t' is the thread identifier of the corresponding wrapper method invocation (similarly for uncompleted invocations). It is easy to see that $\overline{(h|_{\mathsf{AbsAct}})}\ \mathcal{R}_\mathsf{t}\ \overline{(h'|_{\mathsf{AbsAct}})}$ and $h' \in [\![\mathsf{FC}^\sharp]\!]$. Since the execution of an abstract method in h happens within the execution of the corresponding wrapper method, we also have $(h|_{\mathsf{ClAct}}) \sqsubseteq (h'|_{\mathsf{ClAct}})$. □

5 Instantiating Library Parameters and Contextuality

We now define how library parameters are instantiated and show that our notions of linearisability are preserved under such instantiations. To this end, we introduce a partial operation $\circ$ on libraries of §4: informally, for $L_1 : M \to M'$ and $L_2 : M' \to M''$ the library $L_2 \circ L_1 : M \to M''$ is obtained by instantiating abstract methods in L_2 with their implementations from L_1. Note that L_1 can itself have abstract methods M, which are left unimplemented in $L_2 \circ L_1$. Since we assume that different libraries operate in disjoint address spaces, for $\circ$ to be defined we require that the sets of locations of L_1 and L_2 be disjoint, with the exception of those used for method parameter passing. To avoid name clashes, we also require that public non-abstract methods of L_2 not be declared as abstract in L_1 (private non-abstract methods are not an issue, since we identify libraries up to their α-renaming); this also disallows recursion between L_2 and L_1.

DEFINITION 5 (Parameter library instantiation) *Consider* $L_1 : M \to M'$ *and* $L_2 : M' \to M''$ *such that* $(M'' \setminus M') \cap M = \emptyset$ *and* $\mathsf{Locs}_{L_1} \cap \mathsf{Locs}_{L_2} = \{\mathsf{arg}_t\}_{t \in \mathcal{T}}$. *Then* $L_2 \circ L_1 : M \to M''$ *is the library with* $\mathsf{Locs}_{L_2 \circ L_1} = \mathsf{Locs}_{L_1} \cup \mathsf{Locs}_{L_2}$ *obtained by erasing the declarations for methods in* M' *from* L_2, *reclassifying the methods from* $M' \setminus M''$ *in* L_1 *as private, and concatenating the method declarations of the resulting two libraries. We write* $(L_2 \circ L_1)\!\downarrow$ *when* $L_2 \circ L_1$ *is defined.*

We now show that the notions of parameterised linearisability we proposed are ***contextual***, i.e., closed under library instantiations. This property is useful in that it allows us to break the reasoning about a complex library into that about individual libraries from which it is constructed. As we show in §6, contextuality also helps us establish observational refinement.

THEOREM 2 (Contextuality of parameterised linearisability: general case) *For* $L_1, L_2 : M \to M'$ *such that* $L_1 \sqsubseteq L_2$*:*

(i) $\forall L : M'' \to M.\, (L_1 \circ L)\!\downarrow \wedge\, (L_2 \circ L)\!\downarrow \implies L_1 \circ L \sqsubseteq L_2 \circ L.$

(ii) $\forall L : M' \to M''.\, (L \circ L_1)\!\downarrow \wedge\, (L \circ L_2)\!\downarrow \implies L \circ L_1 \sqsubseteq L \circ L_2.$

THEOREM 3 (Contextuality of parameterised linearisability: encapsulated case) *For $L_1, L_2 : M \to M'$ such that $M \cap M' = \emptyset$ and $L_1 \sqsubseteq_{\mathsf{e}} L_2$:*

(i) $\forall L : M'' \to M.\, (L_1 \circ L)\!\downarrow \wedge\, (L_2 \circ L)\!\downarrow \implies L_1 \circ L \sqsubseteq_{\mathsf{e}} L_2 \circ L.$
(ii) $\forall L : M' \to M''.\, (L \circ L_1)\!\downarrow \wedge\, (L \circ L_2)\!\downarrow \implies L \circ L_1 \sqsubseteq_{\mathsf{e}} L \circ L_2.$

The restriction on method names in Definition 5 ensures that the library compositions in Theorem 3 have no public abstract methods and can thus be compared by $\sqsubseteq_{\mathsf{e}}$. Note that if L is ground, then so are $L_1 \circ L$ and $L_2 \circ L$. In this case, Theorems 2(i) and 3(i) allow us to establish classical linearisability from parameterised one.

Stating the contextuality of $\sqsubseteq_{\mathcal{R}}$ is more subtle. The relationship $L_1 \sqsubseteq_{\mathcal{R}} L_2$ allows the use of abstract methods by L_1 and L_2 to differ according to $\mathcal{R}$. As a consequence, for a non-ground parameter library L, their use by $L_1 \circ L$ and $L_2 \circ L$ may also differ according to another relation $\mathcal{G}$. We now introduce a property of L ensuring that a change in L's interactions with its client according to $\mathcal{R}$ (the *rely*) leads to a change in L's interactions with its abstract methods according to $\mathcal{G}$ (the *guarantee*).

DEFINITION 6 (Rely-guarantee closure) *Let $\mathcal{R}, \mathcal{G}$ be relations between histories of type $\emptyset \to M'$ and $\emptyset \to M$, respectively. A library $L : M \to M'$ is $\binom{\mathcal{R}}{\mathcal{G}}$-**closed** if for all $h \in [\![L]\!]$ and $h' : \emptyset \to M'$ we have*

$$(h|_{\mathsf{ClAct}})\ \mathcal{R}\ h' \implies \exists h'' \in [\![L]\!].\, (h''|_{\mathsf{ClAct}} = h') \wedge \overline{(h|_{\mathsf{AbsAct}})}\ \mathcal{G}\ \overline{(h''|_{\mathsf{AbsAct}})}.$$

Due to space constraints, we state contextuality of $\sqsubseteq_{\mathcal{R}}$ only for the case in which library parameters do not have public abstract methods. A more general statement which relaxes this assumption is given in §B.

THEOREM 4 (Contextuality of linearisability up to $\mathcal{R}$) *For $L_1, L_2 : M \to M'$ such that $M \cap M' = \emptyset$ and a relation $\mathcal{R}$ such that $L_1 \sqsubseteq_{\mathcal{R}} L_2$:*

(i) $\forall L : M'' \to M.\, \forall \mathcal{G}.\, M'' \cap M = \emptyset \wedge (L \text{ is } \binom{\mathcal{R}}{\mathcal{G}}\text{-closed}) \wedge$
$(L_1 \circ L)\!\downarrow \wedge\, (L_2 \circ L)\!\downarrow \implies L_1 \circ L \sqsubseteq_{\mathcal{G}} L_2 \circ L.$
(ii) $\forall L : M' \to M''.\, (L \circ L_1)\!\downarrow \wedge\, (L \circ L_2)\!\downarrow \implies L \circ L_1 \sqsubseteq_{\mathcal{R}} L \circ L_2.$

When L in Theorem 4(i) is ground, $\mathcal{G}$ becomes irrelevant. In this case we say that L is $\mathcal{R}$-***closed*** if it is $\binom{\mathcal{R}}{\{(\varepsilon,\varepsilon)\}}$-closed. Hence, from Theorems 1 and 4(i) we get that for any $\mathcal{R}_{\mathsf{t}}$-closed (§3) library L we have $\mathsf{FC} \circ L \sqsubseteq \mathsf{FC}^\sharp \circ L$: instantiating flat combining with a library insensitive to thread identifiers, e.g., a sequential stack or a queue, yields a concurrent library linearisable in the classical sense.

Given two libraries $L_1 : M_1 \to M_1'$ and $L_2 : M_2 \to M_2'$ that do not interact, i.e., $(M_1 \cup M_1') \cap (M_2 \cup M_2') = \emptyset$, we may wish to compose them by merging their method declarations into a library $L_1 \uplus L_2 : M_1 \uplus M_2 \to M_1' \uplus M_2'$, as originally proposed in [7]. Our notions of linearisability are also closed under this composition.

THEOREM 5 (Compositionality of parameterised linearisability) *For $L_1, L_1' : M_1 \to M_1'$ and $L_2, L_2' : M_2 \to M_2'$ such that $(M_1 \cup M_1') \cap (M_2 \cup M_2') = \emptyset$:*

(i) $L_1 \sqsubseteq L_1' \wedge L_2 \sqsubseteq L_2' \implies L_1 \uplus L_2 \sqsubseteq L_1' \uplus L_2'.$
(ii) $L_1 \sqsubseteq_{\mathsf{e}} L_1' \wedge L_2 \sqsubseteq_{\mathsf{e}} L_2' \implies L_1 \uplus L_2 \sqsubseteq_{\mathsf{e}} L_1' \uplus L_2'.$
(iii) $\forall \mathcal{R}, \mathcal{G}.\, L_1 \sqsubseteq_{\mathcal{R}} L_1' \wedge L_2 \sqsubseteq_{\mathcal{G}} L_2' \implies L_1 \uplus L_2 \sqsubseteq_{\mathcal{R} \otimes \mathcal{G}} L_1' \uplus L_2'$, *where $\mathcal{R} \otimes \mathcal{G}$ relates histories if their projections to M_1 actions are related by $\mathcal{R}$ and the projections to M_2 actions are related by $\mathcal{G}$.*

6 Clients and Observational Refinement

A ***program*** P has the form $\mathsf{let}\ L\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n$, where $L : \emptyset \to M$ is a ground library and $C_1 \parallel \ldots \parallel C_n$ is a client such that $C_1, \ldots, C_n$ call only methods in M, written $(C_1 \parallel \ldots \parallel C_n) : M$. Using the contextuality results from §5, we now show that our notions of linearisability imply observational refinement for such programs.

The semantics of a program P is given by the set of its traces $[\![P]\!] \in 2^{\mathsf{Traces}}$, which include actions (t, c) recording the execution of primitive commands c by client threads C_t and the library L, as well as $(t, \mathsf{call}\, m(z))$ and $(t, \mathsf{ret}\, m(z))$ actions corresponding to the former invoking methods of the latter. The semantics $[\![P]\!]$ is defined similarly to that of libraries in §4. In particular, we assume that client threads C_t access only locations in a set $\mathsf{Locs}_{\mathsf{client}}$ such that $\mathsf{Locs}_{\mathsf{client}} \cap \mathsf{Locs}_L = \{\mathsf{arg}_t\}_{t \in \mathcal{T}}$ for any L. Due to space constraints, we defer the definition of $[\![P]\!]$ to §A. We define the ***observable behaviour*** $\mathsf{obs}(\tau)$ of a trace $\tau \in [\![P]\!]$ as its projection to client actions, i.e., those outside method invocations, and lift obs to sets of traces as expected.

DEFINITION 7 (Observational refinement) *For $L_1, L_2 : M \to M'$ we say that L_1 **observationally refines** L_2, written $L_1 \sqsubseteq_{\mathsf{obs}} L_2$, if for any ground library $L : \emptyset \to M$ and client $(C_1 \parallel \ldots \parallel C_n) : M'$ we have*

$$\mathsf{obs}([\![\mathsf{let}\ (L_1 \circ L)\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n]\!]) \subseteq \mathsf{obs}([\![\mathsf{let}\ (L_2 \circ L)\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n]\!]).$$

*For a binary relation $\mathcal{R}$ on histories we say that L_1 **observationally refines** L_2 **up to** $\mathcal{R}$, written $L_1 \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} L_2$, if the above is true under the assumption that L is $\mathcal{R}$-closed.*

Thus, $L_1 \sqsubseteq_{\mathsf{obs}} L_2$ means that L_1 can be replaced by L_2 in any program that uses it while keeping observable behaviours reproducible. This allows us to check a property of a program using L_1 (e.g., the flat combining implementation in Figure 1) by checking this property on a program with L_1 replaced by a possibly simpler L_2 (e.g., the flat combining specification in Figure 2). Using Theorems 2–4, we can show that our notions of linearisability validate observational refinement.

THEOREM 6 (Observational refinement) *For any libraries $L_1, L_2 : M \to M'$:*

(i) $L_1 \sqsubseteq L_2 \implies L_1 \sqsubseteq_{\mathsf{obs}} L_2$.
(ii) $M \cap M' = \emptyset \wedge L_1 \sqsubseteq_{\mathsf{e}} L_2 \implies L_1 \sqsubseteq_{\mathsf{obs}} L_2$.
(iii) $\forall \mathcal{R}.\, M \cap M' = \emptyset \wedge L_1 \sqsubseteq_{\mathcal{R}} L_2 \implies L_1 \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} L_2$.

7 Related Work

Linearisability has recently been extended to handle liveness properties, ownership transfer and weak memory models [3,4,9]. Most of these extensions have exploited the connection between linearisability and observational refinement [1]. The same methodology is adopted in the present work, but for studying two previously unexplored topics: parameterised libraries and the impact that common restrictions on their contexts have on the definition of linearisability. We believe that our results are compatible with the existing ones and can thus be extended to cover liveness and ownership transfer [3,4].

Our work shares techniques with game semantics of concurrent programming languages [11,2] and Jeffrey and Rathke's semantics of concurrent objects [10] (in particular, we use the ? and ! notation from the latter). The proofs of our contextuality theorems rely on the fact that library denotations satisfy certain closure properties related to $\sqsubseteq$, $\sqsubseteq_{\mathsf{e}}$

and $\sqsubseteq_{\mathcal{R}}$, which are similar to those exploited in these prior works. However, there are two important differences. First, prior work has not studied common restrictions on library contexts (such as the encapsulation and closure conditions in Definitions 3 and 4) and the induced stronger notions of refinement between libraries, the two key topics of this paper. Second, prior works have considered all higher-order functions, while our parameterised libraries are limited to second order. Our motivation for constraining the setting in this way is to use a simple semantics and study the key issues involved in linearisability of parameterised libraries without using sophisticated machinery from game semantics, such as justification pointers and views [8], designed for accurately modelling higher-order features. However, it is definitely a promising direction to look for appropriate notions of linearisability for full higher-order concurrent libraries by combining the ideas from this paper with those from game semantics.

Turon et al. proposed CaReSL [13], a logic that allows proving observational refinements between higher-order concurrent programs directly, without going via linearisability. Their work is complimentary to ours: it provides efficient proof techniques, whereas we identify obligations to prove, independent of a particular proof system.

Acknowledgements. We thank Thomas Dinsdale-Young and Ilya Sergey for comments that helped improve the paper. This work was supported by the EU FET project ADVENT.

References

1. Ivana Filipovic, Peter W. O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theor. Comput. Sci.*, 411(51-52), 2010.
2. Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Logic*, 151(2-3), 2008.
3. Alexey Gotsman and Hongseok Yang. Liveness-preserving atomicity abstraction. In *ICALP*, 2011.
4. Alexey Gotsman and Hongseok Yang. Linearizability with ownership transfer. *Logical Methods in Computer Science*, 9, 2013.
5. Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In *SPAA*, 2010.
6. Maurice Herlihy and Eric Koskinen. Transactional boosting: a methodology for highly-concurrent transactional objects. In *PPOPP*, 2008.
7. Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3), 1990.
8. J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2), 2000.
9. Radha Jagadeesan, Gustavo Petri, Corin Pitcher, and James Riely. Quarantining weakness - compositional reasoning under relaxed memory models. In *ESOP*, 2013.
10. Alan Jeffrey and Julian Rathke. A fully abstract may testing semantics for concurrent objects. *Theor. Comput. Sci.*, 338(1-3), 2005.
11. James Laird. A game semantics of idealized CSP. *ENTCS*, 45, 2001.
12. Claudio Russo. The Joins concurrency library. In *PADL*, 2007.
13. Aaron Turon, Derek Dreyer, and Lars Birkedal. Unifying refinement and Hoare-style reasoning in a logic for higher-order concurrency. In *ICFP*, 2013.
14. Hongseok Yang and Peter W. O'Hearn. A semantic basis for local reasoning. In *FoSSaCS*, 2002.

A Additional Semantic Definitions

Semantics of Programs. The semantics of programs is defined similarly to libraries, by first defining a superset of traces $(\!|P|\!) \in 2^{\mathsf{Traces}}$ and then checking each trace $\tau \in (\!|P|\!)$ for feasibility. To define $(\!|P|\!)$, we first define $(\!|C|\!)^{\mathsf{client}}_t \eta \in 2^{\mathsf{Traces}}$ for commands C similarly to $(\!|C|\!)_t \eta$ in Figure 5, but by replacing the definition of $(\!|m()|\!)_t \eta$ by

$$(\!|m()|\!)^{\mathsf{client}}_t \eta = \{(t, \mathsf{call}\, m(z))\, \tau\, (t, \mathsf{ret}\, m(z')) \;\mid\; \tau \in \eta(m,t) \wedge z, z' \in \mathbb{Z}\}.$$

Then for $L = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}} \rangle$ we let

$$(\!|\mathsf{let}\ L\ \mathsf{in}\ C_1 \parallel \cdots \parallel C_k|\!) = \mathsf{prefix}\left(\|^k_{t=1} (\!|C_t|\!)^{\mathsf{client}}_t (\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!) \right).$$

For a trace $\tau \in (\!|P|\!)$, let $\mathsf{client}(\tau)$ and $\mathsf{lib}(\tau)$ be the projection of τ to client and library actions respectively, i.e., those outside or inside method invocations (the corresponding calls and returns included in both cases). We lift client and lib to sets of traces as expected.

We write $\sigma \rightsquigarrow^{\mathsf{client}}_{c,t} \sigma'$ if $\sigma \rightsquigarrow_{c,t} \sigma'$ and $\mathsf{dom}(\sigma), \mathsf{dom}(\sigma') \subseteq \mathsf{Locs}_{\mathsf{client}}$. We also let

$$\begin{array}{l} \sigma \rightsquigarrow^{\mathsf{client}}_{\mathsf{call}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma, \sigma(\mathsf{arg}_t) = z; \\ \sigma \rightsquigarrow^{\mathsf{client}}_{\mathsf{ret}\ m(z),t} \sigma' \text{ iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z]; \end{array}$$

Then for $\tau \in \mathsf{client}([\![P]\!])$ we define $[\![\tau]\!]_{\mathsf{client}} : \mathcal{H} \to \{\mathsf{true}, \mathsf{false}\}$ as follows:

$$\begin{array}{l} [\![\varepsilon]\!]_{\mathsf{client}} \sigma = \mathsf{true}; \\ [\![(t,a)\,\tau]\!]_{\mathsf{client}} \sigma = \text{if } (\exists \sigma'.\, \sigma \rightsquigarrow^{\mathsf{client}}_{a,t} \sigma' \wedge [\![\tau]\!]_{\mathsf{client}} \sigma' = \mathsf{true}) \text{ then true else false}. \end{array}$$

Let $\mathsf{transform}(\tau)$ be the trace τ with call replaced by call? and ret replaced by ret!.

Finally, for

$$P \;=\; \mathsf{let}\ L\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n$$

we let

$$\begin{array}{rl} [\![P]\!] = \{\tau \in (\!|P|\!) \mid & [\![\mathsf{client}(\tau)]\!]_{\mathsf{client}}(\lambda x \in \mathsf{Locs}_{\mathsf{client}}.\, 0) = \mathsf{true} \wedge \\ & [\![\mathsf{transform}(\mathsf{lib}(\tau))]\!]_L(\lambda x \in \mathsf{Locs}_L.\, 0) = \mathsf{true}\}. \end{array}$$

Assumptions on Transformers. We state the properties we require of the transformers $\rightsquigarrow^L_{c,t}$ and $\rightsquigarrow^{\mathsf{client}}_{c,t}$. Let $\sigma, \sigma' \in \mathsf{Locs} \rightharpoonup \mathbb{Z}$ be such that $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\sigma') = \{\mathsf{arg}_t\}_{t \in \mathcal{T}}$, and $\forall t \in \mathcal{T}.\, \sigma(\mathsf{arg}_t) = \sigma'(\mathsf{arg}_t)$. In this case we say that $\sigma \cdot \sigma'$ is defined, denoted $(\sigma_1 \cdot \sigma_2)\downarrow$, and let $(\sigma \cdot \sigma')(x) = \sigma(x)$ if $x \in \mathsf{dom}(\sigma)$, and $(\sigma \cdot \sigma')(x) = \sigma'(x)$ otherwise. We require that any transformer $\rightsquigarrow^L_{c,t}$ satisfy the following properties:

–

$$\begin{array}{l} \forall \sigma, \sigma', t, t', z.\, t' \neq t \wedge \sigma \rightsquigarrow^L_{c,t} \sigma' \implies \\ \qquad \sigma'(\mathsf{arg}_{t'}) = \sigma(\mathsf{arg}_{t'}) \wedge \sigma[\mathsf{arg}_{t'} \mapsto z] \rightsquigarrow^L_{c,t} \sigma'[\mathsf{arg}_{t'} \mapsto z]. \end{array}$$

– whenever $(L_2 \circ L_1)\downarrow$, then

$$\begin{array}{l} \sigma_2 \rightsquigarrow^{L_2}_{c,t} \sigma_2' \implies \\ \qquad (\forall \sigma_1.\mathsf{dom}(\sigma_1) = \mathsf{Locs}_{L_1} \wedge (\sigma_2 \cdot \sigma_1)\downarrow \implies \\ \qquad\qquad (\sigma_2 \cdot \sigma_1) \rightsquigarrow^{(L_2 \circ L_1)}_{c,t} (\sigma_2' \cdot \sigma_1[\mathsf{arg}_t \mapsto \sigma_2'(\mathsf{arg}_t)])) \end{array}$$

$$\sigma_1 \rightsquigarrow_{c,t}^{L_1} \sigma_1' \implies (\forall \sigma_2.\mathsf{dom}(\sigma_2) = \mathsf{Locs}_{L_2} \wedge (\sigma_2 \cdot \sigma_1)\downarrow \implies (\sigma_2 \cdot \sigma_1) \rightsquigarrow_{c,t}^{(L_2 \circ L_1)} (\sigma_2[\mathsf{arg}_t \mapsto \sigma_1'(\mathsf{arg}_t)] \cdot \sigma_1'))$$

– whenever $\sigma \rightsquigarrow_{c,t}^{L_2 \circ L_1} \sigma'$, then

$$\exists.\sigma_2, \sigma_2', \sigma_2', \sigma_1'.\, (\mathsf{dom}(\sigma_2), \mathsf{dom}(\sigma_2') = \mathsf{Locs}_{L_2} \wedge \mathsf{dom}(\sigma_1), \mathsf{dom}(\sigma_1') = \mathsf{Locs}_{L_1} \wedge (\sigma_2 \cdot \sigma_1) = \sigma, (\sigma_2' \cdot \sigma_1') = \sigma')) \wedge \Big((\sigma_2 \rightsquigarrow_{c,t}^{L_2} \sigma_2' \wedge \sigma_1' = \sigma_1[\mathsf{arg}_t \mapsto \sigma_2'(\mathsf{arg}_t)]) \vee (\sigma_1 \rightsquigarrow_{c,t}^{L_1} \sigma_1' \wedge \sigma_2' = \sigma_2[\mathsf{arg}_t \mapsto \sigma_2'(\mathsf{arg}_t)])\Big)$$

We also require that the last two properties hold if the operator $\circ$ is replaced by $\uplus$.

B Proofs

In this section we will state the lemmas needed to prove the results stated in the paper, then giving an outline of the proofs for such results. Proofs of the presented lemmas will be given in separate appendices.

When it will be more convenient, we will use different definitions of linearisability, which can be esasily proven to be equivalent to those presented in the text. For general linearisability, the alternative definition we will use is the following:

DEFINITION 8 *Given two histories h_1, h_2, let $h_1 \simeq h_2 \iff \forall t \in \mathcal{T}.\, h_1|_t = h_2|_t$. We say that $h_1 \sqsubseteq h_2$ if and only if*

(i) $h_1 \simeq h_2$, and
(ii) there exists a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that

$$\forall i.\, h_1(i) = h_2(\pi(i)) \wedge (\forall j.\, i < j \wedge ((h_1(i) \in \mathsf{Act!} \wedge h_1(j) \in \mathsf{Act?})) \implies \pi(i) < \pi(j)).$$

The definition for the encapsulated case is similar.

Often it will be convenient to consider a relation $\mathcal{R}$ between histories as the dual of some other relation $\mathcal{R}'$ between sequences of actions in Act, provided that the image of the elements appearing in $\mathcal{R}'$ under the operation $\bar{\cdot}$ is a subset of Hist. That is, we let $\mathcal{R} = \overline{\mathcal{R}'}$, where $\overline{\mathcal{R}'}$ is the smallest relation such that $h_1 \; \mathcal{R} \; h_2$ implies $\overline{h_1} \; \overline{\mathcal{R}'} \; \overline{h_2}$.

Strong Linearisability Henceforth we will work with a stronger version of linearisability. This takes into account call and returns of public abstract methods which can be performed by clients, which are treated as both **!**-actions and **?**-actions. However we show that, in our language, strong linearisability and parameterised linearisability coincide.

First, we extend the set of actions TrAct with actions from the set $\{(t, \iota), (t, \mathsf{ExtCall}\, m(z)), (t, \mathsf{ExtRet}\, m(z)) \mid m \in \mathcal{M}, t \in \mathcal{T}, z \in \mathbb{Z}\}$, and we denote the resulting set as TrAct^+. The set of traces generated by actions in such a set is denoted as Traces^+. Given a library $L : M \to M'$, we will use actions of the form (t, ι) to denote thread t executing some primitive command outside library L (for example, in the client or in the library parameter), $t, \mathsf{ExtCall}\, m(z)$ to denote the client calling the method $m \in M \cap M'$ in thread t, and $t, \mathsf{ExtRet}\, m(z)$ to denote a method $m \in M \cap M'$,

previously called by the client in thread t, returning value z. The set of external calls and returns is denoted as ExtAct. We let $\mathsf{Act!}^+ = \mathsf{Act!} \cup \mathsf{ExtAct}$, $\mathsf{Act?}^+ = \mathsf{Act?} \cup \mathsf{ExtAct}$.

For $L = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}} \rangle$, we let

$$(\!|L|\!)^+ = \mathsf{prefix}\left(\bigcup_{k>0} \Big\|_{t=1}^{k} \left((t,\iota) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M' \setminus M}} (t, \mathsf{call?}\, m(z))\, (\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)^+(m,t)\, (t, \mathsf{ret!}\, m(z')) \right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M}} (t, \mathsf{ExtCall}\, m(z))(t,\iota)^*(t, \mathsf{ExtRet}\, m(z')) \right) \right)^* \right)$$

where $(\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)^+(m,t)$ is defined as the least fixpoint of a functional $\mathcal{F}_L^+$, defined by replacing the clause $(\mathcal{F}_L\eta)(m,t) = \{\varepsilon\}$ with $(\mathcal{F}_L^+\eta)(m,t) = \{(t,\iota)\}^*$ in Figure 5; also, the clause $(\mathcal{F}_L\eta)(m,t) = \emptyset$ for those methods not appearing in the library is replaced with $(\mathcal{F}_L^+\eta)(m,t) = \{(t,\iota)\}^*$.

The interpretation $[\![\cdot]\!]_L^+$ of traces in Traces^+ is defined as in §4; however, we have to define the transformers for the newly introduced actions. We let

$$\sigma \rightsquigarrow^L_{\mathsf{ExtCall}\, m(z),t} \sigma' \text{ iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z] \qquad \sigma \rightsquigarrow^L_{\mathsf{ExtRet}\, m(z),t} \sigma' \text{ iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z]$$
$$\sigma \rightsquigarrow^L_{\iota,t} \sigma' \qquad \text{iff } \sigma' = \sigma[\mathsf{arg}_t \mapsto z]$$

It is important to note that transformers of ι-actions allow the argument value of the thread executing the command to be changed. This is because, as we said, ι-actions correspond to commands executed by some external entity, and we have to reflect the fact that the location arg_t could be affected by the execution of such a command.

Finally, the set of ***Augmented Traces*** $T(L)^+$ for a library L is defined as

$$\left(\{\tau \in (\!|L|\!)^+(\lambda l \in \mathsf{Locs}_L.0) \mid [\![\tau]\!]^+ = \mathsf{true}\}\right)$$

while the set of its ***Augmented Histories*** is defined as

$$[\![L]\!]^+ = \mathsf{history}^+(T(L))$$

where $\mathsf{history}^+(\tau)$ denotes the projection of τ to elements in $\mathsf{Act}^+ = \mathsf{Act} \cup \mathsf{ExtAct}$.

LEMMA 1 $\forall L : M \to M'$:

(i) $[\![L]\!] \subseteq [\![L]\!]^+$,
(ii) $[\![L]\!]^+|_{\mathsf{Act}} \subseteq [\![L]\!]$.

LEMMA 2 *Let* $L : M \to M'$, $h \in [\![L]\!]^+$*:*

(i) if $h = (h_1)(h_2)$, $h_1|_t$ *is complete and* $h_2|_t = \varepsilon$, *then* $(h_1)(t, \mathsf{ExtCall}\, m(z))(h_2) \in [\![L]\!]^+$ *for any* $m \in M \cap M'$, $z \in \mathbb{Z}$,
(ii) if $h = (h_1)(h_2)(h_3)$, $h_1|_t$ *is complete,* $h_2|_t = \varepsilon$, *then* $(h_1)(t, \mathsf{ExtCall}\, m(z))(h_2)(t, \mathsf{ExtRet}\, m(z'))(h_3) \in [\![L]\!]^+$ *for any* $m \in M \cap M'$, *and* $z, z' \in \mathbb{Z}$.

We are now ready to state our notion of ***strong linearisability***.

DEFINITION 9 *Let h_1, h_2 be two augmented histories. We say that $h_1 \sqsubseteq^+ h_2$ if $h_1|_t = h_2|_t$ for any $t \in \mathcal{T}$, and there exists a permutation π such that $(\forall i.h_1(i)) = h_2(\pi(i)) \wedge (\forall j \leq k(h(j) \in \mathsf{Act!}^+ \wedge h(k) \in \mathsf{Act?}^+) \implies \pi(k) < \pi(k))$.*

Strong linearisability is lifted to libraries in the usual way. $L_1 \sqsubseteq^+ L_2$ if $\forall h_1 \in [\![L_1]\!]^+.\exists h_2 \in [\![L_2]\!]^+.h_1 \sqsubseteq^+ h_2$.

Strong linearisability takes into account external call and returns. However, in a library L, external calls and returns happen independently from other activities, and can be rearranged arbitrarily, as long as the thread-local behaviour of libraries is preserved. Hence, they do not add any discriminating power to linearisability.

THEOREM 7 *For any two libraries $L_1, L_2 : M \to M'$, $L_1 \sqsubseteq^+ L_2$ if and only if $L_1 \sqsubseteq L_2$.*

Composition and Decomposition of Augmented Traces Let us fix two libraries $L_1 : M \to M'$ and $L_2 : M' \to M''$ such that $(L_2 \circ L_1)\downarrow$. Our aim is to decompose an augmented trace $\tau \in T(L_2 \circ L_1)^+$ in two traces $\tau\Uparrow \in T(L_2)^+$ and $\tau\Downarrow \in T(L_1)^+$; these two traces reflect the behaviour described by τ from the point of view of L_2, L_1, respectively. Also, given two traces $\tau'' \in T(L_2)^+$, $\tau' \in T(L_1)^+$ which describe the behaviour of a trace $\tau \in T(L_2 \circ L_1)^+$, from the point of view of L_2 and L_1, respectively, we want the latter trace to be determined uniquely from τ'', τ'. In other words, we want to define an operator $\circ$ such that $\tau'' \circ \tau' = \tau$.

To accomplish this task, in Figure 7 we define a ***Constraint Automaton*** $\mathcal{A}$. This automaton uses two variables nc_2, nc_1, which store the number of pending call invocations of methods implemented in the libraries L_2 and L_1, respectively. The set of its states is $\mathsf{States} = \{\mathsf{Client}, L_2, L_1, \mathsf{Param}\}$; a configuration (s, v_2, v_1) for $\mathcal{A}$ consists of a state s and two values v_2, v_1 for the two variables nc_2, nc_1. The actions performed by $\mathcal{A}$ are triples of the form (α, β, γ), which describe how actions of a single threaded program are seen from the point of view of L_2, L_1 and $L_2 \circ L_1$, repsectively.

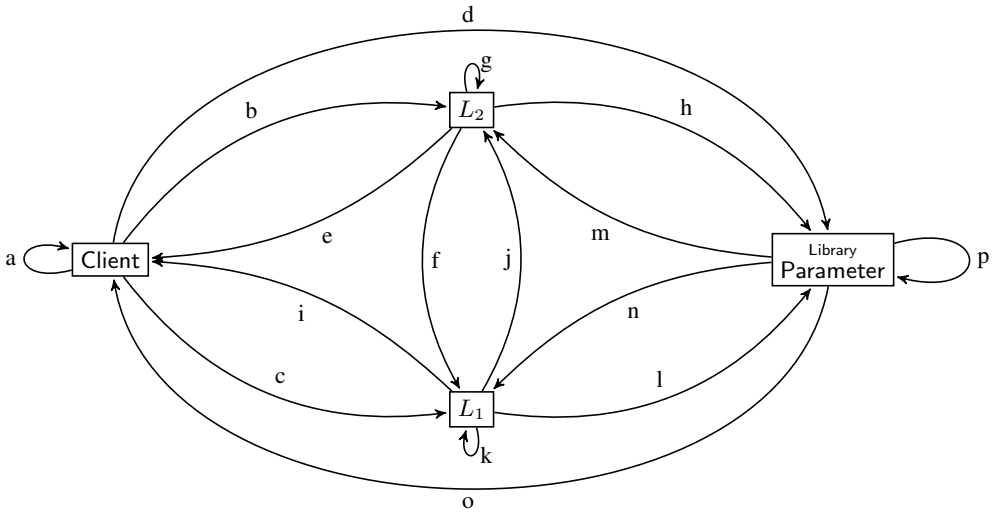
Whenever there exists a path $(\alpha_1, \beta_1, \gamma_1) \cdots (\alpha_n, \beta_n, \gamma_n)$ which can be produced in $\mathcal{A}$, starting from the configuration $(\mathsf{Client}, 0, 0)$, then let $\psi'' = \alpha_1 \cdot \alpha_n$, $\psi' = \beta_1 \cdot \beta_n$ and $\psi = \gamma_1 \cdot \gamma_n$. We define $\psi\Uparrow = \psi''$, $\psi\Downarrow = \psi'$ and $\psi'' \circ \psi' = \psi$. Note that these operations take into account unobservable activities, i.e. ι-actions. Later we will see how we can reason on composition properties of traces that abstract from such actions.

LEMMA 3 *The (partial) operators $(\cdot)\Uparrow, (\cdot)\Downarrow, (\cdot \circ \cdot)$ are well-defined. Also, for any string of actions ψ such that a path $(\cdot, \cdot, \psi)$ can be produced from $(\mathsf{Client}, 0, 0)$ in $\mathcal{A}$, then $\psi\Uparrow \circ \psi\Downarrow = \psi$.*

The operators $(\cdot)\Uparrow, (\cdot)\Downarrow$ and $(\cdot \circ \cdot)$ can be easily extended to augmented traces in Traces^+.

DEFINITION 10 *Let $\tau \in \mathsf{Traces}^+$ be such that $\tau|_t\Uparrow$ ($\tau|_t\Downarrow$) is defined for any $t \in \mathcal{T}$. Then $\tau\Uparrow$ ($\tau\Downarrow$) is defined as the unique trace such that*

(i) $(\tau\Uparrow)|_t = (\tau|_t)\Uparrow$ ($(\tau\Downarrow)|_t = (\tau|_t)\Downarrow$),
(ii) for any $t \in \mathcal{T}$, and for any $i = 1, \cdots, |\tau|$, $\tau(i) = (t, \cdot) \implies (\tau\Uparrow)(i) = (t, -)$ ($(\tau\Downarrow)(i) = (t, -)$).



Label	Action in L_2	Action in L_1	Action in $L_2 \circ L_1$	Constraint	Call counters
a	ι	ι	ι	$\cdot$	$\cdot$
b	call? $m(z)$	ι	call? $m(z)$	$m \in (M'' \setminus M')$	$nc_2 := 1; nc_2 := 0$
c	ExtCall $m(z)$	call? $m(z)$	call? $m(z)$	$m \in (M'' \cap M') \setminus M$	$nc_2 := 0; nc_1 := 1$
d	ExtCall $m(z)$	ExtCall $m(z)$	ExtCall $m(z)$	$m \in M'' \cap M' \cap M$	$nc_2 := 0; nc_1 := 0$
e	ret! $m(z)$	ι	ret! $m(z)$	$m \in (M'' \setminus M') \wedge$ $\wedge nc_2 = 1$	$nc_2 := 0$
f	call! $m(z)$	call? $m(z)$	call $m(z)$	$m \in M' \setminus M$	$nc_2 + +; nc_1 := 1$
g_1	c	ι	c	$\cdot$	$m \notin M'$
g_2	call $m(z)$	ι	call $m(z)$	$m \notin M'$	$nc_2 + +$
g_3	ret $m(z)$	ι	ret $m(z)$	$(nc_2 > 1)$	$nc_2 - -$
h	call! $m(z)$	ExtCall $m(z)$	call! $m(z)$	$m \in M' \cap M$	$nc_2 + +$
i	ExtRet $m(z)$	ret! $m(z)$	ret! $m(z)$	$m \in (M'' \cap M') \setminus M \wedge$ $\wedge nc_2 = 0 \wedge nc_1 = 1$	$nc_1 := 0$
j	ret? $m(z)$	ret! $m(z)$	ret $m(z)$	$m \in M' \setminus M \wedge$ $\wedge nc_2 > 1 \wedge nc_1 = 1$	$nc_2 - -; nc_1 := 0$
k_1	ι	c	c	$\cdot$	$\cdot$
k_2	ι	call $m(z)$	call $m(z)$	$m \notin M$	$nc_1 + +$
k_3	ι	ret $m(z)$	ret $m(z)$	$nc_1 > 1$	$nc_1 - -$
l	ι	call! $m(z)$	call! $m(z)$	$m \in M$	$nc_1 + +;$
m	ret? $m(z)$	ExtRet $m(z)$	ret? $m(z)$	$m \in M' \cap M \wedge$ $\wedge nc_2 > 1 \wedge nc_1 = 0$	$nc_2 - -$
n	ι	ret? $m(z)$	ret? $m(z)$	$m \in M \wedge$ $\wedge nc_1 > 1$	$nc_1 - -$
o	ExtRet $m(z)$	ExtRet $m(z)$	ExtRet $m(z)$	$m \in M'' \cap M' \cap M \wedge$ $\wedge nc_2 = 0 \wedge nc_1 = 0$	$\cdot$
p	ι	ι	ι	$\cdot$	$\cdot$

Figure 7. A constraint automaton representing how the entities executing code change.

Let $\tau_2, \tau_1 \in \mathsf{Traces}^+$ be such that $|\tau_2| = |\tau_1|$, $(\tau_2|_t) \circ (\tau_1|_t)$ is defined for any $t \in \mathcal{T}$, and $\forall i = 1, \cdots, |\tau_2|.\tau_2(i) = (t, \cdot) \implies \tau_1(i) = (t, -)$. Then $\tau_2 \circ \tau_1$ is defined as the unique trace τ such that $\tau|_t = (\tau_2|_t) \circ (\tau_1|_t)$, and $\forall i = 1, \cdots, |\tau_2|.\tau_2(i) = (t, \cdot) \implies \tau(i) = (t, -)$.

LEMMA 4 *The operators $(\cdot)\Uparrow$, $(\cdot)\Downarrow$ and $(\cdot \circ \cdot)$ are well defined for traces in Traces^+. Further, if $\tau\Uparrow$ and $\tau\Downarrow$ are defined, we obtain that $(\tau\Uparrow \circ \tau\Downarrow) = \tau$.*

The decomposition operators $\Uparrow$ and $\Downarrow$ are correct, in the sense that they project a trace $\tau \in T(L_2 \circ L_1)^+$ into traces of L_2, L_1, respectively.

LEMMA 5 *For any $\tau \in T(L_2 \circ L_1)^+$, $\tau\Uparrow \in T(L_2)^+$ and $\tau\Downarrow \in T(L_1)^+$.*

For the operator $\circ$, the situation is slightly more complicated. In fact, this operator is defined only for those traces τ_2, τ_1 which describe a possible behaviour of $L_2 \circ L_1$ as seen from the point of view of L_2 and L_1. This definition, however, does not abstract from unobservable activities or, in general, from actions which are not recorded into augmented histories. However, for our purposes it is necessary to compose two traces τ_2, τ_1 whenever their projections to augmented histories agree with two other traces τ_2', τ_1' for which $\tau_2' \circ \tau_1'$ is defined. Further, the composite traces also should agree on their projection to augmented histories.

DEFINITION 11 *Let τ, τ' be two augmented traces in Traces^+. We say that $\tau \approx \tau'$ if $\mathsf{history}^+(\tau) = \mathsf{history}^+(\tau')$.*

LEMMA 6 *Let $\tau_2 \in T(L_2)^+, \tau_1 \in T(L_1)^+$. If $\tau = \tau_2 \circ \tau_1$ is defined, then $\tau \in T(L_2 \circ L_1)^+$.*

Also, let $L_2' : M' \to M''$ be such that $(L_2' \circ L_1)\downarrow$. For any $\tau_2' \in T(L_2')^+.\tau_2' \approx \tau_2$ there exists $\tau_2'' \in T(L_2')^+$ and $\tau_1' \in T(L_1)$ such that $(\tau_2'' \circ \tau_1')$ is defined, and $(\tau_2'' \circ \tau_1') \approx \tau$.

Contextuality of Linearisability The main property that we need to prove, to show that parameterised linearisability is contextual, is that libraries are pre-linearisation closed. Formally, this can be stated as follows:

LEMMA 7 *Let $L : M \to M'$, and suppose that $h \in [\![L]\!]^+$. Then $\forall h' : h' \sqsubseteq^+ h.h' \in [\![L]\!]^+$.*

Proof of Theorem 2: We actually prove contextuality of $\sqsubseteq^+$; then the result follows from Theorem 7. We only give the details for (i); the proof of (ii) can be performed in a similar style.

Let $h \in [\![L_2 \circ L]\!]^+$. By definition there exists a trace $\tau \in T(L_2 \circ L)^+$ such that $\mathsf{history}^+(\tau) = h$. By Lemma 5 we have that $\tau\Uparrow \in T(L_2)^+$, and $\tau\Downarrow \in T(L)^+$.

Let $h_2 = \mathsf{history}^+(\tau \Uparrow)$. Since $L_2 \sqsubseteq^+ L_1$, we may find a history $h_1 \in [\![L_1]\!]^+$ such that $h_2 \sqsubseteq^+ h_1$.

By Lemma 7 we also have that $h_2 \in [\![L_1]\!]^+$. That is, there exists a trace $\tau'' \in T(L_1)^+$ such that $\mathsf{history}^+(\tau'') = h_2 = \mathsf{history}^+(\tau\Uparrow)$. Therefore $\tau'' \approx \tau\Uparrow$.

By Lemma 6 we know that there exist $\tau' \approx \tau\Downarrow$ and $\tau''' \approx \tau'' \approx \tau\Uparrow$ such that $\tau''' \in T(L_1)^+$, $\tau' \in T(L)^+$ and $\tau''' \circ \tau' \approx \tau$; that is, $\mathsf{history}^+(\tau''' \circ \tau) = h$. Now note that $(\tau''' \circ \tau') \in T(L_1 \circ L)^+$, hence $h \in [\![L_1 \circ L]\!]^+$. Now it is trivial to note that $h \sqsubseteq^+ h$. $\square$

For encapsulated linearisability and up-to linearisability, we assume that libraries have type $M \to M'$, where $M' \cap M = \emptyset$. One implication of this assumption is that, even in the augmented semantics of libraries, there are no actions of the form $(t, \mathsf{ExtCall}\, m(z))$, or $(t, \mathsf{ExtRet}\, m(z))$ in the sets $[\![L_2]\!]^+$ and $[\![L_2 \circ L_1]\!]^+$ (note that such actions can still appear in $[\![L_1]\!]^+$); in general, for libraries of type $L : M \to M'$, with $M \cap M'$, we have that $[\![L]\!] = [\![L]\!]^+$.

We first prove contextuality of up-to linearisability, then we show that contextuality of encapsulated linearisability can be obtained as a special case of this result. The proof differs in style from that of the previous theorems, as this time it will be necessary to rearrange the abstract actions of a library, according to a relation $\mathcal{R}$. The validity of the theorem relies strongly on the assumption that the client interactions of the parameter library can also be rearranged according to the relation $\bar{\mathcal{R}}$, and that invocations of public, abstract methods, do not depend on the behaviour of implemented methods in a library.

DEFINITION 12 *Let, H_1, H_2 be two sets of histories. We define $H_1 \bowtie H_2$ as the largest set such that, whenever $h \in (H_1 \bowtie H_2)$ then*

1. *h is valid, meaning that for any $t \in \mathcal{T}$, $h|_t$ can be generated by the grammar* SHist *of Definition 1,*
2. *there exists two histories $h_1 \in H_1, h_2 \in H_2$ and two strictly monotone functions $f_1 : \{1, \cdots, |h_1|\} \to \{1, \cdots, |h|\}$, $f_2 : \{1, \cdots, |h_2|\} \to \{2, \cdots, |h|\}$, with $\mathsf{img}(f_1) \cap \mathsf{img}(f_2) = \emptyset$, $\mathsf{img}(f_1) \cup \mathsf{img}(f_2) = \{1, \cdots, |h|\}$, such that for any $i = 1, \cdots, |h_1|$, $h_1(i) = h(f_1(i))$ and for any $i = 1, \cdots, |h_2|$, $h_2(i) = h(f_2(i))$.*

Definition 12 states that histories in $H_1 \bowtie H_2$ can be partitioned in two sub-histories, one belonging to H_1, the other belonging to H_2.

DEFINITION 13 *Given two relations $\mathcal{R}_1$, $\mathcal{R}_2$ between histories, we define $\mathcal{R}_1 \oplus \mathcal{R}_2$ to be the largest relation such that, whenever $h \in (\mathcal{R}_1 \oplus \mathcal{R}_2)\ h'$, then there exist h_1, h_1', h_2, h_2' such that $h \in \{h_1\} \bowtie \{h_2\}$, $h_1\ \mathcal{R}_1\ h_1'$, $h_2\ \mathcal{R}_2\ h_2'$ and $h' \in \{h_1'\} \bowtie \{h_2'\}$.*

Next, we define how in a library of the form $L_2 \circ L_1$, the interactions of L_2 with its library parameter have to agree with the interactions of L_1 with its client library.

DEFINITION 14 *For any action $\phi \in \mathsf{Act}^+$, we define*

1. *$\phi^\uparrow$ as the function such that $\phi^\uparrow = \phi$ if $\phi \in \mathsf{Act}$, $(t, \mathsf{ExtCall}\, m(z))^\uparrow = (t, \mathsf{call?}\, m(z))$ and $(t, \mathsf{ExtRet}\, m(z))^\uparrow = (t, \mathsf{ret!}\, m(z))$.*
2. *$\phi^\downarrow$ as the function such that $\phi^\downarrow = \phi$ if $\phi \in \mathsf{Act}$, $(t, \mathsf{ExtCall}\, m(z))^\downarrow = (t, \mathsf{call!}\, m(z))$ and $(t, \mathsf{ExtRet}\, m(z))^\downarrow = (t, \mathsf{ret?}\, m(z))$.*

The operations $h^\uparrow, h^\downarrow$ for histories are defined by lifting the respective operations for actions.

LEMMA 8 *Let $L_1 : M \to M'$, $L_2 : M' \to M''$ be such that $M' \cap M'' = \emptyset$, and $(L_2 \circ L_1)\downarrow$. For any $\tau_2 \in T(L_2)^+$, $\tau_1 \in T(L_1)^+$ such that $\tau_2 \circ \tau_1$ is defined, then $\overline{(\tau_2)|_{\mathsf{AbsAct}}} = (\tau_1|_{\mathsf{ClAct}^+})^\uparrow$.*

Also, suppose that $(\tau_2) \in T(L_2)^+, (\tau_1) \in T(L_1)^+$ are such that $\overline{(\tau_2)|_{\mathsf{AbsAct}}} = (\tau_1|_{\mathsf{ClAct}^+})^\uparrow$. Then there exist $\tau_2' \in T(L_2)^+, \tau_2' \approx \tau_2$, $\tau_1' \in T(L_1)^+, \tau_1' \approx \tau_1$ such that $\tau_2' \circ \tau_1'$ is defined, $(\tau_2' \circ \tau_1')|_{\mathsf{ClAct}} = \tau_2'|_{\mathsf{ClAct}}$ and $(\tau_2' \circ \tau_1')|_{\mathsf{AbsAct}} = (\tau_1'|_{\mathsf{AbsAct}^+})^\downarrow$.

Below we prove a contextual property of up-to linearisability, which is more general than the one stated in Theorem 4. Specifically, we relax the constraint that, in a composite library, the inner component requires to have no abstract methods.

THEOREM 8 *Let $L_1, L_2 : M \to M'$ be two libraries with $M \cap M' = \emptyset$. Let also $L : M'' \to M$ be such that $(L_1 \circ L)\downarrow$, $(L_2 \circ L)\downarrow$. Let $\mathsf{Act}_{M \to M'}$ be the set of actions which can appear in histories of type $M \to M'$. Finally, let $\mathcal{R}^+, \mathcal{R}^-$ be the smallest relations such that $h\ \mathcal{R}\ h'$ implies $(h|_{\mathsf{Act}_{\emptyset \to (M \setminus M'')}})\ \mathcal{R}^+\ (h'|_{\mathsf{Act}_{\emptyset \to (M \setminus M'')}})$, and $(h|_{\mathsf{Act}_{\emptyset \to (M \cap M'')}})\ \mathcal{R}^-\ (h|_{\mathsf{Act}_{\emptyset \to (M \cap M'')}})$.*

Then, for any library library L which is $\binom{\mathcal{R}^+}{\mathcal{G}}-$closed, according to some relation $\mathcal{G}$, then $L_1 \sqsubseteq_{\mathcal{R}} L_2$ implies that $(L_1 \circ L) \sqsubseteq_{(\mathcal{G} \oplus \mathcal{R}^-)} (L_2 \circ L)$.

Proof. Note that the sets $\mathsf{Act}_{\emptyset \to (M \setminus M'')}$ and $\mathsf{Act}_{\emptyset \to (M \cap M'')}$ are disjoint, and they form a partition of $\mathsf{Act}_{\emptyset \to M}$; as a consequence, whenever $h \in \mathcal{R}$ then $h \in \mathcal{R}^+ \oplus \mathcal{R}^-$.

It is also trivial, from the definition of up-to linearisability, that whenever $\mathcal{R} \subseteq \mathcal{S}$, then $L_1 \sqsubseteq_{\mathcal{R}} L_2$ implies $L_1 \sqsubseteq_{\mathcal{S}} L_2$. In particular, we have that $L_1 \sqsubseteq_{(\mathcal{R}^+ \oplus \mathcal{R}^-)} L_2$.

Let now $h \in [\![L_1 \circ L]\!]^+$; since $M \cap M' = \emptyset$, and $(L_1 \circ L)\downarrow$, we also have $M'' \cap M' = \emptyset$, which means that $h \in [\![L_1 \circ L]\!]$ (that is, h does not contain external calls or returns).

By definition, there exists $\tau \in T(L_1 \circ L)^+$ such that $\tau|_{\mathsf{Act}^+} = h$. By Lemma 5, we can split τ in $\tau\Uparrow \in T(L_1)^+$ and $\tau \Downarrow \in T(L_2)^+$, such that $(\tau\Uparrow) \circ (\tau\Downarrow) = \tau$.

By Lemma 8, we also know that $\overline{\tau\Uparrow|_{\mathsf{AbsAct}}} = (\tau\Downarrow|_{\mathsf{ClAct}^+})^\uparrow$. Since $L_1 \sqsubseteq_{(\mathcal{R}^+ \oplus \mathcal{R}^-)} L_2$, we know that there exists a trace $\tau_2 \in T(L_2)^+$ such that $(\tau\Uparrow)|_{\mathsf{ClAct}} \sqsubseteq (\tau_2)|_{\mathsf{ClAct}}$, and $\overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}}\ (\mathcal{R}^+ \oplus \mathcal{R}^-)\ \overline{(\tau_2)|_{\mathsf{AbsAct}}}$.

We let $h_1^+ = \overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}}|_{\mathsf{Act}_{\emptyset \to M \setminus M''}}$, $h_1^- = \overline{\overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}}|_{\mathsf{Act}_{\emptyset \to (M \cap M'')}}}$; similarly, let $h_2^+ = \overline{\overline{(\tau_2)|_{\mathsf{AbsAct}}}|_{\mathsf{Act}_{\emptyset \to (M \setminus M'')}}}$ and $h_2^- = \overline{\overline{(\tau_2)|_{\mathsf{AbsAct}}}|_{\mathsf{Act}_{\emptyset \to (M \cap M'')}}}$. We have that $h_1^+\ \mathcal{R}^+\ h_2^+$, and $h_1^-\ \mathcal{R}^-\ h_2^-$.

Recall now that, for $\tau\Downarrow$, we have that $(\tau\Downarrow|_{\mathsf{ClAct}^+})^\uparrow = \overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}}$. Using the facts that $(\cdot)^\uparrow$ only changes external calls and returns in a history, we get the following:

$$\begin{array}{lll} (\tau\Downarrow|_{\mathsf{ClAct}^+})^\uparrow & = & \overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}} \\ (\tau\Downarrow|_{\mathsf{ClAct}^+})^\uparrow|_{\mathsf{Act}_{\emptyset \to (M' \setminus M'')}} & = & [\overline{(\tau\Uparrow)|_{\mathsf{AbsAct}}}]|_{\mathsf{Act}_{\emptyset \to (M' \setminus M'')}} \\ (\tau\Downarrow)|_{\mathsf{ClAct}} & = & h_1^+ \end{array}$$

Similarly, one can show that $(\tau_\Downarrow|_{\mathsf{ExtAct}})^\uparrow = h_1^-$.

Recall that $(\tau\Downarrow) \in T(L)^+$, so that by Lemma 1 we obtain that $(\tau\Downarrow)|_{\mathsf{Act}} \in [\![L]\!]$. Let $h' = (\tau\Downarrow)|_{\mathsf{Act}}$, $h'' = (\tau\Downarrow)|_{\mathsf{ExtAct}}$. Since the library L is $\binom{\mathcal{R}^+}{\mathcal{G}}$-closed by hypothesis, $h' \in [\![L]\!]$, $h'|_{\mathsf{ClAct}} = h_1^+$, and $h_1^+\ \mathcal{R}^+\ h_2^+$, there exists a trace $\tau' \in T(L)^+$ such that $\tau'|_{\mathsf{ClAct}} = h_2^+$, and $\overline{(h'|_{\mathsf{AbsAct}})}\ \mathcal{G}\ \overline{(\tau'|_{\mathsf{AbsAct}})}$.

By Lemma 2 we can extend the trace τ' to add external calls and returns arbitrarily, provided that the resulting trace is still valid. In other words, we can find a trace $\tau_L \in T(L)^+$ such that $\tau_L|_{\mathsf{ClAct}} = h_2^+$, $\tau_L|_{\mathsf{AbsAct}} = \tau'|_{\mathsf{AbsAct}}$ and $(\tau_L|_{\mathsf{ExtAct}})^\uparrow = h_2^-$; Note that, since external calls and returns can be added arbitrarily, we can also require that they match the calls and returns to abstract methods contained in τ_2: $(\tau_L|_{\mathsf{ClAct}^+})^\uparrow = \overline{(\tau_2|_{\mathsf{AbsAct}})}$. At this point, we have the following:

$$\begin{array}{lll} h'|_{\mathsf{ClAct}} = h_1^+ & \mathcal{R}^+ & h_2^+ = \tau_L|_{\mathsf{ClAct}} \\ (h'')^\uparrow = h_1^- & \mathcal{R}^- & h_2^- = (\tau_L|_{\mathsf{ExtAct}})^\uparrow \end{array}$$

and $\tau\Downarrow|_{\mathsf{Act}} = h', \tau\Downarrow|_{\mathsf{ExtAct}} = h''$, we can infer that

$$\tau\Downarrow|_{\mathsf{Act}^+} (\mathcal{R}^+ \oplus \mathcal{R}_{\uparrow}^- \ \tau_L|_{\mathsf{Act}^+}.$$

Here the relation $\mathcal{R}_{\uparrow}^-$, is defined for histories of external actions, by letting $h_1 \ \mathcal{R}_{\uparrow}^- \ h_2$ if and only if $(h_1)^{\uparrow} \ \mathcal{R}^- \ (h_2)^{\uparrow}$.

Let us consider the traces $(\tau_L) \in T(L)^+$, and $\tau_2 \in T(L_2)^+$; the trace τ_L has been constructed so that $(\tau_L|_{\mathsf{ClAct}^+})^{\uparrow} = \overline{(\tau_2|_{\mathsf{AbsAct}})}$. By Lemma 8 we can find $\tau_L' \in T(L)^+$ and $\tau_2' \in T(L_2)^+$ such that $\tau_L' \approx \tau_L, \tau_2' \approx \tau_2$ and $(\tau_2' \circ \tau_L')$ is defined and in $T(L_2 \circ L)^+$.

Finally, we compare the traces τ and $\tau_2' \circ \tau_L'$. For client actions, we have

$$\tau|_{\mathsf{ClAct}} \sqsubseteq \tau_2|_{\mathsf{ClAct}} = \tau_2'|_{\mathsf{ClAct}} = (\tau_2' \circ \tau_L)|_{\mathsf{ClAct}}$$

For abstract actions, we wish to prove that $\overline{\tau|_{\mathsf{AbsAct}}} \ (\mathcal{G} \oplus \mathcal{R}^-) \ \overline{(\tau_2' \circ \tau_L')|_{\mathsf{AbsAct}}}$. Note that $\tau|_{\mathsf{AbsAct}} = (\tau\Downarrow)|_{\mathsf{AbsAct}^+}^{\downarrow}$, so that we only need to show

(i)
$$\overline{\tau\Downarrow|_{\mathsf{AbsAct}}} \ \mathcal{G} \ \overline{\tau_L|_{\mathsf{AbsAct}}}$$

(ii)
$$\overline{(\tau\Downarrow|_{\mathsf{ExtAct}})^{\downarrow}} \ \mathcal{R}^- \ \overline{(\tau_L|_{\mathsf{ExtAct}})^{\downarrow}}$$

to conclude that $\tau|_{\mathsf{Act}} \sqsubseteq_{(\mathcal{G} \oplus \mathcal{R}^-)} (\tau_2' \circ \tau_L')|_{\mathsf{Act}})$.

For (i), we have that

$$\overline{\tau\Downarrow|_{\mathsf{AbsAct}}} = \overline{h'|_{\mathsf{AbsAct}}} \ \mathcal{G} \ \overline{\tau'_{\mathsf{AbsAct}}} = \overline{\tau_L|_{\mathsf{AbsAct}}}$$

so that there is nothing to prove. For (ii), we have showed that

$$(\tau\Downarrow)|_{\mathsf{ExtAct}}^{\uparrow} \mathcal{R}^- \ (\tau_L|_{\mathsf{ExtAct}})^{\uparrow}$$

But for any history h, which contains only external calls and returns, it is trivial to note that $h^{\uparrow} = \overline{h^{\downarrow}}$. Therefore we have that

$$\overline{(\tau\Downarrow)|_{\mathsf{ExtAct}}^{\downarrow}} \ \mathcal{R}^- \ \overline{(\tau_L|_{\mathsf{ExtAct}})^{\downarrow}}$$

as we wanted to prove. □

Proof of Theorem 4: This is now a straightforward application of Theorem 8. Let $L_1, L_2 : M \to M'$, $L : M'' \to M$ be such that $M \cap M' = \emptyset, M'' \cap M = \emptyset$; suppose that L is $\binom{\mathcal{R}}{\mathcal{G}}$ closed, and that both $(L_1 \circ L)\downarrow, (L_2 \circ L)\downarrow$. Note that we can rewrite any relation $\mathcal{R}$ between histories of type $\emptyset \to M$ as $\mathcal{R} \oplus \{(\varepsilon, \varepsilon)\}$. Theorem 8 gives that $L_1 \sqsubseteq_{\mathcal{R}} L_2$ implies that $(L_1 \circ L) \sqsubseteq_{(\mathcal{G} \oplus \{(\varepsilon,\varepsilon)\})} (L_2 \circ L)$, which can be rewritten as $(L_1 \circ L) \sqsubseteq_{\mathcal{G}} (L_2 \circ L)$. □

Next, we give a special closure property which is satisfied by any library.

LEMMA 9 *Any library $L : M \to M'$ is $\left(\begin{smallmatrix}\overline{\sqsubseteq}\\ \overline{\overline{\sqsubseteq}}\end{smallmatrix}\right)$-closed.*

This property alone is not sufficient to prove the contextuality of encapsulated linearisability. We also need to show that, under the hypothesis that $M' \cap M'' = \emptyset$ in a library of the form $L_2 \circ L_1$, if a call to abstract methods invoked by L_1 is followed by a return of an abstract method previously invoked by L_2 in a history of $L_2 \circ L_1$, then these two actions can be swapped and the resulting history is still included in $L_2 \circ L_1$. This is a consequence of the fact that the order of execution of primitive commands executed by L_2 and L_1, in two different threads, can be swapped. Formally:

LEMMA 10 *Whenever* $\sigma_2 \rightsquigarrow^{L_2}_{c_2,t_2} \sigma_2'$, $\sigma_1 \rightsquigarrow^{L_1}_{c_1,t_1} \sigma_1'$, $t_1 \neq t_2$ *and* $(\sigma_2 \cdot \sigma_1)\downarrow$ *then we have both* $(\sigma_2 \cdot \sigma_1) \rightsquigarrow^{L_2 \circ L_1}_{c_2,t_2} \rightsquigarrow^{L_2 \circ L_1}_{c_1,t_1} \sigma_2' \cdot \sigma')$, *and* $(\sigma_2 \cdot \sigma_1 \rightsquigarrow^{L_2 \circ L_1}_{c_1,t_1} \rightsquigarrow^{L_2 \circ L_1}_{c_2,t_2} \sigma_2' \cdot \sigma')$.

Proof. An immediate consequence of the fact that $\mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) \subseteq \{\mathsf{arg}_t\}_{t \in \mathcal{T}}$, and $t_1 \neq t_2$. □

Proof of Theorem 3 (Outline):

(i) This is a direct consequence of Theorem 4, Lemma 9, and Lemma 10. In fact, note that encapsulated linearisability can be obtained from up-to linearisability $\sqsubseteq_{\mathcal{R}}$, by instantiating the the relation $\mathcal{R}$ to be exactly $\overline{\sqsubseteq}$. That is, for $L_1, L_2 : M \to M'$ such that $M \cap M' = \emptyset$, $L_1 \sqsubseteq_e L_2$ if and only if $L_1 \sqsubseteq_{\overline{(\sqsubseteq)}} L_2$. From theorem 4 and the fact that every library L is $(\frac{\overline{(\sqsubseteq)}}{(\sqsubseteq)})$-closed, Lemma 9, it follows that if $L_1 \sqsubseteq_{\overline{(\sqsubseteq)}} L_2$, then $(L_1 \circ L) \sqsubseteq_{\overline{(\sqsubseteq \oplus \sqsubseteq)}} (L_2 \circ L)$.
More specifically, we have that whenever $\tau|_{\mathsf{Act}} \in [\![L_1 \circ L]\!]$, there exists $\tau' \in T^+(L_2 \circ L)$ such that
(i) $\overline{\tau|_{\mathsf{ClAct}}} = \overline{\tau\Uparrow|_{\mathsf{ClAct}}} \sqsubseteq \overline{\tau'\Uparrow|_{\mathsf{ClAct}}} = \overline{\tau'|_{\mathsf{ClAct}}}$,
(ii) $\overline{\tau|_{\mathsf{AbsAct}}}\ (\overline{(\sqsubseteq)} \oplus \overline{(\sqsubseteq)})\ \overline{\tau'|_{\mathsf{AbsAct}}}$.
Note that in general, the sequences $\tau|_{\mathsf{AbsAct}}$, $\tau'|_{\mathsf{AbsAct}}$ may not be histories; however, we commit an abuse of notation and rewrite condition (ii) as

$$\tau|_{\mathsf{AbsAct}} (\sqsubseteq \oplus \sqsubseteq)\ \tau'|_{\mathsf{AbsAct}}$$

Also, we will refer to $\tau|_{\mathsf{AbsAct}}, \tau'|_{\mathsf{AbsAct}}$ as histories, rather than sequences of actions.
Condition (ii) alone does not guarantee that $\tau|_{\mathsf{AbsAct}} \sqsubseteq \tau'|_{\mathsf{AbsAct}}$, which together with condition (i) would lead to $\tau|_{\mathsf{Act}} \sqsubseteq_e \tau'|_{\mathsf{Act}}$. Rather, it states that
(a) the histories $\tau|_{\mathsf{AbsAct}}$ can be split into two sub-histories, corresponding to the contribution of the libraries L_1, L to $\tau|_{\mathsf{AbsAct}}$, respectively; these sub-histories are $(\tau\Uparrow)|_{\emptyset \to (M \cap M')}$ (the calls that library L_1 performs to methods which are abstract in $L_1 \circ L$) and $(\tau\Downarrow)|_{\mathsf{AbsAct}}$ (the calls that L performs to methods which are abstract in $L_2 \circ L$,
(b) similarly, $\tau'|\mathsf{AbsAct}$ can be split in two sub-histories $(\tau'\Uparrow)|_{\emptyset \to (M \cap M')}$ and $(\tau'\Downarrow)|_{\mathsf{AbsAct}}$,
(c) for such histories, we have that

$$(\tau\Uparrow)|_{\emptyset \to (M \cap M')} \sqsubseteq (\tau'\Uparrow)|_{\emptyset \to (M \cap M')}$$
$$(\tau\Downarrow)|_{\mathsf{AbsAct}} \sqsubseteq (\tau'\Downarrow)|_{\mathsf{AbsAct}}$$

Thus it is possible that in τ, an action of the form $(t_1, \mathsf{call!}\, m_1(z_1))$, where m_1 has been invoked by the library L_2, precedes an action of the form $(t_2, \mathsf{ret?}\, m_2(z_2))$, where the method m_2 has been previously invoked by L_1. However, the precedence order of such actions is not preserved in τ'.
Without loss of generality, let us suppose that

$$\tau|_{\mathsf{Act}} = h_1(t_1, \mathsf{call!}\, m_1(z_1))(t_2, \mathsf{ret?}\, m_2(z_2))h_2$$
$$\tau'|_{\mathsf{Act}} = h_1'(t_2, \mathsf{ret?}\, m_2(z_2))(t_1, \mathsf{call!}\, m_1(z_1))h_2'$$

Using Lemma 10 and Corollary 1, presented later, we can construct a trace $\tau'' \in T^+(L_2 \circ L)$ such that $\tau''|_{\mathsf{Act}} = h_1'(t_1, \mathsf{call!}\, m_1(z_1))(t_2, \mathsf{ret?}\, m_2(z_2))h_2'$ (intuitively,

this is because we can rearrange the order of execution of primitive commands executed after m_2 has been returned and before m_1 is being called).

In general, in $\tau'|_{\mathsf{AbsAct}}$, calls to and returns of abstract methods performed by the library L_2, and calls to and returns of abstract methods performed by L_1, can be arbitrarily rearranged, and the resulting history would still be in $[\![L_2 \circ L_1]\!]|_{\mathsf{AbsAct}}$. That is, we can find a trace $\tau'' \in T^+(L_2 \circ L_1)$ which satisfies (i)-(iii) above, and which also preserves the order of calls to abstract methods performed by L_2 (L_1) and returns of abstract methods previously invoked by L_1 (L_2), with respect to τ. That is, $\tau|_{\mathsf{Act}} \sqsubseteq_e \tau''_{|_{act}}$.

(ii) This part of the Theorem follows directly from Theorem 4 (ii), by instantiating $\mathcal{R}$ with $\overline{(\sqsubseteq)}$. □

Compositionality of Linearisability. Next, we turn our attention to proving that the our notions of linearisability are also compositional. First, a simple result concerning the decomposition and recomposition of traces, with respect to the operator $\uplus$. For a given library L, we let $T(L) = \{\tau \mid [\![\tau]\!](\lambda l \in \mathsf{Locs}_L.0) = \mathsf{true}\}$.

LEMMA 11 *Let $h_1 \in [\![L_1]\!], h_2 \in [\![L_2]\!]$, where $L_1 : M_1 \to M_1', L_2 : M_2 \to M_2'$ are two libraries such that $(L_1 \uplus L_2)\downarrow$. Whenever $h \in (\{h_1\} \bowtie \{h_2\})$, then $h \in [\![L_1 \uplus L_2]\!]$.*

Whenever $h \in [\![L_1 \uplus L_2]\!]$, then $h|_{\mathsf{Act}_{M_1 \to M_1'}} \in [\![L_1]\!]$, $h|_{\mathsf{Act}_{M_2 \to M_2'}} \in [\![L_2]\!]$. that $h_1 \in [\![L_1]\!], h_2 \in [\![L_2]\!]$.

Proof of Theorem 5: Given a library $L : M \to M'$, let $\mathsf{Act!}_{M \to M'}$ be the set of actions of the form $(t, \mathsf{call!}\, m(z)), m \in M$ and $(t, \mathsf{ret!}\, m'(z)), m' \in M'$. The sets $\mathsf{Act?}_{M \to M'}$ is defined similarly. We only give the details for (i); details for the other cases are similar.

Let $L_1, L_2 : M \to M'$ $L : M_L \to M_L'$ be such that $(L_1 \uplus L)\downarrow$, $(L_2 \uplus L)\downarrow$. First, we prove that $L_1 \sqsubseteq L_2 \implies (L_1 \uplus L) \sqsubseteq (L_2 \uplus L)$.

Let $h \in [\![L_1 \uplus L]\!])$. By Lemma 11 we can find two histories $h_1 \in [\![L_1]\!], h_L \in [\![L]\!]$, and such that $h \in \{h_1\} \bowtie \{h_L\}$. That is, we can assume two strictly monotone mappings $f : \{1, \cdots, |h_1|\} \to \{1, \cdots, |h|\}$ and $g : \{1, \cdots, |h_L|\} \to \{1, \cdots, |h|\}$, with disjoint images, such that $h_1(i) = h(f(i))$, $h_2(j) = h(g(j))$ for an $i \leq |h_1|, j \leq |h_L|$.

Since we are assuming that $L_1 \sqsubseteq L_2$, there exists a history $h_2 \in [\![L_2]\!].h_1 \sqsubseteq h_2$. By definition, there exists a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that $h_2(i) = (h_1(\pi(i))$, if $i < j$, $h_1(i) = (t, \cdot), h_1(j) = (t, -)$ then $\pi(i) < \pi(j)$, and whenever $i < j$, $h_1(i) \in \mathsf{Act!}_{M \to M'}, h_2(i) \in \mathsf{Act?}_{M \to M'}$, then $\pi(i) < \pi(j)$.

Any trace $h' \in \{h_2\} \bowtie \{h_L\}$ is uniquely defined by two strictly monotone functions $f' : \{1, \cdots, |h_2|\} \to \{1, \cdots, |h|\}$ and $g' : \{1, \cdots |h_L|\} \to \{1, \cdots, |h|\}$, with disjoint images, such that $h_2(i) = h'(f(i))$, $h_L(j) = h'(g(j))$ for an $i \leq |h_2|, j \leq |h_L|$.

In particular, we consider a trace $h' \in \{h_2\} \bowtie \{h_L\}$ such that, for any i, j with $f(i) < g(j)$, $h_1(i) \in \mathsf{Act!}$ and $h_2(j) \in \mathsf{Act?}$, then $f'(\pi(i)) < g'(j)$. Similarly, if i, j are such that $g(i) < f(j)$, $h_2(i) \in \mathsf{Act!}$ and $h_1(j) \in \mathsf{Act?}$, then $h(i) < h(j)$. Intuitively, this property states that the histories h_2 and h_L are combined so that the precedence order of pairs in the set $\mathsf{Act!}_{M \to M'} \times \mathsf{Act?}_{M_L \to M_L'}$ in h is preserved in h'; similarly for pairs in the set $\mathsf{Act!}_{M_L \to M_L'} \times \mathsf{Act?}_{M \to M'}$. Also, we require that the precedence order of actions performed by the same thread in h is preserved in h'. Note that, since

$h' \in \{h_2\} \bowtie \{h_L\}$, we have that $h'|_{\mathsf{Act}M_2 \to M_2'} \in [\![L_2]\!]$; similarly, $h'|_{\mathsf{Act}_{M_L \to M_L'}} \in [\![L]\!]$. By Lemma 11, it is ensured that $h' \in [\![L_2 \uplus L]\!]$.

Let π' be the permutation defined as

$$\pi'(i) = \begin{cases} f'(\pi(f^{-1}(i))) & \text{if } h(i) \in \mathsf{Act}_{M \to M'} \\ g'(g^{-1}(i)) & \text{if } h(i) \in \mathsf{Act}_{M_L \to M_L'} \end{cases}$$

Note that if $\pi'(i) = \pi'(j)$ then $i = j$, since $f^{-1}, f', \pi, g^{-1}, g'$ are injective (in practice f^{-1}, g^{-1} are also partial, though we use them only on elements for which they are defined). Also, note that for any $i = 1, \cdots, |h|$, $h(i) = h'(\pi'(i))$. To prove this, suppose first that i is such that $h(i) \in \mathcal{H}_{M \to M'}$. In this case $f^{-1}(i)$ is defined, and we get that

$$h(i) = h_1(f^{-1}(i)) = h_2(\pi(f^{-1}(i)) = h'(f'(\pi(f^{-1}(i))).$$

A similar argument follows for the case $h(i) \in \mathcal{H}_{M_L \to M'}$. It remains to notice that the functions f', g' have been chosen so that the induced permutation π' preserves thread local histories, and the precedence order of pairs in $(\mathsf{Act}! \times \mathsf{Act}?)$ with respect to h. That is, π' is a witness permutation to show that $h \sqsubseteq h'$.

We have proved that, whenever $L_1 \sqsubseteq L$, $(L_1 \uplus L)\downarrow$ and $(L_2 \uplus L)\downarrow$, then $(L_1 \uplus L) \sqsubseteq (L_2 \uplus L)$. Simmetrically, we also have that in this case we also have that if $(L_1 \sqsubseteq L_2), (L \uplus L_1)\downarrow$ and $(L \uplus L_2)\downarrow$, then $(L_1 \uplus L_2)\downarrow$. This is because of $\uplus$ being associative.

Now, let $L_1 \sqsubseteq L_1', L_2 \sqsubseteq L_2'$, and suppose that both $(L_1 \uplus L_2)\downarrow$, $(L_1' \uplus L_2')\downarrow$. From the statements above, we can conclude that $(L_1 \uplus L_2) \sqsubseteq (L_1' \uplus L_2) \sqsubseteq (L_1' \uplus L_2')$, as we wanted to prove. □

Soundness With Respect To Observational Refinement It remains to prove that our notions of linearisability imply observational refinement. The key lemma to this lies in readapting the result from [1] to ground libraries of the form $L : \emptyset \to M$.

Theorem 1. *For any two ground libraries $L_1, L_2 : \emptyset \to M$, $L_1 \mathrel{\Theta} L_2 \implies L_1 \sqsubseteq_{\mathsf{obs}} L_2$, where $\Theta \in \{\sqsubseteq, \sqsubseteq_e, \sqsubseteq_{\{(\varepsilon,\varepsilon)\}}\}$.*

Proof. Note that, for ground libraries, the three notions of linearisability considered are equivalent to classical linearisability, as defined in [1]. The result follows then from a direct application of Corollary **40** in [1]. □

Proof of Theorem 6: We only prove statement (iii); the proofs for statements (i) and (ii) are analogous, using theorems 2 and 3 in lieu of Theorem 4.

Let $L_1, L_2 : M \to M'$ be two libraries. Suppose that $L_1 \sqsubseteq_{\mathcal{R}} L_2$, for some binary relation $\mathcal{R}$; then, for any library $\mathcal{R}$-closed ground library $L : \emptyset \to M$ such that $(L_1 \circ L)\downarrow$, Theorem 4 ensures that $(L_1 \circ L) \sqsubseteq_{\{(\varepsilon,\varepsilon)\}} (L_2 \circ L)$. Note that the latter can be rewritten as $(L_1 \circ L) \sqsubseteq (L_2 \circ L)$. The libraries $(L_1 \circ L), (L_2 \circ L)$ are both ground libraries of type $\emptyset \to M'$, so that by Theorem 1 we know that, for any client $(C_1 \parallel \ldots \| C_n)$, then

$$\mathsf{obs}([\![\mathsf{let}\ (L_1 \circ L)\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n]\!]) \subseteq \mathsf{obs}([\![\mathsf{let}\ (L_2 \circ L)\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n]\!]).$$

Since the library parameter L has been chosen to be an arbitrary $\mathcal{R}$-closed library, and the client $(C_1 \parallel \ldots \parallel C_n)$ has also been chosen arbitrarily, we may conclude that $L_1 \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} L_2$. □

B.1 Linearisability Coincides with Strong Linearisability

LEMMA 12 *Let $L : M \to M'$; then*

(i) $(\!|L|\!) \subseteq (\!|L|\!)^+$,
(ii) $(\!|L|\!)^+ \,|_{\mathsf{Act}} \subseteq (\!|L|\!)$.

Proof. Follows immediately from the definition of $(\!|\cdot|\!)^+$. □

LEMMA 13 *For any $L : M \to M'$,*

(i) $\forall \tau \in (\!|L|\!)^+, \sigma \in \mathcal{H}. [\![\tau]\!]^+_L \sigma = [\![\tau|_{\mathsf{TrAct}}]\!]_L \sigma$,
(ii) $\forall \tau \in (\!|L|\!), \tau' \in (\!|L|\!)^+. \tau'|_{\mathsf{TrAct}} = \tau \implies \forall \sigma \in \mathcal{H}. [\![\tau]\!]_L \sigma = [\![\tau']\!]^+_L \sigma$.

Proof (Outline).

(i) It suffices to note that, if $\tau \in (\!|L|\!)^+$, then actions of the form (t, α), with $\alpha \in \{\iota, \mathsf{ExtCall}\, m(z), \mathsf{ExtRet}\, m(z) \mid m \in M', z, z' \in \mathbb{Z}\}$ never happen within two actions $(t, \cdot) \in \mathsf{Act?}$ and $(t, \cdot) \in \mathsf{Act!}$.
whenever $\sigma \rightsquigarrow^L_{\alpha,t} \sigma'$, then $\sigma' = \sigma[\mathsf{arg}_t \mapsto z]$, for some $z \in \mathbb{Z}$. Then we can show that whenever
$$\sigma_0 \rightsquigarrow^L_{\alpha_1,t} \sigma_1 \rightsquigarrow^L_{\alpha_2,t} \cdots \rightsquigarrow^L_{\alpha_n,t} \sigma_n$$
where $\alpha_i \in \{\iota, \mathsf{ExtCall}\, m(z), \mathsf{ExtRet}\, m(z) \mid m \in M', z, z' \in \mathbb{Z}\}$, then $\sigma_n = \sigma_0[\mathsf{arg}_t \mathbb{N} \to z]$. This fact can be used to prove that $\sigma \rightsquigarrow^L_{\mathsf{call?}\, m(z),t} \sigma'$ if and only if $\sigma \rightsquigarrow^L_{\alpha_1,t} \sigma_1 \rightsquigarrow^L_{\alpha_2,t} \cdots \rightsquigarrow^L_{\alpha_n,t} \sigma \rightsquigarrow^L_{\mathsf{call?}\, m(z),t} \sigma'$. Similarly, $\sigma \rightsquigarrow^L_{\mathsf{ret!}\, m(z),t} \sigma'$ if and only if $\sigma \rightsquigarrow^L_{\mathsf{ret!}\, m(z),t} \sigma_0 \rightsquigarrow^L_{\alpha_1,t} \cdots \rightsquigarrow^L_{\alpha_n,t} \sigma_n = \sigma'$. In both cases we assume that $\alpha_i \in \{\iota, \mathsf{ExtCall}\, m(z), \mathsf{ExtRet}\, m(z) \mid m \in M', z, z' \in \mathbb{Z}\}$. Thus, actions $\iota, \mathsf{ExtCall}\, m(z), \mathsf{ExtRet}\, m(z)$ in a trace $\tau \in [\![L]\!]^+$ do not affect the evaluation function; that is, $[\![\tau]\!]^+_L = [\![\tau|_{\mathsf{Act}}]\!]_L$.

(ii) This follows immediately from the statement above. □

Proof of Lemma 1: This is a direct consequence of lemmas 12 and 13. In fact, for any $\tau \in [\![L]\!]$ we have that $\tau \in (\!|L|\!)$, hence $\tau \in (\!|L|\!)^+$. Now

$$[\![\tau]\!]^+_L(\lambda l \in \mathsf{Locs}_L.0) = [\![\tau|_{\mathsf{Act}}]\!]_L(\lambda l \in \mathsf{Locs}_L.0) = [\![\tau]\!]_L(\lambda l \in \mathsf{Locs}_L.0) = \mathsf{true}$$

hence $\tau \in [\![L]\!]^+_L$. Also, for any $\tau \in [\![L]\!]^+$, we have that $\tau \in (\!|L|\!)^+$, which gives $\tau \,|_{\mathsf{Act}} \in (\!|L|\!)$. Also, $[\![\tau|_{\mathsf{Act}}]\!]_L = [\![\tau]\!]^+_L = \mathsf{true}$, hence $\tau|_{\mathsf{Act}} \in [\![L]\!]$. □.

Proof of Lemma 2: This is a direct consequence of Lemma 1. We give the details only for the second statement, as the first is easier to prove. If $h = (h_1)(h_2)(h_3)$, then there exists a trace τ_1, τ_2, τ_3 such that $(\tau_1)(\tau_2)(\tau_3) \in (\!|L|\!)^+$, $[\![(\tau_1)(\tau_2)(\tau_3)]\!](\lambda l \in \mathsf{Locs}_L.0) = \mathsf{true}$ and $\mathsf{history}^+((\tau_1)(\tau_2)(\tau_3)) = (h_1)(h_2)(h_3)$. Also, since $h_1|_t$ is complete, so is $\tau_1|_t$, so that $\tau_1(\mathsf{ExtCall}\, m(z)) \in (\!|L|\!)^+$. Now, since $\tau_1|_t$ is complete, τ_2 immediately follows τ_1 in $(\tau_1)(\tau_2)(\tau_3)$, and $\tau_2|_t = \varepsilon$, we have that $\tau_2 \subseteq (t, \iota^*)$. This gives that $(\tau_1)(t, \mathsf{ExtCall}\, m(z))(\tau_2)(t, \mathsf{ExtRet}\, m(z')) \in (\!|L|\!)^+$, from which $(\tau_1)(t, \mathsf{ExtCall}\, m(z))(\tau_2)(t, \mathsf{ExtRet}\, m(z'))(\tau_3) \in (\!|L|\!)^+$. Since $[\![(\tau_1)(\tau_2)(\tau_3)]\!]_L(\lambda l \in \mathsf{Locs}_L.0) = \mathsf{true}$, and $[(\tau_1)(t, \mathsf{ExtCall}\, m(z))(\tau_2)(t, \mathsf{ExtRet}\, m(z'))(\tau_3)]|_{\mathsf{Act}} = (\tau_1)(\tau_2)(\tau_3)$ we also have that $[\![(\tau_1)(t, \mathsf{ExtCall}\, m(z))(\tau_2)(t, \mathsf{ExtRet}\, m(z'))(\tau_3)]\!]_L(\lambda l \in \mathsf{Locs}_L.0) = \mathsf{true}$, hence $\mathsf{history}^+((\tau_1)(t, \mathsf{ExtCall}\, m(z))(\tau_2)(t, \mathsf{ExtRet}\, m(z'))(\tau_3)) = (h_1)(t, \mathsf{ExtCall}\, m(z))(h_2)(t, \mathsf{ExtRet}\, m(z'))(h_3) \in [\![L]\!]^+$. □

Proof of Theorem 7:

($\sqsubseteq^+$) **implies** ($\sqsubseteq$)**:** Let us first show that strong linearisability implies linearisability. Let $L_1, L_2 : M \to M'$ be such that $L_1 \sqsubseteq^+ L_2$. Suppose that $h \in [\![L_1]\!]$; we need to show that there exists $h_2 \in [\![L_2]\!]$ such that $h_1 \subseteq h_2$. By Lemma 1(i) we know that $h_1 \in [\![L_1]\!]^+$, so we can apply the hypothesis to infer a history $h_2 \in [\![L_2]\!]^+$ such that $h_1 \sqsubseteq^+ h_2$. This constraint implies that $h_2|_t = h_1|t$ for all $t \in \mathcal{T}$; Since $h_1 \in [\![L_1]\!]$, it does not contain any action in ExtAct, nor does h_2. That is, $h_2|_{\mathsf{Act}} = h_2$, and by Lemma 1(ii) we have that $h_2 \in [\![L]\!]$. Finally, the permutation π used to prove that the order of pairs of actions in $\mathsf{Act!}^+ \times \mathsf{Act?}^+$ in h_1 is preserved in h_2, can also be used to prove that the order of pairs of actions in $\mathsf{Act!} \times \mathsf{Act?}$ in h_1 is preserved in h_2. That is, $h_1 \sqsubseteq h_2$.

($\sqsubseteq^+$) **implies** ($\sqsubseteq$)**:** Suppose that $L_1 \sqsubseteq L_2$. Let $h_1 \in [\![L_1]\!]^+$; we need to find $h_2 \in [\![L_2]\!]^+$ such that $h_1 \sqsubseteq^+ h_2$. By Lemma 1(ii) we know that $h_1' = h_1|_{\mathsf{Act}} \in [\![L_1]\!]$. Since $h_1' := h_1|_{\mathsf{Act}}$, there exists a strictly monotone function $f : \{1, \cdots, |h_1'|\} \to \{1, \cdots, |h_1|\}$ such that $h_1(f(i)) = h_1'(i)$. Also, given $h_{\mathsf{ext}} := h_1|_{\mathsf{ExtAct}}$, we can find a function $g : \{1, \cdots, |h_{\mathsf{ext}}|\} \to \{1, \cdots, |h_1|\}$ such that $h_1(g(i)) = h_{\mathsf{ext}}(i)$; further, the images of f and g are disjoint.

By the hypothesis $L_1 \sqsubseteq L_2$ we can find $h_2' \in [\![L_2]\!]$ such that $h_1' \sqsubseteq h_2'$. Therefore, $h_1'|_t = h_2'|_t$ for any thread t, and there exists a permutation π which preserves the order of pairs in $\mathsf{Act!} \times \mathsf{Act?}$, with respect to h_1', in h_2'. Now, let $f' : \{1, \cdots, |h_2'|\} \to \{1, \cdots, |h_1|\}$, $g' : \{1, \cdots, |h_{\mathsf{ext}}|\} \to \{1, \cdots, |h_1|\}$ be two strictly monotone functions, with disjoint images, such that if

(i) if i, j are two indexes such that $f(i) < g(j)$, and $h_1(f(i)) = (t, \alpha)$, $h_2(g(j)) = (t, \beta)$, then $i < j$,

(ii) if i, j are such that $g(j) < f(i)$, and $h_2(g(j)) = (t, \alpha)$, $h_2(f(i)) = (t, \beta)$, then $g'(j) < f'(i)$,

(iii) if i, j are two indexes such that $f(i) < g(j)$, and $h_1(f(i)) \in \mathsf{Act!}$, then $(f'(\pi(i)) < g'(j)$,

(iv) if i, j are two indexes such that $g(j) < f(i)$, and $f(i) \in \mathsf{Act?}$, then $g'(j) < f'(\pi(j))$.

Consider the history h_2 obtained by letting $h_2(f'(i)) = h_2'(i), h_2(g'(j)) = h_{\mathsf{ext}}(j)$. The first two conditions state that h_2 preserves the projections of h_1 to thread-local traces, while the last two conditions state that h_2 preserves the order of pairs in $(\mathsf{Act!} \times \mathsf{ExtAct}) \cup (\mathsf{Act?} \times \mathsf{ExtAct})$, with respect to h_1. Also, since h_2 has been constructed from a history which linearises $h_1|_{\mathsf{Act}}$, preserving the order of its actions, it also preserves the order of pairs of in $(\mathsf{Act!} \times \mathsf{Act?})$. We may conclude that $h_1 \sqsubseteq^+ h_2$.

It remains to show that the augmented history h_2 we have constructed belongs to $[\![L_2]\!]^+$. To this end, we need to show that

(i) if $h_2 = (h_a)(t, \mathsf{ExtCall}\, m(z))(h_b)$ then $(h_a)|_t$ is complete, and either $h_b|_t = \varepsilon$ or $h_b|_t = (\mathsf{ExtRet}\, m(z'))(h_c)$

(ii) whenever $h_2 = (h_b)(t, \mathsf{ExtRet}\, m(z'))(h_c)$ then $h_b|_t = (h_a)(t, \mathsf{ExtCall}\, m(z))$.

Then the result follows from iterated applications of Lemma 2 to the augmented history $h_2' \in [\![L_2]\!]$. The proof of the two statements above is trivial, as $h_2|_t = h_1|_t$ for any thread t, and the two properties above are satisfied by $h_1|_t$. □

B.2 Decomposition and Composition of Histories

Throughout this section we will work extensively with the constraint automaton $\mathcal{A}$, defined in Figure 7. It is therefore convenient to introduce some notation. We remark that we assume the library $L_1 : M'' \to M', L_2 : M' \to M$ to be fixed, and that $(L_2 \circ L_1)\downarrow$. When these assumptions are dropped, the notation introduced in this section has to be made parametric in the set of methods M', which cannot be inferred from the type $M \to M''$ of the composite library $(L_2 \circ L_1)$.

For a given configuration (s, v_2, v_1) of the automaton $\mathcal{A}$, we write $(s, v_2, v_1) \xrightarrow{(\alpha,\beta,\gamma)} (s', v_2', v_2')$ if in configuration (s, v_2, v_1) the action (α, β, γ) is enabled, according to the transitions described in Figure 7, and such a transition leads to the configuration (s', v_2', v_2'). Otherwise, we write $(s, v_2, v_1) \xrightarrow{(\alpha,\beta,\gamma)} \not\rightarrow$.

Given a configuration (s^0, v_2^0, v_1^0) and a triple of strings $(\psi'', \psi', \psi) = (\alpha_1 \cdots \alpha_n, \beta_1 \cdots \beta_n, \gamma_1 \cdots \gamma_n)$, we write $(s^0, v_2^0, v_1^0) \xRightarrow{(\psi'',\psi,\psi)} (s^n, v_0^n, v_1^n)$ if and only if

$$(s^0, v_2^0, v_1^0) \xrightarrow{(\alpha_1,\beta_1,\gamma_1)} (s^1, v_2^1, v_1^1) \xrightarrow{(\alpha_2,\beta_2,\gamma_2)} \cdots \xrightarrow{(\alpha_n,\beta_n,\gamma_n)} (s^n, v_2^n, v_1^n)$$

In this case we say that the string (ψ'', ψ', ψ) is ***accepted*** by $\mathcal{A}_{(s,v_2^0,v_1^0)}$. We write $(s, v_2, v_1) \xRightarrow{(\psi'',\psi',\psi)}$ if there exists (s', v_2', v_1') such that $(s, v_2, v_1) \xRightarrow{(\psi'',\psi',\psi)} (s', v_2', v_1')$.

We can now give the formal definition of the operators $\Uparrow, \Downarrow$ and $\circ$, for sequences of actions $\psi = \alpha_1 \cdots \alpha_n$.

DEFINITION 15 *Given a triple of strings (ψ_2, ψ_1, ψ) such that $(s, v_2, v_1) \xRightarrow{(\psi_2,\psi_1,\psi)}$, we let $\psi\Uparrow_{(s,v_2,v_1)} = \psi_2$ and $\psi\Downarrow_{(s,v_2,v_1)} = \psi_1$. If $(s, v_2, v_1) = (\mathsf{Client}, 0, 0)$ then we define $\psi\Uparrow = \psi\Uparrow_{(\mathsf{Client},0,0)}$ and $\psi\Downarrow = \psi\Downarrow_{(\mathsf{Client},0,0)}$.*

Given a string (ψ'', ψ', ψ) such that $(s, v_2, v_1) \xRightarrow{(\psi'',\psi',\psi)}$, we define $\psi'' \circ \psi' = \psi$.

PROPOSITION 1 *If $(s, v_2, v_1) \xrightarrow{(\alpha_1,\beta_1,\gamma)} (s_1, v_2^1, v_1^1)$ and $(s, v_2, v_1) \xrightarrow{(\alpha_1,\beta_1,\gamma)} (s_2, v_2^2, v_1^2)$, then $(\alpha_1, \beta_1) = (\alpha_2, \beta_2)$ and $(s_1, v_2^1, v_1^1) = s_2, v_2^2, v_1^2)$.*

Proof. It suffices to perform a case analysis over the transitions defined for any configuration(s, v_2, v_1) in Figure 7. Details are left to the reader. □

PROPOSITION 2 *If $(s, v_2, v_1) \xrightarrow{(\alpha,\beta,\gamma_1)} (s_1, v_2^1, v_1^1)$ and $(s', v_2', v_1') \xrightarrow{(\alpha,\beta,\gamma_2)} (s_2, v_2^2, v_1^2)$, then $\gamma_1 = \gamma_2$ and $(s', v_2', v_1') = (s, v_2, v_1)$ implies that $(s_2, v_2^2, v_1^2) = (s_2, v_2^2, v_1^2)$.*

Proof. By a case analysis over the transitions defined for $\mathcal{A}$ in Figure 7. In particular, note that the only actions which agree in the first two components are (a) - $(\mathsf{Client}, v_2, v_1) \to (\iota, \iota, \iota)$ - and (p) - $(\mathsf{Param}, v_2, v_1) \to (\iota, \iota, \iota)$ - which also agree on the third one. □

Proof of Lemma 3: To prove that $\Uparrow, \Downarrow$ are well defined, It suffices to prove that whenever $(s, v_2, v_1) \xRightarrow{\psi_2', \psi_1', \psi)} (s', v_2', v_1')$, and $(s, v_2, v_1) \xRightarrow{\psi_2'', \psi_1'', \psi)} (s'', v_2'', v_1'')$, then $\psi_2'' = \psi_2'$, and $\psi_1'' = \psi_1'$. This can be done via a straightforward induction over $|\psi|$, using Proposition 1.

To prove that $\circ$ is well defined, it suffices to show that whenever $(s', v_2', v_1') \xRightarrow{\psi_2, \psi_1, \psi')}$ and $(s'', v_2'', v_1'') \xRightarrow{(\psi_2, \psi_1, \psi'')}$, then $\psi'' = \psi'$. This can be proved by induction over $|\psi|$, using Proposition 2.

Finally, if ψ is such that $\psi\Uparrow, \psi\Downarrow$ are defined, then $(\mathsf{Client}, 0, 0) \xRightarrow{(\psi\Uparrow, \psi\Downarrow, \psi)}$ by definition. Also, by Definition of $\circ$, we get that $(\psi\Uparrow) \circ (\psi\Downarrow) = \psi$. $\square$

In the following, we let $(t, \psi)\Uparrow = (t, \psi\Uparrow)$, provided that the latter is defined. $(t, \psi)\Downarrow$ and $(t, \psi_1) \circ (t, \psi_2)$ are defined similarly.

Proof of Lemma 4: This is a consequence of Lemma 3, and the definition of $\Uparrow, \Downarrow, \cdot$ for augmented traces. Suppose that τ_1, τ_2 satisfy the constraints required by the definition of $\tau\Uparrow$. We need to show that $\tau_1 = \tau_2$. To this end, let $i : 1 \leq i \leq |\tau|$. If $\tau_1(i) = (t, \alpha_1)$, then $\tau_2(i) = (t, \beta_1)$, by the definition of $\Uparrow$. Also, let $f_1, f_2 : \{1, \cdots, |(\tau|_t)|\} \rightarrow \{1, \cdots, |(\tau)|\}$ be the strictly monotone functions such that $(\tau_1|_t)(i) = \tau_1(f_1(i))$ and $(\tau_2|_t)(i) = \tau_2(f_2(i))$. By the definition of $\Uparrow$ for augmented traces, we have that whenever $\tau_1(j) = (t, \cdot)$ then $\tau_2(j) = (t, -)$ for any j. This implies that $f_1 = f_2$. Also, by Lemma 3, we have that $\tau_1|_t = (\tau|_t)(\Uparrow) = (\tau_2)|_t$. Since $\tau_1(i) = (t, \alpha), \tau_2(t) = (t, \beta)$, both $f_1^{-1}(i)$ and $f_2^{-1}(i)$ are defined. Further,

$$\tau_1(i) = (\tau_1)|_t(f_1^{-1}(i)) = \tau_2|_t(f_1^{-1}(i)) = \tau_2|_t(f_2^{-1}(i)) = \tau_2(i)$$

A similar argument can be applied to show that $\tau \Downarrow$ is well defined.

Next, let τ', τ'' be two augmented traces which satisfy the definition of $\tau_1 \circ \tau_2$. We need to show that $\tau' = \tau''$. Let then $i \in \{1, \cdots, |\tau'|\}$. By definition of $\tau_1 \circ \tau_2$ we have that $\tau_1(i) = (t, \alpha)$ implies $\tau_2(i) = (t, \beta)$ for some β, and $\tau'(t) = (t, \gamma'), \tau''(t) = (t, \gamma'')$ for some γ', γ'' such that $\gamma' = (\alpha \circ \beta), \gamma'' = \alpha \circ \beta)$. Now it is immediate from Lemma 3 to observe that $(\gamma' = \gamma'')$.

It remains to prove that $\tau\Uparrow \circ \tau\Downarrow = \tau$, provided that $\tau\Uparrow, \tau\Downarrow$ are defined. This is trivial, as $(\tau\Uparrow \circ \tau\Downarrow)|_t = (\tau|_t)\Uparrow \circ (\tau|_t)\Downarrow = \tau_{|_t}$, where the last equality has been obtained by applying Lemma 3. Also, if $\tau(i) = (t, \gamma')$, then $(\tau\Uparrow(i) = (t, \alpha), (\tau\Downarrow(i) = (t, \beta)$, which means that $(\tau\Uparrow \circ \tau\Downarrow)(i) = (t, \gamma'')$. We can proceed as above to show that $\gamma' = \gamma''$. $\square$

Proof of Lemma 5: We prove the following statements:

1. whenever $\tau \in (\!|L_2 \circ L_1|\!)^+$, then $\tau\Uparrow \in (\!|L_2|\!)^+$ and $\tau\Downarrow \in (\!|L_1|\!)^+$,
2. if $\sigma \rightsquigarrow_{\gamma, t}^{L_2 \circ L_1} \sigma'$ and $(s, v_2, v_1) \xrightarrow{(\alpha, \beta, \gamma)}$, then $\sigma \rightsquigarrow_{\alpha, t}^{L_2} \sigma'$ and $\sigma \rightsquigarrow_{\beta, t}^{L_1} \sigma'$.

Now suppose that $\tau \in T(L_2 \circ L_1)^+$. We have that $\tau \in (\!|L_2 \circ L_1|\!)^*$, and $[\![\tau]\!]^+_{(L_2 \circ L_1)}(\lambda l \in \mathsf{Locs}_{L_2 \circ L_1}.0) = \mathsf{true}$ Statement (i) ensures that $\tau\Uparrow \in (\!|L_2|\!)^+$, $\tau\Downarrow \in (\!|L_1|\!)^+$, while Statement (ii) gives that $[\![\tau\Uparrow]\!]^+_{L_2}(\lambda l \in \mathsf{Locs}_{L_2}.0) = \mathsf{true}$, $[\![\tau]\!]\Uparrow^+_{L_1}(\lambda l \in \mathsf{Locs}_{L_1}.0) = \mathsf{true}$. By definition, $\tau\Uparrow \in T(L_2)^+$, $\tau\Downarrow \in T(L_1)^+$, as we wanted to prove.

The second statement can be proved by a simple case analysis over the transitions defined for the automaton $\mathcal{A}$, in Figure 7, and by extensively using the properties of transformers introduced in Appendix A. Details are left to the reader.

For the first statement, we let $\tau \in (\!|L_2 \circ L_1|\!)^+$ and we show that for any $\psi : \tau|_t = (t, \psi)$ then $(\mathsf{Client}, 0, 0) \xRightarrow{(\psi\Uparrow, \psi\Downarrow, \psi}$. Further, $(t, \psi\Uparrow \in (\!|L_2|\!)^+|_t$ and $(t, \psi\Downarrow \in (\!|L_1|\!)^+|_t$. An immediate consequence of these two facts is that $\tau\Uparrow, \tau\Downarrow$ are defined and belong to $(\!|L_2|\!)^+, (\!|L_1|\!)^+$, respectively.

Let then $\psi : (t, \psi) \in (\!|L_2 \circ L_1|\!)^+|_t$, and suppose that $(L_2 \circ L_1) = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}} \rangle$. By definition, ψ belongs to the set

$$S = \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M'' \setminus M}} (\mathsf{call}?\, m(z))\, \mathsf{B}_m^t, (\mathsf{ret}!\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

where $\psi' \in \mathsf{B}_m^t$ if and only if $(t, \psi') \in (\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)^+(m, t)$. We can rewrite this set as

$$S = \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M'' \setminus M'}} (\mathsf{call}?\, m(z))\, \mathsf{B}_m^t, (\mathsf{ret}!\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in (M'' \cap M') \setminus M}} (\mathsf{call}?\, m(z))\, \mathsf{B}_m^t, (\mathsf{ret}!\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

where we made an explicit distinction between calls to methods in $M'' \setminus M'$ and calls to methods in $(M'' \cap M') \setminus M$.

That means that either $\psi = \varepsilon$, in which case there is nothing to prove, or $\psi = \psi_1 \cdots \psi_n$ for some $n > 0$ such that, for any $i = 1, \cdots, n-1$, either

(i) $\psi_i = \iota$, or
(ii) $\psi_i = (\mathsf{call}?\, m(z))\psi_i'(\mathsf{ret}!\, m(z'))$ for some $m \in M'' \setminus M'$, $\psi_i' \in \mathsf{B}_m^t$,
(iii) $\psi_i = (\mathsf{call}?\, m(z))\psi_i'(\mathsf{ret}!\, m(z'))$ for some $m \in (M'' \cap M') \setminus M$, $\psi_i' \in \mathsf{B}_m^t$,
(iv) $\psi_i \in (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))$.

Further, ψ_n is a prefix of one of the kind of strings described above. For $i = 1, \cdots, n-1$, we show that $(\mathsf{Client}, 0, 0) \xRightarrow{(\cdot,\cdot,\psi_i)} (\mathsf{Client}, 0, 0)$. This also given that $(\mathsf{Client}, 0, 0) \xRightarrow{(\cdot,\cdot,\psi_n)}$, from which it follows that

$$(\mathsf{Client}, 0, 0) \xRightarrow{(\cdot,\cdot,\psi_1)} (\mathsf{Client}, 0, 0) \xRightarrow{(\cdot,\cdot,\psi_2)} \cdots \xRightarrow{(\cdot,\cdot,\psi_{n-1})} (\mathsf{Client}, 0, 0) \xRightarrow{(\cdot,\cdot,\psi_n)}$$

as we wanted to prove.

which ψ_i takes either the form (ii) or (iii) described above. We only consider the case (iii), and we leave (ii) to the reader. First, note that since $(\psi_i = (\mathsf{call?}\, m(z))\psi_i'(\mathsf{ret!}\, m(z'))$, $m \in (M'' \cap M') \setminus M, \psi_i' \in \mathsf{B}_m^t$, then we have that $(\mathsf{Client}, 0, 0) \xrightarrow{(\mathsf{ExtCall}\, m(z), \mathsf{call?}\, m(z), \mathsf{call?}\, m(z))} (L_1, 0, 1)$, and $(L_1, 0, 1) \xrightarrow{(\mathsf{ExtRet}\, m(z'), \mathsf{ret!}\, m(z'), \mathsf{ret!}\, m(z'))} (\mathsf{Client}, 0, 0)$. Then we need to show that $(L_1, 0, 1) \xRightarrow{(\cdot,\cdot,\psi_1')} (L_1, 0, 1)$.

In practice, we show a stronger statement: for any $\psi' \in \mathsf{B}_m^t$, where $m \in (M'' \cap M') \setminus M$, then $(L_1, v_2, v_1) \xRightarrow{(\cdot,\cdot,\psi')} (L_1, v_2, v_1)$, provided that $v_1 > 0$. From the Definition of $(\!|B_{\mathsf{pub};}; B_{\mathsf{pvt}}|\!)^+$, we can infer that the set $(B_m^t)^+$, which is recursively defined as

$$\mathsf{B}_m^+ = \left(\left(\bigcup_{c \in \mathsf{PComm}} (c) \right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m' \notin M}} (\mathsf{call}\, m'(z))(B_{m'}^t) + (\mathsf{ret}\, m'(z')) \right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m' \in M}} (\mathsf{call!}\, m'(z))(\iota)^*(\mathsf{ret?}\, m'(z')) \right) \right)^*$$

is such that $\mathsf{B}_m^t \subseteq (\mathsf{B}_m^t)^+$. Now we have that $\psi_i' \in \mathsf{B}_m^t$, hence $\psi_i' \in (\mathsf{B}_m^t)^+$. Then either $\psi_i' = \varepsilon$, in which case there is nothing to prove, or $\psi_i' = \psi_i^1 \psi_i^k$, where for any $j = 1, \cdots, k$ either

(i) $\psi_i^j = c$ for some $c \in \mathsf{PComm}$, or

(ii) $\psi_i^j = (\mathsf{call}\, m'(z))\psi_j''(\mathsf{ret}\, m'(z'))$ for some $m' \notin M, z'$ and $\psi_j'' \in (\mathsf{B}_{m'}^t)^+$,

(iii) $\psi_i^j \in (\mathsf{call!}\, m'(z))(\iota)^*(\mathsf{ret?}\, m'(z'))$ for some $m' \in M, z' \in \mathbb{Z}$.

We leave to the reader to check that, for any such ψ_i^j, we have that $(L_1, v_2, v_1) \xRightarrow{(\cdot,\cdot,\psi_i^j)} (L_1, v_2, v_1)$, which in turn gives $(L_1, v_2, v_1) \xRightarrow{\cdot,\cdot,\psi_i'} (L_1, v_2, v_1)$. Note that, to prove case (ii), we need to perform an induction over $|\psi_i^j|$.

We have proved that whenever $(t, \psi) \in (\!|L_2 \circ L_1|\!)^+|_t$, then $\psi\Uparrow, \psi\Downarrow$ are defined; as a consequence, whenever $\tau \in (\!|L_2 \circ L_1|\!)^+|_t$ then $\tau\Uparrow$ and $\tau\Downarrow$ are defined. It remains to prove that $\tau\Uparrow \in (\!|L_2|\!)^+$, $\tau\Downarrow \in (\!|L_1|\!)^+$. We only prove that $\tau\Uparrow \in (\!|L_1|\!)^+$, and we leave the other statement to the reader. To this end, it suffices to prove that $\tau\Uparrow|_t \in (\!|L_2|\!)^+|_t$ for

any $t \in \mathcal{T}$. Recall that whenever $\tau|_t = (t, \psi)$, then ψ belongs to the set

$$S = \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M'' \setminus M'}} (\mathsf{call?}\, m(z))\, \mathsf{B}_m^t, (\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in (M'' \cap M') \setminus M}} (\mathsf{call?}\, m(z))\, \mathsf{B}_m^t, (\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

Now note that, whenever $(\mathsf{Client}, 0, 0) \xRightarrow{(\psi_2, \psi_1, \psi)}$, and $\psi \in S$, then ψ_1 belongs to the set $S{\Uparrow} = \{\psi{\Uparrow} \mid \psi \in S\}$, defined as

$$S{\Uparrow} = \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M'' \setminus M'}} (\mathsf{call?}\, m(z))\, (\mathsf{B}_m^t){\Uparrow}_{(L_2,1,0)}, (\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in (M'' \cap M') \setminus M}} (\mathsf{ExtCall}\, m(z))\, (\mathsf{B}_m^t){\Uparrow}_{(L_1,1,0)}, (callextmz')\right) \cup \left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

where, for any given set of strings X, $X{\Uparrow}_{s,v_2,v_1} = \{\psi_2 \mid (s, v_2, v_1) \xRightarrow{(\psi_2, \psi_1, \psi)}, \psi \in X\}$. First, note that whenever $m \in (M'' \cap M') \setminus M$, then $\mathsf{B}_m{\Uparrow} = \iota^*$, since $m \in \mathsf{Abs}(L_2)$. Therefore, we can rewrite $S{\Uparrow}$ as

$$S{\Uparrow} = \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M'' \setminus M'}} (\mathsf{call?}\, m(z))\, (\mathsf{B}_m^t){\Uparrow}_{(L_2,1,0)}, (\mathsf{ret!}\, m(z'))\right) \cup \left(\left(\bigcup_{\substack{z,z' \in \mathbb{Z} \\ m \in M (\cap M')}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

Suppose that $L_2 = \langle \mathsf{public} : B^2_{\mathsf{pub}}; \mathsf{private} : B^2_{\mathsf{pvt}} \rangle$.

It remains to prove that $(t, (\mathsf{B}_m^t){\Uparrow}_{(L_2,1,0)}) \subseteq (\!|B_{\mathsf{pub}^2}; B^2_{\mathsf{pvt}}|\!)(m, t)$. If $\psi' \in (\mathsf{B}_m^t){\Uparrow}\,(L_2, 1, 0)$, then it means that there exists an index k such that $(t, \psi') \in$

$(\mathcal{F}^+_{L_2\circ L_1})^k$. Here $\mathcal{F}^+_{L_2\circ L_1}$ is the monotone functional such that $\mathsf{lfp}(\mathcal{F}^+_{L_2\circ L_1}) = (\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)^+$, which exists by definition. Then, by induction over k, one can show that $(t, \psi'\Uparrow \in \mathcal{F}^+_{L_2})^k$, form which it follows that $(t, (\mathsf{B}^t_m)\Uparrow_{(L_2,1,0)}) \subseteq (\!|B_{\mathsf{pub}^2}; B^2_{\mathsf{pvt}}|\!)(m,t)$. □

LEMMA 14 *Let τ_2, τ_1 be two traces such that $(\tau_2 \circ \tau_1)\downarrow$. If $\tau'_2 \approx \tau_2$, $\tau'_1 \approx \tau_1$ and $(\tau'_2 \circ \tau'_1)\downarrow$, then $(\tau'_2 \circ \tau'_1) \approx (\tau'_2 \circ \tau'_1)$.*

Proof. This is a direct consequence of the fact that, whenever $(s, v_2, v_1) \xrightarrow{(\alpha,\beta,\gamma)}$ for some (s, v_2, v_1) and (α, β, γ), then $(\cdot, \alpha) \in \mathsf{Act}^+ \Leftrightarrow (\cdot, \beta) \in \mathsf{Act}^+ \Leftrightarrow (\cdot, \gamma) \in \mathsf{Act}^+$. □

Proof of Lemma 6 (Outline): Let $\tau_1 \in T(L_1)^+, \tau_2 \in T(L_2)^+$, and suppose that $\tau_1 \circ \tau_2$ is defined. One can show that $\tau_1 \circ \tau_2 \in (\!|L_2 \circ L_1|\!)^+$ in the same way we have done for the operators $\Uparrow$, $\Downarrow$ in Lemma 5. More specifically, suppose that $L_1 = \langle \mathsf{public} : B^1_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}^1}\rangle$, $L_2 = \langle \mathsf{public} : B^2_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}^2}\rangle$. If $\tau_1 \in T(L_1)^+, \tau_2 \in T(L_2)^+$, and $\tau_1 \circ \tau_2$ is defined, then for any thread t consider the pair (ψ_1, ψ_2) such that $\tau_1|_t = (t, \psi_1)$, $\tau_2|_t = (t, \psi_2)$; one can see that (ψ_2, ψ_1) belongs to the following set:

$$S = \mathsf{prefix}\left((\iota,\iota) \cup \left(\left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in M''\setminus M'}} (\mathsf{call?}\, m(z), \iota)\,(\mathsf{B}')^t_m)(\mathsf{ret!}\, m(z'), \iota)\right) \cup \left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in (M''\cap M')\setminus M}} (\mathsf{ExtCall}\, m(z), \mathsf{call?}\, m(z))\,(\mathsf{B}')^t_m)(\mathsf{ExtRet}\, m(z)\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in M}} (\mathsf{ExtCall}\, m(z), \mathsf{ExtCall}\, m(z))(\iota,\iota)^*(\mathsf{ExtRet}\, m(z'), \mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

where $(\psi'_2, \psi'_1) \in (\mathsf{B}')^t_m$ if and only if $(t, \psi_i) \in (\!|B^2_{\mathsf{pub}}; B^2_{\mathsf{pvt}}|\!)^+(m,t)$ for $i = 1, 2$, and $(\psi'_2 \circ \psi'_1)\downarrow$.

Note that for $m \in M'' \setminus M'$, $(\psi'_2, \psi'_1) \in (\mathsf{B}')^t_m$ implies that $\psi'_1 \in \{\iota\}^*$, while for $m \in (M'' \cap M') \setminus M$ we have that $(\psi'_2, \psi'_1) \in (\mathsf{B}')^t_m$ implies that $\psi'_2 \in \{\iota\}^*$. By definition, $(\psi'_2, \psi'_1) \in (\mathsf{B}')^t_m$ means $(t, \psi_i) \in ((\mathcal{F}^+_{L_i})^n\eta)(m,t)$ for some $n \geq 0$. One can then show, by induction over n, that $(t, \psi_2 \circ \psi_1) \in ((\mathcal{F}^+_{(L_2\circ L_1)})^n)\eta)(m,t)$. If we let $(L_2 \circ L_1) = \langle \mathsf{public} : B_{\mathsf{pub}}; \mathsf{private} : B_{\mathsf{pvt}}$, then we have that $\psi_2 \circ \psi_1) \in \mathsf{B}^t_m$, where the last is defined by letting $\psi'' \in \mathsf{B}^t_m$ iff $(t, \psi'') \in (\!|B_{\mathsf{pub}}; B_{\mathsf{pvt}}|\!)^+(m,t)$.

Consider now the set $S' = \{(\psi'''_2 \circ \psi'''_1) \mid (\psi'''_2, \psi'''_1) \in S\}$. By performing all the calculations, and using the fact that for any $(\psi_2, \psi_1) \in \mathsf{B}^{m,t}_i$ implies that $(\psi_2 \circ \psi_1) \in \mathsf{B}^{m,t}_i$, then we get

$$S' \subseteq \mathsf{prefix}\left(\iota \cup \left(\left(\left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in M''\setminus M'}} (\mathsf{call?}\, m(z))\,(\mathsf{B})^t_m)(\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in (M''\cap M')\setminus M}} (\mathsf{call?}\, m(z))\,(\mathsf{B})^t_m)(\mathsf{ret!}\, m(z'))\right) \cup \left(\bigcup_{\substack{z,z'\in\mathbb{Z}\\ m\in M}} (\mathsf{ExtCall}\, m(z))(\iota)^*(\mathsf{ExtRet}\, m(z'))\right)\right)^*\right)$$

In particular, since we assumed that $(\psi_2, \psi_1) \in S$, we have that $(\psi_2 \circ \psi_1) \in S'$. Now it is immediate to see that $\psi \in S'$ if and only if $(t, \psi) \in (\!|L_2 \circ L_1|\!)^+|_t$, from which we get that $(t, \psi_2 \circ \psi_1) \in (\!|L_2 \circ L_1|\!)^+|_t$.

We have shown that, for any thread identifier t, then $(\tau_2|_t) \circ (\tau_1|_t) \in (\!|L_2 \circ L_1|\!)^+|_t$. By the definition of $\circ$ for augmented traces, and since we are assuming that $(\tau_2 \circ \tau_1)\downarrow$, we obtain that $\tau_2 \circ \tau_1 \in (\!|L_2 \circ L_1|\!)^+$.

Also, note that if $[\![\tau_i]\!]^+_{L_2}(\lambda l \in \mathsf{Locs}_{L_2}.0) = \mathsf{true}$, for $i = 1, 2$, then $[\![\tau_2]\!]^+_{L_2\circ L_1}(\lambda l \in \mathsf{Locs}_{L_2\circ L_1}.0) = \mathsf{true}$. This can be proved by simply showing that, whenever $\sigma_2 \rightsquigarrow^{L_2}_{\alpha,t} \sigma_2'$ and $\sigma_1 \rightsquigarrow^{L_1}_{\beta,t} \sigma_2'$ for some $\sigma_1, \sigma_2, \alpha, \beta$ such that both $\sigma_2 \cdot \sigma_1$ and $\alpha \circ \beta$ are defined, then $\sigma_2 \cdot \sigma_1 \rightsquigarrow^{L_2\circ L_1}_{(\alpha\circ\beta),t} \sigma_2' \cdot \sigma_1'$.

Since $(\tau_2 \circ \tau_1) \in (\!|L_2 \circ L_1|\!)^+$, and $[\![\tau_2 \circ \tau_1]\!]^+_{L_2\circ L_1}(\lambda l \in \mathsf{Locs}_{L_2\circ L_1}.0) = \mathsf{true}$, we obtain that $\tau_2 \circ \tau_1 \in T^+(L_2 \circ L_1)$, as we wanted to prove.

Let us prove the second part of the statement. We say that $\psi_1' \approx \psi_1'$ if $(t, \psi_1) \approx (t, \psi_1')$, for any $t \in \mathcal{T}$. Let $L_2' : M \to M'$ be a library such that $(L_2' \circ L_1)\downarrow$. First we prove the following statement: suppose that $(t, \psi_2') \in T^+(L_2')|_t, (t, \psi_1') \in T^+(L_1)|_t$, $\psi_2' \approx \psi_2$, $\psi_1' \approx \psi_1$ for some ψ_2, ψ_1 such that $\psi_2 \circ \psi_1$ is defined (note that it is not necessary that $(t, \psi_2) \in T^+(L_2')|_t$); then there exist ψ_2'', ψ_1'' such that $(t, \psi_2'') \in T^+(L_2)|_t$, $(t, \psi_1'') \in T^+(L_1)|_t$, $(\psi_2'' \circ \psi_1'')\downarrow$ and $(\psi_2'' \circ \psi_1'') \approx (\psi_2 \circ \psi_1)$.

In order to prove this statement, we show how to construct the desired ψ_2'', ψ_1''. Let $\mathsf{Act}_\mathcal{A}$ be the set of actions which can be performed in the automaton $\mathcal{A}$, that is $\mathsf{Act}_\mathcal{A} = \{\alpha \mid (t, \alpha) \in \mathsf{Act}^+\}$. Define $\psi_i^0 := \psi_i'|_{\mathsf{Act}_\mathcal{A}\setminus\{\iota\}}$, for $i = 1, 2$. Note that $\psi_2^0 \approx \psi_2'$, $\psi_1^0 \approx \psi_1'$, $\psi_0^2 \in T^+(L_2')$ and $\psi_0^1 \in T^+(L_1)$.

If $(\psi_0^2 \circ \psi_1^0)\downarrow$, then it is easy to see that $(\psi_0^2 \circ \psi_1^0) \approx (\psi_2 \circ \psi_1)$. Otherwise, for any string ψ and integer $k \geq 0$, denote $(\psi)_k$ as the prefix of ψ of length k, $(\psi)^k$ as the suffix of ψ starting at the index $k + 1$; that is, $\psi = (\psi)_k(\psi)^k$.

We can choose the smallest integer k for which $(\mathsf{Client}, 0, 0) \xRightarrow{((\psi_2^0)_k, (\psi_1^0)_k, \psi^0)} (s, v_2, v_1)$, but $(\mathsf{Client}, 0, 0) \not\xRightarrow{((\psi_2^0)_{k+1}, (\psi_1^0)_{k+1}, \cdot)}$. Such a k exists, and is strictly smaller than $\min\{|\psi_2^0|, |\psi_1^0|\}$ because we are assuming that $(\psi_2^0 \circ \psi_1^0)$ is not defined. That is, the strings $(\psi_2^0)^k, (\psi_1^0)^k$ are non-empty.

Let $\alpha = (\psi_2^0)^k(1)$, $\beta = (\psi_1^0)^k(1)$. We know that $\alpha \circ \beta$ is not defined. By inspecting the possible transitions of $\mathcal{A}$, knowing that $\alpha \neq \iota, \beta \neq \iota$, we get different possible cases; we only list some of them, and leave the rest to the reader:

1. $\alpha = \mathsf{call?}\, m(z)$ for some m, z. In this case, we define $(\psi_2^1) = (\psi_2^0)$, $(\psi_1^1) = (\psi_1^0)_k \iota (\psi_1^1)_k$; note that in this case we know that either $k = 0$, or there exists an index $j < k$ such that $\psi_1^0(j) = \mathsf{ExtRet}\, m'(z')$, and for any $j < j' < k$, $\psi_1^0(j') \neq \mathsf{ExtCall}\, m''(z''), \mathsf{call?}\, m''(z'')$. Since ψ_1^1 is obtained by inserting a ι-action within a return and a call statement, in ψ_1^0, it follows that $(t, \psi_1^1) \in T^+(L_1)|_t$.
2. $\alpha = \mathsf{ExtCall}\, m(z)$; this case is not possible, in fact, since $\psi_2^0 \approx \psi_2$, $\psi_1^0 \approx \psi_1$, $(\psi_2 \circ \psi_1)\downarrow$ and the assumption that $\beta \neq \iota$, it necessarily has to be $\beta = \mathsf{ExtCall}\, m(z)$, contradicting the assumption that $(\alpha \circ \beta)$ is not defined
3. $\alpha = \mathsf{call}\, m(z)$, then we let $(\psi_2^1) = (\psi_2^0), (\psi_1^1) = (\psi_1^0)_k \iota (\psi_1^1)^k$. In this case, note that there exists an index $j < k$ such that $\psi_1^0(j) = \mathsf{ExtCall}\, m''(z'')$, or $\psi_1^0(j) = \mathsf{ret!}\, m''(z'')$, and for any $j' : j < j' < k$ we get that $j' \neq \mathsf{ExtRet}\, m(z), \mathsf{call?}\, m(z)$. This ensures that $(t, \psi_1^1) \in T^+(L_1)|_t$,
4. $\beta = c \in \mathsf{PComm}$; in this case we let $\psi_2^1 = (\psi_2^0)_k \iota (\psi_2^0)^k$, $\psi_1^1 = \psi_1^0$. Note that the ι-action in ψ_2^1 is never inserted within the execution of some method $m \in M'' \setminus M'$, so that we obtain that $(t, \psi_2^1) \in T^+(L_2')|_t$,
5. $\beta = \mathsf{ret?}\, m(z)$; again, we let $\psi_2^1 = (\psi_2^0)_k \iota$, $(\psi_1^0)_k \iota$, and we obtain that $(t, \psi_2^1) \in T^+(L'2)|_t$.

If $(\psi_2^1 \circ \psi_1^1)\downarrow$, then we let $\psi_2'' = \psi_2^1, \psi'' + 1 = \psi_2^1$. It is easy to see that, since $\psi_2 \approx \psi_2' \approx \psi_2^1$, and $\psi_1' \approx \psi_2' \approx \psi_2^1$, then $(\psi_2^1 \circ \psi_1^1) \approx (\psi_2 \circ \psi_1)$. If $(\psi_2^1 \circ \psi_1^1)$ is not defined, we iterate the procedure above, until reaching an index n for which $(\psi_2^n \circ \psi_1^n)\downarrow$. Note that, in the worst case, we need to insert a ι-action in ψ_2^0 to match any action of ψ_1^0, and vice-versa; that is, the procedure described above terminates after at most $(|\psi_2^0| + |\psi_1^0|)$ iterations.

Now, let $\tau_2 \in T^+(L_2)$, $\tau_1 \in T^+(L_1), \tau_2' \in T^+(L_2')$, and suppose that $\tau_2 \circ \tau_1\downarrow$. We can construct two traces τ_2'', τ_1', by applying the procedure described to each ψ_2^t, ψ_1^t such that $(t, \psi_2^t) = \tau_2'|_t$, $(t, \psi_1^t) = \tau_1|_t$, which preserve the order in which visible actions (that is, including primitive commands) are executed in τ_2', τ_1, and for which $\tau_2''(i) = (t, \cdot)$ implies $\tau_1'(i) = (t, -)$. These last constraint can be satisfied because we know that $|(\tau_2''|_t)| = |(\tau_1'|_t)|$ for any $t \in \mathcal{T}$. We hace that $(\tau_2'' \circ \tau_1')\downarrow$; further, since $\tau_2'' \approx \tau_2' \approx \tau_2$ and $\tau_1' \approx \tau_1$, we also have that $(\tau_2'' \circ \tau_1') \approx (\tau_2 \circ \tau_1)$. □

Proof of Lemma 8 (Outline): Suppose that $L_1 : M \to M'$, $L_2 : M' \to M''$ are such that $(L_2 \circ L_1)\downarrow$, $M \cap M' = \emptyset$ and $M' \cap M'' = \emptyset$. By the definition of $\circ$ for libraries, we also have that $M'' \cap M = \emptyset$. In particular, this means that for such libraries in the automaton $\mathcal{A}$, the transitions marked as (c), (d), (m) and (o) can never happen.

Suppose that $\tau_2 \circ \tau_1$ is defined for $\tau_2 \in T^+(L_2)$, $\tau_1 \in T^+(L_1)$. Then for any ψ_2^t, ψ_1^t such that $\tau_i|_t = (t, \psi_i^t)$, where $i = 1, 2$, we have that $\psi_2^t \circ \psi_1^t$ is defined.

Let us fix a thread identifier $t \in \mathcal{T}$. This means that, for any index $j = 1, \cdots, |\psi_2^t|$, $\psi_2^t(j) \circ \psi_1^t(i)$ is defined; by looking at the transitions defined for $\mathcal{A}$, knowing that (c), (d), (m) and (o) are not possible, it is easy to note that if $(t, \psi_1^t(j)) \in \mathsf{ClAct}^+$, then $\psi_2^t(j) = (\psi_2^t)^\uparrow$. If we also have that $(t, \psi_1^t(j)) \in \mathsf{AbsAct}^+$, then $\psi_2^t(j) \circ \psi_1^t(j) = (\psi_2^t)^\downarrow$, while if $(t, \psi_2^t(j)) \in \mathsf{ClAct}^+$ then $\psi_2^t(j) \circ \psi_1^t(j) = \psi_2^t(j))$.

Then, for any thread $t \in \mathcal{T}$, we have that if $\psi_2^t \circ \psi_1^t$ then $(t, \psi_2^t)|_{\mathsf{AbsAct}^+} = ((t, \psi_2^t)|_{\mathsf{ClAct}^+})^\uparrow$, $(t, \psi_2^t \circ \psi_1^t)|_{\mathsf{ClAct}^+} = (t, \psi_2^t)|_{\mathsf{ClAct}^+}$, and $(t, \psi_2^t \circ \psi_1^t)|_{\mathsf{AbsAct}^+} =$

$((t, \psi_1^t)|_{\mathsf{AbsAct}^+})^{\downarrow}$. Also, the definition of $\circ$ for traces ensures that these properties are preserved by the composition $\tau_2 \circ \tau_1$.

Now suppose that $\tau_2 \in T(L_2)^+, \tau_1 \in T(L_1)^+$ be such that $\tau_2|_{\mathsf{AbsAct}^+} = (\tau_1|_{\mathsf{ClAct}^+})^{\uparrow}$. We can proceed as in Lemma 6 to construct two $\tau_2' \in T(L_2)^+, \tau_1' \in T(L_1)$ such that $\tau_2' \approx \tau_2$, $\tau_1' \approx \tau_1'$ and $\tau_2' \circ \tau_2'$ is defined. Also, by inspecting each action in τ_2', τ_1', it is easy to see that

1. whenever $\tau_2'(i) \in \mathsf{ClAct}^+$ for some $i > 0$, then $(\tau_2' \circ \tau_1')(i) = \tau_2'(i)$ (recall that we are assuming that $L_2 : M' \to M''$ is such that $M'' \cap M' = \emptyset$, so that $\tau_2'(i)$ cannot be an action in ExtAct,
2. whenever $\tau_1'(i) \in \mathsf{AbsAct}^+$ for some $i > 0$, then $(\tau_2' \circ \tau_1')(i) = (\tau_2')(i)^{\downarrow}$

At this point it is immediate to show that $(\tau_2' \circ \tau_1')|_{\mathsf{ClAct}^+} = (\tau_2')$ $lvert_{\mathsf{ClAct}^+}$, and $(\tau_2' \circ \tau_1')|_{\mathsf{AbsAct}^+} = ((\tau_1')|_{\mathsf{AbsAct}^+})^{\downarrow}$. □

B.3 Closure Properties of Histories

LEMMA 15 *Let $L : M \to M'$ be a library; whenever $\sigma \rightsquigarrow^L_{\alpha_1,t_1} \sigma_1 \rightsquigarrow^L_{\alpha_2,t_2} \sigma_2$ such that $t_1 \neq t_2$, and either $\alpha_1 \notin \mathsf{PComm}$ or $\alpha_2 \notin \mathsf{PComm}$, then $\sigma \rightsquigarrow^L_{\alpha_2,t_2} \sigma_1' \rightsquigarrow^L_{\alpha_1,t_1} \sigma_2$.*

Proof. Without loss of generality, suppose that $\alpha_1 \notin \mathsf{PComm}$. The proof in the case $\beta \notin \mathsf{PComm}$ is similar. In this case we have that $\sigma \rightsquigarrow^L_{\alpha_1,t_1} \sigma_1$ implies that $\sigma_1 = \sigma[\mathsf{arg}_{t_1} \mapsto z_1]$ for some $z_1 \in \mathbb{Z}$. Since $t_2 \neq t_1$, by hypothesis, and by the properties of transformers introduced in Appendix A, $\sigma_1 \rightsquigarrow^L_{\alpha_2,t_2} \sigma_2$ implies that $\sigma_1[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})] \rightsquigarrow^L_{\alpha_2,t_2} \sigma_2[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})]$, and that $\sigma_2(\mathsf{arg}_{t_1}) = z_1$. Note that $\sigma_1[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})] = \sigma$, so that the last transformation can be rewritten as $\sigma \rightsquigarrow^L_{\alpha_2,t_2} \sigma_2[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})]$. Now note that $\sigma_2[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})] \rightsquigarrow^L_{\alpha_1,t_1} \sigma_2[\mathsf{arg}_{t_1} \mapsto z_1] = \sigma_2$. We can combine the last two transformations to obtain

$$\sigma \rightsquigarrow^L_{\alpha_2,t_2} \sigma_2[\mathsf{arg}_{t_1} \mapsto \sigma(\mathsf{arg}_{t_1})] \rightsquigarrow_{\alpha_1,t_1} \sigma_2$$

as we wanted to prove. □

COROLLARY 1 *Let $\mathcal{R}$ be the smallest relation such that $h = h_1(t_1, \alpha)(t_2, \beta)h_2$, $t_1 \neq t_2$, where either $\alpha \notin \mathsf{PComm}$ or $\beta \notin \mathsf{PComm}$. Then $h\mathcal{R}h_1(t_2, \beta)(t_1, \alpha)h_2$. Let $\equiv_{\mathcal{R}}$ be the equivalence relation induced by $\mathcal{R}$. Whenever $\tau \in T(L)^+, \tau \equiv_{\mathcal{R}} \tau'$, then $\tau' \in T(L)^+$.*

Proof. By Lemma 15 it is immediate to note that $T(L)^+$ is closed with respect to $\mathcal{R}$, hence it is closed with respect to its transitive closure $\mathcal{R}^+$. Since $\mathcal{R}$ is both symmetric and reflexive, we have that $\mathcal{R}^+$ is exactly $\equiv_{\mathcal{R}}$. □

Proof of Lemma 7 Let $h_1 \in [\![L]\!]^+$, and suppose that $h_2 \sqsubseteq h_1$. By definition, there exists $\tau_1 \in T(L)^+$ such that $\tau_1|_{\mathsf{Act}^+} = h_1$. We show that there exists τ_2 such that $\tau_2|_{\mathsf{Act}^+} = h_2$, and $\tau_1 \equiv_{\mathcal{R}} \tau_2$. Then we also have that $\tau_2 \in T(L)^+$, hence $h_2 \in [\![L]\!]^+$.

Since $h_2 \sqsubseteq h_1$, there exists a permutation π which preserves the order of thread executions and of pairs of actions in $\mathsf{Act}! \times \mathsf{Act}?$ from h_2 to h_1. We can extend such a function to π' which transforms τ_1 in τ_2 according to π, leaving the order of execution of primitive commands unchanged. By definition of $\tau_2, \tau_2|_{\mathsf{Act}^+} = h_2$.

It is important to note that the permutation π' can be obtained as the composition of a finite number of transpositions between adjacent numbers$\pi_1, \cdots, \pi_n$[3] that is $\pi_i = [j_i \mapsto j_i + 1, j_i + 1 \mapsto j]$ for some index j; in particular, let given $\tau^{(0)} = \tau_1, \tau^{(1)} = \pi_1(\tau^{(0)}), \cdots, \tau^{(n)} = \pi_n(\tau^{(n-1)})$. Then $\tau_2 = \tau^{(n)}$. Also, we can choose the permutations π_i so that, given $\tau^{(i-1)}(j_i) = (t, \alpha)$, $\tau^{(i-1)}(j_i + 1) = (t', \beta)$, then $t \neq t'$ and either $\alpha \notin \mathsf{PComm}$ or $\beta \notin \mathsf{PComm}$. This is because the action of the composite permutation π' over τ_1 preserves the order of execution of threads and primitive commands.

For any $i = 1, \cdots, n$, $\tau^{(i-1)}\ \mathcal{R}\ \tau^{(i)}$, which gives $\tau_1 = \tau^{(0)} \equiv_{\mathcal{R}} \tau^{(n)} = \tau_2$, as we wanted to prove. □

LEMMA 16 *For any two given histories h_1, h_2, we have that $h_1\ \overline{(\sqsubseteq)}\ h_2$ if and only if $h_2 \sqsubseteq h_1$.*

Proof. We only prove that $h_1\ \overline{(\sqsubseteq)}\ h_2$ implies $h_2 \sqsubseteq h_1$. The proof for the opposite implication is analogous.

If $h_1 \overline{(\sqsubseteq)} h_2$, by definition $h_1|_t = h_2|_t$ for any thread identidier $t \in \mathcal{T}$. Further, there exists a permutation π such that

(i) for any $i \in \{1, \cdots, |h_1|\}$, $h_1(i) = h_2(\pi(i))$,
(ii) for any $i, j \in \{1, \cdots, |h_1|\}$, if $i < j$, $h_1(i) \in \mathsf{Act?}$ and $h_1(j) \in \mathsf{Act!}$, then $\pi(i) < \pi(j)$.

Clearly we have that, for any $t \in \mathcal{T}$, $h_2|_t = h_1|_t$. Further, consider the inverse permutation of π, denoted as π^{-1}; we have that

(i) for any $i = \{1, \cdots, |h_2|\}$, $h_2(i) = h_2(\pi(\pi^{-1}(i))) = h_1(\pi^{-1}(i))$,
(ii) for any $i, j \in \{1, \cdots, |h_2|\}$, if $h_2(i) \in \mathsf{Act!}$ and $h_2(j) \in \mathsf{Act?}$, then $\pi^{-1}(i) < \pi^{-1}(j)$. In fact, if it were $\pi^{-1}(j) < \pi^{-1}(i)$, since $h_1(\pi^{-1}(j)) = h_2(j) \in \mathsf{Act?}$, $h_1(\pi^{-1}(i)) = h_2(i) \in \mathsf{Act!}$, then it would follow that $j = \pi(\pi^{-1}(j)) < \pi(\pi^{-1}(i)) = i$, contradicting the hypothesis that $i < j$.

□

Proof of Lemma 9 (Outline): We actually prove that any library L is $\left(\overline{\overline{\sqsupseteq}}\right)$-closed. The result follows then from Lemma 16.

First, let $\mathcal{R}_{\mathsf{client}}$ be a binary relation between histories such that whenever $h = h_1(t_1, \alpha)(t_2, \beta)h_2$, where either $\alpha \in \mathsf{ClAct}$ or $\beta \in \mathsf{ClAct}$, then $h_1(t_1, \alpha)(t_2, \beta)h_2\ \mathcal{R}_{\mathsf{client}}\ h_1(t_2, \beta)(t_1, \alpha)h_2$. That is, $\mathcal{R}_{\mathsf{client}}$ is the restriction of the binary relation $\mathcal{R}$ used in the proof of Lemma 15 to client actions; therefore, by the same Lemma, we get that if $h \in [\![L]\!]$ and $h\ \mathcal{R}_{\mathsf{Client}}\ h'$, then $h' \in [\![L]\!]$.

We can proceed as in Lemma 7 to show that if $h \in [\![L]\!]$, and $h \equiv_{\mathcal{R}_{\mathsf{client}}} h'$, then $h' \in [\![L]\!]$. Here $\equiv_{\mathcal{R}_{\mathsf{client}}}$ is the equivalence relation induced by $\mathcal{R}_{\mathsf{client}}$.

Note that, for how $\mathcal{R}_{\mathsf{client}}$ has been defined, we have that whenever $h \equiv_{\mathcal{R}_{\mathsf{client}}} h'$, then $h|_{\mathsf{AbsAct}} = h'|_{\mathsf{AbsAct}}$.

Let $h_1 \in [\![L]\!]$; we have to show that, whenever h' is such that $h_1|_{\mathsf{ClAct}}(\sqsupseteq)h'$, then there exists $h_2 \in [\![L]\!]$ such that $h_2|_{\mathsf{ClAct}} = h'$, and $\overline{h_1|_{\mathsf{AbsAct}}} \sqsupseteq \overline{h_2|_{\mathsf{AbsAct}}}$; with an

[3] It is well known, from group theory, that **(1)** every permutation can be written as a finite composition of transpositions, and **(2)** every transposition can be written as a finite composition of transpositions between adjacent numbers.

abuse of notation, we can rewrite this statement as $h_1|_{\mathsf{AbsAct}} \sqsubseteq h_2|_{\mathsf{AbsAct}}$. Intuitively, $h_1|_{\mathsf{ClAct}} \sqsupseteq h'$ states that whenever two (or more) threads are executing some methods $m_1, \cdots, m_k$ concurrently in h_1, they are also executing concurrently in h'. Therefore, we can construct h_2 so that its abstract actions agree with those of h_1. One can employ the same technique used in Lemma 7 to show that the history h_2 can be generated by a trace τ_2. imposed by h_1. That is, $h_1|_{\mathsf{AbsAct}} \sqsubseteq h_2|_{\mathsf{AbsAct}}$, or equivalently $\overline{h_1|_{\mathsf{AbsAct}}}\ \overline{(\sqsubseteq)}\ \overline{h_2}|_{\mathsf{AbsAct}}$. □

B.4 Compositionality of Linearisability

LEMMA 17 *Whenever* $(L_1 \uplus L_2)\downarrow$ *is defined,* $\sigma_1 \rightsquigarrow^{L_1}_{\alpha,t_1} \sigma_1'$, $\sigma_2 \rightsquigarrow^{L_2}_{\beta,t_2} \sigma_2'$ *for some* $\sigma_1, \sigma_2, \sigma_1', \sigma_2', t_1, t_2$, *such that* $(\sigma_1 \cdot \sigma_2)\downarrow$, $(\sigma_1' \cdot \sigma_2')\downarrow$ *and* $t_1 \neq t_2$, *then* $(\sigma_1 \cdot \sigma_2) \rightsquigarrow^{(L_1 \uplus L_2)}_{\alpha,t_1} \rightsquigarrow^{(L_1 \uplus L_2)}_{\beta,t_2} (\sigma_1' \cdot \sigma_2')$.

Proof. This is an immediate consequence of the fact that $\mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) = \{\mathsf{arg}_t\}_{t \in \mathcal{T}}$ and the assumption $t_2 \neq t_1$. □

Proof of Lemma 11 (Outline): The first part of the statements follows directly from Lemma 17. Suppose that $h \in \{h_1\} \bowtie \{h_2\}$ is a valid history for some $h_1 \in [\![L_1]\!], h_2 \in [\![L_2]\!]$. Then there exist τ_1, τ_2, τ such that $\mathsf{history}(\tau_i) = h_i$, $i = 1, 2$, $\tau \in \{\tau_1\} \bowtie \{\tau_2\}$ such that $\tau \in T(L_1 \uplus L_2)$ and $\mathsf{history}(\tau) = h$. It suffices to show that $[\![\tau]\!]_{L_1 \uplus L_2}(\lambda l \in \mathsf{Locs}_{L_1 \uplus L_2}.0) = \mathsf{true}$. This is true because τ preserves the thread executions of τ_1, τ_2; further, the order in which primitive commands executed by L_1 and L_2 are interleaved does not affect the result of the interpretation function (Lemma 17).

The second part of the Lemma is trivial. Suppose that $L_1 : M_1 \to M_1'$, $L_2 : M_2 \to M_2'$, and $(L_1 \uplus L_2)\downarrow$. Given $h \in L_1 \uplus L_2$ there exists a trace $\tau \in T(L_1 \uplus L_2)$. We can easily construct two traces $\tau_1 \in (\!|L_1|\!), \tau_2 \in (\!|L_2|\!)$ such that $\tau \in \{\tau_1\} \bowtie \{\tau_2\}$. In fact, we can construct τ_1 by choosing the subtrace of τ which contains method invocations and returns in $M_1 \cup M_1'$, and the relative primitive commands executed within an execution of such methods. The trace τ_2 can be constructed similarly, this time selecting method invocaitons and returns in $M_2 \cup M_2'$. The fact that $(M_1 \cup M_1') \cap (M_2 \cup M_2') = \emptyset$, by hypothesis, ensures that $\tau \in \{\tau_1\} \bowtie \{\tau_2\}$.

The properties stated for transformers in Appendix A ensure that both $[\![\tau_i]\!]_{L_i}(\lambda l \in \mathsf{Locs}L_i.0) = \mathsf{true}$, for $i = 1, 2$; it follows that $\tau_i \in T(L_i), i = 1, 2$. Now, since $\tau \in \{\tau_1\} \parallel \{\tau_2\}$, and since $(M_1 \cup M_1') \cap (M_2 \cup M_2') = \emptyset$, it follows that all the actions recorded in h which contains an occurrence of a method $m_1 \in M_1 \cup M_1'$, appear in τ_1; the same is true for actions with occurrences of methods $m_2 \in M_2 \cup M_2'$ and the trace τ_2. Now it is trivial to observe that $\mathsf{history}(\tau_1) = h|_{\mathsf{Act}_{M_1 \to M_1'}}$, $\mathsf{history}(\tau_2) = h|_{\mathsf{Act}_{M_2 \to M_2'}}$. □